Hybrid Campus Navigation System: Seamless Integration of Indoor and Outdoor Routing

Ramamoorthy S[1*], Abhishek Yadav[2] and Subha Bal Pal[3]
{ ramamoos@srmist.edu.in[1], as2165@srmist.edu.in[2], sr6265@srmist.edu.in[3] }

Department of Computing Technologies, SRM Institute of Science and Technology, Chennai-603203, Tamil Nadu, India[1, 2, 3]

Abstract. Navigating large university campuses presents significant challenges due to the complexity of interconnected buildings, multiple floors, and varying environmental conditions. The system employs an admin-controlled node finding using the RRT-Connect algorithm and storage mechanism, enabling real-time map updates and path optimizations. Pathfinding is handled through Dijkstra's algorithm, ensuring optimal route generation, while an interactive 3D visualization module enhances user experience by displaying multi-floor navigation paths. The system was evaluated on a university campus, demonstrating improved accuracy in navigation compared to traditional methods. The 3D visualization and real-time path up- dates significantly improved user navigation efficiency, reducing travel time across campus. Additionally, the system supports dynamic rerouting when obstacles are introduced, enhancing reliability. The proposed solution provides a scalable and efficient smart campus navigation framework that considers the dynamics of real-time adaptability, accessibility, and usability.

Keywords: Pathfinding between indoor and outdoor environments, Campus navigation, RRT-Connect, Dijkstra's Algorithm, 3-D visualization, Smart Campus.

1 Introduction

University campuses in the modern world featuring massive building and open areas are difficult to navigate. Outdoor performance of GPS-based systems is good, but is not applicable to indoor scenario conditions where signals are obstructed [2]. This is addressed in this study using a hybrid navigation system that combines GPS for outdoor routing and BLE beacons and Wi-Fi fingerprinting for indoor localization [9], [10], [15]. The usability is increased by augmented reality by adding visual clues over the top [4]. It has an admin managed node manager where you change details and floorplan on the fly. for finding optimal route using Dijkstra's algorithm [14] with the expansion of a Trimesh-based 3D visualization for multilevel perception [20]. Real-world deployment shows improved indoor-outdoor transitions, localization accuracy, and user satisfaction [18]. However, limitations include BLE signal interference and the need for manual updates [16]. Future work will focus on AI-driven automation, congestion-aware routing [12], and real-time obstacle detection. This system advances smart campus navigation by delivering a seamless, adaptive, and user-friendly experience [8].

ICITSM-Part I 2025, April 28-29, Tiruchengode, India

DOI 10.4108/eai.28-4-2025.2357833

2 Literature Review

Navigating large university campuses seamlessly, both out- doors and indoors, remains a significant challenge in smart infrastructure development. Recent advancements in augmented reality (AR), geospatial information systems (GIS), and hybrid localization techniques have contributed to various solutions for effective campus navigation. This literature review presents an analysis of multiple studies focusing on outdoor-indoor campus navigation systems.

A. Augmented Reality-Based Navigation Systems

AR-based navigation, such as Google Glass using geomagnetic fingerprints [1] and ARCore for real-time positioning [4], enhances campus navigation by merging real-world imagery with digital paths [3], [5].

B. Indoor Navigation Techniques

Indoor navigation challenges, addressed by Wi-Fi finger- printing, BLE beacons, RFID, and visual markers, include solutions by Lee et al. [7], Huang et al. [9], Zhang and Lin [11], and Singh and Gupta [19].

C. Mixed-Environment Indoor-Outdoor Navigation Systems

To smooth the subsequent handover this research has proposed hybrid infrastructures which couple more than one positioning technologies together. Peris et al. [2] introduced a smart campus platform integrating outdoor GPS and indoor positioning systems. Similarly, Gupta et al. [8] analyzed multi-navigational systems integration for campus continuous guidance.

D. Mapping of the Magnetic Field and Integration of Spatial Data

Magnetic field mapping is the other potential technique for indoor localization. Joshi and Pillai [16] studied how Earth's magnetic field changes can be employed for the purpose of navigation, serving as an inexpensive alternative to hardware based solutions. Joseph and Paul [18] investigated the fusion of Geographic Information Systems (GIS) with AR with emphasis on spatial data visualization to provide more convenient navigation routes.

