The University of Nottingham

UNITED KINGDOM · CHINA · MALAYSIA

Multi-Goal Path-Planning & Path-Finding in Simulated Underwater Environments

20215171
Zeyad Hesham Emara

May 2023

Abstract

Multi-goal path-planning and path-finding algorithms are the backbone of many autonomous agent systems, and specifically Autonomous Underwater Vehicles(AUVs) which are an invaluable asset for ocean exploration, salvage and recovery operations and many other applications. In this paper, we will compare two algorithms for path-planning and two for path-finding in a grid based 2D simulated underwater environment with a AUV as an intelligent agent aiming to reach all the goals assigned to it. The algorithms implemented for path-planning include: a genetic algorithm which treats the multi-goal path-planning problem as traveling salesman problem, thus searching for the optimal goal order/solution. A greedy hybrid algorithm which combines A* search with a greedy best first strategy for choosing the target goal. Additionally, the two algorithms which will be compared for path-finding are A* search and Dijkstra. Experiments will be carried out to analyse the task performance of each of the algorithms.

Contents

1 Introduction

Path-planning and path-finding algorithms serve as the backbone of many autonomous agents, as they enable them to navigate their environment efficiently and effectively. These algorithms solve the problem of determining an optimal or near-optimal path from a start location to a goal location while minimizing some cost such as distance, time, or energy expended, with addition of potentially avoiding obstacles throughout the environment.

Autonomous agents that employ these algorithms span a wide array of domains, including robotics, transportation, logistics, video games, exploration and countless other domains. As a result it is increasingly important to be able to compare different Path-planning and path-finding algorithms for each domain and application to not only better understand the strengths and weaknesses of each algorithm but also to determine which algorithm is best for a specific application or domain.

A major domain which utilizes path-planning and path-finding algorithms is the domain of unmanned and autonomous underwater vehicles(AUVs), which are playing a crucial role in ocean exploration, oceanic surveys, salvage operations and many other applications[5, 10].

In this project two path-planning and two path-finding algorithms will be compared in a discrete grid-based 2D simulation of an underwater environment with unknown obstacles, with an autonomous underwater vehicle as an agent aiming to reach all the assigned goals in the least moves/shortest path. The performance of each algorithm will be evaluated and analysed using statistical methods and data visualization methods to understand the applicability of each algorithm to the problem scenario/tasks.

1.1 Aims and Objectives

1.1.1 Aims

The project aims are to answer the following questions through experimentation and analysis:

- How do different path-planing algorithms (Genetic Algorithm and Hybrid Greedy Best First Search) compare in terms of path planning performance? Performance comprises: computation time, total path length to reach all goals. The Algorithms will be tested with different number of goals, different obstacle densities and different environment sizes to form a holistic and complete view of each algorithms performance.

- How do different path-finding algorithms (A* and Dijkstra) compare in terms of path finding performance? Performance comprises: computation time, path length. The Algorithms will be tested with different number of goals, different obstacle densities and different environment sizes to form a holistic and complete view of each algorithms performance.

These questions will be answered through experimentation, collecting results and performing an analysis using statistical methods on said results.

1.1.2 Objectives

The achieve the aims of the project, several objectives were set:

- Conduct a literature review to develop an understanding of:

  - The current methods and algorithms used for path-planning in autonomous agents.
  - The current methods and algorithms used for path-finding in autonomous agents.
  - The field of autonomous underwater vehicles, their applications, the different environment models and representations and the peculiarities of an underwater environment.

- Design and develop an environment for the agent to operate within and to facilitate the testing and evaluation of the path-planing and path-finding methods.
- Design and implement an intelligence agent which uses the combination of path-planning and path-finding search methods to reach all the goals in the environment.
- Design and perform a set of experiments that collect the relevant data about each algorithm's performance in carrying out the task.
- Evaluate each algorithm results, discuss and analyse the result to reach a conclusion about how the different algorithms compare for both tasks; path-planning and path-finding.
- Identify and discuss potential ares of improvement and what future work can contribute to the field of path-planning and path-finding in autonomous vehicles generally and for autonomous underwater vehicles specifically.

2 Related Work

This section will outline the relevant literature surrounding AUVs, the different environment modeling available and the path-planning and path-finding algorithms they employ. Particularly, in the field of grid-based 2D simulations.

2.1 Autonomous Underwater Vehicles

Autonomous Underwater Vehicles (AUVs) are unmanned, autonomous robots capable of performing tasks underwater without directions from humans[5]. Several forms of agent and environment modeling exist for AUVs, from regular 2D grid based models, to 3D and irregular terrain models[5].

2.2 Multi-Goal Path-Planing

The Multi-Goal Path-Planning problem can be formally described as the problem of finding the optimal path for an autonomous agent, which could be a robot or a vehicle, to reach multiple destinations or goals in an environment[4].

