Augmented Reality Navigation for the Visually Impaired

James J Nkhata

Department of Electronic, Electrical & Systems Engineering, University of Birmingham, Birmingham, UK

*jamesjr@casajuegosinc.com

Abstract: **The purpose of this study is to develop a Mobile device Augmented Reality based Assistive Technology (AT) for indoor navigation by blind users. Augmented Reality ARKit Ray-Casting is used to detect real-world objects and place virtual Marker GameObjects as way points for navigation in an AR scene. Unity Physics Ray-Casting is used to detect the user's previously placed Markers while audio and vibration cues give information to the user to aid in navigating the environment. Placenote's Cloud-based mapping engine is used to provide a consistent method to store the positons of Marker objects in the AR rendered environment. In similar, Monocular camera Depth estimation projects, methods have been used that involved image size ratio comparison upon movement, predictions based on semantic information and image to keyframe matching. Most related projects of mobile device navigation solutions have required external sensors, embedded beacon placement, pre-exiting building plans to familiarise the device to the environment. The author of this paper proposes a different implementation that leverages the software and hardware capabilities of mobile devices today. Technical evaluation experiments are carried out to evaluate the accuracy of the ARKit Ray-Casting functionality for the purpose of monocular camera distance estimation between the camera and the physical objects in the environment. The results obtained are satisfactory, demonstrating the capabilities of augmented reality based mobile device AT.**

Keywords: ARKit, blind navigation, augment reality, ar cloud, placenote, travelling aid, mobility aid, navigation, mobile device navigation

1. Introduction

Visual impairment is a condition where there is no perception of light, there is light perception of less than 3/60 or a visual field of less than 10 degrees in the better eye with best correction according to Mordini, Nierling, Wolbring, et al [1].

Assistive technologies are devices or equipment that have been made for the purpose to maintain or improve an individual's abilities and their well-being [2]. Assistive technologies (AT)s are devices designed with the sole purpose of aiding people with disabilities in carrying out activities of daily living. Existing ATs for the blind and visually impaired could roughly be divided into five groups, namely, haptic aids, travelling aids, ATs for accessible information and communication, ATs for daily living and phone / tablet applications for the blind and visually impaired according to, Mordini, Nierling, Wolbring, et al [1].

Augmented Reality (AR) is a way for computer graphics to see the real world with virtual objects superimposed or composited with the real world [3]. AR Software Development Kit (SDK), offers access to augmented reality functionalities for application development. ARKit is an AR SDK that has three distinct features [61], Tracking, Scene understanding, and Rendering. Tracking combines camera sensor data with CoreMotion data to sense how the device moves with high accuracy. Scene understanding is the ability to determine the attributes or properties about the environment around the device. Rendering is the ability to provide constant stream of camera images, tracking information and scene understanding that can used by any renderer. According to Chen, Ling and Zhang [20], due to AR's problem of insufficient computing power, a form of Cloud Computing called AR Cloud can be utilised to provide a better Augmented Reality experience.

Using ARKit and its features, this study proposes an implementation of Augmented Reality (AR) combined with Cloud-computing AR Cloud to develop an Assistive Technology mobile device application that is capable of guiding the blind through indoor environments.

The terms 'visually impaired' and 'blind' are used synonymously in this paper tin reference to individuals who have loss of useful sight.

1.1. Purpose

The purpose of the study was to evaluate the accuracy of Augmented Reality Software Development Kit (SDK), ARKit's Ray-casting as an approach to evaluate how far real-world objects are detected in an indoor environment. The author aim to explore the feasibility of combining Cloud computing with ARKit SDK's for the purpose of obstacle avoidance and navigation for the blind using an iOS (Apple mobile operating system) mobile device.

1.2. The sought to answer these questions

Question 1: Was it possible to accurately measure the distance between a mobile device's monocular camera and the physical objects in the real-world environment using Augmented Reality (AR) ARKit SDK's Ray-casting?

Question 2: Was Augmented Reality (AR) combined with Cloud computing capable of delivering a reliable indoor environment navigation system that would be accessible to blind individuals?

1.3. Challenges and Limitations

Virtual objects place in an Augmented Reality rendered scene always appear in front of real-world objects [27].

2. Literature Review

One of the major challenges faced by the visually impaired is the ability to travel through different and unknown environments [1]. Travelling aids used in history ranged from a walking stick (white cane), to helpers such as guide dogs. So-In, Arch-Int, Phaudphut et al [4] stated that, it was not easy for the visually impaired to find helpers or guide dogs.

Technological advancement led to development of mobile devices that are easily attainable and have hardware capabilities that allow usage for assistive technology applications. By leveraging the high performance hardware found on these mobile devices, guiding systems could be developed for the visually impaired to traverse through indoor environments.

In 2004, Choudhury, Aguerrevere and Barreto [5] reported on an implementation of a blind navigation system that employed the use of a Pocket-PC device to detect surrounding obstacles using ultrasonic range sensors, the travel direction using an electronic compass and give auditory information through the device. This solution required using external sensors which would in turn add bulk to the system and impact its portability to other mobile devices.

So-In, Arch-Int and Phaudphut [4] reported on a mobile phone system architecture to aid the blind in travelling which consisted of voice/speech recognition, Global Positioning System (GPS) and map services, ultrasonic sensor, and image processing service. Their system also required an external sensor and as they concluded, the use of GPS gave inaccurate indoor positioning accuracy problems which was also confirmed by Modsching, Kramer and Hagen [6].

Gallagher, Wise and Li [7] proposed a navigation system for the visually impaired that would use Wi-Fi as part of its indoor positioning method. This meant a system that would only perform well in an area with Wi-Fi signals.

Joseph, Zhang, Dryanovski et al [8] proposed a design for an indoor navigation system for blind-users using augmented reality. They used the floor plan map posted on a building to acquire a semantic plan which used the extracted landmarks from the floor plan (such as room numbers and doors) as parameters to infer the way points to each room. They went on to conclude that it was possible to assist a blind person to navigate and safely reach a destination provided they were augmented with their system to enhance the user's perception of the real world. This method would require the availability of a floor map and most likely the assistance from someone with functional vision to take the picture of the floor plan.

NavCog was a navigation system proposed by Ahmetovic et al. [9]. This was a smartphone mobility aid design to assist visually impaired individuals navigate through an unfamiliar environment. It provided turn-by-turn navigation assistance based on accurate real-time localization while conveying information about nearby points of interest as well as possible accessibility hazards during navigation. Their technical approach involved a Map server, Bluetooth beacons and an application 'NavCog App' running on iOS. The map server stored information about the environment, the Bluetooth beacons were used to localize the smartphone in the environment and the application running on a mobile device guided the user to their destination. The need to have beacons in the environment made this approach also dependent on an individual with functional vision to have them put in the environment.

Monocular camera depth estimation has had many methods developed along the years. Yamaguti, Oe, and Terada [10] proposed a method that used two images, one taken at camera point and the other taken at the point where the camera was moved along the optical axis. The distance was then calculated from the ratio between the two images using Complex Log Mapping. They concluded that the sequential method used to calculate the ratio took a long time to complete.

Saxena, Chung and Ng [11], applied supervised learning approach using a discriminatively-trained Markov Random Field (MRF) model for depth estimation from monocular images. Their method required the system to be trained using multiple images.

Liu, Gould and Koller [12] addressed the problem of depth perception using predicted semantic information. Their approach relied on accurate ground truth information for learning parameters needed for the linear regression model they used.

In 2018, Valentin, Kowdle, Barron, et al [13] proposed a novel pipeline capable of supplying dense and low latency depth maps. Their approach was based on stereo matching the most recent image and past keyframe. This method, however, was impacted when the relative pose (position and orientation) between the current frame and the selected keyframe was imprecise. Ramesh, Nagananda, Ramasangu et al [14] worked on an indoor mobility device for the visually impaired combing imaging geometry, visual odometry, object detection and distance-depth estimation algorithms. Their object detection method required for the system to be trained and while their distance–depth estimation required movement from the user.

The author's approach relies on a form of Cloud computing called AR Cloud to re-localize in a previously visited environment and load virtual objects in the scene to use as landmarks. It also uses ARKit's built-in method of inferring a depth estimation from frames consisting of 2-dimensional (2D) pictures.

3. Background Work

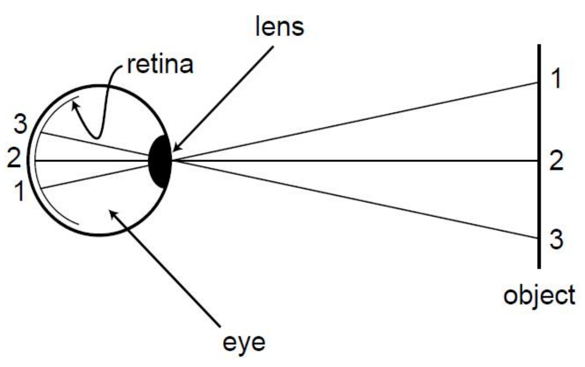
3.1. Augmented Reality

Spatial computing is a way a human can interact with a device that stores and manipulates references to real objects [15]. Augmented reality is a form of Spatial computing that works on two fundamental concepts, identifying features [16] and being able to track and recognize the environment from camera frames. In AR, devices use a process called Simultaneous Localization and Mapping (SLAM) to know more about its position in the world.

