An Internship Report
on

Comparative Study of Search and Sampling based Algorithms & Implementation of GBNN for complete Coverage Path Planning for AUV

Submitted in the fulfillment of
Summer Research Internship

Submitted by

Sneha Dutta **U21ME038**

Under the supervision of

Dr. Jagadeesh Kadiyam
Assistant Professor
CAIR@IIT Mandi

Center for Artificial Intelligence and Robotics
Indian Institute Of Technology Mandi
Parashar Road, Tehsil Sadar, Near Kataula, Kamand, Himachal Pradesh-175005

May 2024 - July 2024

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to all those who made my summer research internship a fruitful and enriching experience. First and foremost, I extend my heartfelt thanks to **Dr. Jagadeesh Kadiyam** for all the invaluable guidance, support, and encouragement throughout my internship and I am profoundly grateful for the opportunities they provided me to learn and grow in my field.I also wish to thank the entire team of CAIR at IIT Mandi for their assistance and collaboration. Their willingness to share knowledge and provide constructive feedback was instrumental in the development of my project. Special thanks to my **Mtech and PhD Scholars** for their support, which made my time at the institution both productive and enjoyable.I extend my deepest gratitude to my parents for their constant support and care, which have been my bedrock throughout this journey. Thank you all for being an integral part of academic and personal growth.

Date: 25/07/2024
Place: IIT Mandi, Mandi, HP-175005

The project titled ”Path Planning of Autonomous Underwater Vehicle” explores various path planning algorithms to enhance the efficiency and effectiveness of autonomous underwater vehicles (AUVs) in performing specific tasks. This research investigates both search-based algorithms such as A*, D*lite, and Dijkstra's, as well as sampling-based algorithms including Probabilistic Roadmap (PRM), Rapidly-Exploring Random Tree (RRT), their optimized versions, and the Rapidly-Exploring Random Graph (RRG) algorithm. A comprehensive comparative study of these algorithms is conducted to evaluate their performance in different scenarios.

The primary focus of this project is the development of a path planning system for an autonomous ship hull cleaning system, which necessitates complete coverage path planning. To achieve this, the Glasius Bio-Inspired Neural Network (GBNN) is employed, facilitating the creation of a 2-dimensional simulation that incorporates obstacle avoidance for static environments as well as dynamic environment.The GBNN approach not only ensures thorough coverage but also enhances the efficiency of the path planning process, making it suitable for applications where comprehensive surface coverage is critical. This research provides significant insights and advancements in the field of autonomous underwater navigation and path planning, contributing to the development of more effective AUV systems for marine applications.

List of Figures

Contents

Chapter 1

Introduction

An autonomous underwater vehicle (AUV) is an underwater vehicle capable of self-propulsion, also known as unmanned underwater vehicle. It is a robotic device that is driven through the water by a propulsion system, controlled by an on board computer and manoeuvrable in three dimension. AUVs are being used to explore underwater environment, mines clearing operation.

1.1 Importance of Autonomous Underwater Vehicle(AUV)

An autonomous underwater vehicle (AUV) is an unmanned underwater self-propelled robot. They are a part of a larger class of unmanned underwater vehicles of which another part is Remotely Operated Vehicles (ROVs). The function of AUVs is to navigate through the water without the assistance of an operator performing surveys and collecting data which is achieved by pre-programming the AUVs at the surface. AUVs are playing some important roles in important cases since their inception. They have a wide range of application varying from commercial uses, research purposes, military applications, air crash investigations etc. They are used to make detailed maps and surveys of the sea floor (oil and gas industries) before building subsea infrastructure. This helps in installation of pipelines and subsea completions in a cost-effective manner with minimum disruption to the environment.

1.1.1 Need of Autonomous Ship Hull Cleaning System

The operating lifetime of a ship is expected as twenty to twenty-five years according to their design consideration. Routine ship hull maintenance work such as hydro blasting for removing corrosion is an essential task for efficient and sustainable operation. . Ship hull maintenance work is conventionally conducted by human labours and it is a tough work conducted under hazardous and heavy equipment. Human faults during the maintenance could cause damages to ship hulls and labourers. Therefore, many robotics and automation solutions have been introduced in recent years to cater to the demand in ship hull maintenance industry since it can resolve the complications of the conventional methods.

1.1.2 Challenges involved in Autonomous Ship Hull Cleaning System

Challenges Standalone Ground Robots: Require crane systems for access. Insufficient with a single crane for large hull areas.Need multiple units around the ship. Ground sensing systems degrade due to obstructions.

Aerial Vehicles (UAVs): Popular for ship hull inspections due to remote operation capabilities.

Equipped with advanced vision and perception systems.
Limited to inspection tasks, unable to perform intense maintenance activities like blasting.

Climbing Robots: Overcome limitations of ground robots and UAVs.
Preferred for ship hull maintenance due to their ability to adhere to vertical and inverted surfaces.

Remaining Challenges:

Adhesion: Ensuring consistent adhesion to the hull surface, especially under varying conditions.
Navigation: Navigating complex geometries of the hull. Dealing with biofouling, which can obscure sensors and make surfaces slippery.
Weather Conditions: Operating in various weather conditions, including high humidity, rain, and extreme temperatures.
Advanced Sensors: Integrating advanced sensors for accurate perception and environment mapping. Developing sensors that can withstand harsh marine environments.
Real-time Decision Making: Developing efficient algorithms for real-time decision-making. Ensuring obstacle avoidance in dynamic and complex environments
Maintenance Activities: Performing intense maintenance activities such as blasting, scraping, and painting. Additional Information
Energy Efficiency: Autonomous systems must conserve energy for extended operations, especially in large hull areas.
Durability: Robots must be durable to withstand the harsh marine environment, including corrosion resistance and mechanical robustness.
Communication: Ensuring reliable communication between robots and control systems, especially in underwater or obstructed environments.

1.2 Objectives

The objective of path planning for an autonomous underwater vehicle (AUV) is to ensure efficient and effective navigation through underwater environments. This involves determining collision-free routes that optimize energy consumption and mission success. Path planning must adapt to dynamic environmental conditions such as currents, temperature gradients, and salinity variations. Additionally, it aims to achieve mission-specific goals, such as thorough area coverage or precise data collection, while maintaining robustness against sensor errors and communication delays. Overall, the objective is to enhance the AUV's operational efficiency, safety, and mission performance.

1.2.1 Effective and efficient path planning for autonomous underwater vehicle

• **Collision Avoidance**: Ensuring the AUV navigates without colliding with underwater obstacles, marine life, or other vessels.
• **Energy Efficiency**: Optimizing the path to minimize energy consumption, extending the vehicle's operational time.
• **Environmental Adaptability**: Adjusting routes based on varying underwater conditions such as currents, temperature gradients, and salinity.
• **Mission Success**: Achieving the specific goals of the mission, such as reaching a designated area, collecting data, or inspecting underwater structures.

- **Robustness**: Developing paths that can adapt to sensor errors, communication delays, and unexpected environmental changes.
- **Coverage**: Ensuring thorough exploration or monitoring of specified areas, especially in mapping or survey missions.
- **Time Efficiency**: Minimizing the time taken to complete the mission while maintaining safety and energy constraints.
- **Data Quality**: Optimizing the path to enhance the quality and relevance of the collected data, whether it's for scientific research, resource exploration, or environmental monitoring.

1.3 Introduction to path planning

Path planning involves finding a collision-free route for a robot from its starting point to its destination and is fundamental in robotics. Ideally, algorithms guarantee a collision-free path if one exists. Path planning is crucial for UAV autonomy, defining optimal routes for various tasks, including navigating uncertain terrains, tracking people in disaster areas, or military applications. It considers natural factors like wind speed, atmospheric pressure, and temperature. Path planning involves three patterns: data streaming, user-assigned tasks, and tasks performed by the planner.

1.3.1 Overview of Graph Based Path Planning

Graph-based path planning is a method used in robotics and artificial intelligence to determine the optimal path from a starting point to a destination. In this approach, the environment is represented as a graph, where nodes represent possible positions or states, and edges represent the paths or transitions between these states. Algorithms like Dijkstra's, A*, and D* are commonly used to find the shortest or most efficient path through the graph. These algorithms evaluate the cost associated with each edge and use heuristics to guide the search process, balancing between exploration and exploitation. Graph-based path planning is advantageous due to its ability to handle complex environments and dynamic obstacles, making it suitable for applications in autonomous navigation, robotics, and game development. However, the computational complexity can increase with the size of the graph, necessitating efficient data structures and optimization techniques.

1.3.2 Overview of Bio Inspired Algorithms for Path Planning

Bio-inspired algorithms are widely used in the path planning of autonomous underwater vehicles (AUVs) due to their robustness and adaptability in dynamic and complex environments. Genetic Algorithms, inspired by natural selection, evolve a population of potential solutions over generations using selection, crossover, and mutation to find optimal paths. Particle Swarm Optimization mimics the social behavior of birds flocking or fish schooling, where a population of particles adjusts their positions based on personal and collective experience to converge on the best solution. Ant Colony Optimization is inspired by the foraging behavior of ants, using pheromone trails to guide the search for optimal paths, effectively handling the exploration and exploitation trade-off. Artificial Bee Colony algorithms simulate the foraging behavior of honeybees, balancing exploration and exploitation by using employed, onlooker, and scout bees to find and refine paths. These algorithms provide robust, efficient, and scalable solutions for the complex path planning challenges faced by AUVs in underwater environments.

Chapter 2

Literature Review

2.1 Ruled Path Planning Framework for Safe and Dynamic Navigation

The paper ”Ruled Path Planning Framework for Safe and Dynamic Navigation” presents a path planning framework designed for safe and dynamic navigation in multi-dimensional environments, with a focus on Autonomous Underwater Vehicles (AUVs)

Summary of the Significant findings and Conclusions

Path Planning Framework: The paper introduces a highly Parametrised Rapidly-exploring Random Graph (PRRG) along with a system of rules for dynamic node selection and the D* Lite search algorithm to create paths. This framework allows for quick adaptation while minimizing computational power requirements.

Real-time Adaptation: The framework is capable of adapting to changes in the environment and mission requirements in real-time. It offers the flexibility to dynamically adjust graph parameters, enabling efficient planning in dynamic environments.

Custom Node Generation: An extension for custom node generation is proposed, allowing for the creation of specific routes online. This feature is particularly useful for survey and inspection applications, such as ship hull inspections.

Validation through Simulation: The capabilities of the framework were tested in a simulation scenario for underwater ship hull inspection. The framework demonstrated its effectiveness in planning safe navigation paths around the ship and creating inspection routes.

Scalability and Multi-dimensional Adaptation: The framework is designed to scale to multiple applications, including navigation in underwater environments that require a 3-dimensional state space. It can adapt to changes in depth, longitudinal, and lateral movements of drones.

Computational Efficiency: The framework proved to be computationally efficient in various scenarios and demonstrated the ability to adapt efficiently to different environments. The set of rules defined a new planning state space suited to specific applications.

Overall, the paper concludes that the proposed path planning framework offers efficient and adaptable solutions for safe navigation in dynamic environments, with potential applications in both aerial and underwater domains.

2.2 Sampling-based Algorithms for Optimal Motion Planning

Summary of the Significant findings and Conclusions

Theoretical Guarantees: The paper establishes connections between sampling-based path planning algorithms and random geometric graphs, providing insights into the probabilistic completeness and asymptotic optimality of these algorithms .

Algorithm Analysis: Various sampling-based path planning algorithms are analyzed in terms of their probabilistic completeness, asymptotic optimality, and computational complexity. The table in the document summarizes the time and space complexity of different algorithms .

Future Directions: The paper suggests potential extensions and improvements for sampling-based path planning algorithms, such as exploring connections with random geometric graphs, analyzing deterministic sampling-based algorithms, and addressing motion planning problems with complex constraints like differential dynamics and temporal/logic constraints .

Practical Applications: The algorithms discussed in the paper have practical applications in robotics, autonomous vehicles, and military scenarios. The paper highlights the importance of considering constraints like differential dynamics and temporal/logic constraints in motion planning problems .

Software Implementation: The authors have released a software library implementing the new algorithms discussed in the paper as open-source software, providing a practical tool for researchers and practitioners in the field .

Experimental Validation: The paper presents experimental results to validate the theoretical findings and demonstrate the effectiveness of the proposed algorithms in real-world scenarios .

Overall, the paper provides valuable insights into the theoretical foundations, practical applications, and future directions of sampling-based path planning algorithms, offering a comprehensive analysis of their performance and potential for further development.

2.3 Toward energy-efficient online Complete Coverage Path Planning of a ship hull maintenance robot based on Glasius Bio-inspired Neural Network

Summary of the Significant findings and Conclusions

Routine Maintenance Requirement: Regular ship hull maintenance is essential for sustainability and efficiency.

Shortcomings of Human Labor: Conventional maintenance work involving human labor has many shortcomings.

Introduction of Maintenance Robots: Robots have been introduced for drydocks to address these shortcomings.

Importance of Energy-Efficient CCPP: Energy-efficient Complete Coverage Path Planning (CCPP) is crucial for ship hull maintenance robots.

Novel CCPP Method: A novel Complete Coverage Path Planning(CCPP) method based on Glasius Bioinspired Neural Network (GBNN) is proposed for a ship hull inspection robot.

Comprehensive Energy Model: The proposed method incorporates a comprehensive energy model for path planning, considering direction changes, distance, and vertical position.

Effectiveness in Dynamic Workspaces: The method performs online path planning, making it effective in dynamic workspaces.

Comparison with State of the Art: The proposed method's behavior and performance were compared against state-of-the-art methods using simulations with Hornbill, a multipurpose ship hull maintenance robot.

Complete Coverage Achievement: The proposed method successfully achieves complete coverage in dynamic workspaces.

Energy Usage Reduction: The proposed method significantly reduces energy usage compared to state-of-the-art methods, confirmed by statistical outcomes from simulations.

