IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

A Fast And Adaptable Algorithm To Optimize Ship Routing Using Machine Learning Techniques

[1]Dr. Meghana G R, [2]Gagan H, [3]Mohammed Faisal D, [4]Suhas S C, [5]Varun V S,

[1]Dept.of Information Science & Engineering,

[1]Jain Institute of Technology, Davangere, VTU, Davangere, Karnataka, India.

Abstract: The report presents a fast and adaptable ship routing algorithm leveraging machine learning techniques to optimize maritime navigation, considering real-time environmental factors. It addresses limitations of traditional rigid routing methods by incorporating dynamic data such as weather, sea currents, and vessel load to improve fuel efficiency, safety, and operational costs. The developed system demonstrates enhanced route planning performance, contributing towards sustainable and intelligent maritime transportation.

Index Terms **-** Ship Routing Optimization, Machine Learning, Maritime Navigation, Adaptive Routing Algorithm.

I. INTRODUCTION

Modern maritime transportation forms the backbone of international trade, with more than 80% of global cargo by volume moving along sea routes. As shipping operations grow in scale and complexity, the need for efficient, safe, and environmentally sustainable route planning is increasingly important. Traditional ship routing methods often rely on fixed paths and static parameters, leaving them ill-equipped to adapt to the real-time challenges of dynamic maritime environments such as sudden weather changes, fluctuating sea currents, and variations in vessel load. These inefficiencies can result in excessive fuel consumption, higher operational costs, increased greenhouse gas emissions, and compromised safety at sea.

To address these limitations, recent research emphasizes the development of intelligent, data-driven routing algorithms leveraging machine learning and hybrid optimization techniques. By incorporating real-time environmental data, vessel characteristics, and multi-objective optimization—including parameters like fuel efficiency, travel time, safety, and emission control—these advanced systems promise greater adaptability and operational reliability. The proposed research aims to develop a fast, scalable, and adaptive ship routing framework that not only enhances route efficiency and maritime safety but also contributes to the global objective of sustainable, low-emission shipping practices, aligning closely with recent advancements in the field.

II. METHODS AND MATERIAL

The methodology for this project centers around developing a fast, adaptive, and intelligent ship routing algorithm using machine learning combined with hybrid optimization techniques. The process begins with data collection and preprocessing, which involves gathering historical and real-time data on ship routes, environmental conditions such as wind, waves, and currents, vessel characteristics, and port information. This data is cleaned and normalized to handle missing or noisy inputs to ensure reliable algorithm performance.

Critical parameters affecting routing decisions—fuel consumption, weather conditions, travel time, safety margins, and emission levels—are modelled and weighted according to specific routing scenarios.

The core algorithm is designed as a hybrid optimization engine that integrates rule-based logic with machine learning models, allowing the system to dynamically adapt to changing maritime conditions while optimizing for multiple objectives. The system is trained on historical data to learn optimal routes under varying conditions and evaluated using metrics such as route efficiency, time savings, fuel consumption, and safety improvements. Two primary pathfinding algorithms are implemented: Dijkstra's algorithm for shortest path routing and an enhanced A-Star algorithm that incorporates environmental factors and dynamically calculated weather costs. The architecture consists of multiple layers—data, processing, and algorithmic computation, application via a Flask server, and presentation through a web interface—facilitating efficient data handling, route computation, and user interaction.

The project utilizes software tools including Python (with libraries such as NumPy, Pandas, Scikit-learn, TensorFlow/PyTorch for machine learning), and optionally MATLAB for simulation and visualization. Data sources include open maritime datasets, real-time marine traffic APIs, and weather data from NOAA, while simulations rely on GIS-based tools for performance validation. The system is optimized for computational efficiency to run on modern hardware equipped with adequate processing power and memory, ensuring scalability across different ship types and maritime zones.

III. Design and Implementation

The solution is architected as a multi-layered system designed to efficiently process geographical data, compute optimal maritime routes, and serve results via a web interface. The core functionality revolves around two primary layers: the Processing/Algorithm Layer and the Data Layer. System Architecture The system is organized into the following key layers, as depicted in the design diagram:

1. Data Layer: This layer is responsible for storing and providing all necessary geographical and environmental data.

• Source Data: Includes external data like GEBCO (General Bathymetric Chart of the Oceans) for depth, coastline shapefiles, and weather grid data.