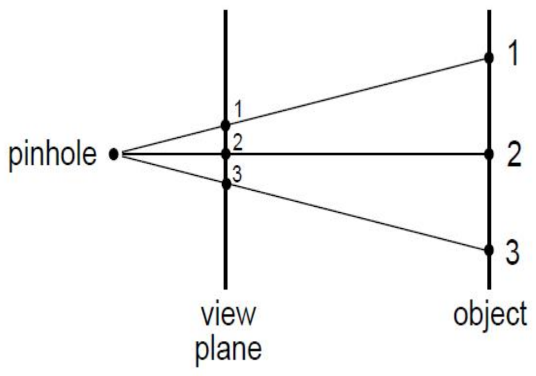
SLAM is a popular model used in robotics for autonomy. Landmarks are detected using sensors and these are compared with a map to localize the robot while trying to add other detected landmarks to the map according to Shelley [17]. Shelley went on to summarise that, early SLAM systems used Filter based SLAM methods to represent robot state and landmarks. This was done by using probability through the weighted averaging of noisy measurements. Key-Frame Based methods maintained a sparse set of key images as well as landmarks detected with the position of the camera. Dense methods (Direct methods) were those that used an entire image for frame to frame tracking. Inertial Aided methods used Inertial Measurement Unit (IMU) and the camera to carry out visual odometry, the process of estimating the egomotion of an agent by using one or many cameras as stated by Scaramuzza and Fraundorfer [18]. IMU consisted of sensors such as accelerometers, gyroscopes and sometimes magnetometers. Shelley went on to discuss, the Multi-State Constrain Kalman Filter, which was stated to be a pure odometry method capable of not having the need to build a map. This method did not estimate the positions of landmarks in the filter, instead it kept camera poses that were used to accurately triangulate landmarks using the data present. The landmarks were used as constraints on the camera poses. All the methods discussed by Shelley have been gradual improvements on previous method's shortcomings. ARKit SDK uses Visual Inertial SLAM (ViSLAM) according to Li, Yang and Chen [19].

3.2. AR Cloud Computing

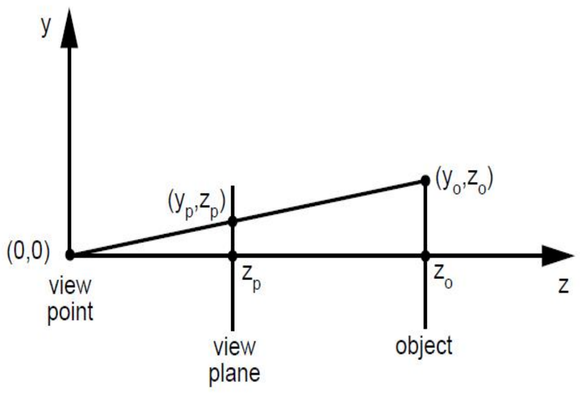
Cloud AR is the latest trend in AR computing as it provides a solution to multi user AR experience and can be



(a)



(b)



(c)

Fig. 1 *Ray-casting Foundations [23]*

(a) inverted object image on retina , (b) screen in front of pinhole, (c) 2-dimensional (2D) representation of an object's projection on to a view plane viewed from a viewpoint

attested by the many start-up companies working on AR Cloud solutions of their own [21].

3.3. Ray-casting

To give a better understanding of the method of distance estimation, the author emulates the function of the human eye. In the human eye anatomy, image recognition is carried out when light from a source enters the eye through the cornea, pupil, and lens until it finally arrives at the back of the eye (retina) as an inverted image, illustrated in Fig.1. (a). The purpose of the retina is to convert the inverted image (light) into neural signals which are sent to the brain for correction and recognition [22]. In computer graphics it is common to represent an eye with a pinhole and a view plane. The image inversion is resolved by putting the view plane in front of the pinhole before the object (pinhole camera model) as illustrated in Fig 1. (b). In perspective projection depending on the relative position of the viewpoint and the view plane, it is possible to geometrically reconstruct a 3-dimensional (3D) object on a 2-dimensional (2D) plane (view plane). A scene or model comprising of many geometric objects arranged in 3D space is used to produce a 2D image that shows the objects as viewed from a particular viewpoint (pinhole) [23][24]. In 2D, the viewpoint could be considered as the origin with its distance to the view plane and the object represented as values in the z-axis, while the height above the viewpoint could be represented in the y-axis (if the view plane is on the x-y plane parallel to the viewpoint with its centre on the z-axis) as illustrated in Fig 1. (c). If an object's distance from the view point is $\mathbf{z_0}$ and its y-coordinate height is $\mathbf{y_0}$, through the use of similar triangles and ratios, its projection $\mathbf{y_P}$ could be solved using equation (1).

$$y_p = \frac{z_p}{z_0}\, y_0, \qquad (1)$$

Ray tracing is an image–order algorithm used for image synthesis [25]. A ray tracer computes one pixel at a time to find the object that is seen at each pixel's position. According to Marschner and Shirley [24], a ray tracer could be divided into three parts, ray generation, ray intersection and shading. Ray generation computes the origin (view point) and direction of each pixel's ray, ray intersection finds the closest object intersecting the ray and shading computes the colour based on the results of the ray intersection. Since a ray could be generated from the view point out to a point on an object,

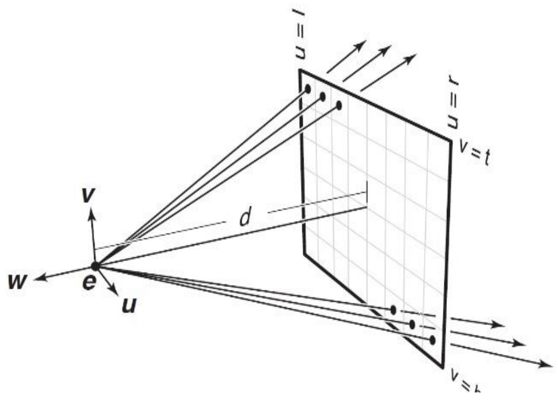


Fig. 2 *Perspective Projection: same origin, different directions [24]*

a correspondence could be determined of that point's pixels (using pixel coordinates of an image to determine the centre of a pixel) on the view plane. The method of sending out a ray to find a point on an object that correspond to a pixel on the view plane is called Ray-casting [23].

A ray could be defined as a directional vector $\mathbf{u_r}$, where $\mathbf{p_r}$ is its point of origin and the distance along the ray is measured by parameter **t**. All the points along **p** satisfy the equation (2) [23] [24].

$$p(t) = p_r + t(U_r),\ t \geq 0,\ (2)$$

Ray intersections might occur with various shapes in 3D space which involve different types of calculations to determine the point of intersection. The author did not discuss all these different types of intersections, but focused on Ray-polygon intersections as it is the most common type of shape used in this study. Given a planar polygon arbitrarily positioned and oriented in 3D space, the plane was defined as a single point on the plane $\mathbf{p_0}$ and a surface normal **n** (perpendicular to the surface) giving the orientation and position of the plane in space according to House [23]. Taking a point $\mathbf{e_0}$ with a vector from the point $\mathbf{p_0}$, the perpendicular distance of $\mathbf{e_0}$ to the plane was given by the vector ($\mathbf{e_0}$ - $\mathbf{p_0}$) projection to the normal **n**. The distance was determined by equation (3).

$$d = n \cdot (\text{e0} - \text{p0}), \qquad (3)$$

To find the ray plane intersection where **d** = 0 (the distance between the $\mathbf{e_0}$ and the plane was equal to 0), equation (2) and equation (3) was combined to give:

$$t = -\frac{n \cdot (p_r - \text{p0})}{n \cdot (U_r)}\ , \qquad (4)$$

Equation (4) gives the value of parameter **t** at which the ray intersected the plane.

A view frustrum is the shape of the region that can be seen and rendered by a perspective camera, anything that falls outside this pyramid will not be rendered or seen by the camera [29]. The view plane is usually used as the near clipping plane.

3.4. Augmented Reality Application Evaluation

Augmented reality applications were evaluated using five classifications of user evaluation methods according to Dünser, Grasset, Billinghurst [26]. Their work was based on classifying Augmented Realty (AR) user-evaluation publications according to evaluation area / evaluation type and evaluation methods / evaluation approaches used. The author was interested in the former, as in the classification of user study approaches and methods. These methods were classified as objective measurements, subjective measurements, qualitative analysis, usability evaluation techniques and informal evaluations. Objective

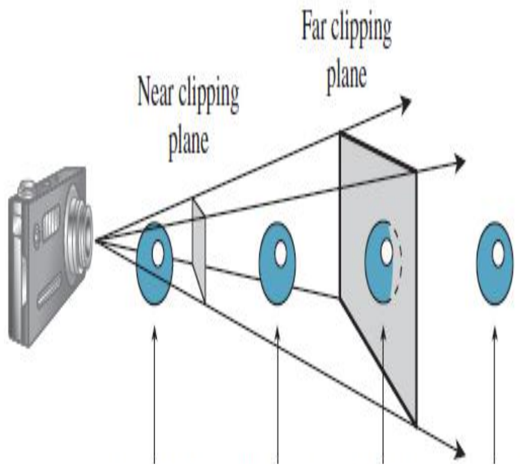


Fig. 3 *Objects outside the view frustum will not be rendered [28]*

measurements were described as studies that included object measurements such as completion times, accuracy, scores, positions, movements and number of actions. Subjective measurements studied users using questionnaires, subjective user ratings or judgements and analysed the results statistically and sometimes, descriptively. Qualitative analysis studied with formal user observations, interviews, classification or coding of user behaviour. Usability evaluation techniques that were used in evaluation techniques usually used for interface usability such as heuristic evaluation, expert based evaluation and task analysis. Informal evaluations were those such as informal observations and collection of feedback from the user.

4. Methodology

In this section the author describes the keywords and concepts to help with further understanding of the task set out in this study.

4.1. Unity

Unity is a real-time development platform [30]. Unity is capable of providing a developer with cross-platform features necessary to produce 3D (3 dimensional), 2D, VR (virtual reality) and Augmented Reality (AR) applications for devices running on different operating systems (Windows, iOS and Android). Unity uses hierarchy component–based design structure that can help speed up development [31]. It supports C# (C sharp) scripting API (application programming interface) natively.

