International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Ar – Based Indoor Navigation System

Mr. Musharaff Sheriff M C[1], Mr. Kumaran G L[2], Mr. Jeeva K[3], Prof. Meena V[4]

Abstract

Navigation plays an important role in daily life, especially in complex indoor environments such as educational institutions, hospitals, shopping malls, airports, and corporate buildings. Although outdoor navigation has been effectively addressed using GPS-based systems, indoor navigation remains a challenging problem due to signal attenuation, multipath effects, and the lack of precise indoor maps, often causing users to experience difficulty in locating specific rooms or facilities. With the advancement of mobile computing and computer vision technologies, Augmented Reality (AR) has emerged as an effective solution by overlaying virtual information directly onto the real environment, thereby providing an intuitive and user-friendly navigation experience. The AR Indoor Navigation System proposed in this project uses ARCore and vision-based localization instead of traditional infrastructure-based methods. A three-dimensional spatial map of the indoor corridor is created using the Multiset mapping application, processed in the cloud, and integrated into the Unity environment to enable accurate real-time localization and navigation.

CHAPTER 1
INTRODUCTION

1.1 Overview

Navigation plays an important role in daily life, especially in complex indoor environments such as educational institutions, hospitals, shopping malls, airports, and corporate buildings. Although outdoor navigation has been effectively addressed using GPS-based systems, indoor navigation remains a challenging problem due to signal attenuation, multipath effects, and the lack of precise indoor maps, often causing users to experience difficulty in locating specific rooms or facilities. With the advancement of mobile computing and computer vision technologies, Augmented Reality (AR) has emerged as an effective solution by overlaying virtual information directly onto the real environment, thereby providing an intuitive and user-friendly navigation experience. The AR Indoor Navigation System proposed in this project uses ARCore and vision-based localization instead of traditional infrastructure-based methods. A three-dimensional spatial map of the indoor corridor is created using the Multiset mapping application, processed in the cloud, and integrated into the Unity environment to enable accurate real-time localization and navigation.

1.2 Objective

The primary objectives of this project are:

- To design and develop an AR-based indoor navigation system for Android devices.
- To enable accurate real-time localization using vision-based mapping techniques.
- To provide intuitive arrow-based navigation guidance overlaid on the real-world environment.
- To display real-time distance information to the destination.

- To ensure smooth performance and usability on mobile devices.
- To develop a cost-effective and scalable solution for indoor navigation in large buildings.

CHAPTER 2
LITERATURE SURVEY

This chapter presents a comprehensive review of existing research works and foundational technologies related to indoor navigation systems, augmented reality, vision-based localization, and spatial mapping. The survey focuses on recent advancements in AR-based indoor navigation, various localization approaches, mapping techniques, and user guidance methodologies. These studies provide the theoretical and technical foundation upon which the proposed AR Indoor Navigation System is designed.

[1] M. Lu et al., "Indoor AR Navigation Framework Based on Geofencing and Image-Tracking with Accumulated Error Correction," Applied Sciences, vol. 14, no. 10, May 2024.

Lu et al. (2024) introduce an indoor AR navigation framework that combines geofencing and image tracking with accumulated error correction mechanisms. The authors address one of the key challenges in AR navigation—pose drift and accumulated tracking errors. Their work demonstrates that continuous correction using visual references significantly improves navigation stability. This reinforces the importance of vision-based localization and map matching used in the proposed Multiset-based approach.

[2] I. K. D. S. et al., "Adaptive AR Navigation: Real-Time Mapping for Indoor Environment Using Node Placement and Marker Localization," Information, vol. 16, no. 6, 2024.

This study presents an adaptive AR navigation system based on real-time mapping and marker-based localization. The authors emphasize dynamic environment handling and real-time updates. Although marker-based, the work demonstrates the importance of reliable reference points for accurate AR alignment. This supports the concept of using spatial anchors and persistent reference features in the proposed system.

[3] R. Zhao, W. Yu and Z. Huang, "A Design of Indoor Navigation System Based on Augmented Reality and Computer Vision," IEEE Access, 2024.

Zhao et al. (2024) present an AR navigation system that uses computer vision techniques for localization and guidance. Their work demonstrates that vision-based AR navigation achieves higher accuracy and better user experience than sensor-only approaches. This directly aligns with the vision-based Multiset and ARCore approach used in this project.

[4] A. Alkady et al., "SINS_AR: An Efficient Smart Indoor Navigation System Based on Augmented Reality," IEEE Access, 2024.

Alkady et al. (2024) present a smart indoor navigation system combining AR visualization with intelligent routing. Their work emphasizes system efficiency, scalability, and real-world deployment. This study supports the practical feasibility of deploying AR indoor navigation systems in real buildings such as educational institutions.

[5] P. Hořejší et al., "Reliability and Accuracy of Indoor Warehouse Navigation Using Augmented Reality," IEEE Access, 2024.

Hořejší et al. (2024) evaluate the reliability of AR navigation in warehouse environments. Their experimental results confirm that AR guidance can achieve high accuracy and operational reliability in structured indoor spaces. This supports the performance expectations and experimental evaluation of the proposed system.

[6] Z. Qiu et al., "NavMarkAR: A Landmark-based Augmented Reality Wayfinding System for En-

hancing Spatial Learning of Older Adults," arXiv preprint, Nov. 2023.
Qiu et al. (2023) propose a landmark-based AR wayfinding system designed to improve spatial learning, especially for elderly users. Their study shows that AR-based visual landmarks and cues significantly enhance user confidence and spatial understanding. This work supports the design choice of using clear, visual AR arrows instead of abstract map-based guidance.

[7] J.-R. Jiang and H. Subakti, "An Indoor Location-Based Augmented Reality Framework," Sensors, vol. 23, no. 3, Jan. 2023.
Jiang and Subakti (2023) propose a general framework for indoor location-based AR systems that integrates localization, mapping, and AR visualization. Their work highlights the importance of accurate pose estimation and stable coordinate alignment between virtual and real environments. The framework demonstrates how AR content can be persistently anchored to physical spaces. This research directly supports the architectural design of the proposed system, particularly the integration of cloud-based localization and spatial anchoring.

[8] S. Reni Hena Helan et al., "Indoor Navigation Using Augmented Reality," International Journal of Engineering Research & Technology (IJERT), vol. 11, no. 03, Jun. 2023.
Helan et al. (2023) describe an AR-based indoor navigation system that provides visual guidance using smartphones. The authors highlight the limitations of GPS indoors and demonstrate how AR can bridge this gap. Their implementation confirms that overlaying navigation information directly on the camera view improves user experience and navigation accuracy. This work further validates the core idea of the proposed AR Indoor Navigation System.

[9] F. Serhan Daniş et al., "An Indoor Localization Dataset and Data Collection Framework with High Precision Position Annotation," arXiv preprint, 2022.
Daniş et al. (2022) focus on the challenges of obtaining high-quality datasets for indoor localization research. Their work highlights the importance of accurate ground truth data and robust feature extraction. This research underlines the necessity of high-quality environment scanning and mapping, which is a critical requirement for the Multiset-based mapping used in this project.

[10] A. Martin, J. Cheriyan, J. J. Ganesh, J. Sebastian and J. V. Jyakrishna, "Indoor Navigation using Augmented Reality," EAI Endorsed Transactions on Creative Technologies, vol. 8, no. 26, Feb. 2021.
Martin et al. (2021) present an AR-based indoor navigation system that overlays virtual navigation cues on real-world environments to improve user wayfinding. Their work emphasizes the usability advantage of AR compared to traditional 2D maps, especially in complex indoor spaces. The authors demonstrate that AR-based visual guidance significantly reduces user confusion and navigation time. This study strongly supports the idea of using arrow-based AR guidance in the proposed system and validates the effectiveness of intuitive visual navigation cues.

[11] T. Kang and Y. Shin, "Indoor Navigation Algorithm Based on a Smartphone Inertial Measurement Unit and Map Matching," arXiv preprint, Sep. 2021.
Kang and Shin (2021) explore indoor navigation using inertial sensors combined with map matching. While their approach is sensor-driven rather than vision-based, the study highlights the importance of map constraints and path correction. This reinforces the role of pre-built maps and constrained navigation spaces, similar to the NavMesh-based path planning used in the proposed system.

[12] Z. Yang, Z. Wei and H. Yang, "An AR-Based Indoor Navigation Vision-Based Location Positioning for Smartphone Users," in Proc. 2018 International Conference on Intelligent Transpo-

rtation Systems, pp. 1123–1128, 2018.

Yang et al. (2018) propose one of the earlier vision-based AR indoor navigation systems. Their research shows that camera-based feature matching can provide acceptable indoor positioning accuracy. This work serves as an early foundation for modern cloud-based vision localization systems such as Multiset.

[13] F. Wang et al., "Joint Activity Recognition and Indoor Localization with WiFi Fingerprints," arXiv preprint, Apr. 2019.

Wang et al. (2019) present a WiFi fingerprint-based indoor localization approach integrated with activity recognition. Although effective, the authors note that signal-based methods suffer from environmental sensitivity and require extensive calibration. This study further justifies the decision to avoid infrastructure-dependent methods and instead adopt vision-based localization in the proposed system.

CHAPTER 3
SYSTEM ANALYSIS
3.1 EXISTING SYSTEM

Indoor navigation in large and complex buildings has traditionally been addressed using conventional methods such as static signboards, printed maps, and human assistance. While these approaches are simple, they are often inefficient, time-consuming, and prone to human error. Users unfamiliar with the building layout frequently experience confusion and waste significant time locating specific rooms or departments.

