Terrain-Constrained A* Algorithm for Autonomous Navigation of Tracked Vehicles in High-Relief Off-Road Terrains

Mullainathan V H

Sivatmiga T N

BACHELOR OF ENGINEERING

Branch: ROBOTICS AND AUTOMATION

May 2025

CENTRE FOR ARTIFICIAL INTELLIGENCE AND ROBOTICS

DEFENCE RESEARCH AND DEVELOPMENT ORGANISATION

Bangalore, India

ACKNOWLEDGEMENT

We are deeply grateful for the successful completion of this project, which would not have been possible without the guidance and support of several individuals. We sincerely thank them for their help throughout the project and acknowledge their role in its completion.

We would like to express our heartfelt gratitude to **Dr. K. Prakasan**, our Principal, for granting us the opportunity to undertake this project and for his continuous support and direction.

We are especially thankful to **Dr. B. Vinod**, Professor, Head of the Department, and Project Guide, Department of Robotics and Automation Engineering, for his exceptional guidance and insightful suggestions throughout the project. His dedicated mentorship, steady encouragement, and timely assistance were instrumental in successfully completing this work.

We extend our sincere appreciation to our Industrial Project Guide, **Mr. Shibumon Alampata**, Scientist – F, Geographical Information Systems Division, Centre for Artificial Intelligence and Robotics, Defence Research and Development Organization, for his invaluable mentorship, technical expertise, and continuous support. His insightful feedback and guidance played a crucial role in shaping the project's direction and ensuring its success.

We are grateful to **Dr. S. Prabhakaran**, Assistant Professor and Tutor, Department of Robotics and Automation Engineering, for his consistent support during both the initial and final stages of the project.

Finally, we extend our thanks to the entire teaching and non-teaching staff of the Department of Robotics and Automation Engineering for their ongoing support, motivation, and cooperation, which significantly contributed to the successful execution of this project.

SYNOPSIS

Autonomous navigation in off-road and unstructured terrains is a critical challenge in military logistics, defense deployment, and exploration scenarios. Classical A* path planning algorithms, although effective in structured grid environments, fail to account for real-world terrain constraints such as elevation gain, slope severity, and physical traversability. This project aims to address these shortcomings by developing a terrain-aware A* path planning system capable of navigating tracked tanker vehicles across high-relief environments like the Eastern Himalayas, where traditional roads are non-existent.

The system incorporates Digital Elevation Models (DEMs) and slope rasters to generate GO/NO-GO terrain masks using user-defined elevation and slope thresholds. A refined A* algorithm is implemented with subpixel search resolution and 32-directional angular movement, improving both path smoothness and realism. A PyQt5-based graphical interface enables users to interactively load terrain data, define planning constraints, visualize computed paths, and export elevation and slope profiles.

The system was evaluated across five test cases with varying start and goal locations. Three variants—Simple A*, Resampling A*, and Terrain-Constrained A*—were benchmarked using metrics such as path length, elevation gain, average and maximum slope, slope violations, and compliance percentage. The Terrain-Constrained A* consistently produced safe and realistic paths with 100% slope compliance, validating its suitability for heavy tracked vehicles. While it incurred higher computation time, the safety and feasibility of generated paths make it ideal for defense and tactical applications.

The project demonstrates that integrating terrain awareness into A* planning significantly enhances its deployability in real-world, off-road scenarios. Future work will focus on incorporating additional terrain parameters such as soil load-bearing capacity, water body depth, and dynamic terrain updates to support real-time, mission-critical navigation.

CONTENTS

CHAPTER **Page No.**

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

S. No.	Abbreviation	Abbreviated Term
1	A*	A-Star Algorithm
2	DEM	Digital Elevation Model
3	GUI	Graphical User Interface
4	GEE	Google Earth Engine
5	QGIS	Quantum Geographic Information System
6	GDAL	Geospatial Data Abstraction Library
7	SLAM	Simultaneous Localization and Mapping
8	ORB-SLAM3	Oriented FAST and Rotated BRIEF SLAM version 3
9	MPC	Model Predictive Control
10	DWA	Dynamic Window Approach
11	UGV	Unmanned Ground Vehicle
12	PTFM	Probabilistic Terrain Feasibility Map
13	PRM	Probabilistic Roadmap
14	IDW	Inverse Distance Weighting
15	CSV	Comma-Separated Values
16	SRTM	Shuttle Radar Topography Mission
17	PyQt	Python Qt Toolkit
18	CPU	Central Processing Unit
19	LIDAR	Light Detection and Ranging
20	ROS	Robot Operating System

CHAPTER 1

INTRODUCTION

This chapter provides an overview of the motivation, challenges, and scope of autonomous path planning in off-road and unstructured terrain environments. It outlines the limitations of traditional A* algorithms when applied to high-resolution digital elevation models (DEMs), where terrain constraints such as slope and elevation changes significantly affect feasibility. The chapter introduces terrain-aware enhancements implemented in this project, including slope filtering, subpixel resolution planning, and GUI-based interaction, laying the foundation for in-depth technical development and evaluation presented in subsequent chapters.

1.1 BACKGROUND

Autonomous navigation has become a crucial capability in military logistics, disaster response, and environmental exploration. In regions such as mountainous terrains, riverine floodplains, and polar deserts, conventional road infrastructure is either unreliable or non-existent. Navigation through such terrains requires robust path planning systems capable of leveraging terrain features like elevation, slope, and load-bearing properties.

Traditional A* algorithms, though widely used in structured indoor and urban environments, are inadequate for unstructured or off-road scenarios due to their limited ability to account for terrain gradients or physical mobility constraints. With the increasing deployment of tracked military tankers and heavy unmanned ground vehicles (UGVs), path planning algorithms must ensure safety, efficiency, and kinematic feasibility under challenging terrain.

1.2 PROBLEM STATEMENT

Classical A* path planning algorithms do not consider real-world terrain factors such as elevation changes or slope steepness. This results in impractical paths with sharp turns, terrain collisions, and unrealistic elevation profiles. Furthermore, applying these algorithms

directly on high-resolution DEMs leads to large search spaces and high computation costs. They also lack mechanisms to filter out unsafe paths such as those crossing steep gradients or unstable terrain.

To address these limitations, a terrain-constrained path planning system is needed that uses elevation and slope data to generate paths feasible for heavy tracked vehicles. The system should support configurable filters for slope limits, and generate exportable path data for further analysis or integration with tactical navigation systems.

1.3 MOTIVATION

Real-world applications such as logistics deployment in defense, border surveillance, autonomous supply delivery in inaccessible terrains, and rescue missions require terrain-aware route planning. Tankers operating in snow-covered high-altitude zones or flooded plains must not only traverse optimal paths but also avoid unstable slopes or exceed their traction limits.

Although high-accuracy localization systems like ORB-SLAM3 exist, they are not coupled with terrain-aware planners. This project aims to bridge this gap by developing an A* variant tailored for DEM-based terrains, capable of subpixel accuracy, slope enforcement, and full path export with slope diagnostics. The project lays groundwork for future improvements in real-time feasibility checks, soil/load-bearing filtering, and hydrological layer analysis.

1.4 OBJECTIVES OF THE PROJECT

The objectives of the project are as follows:

1 **Terrain-Constrained Path Planning:** To design and implement an A* algorithm that integrates slope-based GO/NO-GO masking for effective path planning on DEMs.

2 **Algorithmic Enhancements:** To apply subpixel search, 32-directional movement, and terrain-safe heuristics for smoother and more realistic paths.

3 **Visualization and Export:** To develop a PyQt-based GUI that allows raster loading, start/goal interaction, path computation, and elevation/slope visualization. The GUI must export all paths as CSV with slope and elevation.

4 **Performance Analysis:** To compare three A* variants (Simple A*, Resampled A*, Terrain-Constrained A*) over five scenarios and evaluate their performance using metrics like computation time, elevation gain, slope violations, and compliance.

5 **Hybrid Planning Strategy:** To include search space pruning methods such as morphological dilation and coarse planning (PRM-inspired) to reduce computational load and guide fine planning stages.

1.5 SCOPE OF THE PROJECT

The scope of this study is limited to high-altitude terrain in the Eastern Himalayas using SRTM-based DEM and slope rasters. The planning algorithm runs entirely in 2D using raster constraints without integrating full vehicle kinematics. However, terrain slope thresholds are used to mimic tracked vehicle constraints (e.g., ±35°), and the GUI supports elevation and slope visual debugging.

The results validate that slope-filtered, subpixel A* outperforms basic A* methods in terrain feasibility. Future extensions may include integrating additional layers like soil bearing capacity, water body depth models, vegetation density, and permafrost masks for improved reliability in defence applications.

1.6 A* ALGORITHM OVERVIEW

The A* algorithm is a well-established pathfinding technique that computes the optimal route between two points in a graph by balancing path cost and heuristic estimates. Its cost function is defined as:

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

Where:

$\boldsymbol{g(n)}$ represents the exact cost from the start node to the current node.

$\boldsymbol{h(n)}$ is a heuristic estimate of the cost from the current node to the goal.

1.6.1 Working Principle

A* maintains two lists: the **open list** (nodes yet to be evaluated) and the **closed list** (already evaluated nodes). At each step, the node with the lowest f(n) is chosen. If this node is the goal, the path is reconstructed; otherwise, its neighbors are added to the open list with updated cost estimates. This continues until a path is found or all options are exhausted.

1.6.2 Heuristics in Grid Environments

The choice of heuristic directly affects the algorithm's performance:

- **Manhattan distance** is used when movement is limited to vertical and horizontal directions.
- **Euclidean distance** is suitable for diagonal or continuous movement.
- **Octile distance** approximates cost in grid-based maps allowing 8-directional movement.

1.6.3 Relevance and Challenges in Off-Road Navigation

While A* is efficient in structured environments, it has limitations when applied to terrain-based, real-world applications. It often generates paths with:

- Sharp turns or frequent direction changes, which are difficult for tracked vehicles.
- Close obstacle hugging, increasing the risk of collisions due to poor clearance.
- Lack of consideration for vehicle kinematics, such as turning radius and directional constraints.

1.6.4 Adaptations for Terrain and Vehicle Constraints

To improve applicability in off-road environments, several strategies have been proposed:

- **Path smoothing:** Removes sharp corners post-planning.
- **Obstacle inflation:** Ensures safer margins by enlarging obstacle boundaries.
- **Hybrid A*:** Incorporates vehicle motion models and continuous state evaluation for smoother, kinematically feasible paths.
- **Modified heuristics:** Penalize abrupt direction changes.
- **Dynamic replanning:** Algorithms like D* Lite or DWA offer adaptability in changing environments.

In this project, although Hybrid A* was explored conceptually, only selective principles (e.g., slope-aware cost modeling and directional filtering) were implemented due to system constraints in the interface. These adaptations were instrumental in improving path feasibility for tracked vehicle operations in real terrains.

1.7 ORGANIZATION OF THE REPORT

This report is structured into six comprehensive chapters that progressively build the context, methodology, implementation, and evaluation of the proposed terrain-constrained path planning system. Each chapter is organized to ensure clarity, logical flow, and depth of technical detail to support academic, engineering, and defense-oriented perspectives.

- **Chapter 1 – Introduction:** Introduces the context and significance of terrain-aware autonomous path planning, especially in defense and off-road environments. It outlines the problem with classical A* in complex terrains and presents the motivation, objectives, scope, and a brief overview of A* adaptations relevant to this study.

- **Chapter 2 – Literature Survey:** Reviews prior research on terrain-based path planning, digital elevation models (DEMs), slope analysis, and hybrid methods such as PRM and Hybrid A*. It identifies gaps in slope-constrained navigation and the integration of real-world terrain data into planning algorithms.
- **Chapter 3 – System Architecture and Design:** Describes the architecture of the proposed system, including data flow from terrain raster preprocessing to path computation. It covers components like GO/NO-GO masking, slope filtering, and the hybrid planning logic, supported by design diagrams.
- **Chapter 4 – Implementation:** Presents the complete development workflow, including terrain data loading, A* algorithm variants, and GUI features. It explains how the system integrates subpixel search, 32-directional movement, slope constraints, and path visualization. Scripts for exporting results and performance metrics are also described.
- **Chapter 5 – Results and Evaluation:** Details the testing of Simple A*, Resampling A*, and Terrain-Constrained A* across five start–goal pairs. Metrics such as path length, elevation gain, slope compliance, and computation time are analyzed with visual outputs and summary tables. A comprehensive comparison is provided to evaluate each variant’s practicality.
- **Chapter 6 – Conclusion and Future Work:** Summarizes the effectiveness of terrain-constrained A* for safe and feasible navigation in unstructured terrains. It outlines future improvements such as the inclusion of soil load-bearing data, water body avoidance logic, and integration with real-time SLAM and onboard computing for defense logistics.

CHAPTER 2

LITERATURE REVIEW

This chapter presents a comprehensive review of prior research and methodologies relevant to terrain-aware path planning using the A* algorithm. It begins by examining the classical A* framework and its adaptations for unstructured environments, followed by a survey of enhancements such as slope-aware cost modeling, any-angle path planning (Theta*), and hybrid strategies involving Probabilistic Roadmaps (PRM). The chapter also explores the role of terrain modeling using Digital Elevation Models (DEMs), integration with GIS tools, and localization systems like ORB-SLAM3. Through this review, the strengths, limitations, and research gaps in existing approaches are identified, forming the foundation for the design and implementation of the proposed system.

2.1 OVERVIEW OF TERRAIN-AWARE PATH PLANNING

Path planning in off-road or unstructured terrains has emerged as a crucial area in autonomous navigation due to its wide-ranging applications in military operations, planetary exploration, and rescue missions. Unlike structured urban environments, off-road terrain introduces complex navigational constraints like elevation differences, steep slopes, and varied surface conditions. Classical path-planning methods, such as the A* algorithm, require adaptations and enhancements to effectively handle these complexities. Recent research has increasingly focused on integrating terrain characteristics into planning algorithms, improving both path feasibility and computational efficiency.

This chapter explores key research advances in terrain-aware path planning, specifically highlighting enhancements made to the A* algorithm and related hybrid strategies. A detailed review is conducted on slope and elevation-aware planning, any-angle path planning, and strategies for integrating A* with Probabilistic Roadmap (PRM), vehicle kinematics, SLAM localization, and GIS-based terrain modeling.

2.2 CORE A* ALGORITHM AND VARIANTS

The A* algorithm is a graph-based search method widely used for its simplicity, determinism, and ability to find optimal paths using heuristic evaluations. However, classical A* faces critical challenges in unstructured terrains, prompting the development of specialized variants.

- **Slope-aware A*** [1] significantly improves traditional A* by introducing terrain slope information directly into cost calculations. This method employs a hybrid heap and array data structure, reducing complexity and speeding up the algorithm by approximately 550 times compared to traditional implementations. The bidirectional search approach further optimizes computational performance by narrowing down the search space.
- **Theta*** [3], an important variant of A*, addresses the inherent zig-zag artifact seen in classical grid-based A*. Unlike conventional methods that limit movement directions to grid adjacencies, Theta* uses line-of-sight checks to connect nodes directly, enabling smoother, shorter, and more realistic paths, thus achieving any-angle path planning.
- **LIAN Algorithm (Angle-Constrained A*)** [4] introduces explicit angular constraints between nodes, enforcing smoother transitions and more realistic trajectories suitable for wheeled robots and UAVs. While paths generated by LIAN are not guaranteed to be the absolute shortest, they ensure feasibility regarding vehicle maneuverability.

Algorithm Variant	Terrain Awareness	Slope Constraint	Path Smoothness	Computational Cost	Real-world Feasibility
Classical A*	No	No	Poor (zig-zag)	Low	Low – ignores physical constraints
Resampling A*	Indirect via finer grid	No	Improved but unsafe	Moderate	Moderate – lacks terrain logic
PRM + A*	Moderate	Optional	High (with post-process)	Low (due to pruning)	High with hybrid enhancements
Hybrid A*	High	Partial (with motion models)	Very High	High	High – used in mobile robotics
Terrain-Constrained A*	High (DEM + Slope)	Yes (≤ 35°)	High (subpixel, smooth)	High	Very High – ideal for defense UGV

Fig 2.1: Comparative table of A* variants and their terrain integration

2.3 TERRAIN MODELING APPROACHES

Accurate terrain modeling is foundational to effective off-road path planning. Various researchers have introduced terrain-specific cost models within the A* algorithm framework to address terrain complexity realistically.

- **Wu et al. (2019)** [5] presented an improved A* approach tailored for planetary rovers. They incorporated travers ability metrics based on slope, roughness, and obstacle height derived from terrain data, significantly enhancing path safety and navigability on

lunar and Mars-like environments. This demonstrated the effectiveness of terrain-integrated cost maps in challenging extraterrestrial scenarios.

- **Saad et al. (2021)** [6] introduced a composite metric routing method that integrates path length and energy consumption for path planning on natural terrains. Although their results showed that composite metrics produced better energy-efficient paths compared to classical A*, they acknowledged that the integration of real-time terramechanics could further optimize path quality and practicality.
- The paper on **GIS and AI in Military Operations** [10] emphasizes the integration of Geographic Information Systems (GIS) to generate terrain maps incorporating elevation, slope, ruggedness, and hydrological data. Although A* was not explicitly discussed, the authors underscore how such detailed terrain information substantially supports path-planning algorithms by providing richer cost maps and navigability insights.

2.4 ENHANCEMENTS AND HYBRID STRATEGIES

Several studies have explored hybrid path-planning frameworks, combining A* with auxiliary algorithms to mitigate computational limitations and improve path quality:

- **The Two-Stage PRM + A* method proposed by Zheng et al.** (2024) [2] uses a hybrid approach, initially employing a non-uniform Probabilistic Roadmap (PRM) algorithm to rapidly produce a coarse initial path. The PRM-generated path is then refined using morphological dilation, followed by precision planning via A*. This approach drastically reduces computational complexity (up to 60× improvement) while maintaining path optimality in complex terrains.
- **The KTH Thesis on Hybrid A*** [7] describes extending A* to incorporate vehicle kinematic constraints and real-time planning requirements, demonstrating the generation of drivable trajectories specifically for car-like robotic vehicles in semi-structured environments. Although computationally heavier than traditional A*, the Hybrid A* variant showed significant improvements in path realism and drivability, highlighting its potential for real-time autonomous vehicle navigation.
- Studies on autonomous navigation for tracked vehicles, such as the **Komodo UGV and GPS-based UGV systems** [8], noted that classical A* algorithms, while optimal, were computationally burdensome in real-time scenarios. They found model predictive control (MPC) algorithms more suitable for rapid, reactive decisions in dynamic

environments, suggesting that a hybrid approach using A* for strategic-level planning and MPC for tactical, real-time adjustments could enhance navigational effectiveness.