Validation of Method: Validation through simulations shows the proposed method's ability to realize complete coverage in dynamic workspaces and surpass state-of-the-art methods in energy efficiency.

Contribution to Energy-Efficient CCPP: The proposed method contributes significantly to the development of energy-efficient CCPP methods for ship hull maintenance robots, improving their overall performance and sustainability.

2.4 Glasius bio-inspired neural networks based UV-C disinfection path planning improved by preventive deadlock processing algorithm

Summary of the Significant findings and Conclusions

Simulation Model Development: A simulation model was developed to test CCPP algorithms for UV-C disinfection. The model included a novel control architecture combining GBNN for map discretization, a motion strategy for path planning in the presence of obstacles, UV-C radiation dose estimation, a speed controller for adjusting robot speed, and a pure pursuit controller for correct robot movement.

Preventive Deadlock Processing: An original preventive deadlock processing approach was proposed with the ERGA to address deadlock issues. The robot could anticipate deadlocks by evaluating surrounding cells and react to avoid blockages, thus improving operational efficiency.

Comparison of Movement Strategies: The study compared the performance of spiral and boustrophedon movement strategies. It was observed that the spiral movement had advantages such as fewer deadlocks, reduced battery usage, and lower radiation dose overpasses compared to the boustrophedon movement.

Control Architecture Efficiency: The integration of GBNN, UV-C estimation, speed controller, and pure pursuit control in the control architecture enabled effective UV-C CCPP. The preventive deadlock processing and escape route generator algorithms further enhanced the efficiency of the disinfection process.

Potential Applications: The research findings have implications for combating the spread of infectious diseases like COVID-19. The developed strategies and algorithms can improve the effectiveness of UV-C disinfection processes in complex environments with obstacles, leading to more regular radiation levels and successful completion of disinfection tasks.

Overall, the study highlights the importance of innovative control architectures and preventive strategies in enhancing UV-C disinfection path planning, with practical implications for improving disinfection processes in various settings.

2.5 Complete Coverage Autonomous Underwater Vehicles Path Planning Based on Glasius Bio-Inspired Neural Network Algorithm for Discrete and Centralized Programming

Efficient Path Planning: The introduction of the Glasius bio-inspired neural network (GBNN) algorithm offers a novel approach to path planning for autonomous underwater vehicles (AUVs). This algorithm demonstrates efficiency with a reduced calculation burden, leading to quicker path planning and improved overall efficiency in AUV operations.

Multi-AUV Cooperation: The study showcases that through the proposed algorithm, multi-AUVs can collaboratively plan reasonable and collision-free coverage paths. By working together, these vehicles achieve full coverage on the same task area, showcasing the benefits of division of labor and cooperation in underwater missions.

Collision-Free Coverage: The algorithm ensures that multi-AUVs can cover designated areas without collisions between vehicles and without repeating coverage. This feature enhances task execution efficiency and ensures that the mission is completed effectively and without interruptions.

Efficient Full Coverage: The algorithm based on GBNN and discrete and centralized programming proves to be effective in enabling AUVs to efficiently cover designated areas. By leveraging division of labor and cooperation, the algorithm facilitates multi-AUVs in achieving full coverage without encountering collisions.

Simulation Validation: Through simulation experiments, the study confirms the algorithm's effectiveness in planning reasonable and collision-free paths for multi-AUVs. These results demonstrate the algorithm's capability to successfully achieve full coverage in large water ranges, highlighting its practical applicability in real-world scenarios.

Cooperative Multi-AUV Systems: The research underscores the importance of cooperative multi-AUV systems in enhancing coverage efficiency and reducing mission time. By leveraging advanced algorithms like GBNN and strategic cooperation, AUVs can optimize their path planning processes and achieve mission objectives effectively in underwater environments.

Chapter 3

Path Planning Algorithms

Path planning is a fundamental aspect of robotics and autonomous systems, involving the determination of a viable route from a starting point to a destination while avoiding obstacles and optimizing certain criteria such as distance, time, or energy consumption. Various algorithms have been developed to address the challenges of path planning, each with its strengths and suitable applications.

3.1 Classification of Path Planning Algorithm

The various path planning algorithms can be classified in the following way.

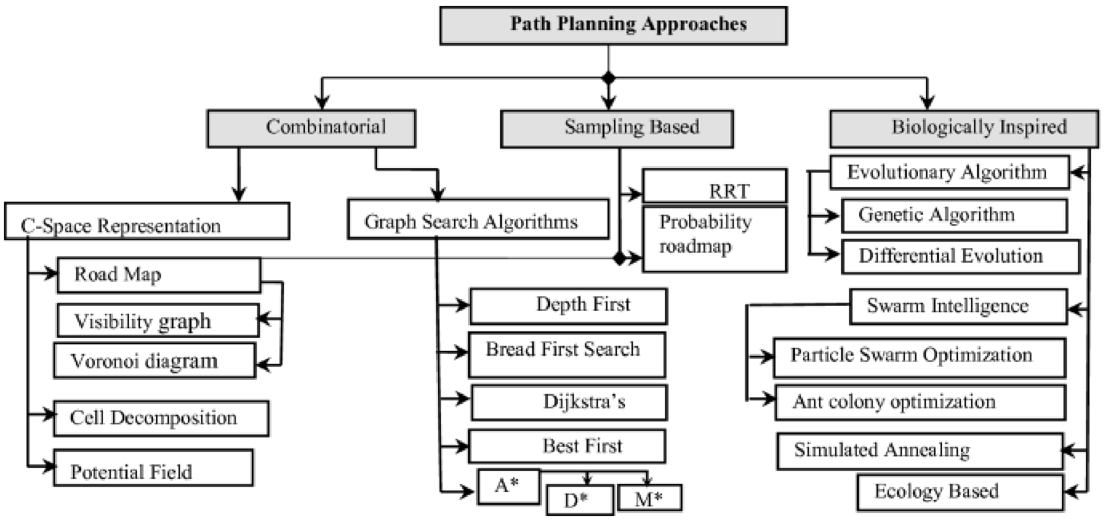


Figure 3.1 Classification of path planning algorithms

In this case focus will be more on Graph based search Algorithms and Sampling based algorithms, as those are considered mainly for the path planning of Autonomous Underwater Vehicle(AUV).

Before dive into the details and implementation of the various algorithms, it is important to consider the factors that determine which type of algorithm to use in different scenarios.

3.2 Use Cases for Various Algorithm

3.2.1 Search Based Algorithm

Known Environments: Use search-based algorithms like A* when the environment is completely known and well-defined.
Optimality: When optimality is a critical factor, as these algorithms guarantee finding the shortest path if one exists.
Discrete Spaces: Suitable for grid-based or discrete configuration spaces where the state space can be easily divided into manageable units.
Complex Constraints: Ideal for scenarios with complex constraints or specific requirements that can be efficiently handled using heuristic-based searches.
Guaranteed Feasibility: When a feasible path must be guaranteed, as search-based algorithms explore systematically and ensure that all possible paths are considered.

3.2.2 Sampling Based Algorithm

High-Dimensional Spaces: Use sampling-based algorithms like RRT or PRM for high-dimensional configuration spaces where searchbased algorithms may be impractical.
Unknown or Dynamic Environments: Suitable for partially known or dynamic environments where the configuration space may change over time.
Efficiency Over Optimality: When the primary goal is to quickly find a feasible path rather than the optimal one, as these algorithms can rapidly explore large spaces.
Continuous Spaces: Ideal for continuous configuration spaces where discretization is difficult or infeasible.
Scalability: When the problem involves multiple robots or complex motion planning tasks, sampling-based algorithms can efficiently handle the complexity and scale.

3.3 Search Based Algorithms

Search-based algorithms for path planning are methods used to find an optimal path from a starting point to a goal within a defined space. These algorithms explore the search space systematically to determine the best route, often based on criteria such as distance, cost, or time. Examples include the A* algorithm, which uses heuristics to guide the search more efficiently, and the Dijkstra algorithm, which guarantees the shortest path in weighted graphs. These algorithms are widely used in robotics, autonomous vehicles, and computer games due to their robustness and ability to handle various environments and constraints.

3.3.1 Dijkstra's Algorithm

Dijkstra's algorithm is a popular graph search algorithm used to find the shortest path between nodes in a graph, which may represent, for example, road networks. It works by starting at the source node and exploring all neighboring nodes, iteratively expanding the shortest known path until the destination node is reached. The algorithm uses a priority queue to efficiently select the next node with the shortest tentative distance. Dijkstra's algorithm guarantees the shortest path in graphs with non-negative edge weights.

3.3.2 Pseudocode of Dijkstra's Algorithm

Algorithm 1 Dijkstra's Algorithm

```
Require: Dijkstra$G, w, s$
 1: Initialize $d[v] \leftarrow \infty$ for all vertices $v \in G$
 2: $d[s] \leftarrow 0$
 3: Initialize priority queue $Q \leftarrow \{s\}$
 4: while $Q$ is not empty do
 5:     $u \leftarrow$ Extract-Min($Q$)
 6:     for each neighbor $v$ of $u$ do
 7:         if $d[u] + w(u, v) < d[v]$ then
 8:             $d[v] \leftarrow d[u] + w(u, v)$
 9:             Decrease-Key($Q, v, d[v]$)
10:         end if
11:     end for
12: end while
13:
14: return $d$
```

3.3.3 A* Algorithm

A* Search algorithm is one of the best and popular technique used in path-finding and graph traversals. It is informed search algorithm which also incorporates use of heuristic function.

A* is a heuristic function, which differs from an algorithm in that a heuristic is more of an estimate and is not necessarily provably correct. A* expands paths that are already less expensive by using this function:

$$f(n) = g(n) + h(n)$$

where, $f(n)$ = total estimated cost of path through node n
$g(n)$ = actual cost so far to reach node n
$h(n)$ = estimated cost from n to goal. This is the heuristic part of the cost function, so it is like a guess.

3.3.4 Pseudocode of A* Algorithm

3.3.5 Lifelong Planning A*

An incremental heuristic search algorithm based on A*,first described by **Sven Koenig** and **Maxim Likhachev** in 2001.
Key Characteristics: **Incremental version of A*** Adapts to changes in the graph without recalculating the entire graph.

Algorithm 2 A-star algorithm

```
1: **Make** $P$[start] as **open list**.
2: **while** open list $\neq$ empty **do**
3:   **select** the node $P[i]$ from the **open list** whose value of evaluation function $F(P[i])$ is smallest.
4:   **Make** $P[i]$ as **close list**
5:   **if** $P[i] = P$[end] **then**
6:     **return** "path is found"
7:     **Select** the successor node $P[j]$ around the node $P[i]$ calculate $F(P[j])$
8:     $H \leftarrow$ Calculate the heuristic search cost by the Manhattan distance from the target point.
9:     $G \leftarrow$ The cost of moving to the current position + the next move cost.
10:    $F \leftarrow H + G$
11:    **if** $P_j$ belongs to obstacle or close list node **then**
12:      **continue**
13:    **end if**
14:    **Mark** $P[j]$ as **open list**.
15:    **if** $P[j]$ belongs to open list and $F(P[j]) < F(P[m])$ when $P[m]$ was marked as **close list** **then**
16:      **Set parent** node of $P[j]$, $F(P[i]) = F(P[j])$
17:    **end if**
18:  **end if**
19: **end while**
20: **return** "the path cannot be found"
```

Updates the g-values (distance from start) from the previous search during the current search to correct them when necessary.
Utilizes a heuristic, which is a lower boundary for the cost of the path from a given node to the goal.
Heuristic Properties: A heuristic is admissible if it is guaranteed to be non-negative (zero being admissible). A heuristic should never be greater than the cost of the cheapest path to the goal.

3.3.6 D* Lite Algorithm

D* Lite is not based on the original D*, but implements the same behaviour. It is simpler to understand and can be implemented in fewer lines of code, hence the name "D* Lite".

D* Lite is an incremental heuristic search algorithm which is based on Lifelong Planning A*(LPA*), it has completely obsoleted D*. Thus, there is never any reason to use D*; use D*-Lite instead.

The Connection between D* Lite and LPA* Algorithm

LPA*, that repeatedly determines shortest paths between the start vertex and the goal vertex as the edge costs of a graph change.

Use of LPA* to develop D* Lite, that repeatedly determines shortest paths between the current vertex of the robot and the goal vertex as the edge costs of a graph change while the robot moves towards the goal vertex.

D* Lite does not make any assumptions about how the edge costs change, whether they go up or down, whether they change close to the current vertex of the robot or far away from it, or whether they change in the world or only because the robot revised its initial estimates. D* Lite can be used to solve the goal-directed navigation problem in unknown terrain.

Search Direction : It is needed to switch the search direction of LPA*. The LPA* searches from the start vertex to the goal vertex and thus its g-values are estimates of the start distances.D* Lite searches from the goal vertex to the start vertex and thus its g-values are estimates of the goal distances. It is derived from LPA* by exchanging the start and goal vertex and reversing all edges in the pseudo code.

Thus, D* Lite operates on the original graph and there are no restrictions on the graph except that it needs to be able to determine the successors and predecessors of the vertices, like LPA*.

