

AERSP 497 Autonomy

Project 3

Name	Contribution	Contribution %	Honor Statement Sign
Jackson Fezell	Code & Report	34	JF
Nicholas Giampetro	Code & Report	33	NG
Ankit Gupta	Code & Report	33	AG

Introduction

This project involved the deployment of a UAS to traverse a designated area, with the objective of delivering a package while avoiding obstacles like a tree line. This task requires a profound comprehension and application of Dijkstra's algorithm and the A* algorithm. Moreover, the project extended into the exploration of various planning algorithms using a MATLAB Toolboxes. This project demanded us to delve into different algorithmic strategies, experiment with parameter variations, and analyse the effects of specific modifications.

Part A

Our task was to write a script which implements Dijkstra and A* algorithm in an environment where a UAS is traversing an environment while avoiding obstacle (stand of trees).

Dijkstra Algorithm -This method is used for finding the shortest path between nodes in a graph, which could represent, for example, road networks. It works by initially setting the distance to the start node as 0 and all other nodes as infinity. The algorithm then repeatedly selects the nearest unvisited node, calculates the distance to each of its neighbours, and updates the neighbour's distance if a shorter path is found. This process continues until all nodes have been visited. The result is the shortest path from the start node to all other nodes in the graph.

A* Algorithm - Building upon the principles of Dijkstra's Algorithm, A* introduces a heuristic into the equation, which estimates the distance to the goal from each node. This heuristic guides the search, allowing the algorithm to prioritize paths that are leading closer to the destination. As a result, A* can find the shortest path more efficiently than algorithms that search blindly. It balances between the cost to reach the node (known as the g-score) and the estimated cost from that node to the goal (the h-score).

Results

NOTE: To switch between Dijkstra and A* change the value of RUN_DIJKSTRA from true to false and vice versa in the code (Proj3_Q1.m)

Proj3_Q1.m code is designed for a UAS package delivery simulation, utilizing pathfinding algorithms for navigation. It starts by setting up an 11x11 grid as the map, marking the start and goal locations, and placing obstacles represented by trees. The user can toggle between Dijkstra's and A* algorithms using the RUN_DIJKSTRA variable. The code then calculates the minimum travel costs from the start to all other nodes using the chosen algorithm.

Dijkstra's focuses solely on the actual travel cost, while A* adds a heuristic based on the distance to the goal. After computing the travel costs, the code backtracks from the goal to the

start node, determining the optimal path. Finally, it visualizes the results with a plot showing the grid, the optimal path, and the costs associated with each node, providing a clear and practical representation of UAV pathfinding in an obstacle-laden environment.