Scene: this is used as a container for environments and menus of the game (in our case Augmented Reality application) [32].

Scripts: are used to allow the developer to trigger game events, modify components over time and respond to user input [33].

GameObject: is a container for many different components. Components are what implement functionality to GameObjects. Examples of some common objects are Directional light and Camera objects [34].

Ray-casting: is a useful way of detecting objects within the scene based on its onscreen image [35]. Unity uses physics ray-cast method.

Colliders: are components used to define the shape of objects for the purpose of physical collisions. Colliders are rough estimations of an object’s mesh [36].

Camera ViewportPointToRay: This is a function used to return a Ray from the camera through a viewport point. These coordinates are normalized relative to the camera where coordinate (0, 0) is the bottom-left of the camera and (1, 1) is the top-right of the camera [37].

Camera ScreenToViewportPoint: This function is used to transform the position from the device screen to viewport space. The Screen space is defined in pixels, with coordinate (0, 0) the bottom-left of the screen and coordinate (pixelWidth, pixelHeight) being the width and height of the device. The z position is in world units of the camera [38].

4.2. ARKit

ARKit is Apple’s proprietary SDK used to produce Augmented Reality (AR) experience by integrating iOS device camera and motion features [39]. ARKit SDK can be used in Unity 2019.1.8f1 development platform as a plugin Unity-ARKit-Plugin (now deprecated as of June 3 2019) or through the Unity package ARFoundation [40]. ARFoundation provides multi-platform support for Augmented Reality’s latest features from ARKit and ARCore [41] (Android’s AR SDK).

ARSession: is an object responsible for coordinating processes carried out by ARKit such as reading data from motion sensing hardware, controlling the device’s built-in camera and performing image analysis on captured camera images in order to create an augmented reality experience [42].

ARWorldTrackingConfiguration: is a configurable class that establishes a correspondence between the real-world and a virtual 3D-coordinate space to deliver an immersive augmented reality experience. The correspondence is maintained by tracking the device running the AR application’s motion. This motion is tracked using the device’s 6 degrees of freedom (6DOF): namely; the three rotation axes (roll, pitch, and yaw), and the three translation axes (movement in x, y, and z) [43].

Plane Detection: this is the options ARKit uses to detect flat surfaces in captured images. ARPlaneDetectionHorizontal (horizontal surfaces) are planar surfaces that are perpendicular to gravity. ARPlaneDetectionVertical (vertical surfaces) are surfaces that are parallel to gravity.

HitTest: Searches for real-world objects or ARAnchors in the captured camera image. The results returned are in the form of a list sorted from nearest to furthest in regards to the distance from the camera [44].

ARHitTestResult: This is information about a real-world surface found by examining a point on the screen. An array ARHitTestResult object is returned describing the WorldTransform (Pose of the hit Test result relative to world coordinate system), LocalTransform (Pose of the hit test result relative to the nearest anchor or feature point) and Distance (in meters from the camera to the detected surface). [45].

ARHitTestResultType: it specifies the type of hit-test to search for or the result desired [46]. The types available are FeaturePoint, EstimatedHorizontalPlane, EstimatedVerticalPlane, ExistingPlaneUsingExtent, ExistingPlane and ExistingPlaneUsingGeometry.

ARAnchor: is a position and orientation (Pose) of something of interest in the physical environment [47].

ARFrame and ARCamera: ARFrame is a video image captured as part of a session with position tracking information [48]. ARCamera is the information about the camera position and imaging characteristics for a given frame.

4.3. Placenote

Placenote is a Cloud-based mapping engine that extends ARKit functionality while providing a way to save AR content in physical locations, indoors and outdoors [49]. Placenote uses ARKit's tracking functionality and a custom implementation of point cloud mapping and localization that provides enhanced persistence (removing reliance for ARKit's persistence functionality) and simple to use cloud backend (manageable online as well as through Unity scripts). The decision to use Placenote as opposed to the ARCore's cross-platform Cloud Anchors [50] or ARKit's (Version 2.0) ARWorldMap [51] was cloud backend solution. Placenote also had other features such as map searching that offered a method to carry out a map data query based on Global Positioning System (GPS) coordinates (as well as other criteria) [49]. Placenote also offered support to the only iOS devices that were available to the author, an iPhone 6s and iPad (6th Generation).

API Key: An application programming interface key is a unique encrypted alphanumeric string that is passed in to an API to identify the calling application or user without the need to access the user's private data [52].

5. Implementation and Evaluation

In this section the author described how some of the major functions were implemented and worked together in order to produce the proof of concept for the Augmented Reality Navigation for the visually impaired. The author further described how the augmented reality navigation for the visually impaired application was evaluated. It determined the accuracy of the system in measuring distance to physical objects in the world space through the various ARHitTestResultTypes, as well as its usability.

5.1. Augmented Reality Navigation for Visually Impaired (Blind) Application

The application was built using Unity editor 2019.1.8f1 to manage the user interface, custom Marker GameObject manipulation (creation and deletion) as well as integration with the ARKit and Placenote SDKs. ARKit 2.0 was used to add AR functionality to the application through Unity-ARKit-Plugin (only one compatible with Placenote as of the writing of this thesis). Placenote 1.6.12 was used to add Cloud-based mapping engine to the application. Apple's voice-controlled voice assistant Siri [53] feature included with most post 12 October 2011 iOS devices was also an attractive feature to utilise for launching the augmented reality navigation for visually impaired application.

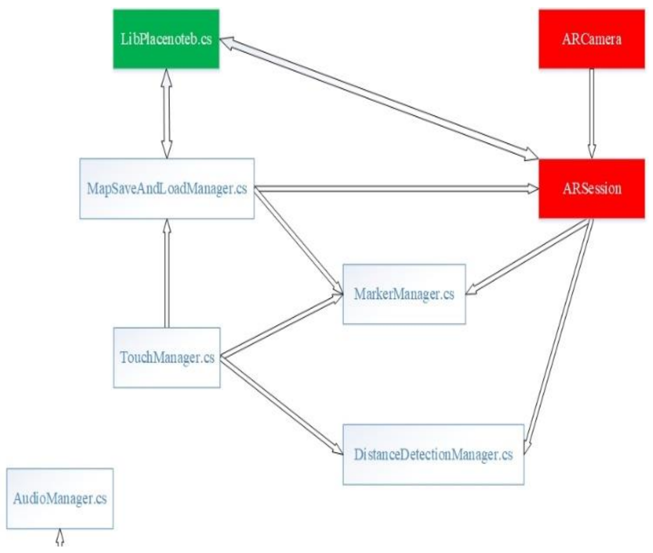


Fig. 4 *AR Navigation for the Viusally Impaired Overview*

System Overview:

Initializing ARKit and Placenote in Unity: In order to use Placenote's cloud based features, an API key had to be obtained from the Placenote developer portal (www.placenote.com). This key was generated by the website once a developer account was created with Placenote's developer portal. The API key was then placed in the LibPlacenote script element's 'API key' box in the inspector panel of the unity editor. LibPlacenote script was an element attached to the 'PlacenoteCameraManager.cs' script which was one of the objects in the 'AR navigation' scene (the name of the container for the Augmented Reality Navigation for the visually impaired application) in unity editor's scene hierarchy. A session of ARKit had to be created every time the application was run. This was done by creating an ARKit session handler and instantiating it with the 'GetARSessionNativeInterface()' function from the 'UnityARSessionNativeInterface.cs' script. This was then followed by configuring the ARKit session (ARKitWorldTrackingSessionConfiguration). Plane Detection (both horizontal and vertical) through the 'UnityARPlaneDetection.cs' script, 'UnityARAlignmentGravity' (aligning the session with device's gravity sensor data for use with plane detection) through the 'UnityARAlignment' enumeration (y-axis matches the direction of gravity as detected by the device's motion sensing hardware) [54], 'getPointCloudData' and 'enableLightEstimation' were set to true and 'targetFrameRate' set to 60 frames per second. Lastly, a Listener had to be registered to the session for events published ('OnPose' and 'OnStatusChange') by the 'LibPlacenote.cs' script (Placenote) using 'Instance.RegisterListener()' function. These settings were done in the custom 'MapSaveAndLoadManager.cs' script.

Marker GameObjects: One of the fundamental elements required to enable the visually impaired individual to navigate an area was the ability to add, remove and detect Marker GameObjects in the environment without using their sight. The GameObject used as the marker was a custom created 3D object of x, y and z dimensions (1, 0.6, and 0.6) in meters with position coordinates of (0, 1, 0 in x, y and z). The y-axis position value of 1 made sure that the marker was placed at least a meter above the y-axis of the unity world coordinates. The marker template was saved in the Assets/Scene/ARNavigation/Models/Marker folder as a prefab (prefabricated object) which could later be placed within an AR scene through screen touch commands. To allow the user to interact with the marker beyond placing it in an AR environment, a box collider was added to the centre coordinate (0, 0, 0) of the marker GameObject of size 0.8 x 2.34 x 1.6 (x, y and z) as demonstrated in Fig. 6. This allowed the object to respond to physics ray-cast (unity) collisions as