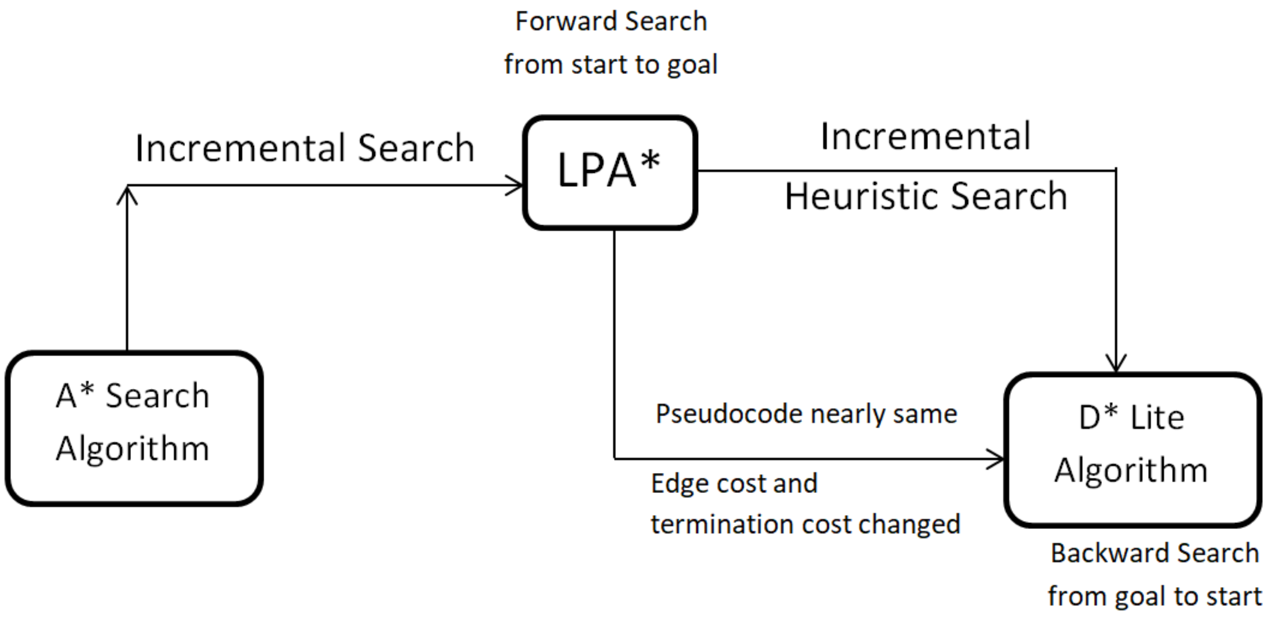


Figure 3.2 Depicts the interrealtion between A*, LPA* and D* Lite

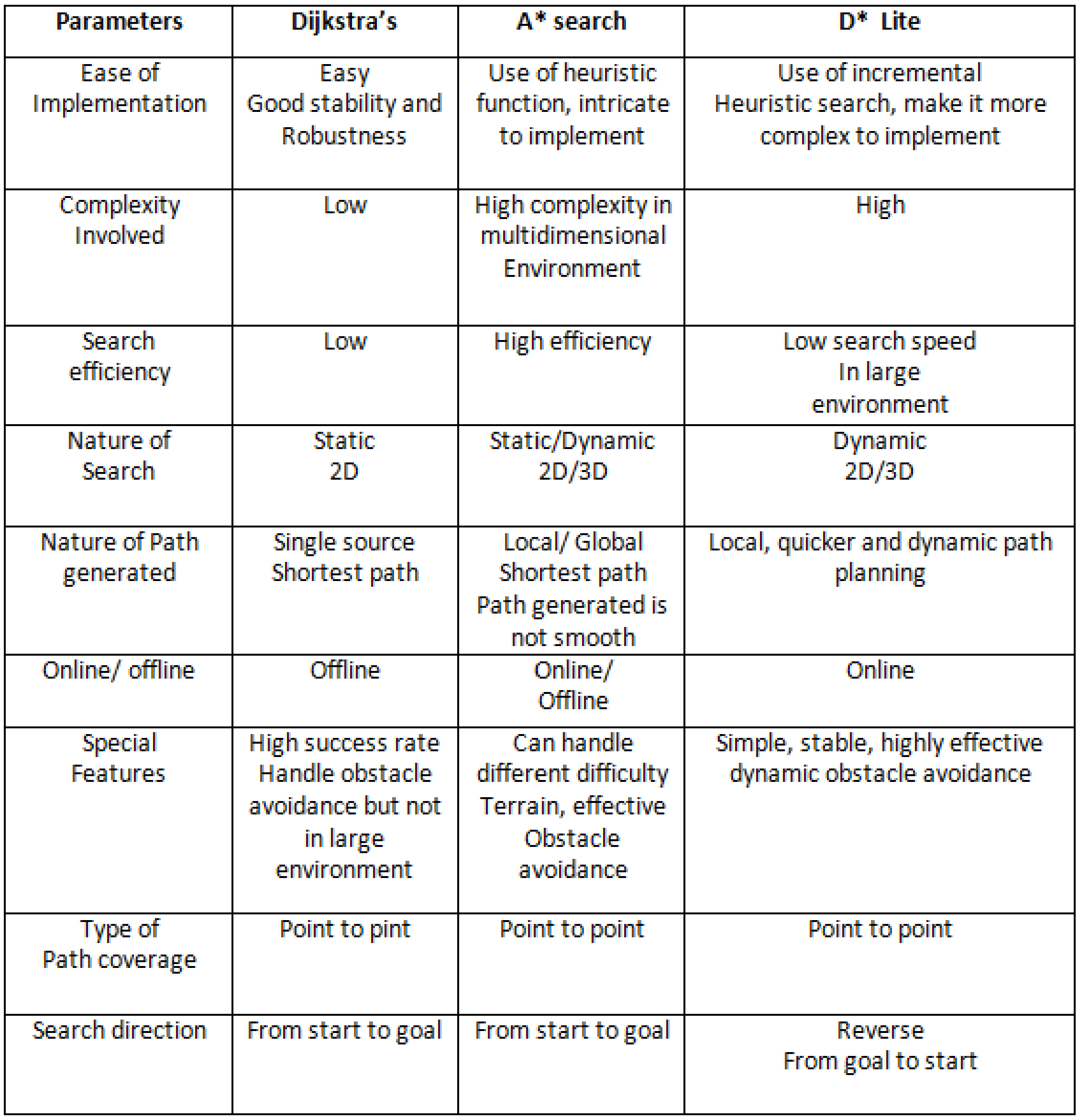
3.3.7 Pseudocode of D* Lite Algorithm

D* Lite

```
Key$s$ return $[\min(g(s), rhs(s)) + h(s_{start}, s); \min(g(s), rhs(s))]$ UpdateVertex$s$ if $s \neq s_{goal}$ then
3:     $rhs(s) \leftarrow \min_{s' \in Succ(s)}(c(s,s') + g(s'))$
4: end if
5: if $s \in OPEN$ then
6:     $OPEN.remove(s)$
7: end if
8: if $g(s) \neq rhs(s)$ then
9:     $OPEN.insert(s, Key(s))$
10: end if
```

```
    ComputePath
11: while $OPEN.TopKey() < Key(s_{start})$ or $rhs(s_{start}) \neq g(s_{start})$ do
12:   $k_{old} \leftarrow OPEN.TopKey()$
13:   $u \leftarrow OPEN.Pop()$
14:   if $k_{old} < Key(u)$ then
15:     $OPEN.insert(u, Key(u))$
16:   else if $g(u) > rhs(u)$ then
17:     $g(u) \leftarrow rhs(u)$
18:     for all $s \in Pred(u)$ do
19:       UpdateVertex($s$)
20:     end for
21:   else
22:     $g(u) \leftarrow \infty$
23:     for all $s \in Pred(u) \cup \{u\}$ do
24:       UpdateVertex($s$)
25:     end for
26:   end if
27: end while
    Main
28: for all $s \in S$ do
29:   $rhs(s) \leftarrow g(s) \leftarrow \infty$
30: end for
31: $rhs(s_{goal}) \leftarrow 0; k_m \leftarrow 0$
32: $OPEN.insert(s_{goal}, Key(s_{goal}))$
33: ComputePath()
34: while $s_{start} \neq s_{goal}$ do
35:   $s_{start} \leftarrow \arg\min_{s' \in Succ(s_{start})}(c(s_{start}, s') + g(s'))$
36:   Move to $s_{start}$
37:   Scan for cell changes in environment (e.g., sensor ranges)
38:   if cell changes detected then
39:     $k_m \leftarrow k_m + h(s_{last}, s_{start})$
40:     $s_{last} \leftarrow s_{start}$
41:     for all $s \in CHANGES$ do
42:       $c(s, s') \leftarrow$ cost of travel across cell $s$ to cell $s'$
43:     end for
44:     for all $s \in Pred(s') \cup \{s'\}$ do
45:       UpdateVertex($s$)
46:     end for
47:     ComputePath()
48:   end if
49: end while
```

3.3.8 Comparison between the search based algorithms



Parameters	Dijkstra's	A* search	D* Lite
Ease of Implementation	Easy Good stability and Robustness	Use of heuristic function, intricate to implement	Use of incremental Heuristic search, make it more complex to implement
Complexity Involved	Low	High complexity in multidimensional Environment	High
Search efficiency	Low	High efficiency	Low search speed In large environment
Nature of Search	Static 2D	Static/Dynamic 2D/3D	Dynamic 2D/3D
Nature of Path generated	Single source Shortest path	Local/ Global Shortest path Path generated is not smooth	Local, quicker and dynamic path planning
Online/ offline	Offline	Online/ Offline	Online
Special Features	High success rate Handle obstacle avoidance but not in large environment	Can handle different difficulty Terrain, effective Obstacle avoidance	Simple, stable, highly effective dynamic obstacle avoidance
Type of Path coverage	Point to pint	Point to point	Point to point
Search direction	From start to goal	From start to goal	Reverse From goal to start

Fig 3.3 Comparison between Different Search based Path Planning Algorithm

3.4 Sampling Based Algorithm

Sampling-based algorithms are a class of algorithms used in robotics and path planning to find feasible paths in high-dimensional spaces. They work by randomly sampling points in the search space and connecting them to form a graph or tree, effectively exploring the space without needing a complete model. These algorithms are particularly useful in environments with complex or unknown obstacles, as they can efficiently find paths with fewer computational resources compared to exhaustive search methods. Popular examples include the Probabilistic Roadmap (PRM) and Rapidly-exploring Random Tree (RRT) algorithms.
Several features of Sampling based algorithm
Computational Savings: Sampling-based methods offer computational efficiency by avoiding explicit construction of obstacles in the state space, providing probabilistic completeness guarantees as

the number of samples increases.

Influential Algorithms: Probabilistic RoadMaps (PRMs) and Rapidly-exploring Random Trees (RRTs) are highlighted as influential algorithms in the field, with PRMs focusing on multiple-query methods and constructing a graph connecting sampled points, while RRTs emphasize incremental tree construction.

Visibility Properties: The success of sampling-based algorithms is attributed to the assumption of good "visibility" properties in practical robotic applications, contributing to the exponential decay of the probability of failure in finding a solution .

Algorithm Differences: While both PRMs and RRTs connect randomly sampled points, they differ in their graph construction approaches, with PRMs being multiple-query methods and RRTs focusing on incremental tree building.

Completeness Guarantees: Sampling-based algorithms provide probabilistic completeness guarantees, ensuring that a solution will be found if one exists as the number of samples approaches infinity.

3.4.1 Probabilistic Road Map(PRM)

The Probabilistic Roadmap (PRM) planner is a motion planning algorithm used in robotics to find a path from a starting point to a goal point while avoiding obstacles. PRM works by constructing a roadmap (graph) of feasible paths through the robot's configuration space.
In the construction phase, the algorithm begins by randomly sampling points within the configuration space. Each sampled point is checked to ensure it lies in the free space, meaning it does not collide with any obstacles. These points are then connected to their nearby neighbors using a local planner, forming edges in the graph. This process is repeated until the graph is sufficiently dense, meaning it has enough points and connections to provide a good approximation of possible movements within the environment.
Once the roadmap is constructed, the query phase begins. The start and goal configurations are added to the graph, and connections are made from these points to the nearest points in the roadmap. The algorithm then uses a graph search technique, such as Dijkstra's shortest path algorithm, to find the optimal path from the start to the goal configuration. By leveraging random sampling and local planning, PRM efficiently explores the configuration space and finds feasible paths around obstacles.

3.4.2 Pseudocode of PRM

Algorithm 3 PRM (preprocessing phase)

```
1: $V \leftarrow \emptyset$; $E \leftarrow \emptyset$
2: for $i = 0$ to $n$ do
3:     $x_{\text{rand}} \leftarrow \text{SampleFree}(i)$
4:     $U \leftarrow \text{Near}(G = (V, E), x_{\text{rand}}, r)$
5:     $V \leftarrow V \cup \{x_{\text{rand}}\}$
6:     for all $u \in U$, in order of increasing $\|u - x_{\text{rand}}\|$ do
7:         if $x_{\text{rand}}$ and $u$ are not in the same connected component of $G = (V, E)$ then
8:             if $\text{CollisionFree}(x_{\text{rand}}, u)$ then
9:                 $E \leftarrow E \cup \{(x_{\text{rand}}, u), (u, x_{\text{rand}})\}$
10:            end if
11:        end if
12:    end for
13: end for
14: return $G = (V, E)$
```

3.4.3 PRM* - An Optimized version of PRM

The PRM* algorithm represents a significant advancement in sampling-based motion planning, aiming to address the limitations of existing algorithms and provide asymptotically optimal solutions. Key points about the PRM* algorithm include:
Optimality: PRM* is designed to offer solutions that converge almost surely to the optimal value as the number of samples increases, distinguishing it from traditional PRM and RRT algorithms.
Computational Efficiency: Despite its focus on optimality, PRM* maintains computational efficiency within a constant factor of its probabilistically complete counterparts, ensuring practical applicability in real-world scenarios.
Variable-Radius Approach: PRM* incorporates a variable-radius strategy, scaling the radius with the number of samples to balance optimality and efficiency in multiple-query problems.
Batch Processing: PRM* operates as a batch variable-radius PRM, facilitating multiple-query applications by adjusting the radius dynamically based on the sample size to achieve optimal solutions.
Connectivity and Performance: By building a connected roadmap incrementally, PRM* ensures similar performance to PRM* in a single-query setting and provides anytime solutions that can be continuously improved with additional computation time.

Overall, the PRM* algorithm represents a significant advancement in sampling-based motion planning, offering a balance between optimality and computational efficiency while providing solutions that converge to the optimal value with high probability. Its innovative approach to variable-radius sampling and batch processing makes it a valuable tool for addressing complex motion planning problems in robotics and related fields.