• Stored Artifacts: Data is pre-processed and stored as csvdata/objects (pickle files) to enable fast loading. Key objects include the comprehensive Grid structure, KD-Tree, and bathymetry/weather mappings.

• Model Layer: The MODELS/ section represents the low-level data structure for Earth's surface analysis, including the Grid Cell model that stores latitude, longitude, bathymetry, land status, and weather information.

2.Processing/Algorithm Layer: This is the core computational layer where the route calculation occurs. It is defined by the Grid Generation and Filtration components.

• Grid Generation: This component manages the spatial indexing of the data. The Grid object is initialized by loading the pre- processed cell data and the KD-Tree. o Grid: A 2D array of Grid Cell objects, representing the sampled ocean surface. o Spatial Sampling + Exclusion: The initial process that creates the grid from GEBCO and coastline data, excluding areas that are land, too shallow, or near the coastline.

• Filtration (Pathfinding Algorithms): This component performs the pathfinding computation. o KD-Tree (Nearest Neighbor Search): Used to quickly find the closest valid Grid Cell for any given starting or ending latitude/longitude point. o Haversine Formula: Used in all algorithms to accurately calculate the great-circle distance between two lat/lon points on the globe, providing the core cost metric.

3. Application Layer (Flask Server): A Flask Server uses a web framework to receive user requests (start/end coordinates, initial speed/time) and calls the pathfinding methods in the Processing Layer.

4. Presentation Layer: The Webpage component is responsible for visualizing the input form, the computed path, and associated data to the user.

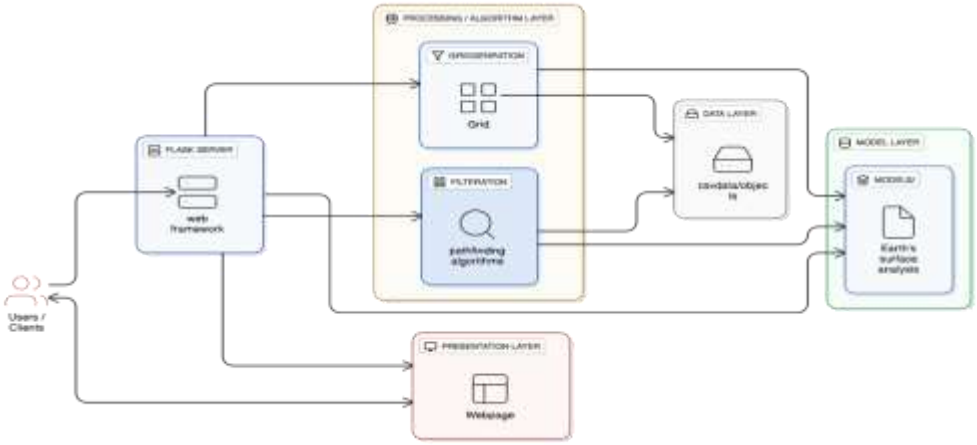

Fig 1: system design

Implemented Pathfinding Algorithms

Grid Generation Algorithm (Custom Spatial Grid)

Algorithm Type: Custom grid sampling & filtering algorithm using:
GEBCO + Shapely + Geopy.

Pseudocode:

Divides the map into cells (e.g., 1 km or 10 km resolution).
For each cell:
Extracts bathymetry depth from GEBCO NetCDF.
Checks if the point is on land (depth $\geqslant 0$).
Finds the nearest coastline using shapely.nearest_points().
Computes distance to coast with geopy.geodesic().
Mark cells within 22 km of the coast as “near coastline” to avoid during routing.

Used For:

Generating grid datasets used later in route planning (as input to pathfinding).

