NOVEL APPROACH TO 3-DIMENSIONAL PATH-PLANNING USING DISPERSIVE FLIES OPTIMIZATION (DFO) FOR EFFICIENT PERFORMANCE TRADE-OFF AND EXPLORATION-EXPLOITATION BALANCE.

Muhammad Khan - 001057769

First Supervisor : Dr Hooman Oroojeni
Second Supervisor : Dr Mohammad Majid al-Rifaie

Project Report submitted in Partial Fulfillment of Bachelor (BSc) Computer Science

Word Count : 16216 Words

School of Computing and Mathematical Sciences
University of Greenwich

Acknowledgement

I would like to express immeasurable gratitude to the following individuals who have contributed towards the successful completion and implementation of this research project:

My first supervisor, Dr Hooman Oroojeni and second supervisor, Dr Mohammad Majid al-Rifaie for their collective technical expertise, suggestions, mentorship, advice, encouragement and prompt answers whenever I had urgent questions or needed a sense of direction. This work would not have been possible without them. Furthermore, their research and publications where the chosen algorithm, Dispersive Flies Optimization has been utilized successfully contributed indirectly yet significantly towards this project.

Abstract

Autonomous navigation via path-planning is a challenging problem that needs to address the constraints of finding the shortest obstacle free-path from start to goal. This problem complexity is increased when considering 3-Dimensional path-planning where the carrier module such as Unmanned Aerial vehicle (UAV) and Unmanned Underwater vehicle (UUV) must deal with vertical axis of movement and obstacle avoidance where conventional graph-search algorithms suffer from high computation time. Swarm-Intelligence algorithms have demonstrated significant improvement in addressing this problem via reducing the computation time with minimal trade-off in path-optimality. Dispersive Flies Optimization (DFO) is a simplistic convergence-independent swarm-optimizer that has showcased out-performance in notable benchmark functions which has yet been implemented for path-planning experiments, which is a notable research gap. The experiment implementation therefore addresses this research gap via application of the proposed algorithm towards 3-Dimensional environments for finding feasible paths, where distance is measured using Euclidean distance metric and obstacle avoidance is implemented via applying penalty score to fitness function when collision paths are detected. The proposed methodology successfully outperforms RRT and ACO for standard path-planning objectives which are minimization of path-cost, path-risk and computation time whilst outperforming A* algorithm in taking significantly shorter computation-time in complex environments.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Global Path-Planning

Path-Planning is defined as the process of planning feasible un-occupied paths for navigation of carrier module from one point to another such that the path is collision-free where the main performance indicator is path-length and path-safety. Path-planning algorithms are therefore computational algorithms that are responsible for computing the least-cost path in addition to ensuring safety, efficiency and collision avoidance from one point to another and thorough the journey where the navigation process is broken down to discrete and separate movements to satisfy local environment constraints. (Aggarwal & Kumar 2020).

Path planning algorithms can be categorized into Global planning algorithms and local planning algorithms whereby in global planning algorithms, the algorithm with the carrier module has complete information about the map and positions of static obstacles in the environment, thus is able to generate a complete path from origin to destination prior to departure which is known as offline planning (Xu & Shum 2019) whereas local planning algorithms navigate dynamically by relying on input data to build a map as the carrier module navigates through the environment (Chen & Zhu 2021). The research shall focus on global (offline) path-planning methods thorough the literature review where the proposed methodology aims to optimize global-path planning in 3-Dimensional Environment

Path-planning is classified as a non-deterministic polynomial-time hard problem ("NP-Hard") in the context of computing a continuous path from origin to destination in a 3D workspace because the difficulties increase exponentially due to kinematic constraints, and the problem complexity is directly proportional to the degrees of freedom. (Chen & Wei 2021), which is an increment when considering 3D environments compared to 2D environments due to higher degrees of freedom of movement.

1.1.1 NP-Hard Problem

P-Hard are Polynomial algorithms which are algorithms that can be verified and solved within polynomial time. NP-Algorithms are Non-Polynomial algorithms where the can be verified but not solved within polynomial time thus having exponential complexity as size (of environment or complexity) increases.

NP-Hard problem is the abbrievation for Non-Deterministic Polynomial-Time Hard which are problems that are at least as difficult as the most difficult problems in the NP class where every problem in NP is reduced to NP-Hard (Gass & Harris 2001) Where a solution exists, it can be verified in polynomial time but it will take a long time to find the optimal solution, where NP-Hard problems take very long to verify and solve. These are among the hardest problems that exist in computer science. The Travelling Salesman Problem is a closely related NP-Hard problem which is also concerned with path-finding in a directed graph.

1.2 Current Use-Uses of UAVs

1.2.1 Risk-Management

UAVs are extremely versatile in performing dangerous missions in situations such as hazardous environments, high altitude surveillance, conflict zones, search and rescue missions and power line inspections and are able to execute such demanding missions quicker and cost-effectively when compared to using ground vehicles and helicopters. Most importantly, UAVs are able to offload risks and hazards faced by humans by performing missions autonomously and requiring minimal input and cost to operate (Ruiz Estrada & Ndoma 2019).

1.2.2 Emergency Response

UAVs proven to be an invaluable resource for emergency response services, particularly during the 2021 wildfires which devastated parts of North America (Uchiyama & Atkins 2021). These UAVs were instrumental in fast detection, localization and communication with firefighters which allowed quick emergency response to limit the propagation of the flames. UAV, due to its autonomous flight capability and when equipped with camera, can provide real-time visual information to aid emergency services.

1.2.3 Communication

UAVs have also garnered much in interests in implementation in Vehicle Ad-Hoc Network(VANET) and Flying Ad-hoc Network (FANET) in the field of communication. VANET is a decentralized vehicle to vehicle communication protocol which has been widely implemented for intelligent locomotion and entertainment systems for smart-city based projects (Raza et al. 2021). Unmanned aerial vehicles have been widely used as carrier modules to provide communication coverage to a specified coverage region due to their low-cost deployment, successes in load-balancing, increased throughput and better packet delivery ratio (Khan et al. 2017).

1.2.4 Patrolling

The UK Border Force have conducted numerous pilot schemes and trials for using UAVs for the surveillance of the English Channel for detection of illegal fishing vessels and boats carrying victims of human-trafficking whereas United States of America have been using drones since early 1970s to patrol the USA-Mexico border (Koslowski 2021) where presently, UAVs equipped with electro-optical, infrared cameras are quickly replacing manned drone systems and most recently, Wide-Intelligence Surveillance System (WAPS) over a wide area of a region.

1.2.5 Agriculture

UAVs are currently used in the agriculture sector for surveying and mapping over the farmland region, monitoring the health of crops and detecting disease using computer-vision technology and spraying of water and pesticide over crops (Yinka-Banjo & Ajayi 2020). It is predicted that the agriculture drone market is worth USD 32.4 billion.

1.2.6 Last-Mile Delivery

Last-mile delivery is defined as the final delivery stage where a parcel is transported from the logistics hub or warehouse to the final destination, which is the location of the customer. Albeit the seemingly straightforward process, this delivery stage is the most costly part of the supply-chain which accounts for 53 percent of the retailer shipping cost due to economic factors such as additional fuel requirement, driver wages, failed deliveries and refund costs (Remer & Malikopoulos 2019)

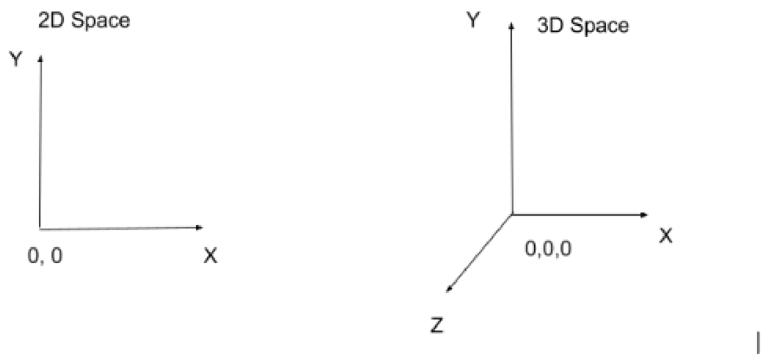
Various industries have proposed the use of Drones and UAVs as a cost-cutting measure for the last-mile delivery problem, where corporations such as AMAZON, DHL and FedEx have been major investors and adapters of this market. Amazon patented a ”beehive-like” structure for multiple UAV take-off and landing in urban areas to support its Prime-Air last mile delivery system (Jung & Kim 2017) via UAVs for rapid 30-minute delivery to customers from the warehouse. However, Prime Air failed to meet projections to be fully operable by 2021 due to lack of UAV design plans received and lay-offs due to the COVID-19 pandemic, thereby still in the testing phase (). Zipline is an American drone delivery company that has expanded its operations both within the USA and in East African countries of Rwanda and Ghana (Nisingizwe et al. 2022). The company has been successful where UAVs have been utilized for delivery of blood, platelets, plasma and vaccines via cold storage which are transported using the UAVs.

It is noted that the last-mile delivery process will involve low-altitude flight and navigation where the environment is dense with static and moving obstacles where there is lack of industry-wide implementation as most present-day UAV missions such as those conducted for surveillance, inspection and espionage is conducted from high altitude where the path-planning process does not involve obstacle detection and avoidance whereby this constraint is one of the major obstacle involved against mass market adoption of UAVs (Wang et al. 2022).

1.3 Research Motivation

2-Dimensional path-planning is concerned with the movement of x and y directions where the traversal typically occurs forwards and backwards in the x-axis and left and right in the y-axis (Yang et al. 2016), this is significantly more complex when dealing with 3-Dimensional UAV path-planning where continuous movement is necessary to model continuous UAV flight. The complexity of 3-Dimensional path-planning and navigation is an exponential increment when compared to 2-Dimensional path-planning, where the latter is obsolete and thereby cannot qualify for planning in complex environments (Song et al. 2019).

Figure 1: 2D and 3D Environments



Whilst path length is indeed an important factor to assess the performance of Algorithms, where exact-methods such as Dijkstra and A* algorithm have contributed significantly towards optimal-length path-planning, however these methods have the shortcoming of facing significant computational overload and thus taking long computation time to plan a feasible path (Ashish et al. 2021) whereas sampling based methods, albeit improving the performance in terms of computation time, still face significant computational overload, thereby converges slowly when planning in complex and cluttered environments, additionally the paths produced by sampling-based methods are suboptimal (Al-Ansarry & Al-Darraji 2021). Traditional meta-heuristic and swarm-intelligence methods such as Genetic Algorithm, Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) have been proposed for path-planning in many established research publications however these methods in their basic forms still suffer from premature-convergence, long computation time and complex hyperparameter settings required to optimally tune each to suit various environments (Karur et al. 2021).

This shortcoming makes these methods unsuitable for real-time path-planning in complex and cluttered environments, thus require expensive hardware requirements to compensate for the high computational cost associated, which is not-economic.

The recent publications as reviewed in literature findings have suggested the success of new-generation of swarm-intelligence based methodologies in optimally balancing between computing feasible path and within feasible computation time. By effectively performing optimal trade-off such that minimal sacrifice is made on the part of path optimality whilst the computation time is maximally reduced, significant contribution can be made towards addressing the research gaps that exist where there are only few recent works that have addresses the multi-objective optimization of both path-length, path-safety and computation time for 3-Dimensional path-planning.

This hypothesis and multi-objective optimization problem is further investigated and addressed by utilizing a powerful yet simple swarm-intelligence algorithm known as Dispersive Flies Optimization, DFO (Al-Rifaie 2014*b*) which is one of new generation of powerful and minimal hyperparameter swarm-intelligence algorithms which has showcased its outperformance on various benchmark evaluations and optimization problems.

1.4 Research Objectives

1.4.1 Objective 1 : Proof-of-Concept

To successfully implement DFO as a first-time approach towards path-planning using the minimalist swarm optimizer such that its outperformance on benchmark evaluations is reproducible and applicable to the domain of path-planning.

1.4.2 Objective 1 : Minimization of Path-Cost

Path-Cost is represented by the distance from the start coordinates to the goal coordinates in the x, y and z plane. The fitness function must take into consideration that the distance traversed is not a one-dimensional trajectory from start to goal and hence, must take into consideration the average path traversed in both linear distance (x and y-axis) and the altitude changes (z-axis) which also incur additional length to the overall path. Therefore, the goal of path-cost is to obtain the shortest path across all dimensions.

1.4.3 Objective 2 : Ensuring Path-Safety

Path-safety is represented by obtaining collision-free paths from start to goal coordinates such that the probability of collision is minimized to zero. Thus, this objective is concerned with minimization of collision risk.

1.4.4 Objective 3 : Minimization of Computation Time

The proposed methodology must compute the optimal paths within feasible computation time for all environments, such that it is kept minimum.

1.4.5 Objective 4 : Exploration and Exploitation Balance

Exploration is defined as the ability of the algorithm to search for globally optimal solutions in the global search space far from the current solution position, whereas Exploitation is defined as the ability of the algorithm to locally search the surrounding local regions nearby the position of the current solution position (Hussain et al. 2019). DFO, through the mechanism of disturbance, possesses the ability to randomly restart the position of a percentage of agents in the search space to search for better solutions, thereby escaping local minima trap whilst ensuring fast convergence.

1.4.6 Objective 5 : Simulation

To simulate and visualize the collision-free navigation path produced by the proposed and baseline methodologies, such that visual observations can be made regarding path smoothness and proximity to obstacles.

2 Literature Review

The review of past work, both recent and former approaches, are broken down to the respective categories based on the planning techniques employed by the respective algorithms. The categories of algorithms discussed shall cover the contemporary approaches such as exact-methods, sampling-based methods, evolutionary algorithms and swarm-intelligence based algorithms

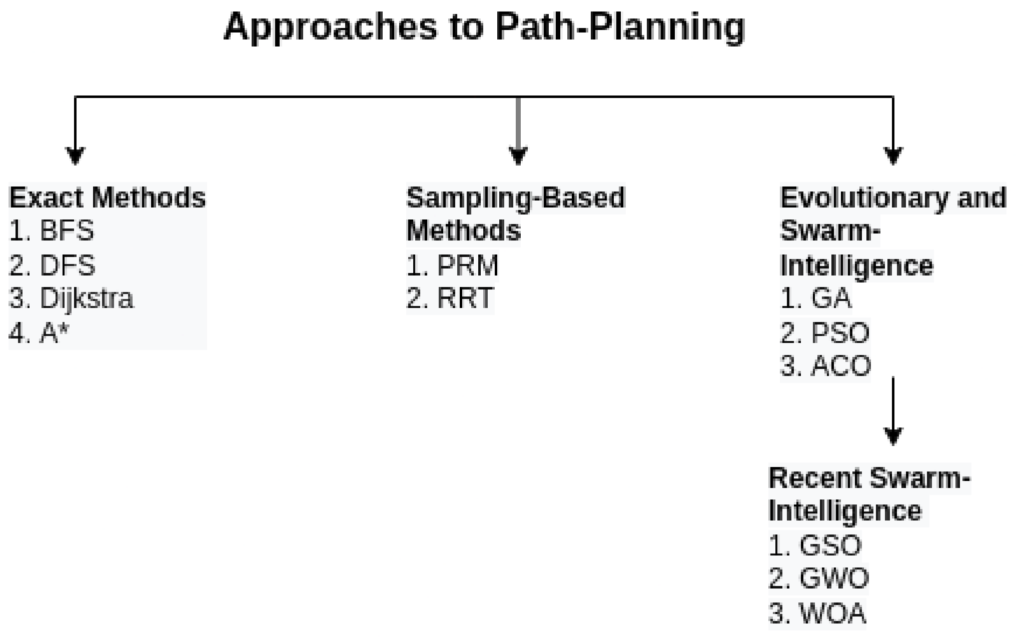


Figure 2: Approaches to Path-Planning

2.1 Taxonomy of global autonomous path-planning

Autonomy is defined as the ability of intelligent systems such as robots to make decisions and carry out tasks without human input whereas autonomous navigation is the ability of the robot to firstly localize its relative position in the search space and, secondly to navigate from one point to another whilst taking into account external constraints such as terrain and presence of obstacles. Autonomous navigation is achieved via a series of discrete steps that are computed according to environment constraints via use of search algorithms which can differentiate between occupied and unoccupied spaces to navigate from one point to another, thus generating the path (Pisarov & Mester 2020).

2.2 Classic Planning Methods

2.2.1 Breadth-First Search and Depth First Search

Breadth-First Search, BFS was invented by Konrad Zuse in 1945 (Giloi 1997) and subsequently reinvented in 1959 by Edward F Moore (Moore 1959) where it was applied to graph search and by (Lee 1961) who applied BFS towards finding path connections for wire-routing in printed circuit board (PCB) design. BFS is an uninformed search algorithm which does not have information about the distance the robot has travelled from the start position nor how far the robot is from the end position as it was originally used to search a tree data structure, the starting point is the root of the search tree and thereby, the mechanism of search is employed by exploring all nodes at the current depth and subsequently moving on to the next depth. To keep track of visited but unexplored child nodes, a queue data structure is used.

BFS iterates over all possible paths (edges) until the goal node is reached and henceforth, runs in linear time which is represented by the big O notation of 0(n) and algorithmic notation of O(V+E) which implies that the computation time increases linearly as the number of vertices (nodes) and edges (path) increases.

Depth-First Search, DFS was first conceptualized by Charles Pierre Tremaux in the 19th century for maze-solving problem. DFS is an uninformed search algorithm, similar to BFS which does not have information about its relative position nor distance from start and end points. Furthermore, similar to BFS, DFS is used to search a tree data structure from the root node and differs from BFS whereby the mechanism of search is by exploring the tree as deep as possible i.e. exploring a node and its descendants until reaching the deepest node which has no children nodes whereby if the solution is not found, DFS will perform back-tracking which is reversing its course back to the root node and repeating the same depth-traversal on the neighbouring node on the right and this process will repeat consecutively until the solution is found. The explored nodes in DFS are stored in a stack data structure which follows the first-in last-out methodology.

DFS does not consider the number of nodes prior to exploring the nodes, thus functions as a blind-search algorithm, similar to BFS and thereby its complexity increases linearly with the number of nodes and edges(size of graph), thus operating in linear time with a worst-case time complexity denoted by O(n). This denotes that DFS, similar to BFS may produce optimal solution for a tree with a finite number of nodes and vertices however with overly large or infinite trees, DFS can get stuck in an infinite loop without being able to return a solution.

Notable implementations of BFS and DFS used in autonomous path-planning include works by (Randria et al. 2007) whereby BFS and DFS were used as pathfinding algorithms to navigate 2-dimensional indoor maze environments by using maze-images captured by on-board cameras as input to compute the environment whereby the number of checks and moves are used to assess the impact of environment complexity on BFS and DFS algorithms respectively. The experimental results showcased that the two algorithms were able to navigate a simple environment with a single obstacle within feasible and with low computational time of 0.06 and 0.08 seconds respectively, however as the number of obstacles and environment size increased, the number of moves and checks increased by a large margin. For the largest environment with cluttered obstacles, the path generated was still feasible however the computation time grew exponentially where it took 401.428 and 347.800 seconds respectively. In the experiments conducted, it was shown that Genetic Algorithm, albeit having the longest computation time in the simplest environment, showcased the shortest computation time of 17.716 seconds and 22.042 seconds in medium and complex environments respectively. This observation further corroborate the shortcomings of DFS and BFS in large complex environments whereby the computation time increases linearly with the size and complexity of environment, albeit in the end all compared metrics are able to plan equally optimal paths upon completion of path computation. The performance of genetic algorithm provides more validity about the potential of meta-heuristics in computing paths in shorter computation time when size and complexity of problem increases

There are significant research gaps that exist where BFS and DFS have only been applied to limited 2-Dimensional path-planning experiments which are not recent, thus suggesting that these methodologies are obsolete for present-day UAV path-planning use case.

2.2.2 Dijkstra's Algorithm

Dijkstra's algorithm (Dijkstra 1959) is a graph-based, uninformed path-planning algorithm that is able to reliably compute an optimal shortest-distance path from the start node to the goal node which can be all vertices in a graph and therefore was a candidate solution for solving the single-source shortest path problem.

Initially, in its original form, the algorithm was tasked with finding the shortest path between two nodes however the dominant and most common variant of Dijkstra is characterized by a fixed start node and is tasked with computing the shortest path, which are represented as edges, from the start node to all other nodes present in the graph.

Dijkstra's algorithm is a Breadth-First search algorithm whereby it is an uninformed algorithm that performs blind search in a graph for the solution but presents a major modification whereby the node with the lowest cost is expanded as opposed to the node with the lowest depth as characterized by BFS, thus is able to compute more optimal paths with the same time complexity as BFS. This implies that the vertices are processed from lowest to highest distance from the start node with the goal of selecting the shortest distance between the vertices which is called link-distance. Below equation represents the length of the path which is denoted as the sum of weights of its edges.

$$p = \sum_{1}^{k} w\left(v_{i-1}, v_i\right) \tag{1}$$

Where p is the length between previous and current vertex, w is the weight function, vi-1 is the previous vertex and vi is current vertex. Therefore, this equation can be represented as the cost of moving from start to current node as shown below.

$$f(n) = g(n) \tag{2}$$

where g(n) is the total accumulated cost from start to current position.

At every iteration (loop), the node with minimum distance from the root node is chosen.

The data-structure employed to store and query visited nodes during back-tracking is a minimum priority queue whereby the vertices with the shortest distance between one another (link-distance) are prioritized and removed at every loop and if the node chosen is the goal node, then it will be returned, thus Dijkstra has a worst-case time complexity of O(nlogn) or O(E + V)logV which is an improvement compared to O(n) demonstrated by BFS and DFS due to the clever nature of the priority queue data structure that pops elements according to associated priority rather than based on the first in, first out (FIFO) method thus having a lower computational load when the number of vertices increase.