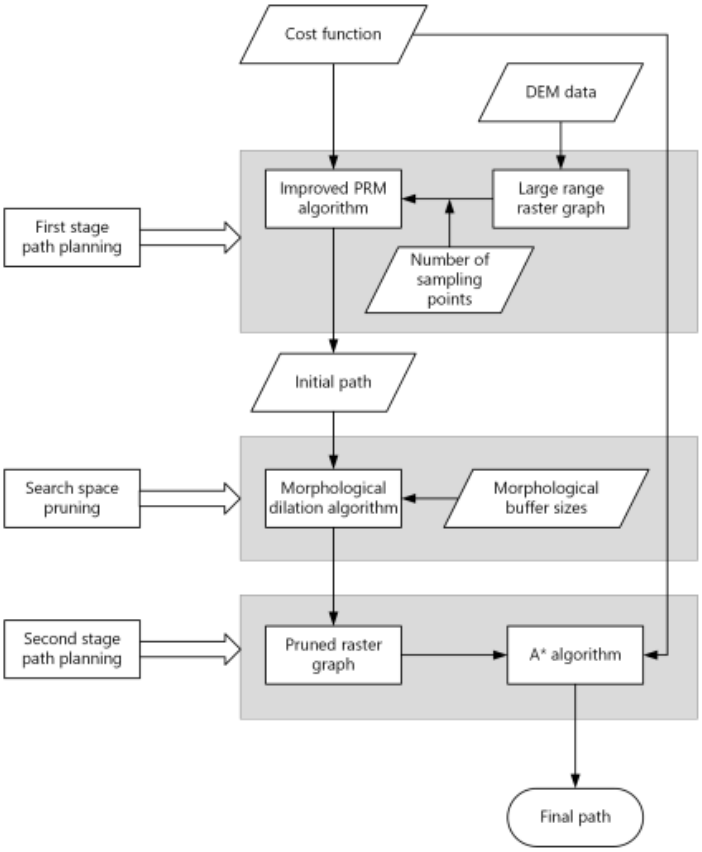

Fig 2.2: Diagram of PRM + A* two-stage planning

2.5 SLAM AND GIS SUPPORT FOR A*

Effective path planning in unstructured terrains is significantly bolstered by advanced localization systems and detailed terrain mapping:

ORB-SLAM3 [9] provides a highly accurate simultaneous localization and mapping (SLAM) framework. Though ORB-SLAM3 itself is not a planning algorithm, its precise localization outputs enable terrain-aware planners like A* to maintain accurate positioning within complex off-road environments, especially beneficial for multimap navigation scenarios.

GIS integration further enhances A*-based path planning by supplying detailed and accurate terrain maps. Comprehensive elevation and slope data from GIS are essential inputs for cost calculations in terrain-aware A* variants, enabling more informed and realistic pathfinding decisions, particularly beneficial for military logistics and disaster-response missions [10].

2.6 SUMMARY AND RESEARCH GAPS

This literature survey identifies significant advancements in terrain-aware path planning, specifically highlighting enhancements to A*, hybrid methodologies, and supportive localization and mapping technologies. Key insights include:

- Classical A* is computationally expensive and impractical for large, complex terrains without modifications.
- Hybrid approaches (PRM + A*, Hybrid A*, Theta*) provide significant computational improvements and generate more realistic paths.
- Explicit integration of terrain features like slope, elevation, and energy cost substantially improves path feasibility and operational practicality.
- Localization and mapping technologies (e.g., ORB-SLAM3, GIS) provide critical support, improving planner accuracy and navigational reliability.

However, despite these advancements, several research gaps remain:

- Real-time adaptivity to dynamic terrain changes (e.g., obstacles, surface conditions) within the A* framework remains limited.
- Effective integration of multiple optimization criteria (energy consumption, risk assessment, soil conditions) has yet to be fully explored.
- Further enhancements in computational efficiency, such as GPU acceleration and smarter sampling or pruning techniques, could significantly improve scalability.

Addressing these gaps will be crucial to advancing terrain-aware autonomous navigation technologies and widening their applicability in diverse, mission-critical domains.

CHAPTER 3

SYSTEM ARCHITECTURE AND DESIGN

This chapter outlines the overall system architecture and design strategy for the terrain-constrained A* path planner. It describes the modular structure of the system, including terrain data preprocessing, the core A* planning logic, and visualization components. Key processes such as DEM and slope map generation, GO/NO-GO area masking, cost function design, and heuristic formulation are detailed. The chapter also introduces enhancements like morphological dilation, PTFM integration, and support for advanced A* variants such as Theta*. These components collectively form a scalable and adaptable framework for realistic path planning in unstructured terrains.

3.1 OVERALL SYSTEM ARCHITECTURE

The proposed terrain-constrained A* path planner is structured into three primary functional modules to ensure effective handling of large-scale terrain data, efficient route computation, and intuitive visualization:

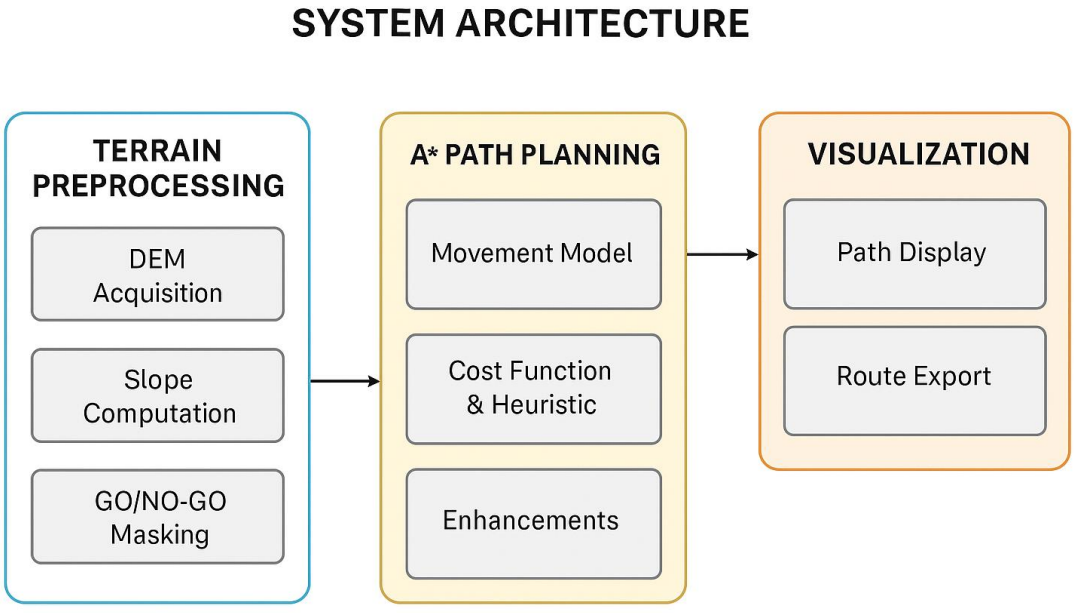

Fig 3.1: Block diagram of the system architecture

- Terrain Preprocessing Module: This module involves acquiring and processing terrain data, generating detailed Digital Elevation Models (DEMs), slope maps, and creating navigability masks (GO/NO-GO areas) based on terrain conditions.
- A Core Planning Module*: At the system's heart is the enhanced A* algorithm, designed to efficiently calculate optimal paths that account for terrain constraints, elevation changes, slope-based traversal costs, and strategic heuristic choices.
- Visualization and Route Output Module: Provides intuitive graphical interfaces for route visualization, DEM overlay, elevation profiling, and path export, facilitating interactive use and detailed analysis.

3.2 TERRAIN DATA ACQUISITION AND PREPROCESSING

3.2.1 DEM Sourcing Using Google Earth Engine (SRTM)

Digital Elevation Models (DEMs) serve as the foundational data layer for the terrain-aware path planning system. In this project, DEM data was sourced through Google Earth Engine (GEE) using the USGS/SRTMGL1_003 dataset, which offers a spatial resolution of 30 meters. The GEE platform enables efficient processing and export of large-scale elevation data without the need for local computation resources.

The DEM was accessed and processed using GEE's JavaScript API. After selecting the relevant DEM image and clipping it to the desired region, the data was exported in GeoTIFF format using *Export.image.toDrive*. This format ensured compatibility with GIS tools and local raster processing libraries used in subsequent modules.

- Source Dataset: USGS/SRTMGL1_003
- Resolution: 30 meters
- Export Format: GeoTIFF
- Processing Environment: Google Earth Engine

3.2.2 Slope Map Generation in Google Earth Engine

To evaluate terrain steepness and assess travers ability for path planning, a slope map was generated from the SRTM DEM using GEE's built-in *ee.Terrain.slope()* function. This function calculates the slope (in degrees) for each grid cell by analysing elevation change with respect to its surrounding cells.

The slope raster was then exported as a GeoTIFF to be used in offline analysis, such as:

- Enforcing slope-based traversal constraints in the A* planner.

- Filtering GO/NO-GO areas based on slope thresholds.
- Computing cost penalties associated with steeper regions.

This GEE-based approach allowed efficient extraction of both DEM and slope information in a reproducible, scalable workflow.

- **Method Used**: *ee.Terrain.slope()*
- **Resolution**: 30 meters (matched to DEM)
- **Output Format**: GeoTIFF
- **Purpose**: Input to terrain-aware cost functions and GO/NO-GO masking

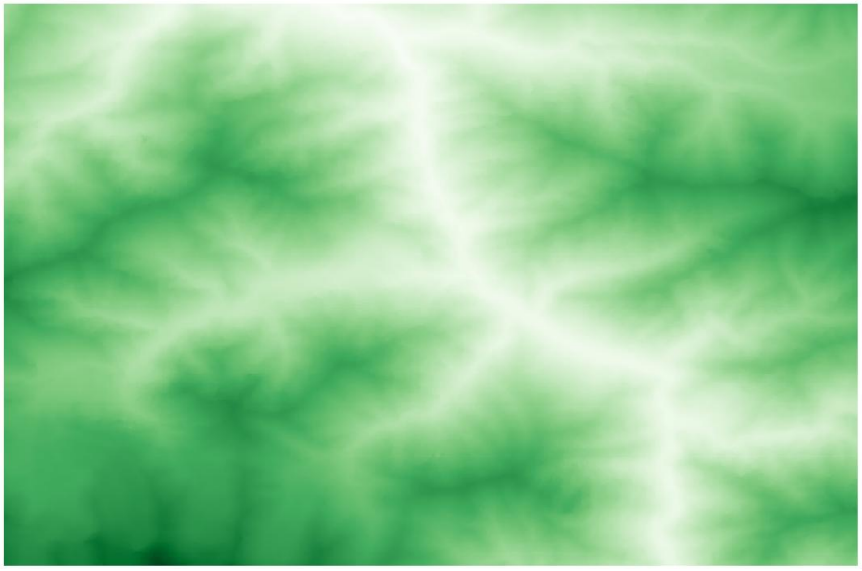

Fig 3.2: DEM and slope preprocessing workflow using GEE

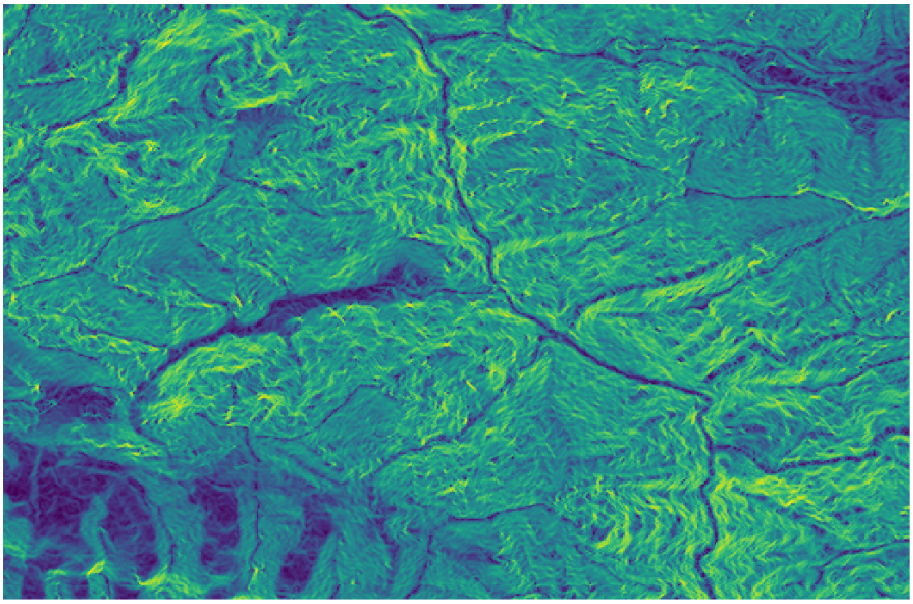

Fig 3.3: Example of slope raster generated from DEM using GEE

3.2.3 GO/NO-GO Masking Based on Elevation/Slope Thresholds

Once DEM and slope maps are available, the next step is to classify terrain travers ability through GO and NO-GO masking:

- Elevation Thresholds: Terrain elevations exceeding predefined operational limits are masked as NO-GO areas.
- Slope Thresholds: Cells with slope values beyond vehicle-specific thresholds (e.g., > 35° for typical military vehicles) are also classified as NO-GO.
- The resulting binary masks (GO = traversable, NO-GO = non-traversable) significantly narrow the search space for the A* planner, improving computational efficiency.

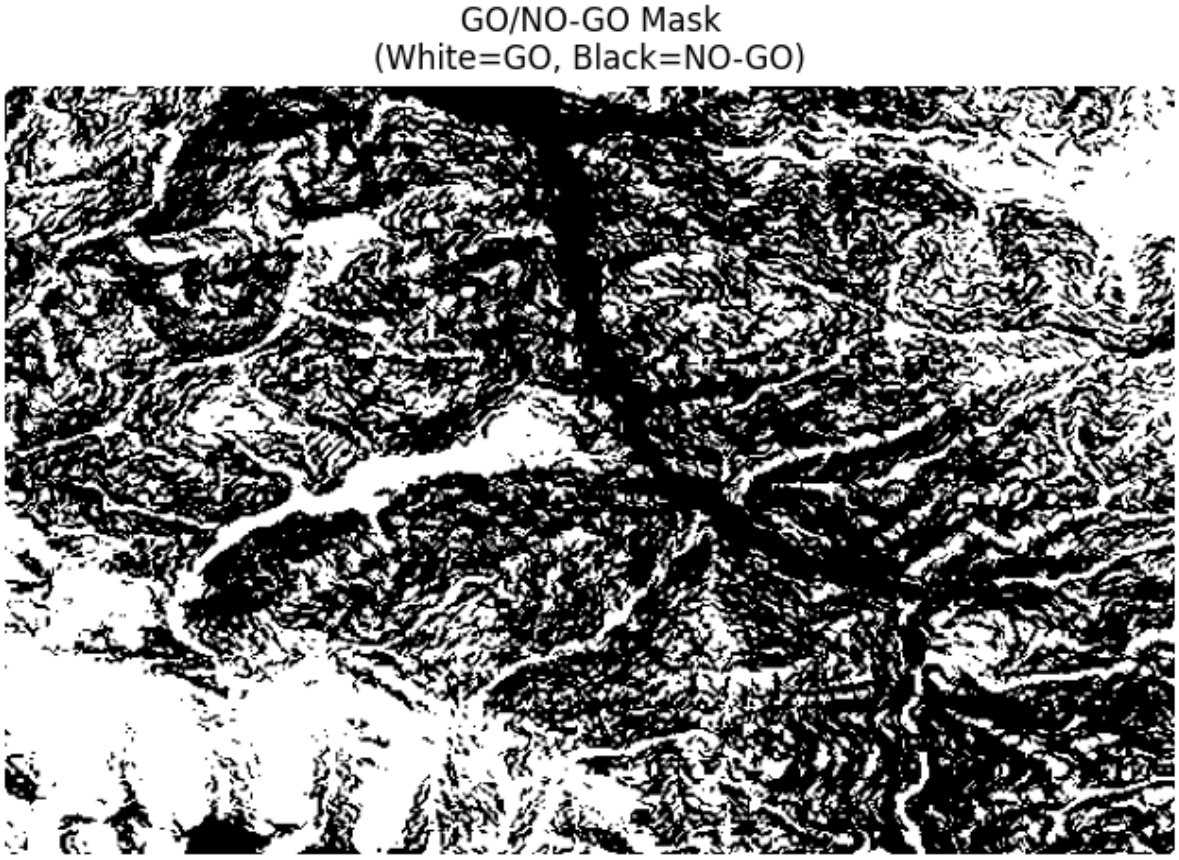

Fig 3.4: GO/NO-GO terrain masking based on elevation and slope

3.3 CORE DESIGN

3.3.1 Movement Model (8-directional, Subpixel)

The A* algorithm traditionally supports either 4-directional or 8-directional grid movements. This system adopts an 8-directional model for better realism. Furthermore, advanced implementations employ subpixel (continuous angle) path refinement, allowing smoother paths and eliminating grid-induced zig-zags, which enhances realistic navigation.

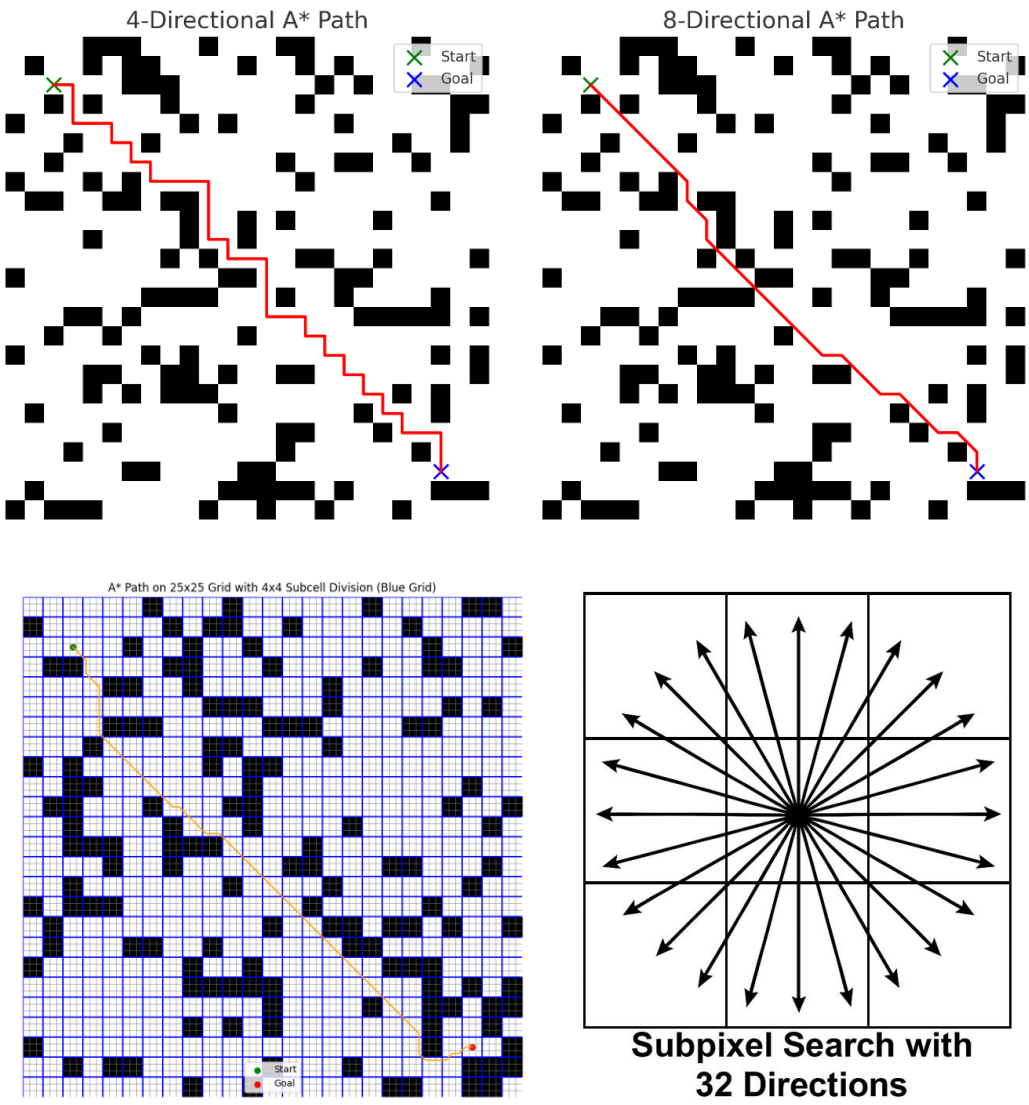

Fig 3.5 compares different A* movement models. The top-left image shows a 4-directional A* path with sharp, stair-like transitions. The top-right image depicts smoother movement enabled by 8-directional traversal. The bottom-left demonstrates subpixel path refinement using a 4×4 grid within each cell, while the bottom-right illustrates 32-directional angular search for high-resolution path smoothing. These models reflect increasing spatial resolution and directionality critical for fine-tuned navigation in unstructured terrains.