Algorithm Grid Generation:
Input: GEBCO bathymetry data, Coastline shapefile
Output: Grid CSV with bathymetry and land/sea flags

```
Begin
  Define grid resolution (km)
  Generate latitude range
  For each latitude do
    Compute longitude spacing using cos(latitude)
    For each longitude do
      Read bathymetry depth from GEBCO
      If depth ⩾ 0 then
        Mark cell as Land
      Else
        Mark cell as Sea
      Endif

      Find nearest coastline point
      Compute distance to coastline
      If distance < exclusion_radius then
        Mark cell as Near-Coastline
      Endif
```

```
        Store grid cell attributes
      End For
      Store column count for row
    End For
    Save grid data to CSV
End
```

KD-Tree (Nearest Neighbor Search)

Algorithm Type: Spatial Indexing / Nearest Neighbor Search.
Library: sklearn. neighbors. KDTree.

KD-Tree partitions space so that finding the closest point to a given coordinate happens in logarithmic time.

How it works here:

All grid points' (latitude, longitude) are converted into 3D Cartesian coordinates (Earth's sphere approximation).

A KD-Tree is built from these points (build_KDTree()).

Whenever a user gives a start or end coordinate, the system calls:

distance, index = kdtree.query([point])

to find the nearest valid grid cell — this ensures the start/end points align with the grid.

Used In: Used before pathfinding begins.

Pseudocode:

```
Algorithm Nearest_Cell_Search
Input: Latitude, Longitude
Output: Nearest Grid Cell

Begin
  Convert lat, lon to Cartesian coordinates
  Query KD-Tree for nearest neighbor
  Return corresponding grid cell
End
```

Haversine Formula

Algorithm Type: Mathematical distance formula on a sphere.
Purpose: Computes distance between two latitude/longitude points accounting for Earth's curvature.

How it works:

$$d = 2R \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\Delta\lambda}{2}\right)}\right) \qquad \text{(eq-1)}$$

Used In:

Dijkstra and A* path cost calculations.
Weather-influenced heuristic calculations.
Distance filtering to the coastline and between grid cells.

Pseudocode

```
Algorithm Haversine
Input: lat1, lon1, lat2, lon2
Output: Distance in kilometers
```

```
Begin
   Convert all angles to radians
   Apply the haversine formula
   Return R × central_angle
End
```

Dijkstra's Algorithm

Algorithm Type: Graph shortest path (uniform cost search).

Concept:

Starts from a source node (grid cell).
Explores all reachable neighbors with the smallest cumulative distance.
Always picks the cell with the lowest total distance seen so far (using a priority queue).
Stops when reaching the destination.

In this project:

The ocean grid is treated as a graph:
Each cell = node.
Distance between adjacent ocean cells = edge cost.
The algorithm avoids:
Land cells (is_land = True).
Shallow or near-coastline cells.
Distance between neighbors computed via Haversine formula.
Stores previous nodes to reconstruct the shortest path.

Pseudocode:

```
Algorithm Dijkstra_Routing
Input: Start cell, End cell
Output: Shortest path

Begin
   Initialize all distances to infinity
   Set start distance = 0
   Push start cell into priority queue

   While queue not empty do
      Extract cell with minimum distance
      If destination reached then break

      For each neighbor cell do
         If neighbor is sea and safe then
            Compute distance using Haversine
            If new distance < stored distance then
               Update distance and predecessor
               Push neighbor into queue
            EndIf
         EndIf
      EndFor
   EndWhile

   Reconstruct path using predecessor map
End
```

A* (A-Star) Pathfinding Algorithm

Algorithm Type: Heuristic shortest path search (extension of Dijkstra).
Concept:
A* combines:
g(n) = cost from start → current node
h(n) = heuristic estimate from current → goal
Final score:
$$f(n) = g(n) + h(n) \qquad \text{(eq-2)}$$
In this project:
g(n) = cumulative distance (same as Dijkstra)
h(n) = heuristic = distance to goal (Haversine)
weather-based cost using get_cost()
get_cost() considers:
Wave height (Thgt)
Wave direction (Tdir)
Wave period (Tper)
Ship heading vs wave direction
→ Calculates how hard or easy it is to travel through that cell (dynamic cost).