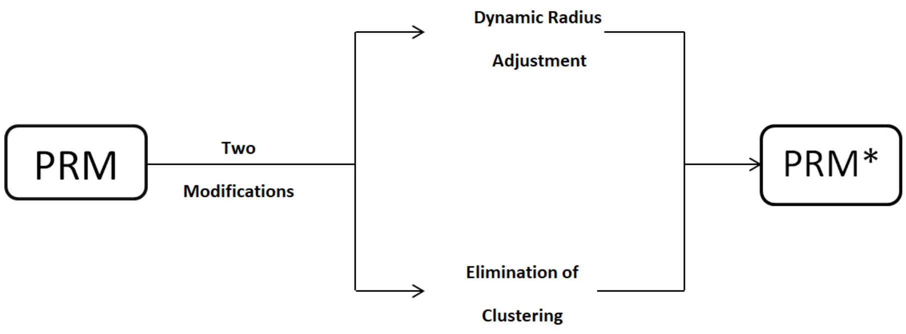


Fig 3.4 Depicts the relation between PRM and PRM*

3.4.4 Rapidly Exploring Random Tree(RRT)

Rapidly-exploring Random Trees (RRT) is a widely used algorithm that addresses both challenges simultaneously. RRT is designed to efficiently explore large, high-dimensional spaces by incrementally building a tree structure. The algorithm begins at the starting point and expands by randomly sampling points within the configuration space. For each sampled point, the algorithm finds the nearest node in the existing tree and attempts to create a new edge towards the sample. If the edge is collision-free, the new point is added as a node, and the process repeats.

The primary advantage of RRT is its ability to quickly cover the configuration space, making it well-suited for high-dimensional and complex environments. However, the paths generated by RRT are not guaranteed to be optimal. The focus is on finding a feasible path rather than the shortest path. Despite this, RRT's efficiency and simplicity make it a popular choice for many practical applications in robotic path planning.

3.4.5 Pseudocode of RRT Algorithm

Algorithm 4 BuildRRT

```
Require: Initial configuration $q_{\text{init}}$, number of vertices in RRT $K$, incremental distance $\Delta q$
Ensure: RRT graph $G$
1: $G$.init($q_{\text{init}}$)
2: for $k = 1$ to $K$ do
3:     $q_{\text{rand}} \leftarrow$ RAND_CONF()
4:     $q_{\text{near}} \leftarrow$ NEAREST_VERTEX($q_{\text{rand}}, G$)
5:     $q_{\text{new}} \leftarrow$ NEW_CONF($q_{\text{near}}, q_{\text{rand}}, \Delta q$)
6:     $G$.add_vertex($q_{\text{new}}$)
7:     $G$.add_edge($q_{\text{near}}, q_{\text{new}}$)
8: end for
9: return $G$
```

3.4.6 RRT* - An Optimized version of RRT

Rapidly-exploring Random Trees (RRT) are a class of algorithms designed for path planning and exploration in robotics and artificial intelligence. RRT* represents an enhancement over the basic

RRT algorithm, introducing optimizations that aim to find near-optimal paths efficiently, especially in complex and obstacle-rich environments.

How RRT* is different from RRT
RRT* retains the fundamental principle of iteratively expanding a tree structure from an initial configuration towards a goal configuration, but introduces two crucial improvements:
Cost-Optimized Nodes: Each vertex in RRT* maintains a cost metric relative to its parent node, referred to as cost(). This metric records the distance traveled from the initial node to the current vertex via its parent. When a new node is added, RRT* evaluates a neighborhood of vertices within a fixed radius. If a path to the new node offers a cheaper cost() than the current closest path, the tree structure is updated to reflect this more cost-effective path. This approach tends to straighten paths and eliminate the cubic growth characteristic of basic RRT.
Dynamic Rewiring: After connecting a new node to its closest neighbor with an improved path, RRT* further evaluates neighboring nodes to determine if rewiring these nodes to the new node could reduce their cost(). This feature promotes smoother paths and enhances the adaptability of the tree structure to changing environments or goals.

3.4.7 Pseudocode of RRT* Algorithm

Algorithm 5 RRT* Algorithm

```
1: Inputs: $n$ (number of iterations), $r$ (radius), $G(V, E)$ (graph with vertices and edges)
2: for $itr = 1$ to $n$ do
3:     $X_{new} \leftarrow$ RandomPosition()
4:     while Obstacle($X_{new}$) do
5:         $X_{new} \leftarrow$ RandomPosition()
6:     end while
7:     $X_{nearest} \leftarrow$ Nearest($G(V, E), X_{new}$)
8:     Cost($X_{new}$) $\leftarrow$ Distance($X_{new}, X_{nearest}$)
9:     $(X_{best}, X_{neighbors}) \leftarrow$ findNeighbors($G(V, E), X_{new}, r$)
10:    $Link \leftarrow$ Chain($X_{new}, X_{best}$)
11:    for $x' \in X_{neighbors}$ do
12:        if Cost($X_{new}$) + Distance($X_{new}, x'$) $<$ Cost($x'$) then
13:            Cost($x'$) $\leftarrow$ Cost($X_{new}$) + Distance($X_{new}, x'$)
14:            Parent($x'$) $\leftarrow X_{new}$
15:            $G \leftarrow G \cup \{(X_{new}, x')\}$
16:        end if
17:    end for
18:    $G \leftarrow G \cup$ Link
19: end for
20: return $G$
```

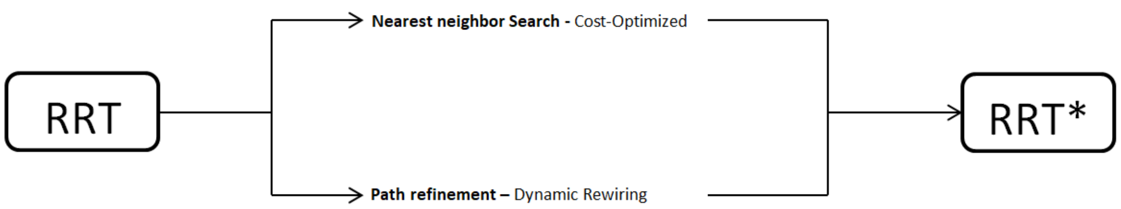

RRT
Nearest neighbor Search - Cost-Optimized
Path refinement – Dynamic Rewiring
RRT*

Figure 3.5 Depicts the relation between RRT and RRT*

3.4.8 Rapidly Exploring Random Graph(RRG)

Rapidly exploring random graphs (RRG) are a fundamental structure in the field of path planning and robotics, particularly when navigating complex and high-dimensional spaces. Unlike traditional tree-based methods, RRG maintains a graph structure where each node can connect to multiple other nodes, facilitating more efficient exploration and connectivity.
The RRG algorithm constructs a graph incrementally, where each node represents a point in the configuration space and is connected to other nodes based on specific criteria. Key properties of RRG include:

Graph Connectivity: The entire graph may consist of a single interconnected component or multiple disconnected sub-graphs, allowing for flexible representation of the explored space.

Node Distribution: The minimum distance r between any two nodes ensures that nodes are not too densely packed, which helps to control computational costs while maintaining quality. The distance r is generally proportional to the step size used to expand nodes.

Node Attributes: Each node is characterized by its location in the configuration space and a color that indicates its origin. The color represents the initial node from which the current node was derived, ensuring that the lineage of nodes is tracked.

Collision-Free Connections: Nodes within a distance of less than 2*stepsize are connected if the path between them is free of obstacles. This promotes efficient exploration by ensuring that close nodes can share information about the space.

Redundant Connectivity: Unlike the RRT algorithm, RRG allows for multiple connections between nodes of the same color, enhancing the robustness of the graph. Nodes can also connect to nodes of different colors through bridges, which link different sub-graphs and improve overall connectivity.

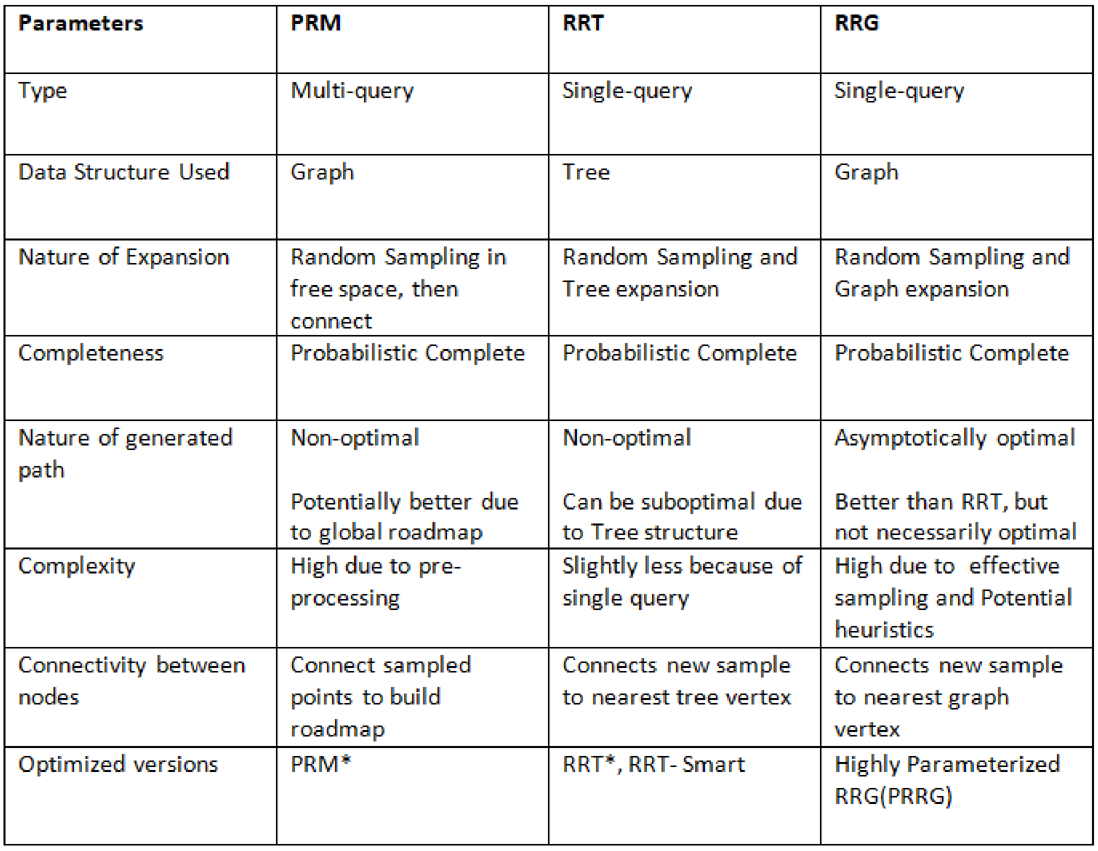
Exploration Boundary: An important feature of RRG is the ability to construct a boundary or cover around all nodes, defining the explored region of the configuration space. New nodes are not generated within this cover once it encompasses the entire free space, allowing the algorithm to terminate efficiently.

3.4.9 Pseudocode of RRG Algorithm

Algorithm 6 RRG Expansion

```
Require: $\tilde{x}_{pos}$, $d_{min}$, $d_{max}$
 1: $G \leftarrow$ initGraph($\tilde{x}_{pos}$)
 2: while !explorationFinished() do
 3:     $V_{av} \leftarrow$ retrieveMap()
 4:     $\tilde{x}_{rand} \leftarrow$ randomlySamplePoint($V_{av}$)
 5:     $\tilde{x}_n \leftarrow$ findNearestNeighbour($G, \tilde{x}_{rand}$)
 6:     if $d(\tilde{x}_{rand}, \tilde{x}_n) \geq d_{min}$ then
 7:         $\tilde{x}_{rand} \leftarrow$ alignSamplePoint($\tilde{x}_{rand}, d_{max}$)
 8:         $N_d \leftarrow$ findNodesInRadius($G, \tilde{x}_{rand}, d_{max}$)
 9:         $N_c \leftarrow \emptyset$
10:         for $\tilde{x}_d \in N_d$ do
11:             $N_c \leftarrow N_c \cup$ steer($V_{av}, \tilde{x}_{rand}, \tilde{x}_d$)
12:         end for
13:         if $N_c \neq \emptyset$ then
14:             addNodeToGraph($G, \tilde{x}_{rand}, N_c$)
15:         end if
16:     end if
17: end while
```

3.4.10 Comparison between Sampling based Algorithms



Parameters	**PRM**	**RRT**	**RRG**
Type	Multi-query	Single-query	Single-query
Data Structure Used	Graph	Tree	Graph
Nature of Expansion	Random Sampling in free space, then connect	Random Sampling and Tree expansion	Random Sampling and Graph expansion
Completeness	Probabilistic Complete	Probabilistic Complete	Probabilistic Complete
Nature of generated path	Non-optimal Potentially better due to global roadmap	Non-optimal Can be suboptimal due to Tree structure	Asymptotically optimal Better than RRT, but not necessarily optimal
Complexity	High due to pre-processing	Slightly less because of single query	High due to effective sampling and Potential heuristics
Connectivity between nodes	Connect sampled points to build roadmap	Connects new sample to nearest tree vertex	Connects new sample to nearest graph vertex
Optimized versions	PRM*	RRT*, RRT- Smart	Highly Parameterized RRG(PRRG)

Figure 3.6 Comparison between various Sampling Based Path Planning Algorithms

3.4.11 Summary of Path Planning Algorithms

In this chapter,we have gone through various search and sampling based algorithms and have reviewed the comparison of those algorithms through table.
Path planning algorithms can be broadly categorized into search-based and sampling-based methods. Search-based algorithms like Dijkstra's, A*, and D* Lite provide deterministic and optimal solutions, excelling in well-defined, static environments. On the other hand, sampling-based algorithms such as RRT, RRT*, PRM, PRM*, and RRG are better suited for high-dimensional, complex spaces, offering quicker, feasible paths without guaranteeing optimality. Each algorithm has unique strengths, making the choice dependent on specific application requirements and environmental constraints.