Notable research and application of Dijkstra's algorithm applied to robot path-planning include publication by (Luo et al. 2020) whereby a 2 dimensional grid-based map was modelled according to three scenarios of increasing size and number of obstacles which are similar to the experimental settings used for aforementioned DFS and BFS. The author performed preliminary experiments where Dijkstra was found to under-perform in environments with curved surfaces whereby when the curvature increases, the deviation from optimal length increases, thus a modified Extended Dijkstra's Algorithm using a Delaunay triangulation was proposed to address the constraint to computing feasible paths in environments with curved surfaces where the path-lengths showcased slight improvement over traditional Dijkstra's algorithm.

Variants of Dijkstra's algorithm include multi-objective Dijkstra proposed by (Fink et al. 2019) that uses global map information from terrain data for optimal traversal in 3-Dimensional configuration space which was employed in the Global Rover Horizontal Optimization Planner (GRTOP) system for Mars terrain exploration. Dijkstra's algorithm was extended to consider multiple objectives by setting the weight between neighbouring nodes to a linear combination of multiple weights corresponding to a goal whereby the optimal path is found according to combination metrics that include 3-Dimensional Euclidean distance between start to end nodes, smoothness of path and minimization of altitude change which are all pivotal for safe and efficient traversal. Simulation experiments showcased the robustness of Dijkstra's algorithm as a global planner and its applicability to 3-dimensional path-planning to find globally-optimum paths when the environment map is known. The multiple objectives in addition to path length such as path-smoothness and minimization of altitudes are representations of real-life constraints which still exist as a major research gap in past and present path-planning research papers.

2.2.3 A* Algorithm and its Variants

A* algorithm (Hart et al. 1968) unlike its graph-traversal predecessors, is an informed and intelligent search algorithm which considers position and distance of nodes relative to start and goal positions which implies that each node will be assigned a weight corresponding to its relative distance to the goal from the start node. Therefore, A* stores a tree of paths starting from the root and extending each path from one node to another based on the connection between nodes that result in the lowest cost. At each iteration, the algorithm determines the path to be extended by analyzing the current path cost and the estimated cost to extend the path until the goal node which is represented in equation below.

$$f(n) = g(n) + h(n) \tag{3}$$

Where n is the next node on the path, f(n) is the total path cost, g(n) is the cost(length) of the path from start node to n and h(n) is a heuristic function which is responsible for estimating the cheapest path from n to goal node. This heuristic function has the property of being admissible which is defined as a property whereby the heuristic function never overestimates the cost of computing the path to the goal and always returns the lowest i.e. optimal cost path from the current node to the path.

The data structure employed to store and select the minimum cost nodes for expansion is similar to Dijkstra, which uses a priority queue which is referred to as the "open set". At each iteration, the nodes with the lowest cost prioritized for removal until the goal node is removed which is the termination criterion for A* algorithm. Furthermore, the intelligent nature of the algorithm allows it accurately produce the correct sequence of steps corresponding to the computed path length by ensuring each node in the path keeps track of the node preceding it, therefore upon producing the path cost, the goal node will point to its predecessors and this repeats for each node until start node is reached. A* algorithm is noted to be faster than Dijkstra whilst being able to reliably compute optimal cost paths owing to its heuristic function when it is assumed that the heuristic function is consistent. The advantages of A* algorithm are as follows.

1. Admissible : If a solution exists, the first solution thereby found by A* is optimal.

2. Complete : If a solution exists, it can be found in finite time.

3. Optimal : Owing to its heuristic function, the solutions computed by A* are optimal in terms of path cost.

However, in spite of its robust capabbility in producing optimal paths, the algorithm only maintains its property of being complete if the branching factor is finite, thus in the event of infinite branching, A* will not have any termination condition. Since every action has a fixed cost, large and complex environment will result in longer computation time for A* algorithm due to its worst-case time complexity of O(logn). Lastly, it is also noted that the performance of A* is heavily reliant on the accuracy of its heuristic function that computes h(n).

Notable implementations of A* algorithm applied to UAV path-planning include works by (Liu et al. 2020) where various approaches such as A*, PRM, RRT and VFH are applied and their performance compared according to multi-objective success metrics that include minimizing path length, producing collision free paths and minimizing computation time. The experiments are carried out in five 2-dimensional environments with increasing complexity and concentration of obstacles that aims to compare the feasibility of graph-based algorithms. Simulation results for environments with 10 obstacles with two different instances showed path length of 165 with CPU time of 0.093 and 0.063 seconds respectively which is relatively shorter compared to sampling based methods like RRT. For environments with 60 obstacles and two instances, the path length obtained was 165 and 173 respectively which took 0.054 and 0.051 seconds respectively which was shorter in both distance and computation time compared to PRM, RRT and VFH respectively. In highly complex environments with 80 and 120 obstacles of varying sizes and 2 instances each, it is observed that the path length is 165 and 167, and 367 and 565 respectively which took 0.065, 0.056, 0.343 and 0.390 seconds respectively which showcased, on average, shorter path lengths than PRM, RRT and VFH and faster computation time compared to PRM and VFH albeit taking longer to compute compared to random sampling based method like RRT. The paper explained and showcased the optimal performance of A* algorithm in computing shorter path lengths compared to local and other graph-based methods when dealing with static environments. However, it is noted that A* algorithm, in its base form is not suitable for navigation and obstacle avoidance in dynamic environments due to its global planning nature whereby it is an informed search algorithm that requires global map information.

(Duchoň et al. 2014) have implemented A* algorithm for grid-based 2-Dimensional path-planning in obstacle filled environments. It is reported that the paths computed by A* are indeed optimal and that A* always returns the shortest path if one exists, however for large sized environments where A* was simulated in a map containing 60 000 cells, the computation time took over one hour which makes it very infeasible for real-time planning. This finding provides the hypothesis that A*, albeit its capability in generating optimal paths, suffers the same problems of computational overload in increasing environmental complexity as Dijkstra, BFS and DFS albeit relatively improved.

Variants of A* algorithm such as D*, D* lite and Anytime A* family of algorithms are introduced to address the problem of long computation time when dealing with complex environments by allowing trade-offs between path optimality for better computation time by inflating the heuristic function.

D* Algorithm (Stentz 1994) is a variant and modification of A* algorithm for application in dynamically changing environments where the environment information is partially known and robot is reliant upon sensors for detection of unknown obstacles and thereby change in path cost during navigation state. The data structure used by D* is similar to A* and Dijkstra whereby it maintains a list of nodes to be evaluated as follows.

NEW : Never before placed in the OPEN list OPEN : Currently placed in OPEN list CLOSED : No longer in OPEN list RAISE : The cost of the node is higher than previously in OPEN list LOWER : The cost of the node is lower than previously in OPEN list

The mechanism of action of the algorithm starts with expansion phase whereby D* commences search in the backward direction i.e. from the goal node. The algorithm will iteratively select nodes from the OPEN node list for evaluation and propagates the current node's change to all its neighbouring nodes and place them in the OPEN list. Similar to A*, each expanded node has a back-pointer with reference to next node leading to the end with its corresponding cost. The termination condition is whereby the next expanded node is the start node.
When changes in environment are detected by on-board sensors such as obstacles in the navigation path, the affected nodes are re-placed in the OPEN list which is marked to RAISE and its neighbours are checked as well for lower cost options and if not found, the RAISE state is propagated to all the nodes that have back-pointer connections to it for evaluation and when a node in the RAISED state can have its cost lowered, the back-pointers are concurrently updated and thereby, the LOWER state is propagated to its neighbouring nodes. By this mechanism of RAISE and LOWER state lists, the path can be updated accordingly according to changing node costs when changes in environment are detected.

Simulation results have showcased favourable computation time when compared with A* algorithm when dealing with complex environments albeit trade-off is observed in path length. Secondly, D* algorithm also is successful in conducting real-time navigation in dynamically changing environment where obstacle avoidance and edge changes are addressed by use of RAISE and LOWER lists.

Anytime Dynamic A* (Likhachev et al. 2005) is a well known and studied extension of this method which is applicable to complex and dynamic environments for path-planning under feasible computational time. The author expressed the motivation for a dynamic and flexible planning and re-planning mechanism to address the uncertain and dynamic constraints of real-world and real-time use-cases. The no free-lunch theorem is accurately applicable here whereby optimal solutions such as A* and Dijkstra are able to compute optimal paths however at the cost of extremely long computation time. ADA* makes a trade-off by being able to produce suboptimal paths within available computation time. ADA* conducts a series of searches with decreasing inflation factors in each iteration whereby initially the inflation factor, e is initially set to a high value to generate an initial suboptimal path within short computation time and this inflation factor is decayed at each iteration to produce incrementally improved solutions (paths) until an optimal path is found. When a change in environment is detected, such as introduction of new obstacles or change in edge costs, the present suboptimal solution will be invalidated and the inflation factor will be increased again to re-produce a new suboptimal solution and repeat the same process of decaying the inflation to incrementally re-produce a new path. Experiments are conducted in 2-dimensional simulation environment to model the behaviour of a robotic arm-manipulator with three degrees of freedom. Simulation results in dynamic environments of size 50x50 with varying probabilities of obstacles appearing in the path of the robot found that the author proposed ADA* generated better trajectories compared to its predecessors such as D* lite and ARA* whilst processing fewer states thus significantly reducing computation time concurrently with incrementally producing better paths in each iteration compared to D* and ARA*. The decay factor used is an excellent yet simple strategy at dynamically optimizing the heuristic function of A* algorithm which can be implemented as a dynamic mechanism to manipulate the delta values at each iteration.

Recent variations of A* algorithm include Time Efficient A* algorithm (Guruji et al. 2016) which showcased 95 percent reduction in computation time compared to conventional A* algorithm by only calculating the value of heuristic function before the collision check phase instead of fetching and calculating the heuristic function for all nodes, thus reducing the number of nodes to be processed.

2.2.4 Sampling-Based Algorithms - PRM and RRT with their variants

Probabilistic Roadmap Method (Kavraki et al. 1996) is a step forward for motion planning whereby the algorithm also takes into consideration the problem of collision avoidance. The mechanism of action of PRM is by acquiring random samples from the configuration space of the robot i.e. the environment map and testing the environment for presence of free spaces. These configurations are then connected to each other by using an integrated local-planner until this reaches the start and goal configurations, upon which a path is determined between the start and goal configurations by using a graph-search algorithm. The mechanism of action of PRM consists of two phases as follows :

Phase 1 : Construction Phase
This is the graph construction phase whereby random configuration is applied to the search space and each configuration is connected to neighbour configuration using k nearest neighbour method. This process continues until a dense network of configurations are constructed and connected.

Phase 2 : Query Phase
The start and goal configurations are connected to the overall graph and a graph-based path-planning algorithm, notably and commonly, Dijkstra's algorithm is used to "query" for the shortest connected path between the dense configuration networks.

Rapidly-Exploring Random Trees (RRT) (LaValle et al. 1998) is an improved sampling based algorithm that is introduced to address the expensive computational complexities of exact path-planning and is designed to search non-convex and high-dimensional spaces in an efficient manner when dealing with time-constrained and complex situations. RRT is a widely used algorithm in both academia and industry due to its speed, reliability and direct applicability in non-holonomic and kinodynamic planning as opposed to static-planning employed by heuristics such as A* and Dijkstra. Furthermore, unlike PRM which requires connections between its configurations in free-space between start and end, RRT simply samples the environment or search-space by growing a tree with its root being the start node and extending the sample by attempting connection between nearest tree state which passes through a free space. By nature, RRT is biased towards extension of its tree towards large unsearched regions of the environment. The steps involved in computing a path from start (root of tree) to end (leaf) is as follows.

1. Random point is sampled in the free-space of the environment which is devoid of obstacles and a node is created in the sampled random point. This is represented by equation below.

$$(x_1, y_1) - (x_0, y_0) = (v_1, v_2) = v \tag{4}$$

Where (x1, y1) is the randomly generated point whereas (x0, y0) is the nearest vertex in the search tree whilst v is the non-normalized vector.

2. The closest neighbouring node to the sampled random point is found and a path with feasible distance is calculated between the two points. In the presence of obstacles, the path computing continues to the next iteration to iteratively traverse the obstacle and once the obstacle free path is produced, the random sampled position is inserted as a child node to

the nearest neighbouring node as its parent. To obtain the desired point, equation below shows the process to obtain u as a normalized unit vector by multiplication operation wt=ith deltaq parameter which denotes a step-size.

$$\mathbf{u} = \frac{v}{\|v\|} = \left(\frac{v_1}{\sqrt{v_1^2 + v_2^2}}, \frac{v_2}{\sqrt{v_1^2 + v_2^2}} \right) \tag{5}$$

3. This process continues until random sampled position is within a feasible distance from goal position and once goal is reached, the tree is returned.

4. From the goal node, the tree is traversed backwards until the start node to compute the fully connected path between start and end.

(Lajevardy et al. 2015) performed objective comparison to evaluate and compare the performances of graph and sampling based algorithms where A* algorithm and RRT are compared for 3-Dimensional UAV path-planning which is due to their mass utilization mobile robot path-planning in industry and academia. The main theoretical differences between the 2 well-known approaches are explained according to the author's literature review, most importantly the trade-off between the 2 approaches where A* favours path-optimality and RRT favours efficient computation time. Experiments are conducted in 3-Dimensional grid environments with static obstacles for three different maps of increasing obstacles and hence complexities. It is concluded that RRT performs well in evenly distributed environments whereas A* always guarantees optimal paths albeit taking longer to compute such paths in complex environments.

2.2.5 Summary of Exact and Sampling-based Methods

Table 1: Summary and Comparison of Conventional Methods

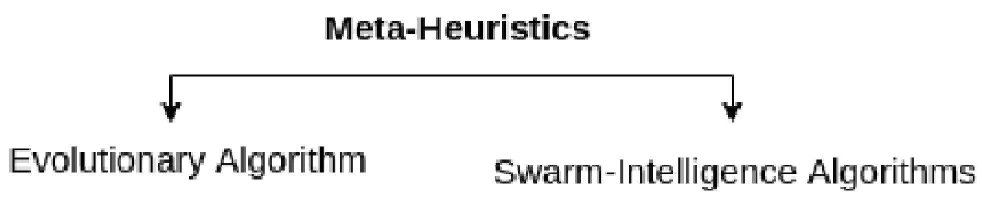
Algorithm	Mechanism	Strength	Weakness
BFS	Exploration of nodes at present depth then next depth No exploitation mechanism.	Able to compute optimal solutions for smaller problems or size of environment	High computational overload for larger problems or environments
DFS	Exploration of depth of tree until deepest node then back-track and explore depth node.	Able to compute optimal solutions for smaller problems or size of environment	High computational time for larger problems or environments Vulnerabiloty to infinite loop
Dijkstra	Same as BFS but lowest cost node expanded.	Improved path optimality within same time complexity of BFS Always returns shortest path	High computational time for larger problems of environments Blind search thus waste time and resources
A*	Informed search, each node assigned a weight depending on distance to goal from start. Forms connection between nodes based on current and estimated cost.	Always retuns the optimal path Better computation time for small and medium sized environments	Unsuitable for navigation in obstacle rich environment if in base form Long computation time
A* Variants	D* uses different lists to store nodes according to optimality heirarchy ADA* inflates heuristic function.	Improved computation speed Applicability to complex and dynamic environments	
PRM	Free space of environment is randomly sampled from start to goal Graph search algorithm used to connect path from start to goal	Suitable for obstacle rich environment Probabilistically complete	Does not always returm optimal paths Does not generate nodes in small gaps
RRT	Extension of tree in free-space where node is created and nearest neighbour node to build path with feasible distance and continue until goal node reached	High computation speed Suitable for high dimension environment Address Non-holonomic and kinodynamic constraint	Paths generated are sub-optimal as state graph is biased towards future expansion

2.3 Meta-Heuristics

Meta-heuristics are defined as a high-level and independent approach in developing heuristic optimization strategies (Du et al. 2016). In contrast with exact methods which are proven to provide an optimal solution within finite time, which in certain cases or with increased complexity might take excessively long to provide a solution, the goal of meta-heuristics are to flexibly compute the suitable trade-off to balance between the quality of the solution and computation time such that a pareto-optimal solution i.e. a sufficiently good solution can be generated efficiently within a relatively shorter computation time. This balancing characteristic of meta-heuristics eliminates the vulnerability of such methods towards exponential and combinatorial explosion (Muazu et al. 2022), which is a phenomenon experienced by most exact methods whereby the computational time increases exponentially or combinatorially respectively when the problem size or domain increases. The literature review section will briefly describe notable works carried out in the category of conventional planning methods and elaborate in depth the meta-heuristics based planning methods due to the meta-heuristics based implementation objective.

The taxonomy of meta-heuristic based methods include evolutionary algorithms where the solutions are improved via generational breeding and reproduction and swarm-intelligence based techniques where a population of agents search the environment to iteratively find the optimal solution.

Figure 3: Meta-Heuristics Categories



All population-based meta-heuristic algorithms have a common feature whereby search and convergence mechanisms are composed of exploration and exploitation phases where during the exploration phase, operators i.e. parameter settings are responsible to control the degree of search space exploration where the search process is stochastic i.e. randomized. Thereafter, exploitation occurs to perform in-depth search of promising areas that are found during the previous exploration phase. Each meta-heuristic has its own mechanism of balancing exploration and exploitation which till-date remains an academic challenge and is an interesting research field.

2.3.1 Genetic Algorithm

Genetic Algorithm (Holland 1984) abbreviated as GA, belong to the category of population-based evolutionary algorithms that exhibit the phenomena of natural-selection whereby the optimal individuals from a population are selected for reproduction to produce genetically superior off-springs that inherit desirable traits from both parents and thus iteratively, with each successive generation producing better solutions than the previous generations. The mechanism of action of Genetic Algorithm is composed of five phases.

Phase 1 : Initial Population
A set of initial individuals (population) is initialized. Each individual is composed of genes and the genes are joined to form the chromosome. Several chromosomes constitute the population. Individuals from the population with desirable characteristics have better fitness scores.

Phase 2 : Selection
Individuals with desired characteristics (according to fitness score) are selected to reproduce with one another

Phase 3 : Modification
Modification operations involve cross-over which is performed to stochastically generate the offspring from an existing population of selected individuals. Mutation (often with low probability) is performed on new offspring to promote diversity within the population and prevent premature convergence.

Termination
Termination occurs once population has converged whereby the subsequent offspring dont differ significantly from previous offspring. Hence the optimal solution(s) has been found.

Notable research and implementation of Genetic Algorithm applied to address the excessive computation-complexity and thus computation time within the scope of path-planning include works by (Xin et al. 2019) where GA is applied to a multi-objective path-planning experiment for a 2-dimensional environment with multiple static obstacles.
Further enhancement is introduced where improved cross-over operator is used to involve parents with both good and bad fitness values to promote increased diversity among the population and thereby improve exploration of the search space.

Further implementations include works by (Sonmez et al. 2015) where Genetic Algorithm is applied to 3-Dimensional path-planning and obstacle avoidance. The paper clearly explained the challenges and concerns of modelling and planning in 3-dimensional environments where there is significant increase in complexity due to the infinite degrees of freedom for the mobile robot as compared to finite degrees of freedom for 2-dimensional navigation. Whilst exact planning algorithms such as Dijkstra and A* will in all cases be able to compute the optimal solution in finite time, however the added complexity due to increase in dimensionality will incur higher computation costs. The implementation of GA was thus, according to the author's justification, most befitting to address this issue. The 3-Dimensional environment was modelled using terrain data and population of 25 chromosomes are used with a crossover and mutation rate of 80 percent and 20 percent respectively. Elitism is introduced with a rate of 20 percent to speed up convergence where the "elite" individuals from the present generation are carried over to the next generation without any changes to ensure the optimal solution attained in previous generations do not degrade in subsequent generations.

Genetic algorithm has been used in many path-planning experiments, whereby the general consensus is that the performance of genetic algorithm is heavily dependant on its operators which include crossover and mutation rates along with selection strategy. According to (Lamini et al. 2018) many cross-over operators in literature are prone to generating infeasible paths, therefore have suggested modified cross-over operations as a measure to increase path feasibility via arbitrary selection of mutual nodes from 2 parents and the existing nodes selected before next generation as a measure to avoid local-minima trap. The exploration-exploitation balance in GA is a difficult due to the multiple operators which affect its convergence rate, search bias and computation time, thus is a widely researched topic with ongoing research being conducted along with various works such as (Abhishek, Ranjit, Shankar, Eappen, Sivasankar & Rajesh 2020) and (Châari et al. 2014) suggesting the hybridization of GA with another algorithm to efficiently provide such balance.

2.3.2 Particle Swarm Optimization

Particle Swarm Optimization (Kennedy & Eberhart 1995) is a population and swarm-intelligence based optimization algorithm that takes inspiration from the social behaviour of a flock of birds and/or school of fishes that cooperate within members of their respective population to search for food sources. PSO performs global search and optimization by a mechanism of collective agents, denoted as particles of which all particles have global and local knowledge of their respective location within the multi-dimensional environment. PSO parameters include Pbest which represents personal best score at time t, G best which represents global best fitness score in the population and Lbest which is value obtained by choosing best and closest node.

In addition, PSO has another vector called velocity which is the mechanism of locomotion for particles to update their positions towards better solutions in the environment. Thus, it represents the distance travelled by the particle at each iteration which depends on best previous pbest (particle memory) and best previous gbest (swarm memory). Below equations represent the update equations for both velocity and position of particles as follows

$$\begin{gathered} \left\{V_i^{k+1} = wV_i^k + c1r1\left(P_i - X_i^k\right) + c_2r_2\left(P_g - X_i^k\right)\right. \\ \left\{X_i^{k+1} = X_i^k + V_i^{k+1}\right. \end{gathered} \tag{6}$$

Where i is the population size, k is the number of iterations, w is the inertia weight, c1 and c2 are local and global coefficient constants respectively which are by convention set to 2 and finally r1 and r2 are random numbers in the range between 0 and 1. Vi(k) and vi(k+1) are the velocity at iteration k and iteration k+1 respectively whereas Xi(k) and Xi(k+1) are the position at position k and iteration k+1 respectively.