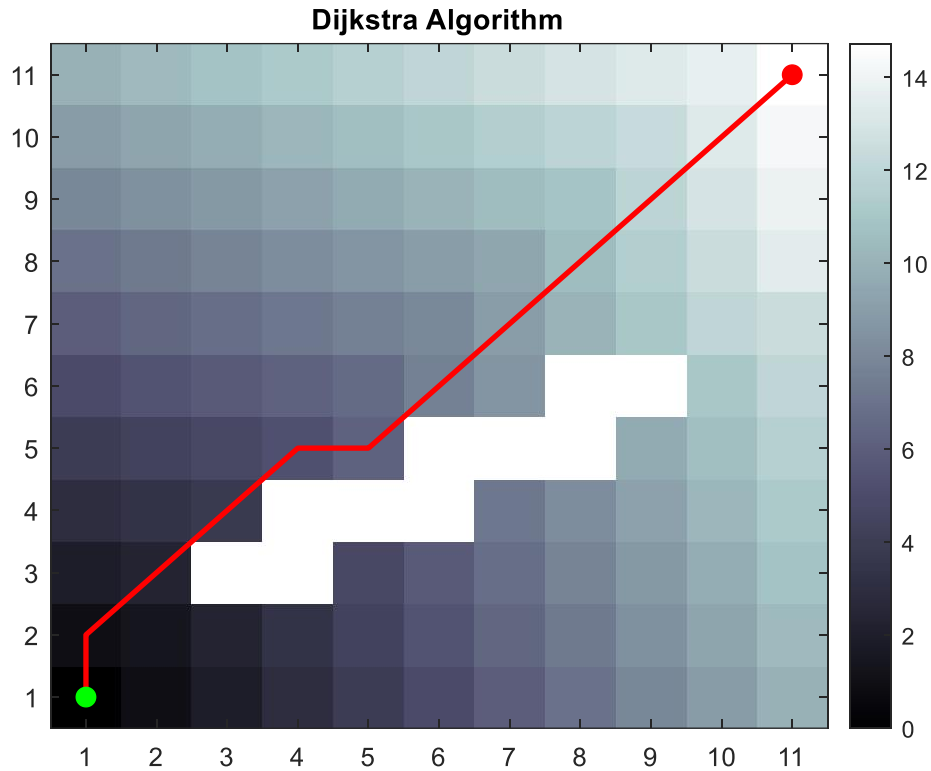


Figure 1: Dijkstra Algorithm

When utilizing Dijkstra's Algorithm in the 11x11 grid map, the chosen path is illustrated in red in Figure 1. The obstacles, representing a tree line, are indicated by the white blocks, as per the problem's description. The algorithm computes the total cost of traversing from the starting point at (1,1) to the destination at (11,11), which is 14.7279. This cost calculation considers the varying travel costs based on direction: moving from one block to another either vertically or horizontally incurs a cost of 1, whereas diagonal movements have a higher cost of $\sqrt{2}$.