Chapter 4

Implementation of Glasius Bio Inspired Neural Network(GBNN) for Complete Coverage Path Planning

4.1 Introduction

In this chapter, the method of complete coverage path planning of AUV with the integration of Glasius Bio Inspired Neural Network(GBNN) algorithm is being discussed.First, the underwater workspace of the AUV is discretized to create a two-dimensional grid map. A corresponding neural network is then constructed based on this grid map. Next, the neural network activity is continually updated using the GBNN algorithm according to the grid map's state. Finally, the complete coverage path for the AUV is determined through the path planning strategy.And the obstacle avoidance is also achieved using specific technique.

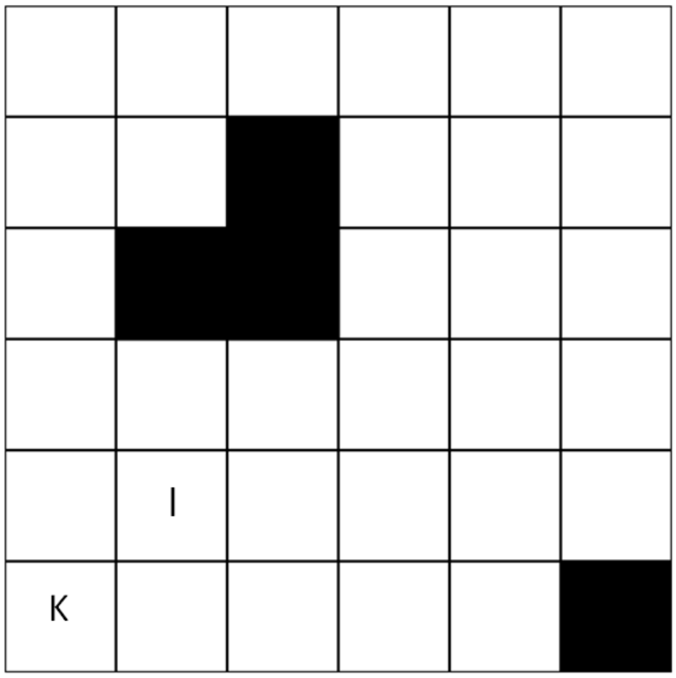


Figure 4.1 Model of two-dimensional underwater environment

And the below figure will represent the two dimensional neural network of the grid

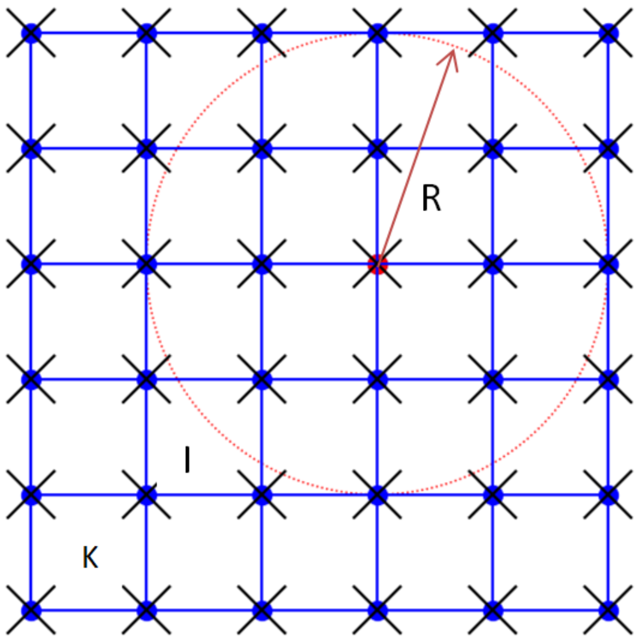


Figure 4.2 Two dimensional Neural Network

4.2 Glasius Bio Inspired Neural Network(GBNN)

GBNN algorithm is a discrete-time Hopfield-type neural network, whose dynamic model is described as

$$x_i(t+1) = g\left(\sum_{j=1}^{M} W_{ij}[x_j(t)]^+ + I_i\right)$$

Where the transfer function is chosen as :

$$g(x) = \begin{cases} -1, & x < 1 \\ \beta x \ , & 0 \leqslant x < 1,\ \beta > 0 \\ 1, & x \geq 1 \end{cases}$$

where,$x_i(t+1)$ is the neural activity of the i^{th} neuron at $(t+1)$

$[x_j(t)]^+$ is the neural activity of j^{th} neuron at t and β is a positive scalar constant

j^{th} neuron is laterally connected with i^{th} neuron

M is the number of neural connections of the i^{th} neuron to its neighboring neuron within the R, which is receptive field

W_{ij} is the connection weight between ith and jth neuron and it is defined as

$$W_{ij} = \begin{cases} e^{-\alpha|i-j|^2}, & \text{if } 0 < |i-j| \leq R \\ 0, & \text{if } |i-j| > R \end{cases}$$

where $|i-j|$ is the Euclidean distance between ith neuron and jth neuron, α and R are all positive constants, and R is the radius of the receptive field

$W_{ij} = W_{ji}$ and I_i is the external input of the ith neuron, which is given as

$$I_i = \begin{cases} +E, & \text{if it is an uncovered area} \\ -E, & \text{if it is an obstacle area} \\ 0, & \text{if it is a covered area} \end{cases}$$

Where E is the external excitatory input and E is the external inhibitory input.To ensure that the neural activities in uncovered areas are at their highest while those in obstacle areas are at their lowest, the external input E must exceed the total excitatory inputs from the lateral neural connections, which means,

$$E \gg \sum_{j=1}^{M} W_{ij}[x_j(t)]^+, \quad E \gg 1 \quad \text{and} \quad \sum_{j=1}^{M} W_{ij}[x_j(t)]^+ \in [0,1)$$

The neural activity is bounded in the finite interval $\in [-1,1]$, which guarantees that the neural network is stable. If the i-th neuron is uncovered, its external inputs include the external excitatory input: $I_i = +E$ and the sum of excitatory inputs:

$$\sum_{j=1}^{M} W_{ij}[x_j(t)]^+$$

and

$$\sum_{j=1}^{M} W_{ij}[x_j(t)]^+ + E \gg 1,$$

therefore, the neural activity of the uncovered neuron is 1. If the i-th neuron is an obstacle, its external input is made of the external inhibitory input: $I_i = -E$ and the sum of excitatory inputs from the lateral neural connections:

$$\sum_{j=1}^{M} W_{ij}[x_j(t)]^+,$$

and

$$\sum_{j=1}^{M} W_{ij}[x_j(t)]^+ - E < 0$$

which guarantees that the neural activity of the obstacle area is -1. If the i-th neuron is covered, its external input: $I_i = 0$ and the excitatory inputs only from the sum of excitatory inputs from the lateral neural connections, due to the

$$\sum_{j=1}^{M} W_{ij}[x_j(t)]^+ \in [0,1),$$

the neural activity of the covered area is $\beta \sum_{j=1}^{M} W_{ij}[x_j(t)]^+ \in [0,1)$, generally $0 < \beta \leq 1$.

4.2.1 GBNN Algorithm

4.2.2 Outline of summary of GBNN Algorithm

Dynamic Neural Activity: The neural activity changes dynamically in response to the environment.

Propagation of Positive Neural Activity: Positive neural activity can spread throughout the entire workspace via lateral connections among neurons, attracting the AUV to visit uncovered areas.

Algorithm 7 Overall GBNN Operation

```
Require: Metric map, Updates for map
Ensure: Robot navigation path
 1: Initialization:
 2: Create grid map
 3: Construct corresponding neural network
 4: Set initial position of the robot as current position
 5: Initiate coverage
 6: while number of uncovered cells $\neq 0$ do
 7:     Change the status of current cell to covered
 8:     Update neural activities
 9:     Select next cell for navigation
10:     Robot moves to next cell
11:     Set new location as the current location
12:     if update for grid map is available then
13:         Update grid map
14:     end if
15: end while
```

Localization of Negative Neural Activity: Negative neural activity remains localized and does not propagate outward, ensuring that obstacle areas have a local repulsive effect to push the AUV away and avoid collisions.

Covered Areas:The external excitatory inputs to neurons in covered areas become zero, making these areas unattractive to the AUV.The excitatory inputs in covered areas originate solely from the sum of incentive values transmitted by adjacent neurons.

Gradual Decay in Covered Areas:As uncovered areas become covered, the excitatory inputs in these covered regions decrease.Consequently, the neural activities in covered areas steadily decay to zero.

4.3 Hopfield Neural Network

The Hopfield Neural Networks, invented by Dr John J. Hopfield consists of one layer of n fully connected recurrent neurons. It is generally used in performing auto-association and optimization tasks. It is calculated using a converging interactive process and it generates a different response than our normal neural nets.

Since Glasius Bio Inspired Neural Network is a discrete time Hopfield type neural network, it is important to highlight the fundamental concepts of Hopfield Neural Networks.

4.3.1 Hopfield Neural Network Architecture

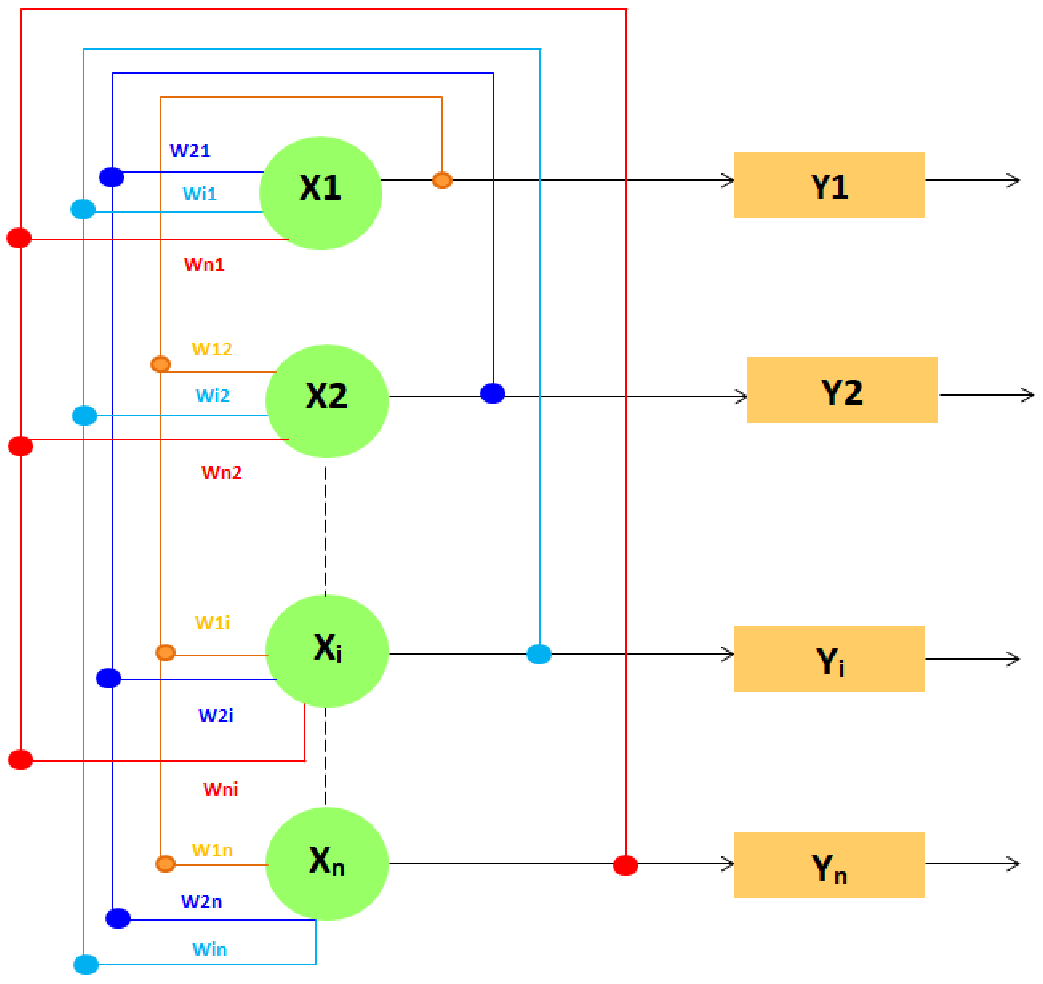


Figure 4.3 Architecture of Hopfield Neural Network

4.3.2 Characteristics of Hopfield Neural Network

Weight Symmetry: The weights between neurons are symmetric, $W_{ij} = W_{ji}$. No self-connections that is $W_{ii} = 0$.

Neural Outputs: Each neuron has both an inverting and a non-inverting output.The output of each neuron serves as an input to all other neurons except itself.

Output Behavior:Operates in a discrete manner, providing finite distinct outputs, typically in two forms
Binary (0/1)
Bipolar (-1/1)

Connectivity: Each unit (neuron) is connected to every other unit in the network, except with its own ie no self connection.

Calculation Method:Operates using a converging interactive process.Generates a different response compared to traditional neural networks.

4.4 Complete Coverage Path Planning Strategy

In this section, the strategy which is being used for the complete coverage path planning is discussed. In this section, the $Theta^{*} Algorithm$ is used for any angle path planning and good obstacle avoidance and the $Boustrophedon Algorithms$ is used to create the complete coverage path like a lawn-mower.

4.4.1 Theta-star Algorithm

Theta* is an efficient and flexible path planning algorithm introduced by Nash et al. in 2007. It is designed to perform any-angle path planning on grids, which allows it to produce paths that are not restricted to grid edges. This flexibility results in more natural and shorter paths compared to traditional grid-based algorithms like A*. Features of Theta star Algorithm

Any-Angle Path Planning:Contrastive to A*, which restricts movement to grid edges, Theta* allows movement directly between any two points, as long as there are no obstacles in the way. This results in paths that are more direct and resemble true shortest paths.