In recent years, several digital indoor navigation solutions have been proposed based on technologies such as Bluetooth beacons, RFID tags, Wi-Fi fingerprinting, and QR-code-based systems. Although these systems provide a certain level of automation, they suffer from several inherent limitations. Infrastructure-based systems require the installation and maintenance of additional hardware components throughout the building, which increases deployment cost and maintenance complexity. Furthermore, signal-based localization methods often suffer from accuracy issues due to signal interference, multipath effects, and environmental changes.

Some existing systems provide navigation guidance through two-dimensional maps displayed on mobile screens. However, interpreting 2D maps and translating them into real-world movement is cognitively demanding, especially for first-time users. This results in poor user experience and reduced navigation efficiency. Additionally, most existing systems do not provide real-time, context-aware visual guidance aligned with the user's actual surroundings.

Thus, the existing systems can be summarized to have the following limitations:

- Dependence on additional infrastructure such as beacons or tags.
- High installation and maintenance cost.
- Limited localization accuracy and reliability.
- Lack of intuitive and immersive user guidance.
- Increased cognitive load due to reliance on 2D maps.
- Poor scalability and adaptability to layout changes.

These limitations highlight the need for a more intelligent, accurate, and user-friendly indoor navigation solution.

3.2 PROPOSED SYSTEM

The proposed AR Indoor Navigation System introduces a vision-based, infrastructure-free approach for indoor navigation using Augmented Reality technologies. The system leverages ARCore and Multiset

mapping to achieve accurate real-time localization and intuitive navigation guidance without requiring any additional physical infrastructure such as beacons or markers.

In the proposed system, the indoor environment is first scanned using the Multiset application to generate a detailed three-dimensional spatial map. This map is uploaded to the Multiset cloud and later imported into the Unity development environment using the Multiset SDK. Navigation paths are generated using Unity's NavMesh system over the reconstructed 3D environment.

When the user launches the application, the camera is used to scan the surrounding environment. The system performs vision-based localization by matching the live camera feed with the stored spatial map. Once localization is successful, the user can select a destination from the list of predefined Points of Interest (POI). The system computes the optimal path and overlays augmented reality arrows along the real-world corridor. Additionally, the real-time distance to the destination is continuously displayed. When the user reaches the destination, a confirmation message is shown.

The proposed system offers the following advantages:

- Infrastructure-free operation using vision-based localization.
- High accuracy and reliability in indoor positioning.
- Intuitive and immersive AR-based navigation guidance.
- Reduced cognitive load compared to traditional 2D map-based systems.
- Cost-effective and scalable solution for large buildings.
- Easy adaptability to environmental changes by rescanning the area.

3.3 IDEATION AND BRAINSTORMING

The ideation phase of the project focused on identifying the fundamental challenges associated with indoor navigation and exploring modern technological solutions to address them. Brainstorming sessions were conducted to analyze existing navigation methods, their shortcomings, and potential improvements using emerging technologies.

Several ideas such as QR-code-based navigation, Bluetooth beacon-based systems, and Wi-Fi fingerprinting were initially considered. However, these approaches were found to be either infrastructure-dependent or insufficiently accurate. The idea of using Augmented Reality was then explored due to its ability to provide intuitive, real-world-aligned guidance.

Further brainstorming led to the decision to adopt a vision-based localization approach using ARCore and Multiset mapping, as this eliminates the need for additional hardware and provides higher accuracy. The use of Unity was finalized due to its robust support for AR development, cross-platform capabilities, and integration with advanced navigation and rendering systems.

The final concept was thus shaped into an AR-based indoor navigation system that overlays arrows directly onto the real-world environment, providing a natural and user-friendly navigation experience.

3.4 PROBLEM SOLUTION FIT

The proposed AR Indoor Navigation System effectively addresses the limitations identified in existing systems. By eliminating the need for external infrastructure and relying on vision-based localization, the system significantly reduces deployment cost and complexity.

Experimental testing demonstrates that the system provides accurate localization, stable navigation guidance, and smooth performance. The AR-based visualization ensures that users can follow navigation instructions intuitively without interpreting complex maps. The system thus achieves a strong alignment between the identified problem and the proposed solution.

Hence, the developed solution is well-suited for real-world deployment in environments such as educational institutions, hospitals, office buildings, and shopping malls.

CHAPTER 4
SYSTEM DESIGN AND ARCHITECTURE
4.1 OVERALL SYSTEM ARCHITECTURE

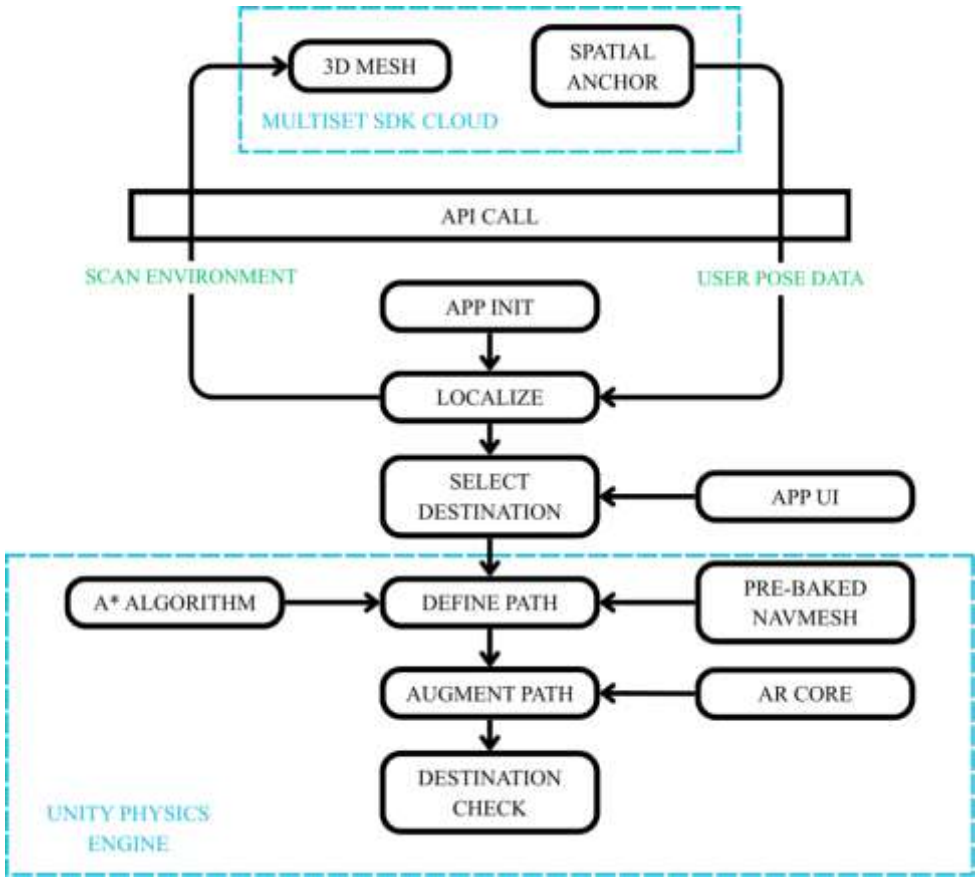


Fig 4.1 : System Architecture

The AR Indoor Navigation System is designed using a modular and layered architecture that integrates ARCore, Multiset SDK, Unity Engine, and Android platform services to provide an accurate and real-time indoor navigation solution.

The architecture consists of three major functional stages:

1. Application Initialization and AR Setup
2. Localization using Multiset Cloud and 3D Spatial Map
3. Navigation using NavMesh and AR Visualization

The system follows a pipeline-based processing model where each stage depends on the successful completion of the previous stage. The complete navigation process is initiated only after accurate localization is achieved.

At a high level, the system integrates the following major components:

- AR Foundation and ARCore for camera access, tracking, and AR rendering
- Multiset SDK and Multiset Cloud for spatial mapping and localization
- Unity Engine for scene management, physics, UI, and rendering
- NavMesh and A* pathfinding for route computation

- AR rendering pipeline for arrow-based navigation guidance

4.2 APPLICATION STARTUP AND INITIALIZATION MODULE

When the application is launched, the AR Foundation framework initializes the ARCore subsystem and prepares the camera, motion tracking, and environment understanding services. Simultaneously, the Unity runtime initializes the scene, user interface components, and core system managers.

During this phase:

- The AR session is created using the AR Foundation.
- ARCore starts tracking device motion and camera pose.
- The Multiset SDK is initialized and prepares the connection to the Multiset Cloud.
- The application loads the basic UI and displays instructions for scanning the environment.

This stage ensures that all required subsystems are active before entering the localization phase.

4.3 LOCALIZATION ARCHITECTURE AND WORKFLOW

Localization is the most critical part of the system and is entirely based on vision-based spatial matching using Multiset SDK and cloud-stored 3D mesh data.

4.3.1 Localization Workflow

The localization process follows the sequence shown in the workflow diagram:

1. **Camera Scan:** The user scans the surrounding environment using the device camera.
2. **Multiset SDK Processing:** The live camera frames are passed to the Multiset SDK, which extracts visual features.
3. **Cloud Matching:** The extracted features are sent to the Multiset Cloud, where they are matched against the stored 3D spatial mesh of the building.
4. **3D Mesh Matching:** If sufficient matches are found, the system identifies the corresponding location in the stored map.
5. **User Pose Estimation:** The precise position and orientation (pose) of the user in the mapped environment are returned to the application.