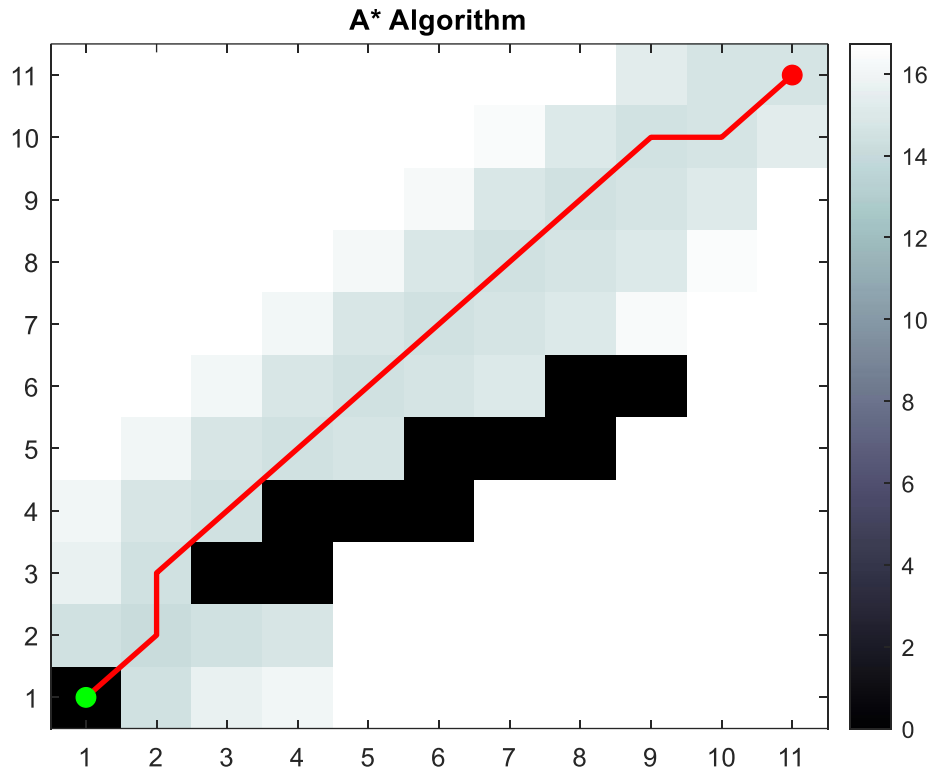


Figure 2: A* Algorithm

When utilizing A* Algorithm in the 11x11 grid map, the chosen path is illustrated in red in Figure 2. The obstacles, representing a tree line, are indicated by the black blocks, as per the problem's description. The algorithm computes the total cost of traversing from the starting point at (1,1) to the destination at (11,11), which is 14.7279. This cost calculation considers the varying travel costs based on direction: moving from one block to another either vertically or horizontally incurs a cost of 1, whereas diagonal movements have a higher cost of $\sqrt{2}$.