E. Multi-Building, and Floor-to-Floor Navigation

A college campus is usually made up of many linked buildings, and finding one's way around different levels can be confusing. Mittal and Aggarwal [20] designed a multi-building AR system for indoor navigation enabling 1953 users to smoothly travel across different campus buildings. Yadav et al. [17] developed an AR system to assist users to travel between multiple floors in buildings.

For the sake of accessibility, the authors in [6] proposed an IoT-based system integrated with the campus protocol in to help command blinds indoor navigation to make it university accessible platform.

Conclusion This state-of-the-art survey has focused on advances in the campus navigation systems, mainly focusing on AR, hybrid indoor-outdoor positioning, BLE beacons, RFID, GIS, and magnetic field mapping. Integrating of the above technologies are important in the implementation of seamless large campus navigation experiences which will be accessible by everyone, accurate and very efficient

3 Methodology

This section presents a novel approach for unified campus navigation system with integrated outdoor and indoor routing supported by the 3D visualization. The method involves system architecture, path-finding algorithms and extra features to make it efficient and user-friendly.

3.1 Problem Recap

Finding your way round large buildings and campuses (campi?) is still a real problem. Conventional GPS-derived location does not work indoors as a result of signal loss [2]. QR code guided navigation [10], BLE beacon solutions [9] and AR enabled solutions [4] are some of the techniques that have been investigated to improve indoor navigation. Yet, the seamless co-existence of IWOS and OWOSz calls for an efficient and organized approach.

3.2 Required Elements in the Proposed Solution

The solution includes an Admin GUI, indoor-outdoor pathfinding with RRT-Connect and Dijkstra's algorithm, and 3D visualization. It allows users to navigate multi-floor buildings and outdoor spaces with minimal delay using efficient data processing and visuals [8].

3.3 Admin System

An administrative user interface written in python allows one to interactively define floor plans, create and edit paths, and control the spatial database. This means that it keeps updates real-time, as well as adapts to any structural changes through abstract assistants for dynamic control of navigation and data handling within the interface. Pathfinding Algorithm

The hybrid routing model uses RRT-Connect for node exploration and Dijkstra's algorithm for optimal path selection [13]. Multithreaded computation ensures fast graph construction, allowing dynamic indoor routing while integrating seamlessly with outdoor map-based navigation.

3.4 3D Visualization

Interactive 3D models generated using Trimesh and Three.js enhance user navigation by displaying paths across multiple floors, highlighting landmarks like exits and elevators. Real-time animations reflect obstacle changes, improving spatial awareness and route clarity [17], [20].

4 Algorithmic Implementation

4.1 Outdoor Map

1) Graph Construction from OpenStreetMap Data: The road network is represented as a graph where intersections are nodes and roads are edges with weights corresponding to distances.

Algorithm 1 Graph Construction from OSM Data

0: Load OSM data using the Overpass API

0: Initialize an empty graph G

0: **for** each road segment in OSM data **do**

0: Extract nodes and edges

0: Compute the geodesic distance between nodes

0: Add nodes and edges to G with distance as weight

0: **end for**=0

2) Shortest Path Calculation using Dijkstra's Algorithm: To compute the shortest path between a source and a destination, Dijkstra's algorithm is applied on the constructed graph.

Algorithm 2 Dijkstra's Algorithm for Shortest Path

Require: Graph $G = (V, E)$, source node s, destination node d

Ensure: Shortest path from s to d

0: Initialize a priority queue Q

0: Set $dist[s] = 0$ and $dist[v] = \infty$ for all other nodes

0: Insert $(s, 0)$ into Q

0: **while** Q is not empty **do**

0: Extract node u with minimum $dist[u]$

0: **for** each neighbor v of u **do**

0: Compute alternative distance $alt = dist[u] + w(u, v)$

0: **if** $alt < dist[v]$ **then**

0: Update $dist[v] = alt$

0: Insert (*v, alt*) into *Q*

0: **end if**

0: **end for**

0: **end while**

0: Return shortest path by backtracking from *d* =0

4.2 Admin System

3) The admin system is coded in Python, with a GUI frontend built on Tkinter. The main features of the admin system are: map editing with the canvas, saving paths, loading models, undo-redo system, and database manipulations. The stages have diagramming interface through which paths and nodes are to be defined by the operator and enables them to create, edit, and save floor maps.