Given a set of points which represent goals $G = \{g1, g2, ..., gn\}$ in a given environment map, and a start point s, the problem is to find path with the minimum cost which visits each point in G exactly once[4].

The Multi-Goal Path-Planning problem in this form is a generalization of the traveling salesman problem which is widely studied and countless exact, heuristic and meta-heuristic solvers exist for solving it[2].

The Multi-Goal Path-Planning is encountered a lot in the field of AUVs as AUVs often need to find the path plan which minimizes the distance moved, energy expended and other criteria such as path safety[5].

3 Design and Methodology

In this section, the design of the environment, AUV agent and the algorithms implemented will be outlined with justifications made with respect to design choices.

3.1 Environment

The environment is a grid based 2D simulation of an underwater environment. The environment is initially unknown to the AUV and it contains a certain density of obstacles.With the obstacle density being the percentage of the grids in the environs which are impassable. It contains a certain number of goals places at a location(x,y) coordinate.

The grid is initialized with a certain width and height which represent the number of cells horizontally and vertically. Each cell in the grid is identified by a pair of coordinates (x, y). The grid permits all directions of movement(orthogonal and diagonal) with all cells having the same weight for traversing them. The environment size, obstacle densities and number of goals are variable and can be set by the user.

Using a 2D grid simplifies the problem space while still providing enough complexity to test path-planning and path-finding algorithms[1]. While the real world is 3D, in many scenarios (especially in underwater environments), it's reasonable to approximate the environment as 2D[1]. This approach is computationally efficient and provides a clear visualization of the AUV's behavior. Moreover, having a variable grid size allows the simulation to scale. Larger grids can represent more expansive environments and can be used to test the efficiency of an algorithm and how scalable an algorithm is. Furthermore, assuming a uniform cost for traversing grid cells simplifies the problem and makes it easier to compare the performance of different algorithms. It's a reasonable assumption when the underwater environment has a uniform current and the AUV's speed is constant. Finally, allowing both types of movements gives the AUV more flexibility in choosing its path and can lead to more direct (and thus shorter) paths to the goal. This also makes the movement of the AUV more realistic, as a real-world AUV is not restricted to orthogonal movements.

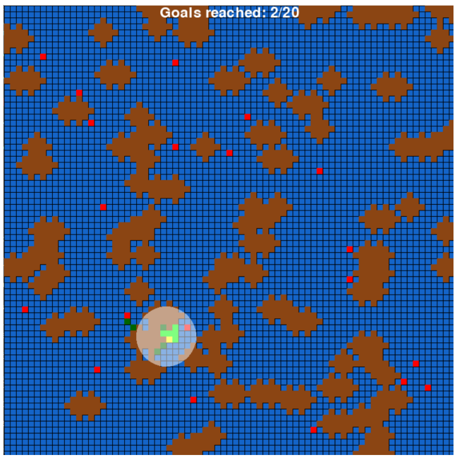


Figure 1: 2D grid based environment. Blue cells are passable, Brown are obstacles, Red are goals, Yellow is the AUV, transparent circle is the AUV obstacle detection range/sensor range.

These are the Environment parameters:

Parameter	**Value**
Obstacle Density Low	0.1
Obstacle Density Medium	0.25
Obstacle Density High	0.4
Small Environment Size	(50, 50)
Medium Environment Size	(75, 75)
Large Environment Size	(150, 150)
Number of Goals	10
Number of Goals	20
Number of Goals	30

Table 1: Environment Parameters

3.2 Agent

The agent design incorporates several intelligent based methods to achieve its tasks and interact with its environment.

This includes:

- Search-based AI: The AUV uses a search-based path-finding approach to navigate its environment. It employs either the A* search or Dijkstra's algorithm to find the shortest path to a goal. Moreover, the AUV uses a path-planning algorithm to determine what order to visit each goal in.

- Reactive AI: The AUV has a reactive component. It uses its sensors to detect obstacles in its vicinity and updates its knowledge of the environment accordingly. This is then used to update the AUV's path, ensuring that it avoids the detected obstacles. So essentially, the AUV is able to make decisions based on its current sensor inputs.

- State-based AI: The AUV maintains a state, which includes its current position and the list of detected obstacles. This state is used to make decisions and is updated as the AUV moves around and detects obstacles. This influences the AUVs path-finding component as when a new object is detected, it checks weather its current path collides with any of the discovered obstacles, and recomputes a new path if the old path will result in a collision with the obstacle.

- Goal-oriented behavior: The AUV exhibits goal-oriented behavior. It has a list of goals a path-planning algorithm to determine the which goals to navigate to first, then it uses path-finding search algorithms to find the path to the goal. Once a goal is reached, it is removed from the list of goals. The AUV continues this behavior until all goals have been reached.

