VTQuestAR: An Augmented Reality Mobile Software Application for Virginia Tech Campus Visitors

Zhennan Yao

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Application

Osman Balci, Chair
Reza Barkhi
Liqing Zhang

December 16, 2020

Blacksburg, Virginia

Keywords and phrases: Augmented reality, Core Data database, image recognition, iPhone / iPad software application, machine learning

VTQuestAR: An Augmented Reality Mobile Software Application for Virginia Tech Campus Visitors

Zhennan Yao

ABSTRACT

The main campus of Virginia Polytechnic Institute and State University (Virginia Tech) has more than 120 buildings. The campus visitors face problems recognizing a building, finding a building, obtaining directions from one building to another, and getting information about a building. The exploratory development research described herein resulted in an iPhone / iPad software application (app) named *VTQuestAR* that provides assistance to the campus visitors by using the Augmented Reality (AR) technology. The Machine Learning (ML) technology is used to recognize a sample of 31 campus buildings in real-time. The *VTQuestAR* app enables the user to have a visual interactive experience with those 31 campus buildings by superimposing building information on top of the building picture shown through the camera. The app also enables the user to get directions from the current location or a building to another building displayed on a 2D map as well as an AR map. The user can perform complex searches on 122 campus buildings by building name, description, abbreviation, category, address, and year built. The app enables the user to take multimedia notes during a campus visit. Our exploratory development research illustrates the feasibility of using AR and ML in providing much more effective assistance to visitors of any organization.

VTQuestAR: An Augmented Reality Mobile Software Application for Virginia Tech Campus Visitors

Zhennan Yao

GENERAL AUDIENCE ABSTRACT

The main campus of Virginia Polytechnic Institute and State University (Virginia Tech) has more than 120 buildings. The campus visitors face problems recognizing a building, finding a building, obtaining directions from one building to another, and getting information about a building. The exploratory development research described herein resulted in an iPhone / iPad software application named *VTQuestAR* that provides assistance to the campus visitors by using the Augmented Reality (AR) and Machine Learning (ML) technologies. Our research illustrates the feasibility of using AR and ML in providing much more effective assistance to visitors of any organization.

ACKNOWLEDGMENTS

First, I would like to thank Dr. Osman Balci. This thesis could not be finished without help from him. He provided insightful advice whenever I had questions about the thesis research and writing. Without the Virginia Tech buildings database that he offered, it would have been impossible to develop the app. I am grateful that I have learned so many things during my studies with him.

I would also like to acknowledge Dr. Reza Barkhi and Dr. Liqing Zhang for serving as committee members. They offered invaluable help during the research.

Finally, I would like to thank my friend Jiaying Liu for her support during my graduate study.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS

2D	Two Dimensional
3D	Three Dimensional
AR	Augmented Reality
ML	Machine Learning
API	Application Programming Interface
UI	User Interface
iOS	Apple's mobile Operating System
IDE	Integrated Development Environment
iPadOS	Apple's mobile Operating System

Chapter 1: Problem Definition and Overview

1.1 Introduction

Campus visiting is an essential part of recruiting students. According to statistics released by Virginia Tech Daily [Minnich 2019], there were over 50,000 visits to the Blacksburg campus in 2019. However, a college catalog or website will not let people know all the things about the school. And it is essential to go to campus and get a feel. Generally, students go to campuses with family members before they decide which school to attend. A popular way to get to know the campus is to schedule a campus tour with campus tour guides, but this service is not always available. If users visit campus without a campus tour guide, they will find it challenging to recognize a building, know the information of a building, and so on. Therefore, a mobile application is vital for Virginia Tech campus visitors to overcome such challenges and possibly increases the chance to attend Virginia Tech. Though the majority of colleges provide campus tour guide services, only a few schools offer mobile applications assisting campus visitors, especially the applications with emerging technologies such as augmented reality (AR) and machine learning (ML). As a result, we developed *VTQuestAR* to explore the feasibility and effectiveness of integrating AR and ML in a mobile app for campus visitors in our exploratory development research.

1.2 Statement of the Problem

The main campus of Virginia Polytechnic Institute and State University (Virginia Tech) has more than 120 buildings. The campus visitors face problems recognizing a building, finding a building, obtaining directions from one building to another, and getting information about a building.

1.3 Statement of the Objectives

The exploratory development research described herein aims to accomplish the following objectives:

1. Explore the feasibility of using AR and ML technologies in providing effective assistance to visitors of any organization by using Virginia Tech campus as an example.

2. Develop an iPhone / iPad software application to illustrate the feasibility of using AR and ML technologies in providing effective assistance to visitors of any organization.

3. Develop a Core ML model to recognize a sample of 31 Virginia Tech campus buildings in real-time.

4. Use AR to enable campus visitors to have a visual interactive experience with those 122 campus buildings by superimposing building information on top of the building picture shown through the camera.

5. Enable campus visitors to get directions from the current location or a building to another building displayed on a 2D map as well as an AR map.

6. Enable campus visitors to perform complex searches on 122 campus buildings by building name, description, abbreviation, category, address, and year built.