3.3.2 Cost Function Design (Slope Penalty, Euclidean)

The core of the A* algorithm's realism is the terrain-aware cost function. This is formulated as:

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

- $\boldsymbol{g(n)}$**:** Actual traversal cost from start to current node. Includes distance traveled (Euclidean or Manhattan) and terrain penalties (slope-related traversal costs).

- Slope Penalty: Traversal difficulty increases exponentially with slope steepness; hence, slope cost is included as a major penalty term using empirical formulas.
- Euclidean Distance: Provides straightforward and computationally efficient distance measurements.

3.3.3 Heuristic Choices

Effective heuristics significantly enhance A* efficiency. In this system:

- Euclidean distance heuristic: Default heuristic providing consistent and admissible estimates.
- Slope-aware heuristic: Weighted heuristics that integrate elevation differences, further guiding the search towards feasible paths.

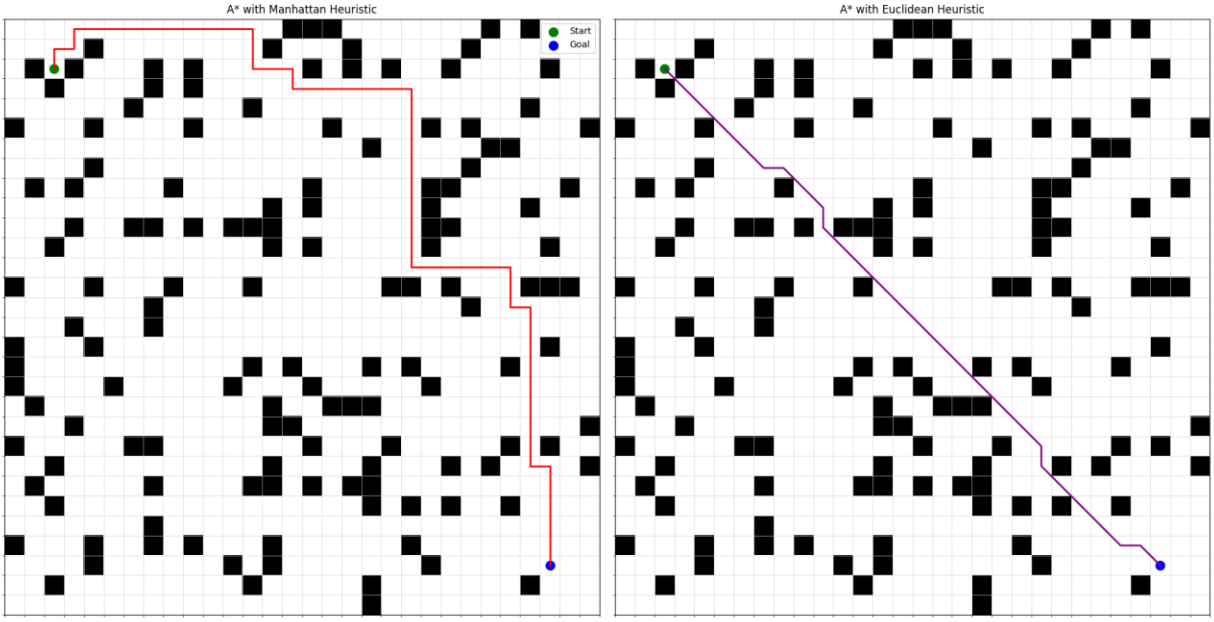

Fig 3.6: Illustration of heuristic influence on pathfinding

3.4 ENHANCEMENTS

3.4.1 Morphological Dilation and Search Space Pruning

To manage computational complexity on large DEMs, a morphological dilation technique is applied:

- Initial coarse path derived by a preliminary planner undergoes morphological dilation.
- A narrowed corridor (buffer) around this initial path serves as a refined search area for precise A* planning.
- Significantly reduces computational load without sacrificing path optimality.

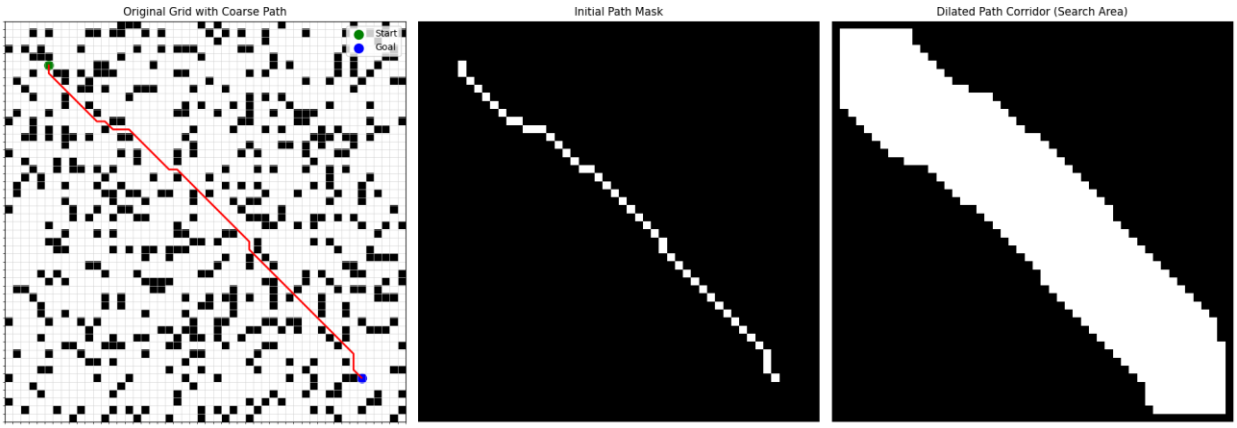

Fig 3.7: Morphological dilation to narrow down the search space

3.4.2 Integration with Probabilistic Terrain Feasibility Map

A terrain-constrained two-stage planning strategy is employed, using a probabilistic terrain feasibility map in conjunction with a refined A* algorithm:

- Stage 1 – Terrain Feasibility Mapping: A probabilistic costmap is generated from Digital Elevation Models (DEMs) and slope rasters. This map identifies traversable (GO) and non-traversable (NO-GO) regions based on elevation range and slope thresholds, effectively encoding local terrain safety and mobility constraints into a raster format.
- Stage 2 – Refined A*: A modified A* algorithm operates over this probabilistic map with subpixel resolution and 32-directional angular search, producing high-fidelity paths that respect both terrain topology and vehicular mobility constraints. This stage enhances path realism and computational efficiency by focusing the search within valid terrain zones and allowing smoother, more natural movement.

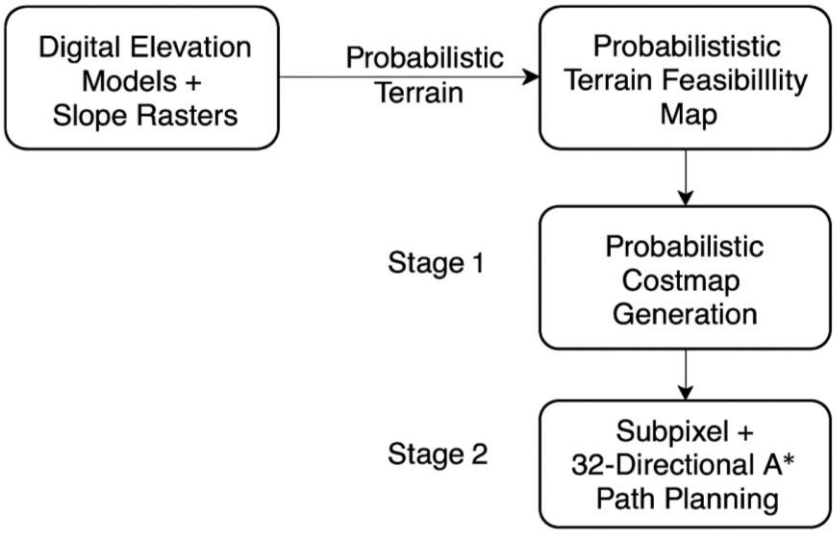

Fig 3.8: Workflow diagram of probabilistic terrain map-based A* planning

3.5 Visualization and Route Output

A comprehensive visualization module complements the planning algorithm, featuring:

- **Interactive GUI**: Built using PyQt5 and matplotlib, enabling intuitive interaction with DEMs, slope maps, and paths.
- **DEM Overlay**: Routes visualized directly on terrain data, allowing easy validation of path feasibility.
- **Elevation Profile Plots**: Generated automatically, detailing elevation gain/loss along the route for detailed path analysis.
- **Export Functions**: Provides route data export in standard GIS formats (e.g., CSV, Shapefile) for further analysis and integration with other systems.

CHAPTER 4

IMPLEMENTATION

This chapter describes the technical implementation of the terrain-aware A* path planning system. It elaborates on the software frameworks, terrain datasets, algorithmic design, GUI development, and visualization features that together form the core of the project. The chapter also presents a structured analysis of how different A* implementations were experimented with for comparative evaluation and includes references to code modules documented in Appendix A.

4.1 TOOLS AND FRAMEWORKS USED

The terrain-constrained A* path planning system is implemented using several widely-used open-source tools and frameworks, ensuring flexibility, interoperability, and ease of maintenance:

- Python 3.11: Chosen for its powerful libraries, readability, and rapid prototyping capabilities.
- PyQt5: Provides an intuitive and interactive graphical user interface (GUI) framework, supporting real-time visualization of DEM and computed paths.
- QGIS: Utilized for geospatial data preprocessing, DEM visualization, and generating slope maps and masks.
- Rasterio and GDAL: Facilitate efficient geospatial data handling, raster file management (DEM, slope data), and transformations.
- NumPy and SciPy: Essential for efficient numerical operations, DEM data manipulation, morphological dilation, and fast terrain cost calculations.
- Matplotlib: Used extensively for plotting route visualizations, elevation profiles, and heuristic visual debugging.

4.2 DESCRIPTION OF TERRAIN DATASETS

The DEM and slope raster inputs were obtained from publicly available sources, particularly the SRTM and ALOS datasets. Preprocessing was done in GEE to compute slope

using the ee.Terrain.slope() operator and resample all data to a unified resolution. For the implementation, a selected region was extracted, and slope and elevation constraints were applied. No ground truth data was used for validation, but elevation profiles and slope overlays were analyzed for assessing terrain realism.

4.3 A* ALGORITHM IMPLEMENTATION

The A* algorithm is implemented across three progressively refined variants. Each version was tested on the same DEM region to study its effectiveness under various conditions.

4.3.1 Data Loading and Grid Construction

DEM and slope rasters are read using Rasterio into NumPy arrays. Each cell in the raster corresponds to a node in the terrain grid, storing its elevation and slope value. A PyQt5-based GUI allows the user to define operational thresholds for elevation. Based on these thresholds, a binary mask is computed where cells are marked valid (GO) or invalid (NO-GO). This mask is used to limit the A* search to feasible regions only.

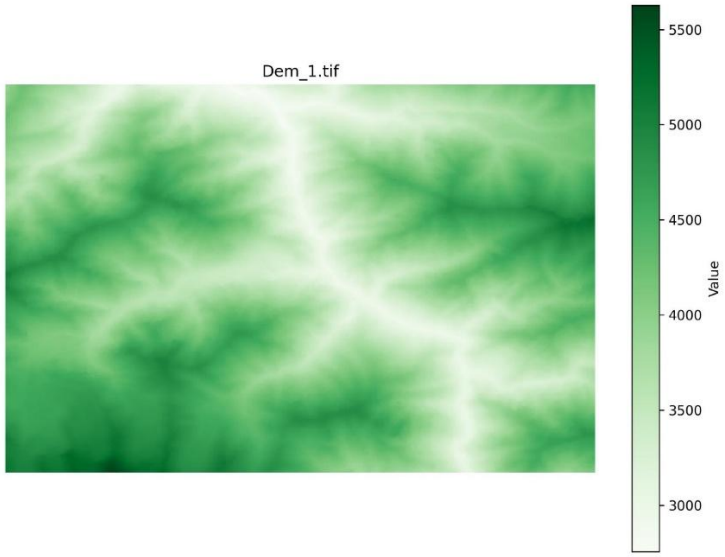

Fig 4.1: DEM raster grid

4.3.2 Cost and Heuristic Formulation

The movement cost, or g(n), is calculated using Euclidean distance between successive points. This supports diagonal and smooth paths when subpixel or 32-directional models are used. The heuristic, h(n), is also based on Euclidean distance to ensure admissibility. In the final implementation, slope values are used as hard constraints (filtering invalid cells) but not directly penalized in the cost function, maintaining computational efficiency.

4.3.3 Terrain Constraint Enforcement

The GO/NO-GO mask is constructed by filtering cells using both elevation and slope thresholds. Elevation is restricted between user-defined minimum and maximum bounds, while slope is limited to a maximum of 35 degrees. This preprocessing step eliminates non-traversable terrain before path planning, significantly reducing the search space and enhancing navigability.

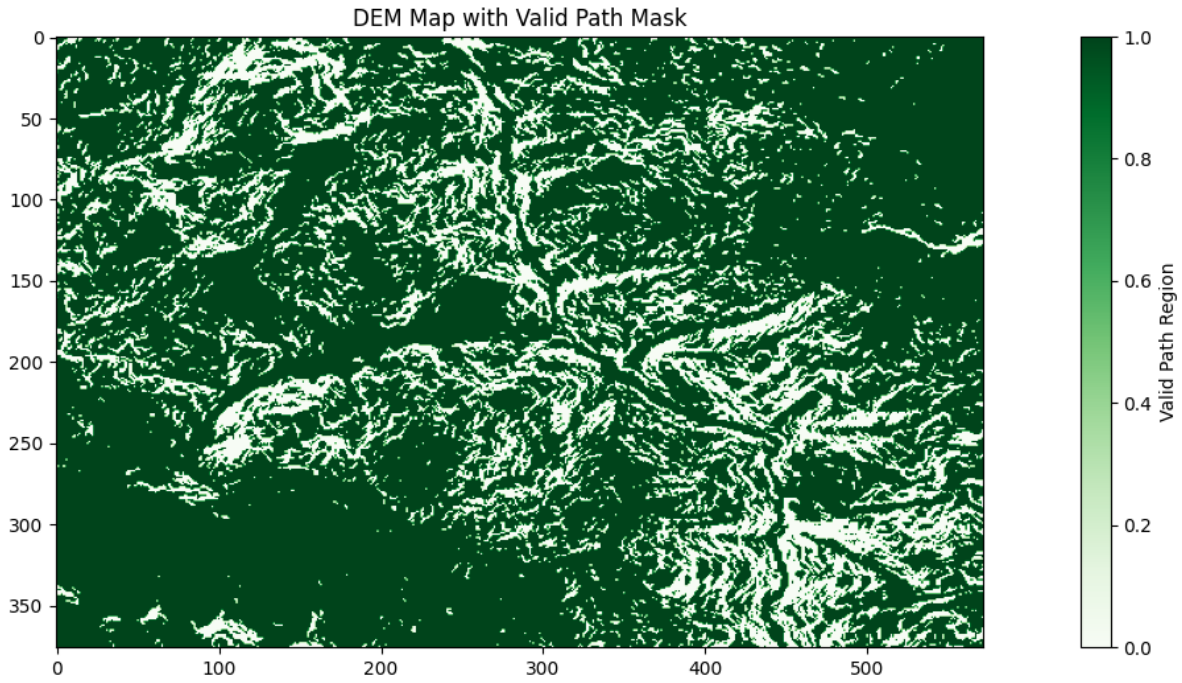

Fig 4.2: Visualization of GO/NO-GO masks applied to terrain data

The full implementation of the A* path planning system, including GUI integration and subpixel resolution logic, is provided in **Appendix A**.

4.4 TWO-STAGE PLANNING WORKFLOW

As introduced in Chapter 3, the system adopts a terrain-constrained two-stage planning approach. First, a **probabilistic terrain feasibility map** is constructed by masking terrain using elevation and slope. This forms the base surface for planning. Next, a refined A* planner with subpixel resolution and 32-directional search operates over this constrained region.

During preprocessing, DEM and slope rasters are filtered and resampled. The valid terrain is represented as a binary matrix, which acts as an admissible search space. The A* planner uses this map to limit exploration and improve efficiency. The two-stage design separates constraint modelling from algorithmic logic, making the system modular and easier to optimize.

4.5 GUI AND VISUALIZATION FEATURES

The system includes a custom-built GUI developed using PyQt5, which facilitates interactive terrain inspection, path planning execution, and result visualization. Users can load DEM and slope rasters, apply terrain constraints, and define start and goal points through an intuitive mouse interface. Once the path is computed, the interface overlays the route onto the terrain mask in real-time, allowing users to assess its feasibility.

Key features include:

- Interactive loading of elevation (DEM) and slope raster files in GeoTIFF format.
- Elevation threshold input via dialog boxes to define user-specific operational ranges.
- Visual overlay of the GO/NO-GO mask on the terrain background for contextual navigation.
- Real-time rendering of the computed A* path, with color-coded slope segments (green for ≤10°, yellow for 11–25°, red for 26–35°).
- Zoomed-in path visualization with terrain masking for local inspection.
- Automatic generation of an elevation profile with slope annotation along the path.
- Display of visited nodes to illustrate A* search coverage.