As the environment is a discrete 2D grid based environment, the AUV moves by changing its x and y coordinates by a certain amount (dx and dy respectively). This represents a discrete jump from one cell to another and not a continuous movement. This movement method would be used in conjunction with a path-finding algorithm. Moreover, the environment layout is partially unknown to the agent, as in the agent does not have a map of the obstacle locations, but it knows the goal locations, and its own location within the environment. The AUV is able to utilize a sensor with a fixed radius around its own position to detect obstacles, which are then stored in memory of the AUV so it can account for the detected obstacles when using a path-finding algorithm to navigate to the goal position.

3.3 Path-Planning Algorithms

Two path-planning algorithms are a genetic algorithm which treats the multi-goal path-planning problem as the TSP and solves for the optimal or sub-optimal sequence of goals which minimizes distance, and a hybrid approach to multi-goal path-planning which combines greedy best first search with A* search for determining goal order.

3.3.1 Genetic Algorithm

Genetic algorithms are a type of optimization and search method inspired by the process of natural selection[6]. They use techniques such as mutation, crossover, and selection to generate solutions to optimization problems. Genetic algorithms represent potential solutions as 'children'/'chromosomes' and evolve these solutions over generations to find the best or 'fittest' solutions. The genetic algorithm implemented in this project solves the TSP for the best sequence of goals to minimize total path length to visit all goals.

Algorithm 1 Genetic Algorithm

```
1: procedure Genetic Algorithm($goals$, $populationSize$, $mutationRate$, $numGenerations$)
2:     $population \leftarrow$ InitializePopulation($populationSize$, $goals$)
3:     for $i \leftarrow 1$ to $numGenerations$ do
4:         $newPopulation \leftarrow$ EmptyList
5:         for $j \leftarrow 1$ to $populationSize$ do
6:             $parent1 \leftarrow$ Select($population$)
7:             $parent2 \leftarrow$ Select($population$)
8:             $child \leftarrow$ Crossover($parent1$, $parent2$)
9:             if RandomNumber(0, 1) $< mutationRate$ then
10:                $child \leftarrow$ Mutate($child$)
11:            end if
12:            Append $child$ to $newPopulation$
13:        end for
14:        $population \leftarrow newPopulation$
15:        $bestIndividual \leftarrow$ MaxFitnessIndividual($population$)
16:    end for
17:    return MaxFitnessIndividual($population$)
18: end procedure
```

3.3.2 Hybrid Greedy Best First Algorithm

This algorithm is a greedy best first approach to solving the path-planning problem. The algorithm works by first calculating the shortest distance to all the goals form the position of the AUV, it does this using the A* search algorithm taking into account any obstacles detected by the AUV. The algorithm then selects the closest goal to the AUV's current position. The closest goal is then navigated to by the AUV along the path determined by the path-finding algorithm. Then, once a goal is reached, it is removed from the list of goals and the next closest goal is chosen. This process continues until all goals have been reached. This process is repeated every move, this allows the AUV dynamically change the goal its navigating to based on new input from the environment. For example, if the AUV is currently navigating to a goal and an obstacle is detected, the AUV will then recalculate the shortest known paths from its current position to all other goals. If the obstacle results in a significantly longer path such that the original goal is no longer the closest goal, the AUV will navigate to another goal which is the current closest goal.

A major strength of this algorithm is that its adaptive. By recalculating the path using the A* algorithm whenever new obstacles are detected, the AUV can adapt to changes occurring in the environment. This is a particularly powerful strategy in unknown environments and environments which are very dynamic. Moreover, the utilization of the powerful A* path-finding algorithm guarantees the optimal path to each individual goal, given the current knowledge of obstacles.

Nevertheless, this algorithm has some downsides, chiefly its computational cost, recalculating the shortest path to each goal using the A* algorithm can be computationally intensive, especially for large grids or many goals and when done frequently in a cluttered environment. Moreover, this approach does not guarantee the shortest possible path to visit all goals. It selects the goal with the shortest path at each step, which might not be part of the overall shortest path to visit all goals.

Algorithm 2 Hybrid Greedy Best First

```
procedure AUV Path Plan($startPosition$, $goalPositions$, $obstacles$)
    while $goalPositions$ is not empty do
        for each $goal$ in $goalPositions$ do
            $path[goal] \leftarrow A*(startPosition, goal, obstacles)$
            $pathLength[goal] \leftarrow length(path[goal])$
        end for
        $nextGoal \leftarrow argmin(pathLength)$
        $nextPath \leftarrow path[nextGoal]$
        while not reached $nextGoal$ do
            move along $nextPath$
            if new $obstacles$ detected then
                $nextPath \leftarrow A*(startPosition, nextGoal, obstacles)$
            end if
        end while
        remove $nextGoal$ from $goalPositions$
    end while
end procedure
```

3.4 Path-Finding Algorithms

3.4.1 A* Search

A* Algorithm is a path-finding algorithm that uses a heuristic function to guide its search towards the goal[7]. It maintains a priority queue of nodes, where the priority is determined by the cost to reach the node and the estimated cost from that node to the goal. This allows A* to find the shortest path while exploring fewer nodes than Dijkstra's Algorithm when a suitable heuristic is used[7].