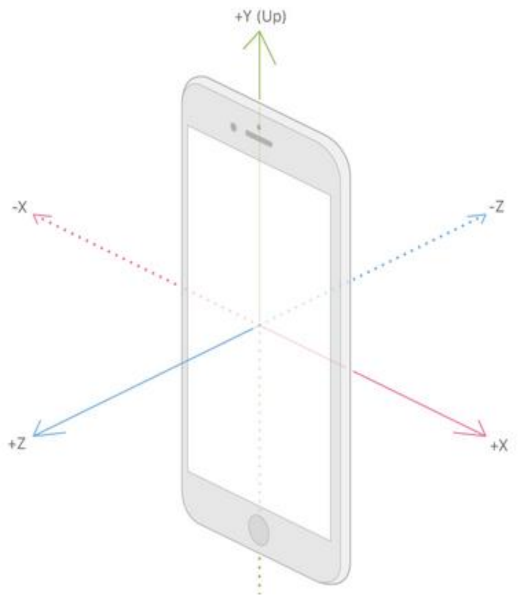
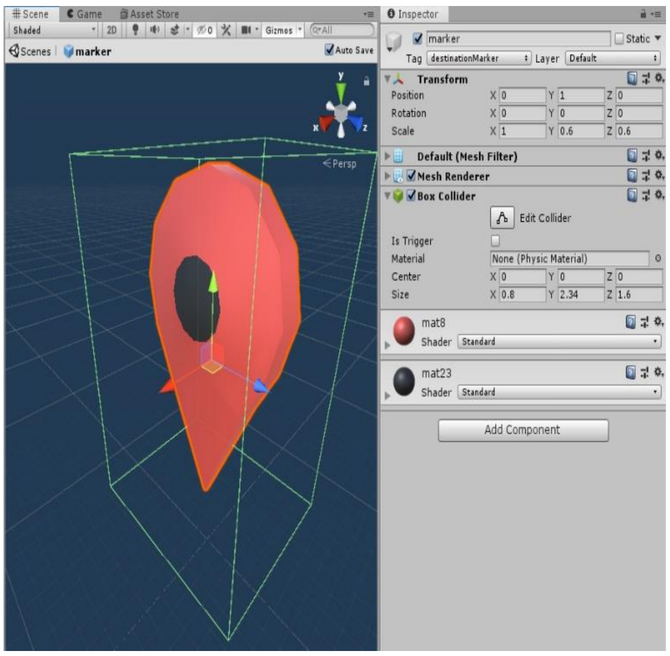


Fig. 5 *In gravity-only alignment, X and Z directions are relative to the device's initial orientation [54]*



```
marker                                   Static
Tag  destinationMarker    Layer  Default
Transform
Position    X 0     Y 1     Z 0
Rotation    X 0     Y 0     Z 0
Scale       X 1     Y 0.6   Z 0.6
Default (Mesh Filter)
Mesh Renderer
Box Collider
            Edit Collider
Is Trigger
Material    None (Physic Material)
Center      X 0     Y 0     Z 0
Size        X 0.8   Y 2.34  Z 1.6
mat8
Shader  Standard
mat23
Shader  Standard
Add Component
```

Fig. 6 *Marker GameObject used as a waypoint*

the box collider would be identified as an extension of the geometry of the marker GameObject.

Adding a Marker: This was achieved by modifying 'ARKitHitTesting.cs' [63] example code from the Placenote website to produce the 'AddMarker()' function in the 'MarkerManager.cs' script. 'AddMarker()' passed the camera's centre point 2D coordinates through the 'ScreenToViewportPoint()' function to the custom 'AddMarkerHitTestWithResultType()' function to perform a HitTest. The 'EstimatedHorizontalPlane' and the 'FeaturePoint' ARHitTestResultTypes were used for the HitTest because they gave the best results when placing marker GameObjects in the scene (vertically placed ARHitTestResultTypes did not give an ideal orientation for the marker GameObjects). HitTest Pose coordinates (after a few matrix transform processes) were used to instantiate a marker model into the scene as well as add its information to the 'ModelObjList' and 'ModelInfoList' (for later reference) using the 'AddModel()' function.

Removing a Marker: 'RemoveMarker()' function in the 'MarkerManager.cs' script was used to remove markers placed in the scene. This function worked by looping through the 'ModelObjList' elements to find the '.name' (custom string entry set when instantiating marker GameObjects using the 'AddModel' function) property value that was equal to the value obtained from the object being 'hit' by the ray-cast's 'GetInstanceID()' function. The matching 'ModelObjList' entry's index was extracted and used to reference the 'ModelInfoList' to remove the objects entry as well.

Detecting Markers: was done using 'MarkerDirectionFinder()' function in the custom 'MarkerManager.cs' class. This used a Physics ray-cast (unity's world space object ray-cast) to cast a ray from the camera using the 'ViewportPointToRay()' function with vector3 coordinates that corresponded to the centre of the screen. The hit ('objectHit') data returned was checked to see if it had a Tag (word linked to a GameObject) of 'destinationMarker'. If the tag returned true, then further action would be carried out like sounding off a proximity alarm, start vibration, setting a flag to start other processes and returning the distance information of the detected marker GameObject.

Marker Occlusion: a custom 'markerActivator' coroutine was used to only activate marker GameObjects that were within a certain range (between 1 – 5 meters) of the camera's position using the 'Vector3.Distance()' function. The markers beyond that range were deactivated. Markers that fell within 0.60 meters of the camera were removed entirely from the current list as they were considered as already visited. A map load would reinstate the markers within the scene after a successful map re-localization.

Creating, Extending and saving a Map: Marker pose data and GPS location data were the elements that made up a map. Creating a map involved starting a mapping session through Placenote's 'LibPlacenote.Instance.StartSession()' function. Map saving used the 'LibPlacenote.Instance.SaveMap()' callback that used 'MapMetadataSettable()' function to convert marker GameObject and GPS location into a serializable format by using the JSON [64]. To extend the map, 'LibPlacenote.Instance.StartSession(true)' function with the 'true' Boolean was used. 'MapSaveAndLoadManager.cs' script managed all the mapping functions.

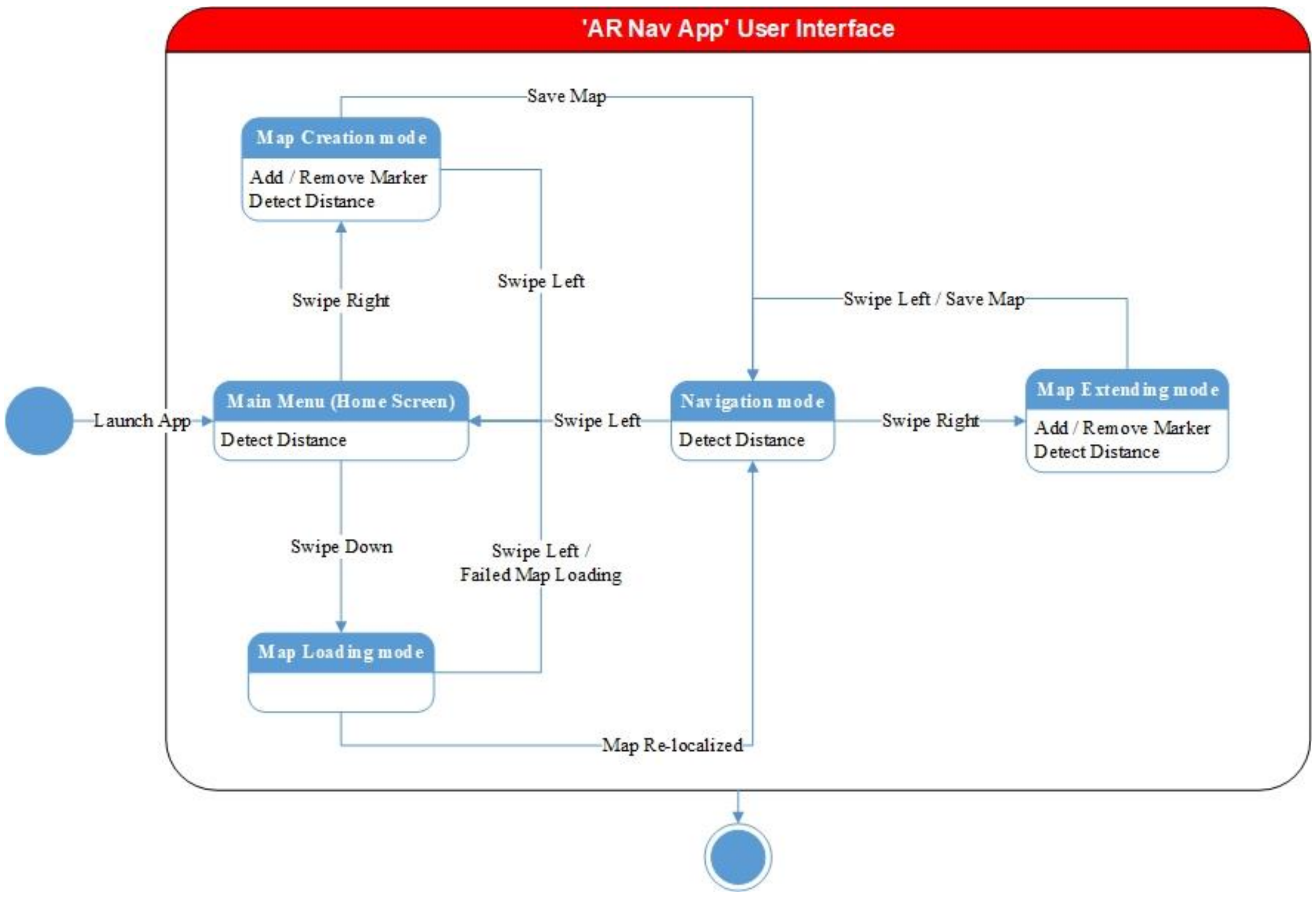
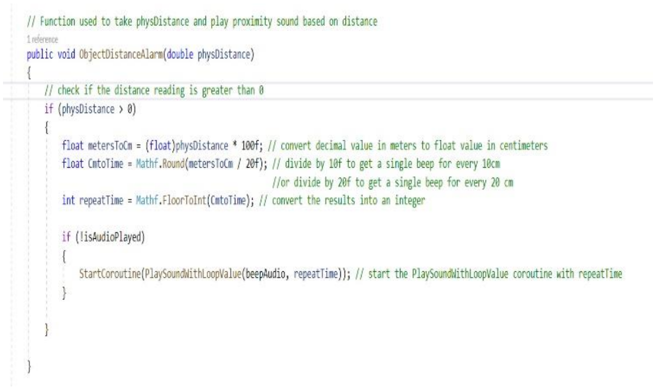


Fig. 7 *Augmented Reality Navigation User Interface state changes using touch screen swipes*

Map Loading: was carried out in a similar manner as Map saving, however a different callback 'LibPlacenote.Instance.LoadMap' and the map metadata was de-serialized. In its current implementation the application would use its last used Map ID to query for a map load. This meant maps could only be loaded if they were the last one used ('/mapIDFile.txt').