Once this process succeeds, the system transitions from the Localization Phase to the Navigation Phase.

4.4 SPATIAL MAP AND CLOUD ANCHOR ARCHITECTURE

The indoor environment is represented digitally using two major components:

- 3D Mesh Map of the building
- Spatial Anchors stored in the Multiset Cloud

The 3D mesh is generated during the offline scanning phase using the Multiset application. This mesh captures the structural geometry and visual features of the corridor. The spatial anchors act as persistent reference points that allow the system to perform consistent localization across different sessions and devices.

During runtime:

- The application queries the Multiset Cloud using an API call.
- The cloud returns the relevant spatial map and anchor information.
- The Multiset SDK aligns the Unity world coordinate system with the real-world environment.

This ensures that all virtual content (arrows, UI indicators, destination markers) is perfectly aligned with the physical environment.

4.5 NAVIGATION SYSTEM ARCHITECTURE

Once localization is successful, the system activates the navigation pipeline.

4.5.1 Destination Selection Module

The user selects the desired destination from the application UI. Each destination corresponds to a Point of Interest (POI) that is predefined in the Unity scene and associated with a specific location on the 3D map.

POI selection is handled using:

- Unity UI Canvas and TextMeshPro
- Collider-based selection logic

4.5.2 Path Planning Module (NavMesh + A* Algorithm)

Navigation path computation is performed using:

- Unity NavMesh system for walkable area representation
- A* (A-star) pathfinding algorithm for shortest path computation

The NavMesh is pre-baked over the imported 3D mesh of the building. When the user selects a destination:

1. The system takes the user's current localized position as the start node.
2. The selected POI position is taken as the goal node.
3. The NavMesh system computes the optimal path using the A* algorithm.
4. The resulting path is converted into a series of world-space waypoints.

4.6 AUGMENTED REALITY PATH VISUALIZATION MODULE

After the navigation path is computed, the system enters the AR visualization stage.

- A sequence of 3D arrow prefabs is instantiated along the computed path.
- These arrows are rendered using AR Foundation and ARCore.
- The arrows are spatially anchored to the real-world floor and corridor.
- As the user moves, the arrow positions and orientations remain stable and correctly aligned.

This provides an intuitive, real-world-aligned navigation experience where the user simply follows the arrows.

4.7 REAL-TIME DISTANCE TRACKING AND DESTINATION CHECK MODULE

The system continuously monitors:

- The user's current position
- The remaining path length

The distance to the destination is computed in real time and displayed on the UI.

A destination check module continuously checks if there is a collision found between goal POI and User, If found :

- The system concludes that the destination has been reached.
- A destination reached message is displayed to the user.
- The navigation session is terminated.

4.8 MODULES AND ALGORITHMS USED

The system is composed of the following major modules and techniques

4.8.1 User Interface Module

- Implemented using Unity Canvas and TextMeshPro
- Provides destination selection menu and distance display

4.8.2 Multiset SDK Module

- Used for environment mapping and cloud-based localization
- Handles spatial anchoring and pose estimation

4.8.3 3D Mesh and Cloud Anchor Module

- Stores the spatial representation of the building
- Provides persistent reference for localization

4.8.4 Environment Recognition Module

- Recognizes surroundings using API-based cloud matching
- Matches real-world scan with stored spatial data

4.8.5 Localization Module

- Computes user position using spatial anchor matching

4.8.6 Shortest Path Computation Module

- Uses Unity NavMesh and A* algorithm for path planning

4.8.7 AR Path Visualization Module

- Uses AR Foundation and ARCore to render arrows in real world

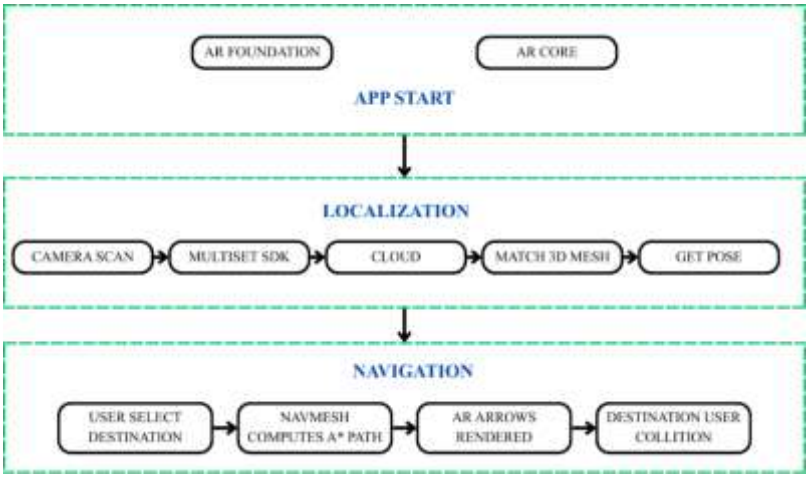


Fig 4.2 : Work Flow

CHAPTER 5
IMPLEMENTATION AND TECHNIQUES

5.1 APPLICATION INITIALIZATION AND AR FOUNDATION SETUP MODULE

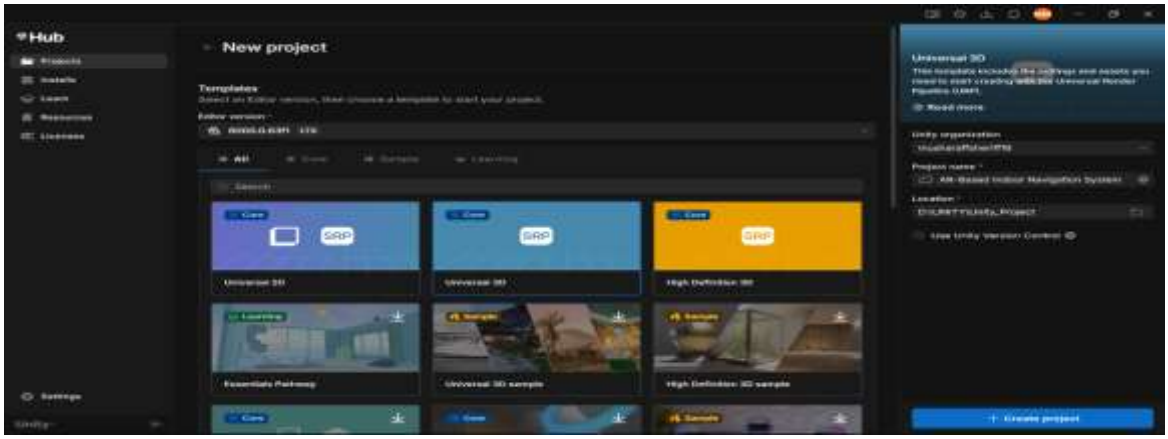


Fig 5.1 : Project Setup

The implementation of the AR Indoor Navigation System begins with the initialization of the augmented reality subsystem using Unity AR Foundation and ARCore. This module is responsible for preparing the device camera, motion tracking, and world tracking services required for AR operation.

When the application is launched, the Unity runtime initializes the AR session, activates the AR camera, and starts the ARCore tracking pipeline. At the same time, the system initializes all core managers including the Multiset SDK manager, UI manager, and navigation manager. The application then displays instructions prompting the user to scan the environment for localization.

This module ensures that the system is fully prepared to perform real-time environment understanding and AR rendering before proceeding to the localization phase.

5.2 ENVIRONMENT SCANNING AND MULTISET SDK INTEGRATION MODULE

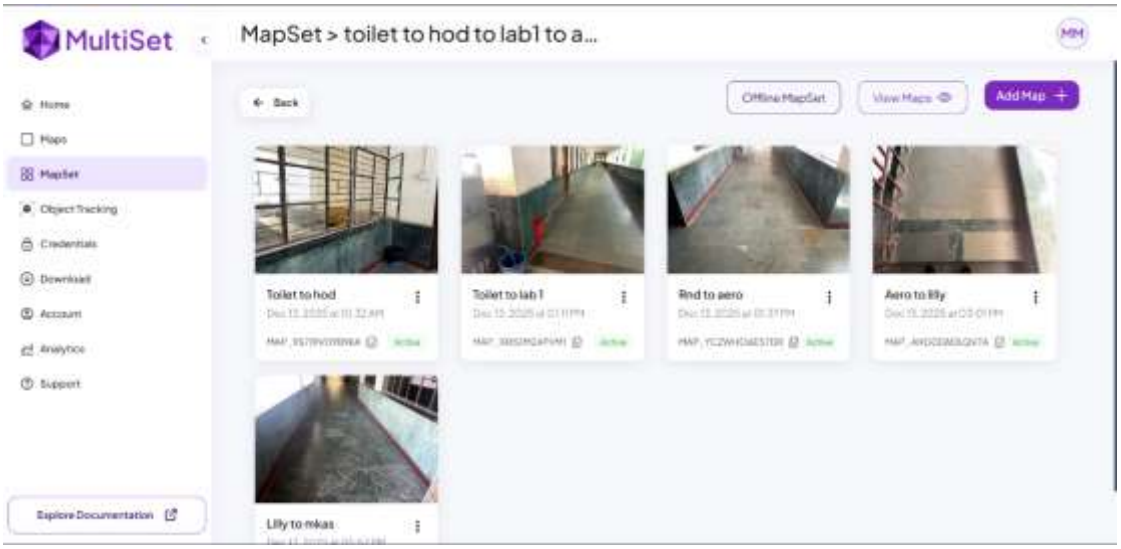


Fig 5.2 : Indoor 3D Mapping

This module implements the interface between the live camera feed and the Multiset SDK. Once the AR session is active, the camera continuously captures frames of the surrounding environment. These frames are forwarded to the Multiset SDK, which extracts visual features and prepares them for cloud-based matching.