The heuristic function used for this implementation is a euclidean distance heuristic [8] which uses a combination of diagonal and orthogonal movements.

Algorithm 3 Heuristic Function

```
function Heuristic($current, goal$)
    $dx \leftarrow abs(current[0] - goal[0])$
    $dy \leftarrow abs(current[1] - goal[1])$
    return $\sqrt{2} \times \min(dx, dy) + abs(dx - dy)$
end function
```

3.4.2 Dijkstra

Dijkstra's Algorithm is a famous graph traversal method used to determine the shortest path from a starting node to all other nodes in a graph[9]. The algorithm maintains a set of un-visited nodes and continually selects the node with the smallest distance from the start node, updating the distances to its neighbours, until all nodes have been visited. It then chooses the shortest path found to the goal node.

4 Implementation

The implementation of this project was carried out in python programming language, using the pygame engine for simulation. The pygame engine is an easy to use and simple game engine in python which allows for developing this king of project and allows for seamless and smooth simulations as well as providing tools for selecting different configurations of environments and different algorithms. Several challenges were presented during the implementation which will be explored in this section.

4.1 Environment

For implementing the environment presented several challenges, chiefly random obstacle creation, and random placement of the AUV and goals. The placement of obstacles should be random to ensure each simulation run is unique, providing diverse scenarios for testing the AUV's path-planning and path-finding algorithms. To tackle this problem I implemented a unique obstacle creation function.

This function generates obstacles in a random manner but not just randomly. The obstacles are created in blobs of random sizes between 3-4, meaning that when an obstacle is created, a few additional obstacles are also created in its immediate vicinity (up, down, left, right).

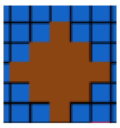


Figure 2: Obstacle Blob

The number of obstacles is determined by the density parameter, which represents the proportion of the grid that should be filled with obstacles. The function continues creating blobs until the desired number of obstacles is reached.

Using blobs to create obstacles can lead to a more realistic representation of underwater environments. As in nature, obstacles (like rocks, reefs, or wrecks) often come in clusters or formations rather than being evenly distributed. Therefore, using blobs can lead to more natural and realistic obstacle distributions in the environment.

Moreover, in tandem with the obstacle generation function I implemented another function for generating the AUV and goal positions. The function first generates a list of available positions on the grid that don't contain obstacles. Then, it randomly selects one of these positions as the starting position for the AUV. The starting position is then removed from the list of available positions to ensure that no goal is placed in the same position. Finally, a number of positions are randomly chosen from the remaining available positions to be the goals.

Together, these two functions prepare the grid environment and the AUV the multi-goal path-planning and path-finding tasks and they ensure that the AUV and the goals are placed in free spaces, and that the distribution of obstacles is realistic. This set-up is essential for testing and comparing the performance of the different algorithms.

4.2 Genetic Algorithm

The genetic algorithm used in this implementation posed some challenges, mainly parameter tuning and testing. Parameter tuning is extremely important for optimizing the genetic algorithms performance and to find optimal parameters. This is because the performance of the GA heavily depends on the chosen parameters(population_sizes, mutation_rates, and num_generations). Grid search provides a systematic way to find the best parameters for the GA. As a result I implemented grid search method for optimizing the GA parameters[3].

4.2.1 Parameter Tuning Grid Search

Grid search is a tuning technique[3] that attempts to compute the optimum values of hyper-parameters. It is an exhaustive search technique which works by defining a grid of hyper-parameters and then evaluating the performance for each point on the grid, which is essentially a brute force technique to find the best parameters. It is fairy time consuming but leads to an optimized set of hyper parameters for the GA.

In my implementation of grid search, the following parameter space was defined:

Population Size	**Mutation Rate**	**Number of Generations**
50	0.01	500
100	0.05	1000
200	0.1	2000

Table 2: GA grid search Parameter Space

I trained the GA with grid search on an environment of the following configuration:

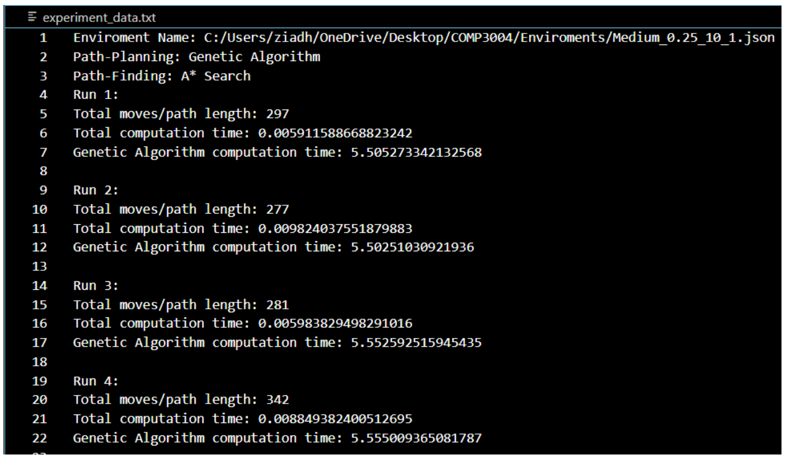
Parameter	**Value**
Obstacle Density	0.25
Number of Goals	20
Map Size	75 x 75

Table 3: Environment Parameters

The best parameters found were: population size 50, mutation rate 0.01, and 1000 generations. This set of parameters was used in all the experiments involving the GA.

4.3 Experimentation and Results

Implementing an efficient experiment framework is crucial for collecting results efficiently and easily. I integrated into my program a save and load functionality to save and load environment data. This is for maintaining the same environment layout, goal positions and AUV starting positions across the different experiments. This is obviously crucial for conducting methodical experimentation where the environment has to be the same when comparing performance of an algorithm to another. Moreover, I integrated into my main method the option to run an algorithm for several independent runs and record the evaluation metrics for each independent run and the average values of these metrics across all runs to a file. Moreover, I implemented a python script that automatically parses the data in the results file and visualizes the data of the results.



```
experiment_data.txt
Enviroment Name: C:/Users/ziadh/OneDrive/Desktop/COMP3004/Enviroments/Medium_0.25_10_1.json
Path-Planning: Genetic Algorithm
Path-Finding: A* Search
Run 1:
Total moves/path length: 297
Total computation time: 0.005911588668823242
Genetic Algorithm computation time: 5.505273342132568

Run 2:
Total moves/path length: 277
Total computation time: 0.009824037551879883
Genetic Algorithm computation time: 5.50251030921936

Run 3:
Total moves/path length: 281
Total computation time: 0.005983829498291016
Genetic Algorithm computation time: 5.552592515945435

Run 4:
Total moves/path length: 342
Total computation time: 0.008849382400512695
Genetic Algorithm computation time: 5.555009365081787
```

Figure 3: Experiment Result File Snippet

The evaluation metrics used are the Path length (number of moves) and the computation time. Additionally when using the GA, the GA computation time will be calculated separately from the path-finding computation time.

Path length is a measure of the efficiency of the path found by the algorithm. In the context of an AUV or any mobile robot, a shorter path generally means that the agent has to travel a shorter distance and thus consumes less energy. Furthermore, if the path is shorter, the AUV can reach its goal faster. It is also worthy of notice that shortest path is not always the best path as other factor might influence the quality of a path like the safety of the path or if the path travels through nodes which have a high coast of traversal. Nevertheless, in this implementation, the shortest path is the best path.

Computation time measures the amount of time the algorithm needs to find a path in seconds, which is a measure of the computational efficiency of an algorithm. This is very critical in many applications where computing a path quickly is required, perhaps to avoid obstacles or for highly dynamic environments. Due to the nature of the project implementation, the AUV is going to have to recompute paths as it detects obstacles that obstruct its path, for this metric we aggregate the computation times of all the path-finding operations carried out in one run.

By considering those two metrics we can develop an understanding of the performance of the path-planning and path-finding algorithms at question. This allows us to identify which algorithm is more suitable for a particular application, environment or set of conditions.

5 Results and Evaluation

This section outlines the experiments designed for answering the questions proposed in section 1.1.1, the results obtained and a discussion in the light of those results.

5.1 Experiments

A set of experiments were designed to answer the questions presented in section1.1.1. Experiments are 2 types, ones which aim to evaluate the performance of path-planning algorithms and ones which aim to evaluate the performance of the path-finding algorithms. The resulting data will be used in the evaluation and ultimately answering the questions posed in this project.