Pseudocode:
Algorithm A_Star_Routing
Input: Start, End, Ship speed, Weather grid
Output: Optimized route

Begin
  Initialize distance[start] = 0
  Push start node into priority queue

  While queue not empty do
    Extract node with lowest priority f(n)
    If close to destination then break

    For each valid neighbor do
      Compute travel distance
      Compute travel time
      Compute heuristic (distance to goal)
      Compute weather cost
      f(n) = α(g + h) + β(weather_cost)

      If new cost < stored cost then
        Update cost and predecessor
        Push neighbor into queue
      EndIf
    EndFor
  EndWhile

  Reconstruct path
End

Meaning:
80% path efficiency (distance/time)
20% environmental penalty (waves, direction, etc.)
Result:
A dynamic route that avoids rough seas, shallow areas, and land — more efficient and safer than Dijkstra.

Key Code:

```
cost = get_cost(r, c, angles[i], end_time, self.wgrid)
priority = 0.8*(heuristic + new_distance) + 0.2*(cost)
heapq.heappush(priority_queue, (priority, (r, c, end_time)))
```

Weather Cost Computation Model

Algorithm Type: Physics-based Cost Function.

Purpose: Quantify environmental resistance due to waves and direction.

In this project:
For each grid cell at a given hour:

Extract:
Wave height (Thgt)
Wave period (Tper)
Wave direction (Tdir)

Compute:

$$\text{speed} = 5 * \frac{(\text{wave\_height})^2}{\text{wave\_period}} \qquad \text{(eq-3)}$$

$$\text{dir} = -\frac{100}{\text{speed}} * \cos(heading - wave\_direction) \qquad (eq-4)$$

$$\text{cost} = \text{speed} + \text{dir} \qquad \text{(eq-5)}$$

This cost feeds into A*, so the ship tends to avoid regions with opposing or high waves.

Pseudocode:
Algorithm Weather_Cost
Input: Grid cell, Ship heading, Time
Output: Environmental cost

Begin
Read wave height, period, direction
Compute speed penalty using wave height
Compute directional resistance
Return combined cost
End

IV. RESULTS AND DISCUSSION

The development and implementation of a fast and adaptive ship routing algorithm using machine learning are expected to yield several impactful outcomes. These outcomes are designed to address the core limitations of existing systems and promote sustainable, efficient, and intelligent maritime navigation.
Adaptive Routing Algorithm: A validated and deployable ship routing algorithm capable of processing environmental and operational data to make intelligent routing decisions. The algorithm will dynamically adjust the optimal path based on changing sea conditions, vessel status, and fuel considerations. Improved Route Efficiency Significant improvements in route optimization leading to:

• Reduced travel time
• Lower fuel consumption
• Minimized emissions This translates directly to cost savings and more eco-friendly operations for shipping companies.

Enhanced Maritime Safety By integrating weather data and risk factors (such as high wave zones, piracy zones, and traffic congestion), the system enhances onboard decision-making and ensures safer route planning, reducing the likelihood of maritime accidents.
Scalability and Flexibility The system is designed to be scalable for:
• Different types of vessels (cargo, tanker, container ships)
• Various maritime regions (coastal, oceanic, high-risk zones) It offers flexibility to accommodate custom parameters and constraints, making it highly adaptable for real-world deployment.

Integration with Commercial Navigation Systems The proposed solution can be integrated with modern commercial maritime software and onboard navigation systems, potentially enhancing decision support systems (DSS) used by ship operators and fleet managers.

Contribution to Sustainable Maritime Practices By reducing carbon emissions and optimizing fuel use, the system supports the global movement toward greener shipping and complies with international regulations such as IMO (International Maritime Organization) environmental guidelines.