Typical of swarm-optimizers, it is composed of exploration and exploitation phases and components. During the initial phases when the velocity is higher, the current solutions are biased towards exploration and in the later phases, the velocities decrease towards 0 thus focusing more on exploitation around pbest positions. The coefficient "w" (inertia coefficient) is responsible for promoting diversity and thus exploration in the population whereas the global coefficient constants, c1 and c2 are responsible to bias towards pbest and gbest respectively. The inertia coefficient is represented by equation below.

$$w = w_{\text{initial}} - \frac{w_{\text{initial}} - w_{\text{end}}}{K} k \tag{7}$$

Where winitial and wend are the values of inertia coefficient respectively.

PSO has been heavily and widely studied in literature and adapted in scientific experiments due to its fast convergence ability but is outperformed by GA when performing in depth-search albeit this is dependant on hyper-parameter variation.

(Ab Wahab et al. 2015) carried out experiments to compare the performance of ten swarm-intelligence algorithms across twenty different benchmark functions where the results observed suggest that there PSO showcases superior optimization capabilities on Colville and Langermann functions. Furthermore, in subsequent experiments, PSO achieved the theoretical optima of zero in the Beale function and along with Ant Colony Optimization (ACO) converges faster compared to other methods. Overall this experiment concluded the superior convergence ability and speed of PSO for optimization problems. These findings are further enhanced by literature findings where PSO is applied to the challenge of path-planning below.

Notable implementations of PSO applied to path-planning include works by (Foo et al. 2006) where PSO was relevantly applied to 3-Dimensional UAV path-planning. It is stated by the author that A* algorithm, albeit its strong global optimization capability is inapplicable to the author proposed environment due to terrain complexity and presence of dynamic threats i.e. obstacles, where PSO, due to its faster convergence and stochastic nature is applicable. Further parameter settings are conducted to optimize PSO towards preferring one of the multiple-objectives that include fuel-efficiency, obstacle avoidance or a balanced approach between both (which would depend on environmental constraints). Experimental results showcased the successful applicability of PSO in different test-cases where the environment is either static thus preferring shorter paths and fuel-efficiency, or dynamic where obstacle avoidance must be prioritized. The flexibility of PSO as a strong global search algorithm coupled with its fast convergence rate and flexibility for different environmental constraints offers further validity and argument for the utilization of swarm-intelligence algorithms for global path-planning.

(Shao et al. 2020) proposed a modified PSO for UAV path-planning as a measure to increase path-optimality and convergence speed. In accordance with the author's literature findings, it is stated that higher particle initial distribution lead to better solutions, thus the author proposed a chaos-based logictic map which offers better uniform distribution during particle initialization phase. Furthermore, an adaptive linear-varying strategy is implemented for acceleration coefficients and maximum velocity to ensure solution optimality within shorter convergence time and iterations. The position update topology is also improved by replacing undesirable particles during each iteration by mutant i.e. variants of desired particles obtained. In the experiment section, monte-carlo simulations are performed for fair and unbiased comparison against the baselines which include with different variants of PSO that include standard PSO and modified genetic algorithm. Experimental results showcase the improved average path optimality and average convergence speed where the author proposed methodology (CIPSO) showcased the lowest fitness values (path length) and computation time. However, the improvements are only applicable for formation UAV path planning whereby in single path-planning, the solutions attained are not always the most optimal. The most interesting experiment proposed by the author was to compare the relationship between performance and number of agents (population) where when the number of agents are reduced, it is found that the solution optimality indeed decreases however the convergence speed increases which forms another research hypothesis whereby it is necessary to adjust the population size such that optimal trade-off between path-length and computation time is achievable.

The general literature consensus for various optimization problems have proven that PSO is a reliably performing approach with strong global search and convergence capability along with fast computation-time. However, such performance is dependant on hyper-parameter variation which is difficult due to its many hyper-parameters that include population size, acceleration constant, inertia weight and velocity (Li et al. 2014). Similar to GA, Exploration-Exploitation balance is heavily dependant upon optimal tuning of its hyper-parameters where literatures such as (Xun et al. 2020) have stated several contradicting weaknesses of PSO such as global-local search imbalance, low convergence rate, low robustness and vulnerability to local-minima trap which is possible due to the complex social behaviour of PSO which is dependant on both position and velocity vectors.

2.3.3 Ant Colony Optimization

Ant Colony Optimization, abbreviated as ACO is a population-based swarm-intelligence algorithm proposed by Marcos Dorigo in 1992 and officially published in the paper titled ”Ant colony optimization: a new meta-heuristic” (Dorigo & Di Caro 1999). This is a probabilistic technique which is tasked with computing feasible paths in a graph. The algorithm is inspired from the behaviour of ants (agent) in their colony (swarm) that use pheromones as a numerical marker for communicating feasible pathways to respective food sources and visibility which is the reciprocal of distance from current position to goal position. During exploration of search space where the ants (agents) deposit pheromone markers to direct subsequent ants to the food sources. The ants retain memory of their respective positions and the quality of their solution (fitness score) which is represented by the the concentration of pheromone marker deposited by the ants whereby a higher concentration represents a better solution. Thus, this pheromone concentration mechanism acts as a positive feed-back mechanism whereby as more ants traverse through that specific path, the concentration of pheromone marker increases hence more ants will traverse through that path to find the optimal solution.

The population of ants are initialized in a discrete environment at certain node positions. At each iteration, the ant moves between nodes whereby at current position of node i, the ant k selects the path j with a certain probability which is defined in equation below

$$P_{ij}^{k}(t) = \begin{cases} \frac{[r_{ij}(t)]^{\alpha} \times [\eta_{ij}(t)]^{\beta}}{\sum_{k \text{ eallowed}}^{k} [r_{ij}(t)]^{\alpha} \times [\eta_{ij}(t)]^{\beta}}, & \text{if jeallowed }_{k} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

Where Allowed k = N-tabuk which is the nodes that the ant can select, Txy is concentration of pheromone deposited during traversal from one node to another, and are parameters to control the influence of pheromone concentration and visibility respectively, nxy is the desirability for traversal from one node to a certain other which is denoted by length between 2 nodes where a shorter length is more desirable.

The pheromone concentration is updated at each iteration by 2 mechanisms which are deposition where when more ants traverse through the path, the concentration increases and by evaporation that represents the gradual dissipation of pheromone concentration at each iteration, which is an important mechanism that controls the exploration-exploitation balance to ensure the algorithm does not get trapped in a local optima which functions similarly to the inertia coefficient of PSO. The pheromone update formula is as follows.

$$T_{ij}(t) = (1-\rho)T_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{9}$$

Where m is the population of ants and p subset (0,1) is the evaporation rate

In research and academia, ACO is successfully and most notably applied to the combinatorial optimization problems such as the Travelling Salesman Problem (TSP) which is considered a standard benchmark for meta-heuristic algorithms. (n.d.) have applied variations of ACO to 20 benchmark datasets from TSPLIB for solving the Generalized TSP where feasible results in terms of path cost and computation time are attained for less than 200 cities.

Notable implementations of ACO in the domain of path-planning include works by (Yue & Chen 2019) where ant colony optimization (ACO) and ant colony optimization with punitive measure (AS-N) are proposed. AS-N presents a negative-feedback mechanism for poorer paths where the volatilization/evaporation rate is increased for poorer solution in addition to the original positive-feedback mechanism that increases deposition rate with the aim of reducing traversal time and thus minimizing computation time. 2-Dimensional grid environment with maze-like obstacles with 2 types of complex obstacle layouts are modelled where the variants of ACO are applied. Simulation results suggest the successful application of ACO when applied to environments of varying complexities however only ACO and its variant are compared and no comparison is performed to compare the path-planning performance of ACO and its variants against other meta-heuristics.

For 3-dimensional path-planning tasks, (Cekmez et al. 2016) proposed a Multi-Colony ACO that is composed of several different ant colonies that use separate pheromone tables to maximize exploration of search space. Each ant colony produces its own optimal solutions and after each iteration, the solutions are exchanged among the neighbouring colonies to update their pheromones. This improvement was proposed to address the notable shortcomings of ACO which include slow convergence and vulnerability to local-minima trap. The experiments are conducted in a 3-Dimensional environment with artificially generated terrains and it is observed that the single colony basic-ACO got stuck, likely in a local-minima when the number of visited points increased to 100 whereas multi-colony ACO is able to handle this by having other colonies of ants choose an alternative path whereby the pheromone concentration for the in-feasible path evaporates over time.

(Dai et al. 2009) have identified 2 notable weaknesses of traditional ACO method when applied to path-planning which include low-efficiency as the initialization stages do not initialize any pheromone markers, thereby leading to increase in convergence time. Furthermore, depending on the pheromone deposition and evaporation rates in ACO, high pheromone deposition to evaporation rate will lead to premature convergence to local-optima whereas higher evaporation rate will lead to exploration bias and thereby longer computation time. Similar to PSO, ACO is also hyper-parameter sensitive. This is addressed by incorporating a new pheromone updating rule proposed by (He et al. 2013) where pheromone is only updated on the best paths where the ants (agents) will concentrate around the optimal path. This methodology adheres to the Max-Min system (Stützle & Hoos 1999) where the pheromone levels are limited between a minimum and maximum value to avoid all the pheromone from being concentrated on one path which will be counter-productive and still lead to sub-optimal paths. This method improves the exploration-exploitation balance by ensuring optimal solutions are preferred where exploitation is maintained and ensuring the difference between optimal and sub-optimal is not significant thus improving exploration concurrently.

2.3.4 Hybrid Path-Planning Methods

Hybridization is defined as combining of strong features from 2 different approaches i.e. algorithms to produce a superior algorithm that acquires the strong traits of both the combined algorithms whilst addressing the shortcomings as well. The goals of hybridization are to optimize the performance objectives of the algorithms which include minimization of path length and computation time, strengthening convergence, increasing path safety and optimization of the hyper-parameters.

As indicated by literature findings, individual approaches are prone towards biased exploration or exploitation search, which in in part due to the experimental environment settings, problem modelling and hyper-parameter settings. Recent works that have attempted to address this issue include works by (Abhishek, Ranjit, Shankar, Eappen, Sivasankar & Rajesh 2020) which propose the hybridization of PSO and GA for 3-Dimensional path-planning. In accordance with aforementioned literature findings, PSO has strong global exploration capability whereas GA has strong global exploitation capability. This results in GA being vulnerable towards pre-mature convergence and hence getting trapped in a local optima, therefore it is the author's argument that PSO aptly complements GA to achieve the objective of balancing exploration and exploitation capability where the strong global search capability of PSO and the strong local search capability of GA are combined. In the implementation section, the authors propose a fitness function which is modelled for global minima search where it is composed of the firstly, the path distance between source and destination which is represented as the euclidean distance of the average distance traversed in the x, y and z axis and secondly, collision avoidance and minimization via penalty applied to increase fitness away from optima when collision occurs where the overall fitness function a combination of metrics of path length and number of airspace collisions i.e. violations as shown below

$$V = \sum_{j=1}^{\text{nobs}} \sum_{i=1}^{100} \frac{v_i}{100} \tag{10}$$

Where V represents violations.

Feasible balance between exploration and exploitation is achieved where PSO is tuned with higher velocity component to produce higher variation in generated solutions whereas GA focuses on crossover and mutation for exploitation.

2.3.5 Glow-Worm Swarm Optimization

Glow-Worm Swarm Optimization (Krishnanand & Ghose 2005), abbreviated as GSO is another swarm-intelligence based algorithm that is inspired from the behaviour of insects in natural phenomena where the foraging behaviour of glow-worms are emulated. It is a variant of ACO where the mechanism of swarm communication and movement is through its light emitting property where the concentration i.e. the intensity of light is controlled by a chemical substance known as luciferin which represents the particular fitness score of the glow-worm.

The most notable contribution of this algorithm is the property whereby similar or differing function values can be used to find more than one optimal solutions which is facilitated by the algorithm behaviour where the swarm is partitioned into disjoint groups that converge at various local optima points to compute more than one solution.

The mechanism of action of GSO starts with initialization of glow-worm populations which are randomly initialized with range and initial luciferin value which is dependant on its relative position in the search-space which denotes its fitness score. Next, at every iteration, the present luciferin value is updated as represented in equation below

$$L_j(t) = (1-\rho)L_j(t-1) + \gamma F\left(p_j(t)\right) \tag{11}$$

Where Lj(t-1) is the previous luciferin intensity, p is luciferin decay constant (between 0 and 1), gamma is luciferin enhancer and F(Pj(t)) is the fitness of the glow worm j at current position where t is the present iteration.

After updating the luciferin value, the glow worms subsequently navigate towards the neighbouring glow worm with brighter light than its own to compute new fitness scores. This is represented in the probability equation below. The glow-worms in the swarm explores its local region to find the neighbour with the highest luciferin content which represents the optimal solution.

$$z \in N_j(t) \text{ iff } d_{jz} < rd_j(t) \text{ and } L_z(t) > L_j(t) \tag{12}$$

Where d denotes distance, z the closest neighbouring glow worm, Nj(t) is the neighbourhood, djz is Euclidean distance, rdj(t) is local decision range and finally, Lz(t) and Lj(t) are the luciferin concentrations for glow worms z and j respectively.

The best neighbour is subsequently selected from the local/neighbouring region where different neighbour have different probabilities for selection depending on luciferin content which is represented by the equation below

$$\text{prob}_{jz} = \frac{L_z(t) - L_j(t)}{\sum_{k \in N_j(t)} L_k(t) - L_j(t)} \tag{13}$$

Where z is composed to neighbourhood Nj(t).

Roulette-wheel method is used to select glow-worms with higher probability value, after which the glow-worm position (pj) is updated according to the selected neighbour position as represented in equation below.

$$p_j(t) = p_j(t-1) + s\frac{p_z(t) - p_j(t)}{\text{Distance}} \tag{14}$$

Where djz is Euclidean distance between glow-worms j and z.

Upon completing all iterations, local decision range rdj is updated using the equation

$$rd_j(t) = \min\left\{rs, \max\left[0, rdd_j(t-1) + \beta\left(nt - \mid N_j(t-1)[)\right]\right\} \tag{15}$$

Where rdj(t-1) is previous radius, rs is radial sensor constant, B is model constant and nt is parameter to restrict neighbourhood size. [Nj(t)] is the current actual neighbourhood size.

GSO has been observed to possess several strong traits that include fast convergence speed, ability to compute good-quality solutions, can deal with non-linear and multimodal optimization problems, thus has been applied to several notable optimization problems such as clustering optimization problem (Tang et al. 2019), multicast routing problem (Deng-xu et al. 2011) and 3-Dimensional path planning.

Within the domain of path-planning, recent works by (Goel et al. 2018) have demonstrated the application of GSO for 3-Dimensional path-planning environments where environment cost-factors for traversal in the 3-dimensional plane (x, y and z-axis) is introduced for scaling the step-size and traversal cost by a pre-determined value (non-algorithmic parameter) which is denoted as cx, cy and cz. The cost-function is represented by the total path-length which is denoted by number of nodes expanded and time taken to reach the goal with additional fuel comsumption metrics. The collision free path is generated by computing an obstacle free-path in the 3-dimensional plane from source to goal.Simulation experiments are conducted in four different experimental settings where in experiment 1, the algorithm successfully showcased generation of feasible paths without collision. In subsequent experiments 2 and 3, GSO showcased good performance in dynamic environments.

2.4 Recent Swarm-Intelligence Based Methods

This section will thoroughly discuss the most recent and notable swarm-intelligence algorithms introduced that have shown remarkable performances across optimization benchmarks and have been relevantly applied to 3-Dimensional path-planning works. Furthermore, the algorithms and their implementations reviewed have proposed several improvements to address the aforementioned shortcomings of classic meta-heuristic methods. Additionally, these algorithms have similar exploitation and exploration mechanism to DFO via search and convergence mechanisms in addition to simplistic nature thus allowing easier tuning of exploration-exploitation balance. More emphasis is placed upon this section of the literature review due to high relevance and correlation to proposed methodology.

2.4.1 Grey-Wolf Optimization - Minimalist Swarm Optimizer

Grey-Wolf Optimization (Mirjalili et al. 2014), abbreviated as GWO, is another recent population-based swarm-intelligence algorithm which is inspired from the social interaction between grey-wolf packs where the social leadership and hunting behaviour of grey-wolves are emulated. Differing from other swarm-intelligence analogy, the element of hierarchy is strictly adhered to in packs of grey-wolves where the domination hierarchy is composed of Alpha wolves which is the leader and thus most dominant individual, Beta wolves which are second in hierarchy that assists Alpha wolves, Delta wolves that are third in hierarchy and at the very bottom, the Omega wolves which are the last to eat in the pack. This is represented algorithmically by denoting the Alpha as the best fitness solution and beta, delta and omega as second, third and worst fitness solutions respectively.

Mathematically the phases of the wolves encircling the prey i.e. the optima during the hunting process is represented in equations below

$$\begin{aligned} \vec{D} &= \left|\vec{C} \cdot \vec{X}_p(t) - \vec{X}(t)\right| \\ \vec{X}(t+1) &= \vec{X}_p(t) - \vec{A} \cdot \vec{D} \end{aligned} \tag{16}$$

where t represents the current iteration, A and C are coefficient vectors and X is the current position of the prey i.e. optimal solution whereas X represents the current position(s) of the grey wolf which are calculated in equation(s) below

$$\begin{aligned}\vec{A} &= 2\vec{a}\cdot\overrightarrow{r_1}-\vec{a}\\ \vec{C} &= 2\cdot\overrightarrow{r_2}\end{aligned}\tag{17}$$

The next phase is called the hunting phase where the Alpha wolf guides the overall hunt

$$\begin{aligned}\overrightarrow{D_\alpha} &= \left|\overrightarrow{C_1}\cdot\overrightarrow{X_\alpha}-\vec{X}\right|\\ \overrightarrow{D_\beta} &= \left|\overrightarrow{C_2}\cdot\overrightarrow{X_\beta}-\vec{X}\right|\\ \overrightarrow{D_\delta} &= \left|\overrightarrow{C_3}\cdot\overrightarrow{X_\delta}-\vec{X}\right|\end{aligned}\tag{18}$$

With beta and delta participating occasionally. It is assumed the Alpha followed by Beta and Delta have the best knowledge of the potential optima.

$$\begin{aligned}\vec{X}_1 &= \vec{X}_\alpha-\overrightarrow{A_1}\cdot\left(\overrightarrow{D_\alpha}\right)\\ \vec{X}_2 &= \vec{X}_\beta-\overrightarrow{A_2}\cdot\left(\overrightarrow{D_\beta}\right)\\ \vec{X}_3 &= \vec{X}_\delta-\vec{A}_3\cdot\left(\overrightarrow{D_\delta}\right)\end{aligned}\tag{19}$$

These three best solutions will be followed by the other members of the population (omega wolves) by updating their positions as represented in equations below

$$\vec{X}(t+1)=\frac{\overrightarrow{X_1}+\overrightarrow{X_2}+\overrightarrow{X_3}}{3}\tag{20}$$

The next phase that follows hunting phase is the attacking phase where the grey wolves attack the prey once it stops moving i.e. is stationary. This is modelled by decaying the value of a from 2 to 0 to decrease the distance between the wolves and the position of prey where [A] value less than 1 makes the wolves attack the prey (exploitation).

During the search process, the wolves search for the prey according to relative positions of alpha, beta, delta and omega where the agents diverge during the search process and converge during the attack process thus are exploring the search space during search where [A] value more than 1 makes the wolves diverge to find better preys i.e. solution (exploration). Another exploration based parameter of GWO is the c value which contains random values in the range [0,2] which denote weights to emphasize or disregard attack.

Benchmark evaluation on 23 classical benchmark functions which consist of 7 unimodal benchmark functions, 6 multimodal benchmark functions and 10 fixed-dimension multimodal functions showcase highly competitive results where in unimodal evaluation, GWO showcased good exploitation capability, in multimodal evaluation, GWO showcased good exploration capability and in composite functions evaluation, showcase excellent local optima avoidance which is due to its divergence capability to escape a local optima thus is a significant contribution towards solving the problems of balancing exploration and exploitation which is faced by well-known metaheuristics such as GA, PSO and ACO.

Within the domain of path-planning, recent notable applications include works by (Kiani et al. 2021) that applied GWO to 3-Dimensional multi-agent path-planning where its performance in simulation experiments are compared to deterministic algorithms which are

Dijkstra, A*, D* and meta-heuristics which are PSO, GSO and others. The problem modelling is described as producing a suitable collision-free path i.e. trajectories from source to target. The path length is represented as minimization of the objective (fitness) function. The author proposed GWO along with the compared algorithms used as benchmarks are evaluated on three 3-dimensional maps : map 1 is a simple environment with simple obstacle layout, map 2 is a moderately complicated environment and map 3 is a complex environment with numerous randomly placed obstacles.

Simulation results for path length and computation time suggest that GWO, albeit producing slightly lengthier i.e. less optimal paths when compared to Dijkstra, A* and D* algorithms, the computation time however was significantly better as GWO took approximately half the time as required by classical path-planning methods in environment 1 and approximately 31.25 percent and 38 percent lesser time in environment 2 and 3 respectively, thus proving to be an efficient trade-off between path-optimality which was minimal for significantly better computation time.

GWO was also compared to notable meta-heuristic methods such as GA, PSO and a recent method which is GSO. Simulation results produced show the paths produced by GWO for all three maps are shorter and takes less time to converge and find those optimal paths even considering the fast convergence ability of PSO and thereby showcasing a successful "proof-of-concept" for the applicability of GW0 for real world and NP-Hard optimization problems to achieve state of the art results. During the experiments, it is observed that GWO is less vulnerable towards local-minima trap due to its minimal parameters to be adjusted compared to PSO and GSO which also subsequently facilitates faster convergence speed.