Navigation and Distance Detection: using the two types of ray-casting methods discussed in previous sections, the user was expected to infer a path way to navigate the environment by using the information they provided as guides. ARKit's HitTest ray-cast was used to notify the user of the distances of objects around them in the environment, while Unity's physics ray-cast was used to detect previously placed marker GameObjects in the AR scene. If the device was facing the direction where a marker was present, the physics ray-cast would return a 'hit' with the marker's collider, notifying the user with a vibration and audio signal from the device. This gave the user an idea of the way to go. By getting within a certain range of the marker GameObject, the coroutine 'RadiusActivator' in the 'MarkerManager.cs' script would deactivate the current marker and activate the next marker in the scene.

User Interface: As the application was intended for individuals that are blind, a suitable interface had to be designed to make it accessible. The application was given a short product name of 'AR Nav App' through unity's 'Project Settings – Player – Product Name' entry field. This allowed the user to launch the deployed application on an AR supported iOS device by using a short voice command ('launch AR Nav') through Apple's Siri assistant. The decision was made to let the user access the application's functionalities through touch and hearing. Screen swipe gestures (see appendix for Table 1) commands implemented in the 'TouchManager.cs' script were used to navigate and access the applications functionalities as illustrated in Fig 7. The touch commands were incorporated with audio notifications that would inform the user of the current process being carried out or the state the application was in. This was achieved by assigning each recorded notification (located in the Assets/Resources/Audio/ folder of the application's source files) to an 'AudioClip' object type variable and playing it back through an 'AudioSource' type object in the 'AudioManager.cs' script. Instead of just simple playback of an AudioClip, some functionalities needed custom methods to play the AudioClip objects with special behaviour. These ranged from having to play the AudioClip object with a delay in seconds (before or after), playing an AudioClip to loop specified number of times, to some that played when the device was in a specific state. 'ObjectDistanceAlarm()' function was used to give proximity sound based on the reading from the 'HitTest' distance reading. It worked by using the 'physDistance' value from the 'DistanceFinder()' function from the 'DistanceDetectionManager.cs' script as a parameter and converting the number into a 'repeatTime' integer. The 'repeatTime' would later be used as a parameter to a coroutine that would play a single beep for every 10 units in the 'physDistance' (example the value 120 will give 12 beeps).



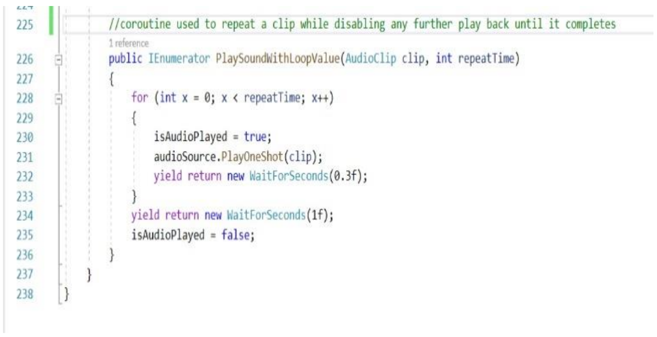
```
// Function used to take physDistance and play proximity sound based on distance
1 reference
public void ObjectDistanceAlarm(double physDistance)
{
    // check if the distance reading is greater than 0
    if (physDistance > 0)
    {
        float metersToCm = (float)physDistance * 100f; // convert decimal value in meters to float value in centimeters
        float CmtoTime = Mathf.Round(metersToCm / 20f); // divide by 10f to get a single beep for every 10cm
                                                        //or divide by 20f to get a single beep for every 20 cm
        int repeatTime = Mathf.FloorToInt(CmtoTime); // convert the results into an integer

        if (!isAudioPlayed)
        {
            StartCoroutine(PlaySoundWithLoopValue(beepAudio, repeatTime)); // start the PlaySoundWithLoopValue coroutine with repeatTime
        }

    }

}
```

Fig. 8 *ObjectDistanceAlarm() function in AudioManager.cs*



```
        //coroutine used to repeat a clip while disabling any further play back until it completes
        1 reference
        public IEnumerator PlaySoundWithLoopValue(AudioClip clip, int repeatTime)
        {
            for (int x = 0; x < repeatTime; x++)
            {
                isAudioPlayed = true;
                audioSource.PlayOneShot(clip);
                yield return new WaitForSeconds(0.3f);
            }
            yield return new WaitForSeconds(1f);
            isAudioPlayed = false;
        }
    }
}
```

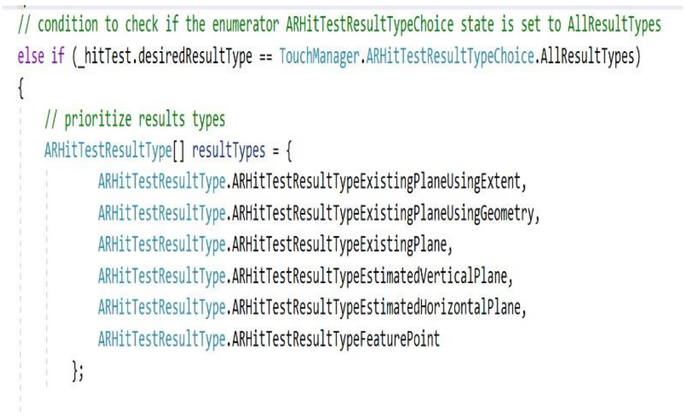
Fig. 9 *PlaySoundWithLoopValue() coroutine in 'AudioManager.cs' script*

Debugging and Testing Interface: To assist with testing, Placenote's Features Visualizer, modified 'PlacenoteARGeneratePlane.cs' script (that does not save planes), User interface text overlays and a 'RayHit Indicator Quad' GameObject were used to provide visual information about the deployed application running on a device.

5.2. Distance Estimation Evaluation

In this section, the author discussed how ARKit's distance estimation method was evaluated. When a camera image was processed by ARKit, a 2D point corresponded to multiple points on a 3D line and returned all the objects detected along that line depending on the ARHitTestResultType specified. When a 'hit' occurred, ARKit's HitTest returned an array that contained the intersection information for all the objects detected sorted from the closest to the camera to the furthest [44]. By using the 'hitResult.distance' property, the distance to the first point of intersection could be found which was used to determine how far the nearest obstacle in the real-space was to the camera. The author decided on an objective measurement approach similar to the work conducted by Bergquist and Stenbeck [55]. The approach involved using a tape measure to establish a ground truth (exact value) for measurements between the device (running the augmented reality navigation for the visually impaired application) and an object in the real-space environment. Another measurement was then carried out simultaneously using ARKit's HitTest 'hitResult.distance' property between the object and the device. This was to be done at varied distances (0.5, 1, 1.5 and 2 meters), while using the different ARHitTestResultTypes. This expected to have two outcomes, the first was to determine the best suited ARHitTestResultType's for each tested object type in the real-space, and the second was to determine the accuracy of the combined efforts of the ARHitTestResultTypes compared to the tape measure's readings.

Procedure: through the use of the 'SwitchDesiredResultType()' function in the 'TouchManager.cs' it was possible to toggle between the six ARHitTestResultTypes while running the deployed application (with a seventh option that used all the ARHitTestResultTypes as an array that prioritized the types according to the order they were listed as shown in Fig. 10). This allowed us to specify an ARHitTestResultType to obtain 'hitTest.distance' values for in the 'DistanceFinder()' function of the 'DistanceDetectionManager.cs' script without having to change the script.



```
// condition to check if the enumerator ARHitTestResultTypeChoice state is set to AllResultTypes
else if (_hitTest.desiredResultType == TouchManager.ARHitTestResultTypeChoice.AllResultTypes)
{
    // prioritize results types
    ARHitTestResultType[] resultTypes = {
            ARHitTestResultType.ARHitTestResultTypeExistingPlaneUsingExtent,
            ARHitTestResultType.ARHitTestResultTypeExistingPlaneUsingGeometry,
            ARHitTestResultType.ARHitTestResultTypeExistingPlane,
            ARHitTestResultType.ARHitTestResultTypeEstimatedVerticalPlane,
            ARHitTestResultType.ARHitTestResultTypeEstimatedHorizontalPlane,
            ARHitTestResultType.ARHitTestResultTypeFeaturePoint
    };
```

Fig. 10. P*rioritized ARHitTestResultType List for HitTests*

The visual testing tools were activated by putting the application in the 'map creation mode' (swipe right from the main menu screen). These tools were the User interface Text overlays (used to display the HitTest Physical Distance 'PhysDist' and the current ARHitTestResultType selected), Placenote's 'Features Visualizer' used to display point cloud data [62] as green squares in the scene, Modified 'PlacenoteARGeneratePlane.cs' script (used to display the detected planes) and custom 'RayHit Indicator Quad' GameObject (used to show the area in the scene where HitTest is intersecting). Two experimenters were required to carry out the measurements at set tape measure distances (0.5, 1, 1.5 and 2 meters), where one side of the tape measure was held up to the device while the other side was held up to the object being measured. The distance detection 'HitTest' was carried out by using a device touch gesture to place the RayHit Indicator overlay on the object through the device. This was repeated for each ARHitTestResultType.