Part B

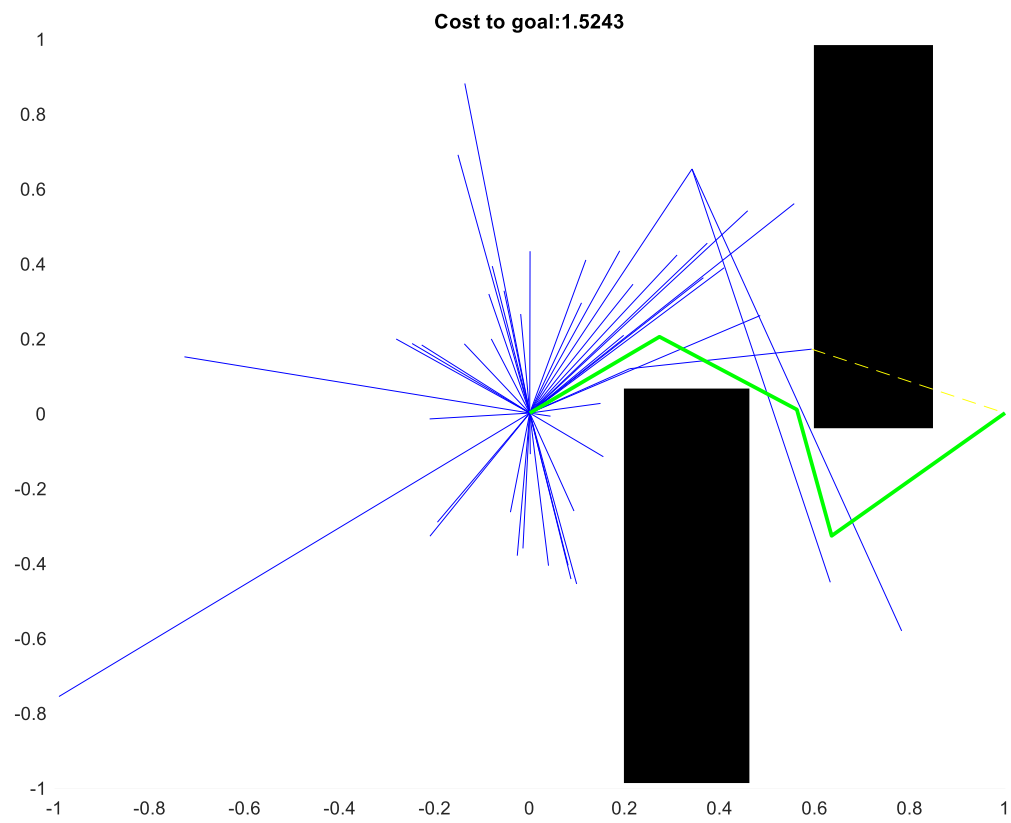


Figure 3: RRT* Method

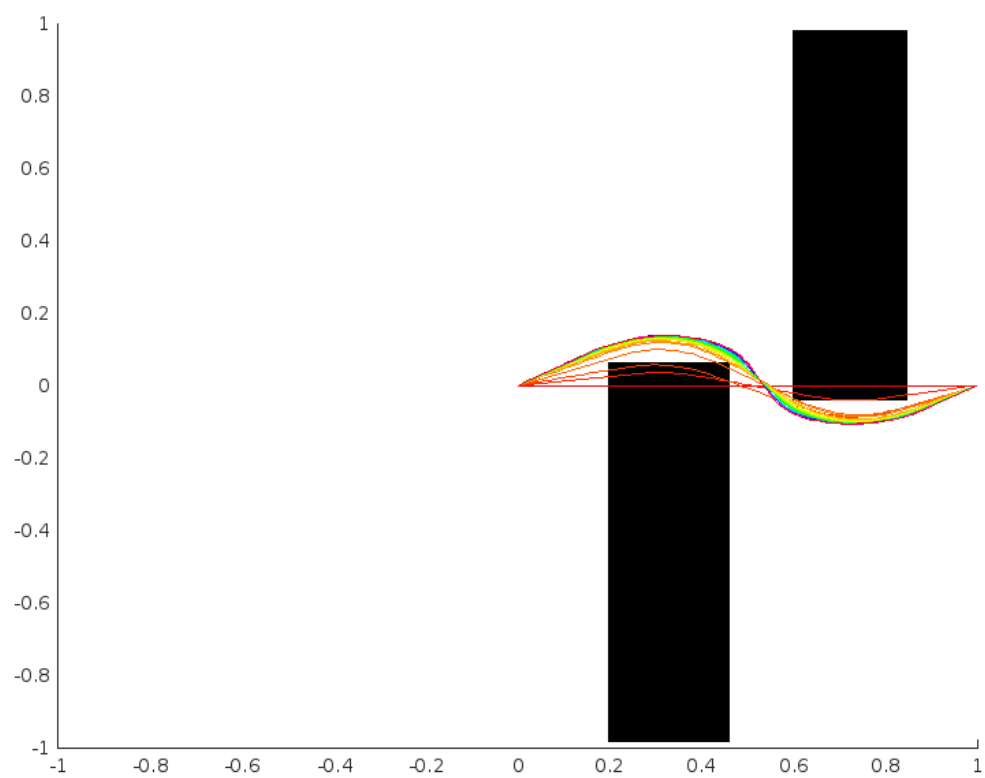


Figure 4 CHOMP Method

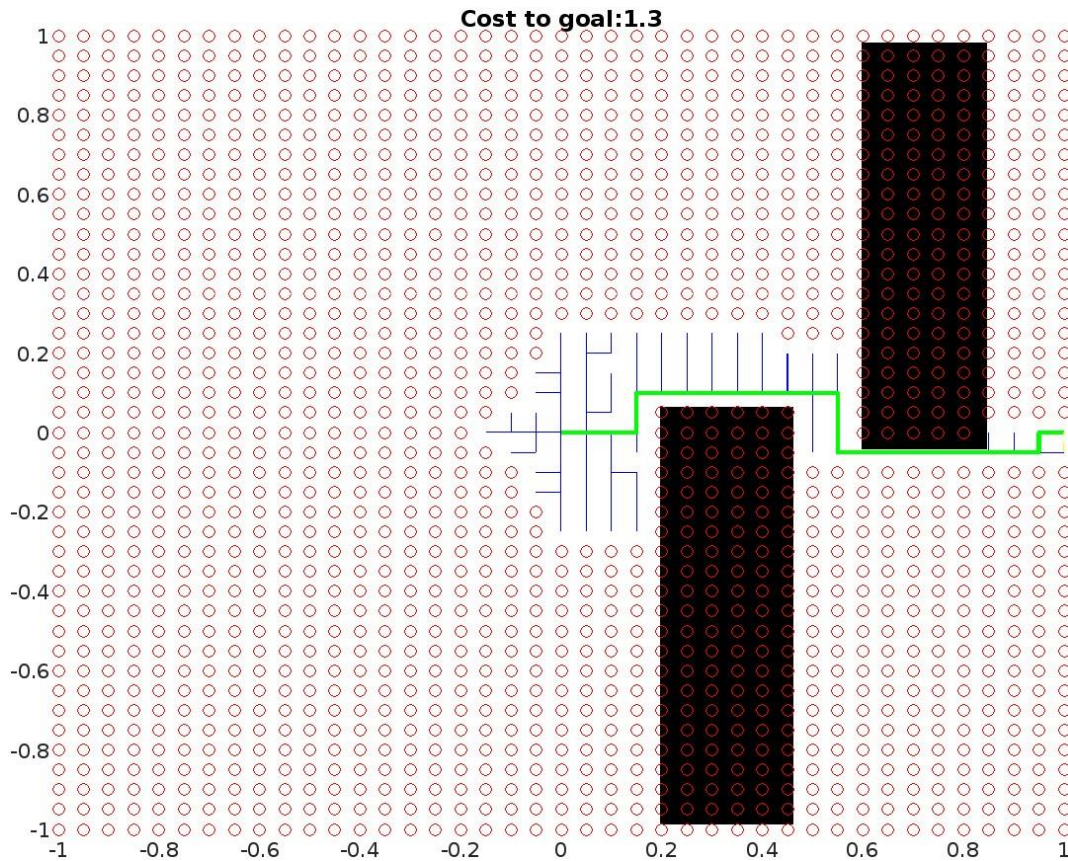


Figure 5 A* Method

Table 1: Cost for Each Method

Method	Total Cost
RRT*	1.524313
CHOMP	1.159691
A*	1.300000

How do the different algorithms behave?

- RRT* (Rapidly exploring Random Trees Star): RRT is an advanced pathfinding and motion planning algorithm, an enhancement of the original RRT. It is designed to rapidly explore and map out complex spaces by generating a randomly growing tree. Unlike its predecessor, RRT*, focuses on optimizing the path, ensuring that it not only finds a feasible route but also continually refines this route to approach the shortest possible path over time.
- CHOMP (Covariant Hamiltonian Optimization for Motion Planning): CHOMP is a trajectory optimization method. It works by iteratively refining a given trajectory to minimize a specific cost function, which usually includes components for smoothness and obstacle avoidance. The key advantage of CHOMP is its ability to produce smooth, collision-free paths, making it ideal for robots that require fluid, natural movements. However, its reliance on an initial trajectory and potential to get stuck in local minima are limitations.
- A* (A Star): A is a popular pathfinding algorithm, and it operates on a grid to find the shortest path between a start and a goal point. A* uses a heuristic approach,

combining the actual cost from the start point and an estimated cost to the goal, to prioritize which paths to explore. This makes it efficient and effective in finding optimal paths in grid-based environments. The algorithm's performance is heavily reliant on the heuristic used and can be computationally demanding in large, complex maps.