The Multiset SDK is configured with the unique API key corresponding to the pre-scanned environment. The SDK establishes communication with the Multiset Cloud and manages data exchange required for localization. This module forms the foundation for environment recognition and pose estimation.

5.3 3D MESH AND CLOUD SPATIAL ANCHOR MANAGEMENT MODULE

The indoor environment is represented digitally using a 3D mesh map and a set of cloud-based spatial anchors generated during the offline scanning phase. This module handles the retrieval and management of this spatial data at runtime.

When the Multiset SDK requests localization, the cloud service compares the live scan data with the stored 3D mesh and spatial anchors. Upon successful matching, the cloud returns the appropriate anchor reference and transformation data. The Unity world coordinate system is then aligned with the real-world coordinate system using this information.

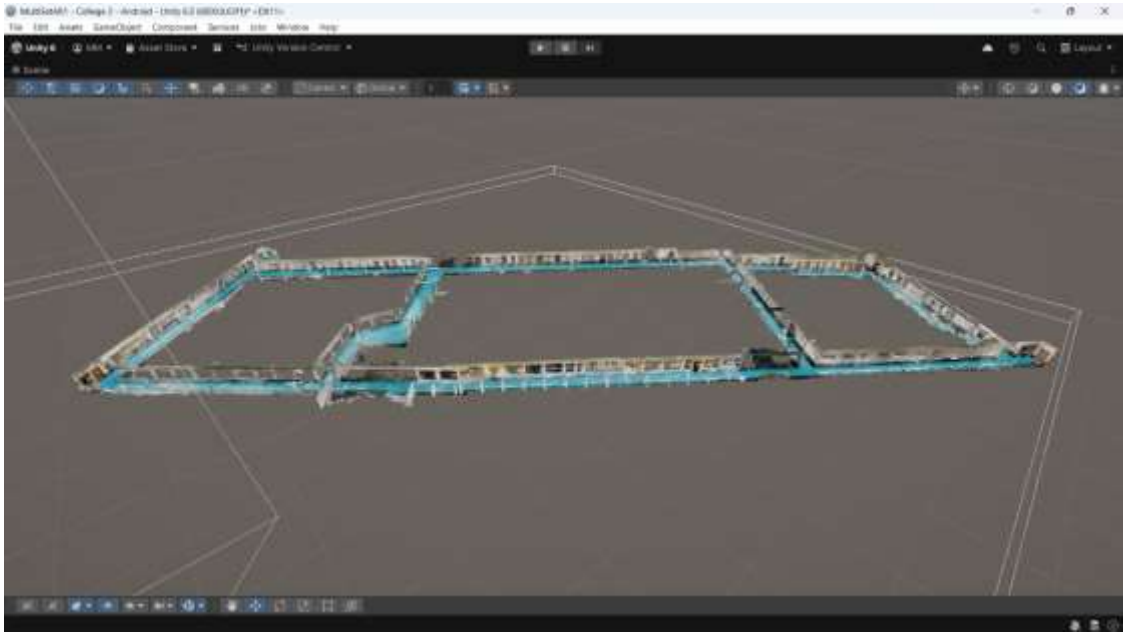


Fig 5.3 : Integration of 3D Mesh

This alignment is critical, as it ensures that all virtual content, including navigation arrows and UI indicators, is accurately registered in the physical environment.

5.4 ENVIRONMENT RECOGNITION AND LOCALIZATION MODULE

Fig 5.4 : Application Localization

This module implements the core localization pipeline of the system. The localization process is entirely vision-based and operates without any external infrastructure.

The workflow is as follows:

- The user scans the environment using the device camera.
- The Multiset SDK extracts visual features from the live camera frames.
- These features are sent to the Multiset Cloud.

- The cloud performs matching against the stored 3D mesh.
- Once a match is found, the system receives the user pose (position and orientation).

The localization process runs continuously to maintain accurate tracking as the user moves. Only after successful localization does the system enable the navigation functions.

5.5 USER INTERFACE AND POINT OF INTEREST (POI) SELECTION MODULE

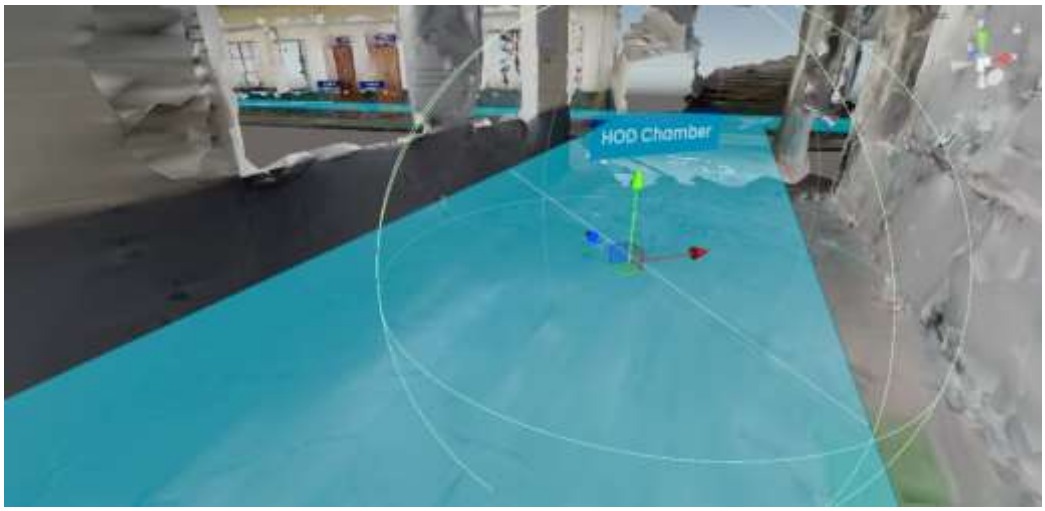
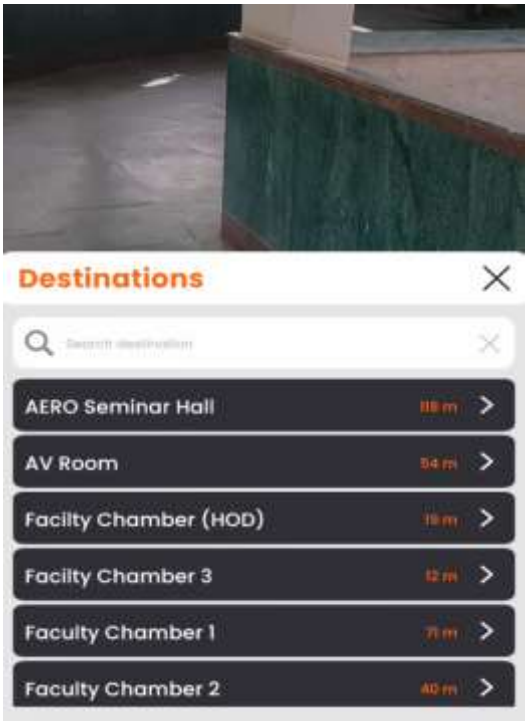


Fig 5.5 : Point Of Interest

The user interface is implemented using Unity Canvas and TextMeshPro to ensure clarity and readability in an AR environment. This module provides:

- A destination selection menu
- A real-time distance display
- A destination reached notification

Each destination in the building is represented as a Point of Interest (POI). POIs are implemented as empty game objects placed at appropriate positions in the 3D map and associated with colliders and identifiers.



5.6 : UI – POI Selection

When the user selects a destination from the UI, the system retrieves the corresponding POI location and forwards it to the navigation module.

5.6 NAVIGATION MESH GENERATION AND PATH PLANNING MODULE (NAVMESH + A* ALGORITHM)

This module is responsible for computing the optimal navigation path between the user's current position and the selected destination.

The 3D mesh of the building is first processed in the Unity editor to generate a pre-baked NavMesh over all walkable surfaces such as corridors and open areas. Obstacles such as walls and pillars are automatically excluded.

At runtime:

- The user's localized position is used as the start point.
- The selected POI is used as the destination point.
- Unity's NavMesh system computes the shortest path using the A* (A-star) algorithm.
- The resulting path is represented as a sequence of waypoints.

5.7 AUGMENTED REALITY PATH VISUALIZATION MODULE (AR ARROWS RENDERING)

Fig 5.7 : AR Rendering

This module is responsible for presenting the computed navigation path to the user in an intuitive and immersive manner using augmented reality.

- A 3D arrow prefab is instantiated repeatedly along the computed path waypoints.
- The arrows are rendered using AR Foundation and ARCore.

- The arrows are spatially anchored to the real-world environment using the localization reference frame.

As the user moves, the AR arrows remain stable in the environment and continuously indicate the correct direction of movement.

5.8 REAL-TIME DISTANCE COMPUTATION AND DESTINATION CHECK MODULE

This module continuously monitors the user's current position and computes the remaining distance to the destination using the NavMesh path length.

The distance value is updated in real time and displayed in the user interface. A destination check routine compares the user and target POI collision. When the collision occurs:

- The system concludes that the destination has been reached.
- A destination reached message is displayed.
- The navigation session is terminated.