2.4.2 Whale Optimization Algorithm - Exploration-Exploitation Balance

Whale Optimization Algorithm (Mirjalili & Lewis 2016), abbreviated as WOA is a population based swarm-intelligence technique where the bubble-net hunting mechanism of humpback whales is mimicked mathematically. Whales are highly intelligent species which share common spindle cells in certain areas of their brain thus enabling them to think, learn, make decisions and communicate similar to humans. Humpback whales perform a special hunting method called bubble-net feeding method where prey are usually hunted close to the surface of the sea via the whales creating a bubble in a spiral shape around the prey. When the location of the prey (optima) is known, the humpback whales will assume that is the best current agent close to the solution and the other agents will update their positions towards this best search agent close to the current agent and this update equation is represented as follows :

$$\begin{aligned} \vec{D} &= \left|\vec{C}\cdot\overrightarrow{X_p}(t)-\vec{X}(t)\right| \\ \vec{X}(t+1) &= \vec{X}_p(t)-\vec{A}\cdot\vec{D}(1) \end{aligned} \tag{21}$$

where t is the current iteration/time step, A and C are coefficient vectors, X is the position of the prey whilst X is the position vector the agent in the population (whale). The values A and C are represented as

$$\begin{aligned} \vec{A} &= 2\vec{a}\cdot\overrightarrow{r_1}-\vec{a} \\ \vec{C} &= 2\cdot\overrightarrow{r_2} \end{aligned} \tag{22}$$

where a decreases linearly from 2 to 0 during the iterations and r1 r2 are random values between [0,1].

Similar to convergence and divergence phases observed in other swarm optimizers, WOA is composed of two phases namely bubble-net attacking phase which represents exploitation whereby this is governed by a shrinking-encircling mechanism where the value of a is decreased thorough the iterations. Secondly, is the spiral-updating position where the distance between the agent and the target is is calculated to model the helix-shaped whale movement as represented in equation below

$$\vec{X}(t+1) = \overrightarrow{D'} e^{bt} \cos(2\pi t) + \overrightarrow{X^*}(t) \tag{23}$$

There is 50 percent probability between selecting one of the two mechanisms denoted above for updating whale position vectors in the environment.v

The second phase is the searching phase, similar to that observed in GWO whereby this represents exploration. When the value of A is greater than 1 or less than -1, this forces the agents to disperse i.e. move away from a certain reference agent where the position vectors of the agents are updated via randomly choosing a new reference agent instead of the current best agent to search for better solutions (global optima). This is represented mathematically below as follows:

$$\begin{aligned} \vec{D} &= \left|\vec{C} X^{\vec{X}}_{\text{rand}} - \vec{X}\right| \\ \vec{X}(t+1) &= \overrightarrow{X_{\text{rand}}} - \vec{A}\vec{D}(3) \end{aligned} \tag{24}$$

Where xrand is a random position of new reference agent.

WOA has underwent 29 benchmark evaluations which are unimodal, multimodal, fixed-dimension multimodal and composite functions where WOA showcased competitive exploitation and exploration capabilities where it is ranked between 1 and 2 for the unimodal and multimodal functions. Furthermore, WOA was the best optimizer for composite test functions thereby showcasing exploration-exploitation balance by being able to escape local minima.

WOA has been implemented for 3-Dimensional path-planning by (Yan et al. 2022) for AUV navigation in underwater environment as a novel method for exploration-exploitation balance. Simulation experiments are carried out in 3-Dimensional grid-environment by using spherical obstacles to represent positions of obstacles i.e. threat in underwater environment. Baseline methods such as GWO, Harris Hawk Optimization (HHO), Marine Predator Algorithm (MPA), Moth-Flame Optimization (MFO), Multiverse Optimizer (MVO), Sine Cosine Algorithm (SCA), Spotted Hyena Optimizer (SHO) and Salp Swarm Algorithm(SSA) are introduced to provide an objective comparison. The results obtained in indicated the outperformance of the author proposed methodology over the compared base-line methods further support the hypothesis and objective whereby WOA successfully avoids pre-mature convergence to local-optima which showcased significant improvement compared to its predecessors such as GWO.

2.4.3 Dispersive Flies Optimization

Dispersive Flies Optimization (al Rifaie 2014*a*), abbreviated as is a population based, swarm-intelligence algorithm which derives inspiration from the phenomena of flies in nature that hover i.e. swarm over food sources. According to aforementioned literature, meta-heuristics are difficult to implement and optimize due to numerous possible combination of hyper-parameters which affect the algorithm performance to bias either towards explorative or exploitative search. DFO represents the new generation of swarm-optimizers which are minimalistic and simplistic in nature, that possesses only one hyper-parameter called the update equation that updates the population positions using position vectors of the flies.

The mechanism of exploration and exploitation of DFO is composed of swarming (convergence) when a food source has been found and dispersion (divergence) when faced with a threat. This is similar to the hunt (convergence) and search (divergence) mechanism observed in GWO and WOA. The swarming and dispersion mechanisms are represented in equations below

$$\vec{x}_i^t = \left[x_{i0}^t, x_{i1}^t, \ldots, x_{i,D-1}^t\right], \quad i \in \{0, 1, 2, \ldots, N-1\} \tag{25}$$

Where i denotes the ith individual, t is the present time step, D is dimension and N is the size of the population.

During the first iteration, when the time step is initialized at 0, the dth component of the ith fly is initialized as observed in equation below that generates a random value between the lower and upper bounds of the dimension, d that generates random values between lower and upper bounds according to the dimension as shown below

$$x_{id}^0 = \mathrm{U}\left(x_{\min,d}, x_{\max,d}\right) \tag{26}$$

Similar to that observed in PSO, GWO and GSO update equations, the agents in a population follow the members of the population i.e. converge towards members who are closer to the optimal solution i.e. have better fitness scores in the population. In DFO, a similar methodology is employed where the position of the flies are independently updated according to the best neighbouring individual relative to the fly position which forms a continuation as each fly has a left and right neighbour thus this process continues until all flies follow their best relative neighbours to converge towards the best fly position in the swarm which represents the closest solution to the global optima in the environment. This positional update equation is represented in the equation below

$$x_{id}^{t+1} = x_{i_n d}^t + u\left(x_{sd}^t - x_{id}^t\right) \tag{27}$$

Where xtid is the position of ith fly in dth dimension at iteration t, xtind is the best neighbour position, xtsd is the best agent in the swarm and u U(0,1) are randomly updated for each dimension.

Despite the minimalist architecture of DFO, an exploration component of DFO called disturbance is employed to displace the position of flies to discover better solutions in the environment, where the element of stochasticity (randomness) is implemented in the update process to reset the position (dispersal) of the flies if the random number generated between U(0,1) is less than the restart threshold (Majid-al Rifaie 2020) thus acting as a protective mechanism against local-minima trap and thereby pre-mature convergence. Henceforth, the disturbance threshold parameter is responsible for enhancing diversity into the population and for optimizing exploration-exploitation balance.

To verify the out-performance of the proposed algorithm in addition to its simplistic nature, (Al-Rifaie 2015) conducted benchmark evaluation on fourteen objective functions is conducted where the performance of DFO is evaluated and compared against other meta-heuristics which are PSO, GA and Differential Evolution (DE). Furthermore, the author also conducted separate tests to verify the role of disturbance threshold, d and its role in helping the flies search for better solutions by comparing it to DFO without the disturbance mechanism called "DFO-C". It was observed that DFO outperforms DE, PSO and GA in 66.67 percent, 58.33 percent and 85.71 percent of the benchmark functions respectively. Furthermore, the tests performed also indicate that adding element of diversity in the population increases the exploration thus preventing premature-convergence where DFO with delta, d of 0.0001 was proven to be 79 percent more efficient and 92 percent more reliable than DFO-C.

DFO has been applied to a variety of optimization tasks in the fields of medicine, animation and optimization of machine-learning techniques. In (Al-Rifaie 2015), DFO is applied to medical imaging task for detection of micro-calcifications on the mammographs where the affected regions of interest processed using the flies from DFO. 50 000 agents are initialized to search for areas of metastasis and calcification in the search space which is the input scan image with fitness value evaluated by calculating number of pixels around pixel chosen by the agent. The strong search capability of DFO is proven by the novel application of DFO towards searching for regions of interest in scan images therefore making DFO a suitable candidate for search-based tasks.

Further novel implementations of DFO include (Alhakbani & al Rifaie 2017) where DFO is tasked with optimization of classification-based Machine-Learning algorithm to solve classification problem known as "class-imbalance" where classification bias occurs where minority data classes are ignored due to majority of training done on more frequently occuring class instances. The kernel hyperparameters c and of Support Vector Machine (SVM) are optimized using DFO to search for the optimal values for each. Experiments are conducted on 8 real-world datasets and comparison is performed with similar techniques such as PSO, Grid-Search and Random-Search methods. The experimental results indeed conclude the and verify the outperformance of DFO which is found to be statistically improve the performance of the classifier and its performance on the 8 datasets when compared against other techniques.

Further works by (Hooman et al. 2018) have utilized the strong search capability of DFO to optimize Deep Learning methods. The optimal weights for Dense Neural Network and Convolution-Dense Neural Network are computed using DFO where each fly holds a score for the Physionet score. Prior to updating the weights, value u from U(0,1) is sampled and if it is less than delta, the weights are updated, especially that of the best fly in the swarm which holds the highest Physionet score. Experimental results conclude the robust performance of DFO outperforms Neural Networks with back-propagation and random search tecnhiques.

The above-mentioned out-performance of DFO across various search and optimization tasks serve as added motivation for implementing Dispersive Flies Optimization for 3-Dimensional path-searching challenge due to its simplistic exploration-exploitation balance, fast convergence, multi-dimensional search capability and proven success in notable real-life implementations.

2.4.4 Summary of Meta-Heuristics Literature Findings

Table 2: Analysis and Comparison of Meta-Heuristics Method

Algorithm	Inspiration	Exploration-Exploitation	Strength	Weakness
GA	Genetic-Crossover via reproduction	Exploration via mutation and crossover Exploitation via Selection	Computationally efficient Strong in-depth search (exploitation)	Prone to local-minima trap (premature convergence) Sensitive to its operators
PSO	Social behaviour of flock of birds	Exploration via higher velocity. bias towards c2 and w to increase diversity Exploitation when velocity approaches 0 and with bias towards c1	Strong global convergence Fast convergence speed Good exploration-exploitation balance capability Suitable for multi-objective optimization	Multiple hyper-parameters Depending on hyper-parameter tuning is biased either towards exploration or exploitation Weak exploitative search Difficult to optimally balance exploration and exploitation
ACO	Pheromone deposition and evaporation of ants when traversing a path	Exploration via evaporation of pheromone to explore better paths Exploitation via deposition of pheromone as ants traverse a path	Suitable for combinatorial optimization such as TSP Positive feedback mechanism via pheromone concentration Can be parallelized	Long convergence time due to inefficient initialization Premature convergence Hyper-parameter sensitive
Hybird Methods	Combination of strong traits from 2 or more different algorithms	Aims to optimize or improve exploration-exploitation balance	Literature findings demonstrate improvement in both solution optimality and computation time	Difficult to implement
GSO	Glow-worm communication via luciferin	Convergence occurs in the direction of brightest neighbour with highest luciferin concentration	Fast convergence speed Good quality solutions Applicable for non-linear and multimodal optimization thus can find multiple optimal solutions	Poor when dealing with high-dimensional problems
GWO	Search and hunting behaviour of packs of grey wolves	Exploration via searching for prey during initial iterations when [A] is more 1 Exploitation via hunting the prey during later half of iterations when [A] is less than 1<1	Excellent and easy Exploration-Exploitation balance Minimal number of hyper-parameters (a and C value)	Still fairly new thus needs extensive literature findings and implementations to various other problem domains such as combinatorial optimization and multi-agent path-planning
WOA	Search and attacking behaviour of hump-back whales	Exploration via searching phase when [A] is less than 1 which forces agents to disperse Exploitation via bubble-net attacking phase where a is decreased from 2 to 0	Excellent and easy Exploration-Exploitation balance Ability to escape local-minima	Still fairly new thus needs extensive literature findings and implementations to various other problem domains such as combinatorial optimization and multi-agent path-planning
DFO	Search and hovering of flies over food sources	Exploration via dispersal and search when random value between (0,1) is less than 1 Exploitation via positional update according to best value	Exploration-Exploitation balance achievable Simple mechanism to escape local minima	No prior implementation for path-planning which remains a research gap

3 Proposed Methodology and Implementation : 3-Dimensional Path-Planning using Dispersive Flies Optimization

In this section, the adaptation of DFO along with relevant modifications and adaptations for 3-Dimensional path-planning are showcased.

3.1 Application of DFO to 3-D Path-Planning

Several adaptations and modifications were necessary to apply DFO towards 3-Dimensional path-planning in the specified environment. The workflow of the proposed methodology is described in sections below.

3.1.1 Define Environmental parameters and Bounds

The path to the directory containing the environment data is defined and the parameters of the search space namely upper and lower bounds are set. The step-size of the environment which defines how far apart each node are is defined for all three axes. The start id and goal id are rounded across the x, y and z-axes and a constraint is introduced where the maximum path cannot exceed path-length of 100. The upper and lower boundaries of the implementation are set as (0, 0, 0) and (25, 25, 25) respectively with step-size set as 0.4 in all axes. This is shown in appendix 9.2.1

3.1.2 Initialization of Agents in the Environment

The population of flies, which are the initial paths are initialized in the environment. The initial paths are chosen such that the corresponding collision map is empty to ensure it is a collision-free path. —This is shown in appendix 9.2.2

3.1.3 Main DFO Loop

In the main DFO loop, the hyper-parameter which defines the disturbance threshold(d-value) and maximum number of iterations are defined. An array matrix is initialized to store all the paths produced. For each iteration, the best record is kept as a mechanism of elitism to maintain the best-found solution in the population. The loop is shown in appendix 9.2.3

In addition to the mechanism of updating each flies relative position according to the best neighbouring fly, external factors such as presence of obstacles need to be taken into consideration as well where the cmap (collision map) information which denotes the position of obstacles and thus collision points in the search-space is known by all flies. Therefore, a novel hierarchical mechanism is introduced as performed in literature finding (Mirjalili et al. 2014) to categorize neighbouring nodes according to four levels of hierarchy to combine both constraints of updating positions towards best neighbouring positions and towards neighbours without collision. Accordingly, the list of neighbours are hierarchically categorized into four different list data structures as follows.
1. nid1 : Neighbour nodes without collision
2. nid2 : Valid neighbour nodes without collision and closer to optima
3. nid1nv : Neighbour nodes without collision which is not visited
4. nid2nv : Valid neighbour nodes without collision, closer to optima and not yet visited

When searching for the best neighbour, the flies will preferably select valid neighbour nodes without collision, closer to the optima and not yet visited (nid2nv) and if this does not exist, then the flies will select neighbour nodes without collision and not yet visited (nid1nv) and if this does not exist, the flies will select valid neighbour without collision and closer to optima (nid2) and in the worst outcome when the three best category of neighbours are unavailable, the flies then resort to choosing the neighbour nodes without collision (general neighbours). When there are no neighbours left, the program breaks and no more update occurs. This can be represented by pseudo-code below.

```
while  at current position
if unvisited and valid neighbour exist (nid2nv) then
  select nid2nv
else if unvisited meighbour exist (nid1nv) then
  select nid1nv
else if valid neighbour exist (nid1) then
  select nid1 any neighbour without collision exist (nid1)

  select nid1
else
  break
end if do
```

The ring-topology is strictly adhered by categorizing the chosen neighbour position to update towards by differentiating between negative and positive neighbours whereby if the neighbour is located behind (x-axis), to the left (y-axis) and below (z-axis) relative to the flies' current position then update occurs in the negative direction towards valid visited neighbour which is closer to goal (nid2) and if the neighbour is located in-front(x-axis), to the right (y-axis) and above (z-axis) then update occurs in the positive direction towards valid unvisited neighbour which is closer to the goal node (nid2v).

This mechanism of assigning weights or hierarchy to the nodes allow the flies to directionally and preferentially explore unvisited nodes whilst preserving the best found solutions using elitism.

3.2 Collision Avoidance

Obstacle avoidance is concerned with the task of the carrier module (UAV) in navigating safely from start position to end position without colliding with obstacles by the use of collision avoidance mechanisms. The implementation of collision avoidance function is to satisfy the second research objective of path-safety.

3.2.1 Obstacle generation

The obstacles are generated artificially with coordinates and dimensions specific to the shapes of the obstacles defined in external comma separate value (CSV) files using separate functions to draw each of the shapes. The obstacle generation technique is in line with literature finding by (Guo et al. 2021) that have suggested that it is reasonable to model obstacle after convex shapes in 3-Dimensional work-space. The relevant environmental configurations and integration steps obtained from (Fei 2021).

Three types of obstacles are generated as shown below

1. Spheric Obstacle that denote radius where UAV cannot traverse through. This can represent no-fly zones where UAV's are barred from operating such as commercial airspaces near airports, military bases and hostile regions.

2. Cylindrical Obstacles which represent concrete or tangible obstacles such as tall buildings, equipment, pedestrians or any random objects that exist in environment.

3. Cone-shaped Obstacles that are irregular-convex obstacle in the workspace which represent irregularly shaped obstacles such as trees.

Accordingly, if there are n obstacles in the environment, then the environment modelling is performed as follows.

$$\Gamma(\boldsymbol{P}) = \left(\frac{x - x_{\text{obs}}}{a}\right)^{2p} + \left(\frac{y - y_{\text{obs}}}{b}\right)^{2q} + \left(\frac{z - z_{\text{obs}}}{c}\right)^{2r} \tag{28}$$

Where a,b,c represents size whereas p,q,r represents shape whereas (xobs, yobs, zobs) are the centre coordinates of the obstacle. P is the surface of obstacle and if value of Gamma P is more than 1, this is outside space and if less than 1 this is inside space. The functions to drawn the obstacles are shown in appendix 9.2.5 which is referenced from (Guo et al. 2021)

3.2.2 Collision Path Detection and Avoidance

To model the conditions for a collision, literature findings from (Abhishek, Ranjit, Thangavelu, Eappen, Sivasankar & A 2020) and (Park et al. 2019) are partly implemented where collision occurs when the distance between the UAV and the obstacle is smaller than or equal to the radius of the obstacle. However, this alone does not take into consideration real-life constraints such as the size and shape of the carrier-module where the lateral ends are still subject to collision. Thus, a safety margin, as advised in literature findings (N & V 2019) is adhered to accommodate and take into consideration the dimensions of the carrier module (UAV) where the UAV must at all times must ensure a minimum safety margin (gap) between itself and the obstacles, whereby if the distance between the UAV and the surface of the obstacle is less than the safety margin, thus collision has occurred and the collision flag is raised to true. Accordingly, the occurrence of collision with nth obstacle is therefore defined as follows.

$$Collision(n) = \begin{cases} No - Collision & d > sm + r \\ P & d \leq sm + r \end{cases} \tag{29}$$

Where d is the minimum distance the UAV must maintain between itself and the centre of the obstacle which is composed of the safety margin - sm and the radius of the convex shapes. If d is less or equal to the radius of shape, the collision with surface of the shape has occurred and a penalty function, P is computed.

Each time a collision is detected in above for-loops, a penalty flag ,f ,is set to true and added as a penalty value, dsg to the fitness function.

3.2.3 Path-Adjustment

The getPath method obtains the path from start to end with random lengths. Thus path-adjustment technique is employed to insert new nodes randomly between two neighbouring nodes to adjust the lengths of the paths such that it is uniform. The insertion points are selected randomly similar to the mutation operation of genetic algorithm as shown below

[1, 2, 3, 4, 5, 6]: length = 6. (Length must be 9.)
[1,7,2,8,3, 4,9,5, 6]: length = 9. (Inserted at 1, 2, and 4.)

3.3 Fitness Function

Fitness function is defined as an objective scoring mechanism to assess the optimality of the solution generated according to the pre-defined objectives. The proposed fitness function for the 3-Dimensional path-planning experiment is a multi-objective fitness function to evaluate the minimization objectives of path-cost, path-safety with the computation time evaluated in parallel.

3.3.1 Path-Cost

The component of path-cost is the distance of the path from start to end returned by the algorithm. The minimum distance between 2 points is computed using Euclidean distance metric which is preferred for continuous trajectory generation compared to Manhattan distance metric which always returns straight lines, thus is not suitable for complex obstacle avoidance and navigation. The equation for path cost using Euclidean distance metric is shown below.

$$F = \sum_{i=1}^{n} \left[\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 + (z_{i+1} - z_i)^2} \right] \tag{30}$$

3.3.2 Path-Safety

Elimination of collision is the second component of the multi-objective fitness function. The UAV must maintain a safe distance between itself and the obstacle by a minimum margin as discussed in the collision avoidance section. Penalty value is added to the total fitness score whenever the position of the UAV breaches the minimum distance between itself and the obstacle as explained in collision avoidance section, which is counted as a collision which significantly increases the fitness score away from the minima which represents the global optimum. This penalty-based collision minimization mechanism is part-inspired from aforementioned literature finding (Abhishek, Ranjit, Thangavelu, Eappen, Sivasankar & A 2020)

3.3.3 Overall Fitness Score

Therefore, the multi-objective fitness function takes into consideration both the adjusted path-length via converging towards shorter paths obtained and the safety of the path via penalization when collision occurs. This is represented by the Euclidean distance metric which is used as the path-distance calculation metric as aforementioned where the penalization is applied to.

$$F = \sum_{i=1}^{n} \left[\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 + (z_{i+1} - z_i)^2} \right] + P \tag{31}$$

Where xi+1, yi+1 and zi+1 represent the distance of next nodes (neighbour nodes) from start position from x, y and z axis, xi, yi and zi represent the distance of current nodes from start position and P represents the penalty function applied for the path when obstacle is detected thus denoting it as a path with collision whereas n represents the total number of paths. This fitness function therefore can be further simplified and denoted as follows.

$$F = \sum_{i=1}^{n} \text{Path Cost in } (x, y, z) + P \tag{32}$$

3.4 Path-Smoothing

The paths produced are jagged with sharp turns, sudden elevation and descent which is due to the grid-based environment used, thereby unsafe for real-time navigation. These non-smooth motion can result in increased energy cost due to sudden acceleration and deceleration, damage to the delicate components of the carrier module and potential collision due to the effect of inertia when sudden change of motion occurs especially near collision points (Ravankar et al. 2018). Therefore, the paths generated are not suitable for continuous flight operation. Therefore, a path-smoothing technique mechanism is proposed to satisfy the kinematic requirement of smooth continuous flight.