7. Enable campus visitors to take multimedia notes during a campus visit.

8. Employ a Core Data database for storage and retrieval of all app data including campus buildings data, multimedia notes, and AR photos.

1.4 Overview of the Thesis

Chapter 2 describes the general technologies we used in the thesis application. Chapter 3 describes the app's Augmented Reality feature, allowing users to take pictures of virtual objects. Chapter 4 describes the Image Recognition feature that enables users to recognize campus buildings using ML and AR. Chapter 5 describes the Directions feature in which users can navigate from the current location or a campus building to another campus building in AR and 2D maps. Chapters 6 describes the Buildings feature that allows users to know campus buildings better in various ways, such as providing complex building search, display buildings in AR, display buildings according to categories. Chapter 7 describes the app's Notes feature, which provides the functionality to take multimedia notes containing images and voice recordings while visiting the campus. Chapter 8 evaluates *VTQuestAR* in several aspects to examine software quality. Last, Chapter 9 presents the conclusion and future work.

1.5 Summary of Contributions

The research contributions can be summarized as follows:

1. Explored the feasibility of using AR and ML technologies in providing effective assistance to visitors of any organization by using Virginia Tech campus as an example.

2. Developed an iPhone / iPad software application to illustrate the feasibility of using AR and ML technologies in providing effective assistance to visitors of any organization.

3. Developed a Core ML model to recognize a sample of 31 Virginia Tech campus buildings in real-time.

4. Used AR to enable campus visitors to have a visual interactive experience with those 122 campus buildings by superimposing building information on top of the building picture shown through the camera.

5. Enabled campus visitors to get directions from the current location or a building to another building displayed on a 2D map as well as an AR map.

6. Enabled campus visitors to perform complex searches on 122 campus buildings by building name, description, abbreviation, category, address, and year built.

7. Enabled campus visitors to take multimedia notes during a campus visit.

8. Employed a Core Data database for storage and retrieval of all app data including campus buildings data, multimedia notes, and AR photos.

Chapter 2: Application Design

This chapter presents the design specification of the iPhone / iPad software application *VTQuestAR*.

2.1 Software Development Cycle

We strictly follow the software life cycle [Balci 2020] as Figure 1 shows, in the design of *VTQuestAR*.

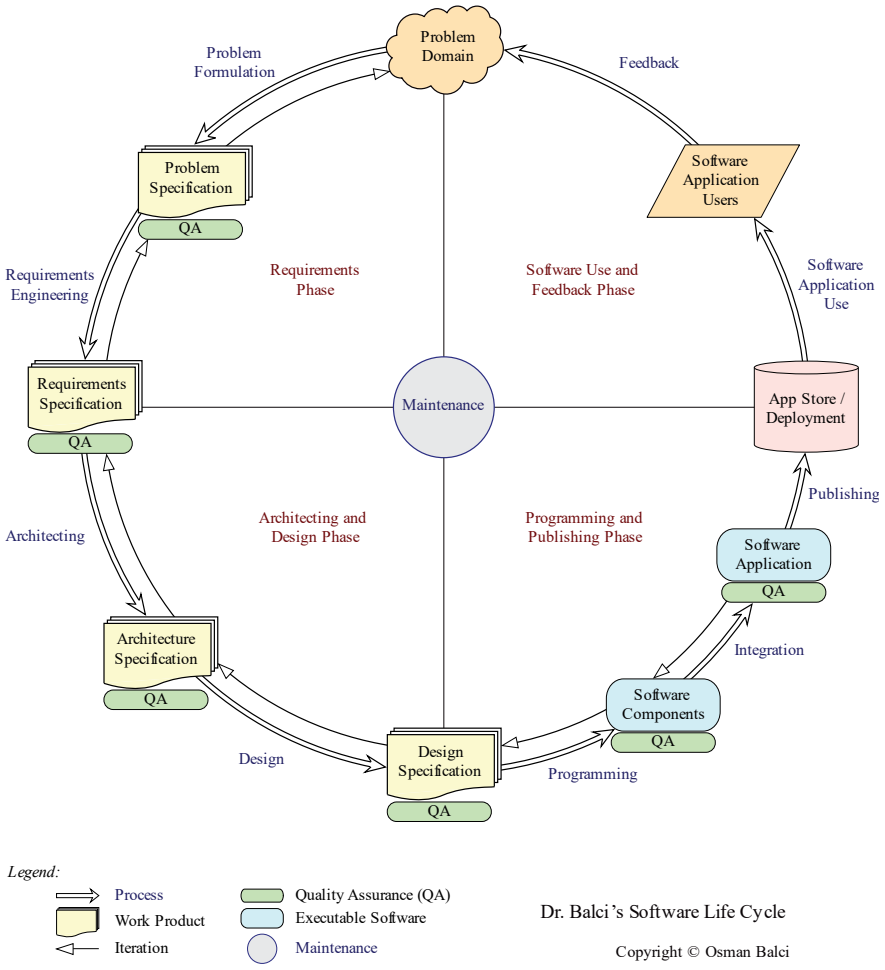

Figure 1: Software Life Cycle [Balci 2020]

2.2 User Interface Design

Figure 2 illustrates the UI prototyping for five tab views. Here, we mainly want to show each tab view's default view to illustrate the initial UI design consideration, so we omit the UI prototyping graphs for sub-views. Detailed prototyping graphs for each tab view will be given in the following chapters.