What happens if you vary the parameters?

RRT*

- **Growth Increment:** Adjusting the length of the branches or growth increment affects how quickly the tree expands through the space. Longer increments can lead to faster exploration but might miss narrow passages.
- **Rewiring Radius:** The radius used for rewiring affects the optimization of the path. A larger radius might find a more optimal path but increases computation time.
- **Sampling Density:** Changing the density of sampling can impact how the algorithm explores the space. More frequent sampling can lead to a more detailed map but increases computational load.

CHOMP

- **Smoothing Term:** Adjusting the weight of the smoothing term in the cost function can lead to smoother or more direct paths. A higher weight produces smoother paths but could potentially avoid more direct routes.
- **Obstacle Avoidance:** The weight given to obstacle avoidance can dictate how cautiously the algorithm plans around obstacles. Increasing this weight can result in longer paths.
- **Learning Rate:** The learning rate controls the speed of convergence during optimization. A higher rate can speed up convergence.

A*

- **Heuristic Weight:** Altering the weight of the heuristic part of the cost function can change the balance between exploration and goal-directed movement. A higher weight on the heuristic can speed up the search but risks missing shorter paths.
- **Grid Resolution:** The resolution of the grid in A* affects the granularity of the path. Higher resolution offers more detailed paths but at the cost of increased computational complexity.
- **Cost Function:** Variations in the cost function can lead to different path choices.