Line-of-Sight Check:The line-of-sight check in the context of the Theta* algorithm is a crucial step that determines whether a direct path between two nodes is free of obstacles. This check is what allows Theta* to create more direct and smoother paths compared to traditional grid-based algorithms like A*.

Optimality and Efficiency:Theta* can generate more optimal paths than A* by minimizing unnecessary turns and zigzagging. Nonetheless, the computational cost of performing line-of-sight checks may make Theta* somewhat less efficient than A* in certain situations. Even so, the benefit of smoother paths often outweighs the extra computational expense.

Theta* improves upon the A* algorithm by using line-of-sight checks to enable direct, any-angle movements between nodes. This leads to smoother and often shorter paths, making it ideal for applications where natural and efficient pathfinding is important. The strength of Theta* comes from its innovative approach to cost functions, heuristic estimates, and direct path validation, distinguishing it from conventional grid-based pathfinding methods.

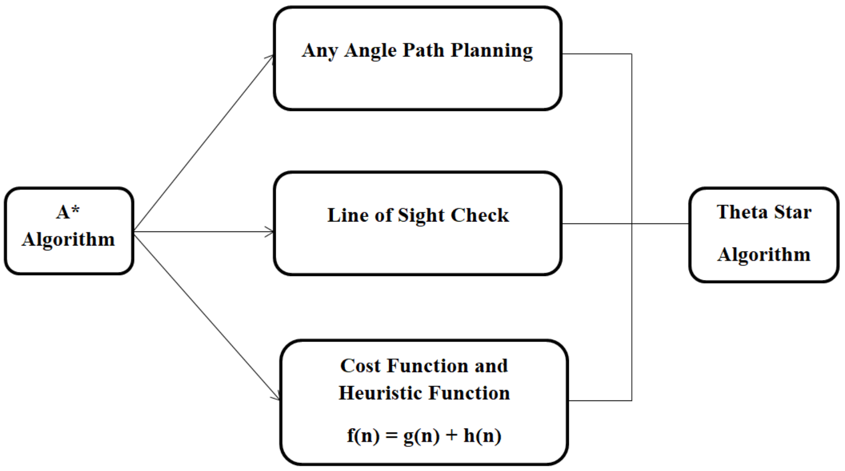


Figure 4.4 Depicts the interrelation between Theta* and A* algorithm

Algorithm 8 Basic Theta*

```
     UpdateVertex$s, s'$ if LineOfSight(parent($s$), $s'$) then Path 2
 2:  if $g$(parent($s$)) + $c$(parent($s$), $s'$) < $g(s')$ then
 3:    $g(s') \leftarrow g(\text{parent}(s)) + c(\text{parent}(s), s')$
 4:    parent($s'$) ← parent($s$)
 5:    if $s' \in$ open then
 6:      open.Remove($s'$)
 7:    end if
 8:    open.Insert($s', g(s') + h(s')$)
 9:  end if
10: else
       Path 1 if $g(s) + c(s, s') < g(s')$ then
12:    $g(s') \leftarrow g(s) + c(s, s')$
13:    parent($s'$) ← $s$
14:    if $s' \in$ open then
15:      open.Remove($s'$)
16:    end if
17:    open.Insert($s', g(s') + h(s')$)
18:  end if
19: end if
```

4.4.2 Reasons for using Theta* Algorithm over A* Algorithm

Initially, an attempt was made to implement the A* Algorithm along with the Boustrophedon algorithm for complete coverage path planning. However, several difficulties were encountered during implementation.
Obstacle Avoidance: The A* algorithm does not allow diagonal turns, which sometimes results in the path being constructed over obstacles, an undesired outcome.

Path Smoothing: Due to the movement restrictions in A*, the generated path is not smooth and contains numerous sharp turns. This is undesirable for an autonomous vehicle, as it would cause jerks in the system.

Efficiency in Path planning: Theta* tends to find shorter and more direct paths compared to A*. This is because Theta* allows any-angle paths, reducing the total travel distance and increasing the efficiency of the path finding process.

Real-World Applicability: In real-world scenarios, the ability to navigate smoothly and avoid obstacles with greater flexibility is crucial. Theta* is more suitable for applications requiring realistic and practical movement, such as autonomous vehicles, due to its capability to handle complex environments more effectively than A*.

4.4.3 Boustrophedon Algorithm

The Boustrophedon algorithm is a widely recognized method for complete coverage path planning, commonly utilized in robotics for activities like lawn mowing, floor cleaning, and autonomous underwater vehicle navigation. The name "boustrophedon" is derived from the Greek words meaning "ox-turning," which describes the back-and-forth motion of an ox plowing a field. This approach guarantees that the entire area is systematically covered without leaving any gaps.

Features of Boustrophedon Algorithms
Decomposition: The workspace is divided into smaller cells or regions that can be easily covered using simple back-and-forth motions.

Coverage Path Planning: Within each cell, the robot follows a systematic path that ensures complete coverage. Typically, this path is a series of parallel lines.

Obstacle Avoidance: The algorithm can handle obstacles by decomposing the space into smaller cells that avoid these obstacles.

Efficiency: It aims to minimize the amount of overlap and backtracking, making the path as efficient as possible.

To optimize time and energy efficiency, the robot must navigate the shortest path, avoiding previously visited areas and unnecessary turns. However, achieving complete coverage sometimes needs revisiting covered cells. Diagonal movements have been disabled, and their use is only allowed when these movements occur in deadlock situations. In most cases, when the environment is full of obstacles, it is impossible to achieve a single pass coverage.In these cases, more advanced strategies are required to complete the entire path of the workspace,as the deadlock detector and the escape route generator.

Algorithm 9 Boustrophedon Algorithm

```
1:  Init count = 0
2:  Start robot motion direction up
3:  while not finished do
4:      if next cell is obstacle or visited then
5:          if count is odd then
6:              Turn 90° clockwise
7:              Move forward one cell
8:              Turn 90° counterclockwise
9:              count = count + 1
10:         else
11:             Turn 90° counterclockwise
12:             Move forward one cell
13:             Turn 90° clockwise
14:             count = count + 1
15:         end if
16:     else
17:         Move forward one cell
18:     end if
19: end while
```

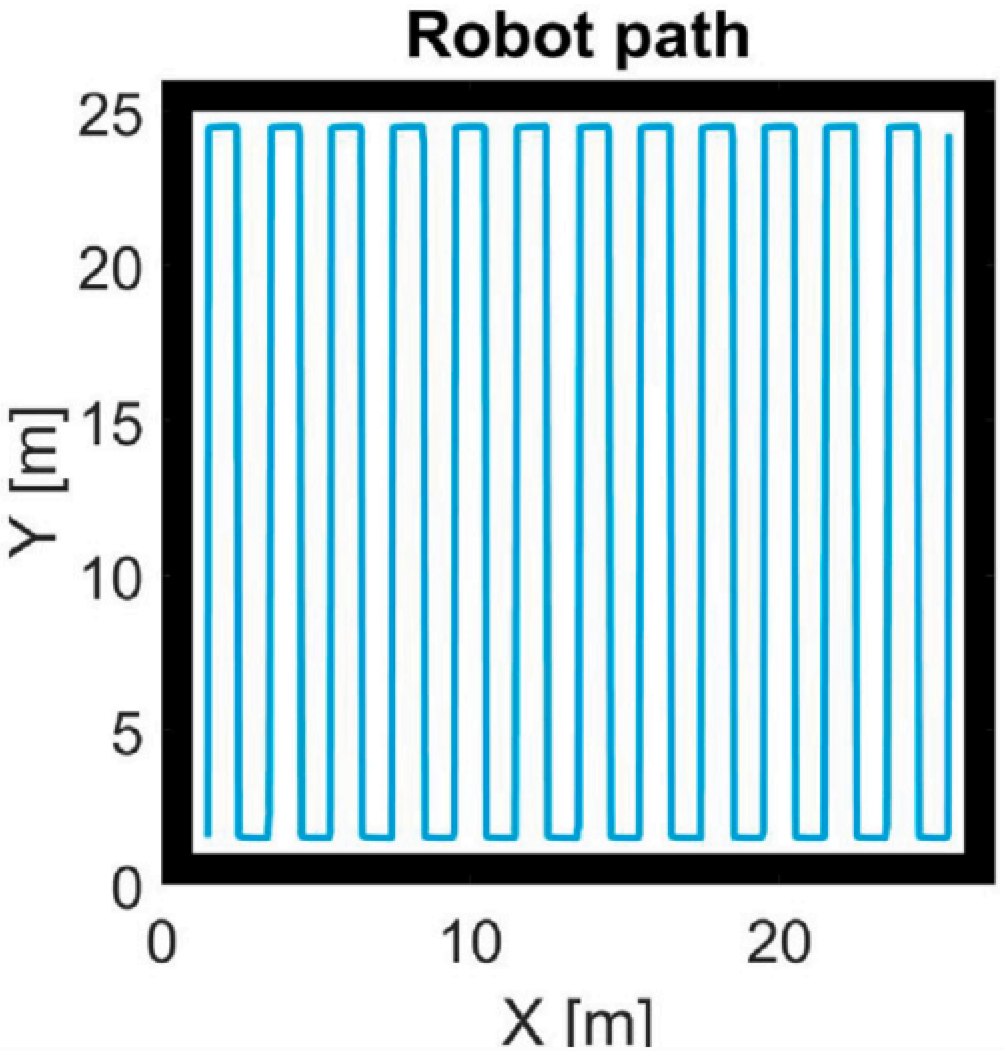


Figure 4.5 Path with Boustrophedon Algorithm for obstacle free grid

4.5 Summary

In this chapter, the comprehensive path planning strategy is outlined. The GBNN algorithm is employed to establish the grid environment and to initialize and update neural activity. Our objective is to achieve complete coverage path planning, with the path mimicking a lawn mower pattern. To achieve this, the Boustrophedon algorithm is implemented to create the lawn mower-like path. Additionally, the integration of the Theta* algorithm ensures effective obstacle avoidance and prevents deadlock. This combination of algorithms improves the efficiency and reliability of the path planning process, making it ideal for complex and dynamic environments.

Chapter 5

Results and Discussions

This chapter presents the results of the study and discusses their implications.Within the framework of the objectives and related research works, the results are analyzed.Algorithm performance, obstacle avoidance and coverage efficiency are the three main parameters comprising the results.Implementing a comprehensive coverage path planning algorithm into practice and assessing its effectiveness in terms of coverage efficiency and obstacle avoidance are the main goals of this study.Plots are being used to indicate the neural activity of the algorithm that was used, and performance analysis of the path's visual representation is included in the results.

5.1 Complete Coverage Path Planning in the Static Underwater Environment

The simulation environment is considered as a 30 X 30 grid, two dimensional underwater workspace,which is represented by a discretized grid map.
As the size of the grid map is 30 $\times$ 30 cells, thus the neural network has 30$\times$30 topologically organized neurons, where all the neural activities are initialized to zero. In addition,the AUV is simplified as a particle and its shape and size are not considered.

The parameters of the proposed CCPP along with GBNN are configured as follows throughout simulations

$$\alpha = 2, \beta = 0.7, R = 2, \text{ for the radius of the receptive field and } E = 100\text{for the external input}$$

These are the main parameters of GBNN algorithm, and these values are kept constant throughout for every simulation.The simulation results of complete coverage path of the AUV in the static and dynamic environment are implemented by Python.

In this case for completely static environment, the obstacles are being added manually and the performance of the algorithm is being evaluated through the visual representations of the path and the neural activities.

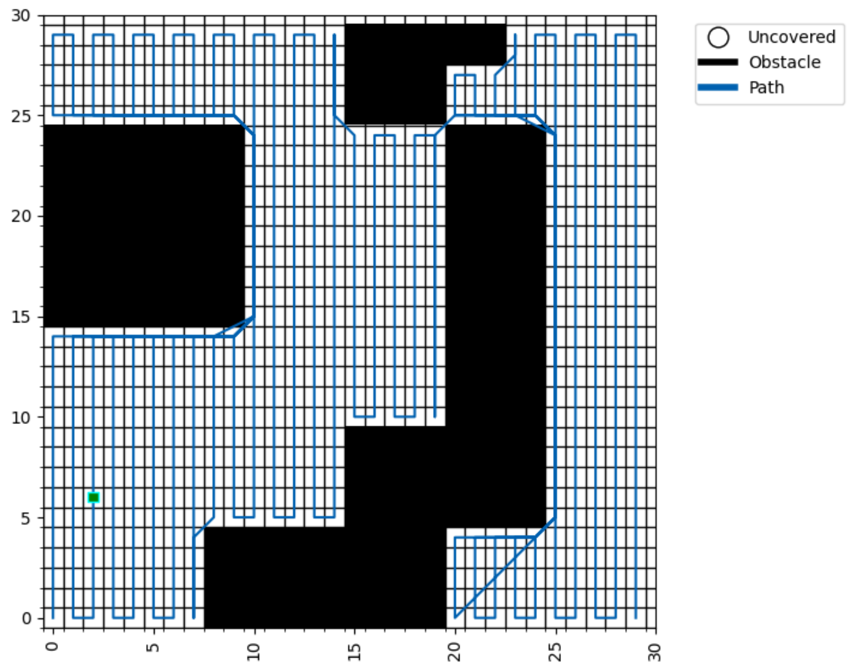


Figure 5.1 Complete coverage path of AUV in static underwater workspace

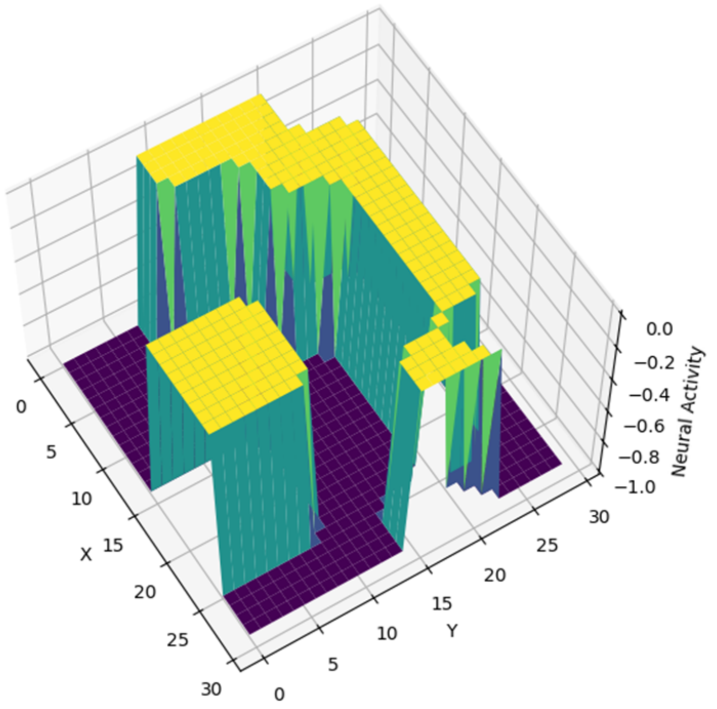


Figure 5.2 The neural activity landscape of neural network corresponding to the grid map

5.1.1 Analysis for the static environment case

Complete Coverage path planning of the AUV in the static environment is presented. In the static environment, obstacles are considerd as still without any moving.