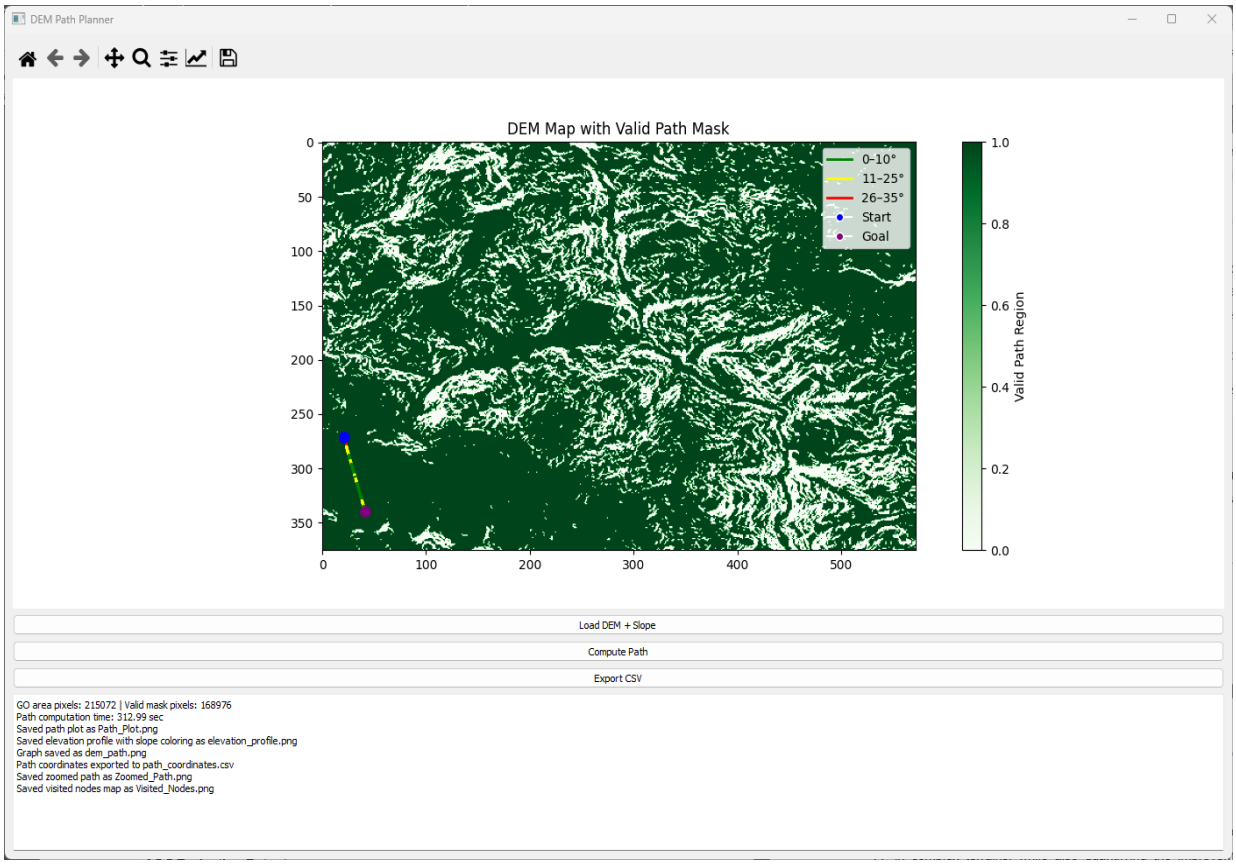

Fig 4.3: Screenshot of the GUI showing computed optimal path over DEM

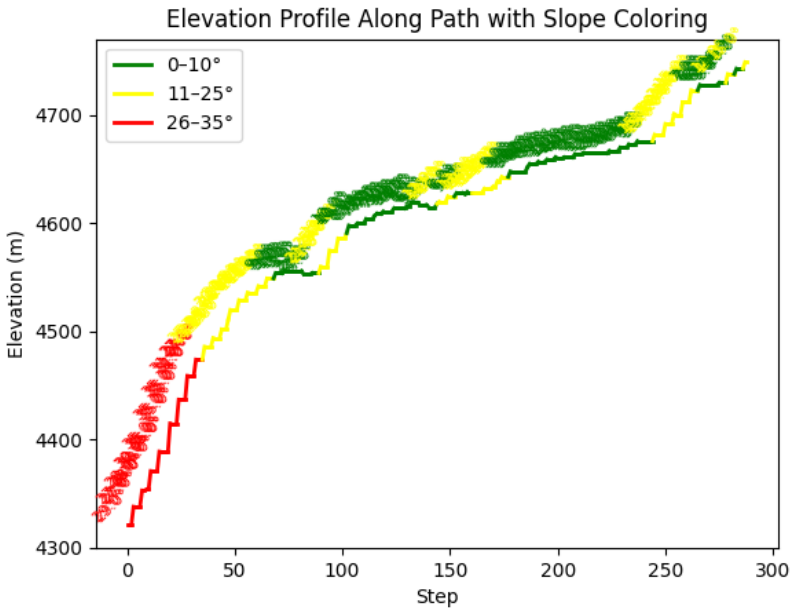

Fig 4.4: Elevation profile plot illustrating path elevation changes and slope ranges

4.6 PATH EXPORT AND EVALUATION MODULES

4.6.1 Path Export Module

The system supports exporting the final computed path for further analysis and integration. Upon successful path computation, the coordinates of the path are converted from pixel positions to geographic coordinates using the original raster transform.

- A CSV file is automatically generated (path_coordinates.csv) containing step number, latitude, longitude, and elevation at each path point.
- While the current implementation supports only CSV export, these coordinates can easily be imported into external GIS software such as QGIS or ArcGIS for further mapping and analysis.

4.6.2 Evaluation Outputs

In addition to path export, the system saves multiple evaluation outputs:

- **Path visualization image** (Path_Plot.png) showing the computed route overlaid on the terrain.
- **Zoomed-in path view** (Zoomed_Path.png) highlighting detailed terrain interaction.
- **Visited nodes plot** (Visited_Nodes.png) visualizing A* exploration coverage.
- **Elevation profile graph** (elevation_profile.png) with slope-based color coding and annotations for terrain assessment.

These outputs are saved automatically during the path planning process and serve as useful documentation for visual and terrain-aware validation.

4.7 Comparative Implementation of A* Variants

To assess the behaviour of A* under varying levels of terrain awareness and movement precision, three distinct implementations were developed and tested on the same digital elevation model with a fixed start and goal position. Each implementation was designed to isolate and highlight specific algorithmic factors such as resolution, directionality, and environmental constraint enforcement.

The first variant is a basic A* implementation using 8-directional grid-based movement over the raw DEM. This version does not enforce any slope or elevation filtering and treats all cells as traversable, revealing common path artifacts such as unnatural zig-zagging and terrain-agnostic routing.

The second implementation introduces subpixel interpolation by resampling the path resolution using fractional step updates. While terrain constraints are still ignored, this variant produces visually smoother and more direct paths, offering insight into the impact of fine-grained movement on path quality in a constraint-free environment.

The final and most advanced variant integrates terrain constraints by enforcing both elevation and slope thresholds through a probabilistic terrain mask. This mask defines GO/NO-GO regions, over which the refined A* search is performed using 32-directional movement and subpixel resolution. In this version, the algorithm achieves more realistic terrain-following behavior, producing routes that conform to both slope feasibility and elevation criteria.

The comparative study helped demonstrate the shortcomings of classical grid-based A* in complex terrains, while also quantifying the improvements introduced by direction expansion and environmental filtering. The code and GUI for all three implementations are included in **Appendix A** for reference and reproducibility.

CHAPTER 5

RESULTS AND EVALUATION

This chapter presents a detailed evaluation of three A* path planning variants—Simple A*, Resampling A*, and Terrain-Constrained A*—tested across five start–goal scenarios in a high-altitude region near the Eastern Himalayas. The terrain, characterized by steep slopes and elevation variability, serves as a realistic testbed for assessing navigation feasibility. Each variant was applied to the same DEM and slope datasets using the GUI-based system introduced in Chapter 4. Results were analyzed based on computation time, path length, elevation gain, and slope compliance, offering both quantitative metrics and visual diagnostics to compare algorithm performance under varying terrain constraints.

5.1 EXPERIMENTAL SETUP

All experiments were performed on a personal workstation equipped with an **Intel Core i5-11400H CPU @ 2.70GHz**, **16 GB RAM**, and running **Windows 11 (64-bit)**. The implementation environment included **Python 3.11**, with relevant libraries such as **NumPy**, **Pandas**, **Rasterio**, **Matplotlib**, **GDAL**, and **PyQt5**, executed via **Visual Studio Code**.

The terrain region selected for testing lies within the **Eastern Himalayan sector**, known for its challenging topography. A **30-meter resolution SRTM Digital Elevation Model (DEM)** was used as the elevation input, while the slope raster was computed from this DEM using **Google Earth Engine (GEE)**. Both rasters were exported in GeoTIFF format and served as inputs to the GUI-based path planning system.

To ensure uniformity across tests, the same terrain data was used in all five test cases. The only varying factors were the **start and goal pixel locations**, chosen to reflect realistic traversal conditions ranging from flat terrain to steep valleys and ridgelines. The system GUI allowed users to load terrain rasters, apply elevation and slope filters, and compute paths using selectable A* variants. Each path output was exported as a CSV with elevation and slope data, and was further analysed using a custom Python script for metric computation.

5.2 EVALUATION METRICS

To assess the performance and feasibility of each A* variant, the following evaluation metrics were used:

- **Computation Time (s)**: Time taken by the algorithm to compute the complete path, measured using Python's time module from start to completion.
- **Path Length (m)**: The total physical distance between the start and goal, calculated by summing the geodesic (Haversine) distances between successive path coordinates.
- **Elevation Gain (m)**: The cumulative sum of all positive elevation changes along the computed path. It reflects the vertical effort required for traversal.
- **Average Slope (°)**: The mean slope value along the path, computed only from slope values less than or equal to 35°, representing realistic terrain traversability.
- **Maximum Slope (°)**: The highest slope encountered along the path, useful for identifying extreme terrain conditions.
- **Slope Violations (>35°)**: The number of path points where the slope exceeds 35°, indicating potentially unsafe segments for tracked vehicles.
- **Slope Compliance (%)**: The percentage of the path that falls within the acceptable slope threshold (≤ 35°), used as a terrain feasibility indicator.

These metrics provide a balanced view of both **path efficiency** (length, time) and **terrain realism** (slope compliance), enabling meaningful comparison between variants.

5.3 COMPARISON OF A* VARIANTS

5.3.1 Test 1 Results:

- Star Pixel: (161, 266)
- Goal Pixel: (201, 153)

Table 5.1: Summary of Test 1 Evaluation Metrics Across Five Test Cases for Three A* Variants

Variant	Computation Time (s)	Path Length (m)	Elevation Gain (m)	Average Slope (Â°)	Max Slope (Â°)	Slope Violations (>35Â°)	Slope Compliance (%)
Simple Astar	0.08	3309.97	491	22.61	58	48	57.89
Resampling Astar	0.91	3811.08	542	20.38	43	17	89.44
Slope_filter	950.86	3732.81	483	18.29	35	0	100

Simple A*:

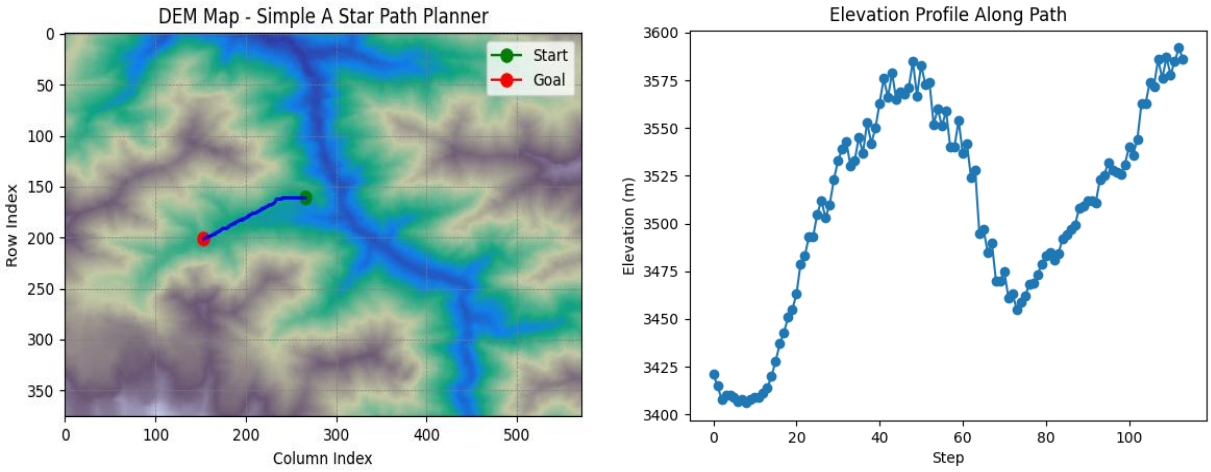

Fig 5.1: Simple A* Path Output – DEM overlay and Elevation Profile

Resampling A*:

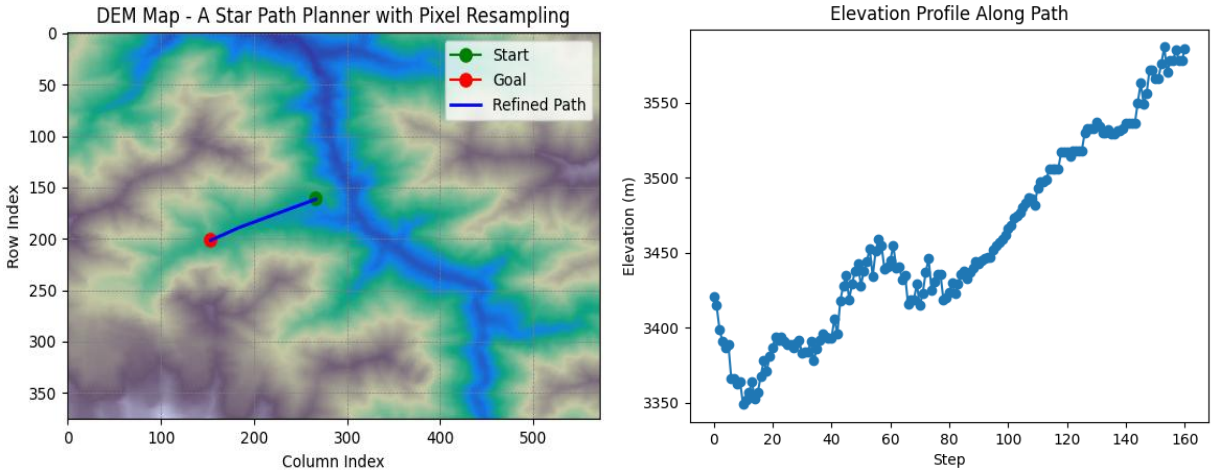

Fig 5.2: Resampling A* Path Output – DEM overlay and Elevation Profile

Terrain-Constrained A*:

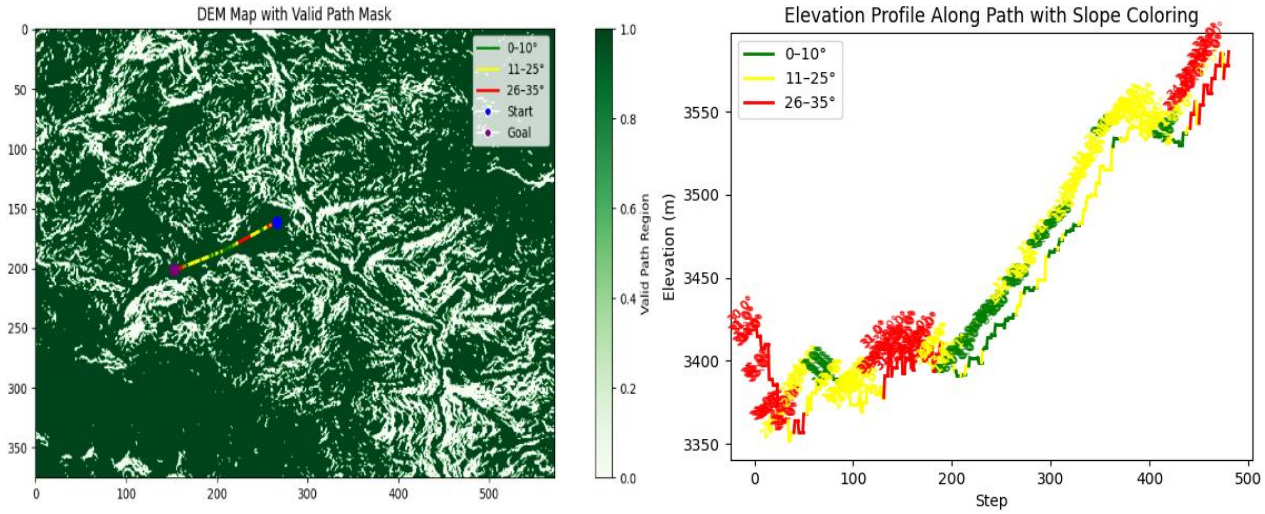

Fig 5.3: Terrain-Constrained A* Path on Valid Mask and Elevation Profile with Slope Colouring

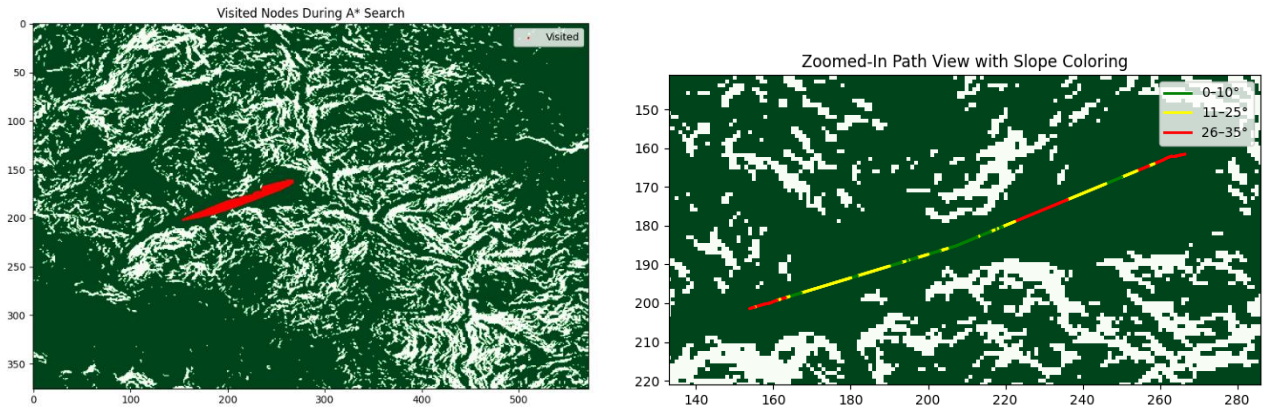

Fig 5.4: Terrain-Constrained A* Visited Nodes Map and Zoomed-In Path Segment

5.3.2 Test 2 Results:

- Star Pixel: (32, 459)
- Goal Pixel: (46, 561)

Table 5.2: Summary of Test 2 Evaluation Metrics Across Five Test Cases for Three A* Variants

Variant	Computation Time (s)	Path Length (m)	Elevation Gain (m)	Average Slope (Â°)	Max Slope (Â°)	Slope Violations (>35Â°)	Slope Compliance (%)
Simple Astar	0.03	2670.33	656	21.7	42	16	84.47
Resampling Astar	1.85	2670.37	605	21.14	47	13	87.38
Slope_filter	881.88	2779.96	613	20.01	35	0	100