What happens if you inflate the heuristic?

Inflating the heuristic in the A* algorithm transforms it into Weighted A*, a variant that prioritizes search speed over path optimality. This adjustment leads to the algorithm favouring nodes that appear closer to the goal, potentially reducing the overall search time by exploring fewer nodes. However, this efficiency comes at a cost: the paths found may not be the shortest possible, as the algorithm tends to focus more on goal direction than on exploring alternate routes that might offer shorter distances. The degree of heuristic inflation determines the balance between search speed and path optimality.

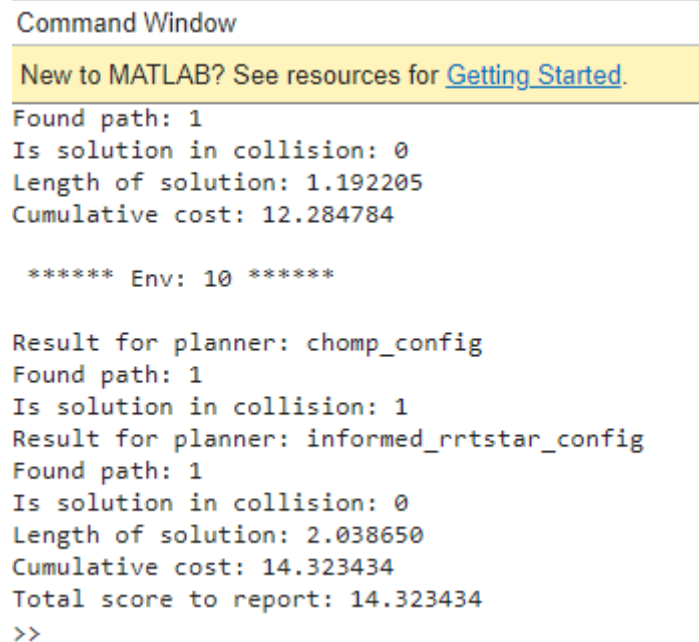
How do you turn RRT* into RRT?

To transform RRT* into RRT, the algorithm must focus on rapid space exploration rather than path optimization. This involves removing RRT*'s path optimization steps, which refine the tree by seeking shorter connections. In RRT, new nodes are directly connected to the nearest existing node in the tree, without considering potential path improvements or rewiring the tree for lower costs. This change shifts the algorithm's purpose from finding the most efficient path to efficiently exploring and mapping the space.

Table 2: Algorithms Functionality

	CHOMP	A*	RRT*
Environment 1	Yes	Yes	Yes
Environment 2	No	Yes	Yes
Environment 3	No	Yes	No

Planning Challenge



```

Command Window

New to MATLAB? See resources for Getting Started.

Found path: 1
Is solution in collision: 0
Length of solution: 1.192205
Cumulative cost: 12.284784

***** Env: 10 *****

Result for planner: chomp_config
Found path: 1
Is solution in collision: 1
Result for planner: informed_rrtstar_config
Found path: 1
Is solution in collision: 0
Length of solution: 2.038650
Cumulative cost: 14.323434
Total score to report: 14.323434
>>

```

Figure 6: Screenshot of the Command Window in MATLAB

The 2 planning algorithms used were.

- chomp_config
- informed_rrtstar_config

The total score achieved while running the run_planner_challenge.m was 14.323434 as shown in Figure 6.

Conclusion

This project provides an insightful exploration into the use of advanced pathfinding algorithms for UAS for package delivery, employing like Dijkstra's and A*. This project shows our proficiency in applying theoretical algorithmic concepts to practical scenarios but also highlighted the nuances and differences between Dijkstra's and A* algorithms. Furthermore, the project extended our understanding of algorithmic planning through the exploration of various planning algorithms using a MATLAB Toolboxes, challenging us to develop and analyse different algorithms.