In literature, there are many path-smoothing techniques such as interpolation based path smoothing (Páleš & Rédl 2015) and cubic B-spline curve Elbanhawi et al. (2015). For the purpose of simplicity and to keep computational complexity at minimal, a moving average is proposed. Moving average is defined as the unweighted mean of previous amount of data points. The data points in this case are the previous nodes where the mean is calculated over the last k number of points. Moving average are computationally cheap to implement as the it is only composed of 3 arithmetic steps and uses a FIFO (first in first out) structure to add the nodes hence will not significantly increase computation time. The formula for moving average used is shown below

$$y(k) = \frac{1}{N} \sum_{i=k}^{N+k-1} x_i \tag{33}$$

3.4.1 High-Level Summary flowchart of Proposed Methodology

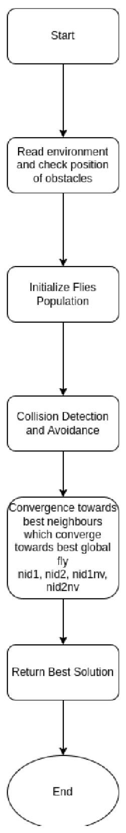

Start
Read environment and check position of obstacles
Initialize Flies Population
Collision Detection and Avoidance
Convergence towards best neighbours which converge towards best global fly nid1, nid2, nid1nv, nid2nv
Return Best Solution
End

Figure 4: Flowchart of Proposed Methodology

4 Experiment

Experiment is conducted to address the research gaps as outlined in the research objectives. Prior to performing the experiments, it is ensured that the environmental settings are similar for all algorithms where applicable such as step-size - which is applicable for DFO, ACO and RRT algorithms. Secondly, the hyper-parameter settings for each algorithm has different effects on each algorithm thus complete equalization of hyper-parameters such that the net effect is similar for all compared algorithms is not possible. Thus, each algorithm will be compared according to the best relative equalization through trial and error with part setting through literature reading.

The experiment is composed of two phases for each environment where the first phase will be focused on observing and selecting the optimal hyper-parameter (disturbance threshold and population-size) value for DFO, where a modified DFO with decaying disturbance threshold is additionally proposed.

In the second phase where baseline comparison is performed, the baseline methods which are A* Algorithm, Rapidly Exploring Random Trees (RRT) and Ant Colony Optimization are used for performance comparison where A* algorithm is tasked with defining the global-optimal path length in each environment for which percentage error is calculated for the proposed methodology and other baseline methods as an additional comparison metric.

4.1 Experimental Settings

The baseline methods and testing environment is taken and adapted from (Fei 2021) from the url : github.com/ZYunfeii/UAVObstacleAvoidingDRL
Whereas the proposed methodology is original work albeit minor adaptations such as function to compute moving average, environment and functions to draw obstacles into environment.
The step-size parameter is set to 0.4 units in all dimension of movement (not applicable for A*) which denotes how far apart each node is spread in the environment. The proposed simulations are conducted using MATLAB Online which uses Intel(R) Xeon(R) Platinum 8259CL CPU @ 2.50GHz with 8 gigabytes of RAM. Due to the online environment used there may be variation of computation time observed due to variation in connection speed. Below table is the parameter values used for the 3-Dimensional environments proposed.

Table 3: Experimental and Environmental Setting

Parameter	Environment 1	Environment 2	Environment 3
Start Coordinate	(0,0,0)	(0,0,0)	(0,0,0)
Goal Coordinate	(6,6,0)	(10,11,2)	(15,7,2)
Number of Spherical Obstacles	1	3	9
Number of Cuboidal Obstacles	0	1	29
Number of Cone-like Obstacles	0	1	1
Total Number of Obstacles	1	4	39
Obstacle Placement	Centre	Random	Random

4.1.1 Hyper-Parameter Settings

The relevant parameters for A* algorithm and RRT are the step sizes where as aforementioned, step size of 0.4 is used for RRT whereas step-size of 0.7 is used for A* as smaller step-size will result in extremely longer computation-time which will render computation infeasible. The optimal hyper-parameter setting for ACO is partially according to (Sariff & Buniyamin 2010) where the pheromone deposition rate, alpha is 1 and the pheromone evaporation rate, rho is 0.5 denoting 50 percent evaporation rate. It is noticed that smaller rou value results in failure to run program, which could be due to high computational load. The population size chosen is 15 to favour competitive computation with other compared methods. Trial and error method is used to pareto-optimally select the suitable hyper-parameters of delta and population size. DFO with decaying delta value is additionally proposed as an attempt to balance between exploration and exploitation as observed in GWO and WOA where the disturbance is set to a higher value for the first few iterations to emulate higher global exploration and thereafter decays to low delta value of 0.0001 to emulate local exploitative search of promising regions. Three variants are proposed after thorough trial and error combination.

4.2 Experimental Results : Environment 1

4.2.1 DFO Environment 1

Table 4: Environment 1 DFO Delta Value Comparison

DFO (Population 100)	**Mean Length**	**Mean Time**	**Average Path Optimality**	**Shortest Path / Computation Time**
DFO(d=0.0001)	17.57333333	11.40566667	14.4895	
DFO(d=0.0005)	17.6782	11.0896	14.3839	
DFO(d=0.001)	18.98075	11.02725	15.004	
DFO(d=0.01)	20	11.0094	15.5047	
DFO Decay {0.1 - 0.0001}	**18.0682**	**10.9864**	**14.5273**	**Shortest Time**
DFO decay {0.01 - 0}	18.251	11.2714	14.7612	
DFO decay (exponential)	**17.5082**	**11.3086**	**14.4084**	**Shortest Path**
DFO (Population 50)				
DFO(d=0.0001)	**18.0614**	**6.6416**	**12.3515**	**Shortest Path Shortest Time**
DFO(d=0.0005)	18.4028	6.649	12.5259	
DFO(d=0.001)	19.0714	6.8272	12.9493	
DFO(d=0.01)	20	6.9004	13.4502	
DFO Decay {0.1 - 0.0001}	18.5012	6.66	12.5806	
DFO decay {0.01 - 0}	18.7466	6.654	12.7003	
DFO decay (exponential)	18.2734	6.6508	12.4621	
DFO (Population 15)				
DFO(d=0.0001)	19.1742	3.8404	11.5073	
DFO(d=0.0005)	19.7796	3.8728	11.8262	
DFO(d=0.001)	20.77	3.8296	12.2998	
DFO(d=0.01)	21.12	3.6732	12.3966	
DFO Decay {0.1 - 0.0001}	**19.477**	**3.4768**	**11.4769**	**Shortest Time**
DFO decay {0.01 - 0}	19.5132	3.6092	11.5612	
DFO decay (exponential)	**19.14**	**3.5518**	**11.3459**	**Shortest Path**

4.2.2 Baseline Comparison for Environment 1

Table 5: DFO versus Baseline Environment 1

Algorithm	**Average Path Length**	**Average Computation Time**	**Percentage Error from Global Optima**	**Average Optimality Score**
A*	10.40954	13.7998716	0% (NA)	12.1047058
RRT	**13.447976**	**3.7983174**	**29.189%**	**8.6231467**
ACO	12.923797	22.3177945	24.153%	17.6207959
DFO Pop 100 (Best)	17.5082	11.3086	68.194%	14.4084
DFO Pop 50 (Best)	18.0614	6.6416	73.51%	12.3515
DFO Pop 15 (Best)	19.14	3.5518	83.870%	11.3459

3.7983174

4.2.3 DFO Environment 2

Table 6: Environment 2 DFO Delta Value Comparison

DFO (Population 100)	Mean Length	Mean Time	Average Path Optimality	Shortest Path / Computation Time
DFO(d=0.0001)	**21.0778**	**13.6998**	**17.3888**	**Shortest Path**
DFO(d=0.0005)	21.5698	13.621	17.5954	
DFO(d=0.001)	22.32	13.6228	17.9714	
DFO(d=0.01)	23.9294	13.5026	18.716	
DFO Decay {0.1 - 0.0001}	21.7486	13.3862	17.5674	
DFO decay {0.01 - 0}	21.855	13.3634	17.6092	
DFO decay (exponential)	**21.477**	**13.1666**	**17.3218**	**Shortest Time**
DFO (Population 50)				
DFO(d=0.0001)	**20.979**	**7.934**	**14.4565**	**Shortest Path Shortest Time**
DFO(d=0.0005)	22.6176	8.054	15.3358	
DFO(d=0.001)	22.8292	7.9968	15.413	
DFO(d=0.01)	23.638	8.082	15.86	
DFO Decay {0.1 - 0.0001}	22.7078	7.9894	15.3486	
DFO decay {0.01 - 0}	22.8416	8.047	15.4443	
DFO decay (exponential)	22.3694	8.1746	15.272	
DFO (Population 15)				
DFO(d=0.0001)	**23.7184**	**4.2598**	**13.9891**	
DFO(d=0.0005)	23.486	4.2368	13.8614	
DFO(d=0.001)	23.7888	4.2498	14.0193	
DFO(d=0.01)	**24.7294**	**4.228**	**14.4787**	**Shortest Time**
DFO Decay {0.1 - 0.0001}	23.531	4.365	13.948	
DFO decay {0.01 - 0}	**23.3934**	**4.3374**	**13.8654**	**Shortest Path**
DFO decay (exponential)	23.9982	4.2996	14.1489	

4.2.4 Baseline Comparison for Environment 2

Table 7: DFO versus Baseline Environment 2

Algorithm	Average Path Length	Average Computation Time	Percentage Error from Global Optima	Average Optimality Score
A*	16.554899	49.9455498	0%(NA)	33.2502244
RRT	23.3	6.60	68.093%	14.90
ACO	25.545855	15.5671036	84.295%	20.5564793
DFO Pop 100 (Best)	21.0778	13.6998	52.061%	17.3888
DFO Pop 50 (Best)	20.979	7.934	51.348%	14.4565
DFO Pop 15 (Best)	**23.486**	**4.2368**	**69.435%**	**13.8614**

4.2.5 DFO Environment 3

Table 8: Environment 3 DFO Delta Value Comparison

DFO (Population 100)	Mean Length	Mean Time	Average Path Optimality	Shortest Path / Computation Time
DFO(d=0.0001)	**25.7074**	**59.9168**	**42.8121**	**Shortest Path**
DFO(d=0.0005)	**25.7322**	**60.7044**	**43.2183**	**Shortest Time**
DFO(d=0.001)	27.7974	60.8756	44.3365	
DFO(d=0.01)	29.7064	61.7392	45.7228	
DFO Decay {0.1 - 0.0001}	26.5394	65.81	46.1747	
DFO decay {0.01 - 0}	25.9764	60.7178	43.3471	
DFO decay (exponential)	26.8792	62.03	44.4546	
DFO (Population 50)				
DFO(d=0.0001)	**26.652**	**37.165**	**31.9085**	**Shortest Path**
DFO(d=0.0005)	28.1954	36.8232	32.5093	
DFO(d=0.001)	**29.0558**	**36.4438**	**32.7498**	**Shortest Time**
DFO(d=0.01)	30.6372	37.0956	33.8664	
DFO Decay {0.1 - 0.0001}	27.4578	42.3776	34.9177	
DFO decay {0.01 - 0}	27.6372	37.665	32.6511	
DFO decay (exponential)	27.1178	38.632	32.8749	
DFO (Population 15)				
DFO(d=0.0001)	**29.464**	**19.4712**	**24.4676**	**Shortest Time**
DFO(d=0.0005)	30.9064	19.803	25.3547	
DFO(d=0.001)	30.7962	19.756	25.2761	
DFO(d=0.01)	31.4402	19.6552	25.5477	
DFO Decay {0.1 - 0.0001}	**28.3136**	**20.5516**	**24.4326**	**Shortest Path**
DFO decay {0.01 - 0}	29.1366	19.9786	24.5576	
DFO decay (exponential)	28.9832	19.75125	24.267225	

4.2.6 Baseline Comparison for Environment 3

Table 9: DFO versus Baseline Environment 3

Algorithm	Average Path Length	Average Computation Time	Percentage Error from Global Optima	Average Optimality Score
A*	19.257554	1201.724419	0% (NA)	610.4909863
RRT	28.53096	35.39717	48.155%	31.964065
ACO	33.641045	58.5796239	74.690%	46.11033445
DFO Pop 100 (Best)	25.7074	59.9168	33.493%	42.8121
DFO Pop 50 (Best)	**26.652**	**37.165**	**38.398%**	**31.9085**
DFO Pop 15 (Best)	**28.3136**	**20.5516**	**47.03%**	**24.4326**

4.3 Path-Visualization

The paths generated and traversed by each experimented method in each environment are visualized as follows.