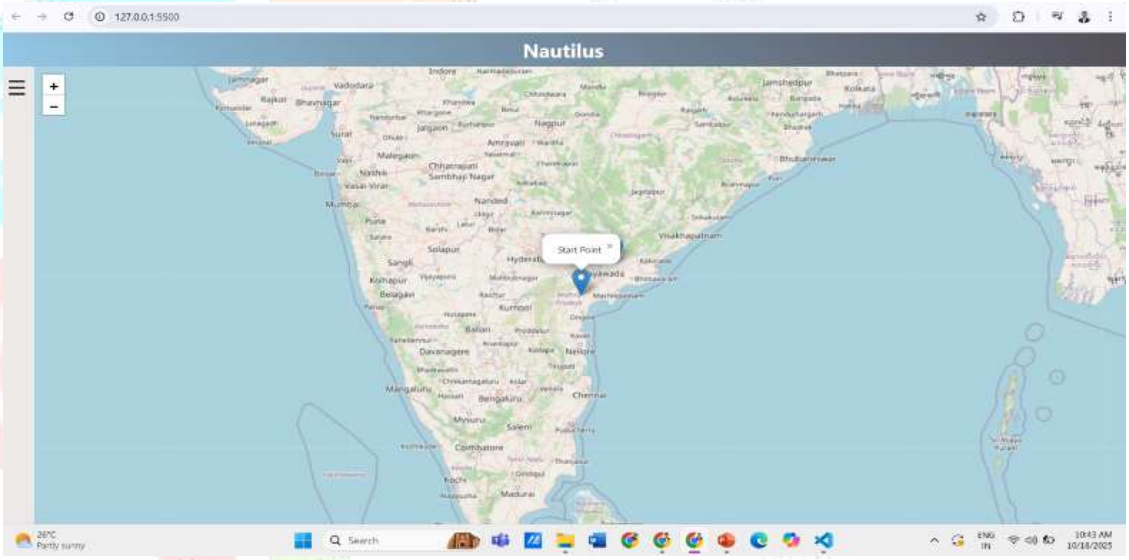

Fig 2: User Interface

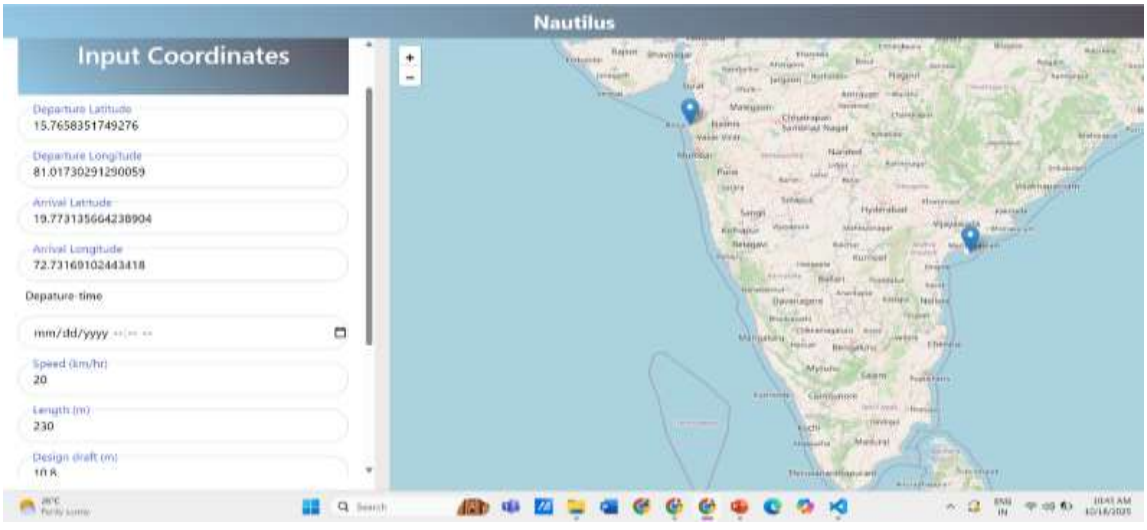

Fig 3: Nautilus user interface showing input coordinates and selected departure and arrival locations on the map.