5.9 UNITY PHYSICS AND RUNTIME MANAGEMENT MODULE

Unity's physics engine is used to manage spatial consistency, collision boundaries, and movement constraints. Although the user physically walks in the real world, the virtual representation of the user and AR content is governed by Unity's coordinate system and physics rules.

This module ensures:

- Stable AR object placement
- Correct spatial alignment
- Smooth runtime behavior

5.10 PERFORMANCE TESTING AND OPTIMIZATION USING UNITY PROFILER

Fig 5.8 : CPU Benchmark

Performance evaluation and optimization were performed using the Unity Profiler on the target device (Realme 5 Pro). The profiler was used to monitor:

- CPU usage
- Memory consumption
- Frame rate
- Rendering time

Based on the profiling results, several optimizations were applied, including:

- Optimization of 3D mesh complexity
- Reduction of unnecessary draw calls
- Efficient use of shaders
- Memory usage control

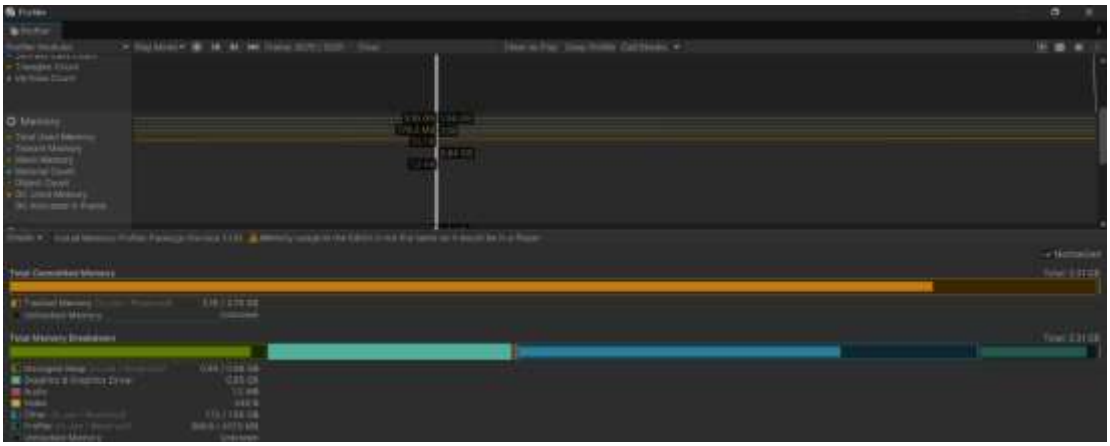


Fig 5.9 : RAM Benchmark

After optimization, the application achieved stable performance with:

- 60 FPS
- 0.28 ms rendering time
- 3.19 GB RAM usage out of 3.39 GB requested

5.11 APPLICATION BUILD AND DEPLOYMENT MODULE

After successful implementation and testing, the application was built as an Android APK using Unity's build system. The APK was installed on the Realme 5 Pro device and tested extensively in the real environment.

CHAPTER 6
RESULTS

Fig 6.1 : App Initialization and Localization

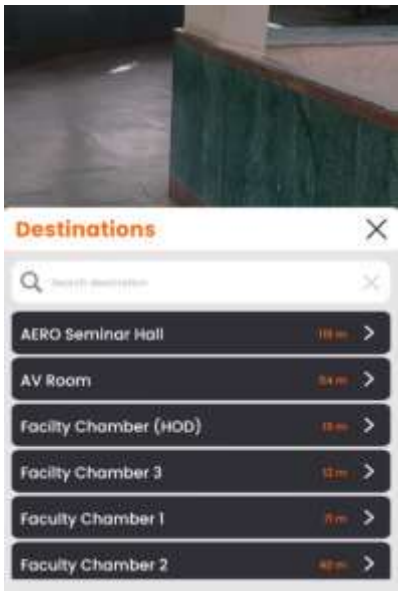


Fig 6.2 : User Destination Selection

Fig 6.3 : Real-Time Distance Calculation and Rendering Arrow

CHAPTER 7
CONCLUSION

In this project, an AR Indoor Navigation System was successfully designed, implemented, and evaluated

using Unity, AR Foundation, ARCore, and the Multiset SDK. The system effectively addresses the limitations of traditional indoor navigation methods by providing an accurate, intuitive, and infrastructure-free navigation solution based on vision-based localization and augmented reality guidance.

The proposed system utilizes a three-dimensional spatial map of the indoor environment and cloud-based spatial anchors to perform precise real-time localization. After successful localization, the system computes the shortest navigation path using Unity NavMesh with the A* pathfinding algorithm and presents the navigation guidance to the user through augmented reality arrows and real-time distance indicators. Performance evaluation using the Unity Profiler confirmed that the system operates smoothly on standard Android devices, maintaining a stable 60 FPS frame rate, low rendering time, and efficient memory utilization.

As a practical deployment and proof of concept, the developed application was successfully implemented and tested in the Computer Science and Engineering department building of Adhiyamaan College of Engineering. A complete indoor map of the department corridor was created using the Multiset scanning process, and all important rooms and locations were configured as Points of Interest. The application was able to localize users inside the department and guide them accurately to the selected destinations, thereby demonstrating the real-world applicability and usefulness of the proposed system in an academic environment.

Thus, the project not only fulfills its technical objectives but also results in a fully functional real-world indoor navigation application for the college CSE department, proving the feasibility, reliability, and practical value of the proposed approach.

CHAPTER 8
SOURCE CODE
ShowNavMesh.cs

```
using UnityEngine;
using UnityEngine.AI;
using UnityEngine.Rendering;

public class ShowNavMesh : MonoBehaviour
{
public static ShowNavMesh instance;
[Tooltip("Color of the NavMesh surface visualization")]
public Color surfaceColor = new Color(0, 1, 0, 0.3f); // Semi-transparent green
[Tooltip("Height offset for the visualization mesh")]
public float heightOffset = 0.01f;
private MeshFilter meshFilter;
private MeshRenderer meshRenderer;
[Tooltip("Enable to see NavMesh")]
public bool IsNavMeshShown = true;
void Awake()
{
instance = this;
```

```
}
void Start()
{
meshFilter = GetComponent<MeshFilter>();
if (meshFilter == null)
meshFilter = gameObject.AddComponent<MeshFilter>();
meshRenderer = GetComponent<MeshRenderer>();
if (meshRenderer == null)
meshRenderer = gameObject.AddComponent<MeshRenderer>();

GenerateNavMeshVisualization();
}
void Update()
{
SetNavMeshVisibility(IsNavMeshShown);
}
public void SetNavMeshVisibility(bool isShown)
{
meshRenderer.enabled = isShown;
}
public void GenerateNavMeshVisualization()
{
SetNavMeshVisibility(IsNavMeshShown);
Mesh navMeshMesh = new Mesh();
navMeshMesh.name = "NavMesh Surface Visualization";
NavMeshTriangulation triangulation = NavMesh.CalculateTriangulation();
Vector3[] vertices = new Vector3[triangulation.vertices.Length];
for (int i = 0; i < vertices.Length; i++)
{
vertices[i] = triangulation.vertices[i] + Vector3.up * heightOffset;
}
navMeshMesh.vertices = vertices;
navMeshMesh.triangles = triangulation.indices;
navMeshMesh.RecalculateNormals();
navMeshMesh.RecalculateBounds();
meshFilter.mesh = navMeshMesh;
Material surfaceMaterial = CreateNavMeshMaterial();
meshRenderer.material = surfaceMaterial;
}
private Material CreateNavMeshMaterial()
{
Material surfaceMaterial;
RenderPipelineAsset currentRP = GraphicsSettings.currentRenderPipeline;
```

```
if (currentRP != null && currentRP.GetType().Name.Contains("UniversalRenderPipelineAsset"))
{
Shader urpShader = Shader.Find("Universal Render Pipeline/Unlit");
if (urpShader == null)
{
urpShader = Shader.Find("Sprites/Default");
}
surfaceMaterial = new Material(urpShader);
surfaceMaterial.color = surfaceColor;

if (urpShader.name.Contains("Unlit"))
{
surfaceMaterial.SetFloat("_Surface", 1); // Transparent
surfaceMaterial.SetFloat("_Blend", 0); // Alpha
surfaceMaterial.SetFloat("_SrcBlend", (float)BlendMode.SrcAlpha);
surfaceMaterial.SetFloat("_DstBlend", (float)BlendMode.OneMinusSrcAlpha);
surfaceMaterial.SetFloat("_ZWrite", 0);
surfaceMaterial.EnableKeyword("_SURFACE_TYPE_TRANSPARENT");
surfaceMaterial.EnableKeyword("_ALPHABLEND_ON");
}
}
else
{
surfaceMaterial = new Material(Shader.Find("Standard"));
surfaceMaterial.color = surfaceColor;
surfaceMaterial.SetFloat("_Mode", 2);
surfaceMaterial.SetInt("_SrcBlend", (int)BlendMode.SrcAlpha);
surfaceMaterial.SetInt("_DstBlend", (int)BlendMode.OneMinusSrcAlpha);
surfaceMaterial.SetInt("_ZWrite", 0);
surfaceMaterial.DisableKeyword("_ALPHATEST_ON");
surfaceMaterial.EnableKeyword("_ALPHABLEND_ON");
surfaceMaterial.DisableKeyword("_ALPHAPREMULTIPLY_ON");
}

surfaceMaterial.renderQueue = 3000;
return surfaceMaterial;
}
}
```