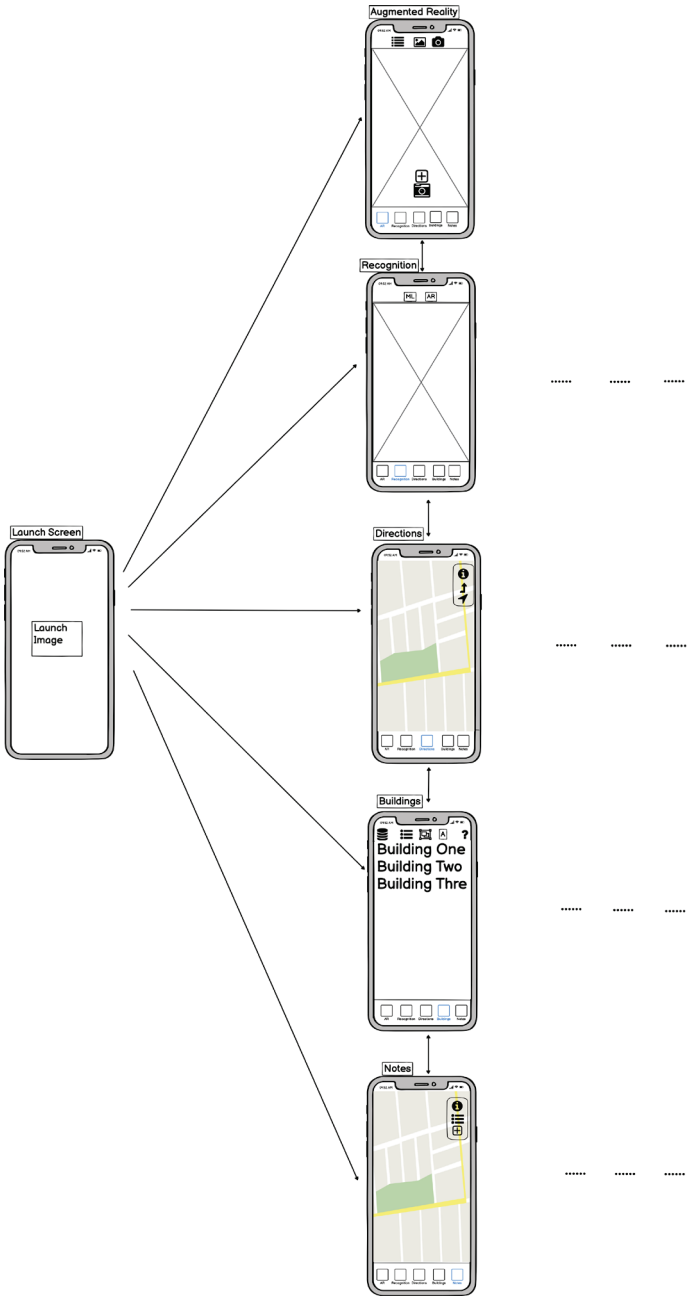

Figure 2: Application Prototyping

Many navigation patterns are existing in iOS and iPadOS development. Hamburger menu or sidebar menu used to be a pervasive navigation pattern. However, it has several drawbacks. First,

the menu and all the menu page contents may be hidden upon the first launch. As a result, it requires users to guess which button triggers the menu bar to show. Second, it does a poor job in terms of efficiency: users usually need to tap on more than two buttons to navigate to the destination scenes. For instance, users first need to tap a sidebar menu button to see the options and then tap another button under the menu to navigate to their destinations.

In 2014, Apple made a fundamental change in thinking on how mobile application navigation should work during the 2014 WWDC Talk: "Designing Intuitive User Experiences" [Apple 2014]. Apple introduced tab bar navigation to replace the sidebar menu navigation. There are several reasons that Apple encourages developers to switch to the new tab bar navigation patterns. First, by using a tab bar navigation pattern, users can easily navigate among views. They do not need to click on a menu button first to see the desired buttons. Besides, it provides an easy-to-follow user interface and decreases the chances of getting confused while switching among views. Therefore, we used tab bar navigation that displays all the buttons for navigation on the top and bottom of the screen in *VTQuestAR,* as shown in Figure 2.

2.3 General Implementation Technologies

The following includes the general implementation technologies of *VTQuestAR.*

2.3.1 Xcode

Xcode is the IDE developed by Apple. Developers use Xcode to build apps and deploy them on Apple platforms, including iPad, iPhone, Apple Watch, and more. It also provides tools to manage the entire mobile application development flow, from creating apps to submitting them to the App Store. Compared to other IDEs, Xcode provides a convenient way to test the app in a simulated environment if a real testing device is not available. Furthermore, developers can use Instrument tools in Xcode to profile or analyze the app, improve performance, and find memory issues such as memory leaks. When developers are coding in the source files, Xcode will scan and point out the syntax errors before compiling. In most cases, Xcode will provide fixes to the errors such that developers can quickly resolve the issues.