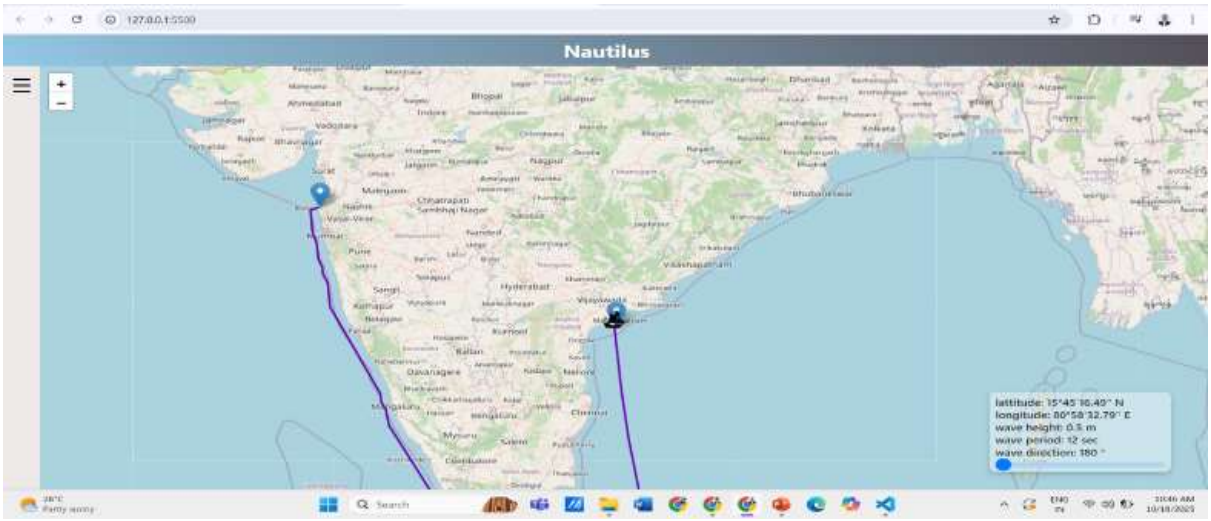

Fig 4: Generated Route

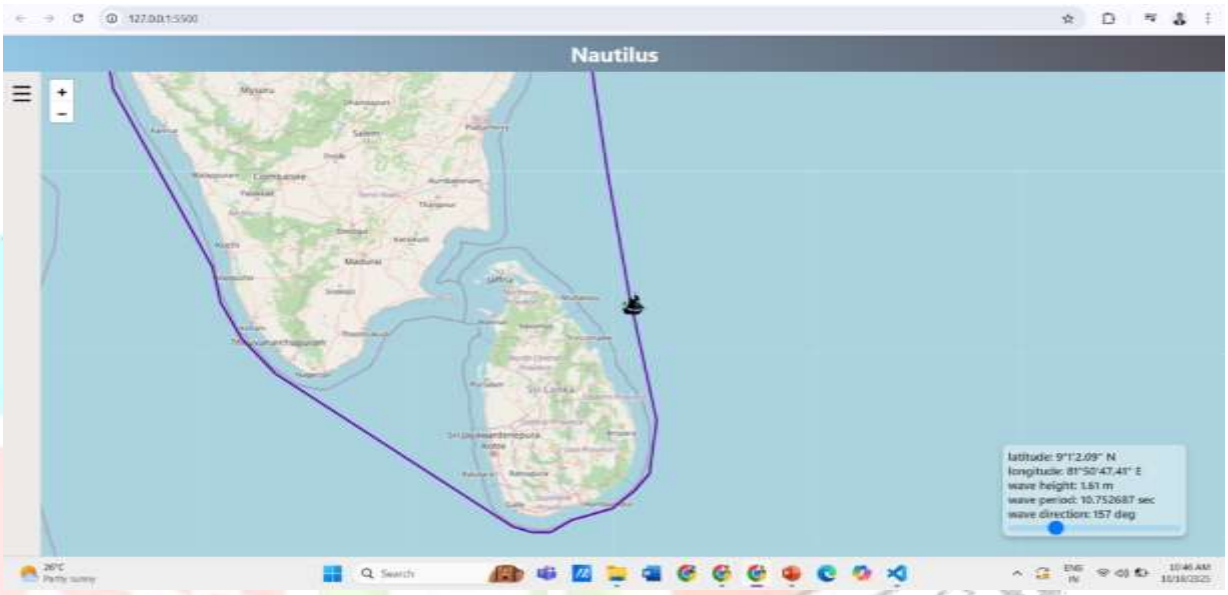

Fig 5: Optimized ship route generated by the Nautilus system with real-time weather information displayed at a selected waypoint.