The experiments are:

- 1.Measure performance of different path-planning approaches:
  - Experimental Setup: Define a standard environment size, obstacle density, and sensor range. Use the same initial and goal states for all algorithms. Use the same path-finding algorithm(A*).
  - Manipulated Variables: The path-planning algorithm.
  - Evaluation Metrics: Path length (number of moves), computation time.
  - Procedure: Run 20 times in the same environment, record the evaluation metrics.
- 2.Measure performance of different path-finding approaches:
  - Experimental Setup: Define a standard environment size, obstacle density, and sensor range. Use the same initial and goal states for all algorithms. Use the same path-planning algorithm(HGBF).
  - Manipulated Variables: The path-finding algorithm.
  - Evaluation Metrics: Path length (number of moves), computation time.
  - Procedure: Run each algorithm 20 times in the same environment and record the Path length (number of moves), computation time for each run.
- 3.Impact of environment size on path-finding task performance:
  - Experimental Setup: Define a standard obstacle density and sensor range, use the same path-planning algorithm(HGBF).
  - Manipulated Variables: Environment size (small, medium, large).
  - Evaluation Metrics: Path length (number of moves), computation time.
  - Procedure: Run the same algorithm on environments of different sizes, with the same obstacle density and sensor range.
- 4.Impact of number of goals on path-planning task performance:
  - Experimental Setup: Define a standard environment size, obstacle density, sensor range, path-finding algorithm(A*).
  - Manipulated Variables: Number of goals.

  - Evaluation Metrics: Path length (number of moves), computation time.
  - Procedure: Run each algorithm multiple times in the same environment with different number of goals.
- 5.Impact of obstacle density on path-finding task performance:
  - Experimental Setup: Define a standard environment size and sensor range, use the same path-planning algorithm(HGBF).
  - Manipulated Variables: Obstacle density.
  - Evaluation Metrics: Path length (number of moves), computation time.
  - Procedure: Run each algorithm multiple times in the same environment size with different obstacle densities.

5.2 Results

For each experiment, the data was gathered, parsed and The Legend indicates the algorithms used. The Title indicates the environment parameters used in the format: Map-Size_Obstacle-Density_Number-of-Goals. All times recorded and shown in the results are in seconds(s), all path lengths are in number of nodes traversed to reach all goals.

5.2.1 Experiment 1

1.Measure performance of different path-planning approaches:

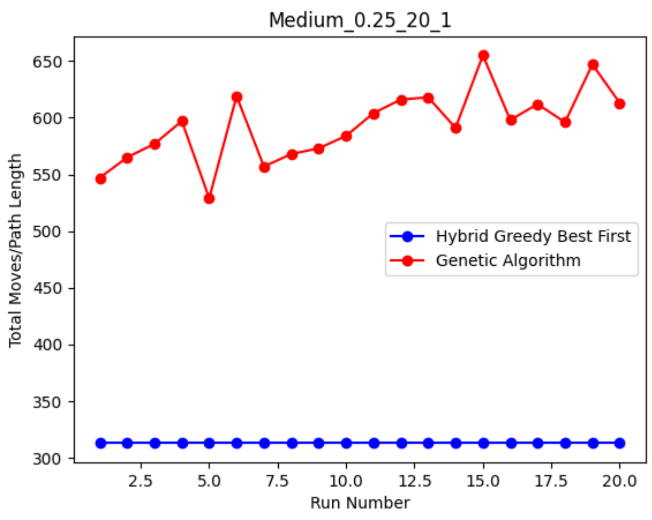


Figure 4: Experiment 1 Path Lengths

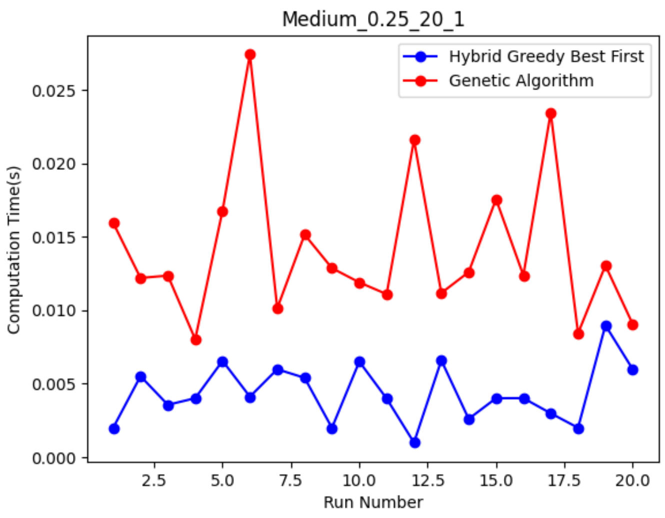


Figure 5: Experiment 1 Computation Times(s)

Algorithm	**Mean Computation Time(s)**	**Mean Path Length**
Genetic Algorithm	10.0	593.3
Hybrid Greedy Best First Search	0.00634	313.0

Table 4: Experiment 1 Results : Environment: Medium_0.25_20 : 20 Runs

5.2.2 Experiment 2

2.Measure performance of different path-finding approaches:

Algorithm	**Mean Computation Time(s)**	**Mean Path Length**
A*	0.00502	313.0
Dijkstra	0.108	303.0

Table 5: Experiment 2 Results : Environment: Medium_0.25_20 : 20 Runs

5.2.3 Experiment 3

3.Impact of environment size on path-finding task performance:

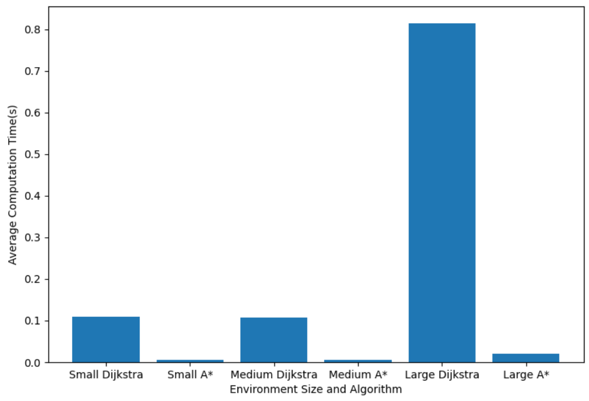


Figure 6: Experiment 3 Results

Algorithm	**Mean Computation Time(s)**			**Mean Path Length**		
	Small	**Medium**	**Large**	**Small**	**Medium**	**Large**
Dijkstra	0.10	0.10	0.81	260.0	303.0	629.0
A*	0.0049	0.0050	0.019	249.0	313.0	639.0

Table 6: Experiment 3 Results : 20 Runs

5.2.4 Experiment 4

4.Impact of number of goals on path-planning task performance:

Algorithm	**Mean Computation Time(s)**			**Mean Path Length**		
	10 Goals	**20 Goals**	**30 Goals**	**10 Goals**	**20 Goals**	**30 Goals**
Genetic Algorithm	5.48	10.07	14.8	309.9	593.3	940.75
Hybrid Greedy Best First Search	0.0048	0.0043	0.0072	242.0	313.0	406.0

Table 7: Experiment 4 Results : Environment: Medium_0.25_Number_of_Goals : 20 Runs

5.2.5 Experiment 5

5.Impact of obstacle density on path-finding task performance:

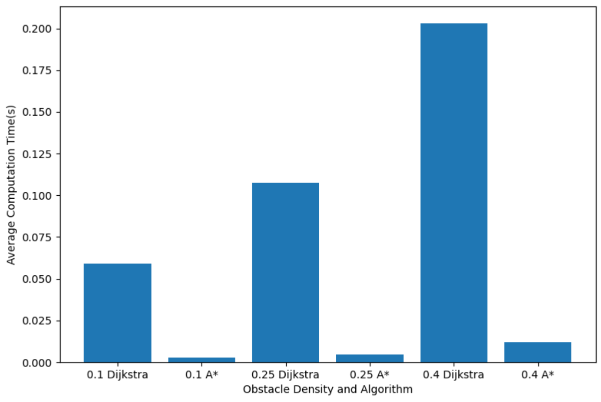


Figure 7: Experiment 5 Results

Algorithm	**Mean Computation Time(s)**			**Mean Path Length**		
	0.1 Density	**0.25 Density**	**0.4 Density**	**0.1 Density**	**0.25 Density**	**0.4 Density**
Dijkstra	0.059	0.10	0.20	280.0	303.0	456.0
A*	0.0025	0.0043	0.012	280.0	313.0	464.0

Table 8: Experiment 5 : Environment: Medium_Obstacle_Density_20 : 20 Runs

5.3 Discussion

The results of the experiments outlined in the previous section provide several insights into the general performance of the algorithms, thus enabling us to give an informed answer to the questions outlined earlier1.1.1.

For the path-planning algorithms, the results indicated that the Hybrid Greedy Best First(HGBF) algorithm outperformed the Genetic Algorithm in all metrics, HGBF was consistently finding shortest paths, took much less computational time and produced more consistent results. As shown in figure 4, the genetic algorithm exhibited a lot of variability in its results. Additionally, the GA took substantially more computation time than HGBF in all experiments. This can be attributed to several aspects. Firstly, in this implementation, the GA assumes that no obstacles are present in the the environment during its search process, as a result the produced path-plan can be obstructed by obstacles in the environment. Moreover, the GA takes a long time to find a solution. The hyper-parameters used resulted in a long run time of the GA.

Furthermore, results in experiment 4 highlight the declining performance of the GA as the number of goals increase with significantly more computation time and path length, while the HGBF algorithm has exhibits negligible increase in computation times. We can conclude from the results that the Hybrid Greedy Best First outperforms the GA in all in all the tested environment layouts.

For path-finding algorithms, A* search was shown to outperform Dijkstra's in all metrics and all environment layouts. Chiefly in larger environment sizes as shown in experiment 3 ??. Moreover, A* produced similar path lengths to Dijkstra's but always in less time. This illustrates that A* is the better path-finding algorithm. Especially, in an underwater environment for an AUV which requires very fast computation speeds and a high computation efficiency to be able to navigate its environment[5].