The initial simulation environment is shown in Fig 5.1,there are multiple irregular obstacles in the grid map of 30X30 and the AUV begins to perform the complete coverage task from (1,1) to (30,1) Fig 5.2 is the neural activity landscape of the neural network corresponding to the grid map.Because of the large external inhibitory inputs (-E), the obstacle areas only have a localized effect, pushing the AUV away to avoid contact, with their neural activity remaining at -1. With large external excitatory inputs(+E), the neural activities of uncovered areas are 1.For the covered areas, their external inputs are zero and the excitatory inputs are only from the lateral neural connection and thus the neural activity of the covered areas decay from 1.

When the AUV arrives at location (20,24), it stuck in the deadlock, where all the neighboring locations are either obstacles or covered areas. For the neuron (20,24), its external input all becomes to be zero and the excitatory inputs are just from the lateral connections and then the neural activity decays..Neurons closer to the uncovered areas receive stronger excitatory inputs, meaning they are more activated or stimulated.The neural activities of neighboring neurons (except those representing obstacles) are higher than the activity at (20, 24) because they are closer to the uncovered areas.Due to the higher activity in neighboring neurons, the AUV can immediately plan a conventional point-to-point path to move away from the deadlock. The AUV doesn't need to wait or stay in place because the neural network helps it find a clear path out of the deadlock quickly.

To conclude, the neural network model enables the AUV to detect high-activity areas near uncovered regions and plan an immediate escape route from a deadlock, ensuring efficient and continuous movement without delays.

5.2 Complete Coverage Path Planning in the Semi Dynamic Underwater Environment

In this section, the dynamic underwater environment is defined as an environment with some dynamic obstacle. In this section, the simulation results of complete coverage path planning of the AUV in dynamic environment with suddenly appearing obstacles at the free configuration space at each iterations.

In this section, the performance of the GBNN algorithm along with the complete coverage path planning in the dynamic obstacle environment is being discussed interms of visualization of path and neural activity as well.

The parameters of the proposed CCPP along with GBNN are configured as follows throughout simulations

$$\alpha = 2, \beta = 0.7, R = 2, \text{ for the radius of the receptive field and } E = 100 \text{ for the external input}$$

These are the main parameters of GBNN algorithm, and these values are kept constant throughout for every simulation.
Obstacle Addition: At each iteration, a single obstacle is added randomly to any of the free (unoccupied) positions within the grid. This ensures that the environment remains dynamic and presents new challenges for path planning in each iteration.

Path Adjustment: With the addition of each new obstacle, the planned path must be recalculated to navigate around the new obstacle. This recalculation involves updating the neural activity of the Glasius Bio-Inspired Neural Network (GBNN), which is responsible for generating the optimal path.

Neural Activity Update: The GBNN adjusts its neural activity to accommodate the new obstacle. This involves re-evaluating the connection weights and neural activations to generate a new path that avoids the obstacle while aiming to minimize path length and the number of turns.

Visualization: The changes in the path and the corresponding neural activity are depicted in plots. These plots illustrate the dynamic nature of the environment and the adaptability of the GBNN in response to the newly introduced obstacles.

By continuously introducing dynamic obstacles and recalculating the path, the simulation effectively demonstrates the robustness and flexibility of the GBNN in real-time path planning applications.

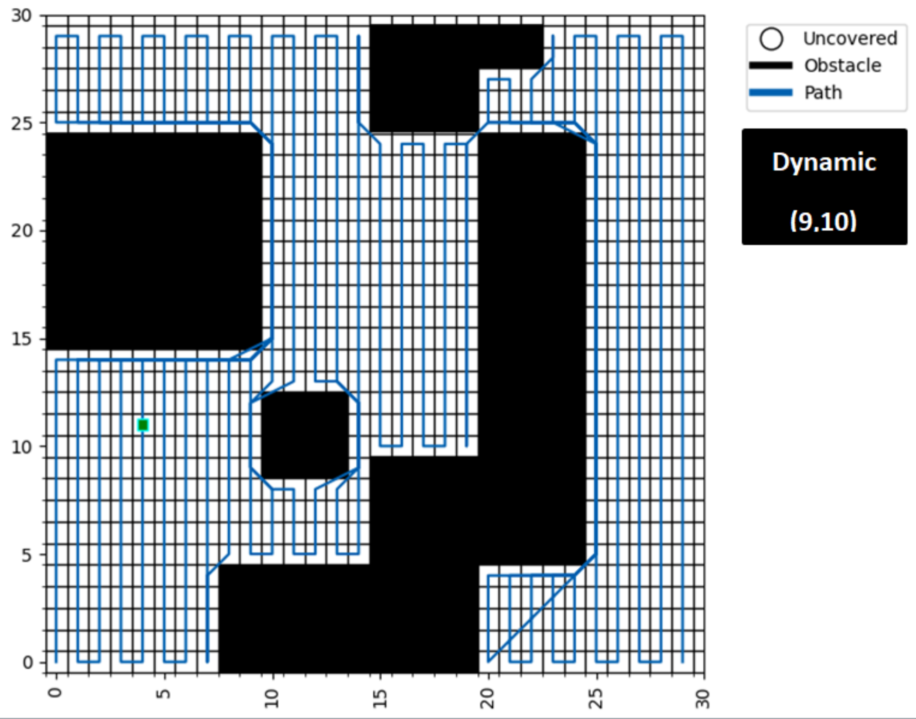


Figure 5.3 Complete coverage path in dynamic obstacle case

Sudden appearance of random obstacle at free space (9,10), the GBNN algorithm along with the complete coverage strategy using Boustrophedon and Theta star algorithm, is working well in detecting dynamic obstacle and the path is being reconstructed according to the situation.
As the path is being changed according to that the neural activities is also being updated and it also represents the effectiveness of the algorithm being used to evaluate the process.

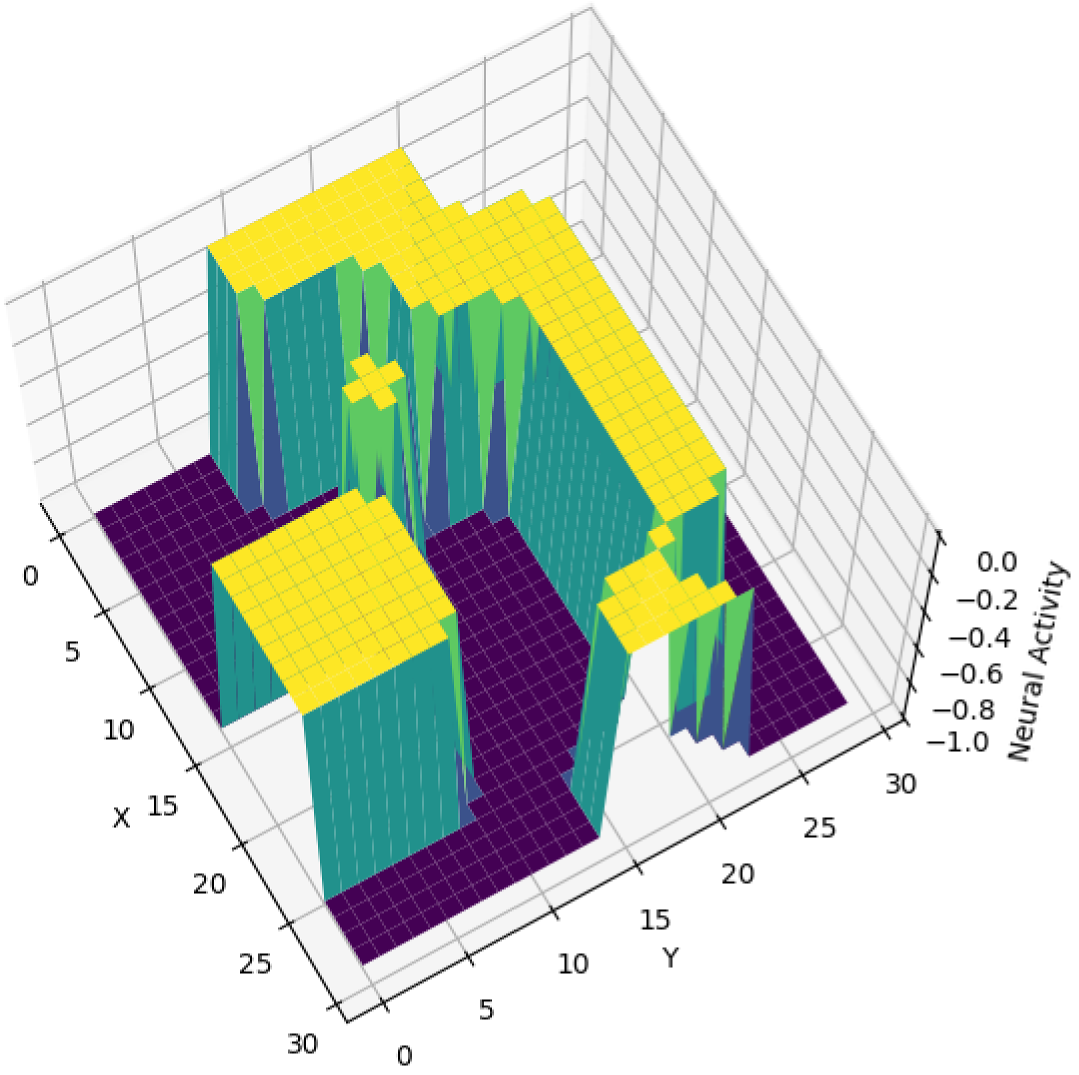


Figure 5.4 Change in Neural Activities due to addition of random obstacle at(9,10)

5.3 Complete Coverage Path Planning in the Fully Dynamic Underwater Environment

In this section, the creation of dynamic environment is being discussed.Creating a dynamic environment with clustered obstacles, the clustering process involves systematically placing groups of obstacles within a grid map. This approach aims to simulate real-world scenarios where obstacles are often concentrated in specific areas rather than being uniformly distributed. **Grid Map Initialization**The process begins with the initialization of a grid map with specified dimensions (rows x columns). Each cell in this grid initially represents free space and is assigned a value of 0. This empty grid serves as the foundation upon which obstacles will be placed.

Calculating the Number of Obstacles To determine the number of obstacles to be placed within the grid, the obstacle-density parameter is used. This parameter represents the proportion of the grid cells that should be occupied by obstacles. By multiplying the total number of grid cells (rows x columns) by the obstacle-density, the total number of obstacles required is calculated.

Clustering Obstacles The next step involves grouping the obstacles into clusters, which includes the following sub-steps: **Cluster Size**: A cluster-size is defined to specify how many obstacles will be grouped together in each cluster. This ensures that obstacles are not spread evenly but are concentrated in specific areas.
Number of Clusters: The number of clusters needed is computed by dividing the total number of obstacles by the cluster-size. This calculation gives the number of obstacle groups to be placed within the grid.
Cluster Placement: For each cluster, a random center point is selected within the grid boundaries. This center point serves as the focal point around which obstacles in the cluster will be placed.
Around each center point, a defined number of obstacles (cluster-size) are placed by randomly selecting offsets within a small range . These offsets determine the relative positions of obstacles within the cluster. To ensure that each obstacle stays within the grid bounds, the exact position (x, y) is calculated. This is achieved by adjusting the center point coordinates with the offsets while making sure they do not exceed the grid limits.
Placing Obstacles: Finally, each calculated position (x, y) where an obstacle is to be placed is marked in the grid map with a value of 1. This marking indicates the presence of an obstacle at that position. By repeating this process for all clusters, the grid map is populated with obstacles arranged in clusters, creating a more realistic and dynamic environment.
The simulation in the complete dynamic environment is depicted below.