Fig 6: Weather Data Integration

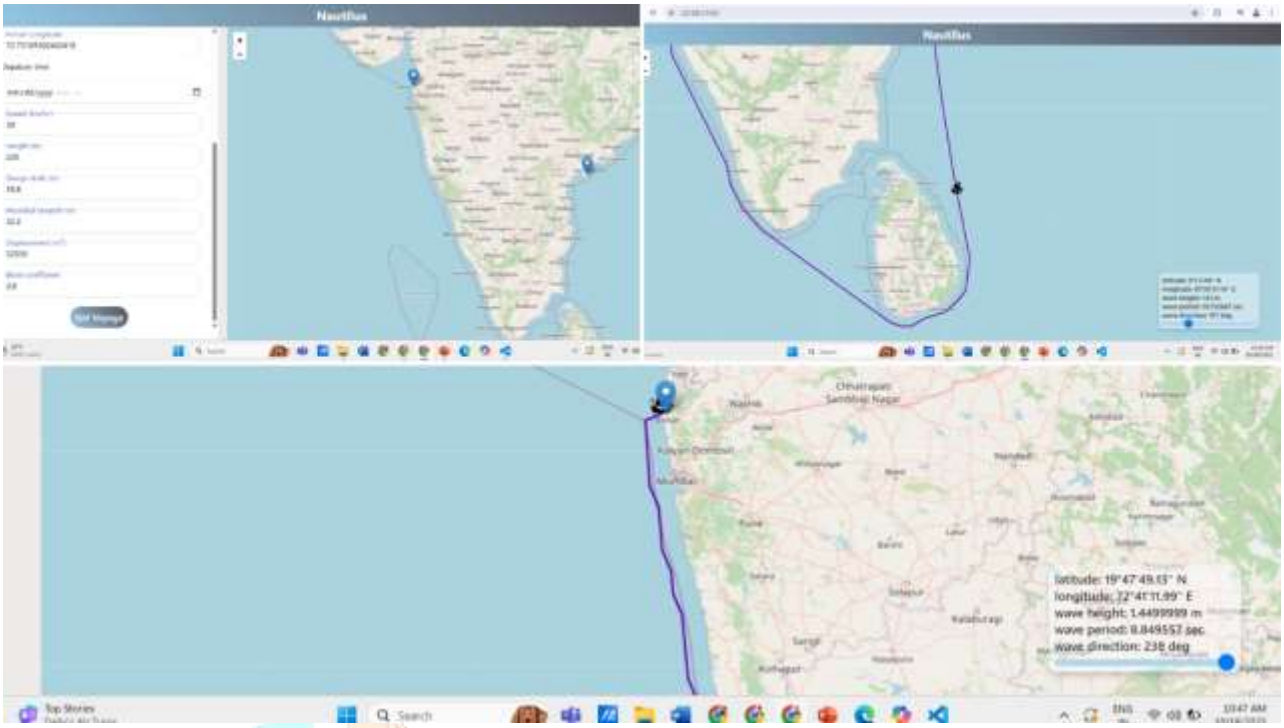

Fig 7: Interactive User Interface for Voyage Parameter Input and Optimized Maritime Route Visualization

V. Conclusion

Although a conclusion may review the main points of the paper, do not replicate the abstract as the conclusion. A conclusion might elaborate on the importance of the work or suggest applications and extensions. Authors are strongly encouraged not to call out multiple figures or tables in the conclusion these should be referenced in the body of the paper

VI. Acknowledgment

The authors express their sincere gratitude to their project guide, Prof. Dr. Meghana G. R., Department of Information Science & Engineering, JIT, Davangere, for her continuous guidance, valuable feedback, and encouragement throughout the development of this work.