Simple A*:

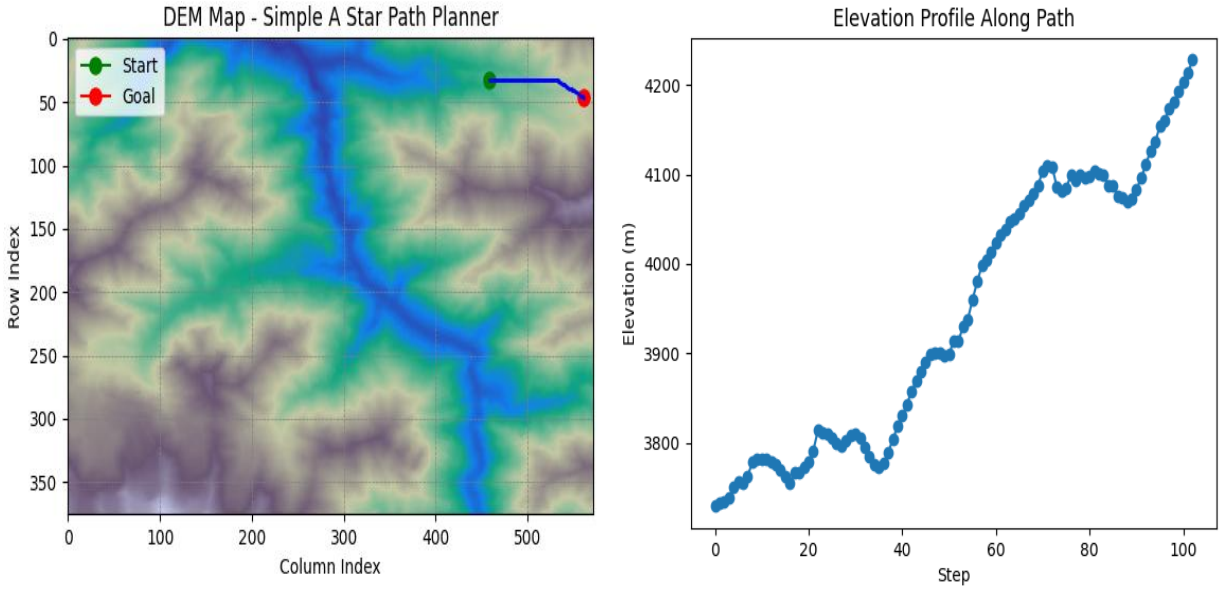

Fig 5.5: Simple A* Path Output – DEM overlay and Elevation Profile

Resampling A*:

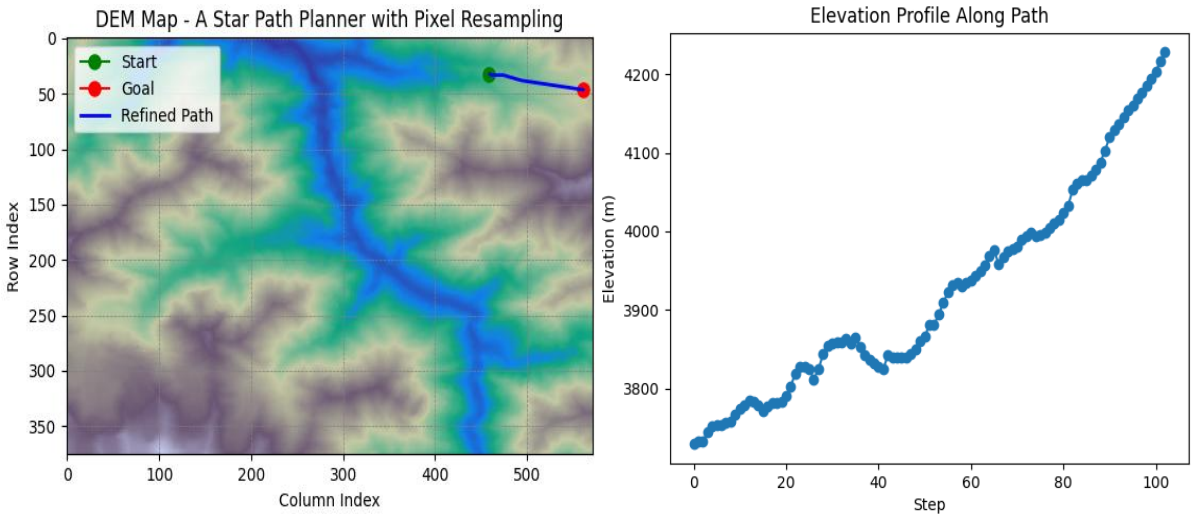

Fig 5.6: Resampling A* Path Output – DEM overlay and Elevation Profile

Terrain-Constrained A*:

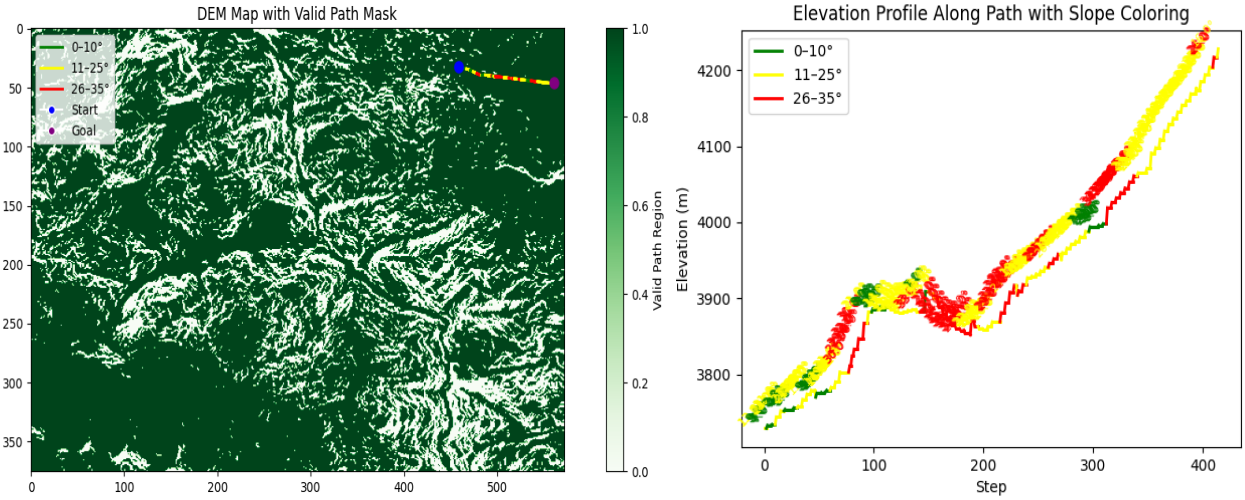

Fig 5.7: Terrain-Constrained A* Path on Valid Mask and Elevation Profile with Slope Colouring

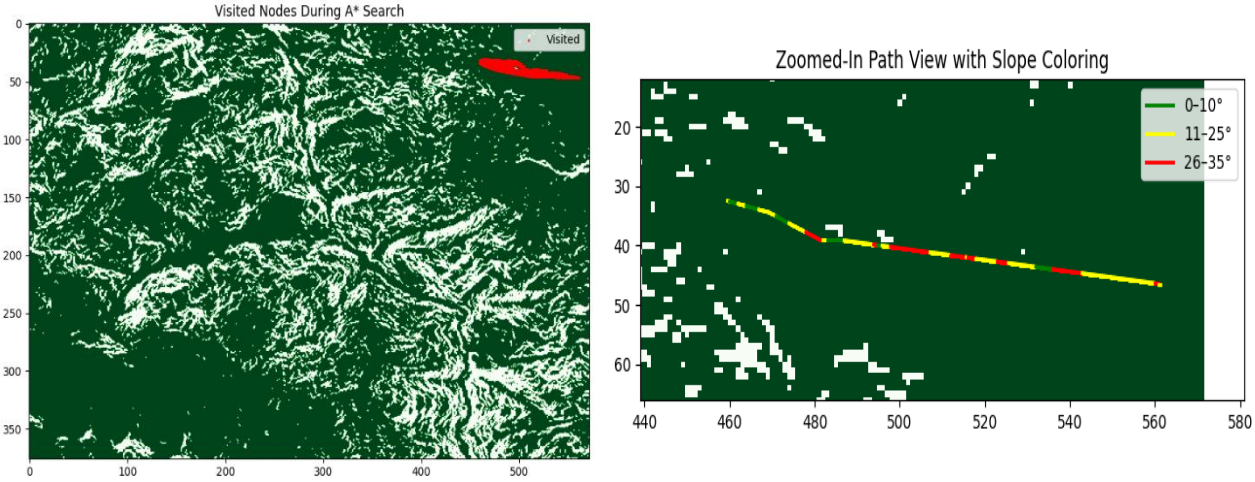

Fig 5.8: Terrain-Constrained A* Visited Nodes Map and Zoomed-In Path Segment

5.3.3 Test 3 Results:

- Star Pixel: (72, 163)
- Goal Pixel: (167, 120)

Table 5.3: Summary of Test 3 Evaluation Metrics Across Five Test Cases for Three A* Variants

Variant	Computation Time (s)	Path Length (m)	Elevation Gain (m)	Average Slope (Â°)	Max Slope (Â°)	Slope Violations (>35Â°)	Slope Compliance (%)
Simple Astar	0.07	3215.14	586	27.51	58	41	57.29
Resampling Astar	0.95	3481.31	708	29.23	55	61	57.93
Slope_filter	2901.24	3981.39	631	27.96	35	0	100

Simple A*:

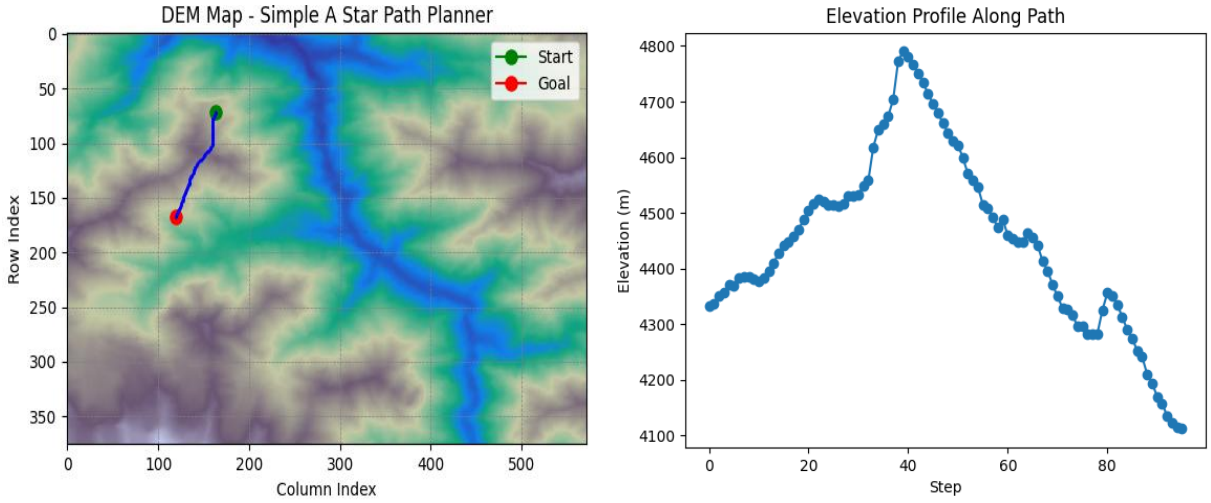

Fig 5.9: Simple A* Path Output – DEM overlay and Elevation Profile

Resampling A*:

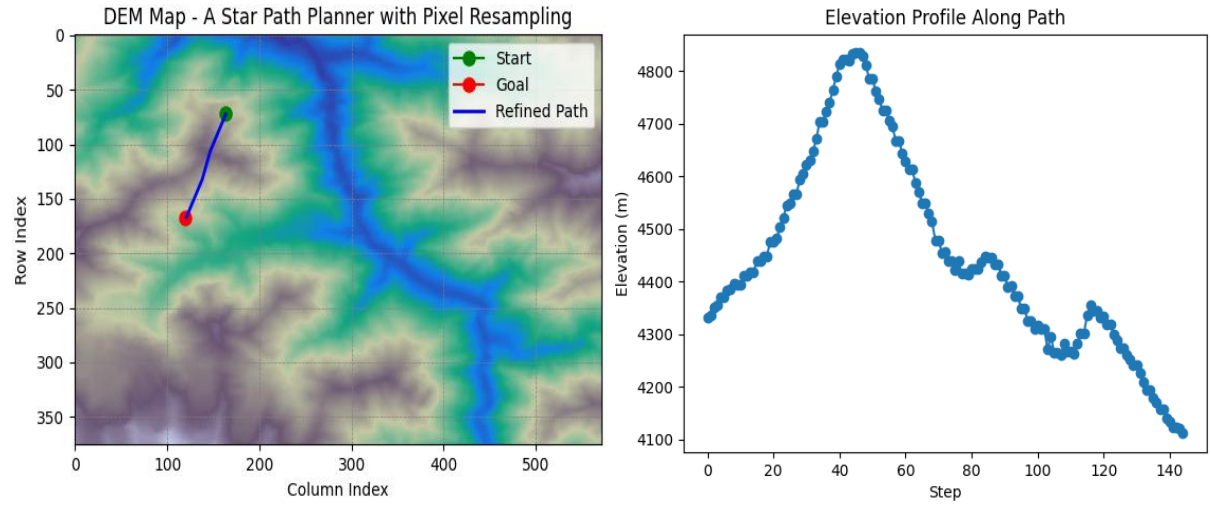

Fig 5.10: Resampling A* Path Output – DEM overlay and Elevation Profile

Terrain-Constrained A*:

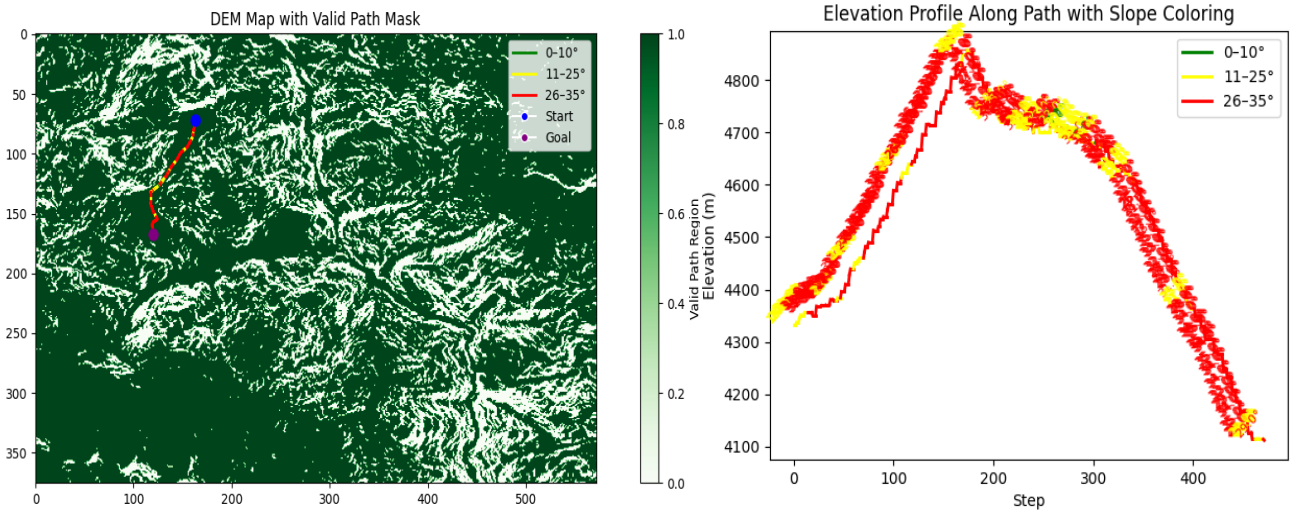

Fig 5.11: Terrain-Constrained A* Path on Valid Mask and Elevation Profile with Slope Colouring

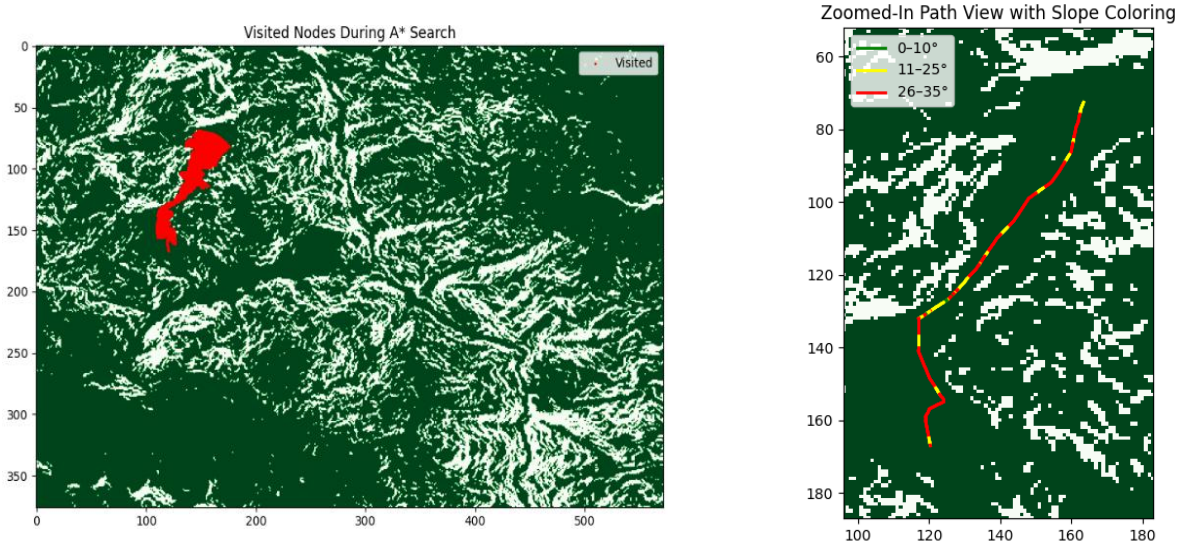

Fig 5.12: Terrain-Constrained A* Visited Nodes Map and Zoomed-In Path Segment

5.3.4 Test 4 Results:

- Star Pixel: (301, 23)
- Goal Pixel: (306, 126)

Table 5.4: Summary of Test 4 Evaluation Metrics Across Five Test Cases for Three A* Variants

Variant	Computation Time (s)	Path Length (m)	Elevation Gain (m)	Average Slope (Â°)	Max Slope (Â°)	Slope Violations (>35Â°)	Slope Compliance (%)
Simple Astar	0.02	2567.98	286	14.4	40	4	96.15
Resampling Astar	1.25	2567.98	291	15.21	40	5	95.19
Slope_filter	613.02	2614.97	285	16.31	34	0	100

Simple A*:

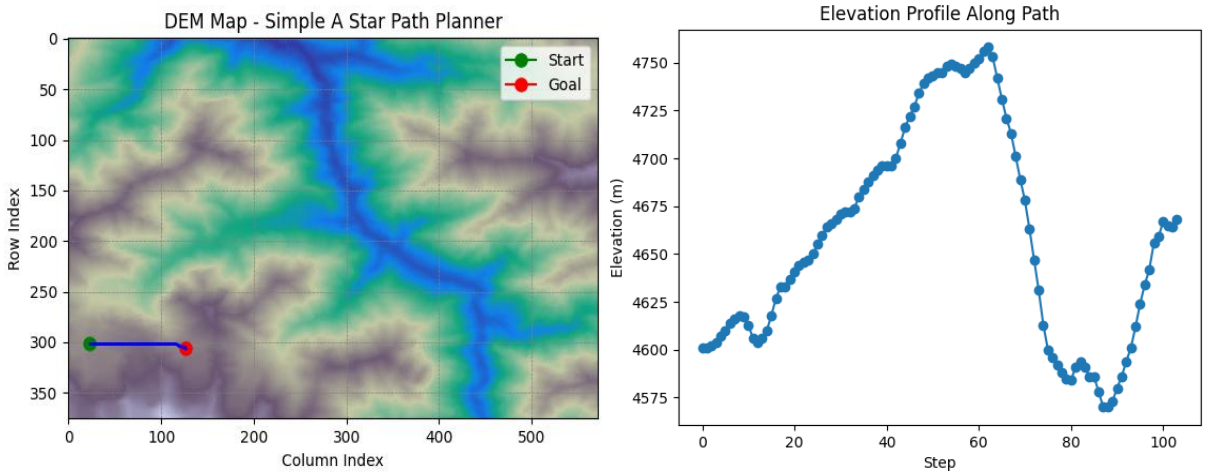

Fig 5.13: Simple A* Path Output – DEM overlay and Elevation Profile

Resampling A*:

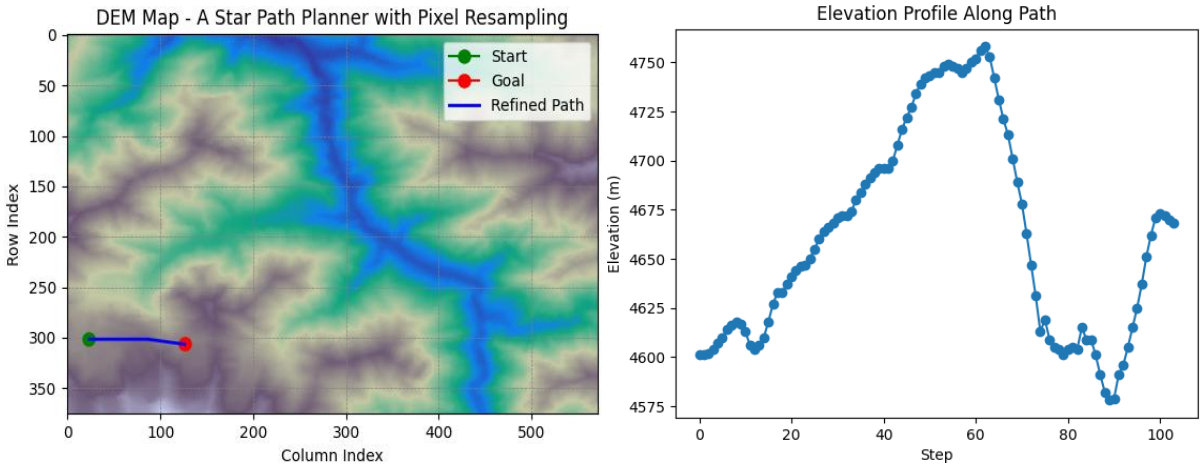

Fig 5.14: Resampling A* Path Output – DEM overlay and Elevation Profile

Terrain-Constrained A*:

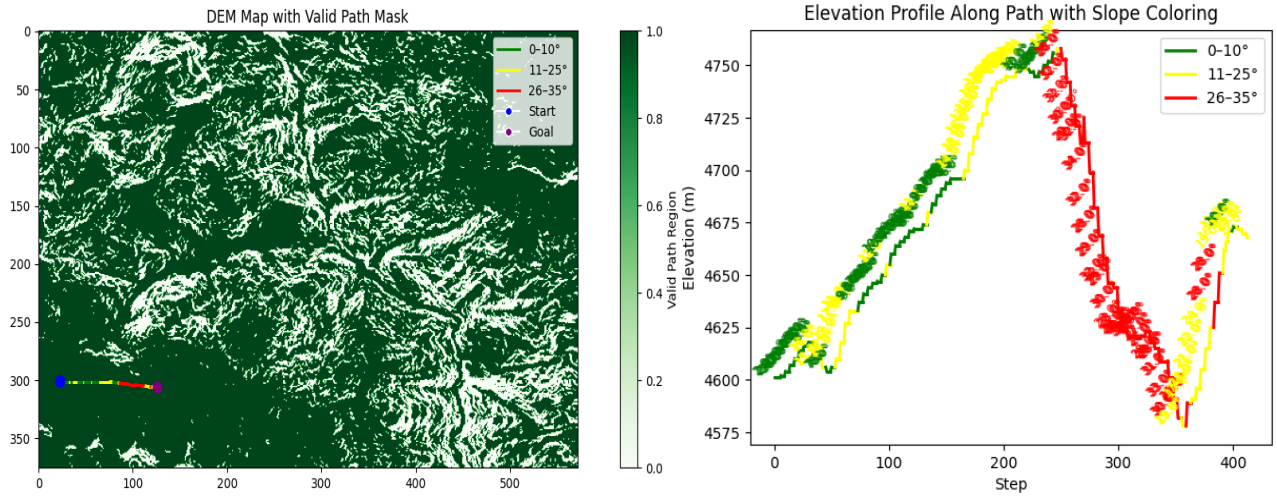

Fig 5.15: Terrain-Constrained A* Path on Valid Mask and Elevation Profile with Slope Colouring

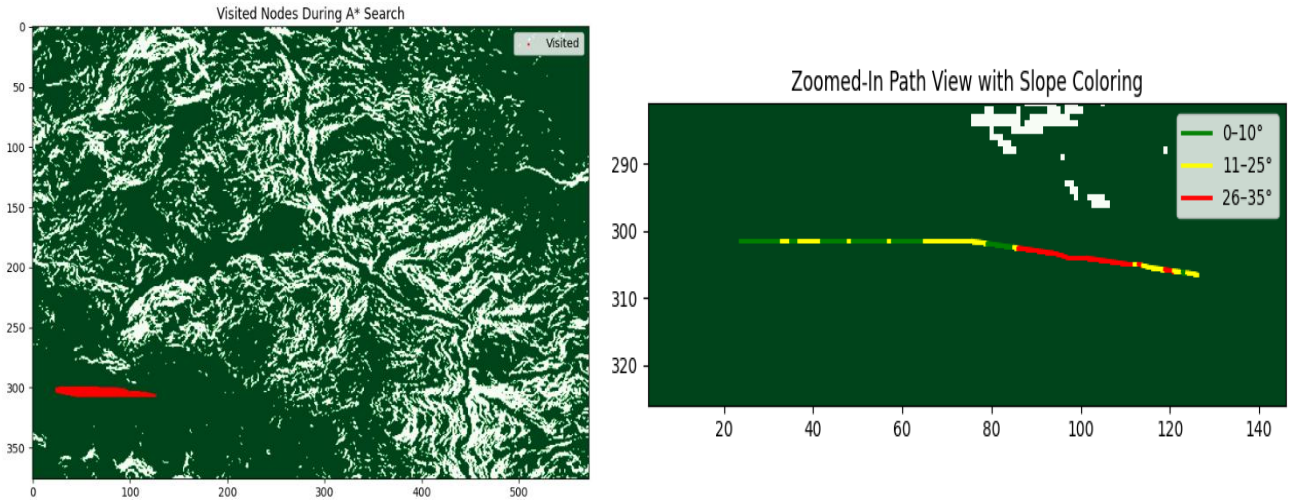

Fig 5.12: Terrain-Constrained A* Visited Nodes Map and Zoomed-In Path Segment

5.3.5 Test 5 Results:

- Star Pixel: (271, 21)
- Goal Pixel: (340, 41)

Table 5.5: Summary of Test 5 Evaluation Metrics Across Five Test Cases for Three A* Variants

Variant	Computation Time (s)	Path Length (m)	Elevation Gain (m)	Average Slope (Â°)	Max Slope (Â°)	Slope Violations (>35Â°)	Slope Compliance (%)
Simple Astar	0.03	2239.2	432	13.51	36	1	98.57
Resampling Astar	0.28	2317.5	436	11.49	38	3	96.3
Slope_filter	335.63	2522.6	438	12.74	35	0	100

Simple A*:

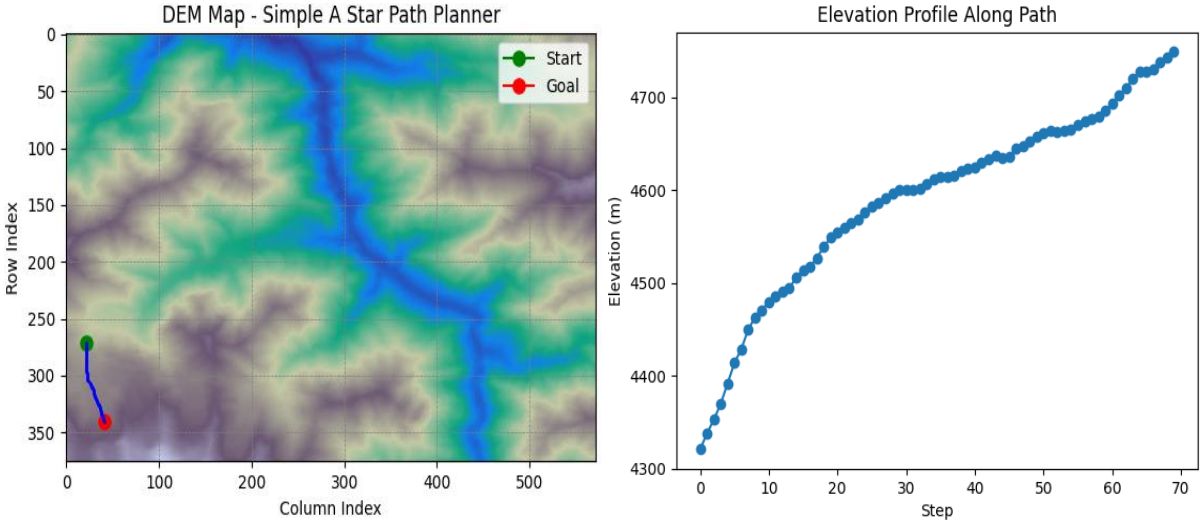

Fig 5.17: Simple A* Path Output – DEM overlay and Elevation Profile

Resampling A*:

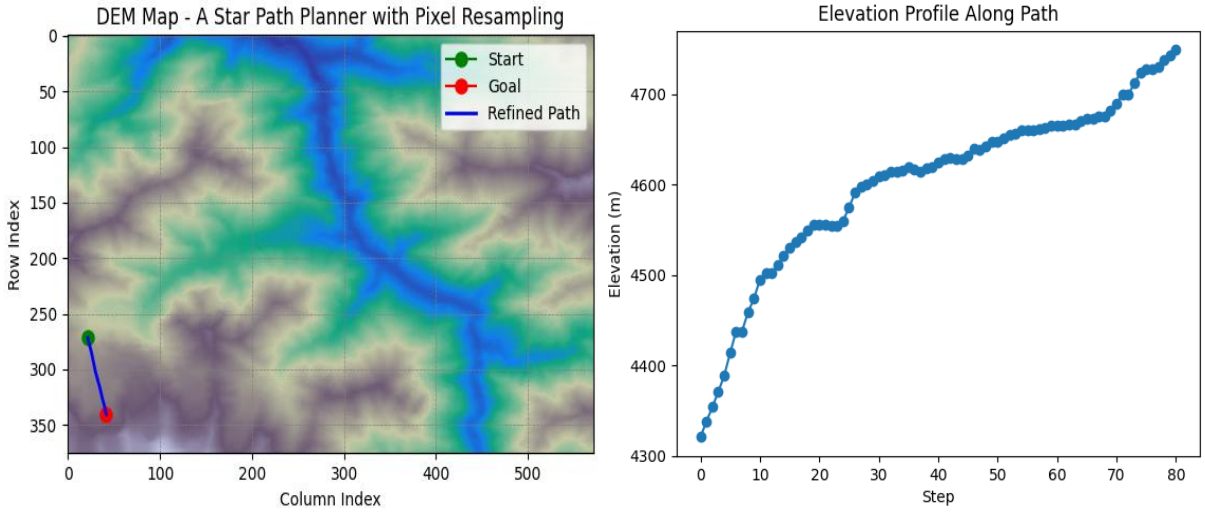

Fig 5.18: Resampling A* Path Output – DEM overlay and Elevation Profile

Terrain-Constrained A*:

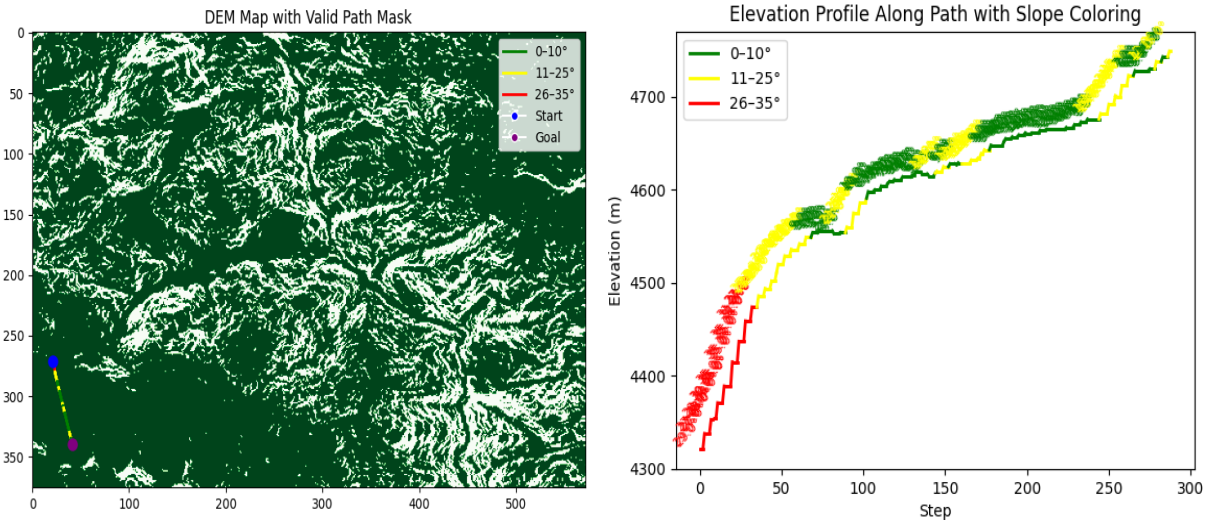

Fig 5.19: Terrain-Constrained A* Path on Valid Mask and Elevation Profile with Slope Colouring

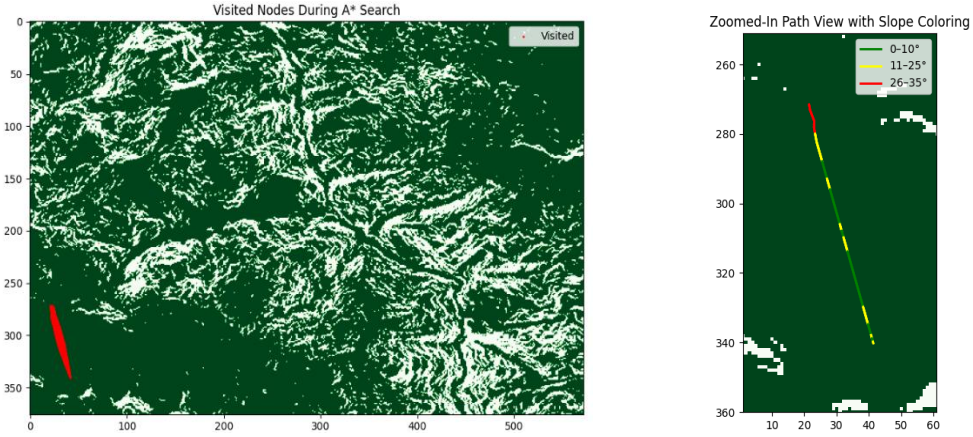

Fig 5.20: Terrain-Constrained A* Visited Nodes Map and Zoomed-In Path Segment

5.4 VISUALIZATION OUTPUTS

Each test case was accompanied by visual artifacts exported from the GUI:

- **Path_Plot.png**: Path over terrain mask
- **Zoomed_Path.png**: Focused segment around complex terrain
- **elevation_profile.png**: Elevation graph with slope coloring
- **Visited_Nodes.png**: All visited nodes during planning
- **path_coordinates.csv**: Detailed coordinate, elevation, and slope data

5.5 COMPREHENSIVE INFERENCE FROM TESTS 1–5

The comparative results obtained from five test cases provide strong evidence in favor of Terrain-Constrained A* as the most reliable path planning strategy for autonomous ground vehicles operating in unstructured and high-relief terrains. The test terrain—representing a challenging segment of the Eastern Himalayas—was carefully chosen to simulate real-world conditions where conventional roads are absent, and terrain navigability is governed by slope gradients and elevation changes.

Key observations from the tests:

- Terrain-Constrained A* consistently maintained 100% slope compliance, successfully avoiding all regions exceeding 35° slope across all test cases. This outcome validates its terrain safety logic and confirms that it can reliably generate traversable paths even in mountainous conditions.
- Despite its longer computation times (ranging from 335 to 2900 seconds), the algorithm’s subpixel resolution and 32-direction angular search enabled smoother transitions and better adherence to terrain feasibility constraints. The extra time cost is justified in scenarios where vehicular safety, mechanical stress, and rollover risks are primary concerns.
- The average slope along paths generated by this variant remained well within the stability thresholds for heavy ground vehicles such as tracked tanker systems, varying between 12.74° to 27.96°, depending on the terrain complexity.
- In contrast, both Simple A* and Resampling A* frequently violated slope constraints. Simple A* recorded slope violations in excess of 40 points in most tests, with compliance dropping to as low as 57%, rendering it impractical for any slope-sensitive vehicle.
- Resampling A* fared marginally better, thanks to smoother interpolation, but still lacked terrain safety enforcement—highlighted by multiple slope violations (up to 61), despite

slightly higher compliance percentages in less rugged regions.

From a mission-critical perspective, particularly in applications involving autonomous fuel tanker convoys, supply delivery vehicles, or rescue platforms traversing unstructured battlefields or disaster zones, Terrain-Constrained A* emerges as the most operationally viable solution. It adapts to terrain topography while minimizing the need for infrastructure, making it ideal for off-road navigation in permafrost, desert, alpine, or forested terrains.

The performance metrics for each test case are summarized individually in Tables 5.1 through 5.5, allowing per-test comparison of the three algorithm variants.