In this thesis, we use Xcode 12 to develop *VTQuestAR*.

2.3.2 Swift

Swift is a powerful and intuitive programming language for iOS, macOS, and beyond. Writing code in Swift is interactive, and the syntax is concise and expressive, with the majority of modern features supported. Meanwhile, it was designed for safety. The compiler will throw a compile-time error if the developer wants to make or use a nil object because the safety feature defines that default objects in Swift can never be nil. Besides the safety feature, Swift was built to be fast. By using the high-performance LLVM compiler technology, Swift code is transformed into an optimized native code [Swift.org 2019]. For features, it supports low-level

primitive types including types, flow control, and operators. It also supports object-oriented features such as protocols and generics.

In this thesis, we use Swift as the programming language to develop *VTQuestAR*.

2.3.3 SwiftUI

SwiftUI is a simple but innovative framework to build user interfaces across nearly all Apple platforms. It is a modern way to create beautiful, clean, and dynamic apps faster than ever before by providing views, controls, and layout structures for declaring an app's user interfaces. Before the release of SwiftUI, options to create user interfaces of an app are the interface builder and storyboard. By using the storyboard or interface builder, developers have to design user interfaces in separate files rather than in code files. Besides, developers need to take constraints such as different screen sizes into account while building user interfaces. Consequently, if an application is complicated, the storyboards will look messy and make debugging challenging.

In addition, the interface builder and storyboard do not know much about the actual code, and the code does not know much about the storyboard or interface builder either. To allow the code and interface builder or storyboard to communicate, developers need to manually set up a relationship by controlling and dragging from the storyboard to the swift file to connect user interface components with code files. However, this is not an efficient way and will possibly break the consistency. If developers delete an outlet that connects the bridge of interface builder or storyboards to the swift file, the application might still compile. However, it will throw errors when the application runs on user devices or simulators, making the debugging process troublesome.

By contrast, SwiftUI makes it easy to build a user interface with minimum lines of code. SwiftUI provides the live preview, allowing developers to check the user interface of the current view in real-time without compiling the whole application. Also, an app will not crash because of the IBOutlet connection issues anymore. With SwiftUI, developers do not need to connect user interface components with code files manually. Instead, there are elements such as HStack, VStack to create efficient user interfaces without any Auto Layout or UI constraint problems.

The minimum system requirement to use the SwiftUI framework is iOS 13 and above. Therefore, we use iOS 14 to develop, test, and deploy *VTQuestAR*.

Chapter 3: Augmented Reality

3.1 Literature Review

Augmented reality is an enhanced version of reality created by using technology to overlay digital information on a scene of something being viewed through a device such as a smartphone camera [Merriam-Webster 2020]. It enhances the real-world environment by utilizing the camera of a device and delivers an integrated virtual experience. Over the past few years, the demand for AR supported applications has considerably increased, and more and more developers start to investigate possibilities of embedding AR in their apps.

Today, there are two primary augmented reality development kits, namely ARCore and ARKit. ARCore was originally launched by Google in 2018 and was designed to be compatible with most Android phones. Likewise, Apple has designed its AR development kit named ARKit, which helps design AR applications on Apple devices. Two giants investing in AR is an obvious sign that the trend of AR app in the smartphone market is surging. Moreover, the increasing number of devices compatible with AR (Figure 3) also proves this point [Mike Boland 2017].

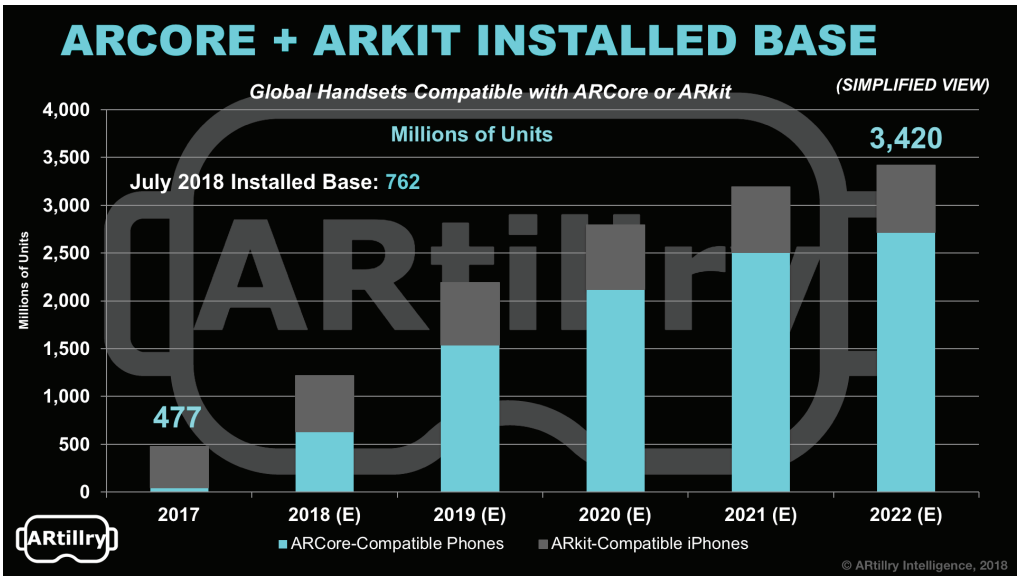

Figure 3: Global Handsets Compatible with ARCore and ARKit [Mike Boland 2017]