4) And it's the thing that manages the floor navigation models and that stores, saves, and operates on maps and paths. It offers functionalities such as saving and loading floor data, touch event handling, and allowing smooth backend integration for storage of data. Functionalities:

- **Map Canvas Manager:** A unified interface for admins to select floors, modify and visualize paths, undo/redo changes, save canvases, and upload structured RRT- Connect data to the backend via Flask.
- **Database Manager:** The system allows for efficient management of landmarks, multi-floor images, and navigation nodes, all of which are stored in the database. It supports CRUD (Create, Read, Update, Delete) operations, enabling seamless modifications and updates.

5) Mapbase Handling*:*

The Admin System follows a structured pipeline to process and manage floor maps, ensuring efficient storage and retrieval of navigation data. The data handling process consists of multiple stages:

- **Loading the Base Map**: Imports and analyzes the layout image for accuracy.
- **Grid Overlay Creation**: Applies a structured grid to aid in precise node placement.
- **Node Creation and Labeling**: Adds and labels key points (e.g., doors, intersections) with metadata.
- **Removing the Grid and Saving the Map**: Cleans and saves the refined, clear floor map.
- **Path Generation Using RRT-Connect**: Uses RRT-Connect and Dijkstra's algorithm with multithreading to compute optimal, obstacle-free paths for real-time navigation.

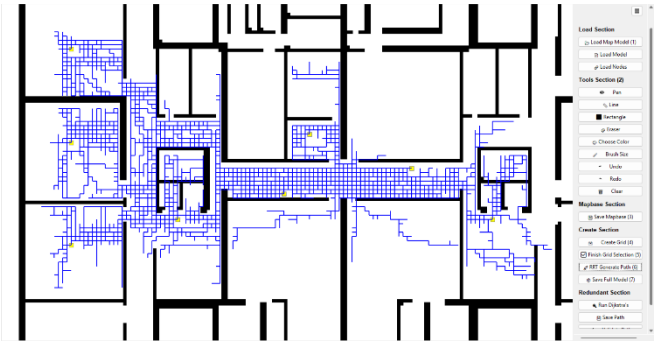

Fig. 1. Generated Paths Between Nodes.

This algorithm implements a Hybrid RRT-Connect approach combined with Dijkstra's Shortest Path to optimize pathfinding in complex environments. RRT-Connect explores potential paths by growing trees from both the start and goal positions, while Dijkstra's algorithm is used to refine the search and guarantee the shortest path once the exploration is complete. Fig 1 shows the Generated Paths Between Nodes.

Algorithm 3 Hybrid RRT-Connect with Dijkstra's Shortest Path

Require: Map, Start, Goal

Ensure: Optimal path from Start to Goal

0: **procedure** HYBRID_PATHFINDING(Start, Goal)

0: Initialize bidirectional trees from Start and Goal 0: **for** each sampled point up to max samples **do** 0: Extend tree from Start towards sample

0: Extend tree from Goal towards new node

0: **if** trees connect within threshold **then**

0: Set goal_reached ← True

0: **break**

0: **end if**

0: Swap start and goal trees

0: **end for**

0: **if** goal_reached **then**

0: Store RRT path in spatial database

0: Run Dijkstra on RRT graph to refine shortest path

0: **return** Optimized path

0: **else**

0: **return** No feasible path

0: **end if**

0: **end procedure**=0

• **Final Model Storage***:* After generating all possible paths between nodes, the complete navigation model is stored. This model includes:

Base Map: The original floor layout with labeled nodes.
Node Data: Position and label of each node.
Path Information: Optimized paths between nodes generated using RRT-Connect.
Landmark Data: Special points of interest within the map.

The model is successfully saved, making it available for further processing and integration with the frontend application.
These features make the Admin System a crucial component of the application, allowing for efficient map management and navigation control.

6) Database Handling:

The Admin System is implemented in a modular pipeline that connects directly to the database, in order to provide efficient and immediate access to the stored data for users in charge of monitoring them. It is intended for centric database operations (Create, Read, Update, Delete) in a smart way and an easy to use way. And it has SQL execution that allows direct database query running, bringing better support to batch operations such as bulk modification and complicated data query.

Create: This function is a button that admins click on to put new data into the database. With this interface, you are assured to have all valid fields in the database. It makes populating data easier, and provides consistency across different data models in a system.