AugmentedSpace.cs

```
using UnityEngine;
public class AugmentedSpace : MonoBehaviour
```

```
{
[Tooltip("Title of the space")]
public string title;
POI[] pois = { };
[Tooltip("Parent of the POIs inside this space")]
public GameObject augmentation;
void Awake()
{
pois = augmentation.GetComponentsInChildren<POI>(true);
}
public POI[] GetPOIs()
{
return pois;
}
}
```

ShowPath.cs

```
using System.Collections;
using MultiSet;
using UnityEngine;
using UnityEngine.AI;

public class ShowPath : MonoBehaviour
{
public static ShowPath instance;
LineRenderer line;
NavMeshPath path;
float _elapsed = 0.0f;
[Tooltip("Path update frequency in seconds")]
public float pathUpdateFrequency = 0.5f; // Update twice per second
[Tooltip("Height of the path above NavMesh")]
public float pathHeightAboveGround = 0.1f; // in meters
Transform a = null;
Transform b = null;
[Tooltip("Prefab to visualize path corners")]
public GameObject cornerVisualizationPrefab;
GameObject[] visibleCorners = { };
[Tooltip("Toggles visibility of path corners")]
public bool isCornersVisible;
bool cornerVisibilityHasChanged = false;
bool forcePathRecalculation = false;
```

```
void Awake()
{
instance = this;
line = GetComponent<LineRenderer>();
line.alignment = LineAlignment.TransformZ;
line.transform.forward = Vector3.up; // Forces it to stay flat and not roll
}

void Start()
{
path = new NavMeshPath();
line.enabled = false;
isCornersVisible = false;
}

void Update()
{
if (a != null && b != null)
{
line.enabled = true;
_elapsed += Time.deltaTime;
if (_elapsed > pathUpdateFrequency || forcePathRecalculation)
{
StartCoroutine(DrawPath(path));
PathEstimationUtils.instance.UpdateEstimation(path.corners);
_elapsed = 0.0f;
forcePathRecalculation = false;
NavMesh.CalculatePath(a.position, b.position, NavMesh.AllAreas, path);
}
}
else
{
line.enabled = false;
SetCornerVisibility(false);
}
if (a != null && b != null && NavigationController.instance.IsCurrentlyNavigating())
{
if (path.status == NavMeshPathStatus.PathPartial || path.status == NavMeshPathStatus.PathInvalid)
{
// handle unreachable route
ToastManager.Instance.ShowAlert("Problem calculating route");
NavigationController.instance.StopNavigation();
}
```

```
}
}

IEnumerator DrawPath(NavMeshPath path)
{
yield return new WaitForEndOfFrame();
line.positionCount = path.corners.Length;
if (isCornersVisible)
{
cornerVisibilityHasChanged = true;
if (cornerVisibilityHasChanged)
{
SetCornerVisibility(true);
}
HandlePathCornerVisualization();
}
else
{
cornerVisibilityHasChanged = true;
if (cornerVisibilityHasChanged)
{
SetCornerVisibility(false);
}
}
for (var i = 0; i < path.corners.Length; i++)
{
Vector3 linePosition = new Vector3(path.corners[i].x, path.corners[i].y + pathHeightAboveGround, path.corners[i].z);
line.SetPosition(i, linePosition);
if (isCornersVisible)
{
UpdateVisibleCorner(i, linePosition);
}
}
}

public void ResetPath()
{
StopAllCoroutines();
a = null;
b = null;
line.positionCount = 1;
}
```

```
public void SetPositionFrom(Transform from)
{
a = from;
forcePathRecalculation = true;
}
public void SetPositionTo(Transform to)
{
b = to;
if (b != null)
{
forcePathRecalculation = true;
NavMesh.CalculatePath(a.position, b.position, NavMesh.AllAreas, path);
}
}
void HandlePathCornerVisualization()
{
int pathCornersCount = path.corners.Length;
if (pathCornersCount > visibleCorners.Length)
{
if (visibleCorners.Length == 0)
{
visibleCorners = new GameObject[pathCornersCount];
}
else
{
GameObject[] newVisibleCorners = new GameObject[pathCornersCount];
for (int i = 0; i < visibleCorners.Length; i++)
{
newVisibleCorners[i] = visibleCorners[i];
}
visibleCorners = newVisibleCorners;
}
}
else if (pathCornersCount < visibleCorners.Length)
{
int elementsToRemoveCount = visibleCorners.Length - pathCornersCount;
GameObject[] newVisibleCorners = new GameObject[visibleCorners.Length -
elementsToRemoveCount];

for (int i = 0; i < visibleCorners.Length; i++)
{
if (i < newVisibleCorners.Length)
{
```

```
newVisibleCorners[i] = visibleCorners[i];
}
else
{
Destroy(visibleCorners[i]);
}
}
visibleCorners = newVisibleCorners;
}
}
void UpdateVisibleCorner(int i, Vector3 newPosition)
{
if (visibleCorners[i] == null)
{
GameObject newCorner = GameObject.Instantiate(cornerVisualizationPrefab, newPosition, Quaternion.identity);
visibleCorners[i] = newCorner;
}
else
{
visibleCorners[i].gameObject.transform.position = newPosition;
}
}
void SetCornerVisibility(bool show)
{
cornerVisibilityHasChanged = false;
foreach (var corner in visibleCorners)
{
if (corner != null)
{
corner.gameObject.SetActive(show);
}
}
}
}
```

NavigationController.cs

```
using UnityEngine;
using UnityEngine.AI;
using UnityEngine.Events;
public class NavigationController : MonoBehaviour
{
```

```
public static NavigationController instance;
private Camera ARCamera;
private SphereCollider ARCameraCollider;
[Tooltip("NavMesh agent child of ARCamera")]
public NavMeshAgent agent;
[Tooltip("Current POI for navigation")]
public POI currentDestination;
[Tooltip("Space that contains POIs")]
public AugmentedSpace augmentedSpace;
private Vector3 lastAgentPosition;
[Tooltip("Minimum distance the agent needs to move before updating path")]
public float positionUpdateThreshold = 0.5f; // in meters
public UnityEvent DestinationArrived = new UnityEvent();
void Awake()
{
instance = this;
ARCamera = Camera.main;
}
void Start()
{
ARCameraCollider = ARCamera.GetComponent<SphereCollider>();
if (currentDestination)
{
StartNavigation();
}
lastAgentPosition = agent.transform.position;
}
void Update()
{
if (agent.isOnNavMesh)
{
agent.isStopped = true;
}
if (IsCurrentlyNavigating() && agent.isOnNavMesh)
{
agent.destination = currentDestination.poiCollider.transform.position;

float distanceMoved = Vector3.Distance(agent.transform.position, lastAgentPosition);
if (distanceMoved > positionUpdateThreshold)
{
lastAgentPosition = agent.transform.position;
ShowPath.instance.SetPositionFrom(agent.transform);
}
```

```
ARCameraCollider.enabled = true;
}
else
{
ARCameraCollider.enabled = false;
}
}
public void SetPOIForNavigation(POI aPOI)
{
currentDestination = aPOI;
StartNavigation();
}
void StartNavigation()
{
lastAgentPosition = agent.transform.position;
ShowPath.instance.SetPositionFrom(agent.transform);
ShowPath.instance.SetPositionTo(currentDestination.poiCollider.transform);
}
public void StopNavigation()
{
if (currentDestination != null)
{
currentDestination = null;
ShowPath.instance.ResetPath();
PathEstimationUtils.instance.ResetEstimation();
}
}
public void ArrivedAtDestination()
{
DestinationArrived.Invoke();
StopNavigation();
NavigationUIController.instance.ShowArrivedState();
}
public bool IsCurrentlyNavigating()
{
return currentDestination != null;
}
public void ToggleAgentVisibility()
{
agent.gameObject.GetComponent<MeshRenderer>().enabled =
!agent.gameObject.GetComponent<MeshRenderer>().enabled;
}
}
```

NavigationUIController.cs

```
using UnityEngine;
using TMPro;
using MultiSet;
public class NavigationUIController : MonoBehaviour
{
public static NavigationUIController instance;
[Tooltip("Label to show remaining distance")]
public TextMeshProUGUI remainingDistance;
[Tooltip("Button to stop navigation")]
public GameObject stopButton;
[Tooltip("SelectList where POIs are shown")]
public SelectList poiList;
[Tooltip("Parent GameObject of POIs selection UI")]
public GameObject DestinationSelectUI;
[Tooltip("Label to show name of current destination")]
public TextMeshProUGUI destinationName;
[Tooltip("Parent GameObject of navigation progress slider")]
public GameObject navigationProgressSlider;
[Tooltip("Navigation Path Material")]
public Material material;
void Awake()
{
instance = this;
}
void Start()
{
ShowNavigationUIElements(false);
DestinationSelectUI.SetActive(false);
destinationName.text = "";
}
void Update()
{
HandleNavigationState();
UpdateRemainingDistance();
}
void HandleNavigationState()
{
if (NavigationController.instance.IsCurrentlyNavigating())
{
destinationName.text = NavigationController.instance.currentDestination.poiName;
```

```
return;
}
destinationName.text = "";
}
public void ToggleDestinationSelectUI()
{
DestinationSelectUI.SetActive(!DestinationSelectUI.activeSelf);
if (!DestinationSelectUI.activeSelf)
{
poiList.ResetPOISearch();
return;
}
poiList.RenderPOIs();
}
public void ResetPoiSearch()
{
poiList.ResetPOISearch();
}
public void RenderPoiCall()
{
poiList.RenderPOIs();
}
public void ClickedStartNavigation(POI poi)
{
NavigationController.instance.SetPOIForNavigation(poi);
ToggleDestinationSelectUI();
ShowNavigationUIElements(true);
}
public void ClickedStopButton()
{
ShowNavigationUIElements(false);
NavigationController.instance.StopNavigation();
}
void ShowNavigationUIElements(bool isVisible)
{
navigationProgressSlider.SetActive(isVisible);
stopButton.SetActive(isVisible);
}
void UpdateRemainingDistance()
{
if (!NavigationController.instance.IsCurrentlyNavigating())
{
remainingDistance.SetText("");
```

```
return;
}
int distance = PathEstimationUtils.instance.getRemainingDistanceMeters();
string distanceText = distance + "";
if (distance > 1)
{
if (material != null)
material.SetFloat("_PathLength", distance);
}
if (distance <= 1)
{
distanceText += " m remaining";
}
else
{
distanceText += " m remaining";
}
remainingDistance.text = distanceText;
}
public void ShowArrivedState()
{
ShowNavigationUIElements(false);
ToastManager.Instance.ShowAlert("You arrived at the destination!");
}
}
```