Data Analysis: The measurements obtained in the distance estimation evaluation was in meters (see appendix Table 2 – Table 8 for measurement results). For each object two readings were made at different distance intervals. This process was done for all the ARHitTestResultTypes. To analyse the data, the Absolute Error [56] was calculated for each entry. The equation to determine the absolute value was given by equation (5), were A_{Test} is the measured value and $A_{ExactValue}$ was the value of the tape measure.

$$\text{Absolute Error} = |A_{Test} - A_{ExactValue}|\,, \qquad (5)$$

The average absolute error was calculated for each object test pairs, repeated for all ARHitTestResultTypes. This was followed by calculating the Percentage Error for each object's distance intervals using equation (6).

$$\text{Percentage Error} = \frac{\text{Absolute Error}}{A_{ExactValue}} \text{ x } 100\,, \qquad (6)$$

Mean Absolute Percentage Error (MAPE) [57] was calculated for each object's distance intervals mean absolute error using equation (7).

$$\text{MAPE} = \frac{1}{\text{n}} \sum_{t=1}^{n} Percentage\ Error\ , \qquad (7)$$

The MAPE gave the accuracy percentage of the distance estimation of the object for that particular ARHitTestResultType HitTest. Equation (8) was used to calculate Mean Absolute Deviation (MAD) [58], were X_i was each percentage error and the X_{Mean} was the mean absolute percentage error.

$$\text{MAD} = \frac{\sum|X_i - X_{Mean}|}{n}\ , \qquad (8)$$

This was done to determine the spread of the percentage error results from the Mean absolute percentage error.

6. Results and Discussion

In this section findings of the study are discussed.

6.1. Results

The distance results obtained for 'EstimatedHorizontalPlane' showed large percentages of MAPE for object 1 and 2 with significantly variation in MAD as shown in Table 9 to 2 decimal places (2 d.p.). Object 4 gave the lowest MAPE value 5.79% with the least MAD value of 1.96%. This was expected as the surfaces measured were not perpendicular to gravity.

Table 9. MAPE and MAD Calculations for EstimatedHorizontalPlane HitTest Results (2 d.p.)

Test	Mean Absolute Percentage Error	Mean Absolute Deviation
Object 1	23.10	11.65
Object 2	15.65	6.40
Object 3	8.33	2.17
Object 4	5.79	1.96

The 'EstimatedVerticalPlane' distance reading worked for object 1 (canvas vertically oriented to the wall) with a MAPE of 5.79 % and MAD of 2.48. The estimations failed for the rest of the objects as shown by the 100% MAPE. This was due to object 2 to 4 not being oriented exactly parallel to gravity. The results calculated results are shown in Table 10 to 2 decimal places (2 d.p.).

Table 10. MAPE and MAD Calculations for EstimatedVerticalPlane HitTest Results (2 d.p.)

Test	Mean Absolute Percentage Error	Mean Absolute Deviation
Object 1	5.73	2.48
Object 2	100	0
Object 3	100	0
Object 4	100	0

'ExistingPlane' demonstrated significantly lower MAPE and MAD values (3% and 0.75%) giving a nearly exact distance reading across the distance intervals. The rest of the objects also returned a low average percentage error across the intervals with little deviation, except for object 3 with a higher MAPE value of 9.42% and 5.08% MAD which was still on the higher accuracy (below 10% MAPE) for the application of distance estimation. The calculations are demonstrated in Table 11 to 2 decimal places (2 d.p.).

Table 11. MAPE and MAD Calculations for ExistingPlane HitTest Results (2 d.p.)

Test	Mean Absolute Percentage Error	Mean Absolute Deviation
Object 1	3	0.75
Object 2	7.13	4.44
Object 3	9.42	5.08
Object 4	7.08	1.46

'ExistingPlaneUsingExtent' showed very low MAPE and MAD values across all objects with the highest values of 7.17% (MAPE) and 5.33% (MAPE) calculated from object 4 as shown in Table 12 to 2 decimal places (2 d.p.).

Table 12. MAPE and MAD Calculations for ExistingPlaneUsingExtent HitTest Results (2 d.p.)

Test	Mean Absolute Percentage Error	Mean Absolute Deviation
Object 1	2.63	1.13
Object 2	4.56	1.94
Object 3	4.15	2.43
Object 4	7.17	5.33

'ExistingplaneUsingGeometry' produced mixed results for all the objects with the worst coming from object 2 with a MAPE value of 9.10% and MAD value of 8.95% as shown in Table 13 (to 2 d.p.). Overall, the result were acceptable for the application (below 10% MAPE).

Table 13. MAPE and MAD Calculations for ExistingPlaneUsingGeometry HitTest Results (2 d.p.)

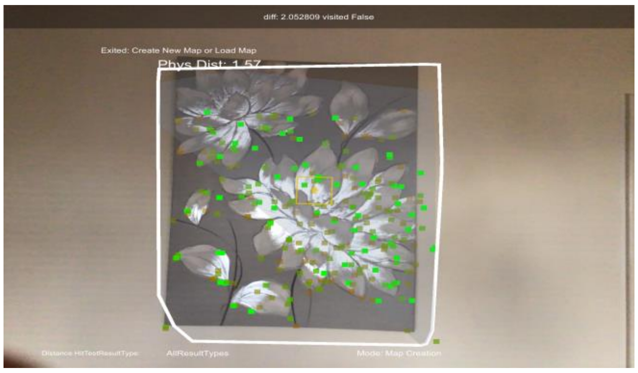
Test	Mean Absolute Percentage Error	Mean Absolute Deviation
Object 1	5.08	2.71
Object 2	9.10	8.95
Object 3	4.40	3.35
Object 4	6.58	4.17

'FeaturePoint' gave good results for all objects with very acceptable MAPE and MAD values for all objects as demonstrated in Table 14 to 2 decimal places (2 d.p.).

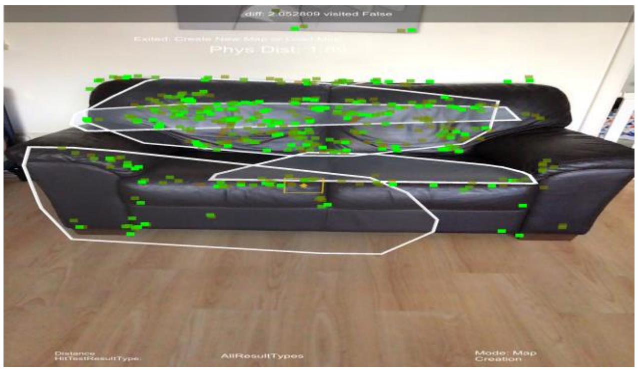
Table 14. MAPE and MAD Calculations for FeaturePoint HitTest Results (2 d.p.)

Test	Mean Absolute Percentage Error	Mean Absolute Deviation
Object 1	3.5	2
Object 2	5.13	2.19
Object 3	5.31	2.84
Object 4	7.88	3.88

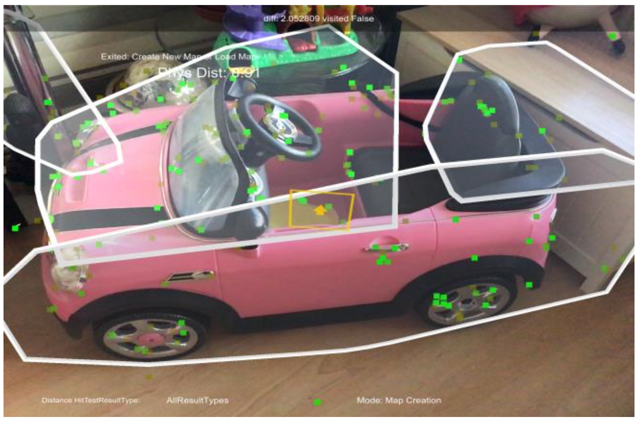
The MAPE and MAD calculations from combined six ARHitTestTypes results are demonstrated in Table 15 to 2


diff: 2.052809 visited False
Exited: Create New Map or Load Map
Phys Dist: 1.57
Distance HitTestResultType: AllResultTypes
Mode: Map Creation

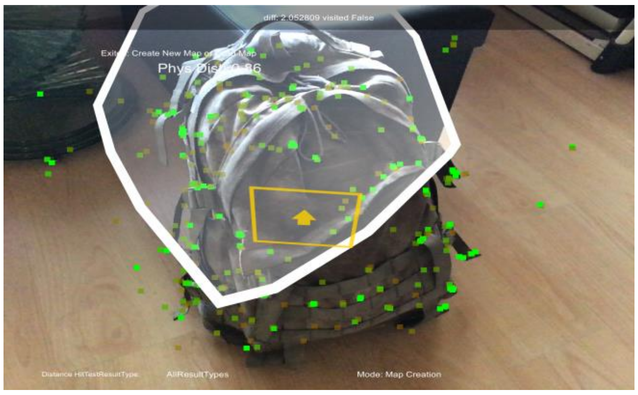
(a)


diff: 2.052809 visited False
Distance HitTestResultType: AllResultTypes
Mode: Map Creation

(b)


diff: 2.052809 visited False
Distance HitTestResultType: AllResultTypes
Mode: Map Creation

(c)


diff: 2.052809 visited False
Distance HitTestResultType: AllResultTypes
Mode: Map Creation

(d)

Fig. 11. *Objects used to evaluate HitTest Distance*

(a) Object 1: Canvas on wall, (b) Object 2: Sofa, (c) Object 3: Ride on Toy Car, (d) Object 4: Backpack

decimal places (2 d.p.). The resulting MAPE and MAD values were the lowest for all the tests done for the HitTest distance estimations.

Table 15. MAPE and MAD Calculations for AllARHitTestResults HitTest Results (2 d.p.)