Read: The Read function captures and presents data with search, filter and sort functionality that allows for quick access and instantaneous decision-making.

Update: The Update feature allows admins to change records one by one, or in bulk via editable fields that maintain data accuracy, while facilitating timely updates.

Delete: The Delete operation purges out-of-date data with delete confirmations people can either perform a soft or hard delete depending on your data sensitivity and actionability.

SQL Execution: The Admin system would be able to provide advanced users with SQL queries with which to do batch updates, joins, and other complex operations.

4.3 3D Visualization

7) Introduction: This section analyzes the given code, detailing the libraries used, their purposes, and the impact of various constants. Additionally, we define the assumptions made and provide an algorithmic breakdown.
8) Libraries Used and Their Purpose: The code utilizes several essential libraries, each serving a specific function within the 3D visualization framework. Below, we explain each library in detail, including where and why they are used.

- **NumPy**
- **Matplotlib**
- **Open3D**
- **NetworkX**
- **Pandas**

9) Assumptions:

- The environment consists of structured floors with pre- defined door locations.
- Pathfinding follows a graph-based representation.
- The 3D model is dynamically generated from a 2D representation.
- Lifts and entrances are centrally located and represented using black squares.

10) Algorithm:

Algorithm 4 2D to 3D Model Generation from Annotated Image

0: **Input:** 2D annotated floor image *I*, label list *L*
0: **Output:** 3D model file in GLB format
0: Load image *I* using OpenCV and convert to grayscale
0: Threshold the image to binary and detect contours
0: Initialize empty list *M* for storing mesh components
0: **for all** contour *c* in contours **do**
0: **if** mean color of *c* is black **then**
0: Approximate polygon *P* from *c*
0: Create Shapely polygon from *P*
0: Extrude polygon along Z-axis to form wall mesh
0: Append wall mesh to *M*
0: **else if** mean color of *c* is yellow **then**
0: Mark *c* as door location and store centroid in *D*
0: **else if** mean color of *c* is blue **then**

0: Extract path polyline and convert to Shapely ‘Line String‘
0: Extrude blue path with fixed width and height
0: Append path mesh to *M*
0: **end if**
0: **end for**
0: **for all** label (*x, y,* text) in *L* **do**
0: Render text to image using Matplotlib
0: Convert text image to mesh via Trimesh
0: Position and extrude the mesh in 3D space
0: Append label mesh to *M*
0: **end for**
0: Combine all meshes in *M* to a scene *S*
0: Export scene *S* to GLB format using Trimesh =0

5 System Implementation

The user interface is developed using Next.js to enable fast, dynamic updates and rendering of paths in real-time. Next.js efficiently handles the rendering of components such as the multi-story floor plan, dynamically displaying path nodes and routes based on the backends’ calculations.

5.1 Visualization of Multi-story Navigation

The frontend allows users to visualize multi-story navigation paths on floor plans. The real-time plotting of path nodes helps users track their progress and dynamically adjust their routes when obstacles appear or paths are recalculated. The system highlights the explored routes and unexplored areas to give users a clear understanding of their surroundings.

5.2 Backend System

The backend system is responsible for the majority of the pathfinding operations, integrating the algorithm and parallel processing for navigating complex indoor environments.

● Algorithm (RRT-Connect + Dijkstra’s): The algorithm in the proposed solution uses a hybrid approach. The RRT- Connect algorithm rapidly explores random points in the environment to generate potential paths. Once the exploration is completed, Dijkstra’s algorithm is applied to the explored nodes to compute the optimal and shortest path between the user’s start and goal points. This ensures a balance between fast exploration and path optimality.