Pokemon Go, a mobile game that went viral a few years ago, used ARKit on Apple platforms and enhances the experience of catching Pokemon to the next level. Ikea, a company that sells furniture and home furnishings, used ARKit to build an app that allows customers to place virtual furniture in an AR scene through the phone cameras to mimic the furniture layout after the furniture is placed into the living room. In general, ARKit is ubiquitous on AR-compatible applications.

ARKit, a mobile AR platform for developing augmented reality apps on Apple devices, is a high-level API providing a simple interface with a robust set of features. The framework is designed by Apple that integrates Apple device camera and motion features to produce immersive AR experiences in applications or games. It combines device motion tracking, camera scene capture, advanced scene processing, and display conveniences to simplify the task of building an AR experience [Apple 2020a].

ARKit can be divided into three distinct layers. The first layer is tracking, which is the core feature of ARKit that enables the tracking of devices in real-time [Apple 2020a]. For example, world tracking makes it possible to get users' relative positions in the physical world without the need to use any external setup and pre-existing knowledge. Apple uses visual inertial odometry, a technology that combines camera images and motion data from the device to get an exhaustive view of device location and orientation. Based on the information collected by the tracking layer, another layer known as scene understanding will determine attributes of the environment around users [Apple 2020a]. Scene understanding provides an interaction with the real-world topology that enables users to place virtual objects in the physical world. Light estimation is a sub-feature of scene understanding that is used to light virtual objects in an AR scene to match the real-world lighting. The last layer is rendering. After tracking is detected and scene understanding is finished, developers have all the information ready for rendering. ARKit can then render AR materials or objects in the physical world such that users will see the AR-enhanced scene on the phone screens through phone cameras.

The essential layer that developers need to take care of is the rendering. There are four distinct types of content rendering technologies in ARKit: Metal, SpriteKit, SceneKit, RealityKit.

Metal is a low-level, low-overhead hardware-accelerated 3D graphic API designed by Apple. To utilize the full power of Metal, developers need to write all the base code without any boilerplate code. In comparison to other rendering technologies, Metal gives developers full control, enabling developers to achieve more complicated functionalities. However, Metal requires more lines of code. By contrast, SpriteKit, RealityKit, and SceneKit are higher-level frameworks. Both RealityKit and SceneKit are used to render 3D graphics in 3D games and 3D AR applications. SpriteKit, however, is used to render 2D graphics. All three higher-level frameworks are based on the Metal API.

SceneKit is a technology we have used to render AR experiences. It is a framework for creating 3D games or add 3D content to apps using high-level scene descriptions. SceneKit provides a clean and straightforward interface for developers compared to Metal API. To deliver 3D content with SceneKit, developers need to create a SCNScene first, which contains a hierarchy of nodes or attributes representing the visual elements [Apple 2020c]. Then we can add child nodes that are instances of SCNNode under the root node to assign attributes such as shape, color, and material of virtual objects for rendering.

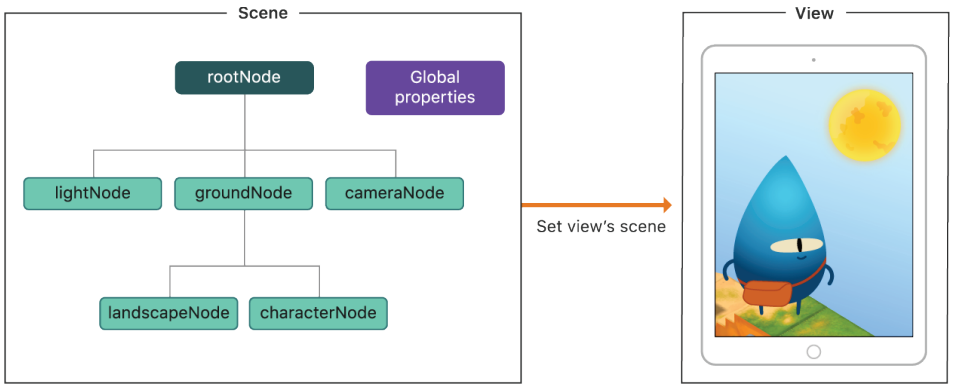

Figure 4: Scene Hierarchy for SceneKit Rendering [Apple 2020c]

Lately, Apple released a terrific technology named LiDAR scanner. According to Apple's introduction, the scanner measures the distance to surrounding objects up to 5 meters away, operating at the photon level at nano-second speeds [Apple 2020b]. LiDAR will make AR applications much faster and more accurate. By integrating LiDAR into ARKit, instant virtual object placement in the physical world will be enabled, meaning that developers do not need to track or scan user surroundings before the application loads because the scanner does all the scanning in nano-seconds. Even though only the latest iPad Pro and iPhone 12 are equipped with LiDAR and developers still do not have full control of the scanner, developers can expect to create many exciting features once Apple grants full access to the scanner in the future.