XROrigin.cs

```
#if ENABLE_VR || UNITY_GAMECORE
#define XR_MODULE_AVAILABLE
#endif
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Assertions;
using UnityEngine.Serialization;
using UnityEngine.XR;
namespace Unity.XR.CoreUtils
{
[AddComponentMenu("XR/XR Origin")]
[DisallowMultipleComponent]
public class XROrigin : MonoBehaviour
```

```
{
[SerializeField]
[Tooltip("The Camera to associate with the XR device.")]
Camera m_Camera;
public Camera Camera
{
get => m_Camera;
set => m_Camera = value;
}
public event Action<ARTrackablesParentTransformChangedEventArgs>
TrackablesParentTransformChanged;
public enum TrackingOriginMode
{
[SerializeField, FormerlySerializedAs("m_RigBaseGameObject")]
GameObject m_OriginBaseGameObject;
public GameObject Origin
{
get => m_OriginBaseGameObject;
set => m_OriginBaseGameObject = value;
}

[SerializeField]
GameObject m_CameraFloorOffsetObject;
public GameObject CameraFloorOffsetObject
{
get => m_CameraFloorOffsetObject;
set
{
m_CameraFloorOffsetObject = value;
MoveOffsetHeight();
}
}

[SerializeField]
TrackingOriginMode m_RequestedTrackingOriginMode = TrackingOriginMode.NotSpecified;
public TrackingOriginMode RequestedTrackingOriginMode
{
get => m_RequestedTrackingOriginMode;
set
{
m_RequestedTrackingOriginMode = value;
TryInitializeCamera();
}
```

```
}
[SerializeField]
float m_CameraYOffset = k_DefaultCameraYOffset;
public float CameraYOffset
{
get => m_CameraYOffset;
set
{
m_CameraYOffset = value;
MoveOffsetHeight();
}
}
#if XR_MODULE_AVAILABLE || PACKAGE_DOCS_GENERATION
public TrackingOriginModeFlags CurrentTrackingOriginMode { get; private set; }
#endif
public Vector3 OriginInCameraSpacePos =>
m_Camera.transform.InverseTransformPoint(m_OriginBaseGameObject.transform.position);
public Vector3 CameraInOriginSpacePos =>
m_OriginBaseGameObject.transform.InverseTransformPoint(m_Camera.transform.position);
public float CameraInOriginSpaceHeight => CameraInOriginSpacePos.y;
#if XR_MODULE_AVAILABLE
static readonly List<XRInputSubsystem> s_InputSubsystems = new List<XRInputSubsystem>();
#endif
bool m_CameraInitialized;
bool m_CameraInitializing;
void MoveOffsetHeight()
{
#if XR_MODULE_AVAILABLE
if (!Application.isPlaying)
return;
switch (CurrentTrackingOriginMode)
{
case TrackingOriginModeFlags.Floor:
MoveOffsetHeight(0f);
break;
case TrackingOriginModeFlags.Device:
case TrackingOriginModeFlags.Unbounded:
MoveOffsetHeight(m_CameraYOffset);
break;
default:
return;
}
#endif
```

```
}
void MoveOffsetHeight(float y)
{
if (m_CameraFloorOffsetObject != null)
{
var offsetTransform = m_CameraFloorOffsetObject.transform;
var desiredPosition = offsetTransform.localPosition;
desiredPosition.y = y;
offsetTransform.localPosition = desiredPosition;
}
}
void TryInitializeCamera()
{
if (!Application.isPlaying)
return;
m_CameraInitialized = SetupCamera();
if (!m_CameraInitialized & !m_CameraInitializing)
StartCoroutine(RepeatInitializeCamera());
}
bool SetupCamera()
{
var initialized = true;
#if XR_MODULE_AVAILABLE
#if UNITY_2023_2_OR_NEWER
SubsystemManager.GetSubsystems(s_InputSubsystems);
#else
SubsystemManager.GetInstances(s_InputSubsystems);
#endif
if (s_InputSubsystems.Count > 0)
{
foreach (var inputSubsystem in s_InputSubsystems)
{
if (SetupCamera(inputSubsystem))
{                    inputSubsystem.trackingOriginUpdated -= OnInputSubsystemTrackingOriginUpdated;
inputSubsystem.trackingOriginUpdated += OnInputSubsystemTrackingOriginUpdated;
}
else
{
initialized = false;
}
}
}
#endif
```

```
return initialized;
}
#if XR_MODULE_AVAILABLE
bool SetupCamera(XRInputSubsystem inputSubsystem)
{
if (inputSubsystem == null)
return false;
var successful = true;
switch (m_RequestedTrackingOriginMode)
{
case TrackingOriginMode.NotSpecified:
CurrentTrackingOriginMode = inputSubsystem.GetTrackingOriginMode();
break;
case TrackingOriginMode.Device:
case TrackingOriginMode.Floor:
case TrackingOriginMode.Unbounded:
{
var supportedModes = inputSubsystem.GetSupportedTrackingOriginModes();
if (supportedModes == TrackingOriginModeFlags.Unknown)
return false;
var equivalentFlagsMode = ConvertTrackingOriginModeToFlag(m_RequestedTrackingOriginMode);
if ((supportedModes & equivalentFlagsMode) == 0)
{
m_RequestedTrackingOriginMode = TrackingOriginMode.NotSpecified;
CurrentTrackingOriginMode = inputSubsystem.GetTrackingOriginMode();
Debug.LogWarning($"Attempting to set the tracking origin mode to {equivalentFlagsMode}, but that is
not supported by the SDK." +
$" Supported types: {supportedModes:F}. Using the current mode of {CurrentTrackingOriginMode}
instead.", this);
}
else
{
successful = inputSubsystem.TrySetTrackingOriginMode(equivalentFlagsMode);
}
}
break;
default:
Debug.LogError($"Unhandled
{nameof(TrackingOriginMode)}={m_RequestedTrackingOriginMode}");
return false;
}
if (successful)
MoveOffsetHeight();
```

```
if (CurrentTrackingOriginMode == TrackingOriginModeFlags.Device ||
m_RequestedTrackingOriginMode == TrackingOriginMode.Device
|| CurrentTrackingOriginMode == TrackingOriginModeFlags.Unbounded ||
m_RequestedTrackingOriginMode == TrackingOriginMode.Unbounded
)
successful = inputSubsystem.TryRecenter();
return successful;
}
void OnInputSubsystemTrackingOriginUpdated(XRInputSubsystem inputSubsystem)
{
CurrentTrackingOriginMode = inputSubsystem.GetTrackingOriginMode();
MoveOffsetHeight();
}
#endif
IEnumerator RepeatInitializeCamera()
{
m_CameraInitializing = true;
while (!m_CameraInitialized)
{
yield return null;
if (!m_CameraInitialized)
m_CameraInitialized = SetupCamera();
}
m_CameraInitializing = false;
}
public bool RotateAroundCameraUsingOriginUp(float angleDegrees)
{
return RotateAroundCameraPosition(m_OriginBaseGameObject.transform.up, angleDegrees);
}
public bool RotateAroundCameraPosition(Vector3 vector, float angleDegrees)
{
if (m_Camera == null || m_OriginBaseGameObject == null)
{
return false;
}
m_OriginBaseGameObject.transform.RotateAround(m_Camera.transform.position, vector,
angleDegrees);
return true;
}
public bool MatchOriginUp(Vector3 destinationUp)
{
if (m_OriginBaseGameObject == null)
{
```

```
return false;
}
if (m_OriginBaseGameObject.transform.up == destinationUp)
return true;
var rigUp = Quaternion.FromToRotation(m_OriginBaseGameObject.transform.up, destinationUp);
m_OriginBaseGameObject.transform.rotation = rigUp * transform.rotation;
return true;
}
public bool MatchOriginUpCameraForward(Vector3 destinationUp, Vector3 destinationForward)
{
if (m_Camera != null && MatchOriginUp(destinationUp))
{
var projectedCamForward = Vector3.ProjectOnPlane(m_Camera.transform.forward,
destinationUp).normalized;

var signedAngle = Vector3.SignedAngle(projectedCamForward, destinationForward, destinationUp);
RotateAroundCameraPosition(destinationUp, signedAngle);
return true;
}
return false;
}
public bool MatchOriginUpOriginForward(Vector3 destinationUp, Vector3 destinationForward)
{
if (m_OriginBaseGameObject != null && MatchOriginUp(destinationUp))
{
var signedAngle = Vector3.SignedAngle(m_OriginBaseGameObject.transform.forward,
destinationForward, destinationUp);
RotateAroundCameraPosition(destinationUp, signedAngle);
return true;
}
return false;
}
public bool MoveCameraToWorldLocation(Vector3 desiredWorldLocation)
{
if (m_Camera == null)
{
return false;
}

var rot = Matrix4x4.Rotate(m_Camera.transform.rotation);
var delta = rot.MultiplyPoint3x4(OriginInCameraSpacePos);
m_OriginBaseGameObject.transform.position = delta + desiredWorldLocation;
return true;
```

```
}
protected void Awake()
{
if (m_CameraFloorOffsetObject == null)
{
Debug.LogWarning("No Camera Floor Offset GameObject specified for XR Origin, using attached GameObject.", this);
m_CameraFloorOffsetObject = gameObject;
}
if (m_Camera == null)
{
var mainCamera = Camera.main;
if (mainCamera != null)
m_Camera = mainCamera;
else
Debug.LogWarning("No Main Camera is found for XR Origin, please assign the Camera field manually.", this);
}
TrackablesParent = new GameObject("Trackables").transform;
TrackablesParent.SetParent(transform, false);
TrackablesParent.SetLocalPose(Pose.identity);
TrackablesParent.localScale = Vector3.one;

if (m_Camera)
{
#if INCLUDE_INPUT_SYSTEM && INCLUDE_LEGACY_INPUT_HELPERS
var trackedPoseDriver = m_Camera.GetComponent<UnityEngine.InputSystem.XR.TrackedPoseDriver>();
var trackedPoseDriverOld = m_Camera.GetComponent<UnityEngine.SpatialTracking.TrackedPoseDriver>();
if (trackedPoseDriver == null && trackedPoseDriverOld == null)
{
Debug.LogWarning(
$"Camera \"{m_Camera.name}\" does not use a Tracked Pose Driver (Input System), " +
"so its transform will not be updated by an XR device.  In order for this to be " +
"updated, please add a Tracked Pose Driver (Input System) with bindings for position and rotation of the center eye.", this);
}
#elif !INCLUDE_INPUT_SYSTEM && INCLUDE_LEGACY_INPUT_HELPERS
var trackedPoseDriverOld = m_Camera.GetComponent<UnityEngine.SpatialTracking.TrackedPoseDriver>();
if (trackedPoseDriverOld == null)
{
```

```
Debug.LogWarning(
$"Camera \"{m_Camera.name}\" does not use a Tracked Pose Driver, and com.unity.xr.legacyinputhelpers is installed. " +
"Although the Tracked Pose Driver from Legacy Input Helpers can be used, it is recommended to " +
"install com.unity.inputsystem instead and add a Tracked Pose Driver (Input System) with bindings for position and rotation of the center eye.", this);
}
#elif INCLUDE_INPUT_SYSTEM && !INCLUDE_LEGACY_INPUT_HELPERS
var trackedPoseDriver = m_Camera.GetComponent<UnityEngine.InputSystem.XR.TrackedPoseDriver>();
if (trackedPoseDriver == null)
{
Debug.LogWarning(
$"Camera \"{m_Camera.name}\" does not use a Tracked Pose Driver (Input System), " +
"so its transform will not be updated by an XR device.  In order for this to be " +
"updated, please add a Tracked Pose Driver (Input System) with bindings for position and rotation of the center eye.", this);
}
#elif !INCLUDE_INPUT_SYSTEM && !INCLUDE_LEGACY_INPUT_HELPERS
Debug.LogWarning(
$"Camera \"{m_Camera.name}\" does not use a Tracked Pose Driver and com.unity.inputsystem is not installed, " +
"so its transform will not be updated by an XR device.  In order for this to be " +
"updated, please install com.unity.inputsystem and add a Tracked Pose Driver (Input System) with bindings for position and rotation of the center eye.", this);
#endif
}
}
Pose GetCameraOriginPose()
{
var localOriginPose = Pose.identity;
var parent = m_Camera.transform.parent;
return parent
? parent.TransformPose(localOriginPose)
: localOriginPose;
}
protected void OnEnable() => Application.onBeforeRender += OnBeforeRender;
protected void OnDisable() => Application.onBeforeRender -= OnBeforeRender;
void OnBeforeRender()
{
if (m_Camera)
{
var pose = GetCameraOriginPose();
```

```
TrackablesParent.position = pose.position;
TrackablesParent.rotation = pose.rotation;
}
if (TrackablesParent.hasChanged)
{
TrackablesParentTransformChanged?.Invoke(
new ARTrackablesParentTransformChangedEventArgs(this, TrackablesParent));
TrackablesParent.hasChanged = false;
}
}
protected void OnValidate()
{
if (m_OriginBaseGameObject == null)
m_OriginBaseGameObject = gameObject;
if (Application.isPlaying && isActiveAndEnabled)
{
if (IsModeStale())
TryInitializeCamera();
else
MoveOffsetHeight();
}
bool IsModeStale()
{
#if XR_MODULE_AVAILABLE
if (s_InputSubsystems.Count > 0)
{
foreach (var inputSubsystem in s_InputSubsystems)
{
TrackingOriginModeFlags equivalentFlagsMode =
ConvertTrackingOriginModeToFlag(m_RequestedTrackingOriginMode);
if (equivalentFlagsMode == TrackingOriginModeFlags.Unknown)
return false;
if (inputSubsystem != null && inputSubsystem.GetTrackingOriginMode() != equivalentFlagsMode)
{
return true;
}
}
}
#endif
return false;
}
}
static TrackingOriginModeFlags ConvertTrackingOriginModeToFlag(TrackingOriginMode mode)
```

```
{
switch (mode)
{
case TrackingOriginMode.NotSpecified:
return TrackingOriginModeFlags.Unknown;
case TrackingOriginMode.Device:
return TrackingOriginModeFlags.Device;
case TrackingOriginMode.Floor:
return TrackingOriginModeFlags.Floor;
case TrackingOriginMode.Unbounded:
return TrackingOriginModeFlags.Unbounded;
}
}
protected void Start()
{
TryInitializeCamera();
}
protected void OnDestroy()
{
#if XR_MODULE_AVAILABLE
foreach (var inputSubsystem in s_InputSubsystems)
{
if (inputSubsystem != null)
inputSubsystem.trackingOriginUpdated -= OnInputSubsystemTrackingOriginUpdated;
}
#endif
}
}
}
```