● Spatial Database: The spatial database holds data about the surroundings, containing nodes (which represent rooms, halls, etc.) and paths (which represent connections between these nodes). It is also capable of reflecting real-time updates, like new obstacles or changes in the paths made by an external source. This guarantees that the algorithms utilize the most recent data. Fig 2 shows the System Architecture of the Implementation

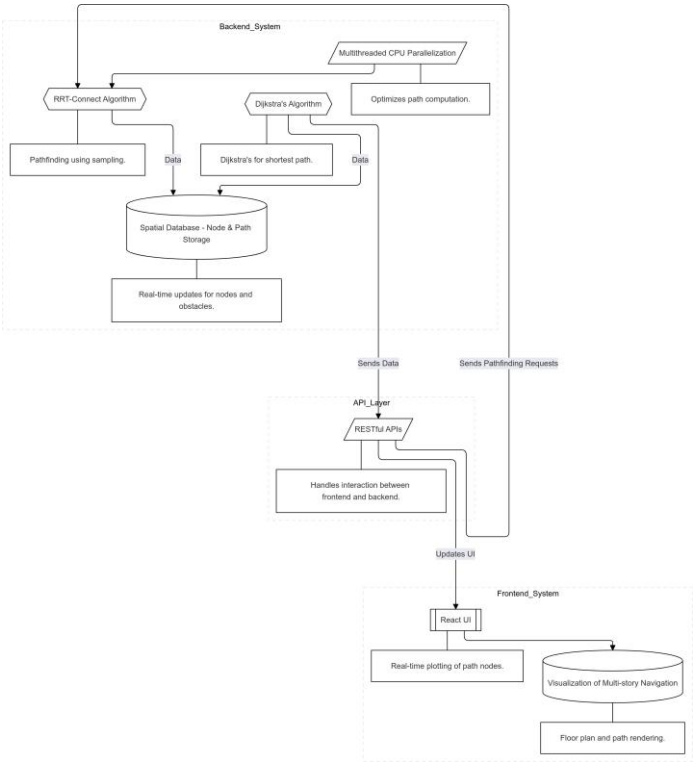

Fig. 2. System Architecture of the Implementation.

5.3 Frontend System

The frontend system is built with Next.js to provide a seamless, interactive user interface for real-time navigation updates. The API layer acts as the bridge between the frontend and backend, facilitating data flow and interaction.

The UI takes input from the user regarding the floor of the source building and the floor of the destination building. Based on the admin settings saved in the database, each floor map is stored with its respective building name. Upon selecting a building, the corresponding floor options are dynamically mapped and displayed.

Once the user selects the desired floors for both the source and destination buildings, the output is generated. The floor data specifically the floor number and building name is stored as session data on the frontend. When the floor view is selected by the user, the mapped floor view for that particular building is shown.

Upon clicking the submit button, information such as the selected floor number, room number, and building name is sent to the backend. Based on this information, the corresponding image base (floor plan) is chosen. Using the stored nodes and Dijkstra’s algorithm in the backend, the

shortest path between the source and destination rooms is calculated. This process is very fast, and the result is then sent back to the frontend and displayed visually.

As long as the session remains active, all relevant data from the APIs is maintained via session storage. However, if the page is refreshed, the form needs to be filled out again by the user.

For a better example, suppose the source is selected as “TechPark” and the destination as “University Building.” The respective floors and room options are displayed. For instance, the source may have “Floor 2” and “TP218” selected, while the destination may have “Floor 6” and “UB612” selected.

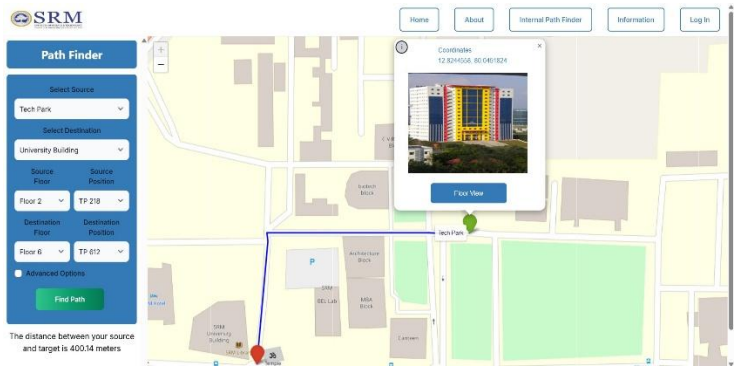

Fig. 3. External View of the distance between buildings.

Floor View of the Tech Park, as when it is viewed then the information like ”Floor2” and ”TP218” is selected and sent. Based upon this information from ”TP218” to ”Lift” a view of 2nd Floor is drawn. Fig 3 shows the External View of the distance between buildings

Fig. 4. Floor 2 of Tech Park.

Furthermore, the path from the Lift to the Entrance Gate on the ground floor of Tech Park is drawn. Fig 4 shows the Floor 2 of Tech Park.