5.6 DISCUSSION

The collective insights from the experimental analysis underscore the critical advantages of Terrain-Constrained A* when applied to path planning for autonomous vehicles navigating **natural, infrastructure-less environments**. Such scenarios are common in:

- Military logistics in high-altitude or enemy-contested zones.
- Disaster response operations where roads are blocked or non-existent.
- Scientific expeditions or oil & gas exploration in remote terrain.
- Supply chain missions for mobile base camps or frontier outposts.

Advantages of Terrain-Constrained A* in these contexts:

1. **Guaranteed Feasibility:** The algorithm ensures all paths remain within slope-safe zones, effectively mimicking a terrain-aware vehicle operator avoiding unstable ground.
2. **Subpixel Smoothness:** Its use of subpixel resolution and 32-directional search minimizes jerky turns, making it better suited for articulated or tracked tanker vehicles with steering and traction limitations.
3. **Safety-Centric Design:** While other variants prioritize speed or simplicity, Terrain-Constrained A* makes safety the core constraint, ensuring that no part of the path demands motion over slopes that exceed vehicular mechanical thresholds.
4. **Adaptability Across Terrain Types:** It was equally effective in steep ridges, broad valleys, and lowland regions, showing universal adaptability regardless of elevation profile.
5. **No Dependency on Roads:** The system operates entirely on raster-based terrain inputs without the need for vectorized roads or maps, making it deployable even in GNSS-denied or unmapped areas.

While Terrain-Constrained A* incurs higher computation time, the primary focus of this phase was to ensure accuracy, safety, and terrain compliance by integrating all available geospatial layers. Optimization of execution speed remains a future objective, with current results prioritizing reliable and feasible path generation for off-road applications.

CHAPTER 6

CONCLUSION AND FUTURE WORK

The objective of this work was to design and evaluate an autonomous terrain-aware path planning system using variants of the A* algorithm, tailored for military logistic applications—specifically, autonomous navigation of tracked tanker vehicles in regions with limited or no road infrastructure. The deployment focus was on high-relief zones such as the Eastern Himalayas, where conventional GPS-enabled navigation is insufficient due to abrupt terrain gradients, slope-induced hazards, and the absence of paved routes.

The proposed system utilized geospatial raster datasets—namely Digital Elevation Models (DEM) and slope maps derived from SRTM and processed via Google Earth Engine (GEE)—to generate terrain-aware cost surfaces. These were used to construct a **probabilistic terrain feasibility map (PTFM)** that informed the navigation logic of enhanced A* variants.

Three A* path planning strategies were implemented:

1. **Simple A***: Used the DEM directly without any terrain constraints. It offered the fastest computation but frequently violated slope tolerances and ignored terrain feasibility.
2. **Resampling A***: Performed subpixel interpolation over DEM to refine path smoothness, yet lacked elevation and slope constraints—rendering it unsafe for critical terrain traversal.
3. **Terrain-Constrained A***: Introduced terrain masking based on operational elevation and slope thresholds (≤35°), enforced subpixel motion discretization, and supported 32-directional search vectors. This variant achieved 100% slope compliance, the most realistic elevation profiles, and avoided operationally dangerous paths.

A GUI-based implementation using PyQt5 was developed to facilitate DEM/slope raster loading, terrain filtering, and interactive path visualization. All results were exported in .csv format for further metric computation and documentation.

6.1 KEY CONTRIBUTIONS

- **Technical Innovation**: Developed an A* variant capable of navigating high-resolution raster maps with slope masking, 32-angle directional movement, and subpixel node expansion—achieving terrain-informed trajectory planning suitable for military platforms.
- **Geospatial Integration**: Successfully coupled elevation and slope rasters using NumPy and Rasterio into a navigability-aware cost map that formed the core of the Terrain-Constrained A*.
- **Performance Validation**: Five diverse test cases validated the robustness of each algorithm. Terrain-Constrained A* maintained superior compliance with mission-critical mobility constraints, including slope violation mitigation and elevation-aware routing.
- **Defense Relevance**: The system architecture supports use in high-altitude logistics missions where wheeled or tracked vehicles (e.g., fuel tankers, munition carriers) must autonomously navigate hostile terrains under GPS-denied or communication-constrained conditions.
- **Exportable Artifacts**: Visual outputs (path overlays, elevation profiles, visited nodes) and metric summaries provide actionable insights for mission planning and control systems.

6.2 LIMITATIONS

Despite its accuracy, the system has current limitations:

- **Computational Overhead**: Terrain-Constrained A* incurs higher computation time (up to ~2900s in worst-case scenarios) due to fine-grained directional expansion and slope filtering. This restricts use in real-time planning contexts.
- **Static Environment Assumption**: The terrain mask and DEM data are static. Dynamic terrain changes (e.g., landslides, flooding) cannot yet be incorporated mid-mission.
- **No Real-Time Sensor Fusion**: The system does not ingest real-time IMU/LiDAR/SLAM data to correct or update terrain traversability in an online manner.
- **Single-Agent Planning**: The current model plans for a single tanker unit without coordination logic for multi-agent logistics convoys.

6.3 FUTURE WORK

The foundation laid by this system is promising for real-world defense applications. The following improvements are envisioned for operational deployment:

1. **Real-Time Optimization**
   - Implement GPU acceleration and optimized priority queues (e.g., Fibonacci Heaps or Fast Marching Trees) to lower computation time.
   - Investigate hierarchical planning strategies (e.g., coarse-to-fine resolution A*) to reduce unnecessary node expansions in flat or low-risk regions.
2. **Dynamic Replanning and Obstacle Avoidance**
   - Integrate Model Predictive Control (MPC) or Dynamic Window Approach (DWA) at the control layer to allow reactive replanning during obstacle encounters.
   - Use real-time LIDAR/SAR data to dynamically update DEM and slope layers during movement.
3. **Terrain Intelligence Fusion**
   - Incorporate additional geospatial layers such as soil moisture index, surface roughness, vegetation density (from Sentinel-2), and trafficability indexes into the feasibility map.
   - Develop a multi-criteria terrain cost model tailored to different vehicle types (e.g., wheeled vs. tracked) and mission priorities (e.g., stealth vs. speed).
4. **SLAM and GPS-Denied Operation**
   - Fuse this path planner with simultaneous localization and mapping (SLAM) systems and inertial navigation to ensure robustness in GPS-denied environments.
5. **Multi-Agent Path Planning**
   - Extend the system for coordinated mission planning involving multiple tanker units, incorporating convoy spacing, redundancy paths, and synchronized arrival constraints.
6. **Hardware-in-the-Loop Simulation**
   - Validate on physical tracked robots or UGVs with ROS2-based simulation environments like Gazebo or CARLA, using real-time feedback from terrain sensors to adjust paths dynamically.
7. **Multi-Layer Terrain Reasoning**
   - For simplicity and clarity of implementation, the current system relied solely on elevation (DEM) and slope raster inputs to derive the terrain feasibility map. However, real-world deployment of autonomous tracked tankers in complex and unstructured

terrains requires more comprehensive terrain reasoning. Future versions should integrate additional environmental layers to better assess terrain load-bearing capacity and hydrological obstructions:

- **Soil Type and Bulk Density**: Incorporating soil classification maps and bulk density data (e.g., from SoilGrids or ISRIC datasets) enables estimation of ground pressure tolerance and sinkage risk—critical for planning safe routes for heavily loaded tracked tankers that exert high ground pressure.
- **Moisture Content and Drainage Class**: Terrain with high soil moisture or poor drainage poses a risk of slippage or immobilization. These can be modeled using remote sensing indices (e.g., NDWI, soil moisture products from SMAP) to influence path costs.
- **Water Bodies and Depth Classification**: Integration of hydrological layers (e.g., rivers, lakes, ponds, and their inferred depth) allows for adaptive navigation decisions. Depending on the vehicle's amphibious capabilities, the planner could:
    - Allow shallow crossings if the depth is within operational limits, or
    - Avoid water bodies entirely to prevent stalling or damage in the absence of bridging equipment.

This multi-criteria terrain modelling approach will significantly enhance the reliability, survivability, and autonomy of tanker vehicle navigation in reconnaissance, resupply, or withdrawal operations across infrastructure-deficient theatres.

6.4 FINAL REMARKS

In terrains where roads do not exist and human error is unacceptable, the deployment of an accurate and terrain-compliant path planning module becomes mission-critical. The Terrain-Constrained A* demonstrated here fulfils that requirement by systematically avoiding operational hazards while leveraging topographical data. Though computationally expensive in its current form, it establishes a strong groundwork for safe autonomous navigation in defence, disaster relief, and remote logistic missions.

BIBLIOGRAPHY

1. Z. Hong, L. Liu, Y. Zhao, and J. Wang, “Improved A-Star Algorithm for Long-Distance Off-Road Path Planning Using Terrain Data Map,” ISPRS International Journal of Geo-Information, vol. 10, no. 11, pp. 1–22, 2021.
2. X. Zheng, M. Ma, Z. Zhong, A. Yang, L. Chen, and N. Jing, “Two-Stage Path Planning for Long-Distance Off-Road Path Planning Based on Terrain Data,” ISPRS International Journal of Geo-Information, vol. 13, no. 6, pp. 1–19, 2024.
3. D. D. L. Nash, S. Koenig, and A. Felner, “Theta*: Any-Angle Path Planning on Grids,” Journal of Artificial Intelligence Research, vol. 39, pp. 533–579, 2010.
4. D. Yakovlev and A. Andreychuk, “Grid-Based Angle-Constrained Path Planning,” Proceedings of the 2015 European Conference on Mobile Robots (ECMR), pp. 1–6, 2015.
5. X. Wu, C. Zhao, and W. Hu, “Planetary Rover Path Planning Based on Improved A* Algorithm,” Journal of Intelligent & Robotic Systems, vol. 95, pp. 883–896, 2019.
6. M. Saad, A. Elshafee, M. Aly, and A. Gad, “A Composite Metric Routing Approach for Energy-Efficient Path Planning on Natural Terrains,” Applied Sciences, vol. 11, no. 15, pp. 1–20, 2021.
7. S. B. Hartzell, “Autonomous Vehicle Navigation in Unstructured Environments Using Hybrid A*,” Master’s Thesis, KTH Royal Institute of Technology, Sweden, 2021.
8. R. W. Komodo and H. F. Azis, “Autonomous Navigation of Tracked Vehicles with GPS Waypoint Tracking,” Procedia Computer Science, vol. 135, pp. 320–327, 2018.
9. C. Campos, R. Elvira, J. J. Gómez Rodríguez, J. M. M. Montiel, and J. D. Tardós, “ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM,” IEEE Transactions on Robotics, vol. 37, no. 6, pp. 1874–1890, 2021.
10. M. Kumar, N. Mishra, and P. Singh, “Application of GIS and Artificial Intelligence in Military Operations: Prospects and Challenges,” Defence Science Journal, vol. 71, no. 6, pp. 739–749, 2021.
11. J. Sánchez-Ibáñez, M. Carrasco-Alvarez, and P. Campoy, “Path Planning for Autonomous Mobile Robots: A Review,” Sensors, vol. 21, no. 23, pp. 1–38, 2021.

Wed Sources:

12. Gillies, S. (2013). Rasterio: geospatial raster I/O for Python programmers. *https://github.com/rasterio/rasterio*
13. QGIS Development Team (2023). QGIS Geographic Information System. Open Source Geospatial Foundation Project. *http://qgis.org*
14. PyQt5 Documentation. *https://doc.qt.io/qtforpython/*
15. Google Earth Engine. *https://developers.google.com/earth-engine/datasets*
16. NumPy Developers. (2023). *https://numpy.org*
17. SciPy Developers. (2023). *https://scipy.org*
18. GDAL/OGR contributors. (2023). GDAL - Geospatial Data Abstraction Library. *https://gdal.org*

APPENDIX

SOURCE CODE LISTINGS

A. PYTHON SCRIPTS

A.1 Simple A*

```
import sys
import numpy as np
import rasterio
import heapq
import matplotlib.pyplot as plt
import pandas as pd
import time
from PyQt5.QtWidgets import (
    QApplication, QMainWindow, QPushButton, QFileDialog, QVBoxLayout,
    QWidget, QInputDialog
)
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as
FigureCanvas, NavigationToolbar2QT as NavigationToolbar

class DEMPathPlanner(QMainWindow):
    def __init__(self):
        super().__init__()
        self.initUI()
        self.dem_data = None
        self.slope_data = None
        self.transform = None
        self.start = (271, 21)
        self.goal = (340, 41)
        self.go_area = None

    def initUI(self):
        self.setWindowTitle("DEM Path Planner with Slope Export")
        self.setGeometry(100, 100, 800, 600)

        self.canvas = FigureCanvas(plt.figure())
        self.toolbar = NavigationToolbar(self.canvas, self)

        self.btn_load = QPushButton("Load DEM + Slope", self)
        self.btn_load.clicked.connect(self.load_rasters)

        self.btn_compute = QPushButton("Compute Path", self)
```

```
        self.btn_compute.clicked.connect(self.compute_path)

        layout = QVBoxLayout()
        layout.addWidget(self.toolbar)
        layout.addWidget(self.canvas)
        layout.addWidget(self.btn_load)
        layout.addWidget(self.btn_compute)

        container = QWidget()
        container.setLayout(layout)
        self.setCentralWidget(container)

    def load_rasters(self):
         dem_file, _ = QFileDialog.getOpenFileName(self, "Open DEM
File", "", "GeoTIFF Files (*.tif)")
        slope_file, _ = QFileDialog.getOpenFileName(self, "Open Slope
File", "", "GeoTIFF Files (*.tif)")
        if dem_file and slope_file:
            with rasterio.open(dem_file) as src:
                self.dem_data = src.read(1)
                self.transform = src.transform
            with rasterio.open(slope_file) as src:
                self.slope_data = src.read(1)
            self.go_area = np.ones_like(self.dem_data, dtype=bool)
            self.plot_dem()
            print("DEM and Slope loaded successfully.")
    def plot_dem(self):
        plt.clf()
        ax = self.canvas.figure.add_subplot(111)
        ax.imshow(self.dem_data, cmap='terrain')
        ax.set_title("DEM Map - A* Path Planner")
        ax.set_xlabel("Column Index")
        ax.set_ylabel("Row Index")
        ax.grid(True, linestyle='--', linewidth=0.5)
        if self.start:
                 ax.plot(self.start[1],  self.start[0],  marker='o',
color='green', markersize=8, label='Start')
        if self.goal:
                  ax.plot(self.goal[1],  self.goal[0],  marker='o',
color='red', markersize=8, label='Goal')
        ax.legend()
        self.canvas.draw()
    def compute_path(self):
        if self.dem_data is None or self.start is None or self.goal
is None:
            print("Missing DEM or start/goal not set.")
            return
        start_time = time.time()
        path = self.a_star()
        end_time = time.time()
        computation_time = end_time - start_time
```

```
        if path:
            self.plot_path(path)
            self.plot_elevation_profile(path)
            self.export_path_csv(path, computation_time)
        else:
            print("No valid path found!")
    def a_star(self):
        rows, cols = self.dem_data.shape
        start_x, start_y = self.start
        goal_x, goal_y = self.goal

        open_set = [(0, start_x, start_y)]
        came_from = {}
        g_score = np.full((rows, cols), np.inf)
        g_score[start_x, start_y] = 0

        directions = [(-1, 0), (1, 0), (0, -1), (0, 1),
                      (-1, -1), (-1, 1), (1, -1), (1, 1)]

        def heuristic(a, b):
            return np.hypot(a[0] - b[0], a[1] - b[1])

        while open_set:
            _, x, y = heapq.heappop(open_set)
            if (x, y) == (goal_x, goal_y):
                return self.reconstruct_path(came_from, (x, y))

            for dx, dy in directions:
                nx, ny = x + dx, y + dy
              if 0 <= nx < rows and 0 <= ny < cols and self.go_area[nx,
ny]:
                    cost = np.hypot(dx, dy)
                    tentative_g = g_score[x, y] + cost
                    if tentative_g < g_score[nx, ny]:
                        g_score[nx, ny] = tentative_g
                        came_from[(nx, ny)] = (x, y)
                         f_score = tentative_g + heuristic((nx, ny),
(goal_x, goal_y))
                        heapq.heappush(open_set, (f_score, nx, ny))
        return None
    def reconstruct_path(self, came_from, current):
        path = []
        while current in came_from:
            path.append(current)
            current = came_from[current]
        path.append(current)
        return path[::-1]

    def plot_path(self, path):
        ax = self.canvas.figure.axes[0]
        y_vals = [y for x, y in path]
        x_vals = [x for x, y in path]
```

```
            ax.plot(y_vals, x_vals, color='blue', linewidth=2,
label='Path')
        ax.legend()
        self.canvas.draw()
        plt.savefig("Path_Plot.png")
        print("Saved Path_Plot.png")

    def plot_elevation_profile(self, path):
        elevations = [self.dem_data[x, y] for x, y in path]
        plt.figure()
        plt.plot(elevations, marker='o')
        plt.title("Elevation Profile")
        plt.xlabel("Step")
        plt.ylabel("Elevation (m)")
        plt.grid(True)
        plt.savefig("elevation_profile.png")
        plt.show()

    def export_path_csv(self, path, computation_time):
        rows = []
        for i, (x, y) in enumerate(path):
            lon, lat = rasterio.transform.xy(self.transform, int(x),
int(y))
            elevation = self.dem_data[int(x), int(y)]
           slope = self.slope_data[int(x), int(y)] if self.slope_data
is not None else "N/A"
            rows.append([i, lat, lon, elevation, slope])

            df = pd.DataFrame(rows, columns=["Step", "Latitude",
"Longitude", "Elevation", "Slope (°)"])

        metadata = [
            ["Path Planning Algorithm", "Simple A*"],
            ["Start Pixel", str(self.start)],
            ["Goal Pixel", str(self.goal)],
            ["Computation Time (s)", f"{computation_time:.2f}"],
            []
        ]
        meta_df = pd.DataFrame(metadata)

        with open("path_coordinates.csv", "w") as f:
            meta_df.to_csv(f, index=False, header=False)
            df.to_csv(f, index=False)

       print("☑ path_coordinates.csv saved with slope and metadata")

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = DEMPathPlanner()
    window.show()
    sys.exit(app.exec_())
```