In *VTQuestAR*, we use ARKit to deliver AR experiences and SceneKit and SpriteKit for rendering virtual objects.

3.2 Design Specification

3.2.1 User Interface Design

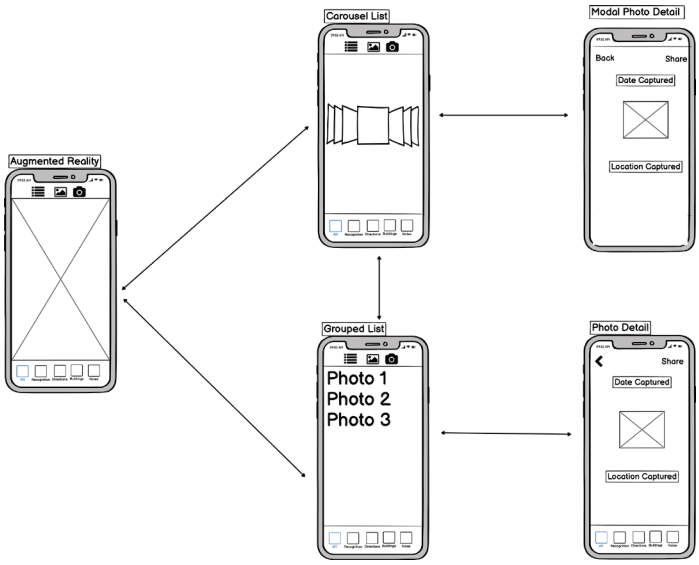

Figure 5: Prototyping of Augmented Reality

In the Augmented Reality scene, users can take photos with virtual objects in the AR scene and effectively check the saved photos. We stick to the tab bar navigation pattern to build the UI, as shown in Figure 5. All buttons in the upper navigation bar are grouped together to improve detectability. We also use two lists with different styles to display the same photo content because the image size in a standard list is restricted by screen size and looks tiny on small screens. Thus, we designed a custom carousel list to make the images larger in size, which helps meet the needs of different user groups. By doing so, users can select their preferred style for viewing the saved photos.

3.2.2 Database Design

In this feature, photos will be stored in the database whenever users tap the capture button. The Core Data database we designed has four attributes: the latitude and longitude of the location and the photo data itself, as shown in Figure 6.

Figure 6: Database Design of Augmented Reality

3.3 Implementation

3.3.1 AR Camera

In the AR Camera scene, users can record beautiful moments of campus visits. However, we do not want to create a camera application that resembles regular camera apps. Instead, we would like to enhance photography experiences by adding AR features. The main objective of AR Camera is to allow users to take photos with virtual objects when they visit the Virginia Tech campus.

The initial scene is shown in Figure 7. The AR Camera button is filled with blue color because the selected button will be marked blue in *VTQuestAR*'s design, and AR Camera is chosen by default in Augmented Reality. With this scene, users can see their surroundings on the phone screen.

Figure 7: AR Camera

Similar to the UI prototyping we introduced, we have designed two stacked buttons: an add button and a capture button. Users can click on the add button to select the virtual object they would like to add to the AR scene. An overlay view for choosing different virtual objects will be provided, as shown in Figure 8.

Figure 8: Add an AR Object

If an item is selected, the object will be added in front of the user in the AR scene. Objects that are successfully added to the scene will maintain their same real-world positions, even if the user walks around or changes the phone orientation.

There are four different types of virtual objects: VT hokie, VT logo, VT truck, and VT helmet (Figure 9). Since we have four different 3D objects, we have to differentiate them before applying certain operations. For instance, if we want to rotate an item, we have to keep other objects unchanged, and that requires the differentiation process. CategoryBitMask is the way we choose to classify objects. The mask is a property of a node that helps define the category to which the node belongs. We take advantage of the mask by assigning each node in the SCNScene with a bit corresponds to a predefined category. When we use SceneKit to render an AR scene, we can compare the CategoryBitMask of each node and operate only on nodes of the desired category without touching other category groups.