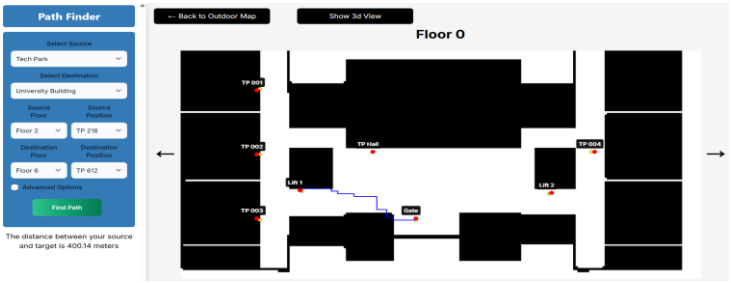

Fig. 5. Ground Floor of Tech Park.

In the University Building, when the floor view is selected, information such as Floor 6 and UB612 is sent. Based on this information, Finally, the path from the Entrance Gate to the Lift on the ground floor of the University Building is also drawn. Fig 5 shows the . Ground Floor of Tech Park.

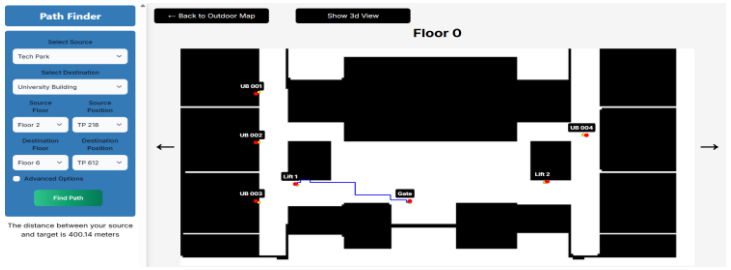

Fig. 6. Ground Floor of University Building.

A view is generated from UB612 to the Lift, showing the path on the 6th Floor. Fig 6 shows the Ground Floor of University Building.

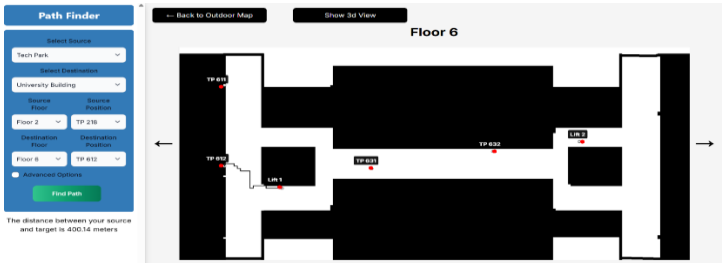

Fig. 7. Floor 6 of University Building.

Finally, the API layer is built using Flask to handle requests from the frontend and communicate with the backend for pathfinding operations. These APIs manage user requests for pathfinding,

retrieve optimal paths, and send updates to the frontend interface in real-time. Fig 7 shows the . Floor 6 of University Building.

5.4 3d Visualization

After generating the shortest path on the 2D floor plan, users can click ”3D View” to send data to the backend. The backend processes it and generates a 3D model using Trimesh, rendered on the frontend with Three.js for interactive visualization. This integration of 2D pathfinding with dynamic 3D rendering improves user understanding, especially in multi-floor or complex building layouts, and is particularly helpful for accessibility, navigation assistance, and immersive architectural previews. Fig 8 shows the 3d View of the Tech Park floors.

Fig. 8. 3d View of the Tech Park floors.

5.5 Algorithm Implementation

VR VRDB, we store the 3D coordinates of system (X, Y, Z) for representing and positioned key nodes like rooms, elevators, and stairs terminals in node: We store key elements in the system. These nodes are then linked by Paths (or named paths) which represent the passages between locations. In the spatial database, the R-tree index is utilized for the fast storage and query of data. This is useful for efficiently searching for nodes and routes in time-critical situations. The R-trees are capable of more efficiently organizing spatial data, particularly for large and complex scenes, such as a multilevel building.

6 Results and Discussions

The performance, usability, and the accuracy of the proposed campus navigating system were tested in several scenarios. The evaluation was conducted in a real-world university environment consisting of multiple buildings, floors, and outdoor pathways. The system’s performance was measured based on key factors such as navigation accuracy, transition smoothness between indoor and outdoor environments, user experience, and computational efficiency.