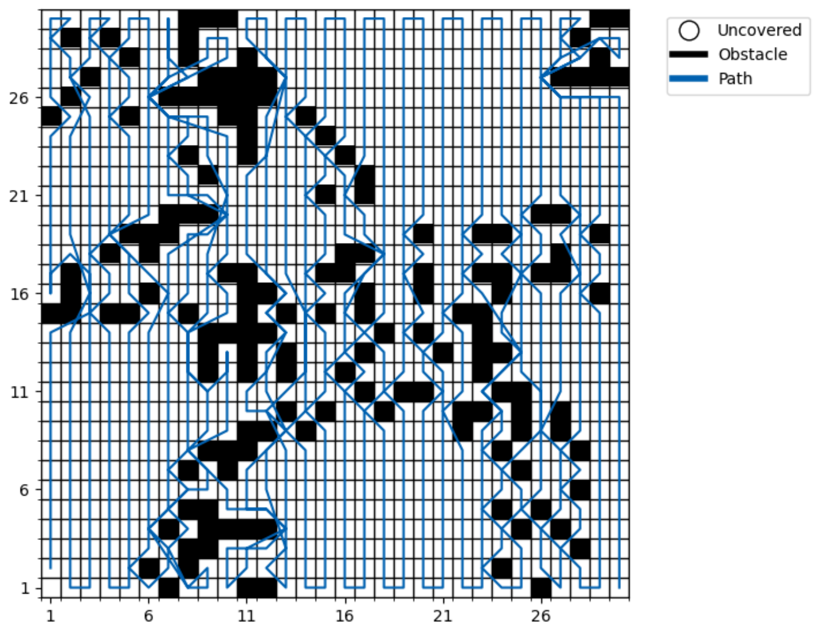


Figure 5.5 Path in completely dynamic obstacle environment

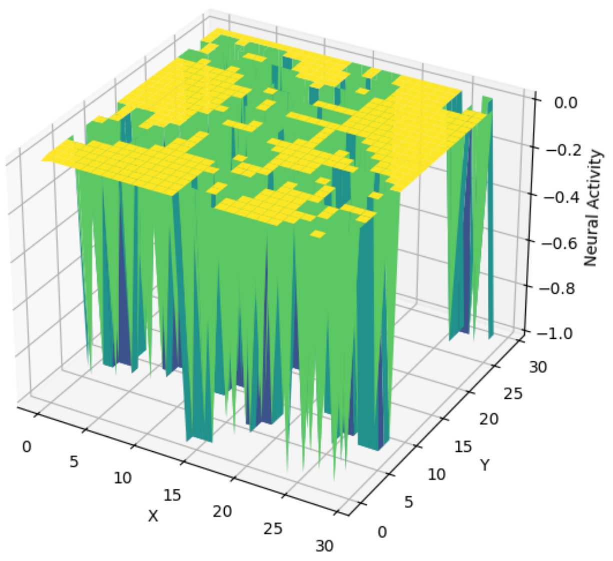


Figure 5.6 The neural activity in completely dynamic environment

5.3.1 Analysis for completely dynamic environment

Obstacles are successfully avoided in the current implementation, but the produced path is not smooth. This is primarily because the algorithm operates in an entirely dynamic context, calculating the path completely online.Although the behavior of the neural activity adheres to the same principles, the path planning requires further optimization. It is required to investigate and incorporate more sophisticated algorithms in order to improve the path's efficiency and smoothness. This report's future scope section explores different approaches to bringing about these enhancements.

Chapter 6

Future Scope

6.1 Introduction

In the results and discussions section of the report, three cases for the simulation is being demonstrated.In the third case, where the environment is completely dynamic, the generated path, while capable of avoiding all obstacles, lacks smoothness and optimality. The algorithm encounters several points where it gets stuck in deadlocks, and the methods it uses to escape from these deadlocks are not always optimal.

To address these issues and enhance the performance of the algorithm, the **Preventive Deadlock Processing Algorithm (PDPA)** and the **Escape Route Generator Algorithm** can be employed. These methodologies can provide more efficient deadlock resolution and ensure smoother, more optimal paths in highly dynamic environments.

6.2 Preventive Deadlock Processing Algorithm

Although the standard GBNN can help the robot escape certain deadlock situations, this method may be slow or may fail to find an exit. This can cause the robot to wander in circles until the neural map updates enough to identify a clear escape route, if it ever does. Depending on the complexity and size of the workspace, the neural approximation might be insufficient to access all non-disinfected areas. To enhance the results of the CCPP strategy, the PDPA has been developed and integrated with the enhanced GBNN algorithm. PDPA outlines the steps designed to anticipate and address deadlock issues. The deadlock estimator is activated at each iteration. However, when the robot moves along a path generated in response to a deadlock estimation, the estimator remains inactive. If the target cell is not reached within the expected number of iterations, the algorithm is re-activated to obtain a new path.

6.3 Escape Route Generator Algorithm

When a deadlock is anticipated, a solution using Dijkstra's algorithm is suggested as an alternative to the neuronal approach. Dijkstra's algorithm calculates the shortest path between two nodes within the same graph, allowing the robot to quickly escape from a deadlock situation.

To use Dijkstra's algorithm during initialization, a graph is created from the map, with nodes representing each non-obstacle cell. Edges are established between each cell and its neighboring cells, all with a constant weight. When a deadlock situation is detected, the ERGA searches for the shortest path between the robot's current position and the remaining non-visited cells.
In summary, the ERGA first calculates the Euclidean distance between the current position and the other candidate cells, ignoring obstacles. These candidates are then sorted by their distance from the current position, from farthest to closest.

Algorithm 10 Escape Route Generator Algorithm (ERGA)

```
**Search for non-visited cells** $N(m,n)$
    $$N_{f=0}(m,n), \forall m \neq k_c, n \neq l_c, 1 \leq m \leq N_x, 1 \leq n \leq N_y, \text{ such that } f(m,n) = 0$$
**All possible escape positions are analysed** $(k_{\text{esc}}, l_{\text{esc}}) \in N_{f=0}$ **Calculate Euclidean distance for each possible escape cell**
    $$d_{\text{esc}}(\text{index}, p_{\text{esc}}) = d_E(p_c, p_{\text{esc}}) = \sqrt{(k_c - k_{\text{esc}})^2 + (l_c - l_{\text{esc}})^2} \forall 1 \leq \text{index} \leq (N_{f=0x} + N_{f=0y})$$
**Ascending order the distances**
    $$d_{\text{esc}}(\text{index}) = d_{\text{esc}}(\text{index}, p_{\text{esc}}) \forall 1 \leq \text{index} \leq (N_{f=0x}+N_{f=0y}) \text{ such that } d_{\text{esc}}(\text{index}, p_{\text{esc}}) \leq d_{\text{esc}}(\text{index}+1, p_{\text{esc}})$$
**Initialize the minimum distance** $d_{\min}$
    $$\text{set } d_{\min} := \infty$$
    $$\text{set } d_{\text{loop}} := \text{true}$$
**All possible ascending ordered escape positions are analysed** $p_{\text{esc}} \in d_{\text{esc}}$ such that $d_{\text{loop}} == \text{true}$ **Calculate distance with Dijkstra's algorithm**
    $$[d_{\text{Dijkstra}}, p_{\text{Dijkstra}}] := \text{Dijkstra}(p_c, p_{\text{esc}})$$
**Check if** $d_{\text{Dijkstra}}$ **is less than** $d_{\min}$ $d_{\text{Dijkstra}} < d_{\min}$ **Update** $d_{\min}$
    $$\text{set } d_{\min} := d_{\text{Dijkstra}}$$
    $$\text{set } p_{\min} := p_{\text{Dijkstra}}$$
**Check if** $d_{\text{Dijkstra}}$ **is the same as** $d_{\text{esc}}$ $d_{\text{Dijkstra}} == d_{\text{esc}}$ set $d_{\text{loop}} := \text{false}$ **Neural behaviour is updated**
    $$\text{set } x(m,n) := 1 \forall (m,n) \in p_{\min}$$
    $$\text{set } I(m,n) := 100 \forall (m,n) \in p_{\min}$$
```

6.4 Real World Underwater Environment Simulation

The current path planning approach has been implemented and tested through simulation using Python, where both the model and vehicle dynamics have not been considered. The Glasius Bio-Inspired Neural Network (GBNN) has been utilized for complete coverage path planning, incorporating

Theta* and the Boustrophedon algorithm. To advance this research, creating a real-world underwater environment simulation using Underwater Simultaneous Localization and Mapping (SLAM) integrated with Gazebo and Robot Operating System (ROS) is essential.

6.4.1 Underwater SLAM

Underwater SLAM is crucial for autonomous underwater vehicles (AUVs) to navigate and map their environment in real-time. Utilizing sonar, inertial measurement units (IMUs), and other underwater sensors, SLAM algorithms help the AUV create a detailed map of its surroundings while simultaneously tracking its position. This real-time mapping and localization are vital for obstacle avoidance and accurate path following.

6.4.2 Gazebo and ROS Integration

Gazebo is a robust simulation tool that provides realistic rendering of complex environments and dynamic interactions. Integrating Gazebo with ROS enables the simulation of underwater environments with high fidelity. ROS offers a flexible framework for writing robot software, allowing seamless communication between various AUV components, such as sensors, actuators, and control algorithms.Using Gazebo and ROS together will facilitate creating a comprehensive underwater simulation environment where the AUV can be tested under various conditions, such as different water currents, varying visibility, and dynamic obstacles. This integration will validate the effectiveness of the path planning algorithms and the robustness of the AUV's control systems.

6.5 Conclusion

Incorporating Underwater SLAM and Gazebo ROS into the path planning project represents a significant step towards creating a realistic simulation environment. This will not only enhance the current research but also pave the way for future developments in autonomous underwater exploration and navigation. By simulating real-world conditions, we can ensure that the AUV is well-prepared to operate efficiently and safely in actual underwater environments.

Bibliography

[1] Complete Coverage Path Planning of Autonomous Underwater Vehicle Based on GBNN Algorithm Daqi Zhu Chen Tian ,Bing Sun, Chaomin Luo *Published online: 10 February 2018, Springer*

[2] Glasius bio-inspired neural networks based UV-C disinfection path planning improved by preventive deadlock processing algorithm Daniel Vicente Rodrigo , J. Enrique Sierra-García , Matilde Santos

[3] Toward energy-efficient online Complete Coverage Path Planning of a ship hull maintenance robot based on Glasius Bio-inspired Neural Network M.A. Viraj J. Muthugala , S.M. Bhagya P ,Samarakoon, Mohan Rajesh Elara *Engineering Product Development Pillar, Singapore University of Technology and Design, Singapore 487372, Singapore*

[4] Complete Coverage Autonomous Underwater Vehicles Path Planning Based on Glasius Bio-Inspired Neural Network Algorithm for Discrete and Centralized Programming Bing Sun, Daqi Zhu , Chen Tian, and Chaomin Luo, Member, IEEE, *IEEE Transactions on Cognitive and Developmental Systems, VOL. 11, NO. 1, March 2019*

[5] Ruled Path Planning Framework for Safe and Dynamic Navigation,Alexandre Cardaillac,Martin Ludvigsen*Faculty of Engineering, Department of Marine Technology Norwegian University of Science and Technology (NTNU) Trondheim, Norway*

[6] Sampling-based Algorithms for Optimal Motion Planning,Sertac Karaman, Emilio Frazzoli,*Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA.*

[7] Rapidly-exploring Random Graphs: Motion Planning of Multiple Mobile Robots,Rahul Kala*School of Cybernetics, School of Systems Engineering, University of Reading, Whiteknights, Reading, UK, RG6 6AY*

[8] Rapidly Exploring Random Trees : A New Tool for Path Planning, Steven M. Lavalle, *Department of Computer Science,Lowa State University*

[9] Path Planning Technologies for Autonomous Underwater Vehicles-A Review,DAOLIANG LI,PENG WANG,LING DU,*College of Information and Electrical Engineering, China Agricultural University, Beijing, 100083, China*

[10] Current Study on A* Algorithm in Autonomous Obstacle Avoidance,Chengyue Zhu,*Shanghai International High School of BANZ, Shanghai, China*

[11] Ahmed, M., Eich, M., & Bernhard, F. (2015). Design and control of MIRA: A lightweight climbing robot for ship inspection. In International letters of chemistry, physics and astronomy: 55, ILCPA

Vol. 55 (pp. 128–135). SciPress Ltd, http://dx.doi.org/10.18052/www.scipress.com/ILCPA.55.128

[12] Likhachev, M., and Koenig, S. 2001. Lifelong Planning A* and Dynamic A* Lite: The proofs. Technical report, College of Computing, Georgia Institute of Technology, Atlanta (Georgia).

[13] Ramalingam, G., and Reps, T. 1996. An incremental algorithm for a generalization of the shortest-path problem. Journal of Algorithms 21:267–

[14] J. Manley, "Autonomous underwater vehicles for ocean exploration," in Oceans 2003. Celebrating the Past ... Teaming Toward the Future (IEEE Cat. No.03CH37492), vol. 1, 2003, pp. 327–331 Vol.1.

[15] ElHalawany, B.M.; Abdel-Kader, H.M.; TagEldeen, A.; Elsayed, A.E.; Nossair, Z.B. Modified A* algorithm for safer mobile robot navigation. In Proceedings of the 2013 5th International Conference on Modelling, Identification and Control (ICMIC), Cairo, Egypt, 31 August 2013–2 September 2013; pp. 74–78.

[16] Pal, A.; Tiwari, R.; Shukla, A. Modified A* algorithm for mobile robot path planning. Soft Comput. Tech. Vis. Sci. 2012, 395, 183–193.

[17] A., Malaterre, L., Tazir, M., Trassoudaine, L., & Checchin, P. (2016). Detecting and analyzing corrosion spots on the hull of large marine vessels using colored 3d lidar point clouds. ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences, 3(3). Akyuz, E.,& Celik, E. (2018).

[18] A survey on pneumatic wall-climbing robots for inspection. In 2016 24th mediterranean conference on control and automation (MED) (pp. 220–225). IEEE. Caldwell, R., et al. (2017).

[19] Complete path planning for a tetris-inspired self reconfigurable robot by the genetic algorithm of the traveling salesman problem. Electronics, 7(12), 344. Le, A. V., Kyaw, P. T., Veerajagadheswar, P., Muthugala, M. A. V. J., Elara, M. R., Kumar, M., et al. (2020).

[20] Complete area coverage path-planning with arbitrary shape obstacles. Journal of Automation and Control Engineering Vol, 7(2). Karapetyan, N., Benson, K., McKinney, C., Taslakian, P., & Rekleitis, I. (2017).

[21] Design and performance analysis of a tracked wall-climbing robot for ship inspection in shipbuilding. Ocean Engineering, 131, 224–230. Ivošević, Š., & Bauk, S. (2018). The use of information technology in the assessment of the corrosion damage on ship hull. In 2018 23rd international scientific-professional conference on information technology (IT) (pp. 1–4). IEEE. Jan, G. E., Luo, C., Lin, H.-T., & Fung, K. (2019).