Figure 9: AR Objects

When there is at least one virtual object in the scene, we can do the following operations: rotation, zoom in and out, move, selection, and deletion. To enable these operations, we initialize gesture recognizers shown in Figure 10 that help recognize different gestures. Then our app can act accordingly when one of the gestures gets detected.

```
let longPressRecognizer = UILongPressGestureRecognizer(target: self, action: #selector(longPressed))
                        self.sceneView.addGestureRecognizer(longPressRecognizer)

    let pinchRecognizer = UIPinchGestureRecognizer(target: self, action: #selector(pinchPressed))
                         self.sceneView.addGestureRecognizer(pinchRecognizer)

 let rotateRecognizer = UIRotationGestureRecognizer(target: self, action: #selector(rotatePressed))
                        self.sceneView.addGestureRecognizer(rotateRecognizer)

    let deleteRecognizer = UITapGestureRecognizer(target: self, action: #selector(deletePressed))
                             deleteRecognizer.numberOfTapsRequired = 2
                        self.sceneView.addGestureRecognizer(deleteRecognizer)

      let selectGesture = UITapGestureRecognizer(target: self, action: #selector(selectPressed))
                               selectGesture.numberOfTapsRequired = 1
```

Figure 10: Gesture Recognizers Initialization

3.3.2 Selection

In order to apply an action on a virtual object, it is required to select an object first with a single tap. The selected object will be marked green and revert to its original color immediately, indicating that the current item has been selected (Figure 11). Once the selection is complete, the user can rotate, zoom in and out any existing object. Without the selection operation, the user cannot do further operations because the selection operation can assign CategoryBitMask to an object. So, we can differentiate and track a virtual object's state only when the object has its own CategoryBitMask value. If there is no item selected, the CategoryBitMask of a virtual object is unknown to us, and we cannot be sure that we are modifying the desired object.

Figure 11: Selection Operation

3.3.3 Deletion

If users would like to delete an object, they can tap the object they want to delete twice. After tapping the object twice, a deletion alert message will be popped up. Users can either tap the Cancel button to dismiss the deletion request or select the Delete button to delete the object from the AR scene, as Figure 12 shows.

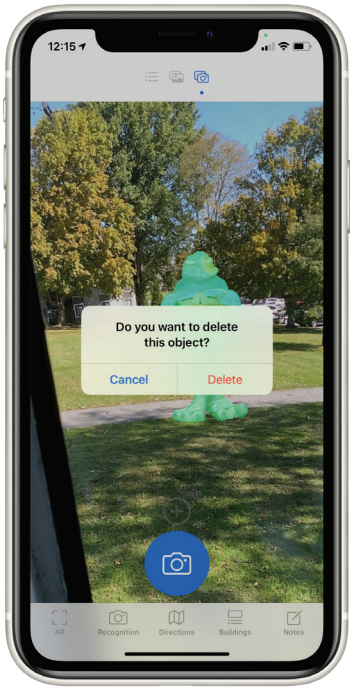

Figure 12: Deletion Operation

3.3.4 Rotation

Before rotating an object, the user must first proceed with a selection. Selection stores the CategoryBitMask of the selected object, and then we can ensure that only the selected item will be rotated. To rotate objects, we take advantage of their Euler angles and modify them according to the direction of the rotation gesture to rotate it clockwise or counterclockwise along its y-axis in three-dimensional space. Therefore, the object which gets rotated is always perpendicular to the ground.

3.3.5 Zoom

Similarly, the prerequisite of a zoom operation is the selection operation. After the object is selected, the user can use a pinch gesture to zoom in and out a virtual object. With categoryBitmask, we can filter the objects in the current AR scene and only change the state of the selected object. Afterward, the scale of the object will be altered to achieve a zoom in and out effect.

3.3.6 Move

A long-press gesture is the first required step to move an object. The user only needs to press and hold an object, making sure not to lift the finger, and then move the finger to move the object in 3D space. As the state of gesture changes, i.e., the user drags the object to a different position on the screen, we use the moveBy function in SpriteKit and the unprojectPoint function in SceneKit to convert the position we touched on the 2D phone screen to real-world 3D coordinates. This helps move the object's position in 3D space (Figure 12). Once the user's finger is lifted from the screen, the move operation ends, and the object position changes.

```
let worldTouchPosition = sceneView.unprojectPoint(SCNVector3(touch.x, touch.y, startingZCord!))

let action = SCNAction.moveBy(x: CGFloat(worldTouchPosition.x - lastMovedLocation!.x), y: CGFloat(worldTouchPosition.y
- lastMovedLocation!.y), z: CGFloat(worldTouchPosition.z - lastMovedLocation!.z), duration: 0.0)
```

Figure 13: Move a Virtual Object in 3D Space

3.3.7 Carousel Image List

When the user saves AR photos, there are two different scenes to display them. One of them is the Carousel Image List scene. In this custom scene, we allow the user to scroll horizontally to see all the photos. Clicking on any of the photos redirects the user to the photo's modal detail view. The detail page includes the image itself, as well as the location and date at which the photo was taken (Figure 14). On the detailed page, the user can check the building location on the map. If the user wants to share a photo on their social media platforms, they may click on the Share button.

Also, the user can long-press a photo to delete it. Any changes will also be reflected in the Standard Image List scene since the Carousel Image List scene demonstrates the same content from the same database model, albeit with a different list style.