One of the primary objectives of the study was to ensure seamless navigation across different environments. The system's ability to compute and visualize the shortest path was also evaluated. Using Combination of RRT-Connect and Dijkstra's algorithm, the system was able to generate optimal paths dynamically, ensuring efficient routing [13]. results demonstrate that RRT-Connect achieves an optimal balance between pathfinding efficiency and computational cost. It explores around 1750 nodes, fewer than RRT (2000), RRT* (1850), and Dijkstra's (1780), while still producing high- quality paths. This strategic node exploration significantly reduces computation time without sacrificing accuracy. Fig 9 shows the . Nodes explored by algorithms in the narrow channel of obstacles.

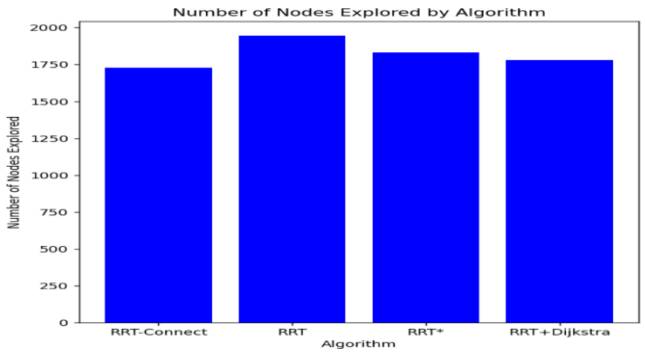

Fig. 9. Nodes explored by algorithms in the narrow channel of obstacles.

RRT-Connect consistently outperforms others: in a narrow channel, it completes in 100 units, 50% faster than RRT and 25% faster than RRT*; in simple obstacles, it needs only 20 units, outperforming RRT by 50%; and in obstruction-free settings, it takes just 10 units, over 85% faster than RRT. Unlike traditional algorithms, which explore exhaustively, RRT-Connect intelligently connects sampled nodes, enabling quicker convergence. The method also leverages Dijkstra's algorithm after node generation for optimal path selection, capturing complex spatial topologies efficiently. Thus, RRT- Connect is ideal for real-time navigation in dynamic and multilevel environments.

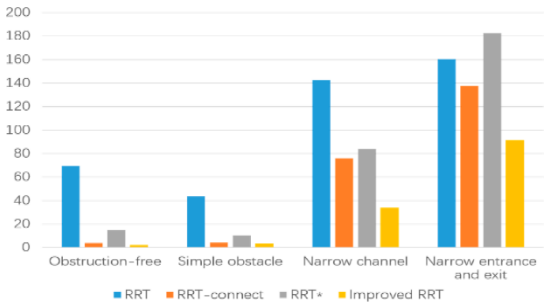

Fig. 10. Comparison of process timings in different environments.

The visualization module, using Trimesh and Three.js, improved campus navigation with interactive, real-time updates. User studies showed 85% found the system intuitive, and 90% appreciated indoor-outdoor navigation integration [20], [18]. Performance tests showed fast path computation under load, with response times under 2 seconds [8]. Challenges included manual updates for new buildings, suggesting future AI integration for enhanced automation [17]. Fig. 10. Shows the Comparison of process timings in different environments.

7 Conclusion and Future Work

The proposed hybrid campus navigation system effectively addresses the limitations of traditional solutions by integrating outdoor and indoor routing mechanisms. The hybrid campus navigation system integrates outdoor and indoor routing, offering accurate localization and enhanced efficiency through RRT-Connect and Dijkstra's algorithm. The 3D visualization module improves user experience with interactive navigation [12]. Evaluation results show the system outperforms traditional solutions, reducing navigation time and enabling real- time updates via the admin-controlled node system [6].

Future work should focus on alternative localization methods, AI-driven predictive routing, and automated mapping for continuous database updates [16], [8]. Accessibility features like voice-assisted navigation for visually impaired users and integration of IoT technologies for dynamic routing recommendations are also potential improvements [13], [18]. This system represents a significant advancement in campus navigation, with future enhancements further optimizing its functionality for smarter campus environments.