Test	Mean Absolute Percentage Error	Mean Absolute Deviation
Object 1	3.73	1.52
Object 2	2.65	1.18
Object 3	4.60	2.20
Object 4	4.86	3.88

6.2. Discussion

Tape Measure Method: From the calculations carried out from the recorded results, the author suspected there were inaccuracies experienced because of the method of manually holding the tape measure with the device running the application. This led to slight hand movements that altered the readings. Another error in our readings came from the inconsistencies in the area where the RayHit Indicator Quad was placed and a reading was taken. They were instances were different areas had to be used for similar pairs because a reading could not be obtained depending on the type of ARHitTestType that was being used.

Distance Estimation Method: The developed system in comparison to similar depth estimation systems [10], [11], [12], [13] and [14] had a few benefits. Our system did not require to one form or another type of training to be performed using images [11] [14] or semantic information [12]. It was done in real-time using ARKit’s ViSLAM frame processing and HitTest method. Our implementation did not require a comparison of images [10] [13] or any movement in the direction of the obstacle [14] from the device to estimate the distance. However, a slight motion of the device horizontally or vertically was required at times during testing to get a good reading of the estimation. This was advantageous in the application for navigation because the author believed it could give the user a chance to get distance readings of the environment while stationary to get familiar with the area before starting to navigate through it. The distance interval minimum value of 0.5 meters to a maximum 0f 2.0 meters were selected as it was estimated that it was the effective range to use for object detection for the navigation of an indoor environment. As for the better ARHitTestTypes for the objects, it was better to use all of them to improve the chances of returning a good estimation. It was noted at times that 'FeaturePoint' ARHitTestType would return a reading that was immediately in front of the camera. Although it was rare it would still give false readings as the ARKit was updating its detected feature point data.

Object Texture: The texture of the object made a significant impact on the distance estimation as illustrated by Fig 11. (a) – (d). The smoother or less textured the surface of an object the lower the density of feature points. This meant objects with less feature points detected were harder to detect as planes. This is problem that has been faced with AR and although others have done work trying to tackle this issue [59]. This work could be included as to further expand on the study in the future.

Light Estimation: The amount of Lighting in the scene played a crucial part in how well ARKit detected feature points from the environment. As not much can be done on how well a scene has light, a low light warning function could have been implemented to inform the user when the lighting is not good enough to use the application.

Sample Size: Although a significant amount of readings were taken for all ARHitTestResultTypes, in reality only two were recorded for each object interval pair. A larger number of readings and objects could have given a better representation of the HitTest distance estimation method performance for the application of navigation.

Time Constraints: Due to time constraints a significant amount of work was not done. Some of this work involved all the evaluation work needed to test the system in its entirety as an Augmented Reality powered Navigation system for the visually impaired. The rest was establishing a multi-user experience that used the same API and GPS data to load and update on a map created by a different device in the same environment.

Power Consumption: The application implemented, always has plane detection on regardless of the mode it is operating in. This has a significant impact on the battery life. An option to switch plane detection on or off can be implemented to address this issue.

7. Conclusion

The study investigated the feasibility of a mobile device application for iOS devices to assist visually impaired individuals navigate through indoor environment using audio and vibration feedback. The usability and limitations of the interface was determined, and future work recommended to be carried out. The research findings suggest that it is possible to accurately measure the distance between a mobile device's monocular camera and the physical objects in the real-world environment using Augmented Reality (AR) ARKit SDK's Ray-casting, hence, the more studies required to address the gaps in existing knowledge.

References

[1] Mordini, E., Nierling, L., Wolbring, G., et al.: 'Assistive technologies for people with disabilities. Part II: Current and emerging technologies'. *EPRS| European Parliamentary Research Service, European Parliament*, Rue Wiertz, Brussels, January 2018, pp. 76

[2] 'World Health Organization', https://www.who.int/disabilities/technology/en, accessed April 2019

[3] Azuma, R.T.: 'A survey of augmented reality', *Presence: Teleoperators & Virtual Environments*,1997, *6*, (4), p. 356

[4] So-In, C., Arch-Int, S., Phaudphut, et al.: 'A new mobile phone system architecture for the navigational travelling blind'. *2012 Ninth International Conference on Computer Science and Software Engineering,* Bangkok, Thailand, May 2012, pp. 54 - 59

[5] Choudhury, M.H., Aguerrevere, D. and Barreto, A.B.: 'A pocket-PC based navigational aid for blind individuals'. *2004 IEEE Symposium on Virtual Environments, Human-Computer Interfaces and Measurement Systems,* Boston, MA, USA, July *2004,* pp. 43-48

[6] Modsching, M., Kramer, R. and ten Hagen, K.: 2006, March. 'Field trial on GPS Accuracy in a medium size city: The influence of built-up', *3rd workshop on positioning, navigation and communication, 2006,* 2006, pp. 209-218

[7] Gallagher, T., Wise, E., Li, et al: 'Indoor positioning system based on sensor fusion for the blind and visually impaired'. *2012 International Conference on Indoor Positioning and Indoor Navigation,* Sydney, NSW, Australia, *November, 2012,* pp. 1-9

[8] Joseph, S.L., Zhang, X., Dryanovski, I., et al.: 'Semantic indoor navigation with a blind-user oriented augmented reality'. *2013 IEEE International Conference on Systems, Man, and Cybernetics,* Manchester, UK*, October 2013,* pp. 3585-3591

[9] Ahmetovic, D., Gleason, C., Ruan, et al.: 'NavCog: A Navigational Cognitive Assistant for the Blind'. Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services, Florence, Italy, September 2016, pp. 90-99

[10] Yamaguti, N., Oe, S. and Terada, K.: 'A method of distance measurement by using monocular camera'. *Proceedings of the 36th SICE annual conference. International session papers,*Tokushima, Japan,*July 1997,* pp. 1255-1260

[11] Saxena, A., Chung, S.H. and Ng, A.Y.: 'Learning depth from single monocular images', *Advances in neural information processing systems, 2006,* pp. 1161-1168

[12] Liu, B., Gould, S. and Koller, D.: 'Single image depth estimation from predicted semantic labels'. *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition,* San Francisco, CA, USA ,*June 2010,* pp. 1253-1260

[13] Valentin, J., Kowdle, A, Barron, J.T., et al.: 'Depth from motion for smartphone AR', ACM Transactions on Graphics (TOG), 2018, 37, pp. 1 - 19

[14] Ramesh, K., Nagananda, S. N., Ramasangu, H.: 'Real-time localization and navigation in an indoor environment using monocular camera for visually impaired'. *2018 5th International Conference on Industrial Engineering and Applications (ICIEA)*, Singapore, April 2018, pp. 122-128

[15] Greenwold, S.: 'Spatial computing'. Master Thesis, *Graduate School of Science in Media Arts and Sciences, Massachusetts Institute of Technology, Massachusetts, 2003*

[16] Lowe, D.G.: 'Distinctive image features from scale-invariant keypoints', *International journal of computer vision*, 2004, *60* (2), pp. 91-92

[17] Shelley, M.A.: 'Monocular visual inertial odometry on a mobile device'. *Master's thesis, Institut für Informatik, TU München, Germany,* 2014

[18] Scaramuzza, D. and Fraundorfer, F.: 'Visual odometry [tutorial]', *IEEE robotics & automation magazine*, 2011, *18* (4), p. 80

[19] Li, J., Yang, B**.**, Chen, D., et al.: 'Survey and evaluation of monocular visual-inertial SLAM algorithms for augmented reality'. Virtual Reality & Intelligent Hardware, 2019, 1(4), pp. 1-5

[20] Chen, M., Ling, C. and Zhang, W.: 'Analysis of augmented reality application based on cloud computing'. *2011 4th International Congress on Image and Signal Processing,* Shanghai, China, October 2011, (Vol. 2, pp. 569-572

[21] 'ARKit and ARCore will not usher massive adoption of mobile AR', https://medium.com/super-ventures-blog/arkit-and-arcore-will-not-usher-massive-adoption-of-mobile-ar-da3d87f7e5ad, accessed September 2019

[22] 'Retina', https://www.healthline.com/human-body-maps/retina#1, accessed August 2019

[23] 'Donald H. House - Computer Graphics', https://people.cs.clemson.edu/~dhouse/courses/405/notes/raycast.pdf , accessed September 2019

[24] Marschner, S. and Shirley, P: '*Fundamentals of computer graphics'* (CRC Press, 2015) pp. 69 – 81

[25] Glassner, A.S.: '*An introduction to ray tracing'* (Elsevier, 1989, edn.) p. 1

[26] Dünser, A., Grasset, R. and Billinghurst, M.: '*A survey of evaluation techniques used in augmented reality studies'* (Human Interface Technology Laboratory New Zealand, 2008)

[27] 'Why is Occlusion in Augmented Reality So Hard?', https://hackernoon.com/why-is-occlusion-in-augmented-reality-so-hard-7bc8041607f9, accessed August 2019

[28] Hughes, J.F., Van Dam, A., Foley, J.D., et al: '*Computer graphics: principles and practice'*. (Pearson Education, 2014, 3rd edn.). p. 303

[29] 'Understanding the View Frustum', https://docs.unity3d.com/2019.3/Documentation/Manual/UnderstandingFrustum.html, accessed August 2019

[30] 'Unit', https://unity.com, accessed August 2019

[31] 'What is Unity', https://conceptartempire.com/what-is-unity, accessed September 2019

[32] 'Scene', https://docs.unity3d.com/2019.3/Documentation/Manual/CreatingScenes.html, accessed September 2019

[33] 'Scripts', https://docs.unity3d.com/2019.3/Documentation/Manual/CreatingAndUsingScripts.html, accessed September 2019

[34] 'GameObject', https://docs.unity3d.com/2019.3/Documentation/Manual/GameObjects.html, accessed September 2019

[35] 'Camera Rays', https://docs.unity3d.com/2019.3/Documentation/Manual/CameraRays.html, accessed September 2019

[36] 'Colliders', https://docs.unity3d.com/2019.3/Documentation/Manual/CollidersOverview.html, accessed September 2019