4.3.1 Environment 1 Visualization

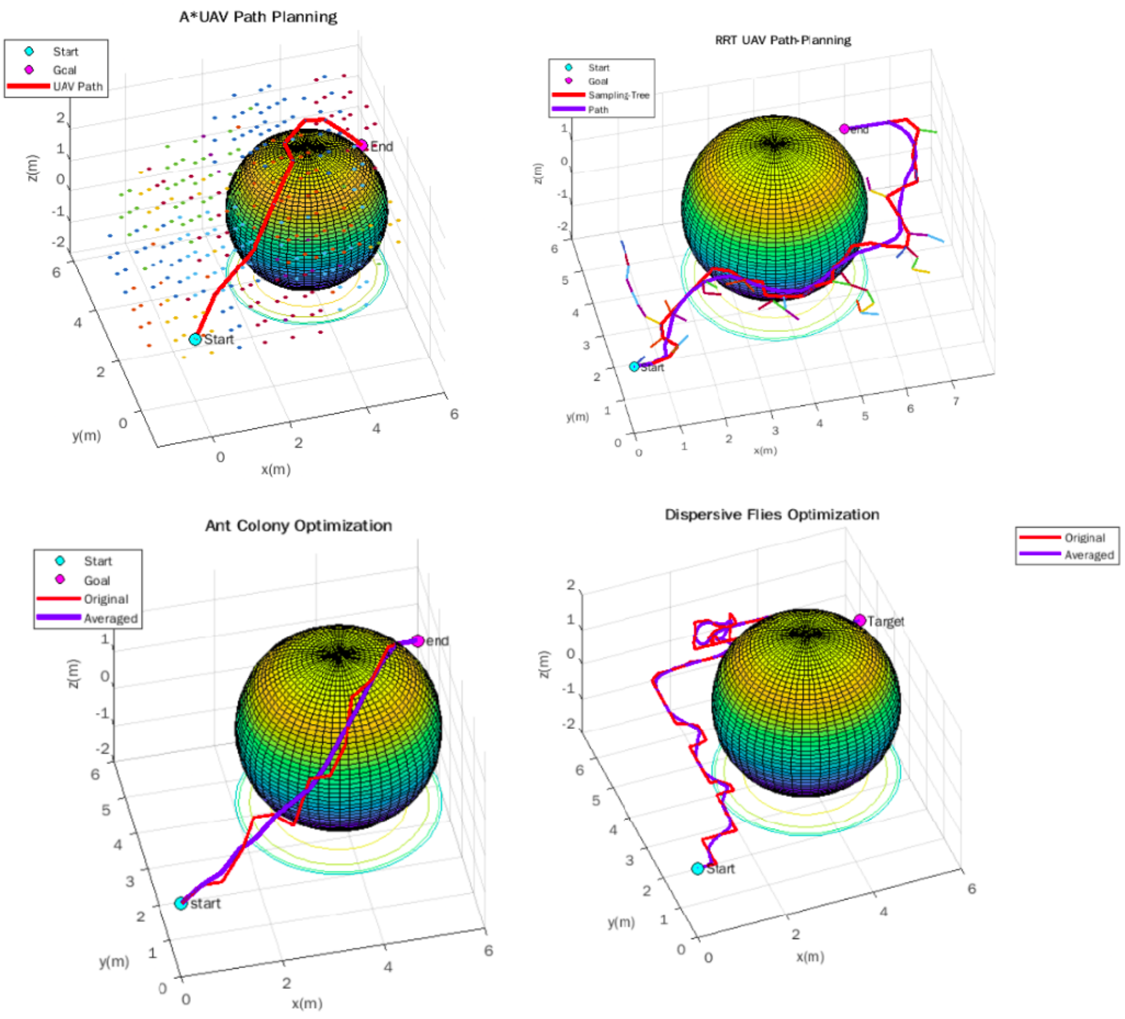


Figure 5: DFO Navigation in Environment 1

4.3.2 Environment 2 Visualization

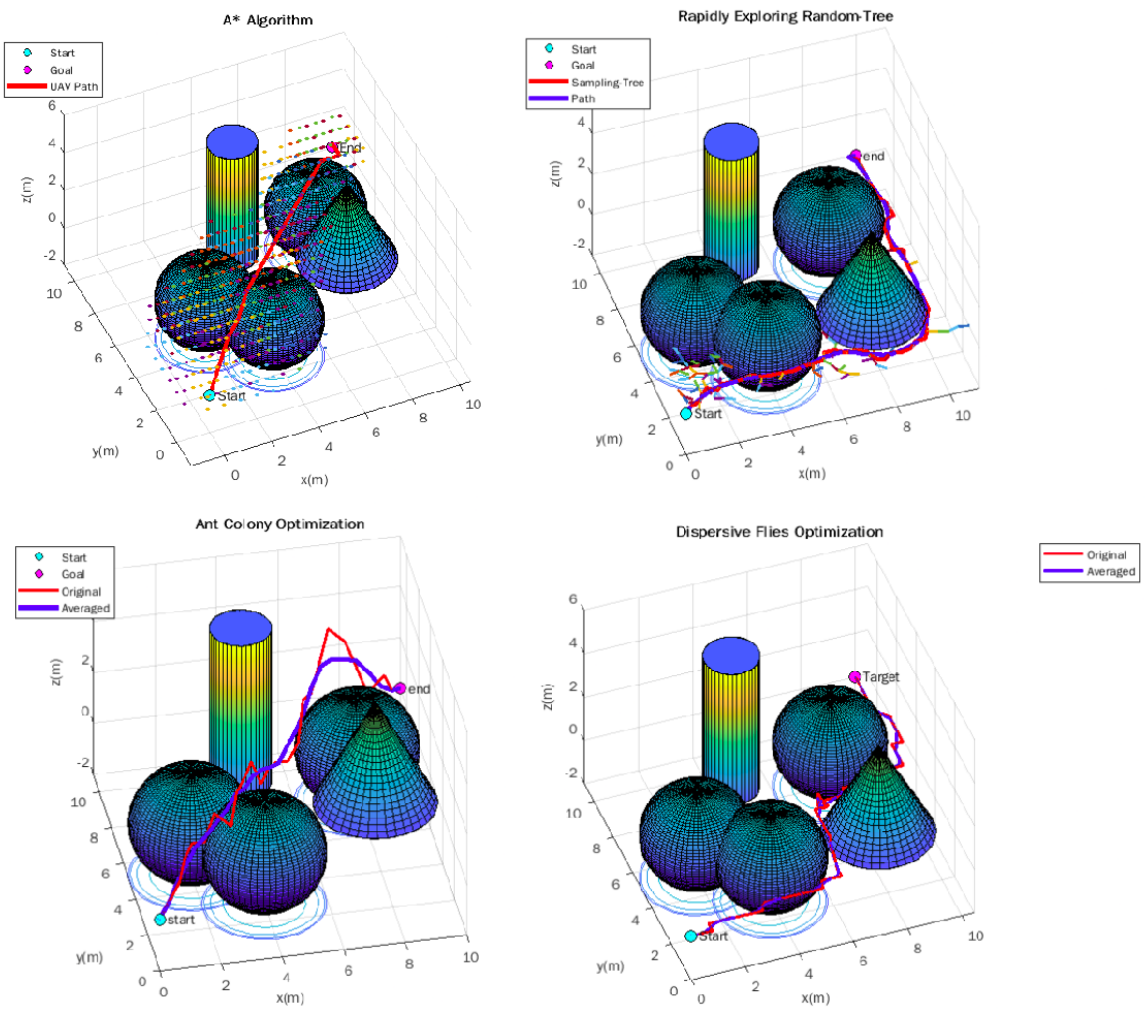


Figure 6: DFO Navigation in Environment 2

4.3.3 Environment 3 Visualization

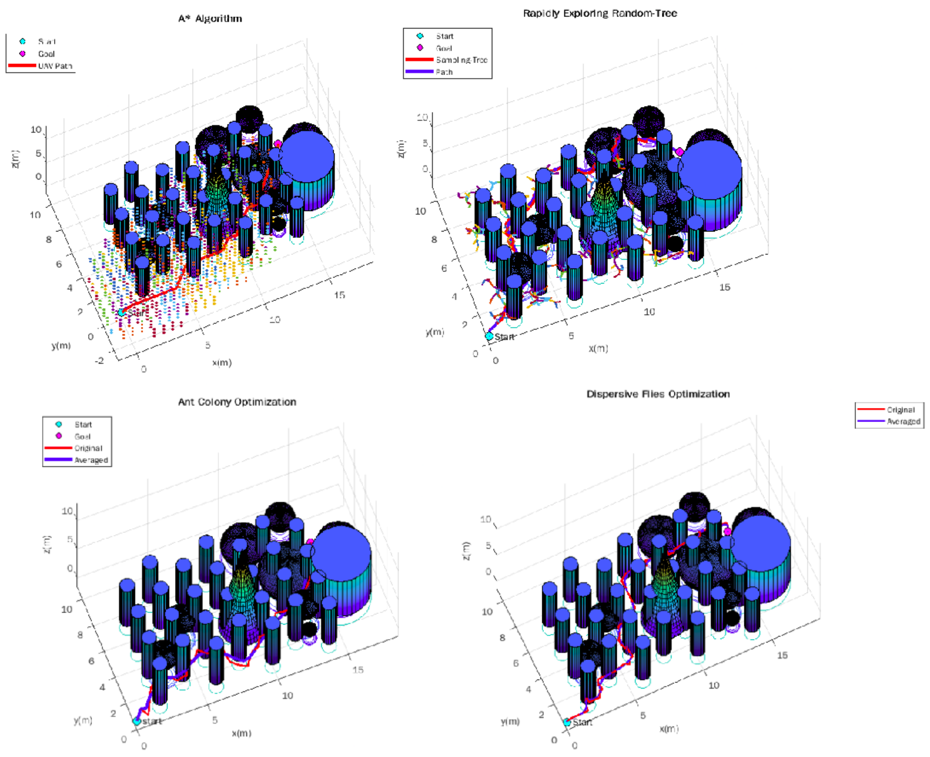


Figure 7: DFO Navigation in Environment 3

4.4 Convergence Curves

4.4.1 Environment 1

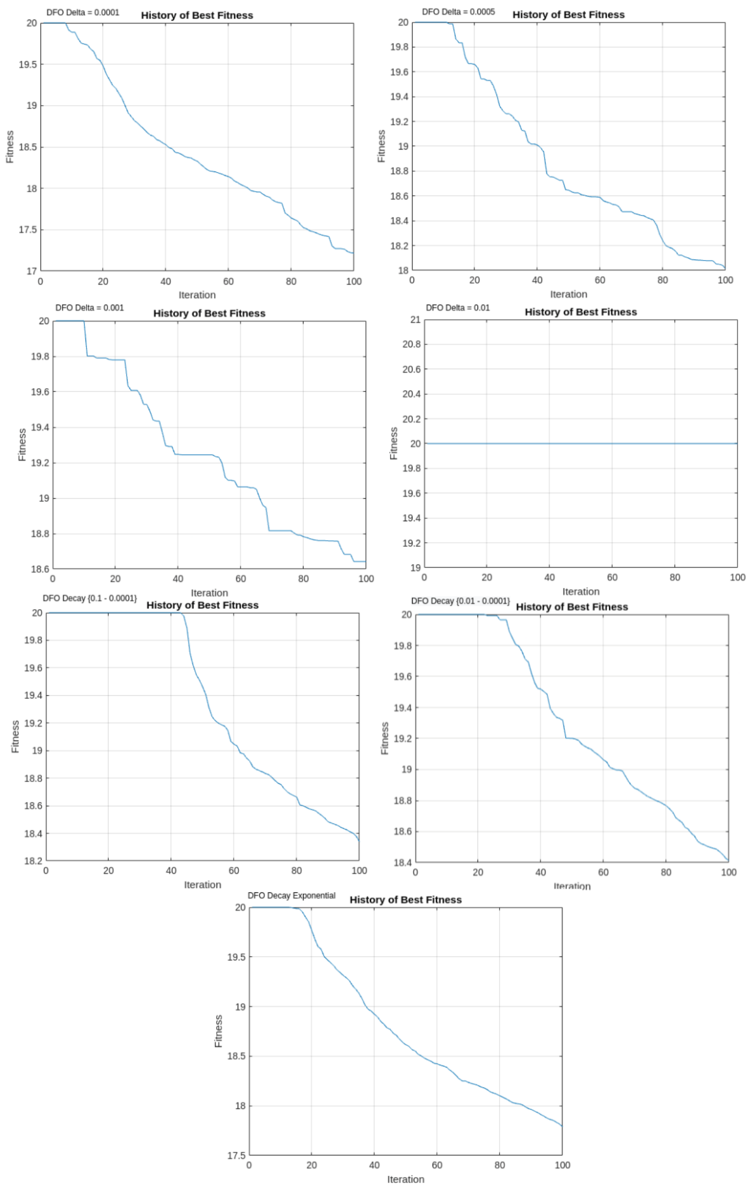


Figure 8: Convergence in Environment 1

4.4.2 Environment 2

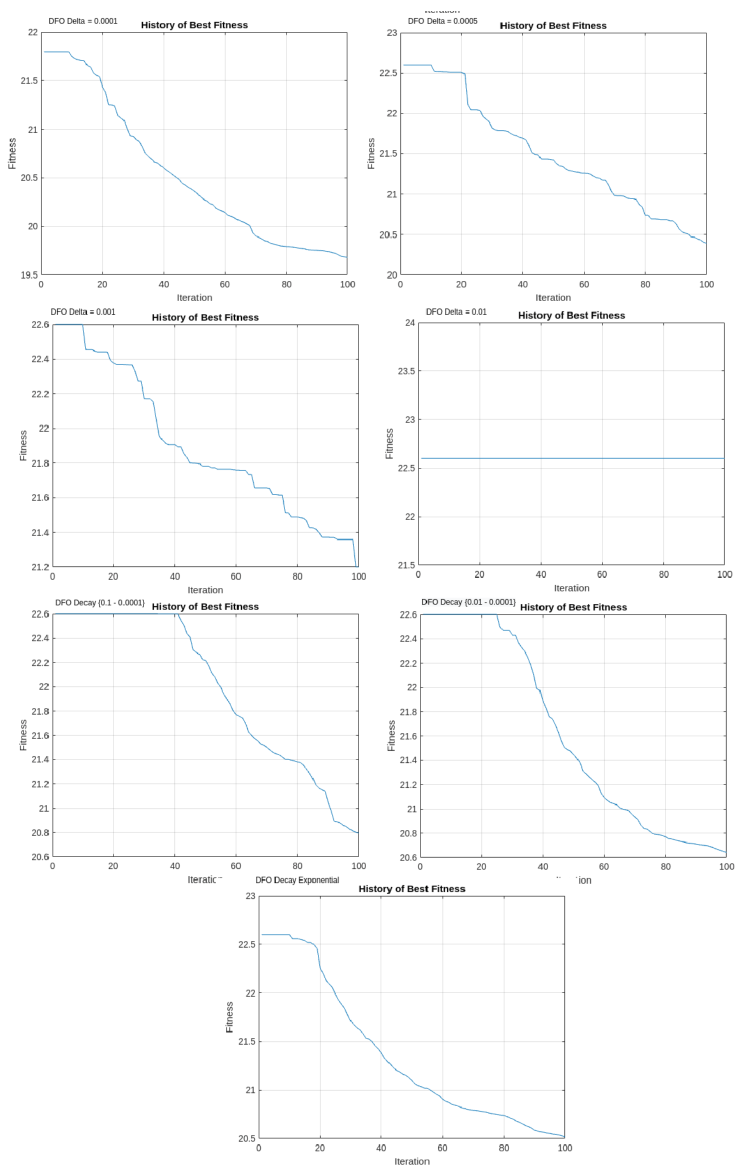


Figure 9: Convergence in Environment 2

4.4.3 Environment 3

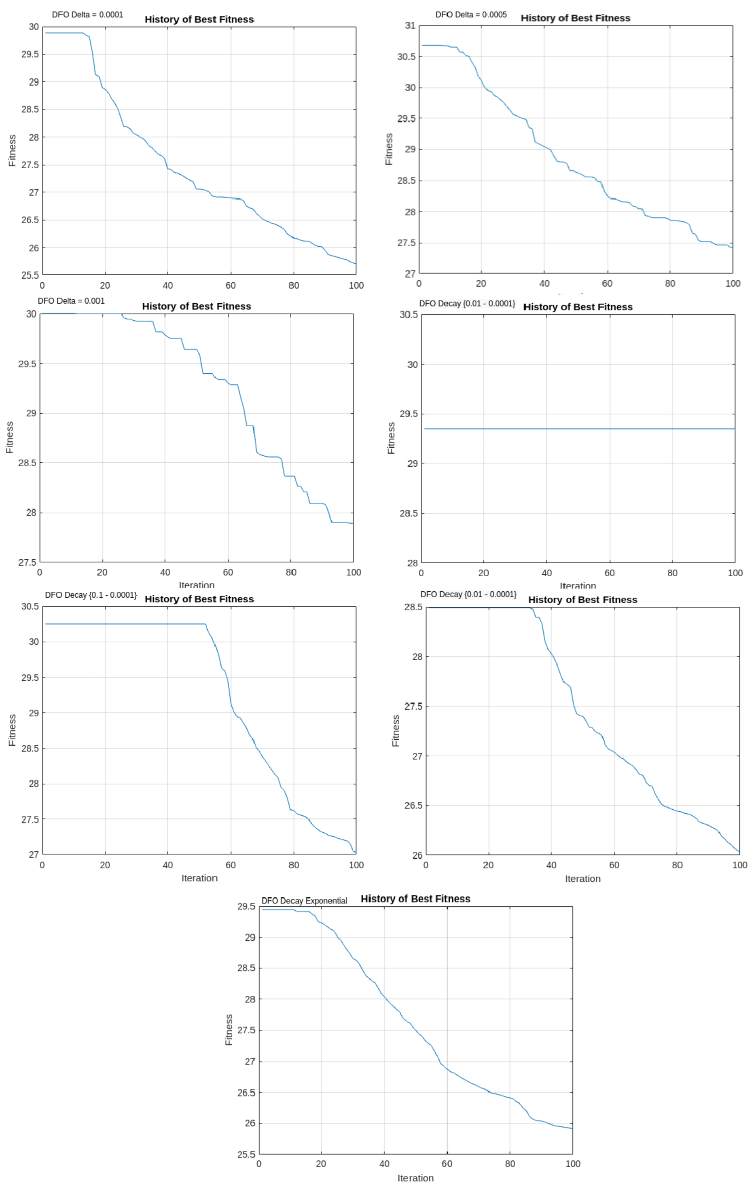


Figure 10: Convergence in Environment 3

5 Discussion

5.1 Effect of Population size on Path Length and Computation Time

5.1.1 Population size and Path Length

There is an inversely proportional relationship between the population size and path length where higher number of agents are able to search for shorter paths in the search space and vice versa. This is observed across all 3 environments where DFO with population size of 100 returns shorter paths on average when compared to DFO with population size of 50 and 15 respectively. This is due to more agents being dispersed across the search space thus being able to search for better solutions by which more agents share information on relative solution optimality where there are more flies per area. The effect is observed to be non-exponential as shown below.

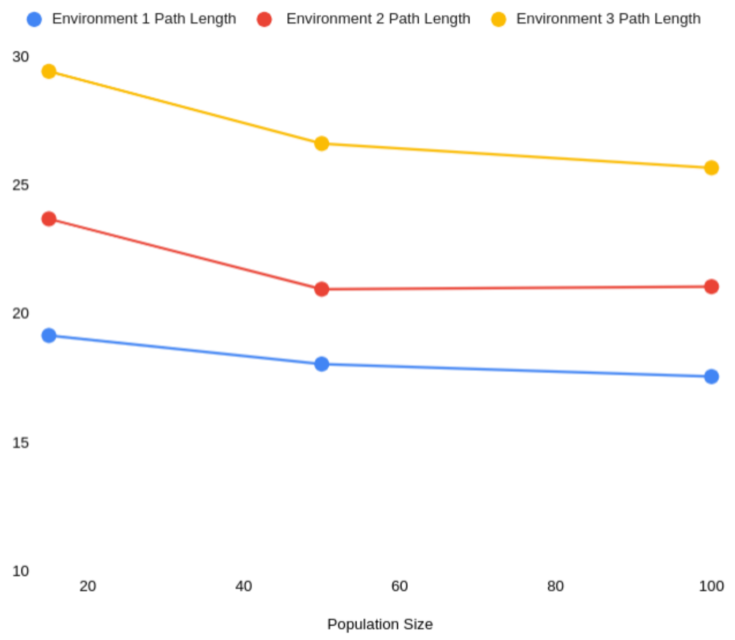


Figure 11: Population Size and Path Length Curve

5.1.2 Population size and Computation Time

There is a directly proportional relationship between population size and computation time where smaller number of agents are able to compute paths within significantly shorter computation time and vice versa. DFO with population size of 15 computes navigation paths in all environments in shorter time compared to DFO with population sizes of 50 and 100 respectively albeit the paths being slightly lengthier. This is because there are lesser number of agents, thereby less information sharing and position updating between neighbours which leads to faster convergence of all agents towards the best fly (agent) position. The effects is observed to be substantial, especially in the highly complex environment 3 as shown below.

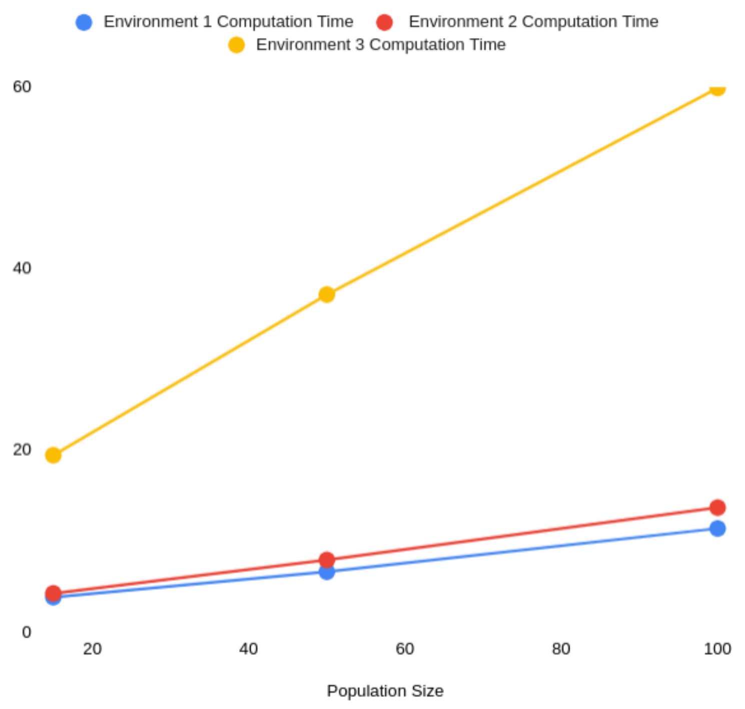


Figure 12: Population Size and Computation Time Curve

5.1.3 Optimal Trade-Off Between Path-Length and Computation-Time

Table 10: Population size Trade-Off effects

Comparison	**Path Length Trade-Off %**	**Computation Time Trade-Off %**	**Net-Optimality Trade-Off**
Population 50 vs 100	-3.544	61.218%	57.674%
Population 15 vs 50	-6.234%	57.57%	51.336%
Population 15 vs 100	-10.13%	193.274%	183.144%

The population size can be increased or decreased to favour between obtaining shorter path-lengths versus shorter computation-time. It is indicative from table 12 above which is taken from environment 3 as a sample that decreasing the population size has significantly smaller negative effect on path length compared to the larger gain in shortening the computation-time. Population sizes 50 and 15 offered well-rounded and balanced paths with shorter computation-time.

5.2 Optimal Delta Settings

The disturbance threshold, delta as aforementioned denotes the percentage of flies which are dispersed in each iteration as a measure of promoting diversity and stochasticity into the population by allowing the dispersed flies to search for better solutions in the search space. The convergence curve is plotted using population size of 100 as sample where same effect is observed across all population sizes.

5.2.1 Environment 1

In environment 1, the best delta settings for population sizes of 100, 50 and 15 is DFO with delta 0.0001 which obtained mean path lengths of 17.786, 18.0526 and 19.1736 which took 48.8069, 29.0532 and 16.0356 seconds respectively where the convergence (as shown in convergence curve) is observed to be the strongest. Higher disturbance settings, delta is observed to be counter-productive due to the smaller environment size and complexity which favour in-depth search. DFO with delta (0.0001) for all iterations where no convergence occurred for 100 iterations.

5.2.2 Environment 2

In environment 2 with medium obstacle complexity, the best delta settings for population sizes of 100 and 15 is DFO with delta 0.0001 which attained path lengths of 20.9769 and 23.1082 which took 13.7864 and 4.5072 seconds respectively whereas for population size of 50, DFO with decaying delta (0.1-0.0001) attained a slightly better path length of 22.2334 where the convergence is observed to be strongest during second half.

5.2.3 Environment 3

In environment 3 with high obstacle complexity and further goal position, the best solutions obtained for population sizes of 100 and 50 is DFO with delta 0.0001 which attained mean path lengths of 25.7074 and 26.652 which took mean time of 59.9168 and 37.165 seconds respectively. The effect of higher initialized delta value is apparent in population size of 15 where DFO with decaying delta (0.1 - 0.0001) obtained the shortest mean path length of 28.3136 which took mean time of 20.9516 where the convergence is observed to be strongest.

5.2.4 Analysis

It is observed that the optimal delta value is dependant on the size and complexity of environment whereby in smaller environments with less obstacle concentration, smaller delta values tuned for in-depth exploitation search is preferred whereas the effect of higher delta becomes apparent for environments 2 and 3 where DFO which is initialized with higher delta value during the first 40 percent of the iterations promotes diversity and thereby exploration of the search space which allowed DFO to obtain better solutions to converge towards during the last 60 percent of the iterations. Furthermore, it is indicative from convergence curves that DFO with Delta Decay and Delta value 0.0001 attained the strongest convergence rates whereas highly disturbed variants such as delta 0.001 and 0.1 attained the weakest convergence.

5.3 Baseline Comparison

The best-case i.e. optimal delta settings were selected across the 3 population sizes. For the purpose of performing objective comparison, the average of mean length, mean computation time and percentage error from global minima as computed by A* algorithm are compared.

5.3.1 Environment 1

In environment 1, DFO under-performed in computing comparably optimal paths where the paths computed by DFO of population sizes 100, 50 and 15 had the largest percentage error of 68.194, 73.51 and 83.870 percent whereas ACO obtained the shortest average path-lengths with only 28.189 percent deviation from the optima computed by A* algorithm. Conversely, DFO with population size 15 showcased the shortest average computation-time of only 3.552 seconds followed by RRT which takes average of 3.8 seconds whereas ACO showcased the longest average computation-time of 22.32 seconds. Albeit attaining the best average optimality score, it is noted that the path generated by DFO with population size 15 is almost twice as long as the global-optima computed by A* algorithm.

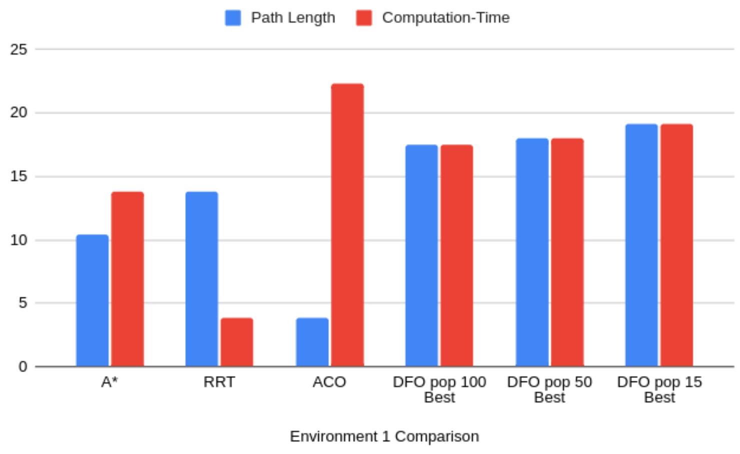


Figure 13: Path Optimality in Environment 1

5.3.2 Environment 2

The paths computed by DFO of population 50 and 100 are the closest to the optimal where the percentage errors are 51.35 and 52.06 percent respectively whereas the computation time of DFO is 72.57, 84.11 and 91.52 percent shorter compared to A* algorithm.

RRT produces the third shortest average path with 68.093 percent difference within 6.60 seconds. DFO with population size 15 has the shortest computation time of 4.24 seconds which is 55.66 percent shorter than RRT whilst the average path length is only 0.792

percent longer than RRT.

Overall, DFO with population size of 15 (exponential delta decay) obtained the best optimality score owing to its low computation-time and comparably shorter path-length. As aforementioned, trade-off between DFO (and ACO) can be performed where different population sizes can affect the results differently.

In environment 2, DFO showcased out-performance in both path-length and computation-time. DFO with population sizes 50 and 100 is able to obtain the closest solutions to the global-optimal paths computed by A* algorithm. It is observed that the computation time of A* algorithm is the longest thus obtaining the lowest optimality score

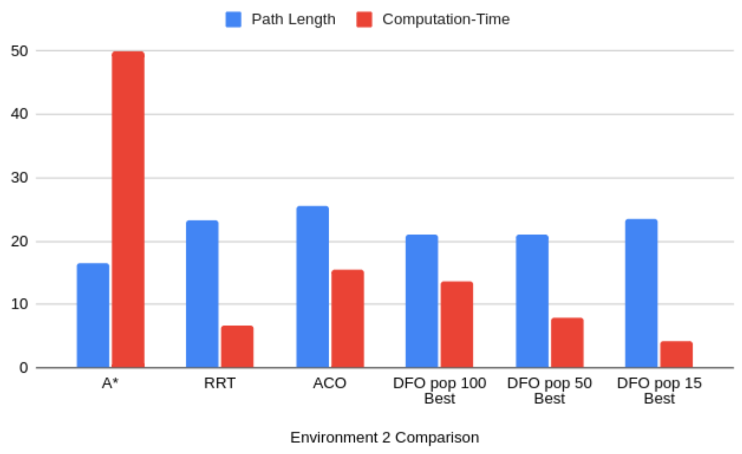


Figure 14: Path Optimality in Environment 2

5.3.3 Environment 3

In environment 3, the strong and fast convergence capability of DFO as mentioned in literature findings are apparent. As expected, A* algorithm indeed produces the shortest average path which is considered the global-optima, however due to the highly complex environment, the computation-time is observed to have grown exponentially where the average computation-time is 1201.72 seconds or 20.03 minutes to compute the path which renders this approach highly infeasible for real-time and real-life navigation requirements. DFO of population size 15 (delta decay 0.1 - 0.0001) outperformed all other methods where the computation time obtained was 193.274 percent shorter than A* algorithm with only 47.03 percent error, thereby having the best average optimality score of 24.4326. This is followed by DFO of population 50 (delta 0.0001) which obtained 99.68 percent shorter computation time with only 38.398 percent deviation from A* algorithm.