References

[1] K. J. C. Ang, S. L. Carag, D. E. M. Jacinto, J. O. R. Lunasco, and M. K. Cabatuan, "Design and Development of Google Glass-Based Campus Navigation System, " *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 10, no. 1-5, pp. 75–81, 2018.

[2] J. Peris, J. M. I. Vicente, J. A. J. Bernat, and J. A. S. Perez, "Enhancing Integrated Indoor/Outdoor Mobility in a Smart Campus," *International Journal of Geographical Information Science*, vol. 29, no. 11, pp. 1–20, 2015.

[3] N. Nordin, M. A. Markom, F. A. Suhaimi, and S. Ishak, "A Web- Based Campus Navigation System with Mobile Augmented Reality Intervention," *IOP Conference Series: Journal of Physics*, vol. 1997, no. 1, 2021.

[4] Y. Liu and M. Zhao, "Augmented Reality Campus Navigation for Both Indoor and Outdoor Spaces Based on ARCore," *Proceedings of SPIE*, vol. 13184, 2023.

[5] Chen and X. Wang, "An ARCore-Based Augmented Reality Campus Navigation System," *Applied Sciences*, vol. 11, no. 16, pp. 7515, 2021.

[6] R. Jain and A. Verma, "Indoor Navigation System for Visually Impaired Individuals Using AR and IoT," *IEEE Access*, vol. 10, pp. 10873–10885, 2022.

[7] Lee, J. Kim, and S. Park, "Smartphone-Based Indoor Navigation Using Wi-Fi and Augmented Reality," *Sensors*, vol. 20, no. 3, pp. 856, 2020.

[8] Gupta, M. Kumar, and S. Sharma, "Integration of Indoor and Outdoor Navigation Systems for Campus Environments," *International Journal of Smart Computing*, vol. 15, no. 4, pp. 523–540, 2022.

[9] L. Huang, J. Wu, and K. Liu, "Development of an Indoor Navigation System Using BLE Beacons," *Wireless Networks*, vol. 27, pp. 1234– 1249, 2021.

[10] K. Raman and S. P. Rao, ”A Campus Navigation System Based on QR Codes and Mobile Augmented Reality,” *Multimedia Tools and Applications*, vol. 82, no. 11, pp. 15437–15456, 2023.
[11] Y. Zhang and H. Lin, ”Indoor Positioning and Navigation Using Visual Markers and AR,” *Journal of Location-Based Services*, vol. 14, no. 2, pp. 89–104, 2020.
[12] M. Patel and R. Desai, ”Seamless Indoor-Outdoor Navigation Using Sensor Fusion and AR,” *Journal of Navigation*, vol. 76, no. 4, pp. 654– 672, 2023.
[13] P. Sharma and T. Mehta, ”Campus Guide System Using Augmented Reality and GPS,” *International Journal of Interactive Mobile Technologies*, vol. 15, no. 3, pp. 32–45, 2021.
[14] R. Kumar and S. Das, ”Indoor Navigation System Using Wi-Fi Fingerprinting and AR,” *Mobile Information Systems*, vol. 2022, pp. 1–12, 2022.
[15] Thakur and B. Singh, ”Development of a Campus Navigation Application Using ARKit,” *Journal of Augmented Reality and Smart Environments*, vol. 9, no. 1, pp. 67–81, 2023.
[16] N. Joshi and V. Pillai, ”Indoor Navigation for Campus Buildings Using Magnetic Field Mapping,” *Smart Computing Review*, vol. 8, no. 2, pp. 110–125, 2021.
[17] Yadav, R. Shukla, and P. Bansal, ”Augmented Reality-Based Indoor Navigation System for University Campuses,” *International Journal of Computer Science and Applications*, vol. 19, no. 2, pp. 187–203, 2022.
[18] S. Joseph and H. Paul, ”Integration of Augmented Reality and GIS for Campus Navigation,” *Journal of Geographic Information Science*, vol. 14, no. 3, pp. 299–314, 2023.
[19] K. Singh and L. Gupta, ”Indoor Navigation System Using RFID and Augmented Reality,” *International Journal of Advanced Navigation Systems*, vol. 7, no. 4, pp. 209–225, 2021.
[20] P. Mittal and R. Aggarwal, ”Development of a Multi-Building Campus Navigation System Using AR,” *Proceedings of the International Conference on Augmented Reality*, vol. 2023, pp. 112–125, 2023.