[37] 'Camera.ViewportPointToRay', https://docs.unity3d.com/2019.3/Documentation/ScriptReference/Camera.ViewportPointToRay.html, accessed September 2019

[38] 'Camera.ScreenToViewportPoint', https://docs.unity3d.com/2019.3/Documentation/ScriptReference/Camera.ScreenToViewportPoint.html, accessed August 2019

[39] 'ARKit | Apple Developer Documentation', https://developer.apple.com/documentation/arkit, accessed August 2019

[40] 'About AR Foundation', https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@2.2/manual/index.html, accessed August 2019

[41] 'ARCore', https://developers.google.com/ar, accessed August 2019

[42] 'ARSession', https://developer.apple.com/documentation/arkit/arsession?language=objc, accessed August 2019

[43] 'ARWorldTrackingConfiguration', https://developer.apple.com/documentation/arkit/arworldtrackingconfiguration?language=objc, accessed August 2019

[44] 'HitTest', https://developer.apple.com/documentation/arkit/arframe/2875718-hittest?language=objc, accessed August 2019

[45] 'ARHitTestResult', https://developer.apple.com/documentation/arkit/arhittestresult?language=objc, accessed August 2019

[46] 'ARHitTestResult', https://developer.apple.com/documentation/arkit/arhittestresulttype?language=objc, accessed August 2019

[47] 'ARAnchor', https://developer.apple.com/documentation/arkit/aranchor?language=objc, accessed August 2019

[48] 'ARFrame', https://developer.apple.com/documentation/arkit/arframe?language=objc, accessed August 2019

[49] 'What is Placenote SDK?', https://placenote.com/docs/unity/about, accessed September 2019

[50] 'Cloud Anchor', https://developers.google.com/ar/develop/java/cloud-anchors/overview-android, accessed September 2019

[51] 'ARWorldMap', https://developer.apple.com/documentation/arkit/arworldmap?language=objc, accessed September 2019

[52] 'Understanding API keys', https://cloud.ibm.com/docs/iam?topic=iam-manapikey, accessed September 2019

[53] 'Siri', https://www.apple.com/uk/siri, accessed August 2019

[54] 'ARWorldAlignmentGravity', https://developer.apple.com/documentation/arkit/arworldalignment/arworldalignmentgravity?language=occ, accessed August 2019

[55] Bergquist, R. and Stenbeck, N.: 'Using Augmented Reality to Measure Vertical Surfaces'. Bachelor Thesis, Linköping University, 2018)

[56] Helfrick A.D. and Cooper, W.D.: 'Modern Electronic Instrumentation and measurement Technique' (Low Price Edition, 2005)

[57] Hyndman, R.J. and Koehler, A.B.: 'Another look at measures of forecast accuracy', *International journal of forecasting*, 2006, *22*, (4), p. 683

[58] Leys, C., Ley, C., Klein, O.: 'Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median', *Journal of Experimental Social Psychology*, 2013, *49*, (4), pp. 765

[59] Hodan, T., Damen, D., Mayol-Cuevas, W., et al.: 'Efficient Texture-less Object Detection for Augmented Reality Guidance'. *2015 IEEE International Symposium on Mixed and Augmented Reality Workshops*, Fukuoka, Japan, October 2015, pp. 81-86

[60] 'Flaticons', https://www.flaticon.com/authors/yannick, accessed August 2019

[61] 'Apple arkit Overview', https://intranet.birmingham.ac.uk/it/innovation/documents/public/Experiments/Apple-ARKit-Overview.pdf, accessed August 2019

[62] Li, X.: 'Feature Points on Point-based Surface and Their Applications'. PhD Thesis, The University of Michigan, 2009

[63] 'Placenote ARKit Features', https://placenote.com/docs/unity/tutorials/arkit-integration, accessed September 2019

[64] 'JSON', https://www.json.org, accessed September 2019

Appendices

Table 1. Touch commands and their corresponding actions for each mode [60]

Device Gesture	Main Menu	Map Creation mode	Navigation mode	Map Extending mode
1-Finger Tap	Detect Distance	Detect Distance	Detect Distance	Detect Distance
Swipe Right	**Map Creation**	_	**Map Extending**	_
Swipe Left		Exit to Main Menu	Exit to Main Menu	Exit to Navigation
Swipe Up	_	Save Map	_	Save Extended Map
Swipe Down	Map Load	_	Map Load	_
2-Finger Tap	_	Add / Remove Marker	_	Add / Remove Marker
2-Finger Hold	Used to playback instruction of how to use the functionalities available in the current mode (main menu, map creation, navigation, map extending)			
3-Finger Tap	Switch between ARHitTestResultTypes for real-space obstacle distance detection (Used in testing and debugging)			

Table 2. Distance measurement (Dist) results in meters for EstimatedHorizontalPlane HitTest with 4 Objects

Test	Dist 1	Dist 2	Dist 3	Dist 4
Tape Measure	0.5	1.0	1.5	2.0
Object 1 Test 1	0.44	1.33	1.58	1.83
Object 1 Test 2	0.77	0.60	1.17	2.20
Object 2 Test 1	0.57	0.91	1.28	1.92
Object 2 Test 2	0.53	0.92	1.20	1.01
Object 3 Test 1	0.55	0.86	1.40	1.81
Object 3 Test 2	0.47	0.91	1.44	1.77
Object 4 Test 1	0.56	0.94	1.47	1.89
Object 4 Test 2	0.47	0.93	1.42	1.95

Table 3. Distance measurement (Dist) results in meters for EstimatedVerticalPlane HitTest with 4 Objects

Test	Dist 1	Dist 2	Dist 3	Dist 4
Tape Measure	0.5	1.0	1.5	2.0
Object 1 Test 1	0.48	1.05	1.4	1.83
Object 1 Test 2	0.49	0.98	1.66	1.86
Object 2 Test 1	0	0	0	0
Object 2 Test 2	0	0	0	0
Object 3 Test 1	0	0	0	0
Object 3 Test 2	0	0	0	0
Object 4 Test 1	0	0	0	0
Object 4 Test 2	0	0	0	0

Table 4. Distance measurement (Dist) results in meters for ExistingPlane HitTest with 4 Objects

Test	Dist 1	Dist 2	Dist 3	Dist 4
Tape Measure	0.5	1.0	1.5	2.0
Object 1 Test 1	0.47	0.95	1.58	2.05
Object 1 Test 2	0.49	0.98	1.51	2.01
Object 2 Test 1	0.41	0.99	1.44	1.84
Object 2 Test 2	0.43	0.95	1.47	1.90
Object 3 Test 1	0.48	0.86	1.35	2.01
Object 3 Test 2	0.40	0.80	1.42	1.97
Object 4 Test 1	0.43	0.91	1.42	1.93
Object 4 Test 2	0.47	0.95	1.39	1.87

Table 5. Distance measurement (Dist) results in meters for ExistingPlaneUsingExtent HitTest with 4 Objects

Test	Dist 1	Dist 2	Dist 3	Dist 4
Tape Measure	0.5	1.0	1.5	2.0
Object 1 Test 1	0.47	1.08	1.49	2.04
Object 1 Test 2	0.5	1.01	1.48	1.96
Object 2 Test 1	0.46	0.93	1.42	1.98
Object 2 Test 2	0.49	0.91	1.46	2.03
Object 3 Test 1	0.44	0.96	1.43	1.96
Object 3 Test 2	0.47	0.98	1.47	2.01
Object 4 Test 1	0.46	0.90	1.46	1.94
Object 4 Test 2	0.41	0.86	1.51	2.02

Table 6. Distance measurement (Dist) results in meters for ExistingPlaneUsingGeometry HitTest with 4 Objects

Test	Dist 1	Dist 2	Dist 3	Dist 4
Tape Measure	0.5	1.0	1.5	2.0
Object 1 Test 1	0.49	1.04	1.53	1.78
Object 1 Test 2	0.47	0.95	1.49	1.80
Object 2 Test 1	0.38	0.93	1.48	2.04
Object 2 Test 2	0.35	1.06	1.53	2.01
Object 3 Test 1	0.44	0.96	1.51	1.99
Object 3 Test 2	0.47	0.91	1.47	2.02
Object 4 Test 1	0.45	0.89	1.47	1.92
Object 4 Test 2	0.42	0.94	1.54	1.98

Table 7. Distance measurement (Dist) results in meters for FeaturePoint HitTest with 4 Objects

Test	Dist 1	Dist 2	Dist 3	Dist 4
Tape Measure	0.5	1.0	1.5	2.0
Object 1 Test 1	0.48	0.99	1.45	1.88
Object 1 Test 2	0.52	1.01	1.49	1.84
Object 2 Test 1	0.48	0.87	1.43	1.87
Object 2 Test 2	0.52	0.94	1.48	1.97
Object 3 Test 1	0.53	0.93	1.55	2.05
Object 3 Test 2	0.49	0.85	1.43	1.96
Object 4 Test 1	0.43	0.91	1.44	1.89
Object 4 Test 2	0.45	0.86	1.47	1.91

Table 8. Distance measurement (Dist) results in meters for AllARHitTestResults HitTest with 4 Objects

Test	Dist 1	Dist 2	Dist 3	Dist 4
Tape Measure	0.5	1.0	1.5	2.0
Object 1 Test 1	0.48	1.04	1.47	2.01
Object 1 Test 2	0.52	1.09	1.42	2.02
Object 2 Test 1	0.46	0.98	1.45	1.96
Object 2 Test 2	0.51	1.00	1.48	1.95
Object 3 Test 1	0.44	0.94	1.48	1.92
Object 3 Test 2	0.53	0.97	1.44	1.99
Object 4 Test 1	0.43	0.87	1.52	1.96
Object 4 Test 2	0.47	1.02	1.49	2.00