A.2 Resampling A*

```
import sys
import numpy as np
import rasterio
import heapq
import matplotlib.pyplot as plt
import pandas as pd
import time
from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton,
QFileDialog, QVBoxLayout, QWidget
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as
FigureCanvas, NavigationToolbar2QT as NavigationToolbar

class DEMPathPlanner(QMainWindow):
    def __init__(self):
        super().__init__()
        self.initUI()
        self.dem_data = None
        self.slope_data = None
        self.transform = None
        self.start = (271, 21)
        self.goal = (340, 41)
        self.go_area = None

    def initUI(self):
         self.setWindowTitle("DEM Path Planner with Resampling and
Slope Export")
        self.setGeometry(100, 100, 800, 600)

        self.canvas = FigureCanvas(plt.figure())
        self.toolbar = NavigationToolbar(self.canvas, self)

        self.btn_load = QPushButton("Load DEM and Slope", self)
        self.btn_load.clicked.connect(self.load_rasters)

        self.btn_compute = QPushButton("Compute Path", self)
        self.btn_compute.clicked.connect(self.compute_path)

        layout = QVBoxLayout()
        layout.addWidget(self.toolbar)
        layout.addWidget(self.canvas)
        layout.addWidget(self.btn_load)
        layout.addWidget(self.btn_compute)

        container = QWidget()
        container.setLayout(layout)
        self.setCentralWidget(container)

    def load_rasters(self):
         dem_file, _ = QFileDialog.getOpenFileName(self, "Open DEM
```

```
File", "", "GeoTIFF Files (*.tif)")
        slope_file, _ = QFileDialog.getOpenFileName(self, "Open Slope
File", "", "GeoTIFF Files (*.tif)")
        if dem_file and slope_file:
            with rasterio.open(dem_file) as src:
                self.dem_data = src.read(1)
                self.transform = src.transform
            with rasterio.open(slope_file) as src:
                self.slope_data = src.read(1)
            self.go_area = np.ones_like(self.dem_data, dtype=bool)
            self.plot_dem()
            print("DEM and slope loaded successfully.")

    def plot_dem(self):
        plt.clf()
        ax = self.canvas.figure.add_subplot(111)
        ax.imshow(self.dem_data, cmap='terrain')
        ax.set_title("DEM Map - Resampling A*")
        ax.set_xlabel("Column Index")
        ax.set_ylabel("Row Index")
        ax.grid(True, linestyle='--', linewidth=0.5)
                ax.plot(self.start[1],   self.start[0],   marker='o',
color='green', markersize=8, label='Start')
        ax.plot(self.goal[1], self.goal[0], marker='o', color='red',
markersize=8, label='Goal')
        ax.legend()
        self.canvas.draw()

    def compute_path(self):
        if self.dem_data is None or self.slope_data is None:
            print("Please load DEM and Slope files first.")
            return

        start_time = time.time()
        path = self.a_star_subpixel()
        end_time = time.time()
        computation_time = end_time - start_time

        if path:
            self.plot_path(path)
            self.plot_elevation_profile(path)
            self.export_path_csv(path, computation_time)
            print(f"Computation time: {computation_time:.2f} s")
        else:
            print("No valid path found!")

    def a_star_subpixel(self, substeps=4):
        rows, cols = self.dem_data.shape
        start = (self.start[0] + 0.5, self.start[1] + 0.5)
        goal = (self.goal[0] + 0.5, self.goal[1] + 0.5)

        open_set = [(0, start)]
```

```
        heapq.heapify(open_set)
        came_from = {}
        g_score = {start: 0}
        directions = [(dx / substeps, dy / substeps)
                      for dx in range(-substeps, substeps + 1)
                      for dy in range(-substeps, substeps + 1)
                      if dx != 0 or dy != 0]

        def heuristic(a, b):
            return np.hypot(a[0] - b[0], a[1] - b[1])

        visited = set()
        while open_set:
            _, current = heapq.heappop(open_set)
             if (int(current[0]), int(current[1])) == (int(goal[0]),
int(goal[1])):
                return self.reconstruct_path(came_from, current)

            if current in visited:
                continue
            visited.add(current)

            for dx, dy in directions:
                neighbor = (current[0] + dx, current[1] + dy)
                r, c = int(neighbor[0]), int(neighbor[1])
               if 0 <= r < rows and 0 <= c < cols and self.go_area[r,
c]:
                    tentative_g = g_score[current] + np.hypot(dx, dy)
                       if neighbor not in g_score or tentative_g <
g_score[neighbor]:
                        came_from[neighbor] = current
                        g_score[neighbor] = tentative_g
                         f_score = tentative_g + heuristic(neighbor,
goal)
                        heapq.heappush(open_set, (f_score, neighbor))
        return None

    def reconstruct_path(self, came_from, current):
        path = []
        while current in came_from:
            path.append(current)
            current = came_from[current]
        path.append(current)
        return path[::-1]

    def plot_path(self, path):
        ax = self.canvas.figure.axes[0]
        y_vals = [y for x, y in path]
        x_vals = [x for x, y in path]
              ax.plot(y_vals,  x_vals,  color='blue',  linewidth=2,
label='Path')
        ax.legend()
```

```
        self.canvas.draw()
        plt.savefig("Path_Plot.png")
        print("Saved Path_Plot.png")

    def plot_elevation_profile(self, path):
        elevations = [self.dem_data[int(x), int(y)] for x, y in path]
        plt.figure()
        plt.plot(elevations, marker='o')
        plt.title("Elevation Profile")
        plt.xlabel("Step")
        plt.ylabel("Elevation (m)")
        plt.grid(True)
        plt.savefig("elevation_profile.png")
        plt.show()

    def export_path_csv(self, path, computation_time):
        rows = []
        for i, (x, y) in enumerate(path):
            lon, lat = rasterio.transform.xy(self.transform, int(x),
int(y))
            elevation = self.dem_data[int(x), int(y)]
            slope = self.slope_data[int(x), int(y)]
            rows.append([i, lat, lon, elevation, slope])

            df = pd.DataFrame(rows, columns=["Step", "Latitude",
"Longitude", "Elevation", "Slope (°)"])

        metadata = [
            ["Path Planning Algorithm", "A* with Pixel Resampling"],
            ["Start Pixel", str(self.start)],
            ["Goal Pixel", str(self.goal)],
            ["Computation Time (s)", f"{computation_time:.2f}"],
            []
        ]
        meta_df = pd.DataFrame(metadata)

        with open("path_coordinates.csv", "w") as f:
            meta_df.to_csv(f, index=False, header=False)
            df.to_csv(f, index=False)
            print("☑ path_coordinates.csv saved with slope and
metadata.")

    def save_graphs(self):
        self.canvas.figure.savefig("dem_path.png")
        print("Graph saved as dem_path.png")

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = DEMPathPlanner()
    window.show()
    sys.exit(app.exec_())
```

A.3 Terrain-Constrained A*

```
import sys
import numpy as np
import rasterio
import matplotlib.pyplot as plt
import pandas as pd
from queue import PriorityQueue
import heapq
from PyQt5.QtWidgets import (QApplication, QMainWindow, QPushButton,
QFileDialog, QVBoxLayout,
                             QWidget, QInputDialog, QTextEdit)
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as
FigureCanvas, NavigationToolbar2QT as NavigationToolbar
import time
class DEMPathPlanner(QMainWindow):
    def __init__(self):
        super().__init__()
        self.initUI()
        self.dem_data = None
        self.slope_data = None
        self.valid_mask = None
        self.transform = None
        self.start = None
        self.goal = None
        self.min_elev = None
        self.max_elev = None
        self.go_area = None
        self.visited_nodes = set()

    def initUI(self):
        self.setWindowTitle("DEM Path Planner")
        self.setGeometry(100, 100, 1400, 900)

        self.canvas = FigureCanvas(plt.figure(figsize=(10, 6)))
        self.toolbar = NavigationToolbar(self.canvas, self)
                        self.canvas.mpl_connect("button_press_event",
self.mouse_click_event)

        self.btn_load = QPushButton("Load DEM + Slope", self)
        self.btn_load.clicked.connect(self.load_dem)

        self.btn_compute = QPushButton("Compute Path", self)
        self.btn_compute.clicked.connect(self.compute_path)

        self.btn_export = QPushButton("Export CSV", self)
                                self.btn_export.clicked.connect(lambda:
self.export_path_csv(self.reconstruct_path({},   self.goal)[0])   if
self.goal else None)

        self.status_log = QTextEdit()
```

```
        self.status_log.setReadOnly(True)

        layout = QVBoxLayout()
        layout.addWidget(self.toolbar)
        layout.addWidget(self.canvas)
        layout.addWidget(self.btn_load)
        layout.addWidget(self.btn_compute)
        layout.addWidget(self.btn_export)
        layout.addWidget(self.status_log)

        container = QWidget()
        container.setLayout(layout)
        self.setCentralWidget(container)

    def log(self, text):
        self.status_log.append(text)

    def load_dem(self):
        options = QFileDialog.Options()
         file_name, _ = QFileDialog.getOpenFileName(self, "Open DEM
File", "", "GeoTIFF Files (*.tif);;All Files (*)", options=options)
       slope_file, _ = QFileDialog.getOpenFileName(self, "Open Slope
File", "", "GeoTIFF Files (*.tif);;All Files (*)", options=options)

        if file_name and slope_file:
            with rasterio.open(file_name) as src:
                self.dem_data = src.read(1)
                self.transform = src.transform

            with rasterio.open(slope_file) as src:
                self.slope_data = src.read(1)

             min_elev, ok1 = QInputDialog.getDouble(self, "Elevation
Input",    "Enter    minimum    elevation    for    GO    areas:",
float(np.min(self.dem_data)))
             max_elev, ok2 = QInputDialog.getDouble(self, "Elevation
Input",    "Enter    maximum    elevation    for    GO    areas:",
float(np.max(self.dem_data)))

            if ok1 and ok2:
                self.min_elev = min_elev
                self.max_elev = max_elev
                  self.go_area = (self.dem_data >= self.min_elev) &
(self.dem_data <= self.max_elev)
                self.valid_mask = self.go_area & (self.slope_data <=
45)
                 self.log(f"GO area pixels: {np.sum(self.go_area)} |
Valid mask pixels: {np.sum(self.valid_mask)}")
                self.plot_dem()
            else:
                self.log("Elevation input canceled.")
```

```
    def plot_dem(self):
        plt.clf()
        plt.imshow(self.valid_mask, cmap='Greens')
        plt.colorbar(label='Valid Path Region')
        plt.title("DEM Map with Valid Path Mask")

        if self.start:
                plt.plot(self.start[1], self.start[0], marker='o',
color='green', markersize=8, label='Start')
        if self.goal:
                 plt.plot(self.goal[1], self.goal[0], marker='o',
color='red', markersize=8, label='Goal')

        if self.start or self.goal:
            plt.legend()

        self.canvas.draw()

    def mouse_click_event(self, event):
        if event.xdata is None or event.ydata is None:
            return
        row, col = int(event.ydata), int(event.xdata)
        if self.go_area is not None and not self.go_area[row, col]:
           self.log("Selected point is not in GO area! Choose another
location.")
            return
        if self.start is None:
            self.start = (row, col)
            self.log(f"Start set at: {self.start}")
        elif self.goal is None:
            self.goal = (row, col)
            self.log(f"Goal set at: {self.goal}")
        self.plot_dem()

    def compute_path(self):
        import time
          if self.dem_data is None or self.slope_data is None or
self.start is None or self.goal is None:
             self.log("Load DEM + Slope, set elevation range, and
select start/goal points first!")
            return

        start_time = time.time()
        result = self.a_star_subpixel()
        end_time = time.time()
        computation_time = end_time - start_time


        if result:
            path, slopes = result
            self.plot_path((path, slopes))
            self.plot_elevation_profile(path)
```

```
            self.save_graphs()
            self.export_path_csv(path, computation_time)
            self.plot_zoomed_path((path, slopes))
            self.plot_visited_nodes()
        else:
            self.log("No valid path found!")

    def a_star_subpixel(self, substeps=4, angular_resolution=32):
        rows, cols = self.dem_data.shape
        start = (self.start[0] + 0.5, self.start[1] + 0.5)
        goal = (self.goal[0] + 0.5, self.goal[1] + 0.5)

        open_set = [(0, start)]
        heapq.heapify(open_set)
        came_from = {}
        g_score = {self._quantize(start): 0}
        self.visited_nodes.clear()

         directions = [(np.cos(theta), np.sin(theta)) for theta in
np.linspace(0, 2*np.pi, angular_resolution, endpoint=False)]

        def heuristic(a, b):
            return np.hypot(a[0] - b[0], a[1] - b[1])

        visited = set()

        while open_set:
            _, current = heapq.heappop(open_set)
            if heuristic(current, goal) < (2.0 / substeps):
                return self.reconstruct_path(came_from, current)

            key = self._quantize(current)
            if key in visited:
                continue
            visited.add(key)
            self.visited_nodes.add(key)

            for dx, dy in directions:
                neighbor = (current[0] + dx / substeps, current[1] +
dy / substeps)
                 if heuristic(neighbor, goal) >= heuristic(current,
goal):
                    continue
                r, c = int(neighbor[0]), int(neighbor[1])
                if (0 <= r < rows and 0 <= c < cols and
                     self.valid_mask[r, c] and self.slope_data[r, c]
<= 35):
                    neighbor_key = self._quantize(neighbor)
                     tentative_g = g_score[key] + np.hypot(dx, dy) /
substeps
                     if neighbor_key not in g_score or tentative_g <
g_score[neighbor_key]:
```

```
                        came_from[neighbor] = current
                        g_score[neighbor_key] = tentative_g
                         f_score = tentative_g + heuristic(neighbor,
goal)
                       heapq.heappush(open_set, (f_score, neighbor))

        return None

    def _quantize(self, point, precision=2):
             return  (round(point[0],  precision),  round(point[1],
precision))

    def reconstruct_path(self, came_from, current):
        path = []
        slopes = []
        while current in came_from:
            x, y = int(current[0]), int(current[1])
            path.append(current)
            slopes.append(self.slope_data[x, y])
            current = came_from[current]
        x, y = int(current[0]), int(current[1])
        path.append(current)
        slopes.append(self.slope_data[x, y])
        return path[::-1], slopes[::-1]

    def plot_path(self, path_slopes):
        path, slopes = path_slopes
        fig = self.canvas.figure
        ax = fig.gca()
        for line in ax.lines:
            line.remove()

          green_patch  =  plt.Line2D([0],  [0],  color='green',  lw=2,
label='0-10°')
         yellow_patch  =  plt.Line2D([0],  [0],  color='yellow',  lw=2,
label='11-25°')
       red_patch = plt.Line2D([0], [0], color='red', lw=2, label='26-
35°')

        for i in range(1, len(path)):
            x0, y0 = path[i - 1]
            x1, y1 = path[i]
            slope = slopes[i]
            if slope <= 10:
                color = 'green'
            elif slope <= 25:
                color = 'yellow'
            else:
                color = 'red'
            ax.plot([y0, y1], [x0, x1], color=color, linewidth=2)

               ax.plot(self.start[1],   self.start[0],   marker='o',
```

```
color='blue', markersize=8, label='Start')
                ax.plot(self.goal[1],   self.goal[0],   marker='o',
color='purple', markersize=8, label='Goal')
        ax.legend(handles=[green_patch, yellow_patch, red_patch,
                           plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='blue', label='Start'),
                           plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='purple', label='Goal')])
        self.canvas.draw()
        plt.savefig("Path_Plot.png")
        self.log("Saved path plot as Path_Plot.png")

    def plot_zoomed_path(self, path_slopes):
        path, slopes = path_slopes
        xs = [x for x, y in path]
        ys = [y for x, y in path]
        margin = 20
        xmin, xmax = int(min(xs)) - margin, int(max(xs)) + margin
        ymin, ymax = int(min(ys)) - margin, int(max(ys)) + margin

        fig, ax = plt.subplots(figsize=(8, 6))
        ax.imshow(self.valid_mask, cmap='Greens')
          green_patch = plt.Line2D([0], [0], color='green', lw=2,
label='0–10°')
         yellow_patch = plt.Line2D([0], [0], color='yellow', lw=2,
label='11–25°')
       red_patch = plt.Line2D([0], [0], color='red', lw=2, label='26–
35°')

        for i in range(1, len(path)):
            x0, y0 = path[i - 1]
            x1, y1 = path[i]
            slope = slopes[i]
            if slope <= 10:
                color = 'green'
            elif slope <= 25:
                color = 'yellow'
            else:
                color = 'red'
            ax.plot([y0, y1], [x0, x1], color=color, linewidth=2)

        ax.set_xlim(ymin, ymax)
        ax.set_ylim(xmax, xmin)
        ax.set_title("Zoomed-In Path View with Slope Coloring")
        ax.legend(handles=[green_patch, yellow_patch, red_patch])
        plt.savefig("Zoomed_Path.png")
        plt.close()
        self.log("Saved zoomed path as Zoomed_Path.png")

    def plot_visited_nodes(self):
        fig, ax = plt.subplots(figsize=(10, 8))
        ax.imshow(self.valid_mask, cmap='Greens')
```

```
        vx = [p[0] for p in self.visited_nodes]
        vy = [p[1] for p in self.visited_nodes]
        ax.scatter(vy, vx, color='red', s=1, label='Visited')
        ax.set_title("Visited Nodes During A* Search")
        ax.legend(loc="upper right")
        plt.savefig("Visited_Nodes.png")
        plt.close()
        self.log("Saved visited nodes map as Visited_Nodes.png")

    def plot_elevation_profile(self, path):
        elevations = [self.dem_data[int(x), int(y)] for x, y in path]
        slopes = [self.slope_data[int(x), int(y)] for x, y in path]
        colors = []
        for s in slopes:
            if s <= 10:
                colors.append('green')
            elif s <= 25:
                colors.append('yellow')
            else:
                colors.append('red')

        plt.figure()
        for i in range(1, len(elevations)):
              plt.plot([i-1, i], [elevations[i-1], elevations[i]],
color=colors[i], linewidth=2)
                  plt.text(i, elevations[i], f"{slopes[i]:.1f}°",
fontsize=8, rotation=45, color=colors[i], va='bottom', ha='right')

        plt.xlabel("Step")
        plt.ylabel("Elevation (m)")
        plt.title("Elevation Profile Along Path with Slope Coloring")
           green_patch = plt.Line2D([0], [0], color='green', lw=2,
label='0-10°')
         yellow_patch = plt.Line2D([0], [0], color='yellow', lw=2,
label='11-25°')
       red_patch = plt.Line2D([0], [0], color='red', lw=2, label='26-
35°')
        plt.legend(handles=[green_patch, yellow_patch, red_patch])
        plt.savefig("elevation_profile.png")
        plt.close()
         self.log("Saved elevation profile with slope coloring as
elevation_profile.png")

    def export_path_csv(self, path, computation_time):
        self.computation_time = computation_time
        rows = []
        for i, (x, y) in enumerate(path):
            lon, lat = rasterio.transform.xy(self.transform, int(x),
int(y))
            elevation = self.dem_data[int(x), int(y)]
            slope = self.slope_data[int(x), int(y)]
            rows.append([i, lat, lon, elevation, slope])
```

```
            df = pd.DataFrame(rows, columns=["Step", "Latitude",
"Longitude", "Elevation", "Slope (°)"])

        # Append metadata as a header-like block before saving
        meta_info = [
            ["Path Planning Algorithm", "Terrain-Constrained A*"],
            ["Start Pixel", str(self.start)],
            ["Goal Pixel", str(self.goal)],
           ["Computation Time (s)", f"{self.computation_time:.2f}"],
            []
        ]
        meta_df = pd.DataFrame(meta_info)

        with open("path_coordinates.csv", "w") as f:
            meta_df.to_csv(f, header=False, index=False)
            df.to_csv(f, index=False)

         self.log("Path coordinates exported to path_coordinates.csv
with slope and metadata.")

    def save_graphs(self):
        self.canvas.figure.savefig("dem_path.png")
        self.log("Graph saved as dem_path.png")

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = DEMPathPlanner()
    window.show()
    sys.exit(app.exec_())
```