However, these conclusions need to be interpreted in light of the limitations of this project. The experiments were based on a limited number of environments and environment layouts. Therefore, while the results provide valuable insights and answer the questions posed, further research is needed to fully understand the performance characteristics of these algorithms under a wider range of conditions.

6 Conclusion

In conclusion, for path-planning, the Hybrid Greedy Best First Algorithm outperforms the genetic algorithm in all metrics, it was found that it was the ideal algorithm for path-planing in a dynamic and unexplored/unknown environment due to its adaptability and ability to dynamically change the target goal based on the input from the environment. Moreover, for path-finding, A* algorithm greatly outperforms Dijkstra's in most metrics, albeit they both produce optimal paths(shortest paths) but the A* search algorithm's better computational efficiency and better computation times results in an optimal path in the shortest time. This is due to the use of the heuristic function in the A* algorithm which guides its search in the most promising nodes only.

6.1 Future Work

There is a lot of potential future work and improvements to this project. The areas of future work can be split into three main domains: Model and environment simulation, AUV implementation and algorithms and methods.

Improving the environment is a potential are of improvement, whereby the environment could be modeled more realistically to include underwater currents, dynamic moving obstacles such as schools of fish or other AUV's. Moreover, the environment could be extended to a continuous 3D environment which better simulate the real life operations of a AUV.

Furthermore, the AUV model could be improved in the future to include more complex systems, such as using reinforcement learning to train it to avoid obstacles by manipulating its control systems in a continuous environment. A more accurate model of an AUV could be implemented by using an element of uncertainty in the AUV's sensors or systems to emulate the conditions of underwater operations which inhibit the processing and gathering of environment data.

Finally, more complex path-planning and path-finding algorithms can be implemented and analysed in the current simulation. Such as the D* or D* lite path-finding algorithms which are able to dynamically react to obstacles by only adjusting the part of the path affected by the detected obstacle[5]. Moreover, more optimization algorithms can be explored for the multi-goal path-planning problem, such as ant colony optimization and particle swarm optimization[5] which both can be used to solve for the TSP represented in this project.

6.2 Reflection

Overall the project was largely a success, all the aims and objectives were meet, the program implementation worked flawlessly, carried out the required experiments and produced results which were analysed to address the questions proposed in this paper. There were some challenges and setbacks however, chiefly in time management during the implementation of the program. Due to stalling to much on the programming side of the project the general was behind the desired time schedule. Moreover, the project could have been expanded upon, especially the idea of the AUV and the underwater environment could have been taken further. A more complex simulation of the underwater environment would have been desirable, as there is much to an underwater environment which can be modeled such as underwater currents, dynamic obstacles, etc. Furthermore, a more complex agent would have expanded the scope of the project to include learning strategies for obstacle detection and avoidance(using reinforcement learning or fuzzy logic controls) instead of the simplistic reactive agent implemented in this project.

7 Bibliography

[1] Zeyad Abd Algfoor, Mohd Shahrizal Sunar, and Hoshang Kolivand. A comprehensive study on pathfinding techniques for robotics and video games. *International Journal of Computer Games Technology*, 2015:1–11, 04 2015.

[2] Jie Chen, Fang Ye, and Yibing Li. Travelling salesman problem for uav path planning with two parallel optimization algorithms. pages 832–837, 2017.

[3] Álvaro Barbero Jiménez, Jorge López Lázaro, and José R. Dorronsoro. Finding optimal model parameters by discrete grid search. pages 120–127, 2007.

[4] Steven M LaValle. *Planning algorithms.* Cambridge university press, 2006.

[5] Daoliang Li, Peng Wang, and Ling Du. Path planning technologies for autonomous underwater vehicles-a review. *IEEE Access*, PP:1–1, 12 2018.

[6] John McCall. Genetic algorithms for modelling and optimisation. *Journal of Computational and Applied Mathematics*, 184(1):205–222, 2005. Special Issue on Mathematics Applied to Immunology.

[7] Masoud S. Nosrati, Ronak Karimi, and Hojat Allah Hasanvand. Investigation of the * (star) search algorithms: Characteristics, methods and approaches - ti journals. *World Applied Programming*, 2012.

[8] D Chris Rayner, Michael Bowling, and Nathan Sturtevant. Euclidean heuristic optimization. 25(1):81–86, 2011.

[9] Stuart J Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearson, 3rd edition.

[10] Russell Wynn, Veerle Huvenne, Tim Le Bas, Bramley Murton, Douglas Connelly, B. Bett, Henry Ruhl, Kirsty Morris, Jeffrey Peakall, Daniel Parsons, Esther Sumner, Stephen Darby,

Robert Dorrell, and James Hunt. Autonomous underwater vehicles (auvs): Their past, present and future contributions to the advancement of marine geoscience. *Marine Geology*, 352, 06 2014.