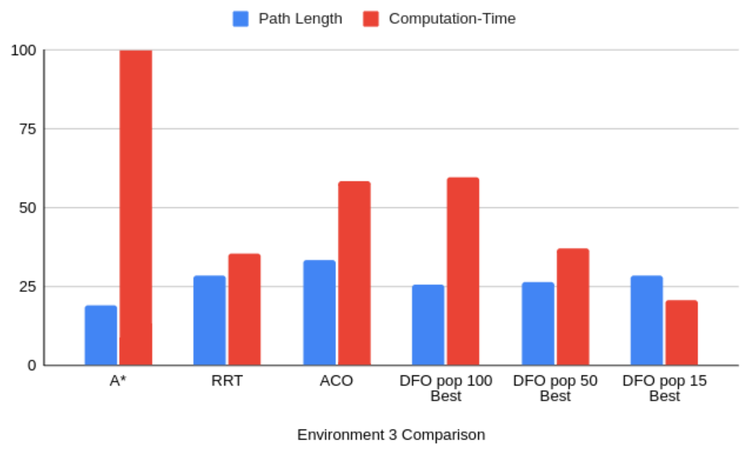


Figure 15: Path Optimality in Environment 3

5.3.4 Analysis

Overall, it is observed that the baseline methods outperformed the proposed DFO in simple environment where A* algorithm obtained the optimal path within shorter computation time whereas RRT obtained the second best path length in the fastest computation time. However as the environment size and complexity increases, it is observed that proposed DFO with optimal hyper-parameter setting for environments 2 and 3 is able to offer flexible and the most optimal trade off between path length and computation time, thus acquiring the highest average path-optimality score where a minor path length trade off results in significantly faster computation time as discussed above. A* algorithm is seen to suffer from extremely high computational cost as the environment complexities increase, especially when observed in environment 3 where the computation time grew exponentially. The trade-off offered by RRT is observed to offset and thereby reduce the computation time but returns sub-optimal and random paths, thereby having the second best average path-optimality score behind DFO. ACO is observed to suffer from pre-mature convergence and is stuck at a local minima where the solutions dont improve with the iterations in environment 2 and 3 respectively.

DFO addresses the shortcomings of A* and RRT by offering much more flexible trade-off options between path optimality and speed via population size and number of iterations. Furthermore, due to the random expansion nature of RRT trees, it is observed during the experiments that there are some outliers in data points as there is a big difference observed between the minimum and maximum path lengths and computation time. DFO returns paths that are not heavily deviant from another when run subsequently. Lastly, DFO addresses the problem of premature convergence faced by ACO where the path lengths obtained are improve iteratively and converge closest to the global-optima as defined by A* algorithm.
Table below presents the general comparison summary for the proposed methodologies in comparison with the baseline methods according to experimental findings.

Table 11: Comparison of Proposed Methodology to Baseline Methods

Metric	**Exact Methods (A*)**	**Sampling-Based (RRT)**	**ACO**	**Proposed Methodology**
Path Optimality	Optimal (Global Optima)	Sub-Optimal	Sub-Optimal	Closest to Global Optima
Obstacle Avoidance	Present	Present	Present	Present
Safety Margin	Absent	Absent	Absent	Present
Best Performing Environment	Small	Small and Medium	Small and Medium	Medium and Complex
Local Minima Vulnerability	Absent	Present	Present	Offset via optimal population, iteration and delta setting.
Convergence	Slow	Fast	Fast	Fastest
Time Efficiency	Low	High	Medium	Best
Computational Overload	Present	Occasional	Low	Lowest
Trade-Off Flexibility	NA	NA	Flexible	Very Flexible

6 Conclusion

The study presented a novel methodology towards 3-Dimensional autonomous UAV path-planning in simple, medium and complex environments. The proposed research objectives for minimization of path-cost and computation time are achieved via implementing Dispersive Flies Optimization whereas the objective of path-safety is achieved via implementation of penalty-based collision avoidance mechanism which is integrated into the overall fitness function. In addition, the objectives of simulation and application are overall achieved

DFO offers the most optimal trade-off between path-cost and computation time where the path lengths obtained in medium and complex environments are the closest to A* whilst having the fastest computation time when compared to other baseline methods such as RRT and ACO thereby serving as a proof-of-concept for the applicability of DFO as an new and alternative method for real-time navigation in complex cluttered environments such as low-altitude navigation and indoor navigation where the environment is obstacle rich and widely-used methods such as A* algorithm suffers from extremely high computational complexity. Furthermore, the proposed methodology, typical of all population-based swarm optimizers can be adjusted to perform suitable trade-off between path-cost and computation time albeit discretion required when adjusting according to environmental requirements.

Despite the observed out-performance of the proposed methodologies in this experiment, it is not possible to conclude that DFO is indeed better than the baseline methods as the experiments are only conducted in limited number of environmental settings with limited performance metrics. This is in accordance with the no-free lunch theorem where the path length and computational cost when averaged across all problems, hyper-parameter settings and environmental settings will be similar for all algorithms.

7 Limitations and Future Work

It is necessary to perform more benchmark evaluations to further strengthen the validity of the proposed methodology with differing environmental sizes and complexity such as conducting benchmark evaluation PathBench(Toma et al. 2021) which is an open source platform for developing and benchmarking the performance of classical and modern path-planning algorithms across various metrics in 2-dimensional and 3-dimensional environments.

The experiments conducted using static obstacles do not fully utilize the capability of DFO which is highly applicable in dynamically changing environments thus future works must consider navigation in dynamic environments.

Further extension can include hybridization of DFO with A* as a measure of increasing path-optimality whilst concurrently reducing computation time. This can be achieved via dispersion and convergence of the agents which act as nodes in the environment where A* algorithm will be used to compute the paths between the nodes to connect between start position through the nodes to the goal position.

A Terminologies

Table 12: Common Terms Used in Report

Term	Definition
Path-Optimality	Combination of measure of path cost, safety and computation-time.
Optimal-Path	Best path in terms of length.
Computation Time	Time taken to generate the paths.
Environmental Complexity	Measure of obstacle number and density in environment.
Exact Methods	Algorithms that return the exact optimal paths everytime.
Sampling-Based Methods	Algorithms that sample the environment prior to planing a path.
Global-Optima/Minima	The best existing path in the environment that all solutions aim to converge towards.
Local-Optima/Minima	Best solution in a local region where there are other better solutions in other regions of the same environment.
Local-Optima/Minima Trap	Situation where an algorithm mistakenly converges towards a local best solution with assumption this is the best solution.

B Path-Planning Process Flowchart

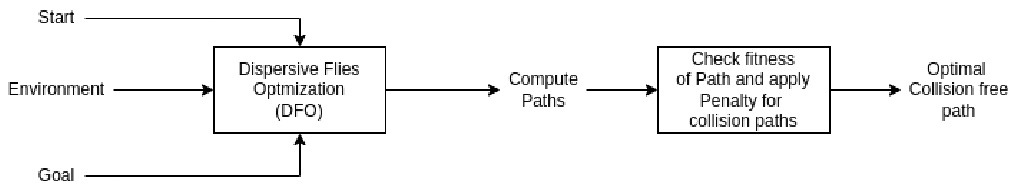

Start
Environment
Goal
Dispersive Flies Optmization (DFO)
Compute Paths
Check fitness of Path and apply Penalty for collision paths
Optimal Collision free path

Figure 16: Environment and Algorithm Flow Chart

C DFO Trial an Error Optimal Delta Parameter For All Environments

Table 13: DFO Delta Environment 1

	DFO 0.0001 pop 100		DFO 0.0005 pop 100		DFO 0.001 pop 100		DFO 0.01 pop 100	
	17.805	11.672	17.854	11.404	19.176	11.064	20	11.116
	17.527	11.418	17.621	11.064	18.85	11.194	20	11.152
	17.388	11.127	17.779	11.127	18.6	10.901	20	10.937
	17.34	11.124	17.592	11.043	19.297	10.95	20	10.934
			17.545	10.81			20	10.908
Mean	17.515	11.33525	17.6782	11.0896	18.98075	11.02725	20	11.0094
Min	17.34	11.124	17.545	10.81	18.6	10.901	20	10.908
Max	17.805	11.672	17.854	11.404	19.297	11.194	20	11.152
Std Dev	0.2089641118	0.2634658422	0.1318245046	0.2128668598	0.3163346066	0.1304642352	0	0.1150078258

	DFO 0.0001 pop 50		DFO 0.0005 pop 50		DFO 0.001 pop 50		DFO 0.01 pop 50	
	17.836	6.679	18.319	6.649	18.963	6.763	20	7.31
	18.302	6.574	18.553	6.844	19.089	6.447	20	6.372
	17.774	6.575	18.312	6.497	19.384	6.522	20	6.468
	18.288	6.859	18.229	7.204	18.981	7.263	20	7.133
	18.107	6.521	18.601	6.646	18.67	7.141	20	7.219
Mean	18.0614	6.6416	18.4028	6.768	19.0174	6.8272	20	6.9004
Min	17.774	6.521	18.229	6.497	18.67	6.447	20	6.372
Max	18.302	6.859	18.601	7.204	19.384	7.263	20	7.31
Std Dev	0.2473455073	0.1343607085	0.1637962149	0.2731107834	0.2571445119	0.364078563	0	0.4442851562

	DFO 0.0001 pop 15		DFO 0.0005 pop 15		DFO 0.001 pop 15		DFO 0.01 pop 15	
	18.802	3.897	18.717	3.977	22.121	4.047	22.4	3.665
	19.21	3.794	19.382	3.858	20.909	3.811	20	3.806
	18.943	3.834	20.36	3.828	21.158	3.884	23.2	3.831
	19.206	3.862	19.719	3.878	20.288	3.845	20	3.51
	19.71	3.815	20.72	3.823	19.374	3.561	20	3.554
Mean	19.1742	3.8404	19.7796	3.8728	20.77	3.8296	21.12	3.6732
Min	18.802	3.794	18.717	3.823	19.374	3.561	20	3.51
Max	19.71	3.8404	19.7796	3.8728	20.77	3.8296	21.12	3.6732
Std Dev	0.3469008504	0.04033980664	0.7924792111	0.06243156253	1.021788383	0.1753219895	1.559487095	0.1444357989

Table 14: DFO Delta Environment 2

	DFO 0.0001 pop 100		DFO 0.0005 pop 100		DFO 0.001 pop 100		DFO 0.01 pop 100	
	20.343	13.693	21.332	14.401	22.835	13.61	24.2	13.607
	21.333	13.459	21.124	13.433	21.765	14.016	23.4	13.492
	20.312	13.368	22.614	13.427	21.949	13.502	23.647	13.487
	22.222	13.901	20.976	13.331	22.645	13.572	24.2	13.433
	21.179	14.078	21.783	13.513	22.406	13.414	24.2	13.494
Mean	21.0778	13.6998	21.5658	13.621	22.32	13.6228	23.9294	13.5026
Min	20.312	13.368	20.976	13.331	21.765	13.414	23.4	13.433
Max	22.222	14.078	22.614	14.401	22.835	14.016	24.2	13.607
Std Dev	0.7922977344	0.2964956998	0.660229657	0.4407788561	0.4538479922	0.2321318591	0.3806859598	0.06358694835

	DFO 0.0001 pop 50		DFO 0.0005 pop 50		DFO 0.001 pop 50		DFO 0.01 pop 50	
	22.042	22.487	22.325	8.175	22.922	8.038	24.2	8.229
	21.842	7.999	22.98	7.978	23.461	7.97	22.496	8.038
	21.859	7.968	23.022	8.027	21.907	7.981	24.447	8.079
	21.775	7.958	22.274	8.036	23.336	8.054	23.647	8.119
	20.979	7.934	22.487	8.054	22.52	7.941	23.4	7.945
Mean	21.6994	10.8692	22.6176	8.054	22.8292	7.9968	23.638	8.082
Min	20.979	7.934	22.274	7.978	21.907	7.941	22.496	7.945
Max	22.042	22.487	23.022	8.175	23.461	8.054	24.447	8.229
Std Dev	0.4146906076	6.494589494	0.3590282719	0.07326322406	0.6345641812	0.04756784628	0.7634156797	0.1045131571

	DFO 0.0001 pop 15		DFO 0.0005 pop 15		DFO 0.001 pop 15		DFO 0.01 pop 15	
	25.077	4.365	24.845	4.28	23.846	4.316	25.8	4.345
	23.745	4.162	23.728	4.231	22.213	4.29	25	4.226
	21.784	4.365	23.459	4.195	23.691	4.233	24.2	4.173
	24.69	4.212	23.019	4.252	24.593	4.206	23.4	4.21
	23.296	4.195	22.379	4.226	24.601	4.204	25.247	4.186
Mean	23.7184	4.2598	23.486	4.2368	23.7888	4.2498	24.7294	4.228
Min	21.784	4.162	22.379	4.195	22.213	4.204	23.4	4.173
Max	25.077	4.365	24.845	4.28	24.601	4.316	25.8	4.345
Std Dev	1.295263796	0.09770209824	0.9151109222	0.03160221511	0.9749857435	0.05073657458	0.940064785	0.06856748501

Table 15: DFO Delta Environment 3

	DFO 0.0001 pop 100		DFO 0.0005 pop 100		DFO 0.001 pop 100		DFO 0.01 pop 100	
	25.612	60.308	25.821	59.735	27.756	59.278	29.346	61.349
	24.693	58.077	27.774	61.59	28.14	60.024	29.432	61.231
	25.601	59.951	24.492	61.472	27.337	64.186	28.632	61.074
3421	26.349	59.332	24.905	60.243	28.743	59.184	30.232	63.182
	26.282	61.916	25.669	60.482	27.011	61.706	30.89	61.86
Mean	25.7074	59.9168	25.7322	60.7044	27.7974	60.8756	29.7064	61.7392
Min	24.693	58.077	24.492	59.735	27.011	59.184	28.632	61.074
Max	26.349	61.916	27.774	61.59	28.743	64.186	30.89	63.182
Std Dev	0.6691870441	1.403376535	1.265225553	0.8024314924	0.6789037487	2.10875385	0.8713040801	0.8586732207

	DFO 0.0001 pop 50		DFO 0.0005 pop 50		DFO 0.001 pop 50		DFO 0.01 pop 50	
	27.397	37.031	28.325	37.023	28.34	37.095	30.232	37.039
	26.873	37.054	27.997	36.37	29.19	36.275	30.232	36.742
	27.483	37.28	28.255	37.514	29.225	35.577	30.232	37.169
	26.404	36.722	28.618	35.931	29.127	35.651	30.8	37.351
	25.103	37.738	27.782	37.278	29.397	37.621	31.69	37.177
Mean	26.652	37.165	28.1954	36.8232	29.0558	36.4438	30.6372	37.0956
Min	25.103	36.722	27.782	35.931	28.34	35.577	30.232	36.742
Max	27.483	37.738	28.618	37.514	29.397	37.621	31.69	37.351
Std Dev	0.9689339503	0.3768554099	0.3198441808	0.6567683762	0.4124605436	0.8969711255	0.6378582915	0.2266490679

	DFO 0.0001 pop 15		DFO 0.0005 pop 15		DFO 0.001 pop 15		DFO 0.01 pop 15	
	28.261	19.469	32.115	19.929	30.244	19.336	30.8	19.887
	29.778	19.495	32.007	20.069	30.063	19.699	32.49	19.697
	27.388	19.495	32.321	19.695	29.279	19.648	29.447	19.259
	32.204	19.268	27.305	19.736	32.743	20.343	31.032	19.168
	29.689	19.629	30.784	19.586	31.652	19.754	33.432	20.265
Mean	29.464	19.4712	30.9064	19.803	30.7962	19.756	31.4402	19.6552
Min	27.388	19.268	27.305	19.586	29.279	19.336	29.447	19.168
Max	32.204	19.629	32.321	20.069	32.743	20.343	33.432	20.265
Std Dev	1.831243157	0.1297389687	2.101082531	0.1935678176	1.384451769	0.3660211743	1.550699584	0.4532308463

Table 16: Modified DFO Decay Trial and Error Selection Environment 1

	DFO Decay 0.1 - 0.0001		DFO Decay 0.01 - 0.001 - 0.0001		DFO Decay Exponential 0.1 - exponential	
	17.953	11.028	18.411	11.434	17.462	11.836
	18.012	10.908	18.278	11.142	17.978	11.327
	18.082	11.059	17.945	11.026	17.57	11.036
	18.213	10.943	18.078	11.021	17.329	11.348
	18.081	10.994	18.543	11.734	17.202	10.996
Mean	18.0682	10.9864	18.251	11.2714	17.5082	11.3086
Min	17.953	10.908	17.945	11.021	17.202	10.996
Max	18.213	11.059	18.543	11.734	17.978	11.836
Std Dev	0.0971478255	0.06141905242	0.2424654615	0.3082625504	0.2968740474	0.3361812011

	DFO Decay 0.1 - 0.0001		DFO Decay 0.01 - 0.001 - 0.0001		DFO Decay Exponential 0.1 - exponential	
	18.392	6.832	18.603	7.183	18.188	6.694
	18.263	6.649	18.66	6.527	18.322	6.652
	18.727	6.804	18.552	6.493	18.199	6.676
	18.658	6.45	19.05	6.554	18.259	6.652
	18.466	6.565	18.868	6.513	18.399	6.58
Mean	18.5012	6.66	18.7466	6.654	18.2734	6.6508
Min	18.263	6.45	18.552	6.493	18.188	6.58
Max	18.727	6.832	19.05	7.183	18.399	6.694
Std Dev	0.1907057944	0.160908359	0.2078335873	0.2965518504	0.08827966923	0.04334974048

	DFO Decay 0.1 - 0.0001		DFO Decay 0.01 - 0.001 - 0.0001		DFO Decay Exponential 0.1 - exponential	
	19.583	3.659	19.672	3.936	18.907	3.715
	19.661	3.432	19.41	3.47	19.521	3.515
	19.392	3.411	19.361	3.699	19.115	3.617
	19.091	3.428	19.6	3.458	18.916	3.457
	19.658	3.454	19.523	3.483	19.241	3.455
Mean	19.477	3.4768	19.5132	3.6092	19.14	3.5518
Min	19.091	3.411	19.361	3.458	18.907	3.455
Max	19.477	3.4768	19.5132	3.6092	19.14	3.5518
Std Dev	0.2419059735	0.1029985437	0.1290957009	0.2079824512	0.2551724907	0.1124686623

Table 17: DFO Decay trial and error tuning environment 2

	DFO Decay 0.1 - 0.0001		DFO Decay 0.01 - 0.001 - 0.0001		DFO Decay Exponential 0.1 - exponential	
	21.974	13.598	20.982	13.396	21.109	13.278
	21.411	13.236	21.666	13.447	21.361	13.088
	22.217	13.243	22.214	13.448	21.739	13.22
	21.893	13.469	22.713	13.328	21.493	13.108
	21.248	13.385	21.7	13.198	21.683	13.139
Mean	21.7486	13.3862	21.855	13.3634	21.477	13.1666
Min	21.248	13.236	20.982	13.198	21.109	13.088
Max	22.217	13.598	22.713	13.448	21.739	13.278
Std Dev	0.4048546653	0.1539340768	0.6494074222	0.1046651805	0.2550568564	0.08005498111

	DFO Decay 0.1 - 0.0001		DFO Decay 0.01 - 0.001 - 0.0001		DFO Decay Exponential 0.1 - exponential	
	22.635	7.947	22.984	8.059	23.041	8.316
	23.173	8.03	22.839	7.997	23.886	8.123
	22.946	8.029	23.449	7.98	21.974	8.136
	22.654	7.989	22.667	8.11	21.38	8.155
	22.131	7.952	22.269	8.089	21.566	8.143
Mean	22.7078	7.9894	22.8416	8.047	22.3694	8.1746
Min	22.131	7.947	22.269	7.98	21.38	8.123
Max	23.173	8.03	23.449	8.11	23.886	8.316
Std Dev	0.3918044155	0.04004122875	0.4323121557	0.05671419575	1.064404434	0.07988929841

	DFO Decay 0.1 - 0.0001		DFO Decay 0.01 - 0.001 - 0.0001		DFO Decay Exponential 0.1 - exponential	
	22.61	4.424	22.948	4.407	24.366	4.355
	23.858	4.399	23.789	4.311	23.674	4.305
	23.444	4.254	22.091	4.391	23.805	4.285
	22.81	4.428	23.634	4.277	25.028	4.237
	24.933	4.32	24.505	4.301	23.118	4.316
Mean	23.531	4.365	23.3934	4.3374	23.9982	4.2996
Min	22.61	4.254	22.091	4.277	23.118	4.237
Max	24.933	4.428	24.505	4.407	25.028	4.355
Std Dev	0.9284643235	0.07574958746	0.9144218392	0.05785153412	0.7267992845	0.04329896073

Table 18: DFO Decay trial and error tuning environment 3

	DFO Decay 0.1 - 0.0001		DFO Decay 0.01 - 0.001 - 0.0001		DFO Decay Exponential 0.1 - exponential	
	25.033	66.211	26.688	61.125	27.75	62.241
	27.357	66.506	26.686	61.065	26.84	61.718
	26.587	65.489	24.497	60.249	27.043	61.339
	27.51	65.485	25.375	60.883	26.236	61.88
	26.21	65.359	26.636	60.267	26.527	62.972
Mean	26.5394	65.81	25.9764	60.7178	26.8792	62.03
Min	25.033	65.359	24.497	60.249	26.236	61.339
Max	27.51	66.506	26.688	61.125	27.75	62.972
Std Dev	0.9988174508	0.514121581	0.9994104762	0.4291400704	0.5754152414	0.6183061539