REFERENCES

1. M. Lu, M. Arikawa, K. Oba, K. Ishikawa, Y. Jin, T. Utsumi and R. Sato, “Indoor AR Navigation Framework Based on Geofencing and Image-Tracking with Accumulated Error Correction,” *Applied Sciences*, vol. 14, no. 10, May 2024.
2. I. K. D. S. et al., “Adaptive AR Navigation: Real-Time Mapping for Indoor Environment Using Node Placement and Marker Localization,” *Information*, vol. 16, no. 6, 2024.
3. R. Zhao, W. Yu and Z. Huang, “A Design of Indoor Navigation System Based on Augmented Reality and Computer Vision,” *IEEE Access*, 2024.
4. A. Alkady, Y. Rizk and D. M. Alsekait, “SINS_AR: An Efficient Smart Indoor Navigation System Based on Augmented Reality,” *IEEE Access*, 2024.
5. P. Hořejší, T. Macháč and M. Šimon, “Reliability and Accuracy of Indoor Warehouse Navigation Using Augmented Reality,” *IEEE Access*, 2024.
6. Z. Qiu, M. Ashour, X. Zhou and S. Kalantari, “NavMarkAR: A Landmark-based Augmented Reality

Wayfinding System for Enhancing Spatial Learning of Older Adults," *arXiv preprint*, Nov. 2023.
7. J.-R. Jiang and H. Subakti, "An Indoor Location-Based Augmented Reality Framework," *Sensors*, vol. 23, no. 3, Jan. 2023.
8. S. Reni Hena Helan, S. J. Vivekanandan, A. Deepak, S. Imran and T. Hemanth, "Indoor Navigation Using Augmented Reality," *International Journal of Engineering Research & Technology (IJERT)*, vol. 11, no. 03, Jun. 2023.
9. F. Serhan Daniş, A. T. Naskali, A. T. Cemgil and C. Ersoy, "An Indoor Localization Dataset and Data Collection Framework with High Precision Position Annotation," *arXiv preprint*, 2022.
10. A. Martin, J. Cheriyan, J. J. Ganesh, J. Sebastian and J. V. Jayakrishna, "Indoor Navigation using Augmented Reality," *EAI Endorsed Transactions on Creative Technologies*, vol. 8, no. 26, Feb. 2021.
11. T. Kang and Y. Shin, "Indoor Navigation Algorithm Based on a Smartphone Inertial Measurement Unit and Map Matching," *arXiv preprint*, Sep. 2021.
12. F. Wang, J. Feng, Y. Zhao, X. Zhang, S. Zhang and J. Han, "Joint Activity Recognition and Indoor Localization with WiFi Fingerprints," *arXiv preprint*, Apr. 2019.
13. Z. Yang, Z. Wei and H. Yang, "An AR-Based Indoor Navigation Vision-Based Location Positioning for Smartphone Users," in *Proc. International Conference on Intelligent Transportation Systems*, pp. 1123–1128, 2018.