The authors also thank Ms. Vaishnavi C. S. and Ms. Thanu M. R., Project Coordinators, for their support during the project period. Special thanks are extended to Dr. H. S. Saraswathi, Professor & Head, Department of IS&E, and Dr. Ganesh D. B., Principal & Director, JIT, for providing necessary facilities and institutional support.

The authors are grateful to the faculty members of the Department of IS&E for their helpful suggestions and support. Finally, the authors acknowledge the constant encouragement provided by their family and friends.

References

[1] Wang, H. (2018). Voyage optimization algorithms for ship safety and energy-efficiency (Licentiate thesis, Chalmers University of Technology). Retrieved. https://research.chalmers.se/publication/503070/file/503070_Fulltext.pdf.

[2] Zhu, Z., Shi, X., & Sun, J. (2024). Multi-Objective Route Planning Model for OceanGoing Ships Based on Bidirectional A-Star Algorithm Considering Meteorological Risk and IMO Guidelines. Applied Sciences, 14(17), 8029. **https://doi.org/10.3390/app14178029.**

[3] Xie, W., Xu, D., & Zhang, T. (2022). Deep Reinforcement Learning-Based Intelligent Ship Route Planning Under Uncertain Maritime Environment. Ocean Engineering, 254, 111245. **https://doi.org/10.1016/j.oceaneng.2022.111245.**

[4] Wei, Z., Wang, Z., & Chen, Y. (2021). AI-Powered Vessel Navigation Using Weather Aware Dynamic Routing Algorithms. IEEE Transactions on Intelligent Transportation Systems, 23(5), 4321–4332. **https://doi.org/10.1109/TITS.2021.3052983.**

[5] Kim, S., Lee, J., & Yeo, G.-T. (2020). Optimization of Maritime Routes with a Hybrid Genetic Algorithm Considering Emission Regulations. Transportation Research Part D: Transport and Environment, 82, 102323. https://doi.org/10.1016/j.trd.2020.102323.

[6] Liu, C., Ma, Y., Cao, C., & Yan, X. (2023). Ship Route Planning in the Pirate Area via Hybrid Probabilistic Roadmap Algorithm within the Context of the Maritime Silk Road. Journal of Navigation and Maritime Safety.

[7] Li, Z., Li, J., & Zhao, C. (2023). Path Planning of Coastal Ships Based on Improved Hybrid A-Star. In: Proceedings of the International Conference on Intelligent Transportation and Smart Cities. Springer.

[8] Tsou, M.-C. (2010). Discovering knowledge from AIS database for application in vessel traffic service. Expert Systems with Applications, 37(5), 3893–3901.

[9] Perera, L. P., Carvalho, J. P., & Guedes Soares, C. (2012). Fuzzy logic-based decision making system for collision avoidance of ocean navigation. Journal of Marine Science and Technology, 17(4), 534–548.

[10] Hinnenthal, J., & Clauss, G. F. (2010). Robust Pareto-optimal routing of ships utilising deterministic and ensemble weather forecasts. Ships and Offshore Structures, 5(2), 105–114.

[11] Mannarini, G., Oddo, P., & Pinardi, N. (2016). A prototype of ship routing decision support system for an operational oceanographic service. Transportation Research Part C, 69, 173–190.

[12] Lin, Y.-H., & Fang, M.-C. (2013). Route planning and collision avoidance for ships using genetic algorithms. Journal of Navigation, 66(5), 733–748.

[13] Chen, P., Huang, Y., Mou, J., & van Gelder, P. (2019). Ship collision candidate detection method: A velocity obstacle approach. Ocean Engineering, 170, 186–198.

[14] Tam, C., & Bucknall, R. (2010). Collision risk assessment for ships. Journal of Marine Science and Technology, 15(3), 257–270.

[15] IMO. (2018). Guidelines on voyage planning. International Maritime Organization, London.
[16] GEBCO Compilation Group. (2023). GEBCO 2023 Grid.
https://www.gebco.net

[17] National Oceanic and Atmospheric Administration (NOAA). (2023). Marine weather and oceanographic data.

https://www.noaa.gov

[18] OpenStreetMap Contributors. (2023). OpenStreetMap data for coastal and maritime navigation. https://www.openstreetmap.org