	DFO Decay 0.1 - 0.0001		DFO Decay 0.01 - 0.001 - 0.0001		DFO Decay Exponential 0.1 - exponential	
	27.682	42.296	28.179	39.361	27.57	38.744
	27.169	41.944	26.282	37.206	26.153	37.976
	27.876	42.66	27.988	37.657	27.764	40.622
	27.4	42.583	28.413	36.885	27.116	37.97
	27.162	42.405	27.324	37.216	26.986	37.848
Mean	27.4578	42.3776	27.6372	37.665	27.1178	38.632
Min	27.162	41.944	26.282	36.885	26.153	37.848
Max	27.876	42.66	28.413	39.361	27.764	40.622
Std Dev	0.3159908227	0.2816723274	0.8592343685	0.9870615482	0.6265526315	1.167891262

	DFO Decay 0.1 - 0.0001		DFO Decay 0.01 - 0.001 - 0.0001		DFO Decay Exponential 0.1 - exponential	
	28.523	20.071	30.229	20.027	29.728	19.878
	27.928	20.808	28.103	20.45	29.066	19.519
	29.251	20.595	29.164	19.97	27.715	19.700s
	27.87	20.816	28.729	19.634	30.374	20.033
	27.996	20.468	29.458	19.812	28.033	19.575
Mean	28.3136	20.5516	29.1366	19.9786	28.9832	19.75125
Min	27.87	20.071	28.103	19.634	27.715	19.519
Max	29.251	20.816	30.229	20.45	30.374	20.033
Std Dev	0.5850011111	0.3063891317	0.7955899069	0.304605975	1.118829612	0.2452568382

D Complete Experimental Data for Baseline Algorithms

Table 19: Baseline Comparison Environment 1

	A*		RRT		ACO pop 50		ACO pop 15	
	Path Length	Comp Time	Path Length	Comp Time	Path Length	Comp Time	Path Length	Comp Time
	10.40954	13.32134	1.46E+01	3.54E+00	11.868235	76.081766	12.923797	22.904599
	10.40954	14.834135	1.42E+01	3.57E+00	11.868235	76.354875	12.923797	20.390789
	10.40954	13.42227	1.25E+01	3.85E+00	11.868235	77.210371	12.923797	22.696306
	10.40954	13.650522	1.30E+01	4.29E+00	11.868235	75.636012	12.923797	22.693922
	10.40954	13.771091	1.29E+01	3.74E+00	11.868235	77.392038	12.923797	22.903358
mean	10.40954	13.7998716	13.447976	3.7983174	11.868235	76.5350124	12.923797	22.3177948
min	10.40954	13.32134	1.25E+01	3.54E+00	11.868235	75.636012	12.923797	20.390789
max	10.40954	14.834135	1.46E+01	4.29E+00	11.868235	77.392038	12.923797	22.904599
stdev	0	0.6050640187	0.8977213789	0.3037280887	0	0.7477776439	0	1.082279675

D.0.1 Environment 2

Table 20: Baseline Comparison Environment 2

	A*		RRT		ACO pop 50		ACO pop 15	
	Path Length	Comp Time	Path Length	Comp Time	Path Length	Comp Time	Path Length	Comp Time
	16.554899	57.722232	2.33E+01	6.65E+00	24.550454	52.088494	25.545855	15.620071
	16.554899	48.031005	2.49E+01	9.54E+00	24.550454	52.201557	25.545855	15.709526
	16.554899	47.84142	2.39E+01	7.19E+00	24.550454	50.868749	25.545855	15.459044
	16.554899	48.169226	2.24E+01	4.70E+00	24.550454	51.418152	25.545855	15.540268
	16.554899	47.963866	2.18E+01	4.91E+00	24.550454	50.732066	25.545855	15.506609
mean	16.554899	49.9455498	2.33E+01	6.60E+00	24.550454	51.4618036	25.545855	15.5671036
min	0	4.348908684	1.215972427	1.965672271	0	0.6756685829	0	0.09892713062
max	16.554899	47.84142	2.18E+01	4.70E+00	24.550454	50.732066	25.545855	15.459044
stdev	16.554899	57.722232	2.49E+01	9.54E+00	24.550454	52.201557	25.545855	15.709526

D.0.2 Environment 3

Table 21: Baseline Comparison Environment 3

	A*		RRT		ACO pop 50		ACO pop 15	
	Path Length	Comp Time	Path Length	Comp Time	Path Length	Comp Time	Path Length	Comp Time
	19.257554	1373.458868	24.04	29.42	34.053126	193.832556	33.641045	59.299924
	19.257554	1076.355162	24.28	21.09	34.053126	192.1252	33.641045	58.804679
	19.257554	1155.359226	33.75	43.59	34.053126	194.744851	33.641045	58.726847
	19.257554	1095.077265	34.76	40.30	34.053126	201.933974	33.641045	58.602115
			25.82	42.58	34.053126	193.267986	33.641045	52.157527
mean	19.257554	1201.724419	28.53096	35.39717	34.053126	195.1809134	33.641045	58.5796239
min	19.257554	1076.355162	24.04406	21.0922	0	3.892580159	0	3.008408339
max	19.257554	1373.458868	34.762	43.59377	34.053126	192.1252	33.641045	52.157527
stdev	0	153.8829144	5.281093752	9.788475137	34.053126	201.933974	33.641045	59.299924

References

(n.d.).

Ab Wahab, M. N., Nefti-Meziani, S. & Atyabi, A. (2015), 'A comprehensive review of swarm optimization algorithms', *PLOS ONE* **10**(5).

Abhishek, B., Ranjit, S., Shankar, T., Eappen, G., Sivasankar, P. & Rajesh, A. (2020), 'Hybrid pso-hsa and pso-ga algorithm for 3d path planning in autonomous uavs', *SN Applied Sciences* **2**(11).

Abhishek, B., Ranjit, S., Thangavelu, S., Eappen, G., Sivasankar, P. & A, R. (2020), 'Hybrid pso-hsa and pso-ga algorithm for 3d path planning in autonomous uavs', *SN Applied Sciences* **2**.

Aggarwal, S. & Kumar, N. (2020), Path planning techniques for unmanned aerial vehicles: A review, solutions, and challenges, *in* 'Computer Communications'.

Al-Ansarry, S. & Al-Darraji, S. (2021), 'Hybrid rrt-a*: An improved path planning method for an autonomous mobile robots', *Iraqi Journal for Electrical and Electronic Engineering* **17**, 1–9.

al Rifaie, M. (2014*a*), Dispersive flies optimisation: A tutorial.

Al-Rifaie, M. M. (2014*b*), Dispersive flies optimisation, pp. 529–538.

Al-Rifaie, M. M. (2015), 'Dispersive flies optimisation and medical imaging', *Studies in Computational Intelligence* .

Alhakbani, H. A. & al Rifaie, M. M. (2017), Optimising svm to classify imbalanced data using dispersive flies optimisation, *in* '2017 Federated Conference on Computer Science and Information Systems (FedCSIS)', IEEE, pp. 399–402.

Ashish, D., Munjal, S., Mani, M. & Srivastava, S. (2021), Path finding algorithms, *in* 'Emerging Technologies in Data Mining and Information Security', Springer, pp. 331–338.

Cekmez, U., Ozsiginan, M. & Sahingoz, O. (2016), Multi colony ant optimization for uav path planning with obstacle avoidance, pp. 47–52.

Chen, J., D. C. Z. Y. H. P. & Wei (2021), A clustering-based coverage path planning method for autonomous heterogeneous uavs, *in* 'IEEE Transactions on Intelligent Transportation Systems'.

Chen, Q., L. Y. W. Y. & Zhu (2021), From topological map to local cognitive map: a new opportunity of local path planning, *in* 'Intelligent Service Robotics'.

Châari, I., Koubâa, A., Trigui, S., Bennaceur, H., Ammar, A. & Al-Shalfan, K. (2014), 'Smartpath: An efficient hybrid aco-ga algorithm for solving the global path planning problem of mobile robots', *International Journal of Advanced Robotic Systems* **11**(7), 94.

Dai, Q., Ji, J. & Liu, C. (2009), An effective initialization strategy of pheromone for ant colony optimization, *in* '2009 Fourth International on Conference on Bio-Inspired Computing', pp. 1–4.

Deng-xu, H., Hua-Zheng, Z. & Gui-qing, L. (2011), Glowworm swarm optimization algorithm for solving multi-constrained qos multicast routing problem, *in* '2011 Seventh International Conference on Computational Intelligence and Security', pp. 66–70.

Dijkstra, E. W. (1959), 'A note on two problems in connexion with graphs', *Numerische mathematik* **1**(1), 269–271.

Dorigo, M. & Di Caro, G. (1999), Ant colony optimization: a new meta-heuristic, *in* 'Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)', Vol. 2, pp. 1470–1477 Vol. 2.

Du, K.-L., Swamy, M. et al. (2016), 'Search and optimization by metaheuristics', *Techniques and Algorithms Inspired by Nature* pp. 1–10.

Duchoň, F., Babinec, A., Kajan, M., Beňo, P., Florek, M., Fico, T. & Jurišica, L. (2014), 'Path planning with modified a star algorithm for a mobile robot', *Procedia Engineering* **96**, 59–69.

Elbanhawi, M., Simic, M. & Jazar, R. N. (2015), 'Continuous path smoothing for car-like robots using b-spline curves', *Journal of Intelligent & Robotic Systems* **80**(1), 23–56.

Fei, Z. Y. (2021), 'Uav$_obstacle_avoiding_drl'$, .

Fink, W., Baker, V., Brooks, A., Flammia, M., Dohm, J. & Tarbell, M. (2019), 'Globally optimal rover traverse planning in 3d using dijkstra's algorithm for multi-objective deployment scenarios', *Planetary and Space Science* **179**, 104707.

Foo, J. L., Knutzon, J., Oliver, J. & Winer, E. (2006), Three-dimensional path planning of unmanned aerial vehicles using particle swarm optimization, *in* '11th AIAA/ISSMO multidisciplinary analysis and optimization conference', p. 6995.

Gass, S. I. & Harris, C. M. (2001), 'Np, np-complete, np-hard', *Encyclopedia of Operations Research and Management Science* p. 581–581.

Giloi, W. K. (1997), 'Konrad zuse's plankalku/spl uml/l: the first high-level," non von neumann" programming language', *IEEE Annals of the History of Computing* **19**(2), 17–24.

Goel, U., Varshney, S., Jain, A., Maheshwari, S. & Shukla, A. (2018), 'Three dimensional path planning for uavs in dynamic environment using glow-worm swarm optimization', *Procedia computer science* **133**, 230–239.

Guo, J., Liang, C., Wang, K., Sang, B. & Wu, Y. (2021), 'Three-dimensional autonomous obstacle avoidance algorithm for uav based on circular arc trajectory', *International Journal of Aerospace Engineering* **2021**, 1–13.

Guruji, A. K., Agarwal, H. & Parsediya, D. (2016), 'Time-efficient a* algorithm for robot path planning', *Procedia Technology* **23**, 144–149.

Hart, P. E., Nilsson, N. J. & Raphael, B. (1968), 'A formal basis for the heuristic determination of minimum cost paths', *IEEE Transactions on Systems Science and Cybernetics* **4**(2), 100–107.

He, Y., Zeng, Q., Liu, J., Xu, G. & Deng, X. (2013), Path planning for indoor uav based on ant colony optimization, *in* '2013 25th Chinese Control and Decision Conference (CCDC)', pp. 2919–2923.

Holland, J. H. (1984), *Genetic Algorithms and Adaptation*, Springer US, Boston, MA.
URL: *https://doi.org/10.1007/978-1-4684-8941-5$_2$*1

Hooman, O. M., Al-Rifaie, M. M. & Nicolaou, M. A. (2018), Deep neuroevolution: Training deep neural networks for false alarm detection in intensive care units, in *'2018 26th European Signal Processing Conference (EUSIPCO)', IEEE, pp. 1157–1161.*

Hussain, K., Salleh, M. N. M., Cheng, S. & Shi, Y. (2019), 'On the exploration and exploitation in popular swarm-based metaheuristic algorithms', Neural Computing and Applications **31***(11), 7665–7683.*

Jung, S. & Kim, H. (2017), 'Analysis of amazon prime air uav delivery service', Journal of Knowledge Information Technology and Systems **12***(2), 253–266.*

Karur, K., Sharma, N., Dharmatti, C. & Siegel, J. E. (2021), 'A survey of path planning algorithms for mobile robots', Vehicles **3***(3), 448–468.*

Kavraki, L., Svestka, P., Latombe, J.-C. & Overmars, M. (1996), 'Probabilistic roadmaps for path planning in high-dimensional configuration spaces', IEEE Transactions on Robotics and Automation **12***(4), 566–580.*

Kennedy, J. & Eberhart, R. (1995), Particle swarm optimization, in *'Proceedings of ICNN'95 - International Conference on Neural Networks', Vol. 4, pp. 1942–1948 vol.4.*

Khan, M. A., Safi, A., Qureshi, I. M. & Khan, I. U. (2017), Flying ad-hoc networks (fanets): A review of communication architectures, and routing protocols, in *'2017 First International Conference on Latest trends in Electrical Engineering and Computing Technologies (INTELLECT)', pp. 1–9.*

Kiani, F., Seyyedabbasi, A., Aliyev, R. & Shah, M. (2021), '3d path planning method for multi-uavs inspired by grey wolf algorithms', Journal of Internet Technology **22**, *743–755.*

Koslowski, R. (2021), 'Drones and border control: An examination of state and non-state actor use of uavs along borders', Research Handbook on International Migration and Digital Technology *p. 152–165.*

Krishnanand, K. & Ghose, D. (2005), Detection of multiple source locations using a glowworm metaphor with applications to collective robotics, in *'Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.', pp. 84–91.*

Lajevardy, P., mousavian, A. & Oskoei, M. (2015), A comparison between rrt and a* algorithms for motion planning in complex environments.*

Lamini, C., Benhlima, S. & Elbekri, A. (2018), 'Genetic algorithm based approach for autonomous mobile robot path planning', Procedia Computer Science **127**, *180–189.*

LaValle, S. M. et al. (1998), 'Rapidly-exploring random trees: A new tool for path planning'.

Lee, C. Y. (1961), 'An algorithm for path connections and its applications', IRE Transactions on Electronic Computers **EC-10***(3), 346–365.*

Li, M., Du, W. & Nian, F. (2014), 'An adaptive particle swarm optimization algorithm based on directed weighted complex network', Mathematical Problems in Engineering **2014**, *1–7.*

Likhachev, M., Ferguson, D. I., Gordon, G. J., Stentz, A. & Thrun, S. (2005), Anytime dynamic a: An anytime, replanning algorithm.,* in *'ICAPS', Vol. 5, pp. 262–271.*

Liu, H., Li, X., Fan, M., Wu, G., Pedrycz, W. & Suganthan, P. N. (2020), 'An autonomous path planning method for unmanned aerial vehicle based on a tangent intersection and target guidance strategy', IEEE Transactions on Intelligent Transportation Systems .

Luo, M., Hou, X. & Yang, J. (2020), 'Surface optimal path planning using an extended dijkstra algorithm', Ieee Access **8**, *147827–147838.*

Majid-al Rifaie, M. (2020), 'Dispersive flies optimisation: A tutorial', Swarm Intelligence Algorithms *p. 135–147.*

Mirjalili, S. & Lewis, A. (2016), 'The whale optimization algorithm', Advances in Engineering Software **95**, *51–67.*
URL: *https://www.sciencedirect.com/science/article/pii/S0965997816300163*

Mirjalili, S., Mirjalili, S. M. & Lewis, A. (2014), 'Grey wolf optimizer', Advances in Engineering Software **69**, *46–61.*
URL: *https://www.sciencedirect.com/science/article/pii/S0965997813001853*

Moore, E. F. (1959), The shortest path through a maze, in *'Proc. Int. Symp. Switching Theory, 1959', pp. 285–292.*

Muazu, A. A., Hashim, A. S. & Sarlan, A. (2022), 'Review of nature inspired metaheuristic algorithm selection for combinatorial t-way testing', IEEE Access **10**, *27404–27431.*

N, A. & V, U. (2019), 'Obstacle avoidance and distance measurement for unmanned aerial vehicles using monocular vision', International Journal of Electrical and Computer Engineering (IJECE) **9**, *3504.*

Nisingizwe, M. P., Ndishimye, P., Swaibu, K., Nshimiyimana, L., Karame, P., Dushimiyimana, V., Musabyimana, J. P., Musanabaganwa, C., Nsanzimana, S. & Law, M. R. (2022), 'Effect of unmanned aerial vehicle (drone) delivery on blood product delivery time and wastage in rwanda: a retrospective, cross-sectional study and time series analysis', The Lancet Global Health **10***(4), e564–e569.*

Park, S., Sugumar, V. & Michael, V. (2019), 'Modelling and simulation of a collision avoidance system'.

Pisarov, J. & Mester, G. (2020), 'The future of autonomous vehicles', FME Transactions **49**, *29–35.*

Páleš, D. & Rédl, J. (2015), 'Bezier curve and its application', Mathematics in Education, Research and Applications **01**, *49–55.*

Randria, I., Ben Khelifa, M. M., Bouchouicha, M. & Abellard, P. (2007), A comparative study of six basic approaches for path planning towards an autonomous navigation, in *'IECON 2007 - 33rd Annual Conference of the IEEE Industrial Electronics Society', pp. 2730–2735.*

Ravankar, A., Ravankar, A. A., Kobayashi, Y., Hoshino, Y. & Peng, C.-C. (2018), 'Path smoothing techniques in robot navigation: State-of-the-art, current and future challenges', Sensors **18***(9).*
URL: *https://www.mdpi.com/1424-8220/18/9/3170*

Raza, A., Bukhari, S. H., Aadil, F. & Iqbal, Z. (2021), 'An uav-assisted vanet architecture for intelligent transportation system in smart cities', International Journal of Distributed Sensor Networks **17***(7), 155014772110317.*

Remer, B. & Malikopoulos, A. A. (2019), 'The multi-objective dynamic traveling salesman problem: Last mile delivery with unmanned aerial vehicles assistance', 2019 American Control Conference (ACC) .

Ruiz Estrada, M. & Ndoma, A. (2019), 'The uses of unmanned aerial vehicles –uav's- (or drones) in social logistic: Natural disasters response and humanitarian relief aid', Procedia Computer Science **149**, *375–383.*

Sariff, N. & Buniyamin, N. (2010), Genetic algorithm versus ant colony optimization algorithm - comparison of performances in robot path planning application., pp. 125–132.

Shao, S., Peng, Y., He, C. & Du, Y. (2020), 'Efficient path planning for uav formation via comprehensively improved particle swarm optimization', ISA Transactions **97***, 415–430.*
***URL:** https://www.sciencedirect.com/science/article/pii/S0019057819303532*

Song, B., Qi, G. & Xu, L. (2019), 'A survey of three-dimensional flight path planning for unmanned aerial vehicle', 2019 Chinese Control And Decision Conference (CCDC) *.*

Sonmez, A., Koçyiğit, E. & Kugu, E. (2015), 'Optimal path planning for uavs using genetic algorithm', 2015 International Conference on Unmanned Aircraft Systems, ICUAS 2015 *pp. 50–55.*

Stentz, A. (1994), The d algorithm for real-time planning of optimal traverses., Technical report, Carnegie-Mellon Univ Pittsburgh Pa Robotics Inst.*

Stützle, T. & Hoos, H. (1999), 'Max-min ant system', **16***.*

Tang, Y., Wang, N., Lin, J. & Liu, X. (2019), Using improved glowworm swarm optimization algorithm for clustering analysis, in *'2019 18th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)', pp. 190–194.*

Toma, A.-I., Hsueh, H.-Y., Jaafar, H. A., Murai, R., Kelly, P. H. & Saeedi, S. (2021), 'Pathbench: A benchmarking platform for classical and learned path planning algorithms', 2021 18th Conference on Robots and Vision (CRV) *.*

Uchiyama, H., L. J. & Atkins, K. (2021), Effective drone usage for wildfire coverage in victoria.

Wang, W., Gope, P. & Cheng, Y. (2022), 'An ai-driven secure and intelligent robotic delivery system', IEEE Transactions on Engineering Management *p. 1–16.*

Xin, J., Zhong, J., Yang, F., Cui, Y. & Sheng, J. (2019), 'An improved genetic algorithm for path-planning of unmanned surface vehicle', Sensors **19***(11), 2640.*

Xu, S., H. E. & Shum, H. (2019), A hybrid metaheuristic navigation algorithm for robot path rolling planning in an unknown environment, in *'Mechatronic Systems and Control'.*

Xun, L., Dandan, W., Jingjing, H., Muhammad, B. & Ma, L. (2020), 'An improved method of particle swarm optimization for path planning of mobile robot', Journal of Control Science and Engineering **2020***.*

Yan, Z., Zhang, J., Zeng, J. & Tang, J. (2022), 'Three-dimensional path planning for autonomous underwater vehicles based on a whale optimization algorithm', Ocean Engineering **250***, 111070.*
***URL:** https://www.sciencedirect.com/science/article/pii/S0029801822004838*

Yang, L., Qi, J., Song, D., Xiao, J., Han, J. & Xia, Y. (2016), 'Survey of robot 3d path planning algorithms', Journal of Control Science and Engineering **2016***, 1–22.*

Yinka-Banjo, C. & Ajayi, O. (2020), 'Sky-farmers: Applications of unmanned aerial vehicles (uav) in agriculture', Autonomous Vehicles *.*

Yue, L. & Chen, H. (2019), 'Unmanned vehicle path planning using a novel ant colony algorithm', EURASIP Journal on Wireless Communications and Networking **2019***(1), 1–9.*