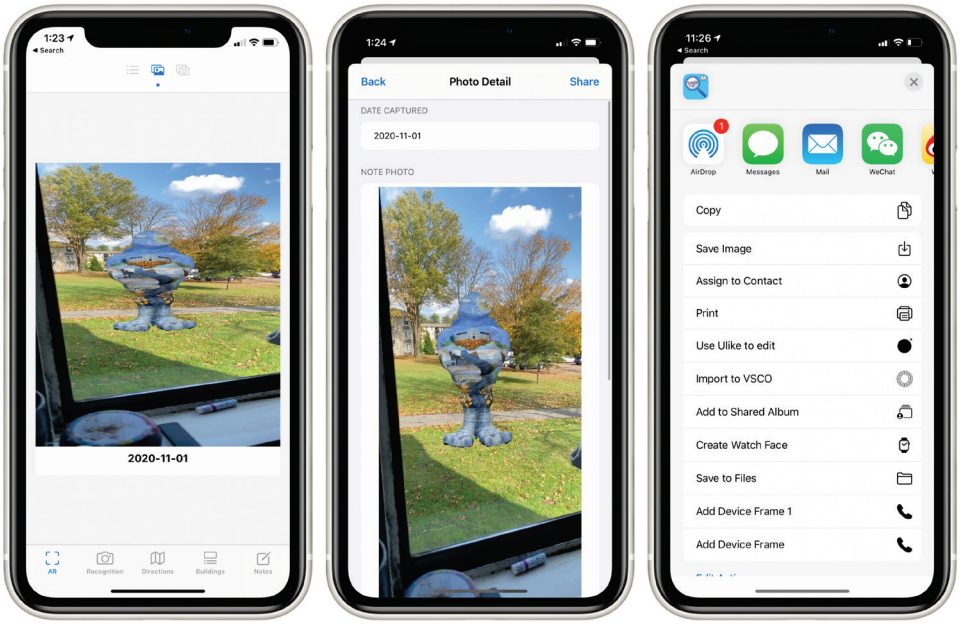

Figure 14: Carousel Image List

3.3.8 Standard Image List

This view also displays the same content as the Carousel Image List, and only the list style is different because the items are stacked vertically, as Figure 15 shows. A thumbnail with the image's date and the image’s location is arranged in a single list item in this view. The user can scroll any image item from right to left to delete it. Likewise, when a list item is tapped, a detailed view of the image is displayed to the user.

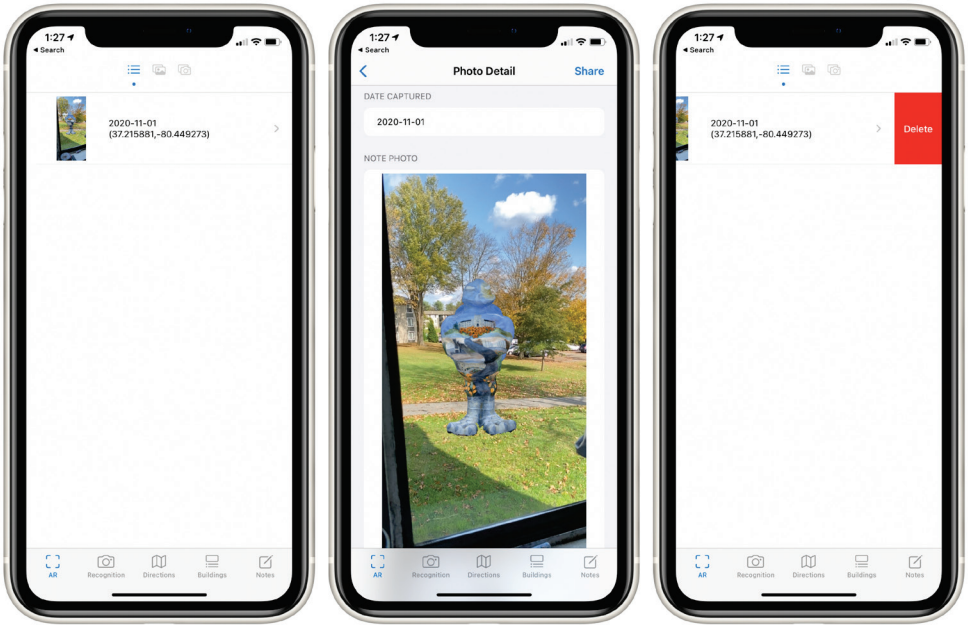

Figure 15: Standard Image List

Chapter 4: Image Recognition

4.1 Literature Review

4.1.1 ML Recognition Literature Review

Image recognition refers to the technology that enables the recognition of places, signs, buildings and other objects in images. It is usually built based on computer vision, an interdisciplinary scientific field that deals with how computers understand and interpret digital images or videos. By using images or videos captured from cameras, along with deep learning or machine learning models, we can train machines to classify objects accurately. For example, human brains can quickly identify a dog or a car since our brains have grown up learning how to distinguish between a dog or a car instinctively. Similarly, we can train machines through machine learning or deep learning algorithms to build an image classifier model to recognize images.

Building a successful image recognizer through machine learning typically requires three essential steps: preprocessing, feature extraction, and learning algorithms for recognition [A. Neetu 2015], as described in Figure 16.

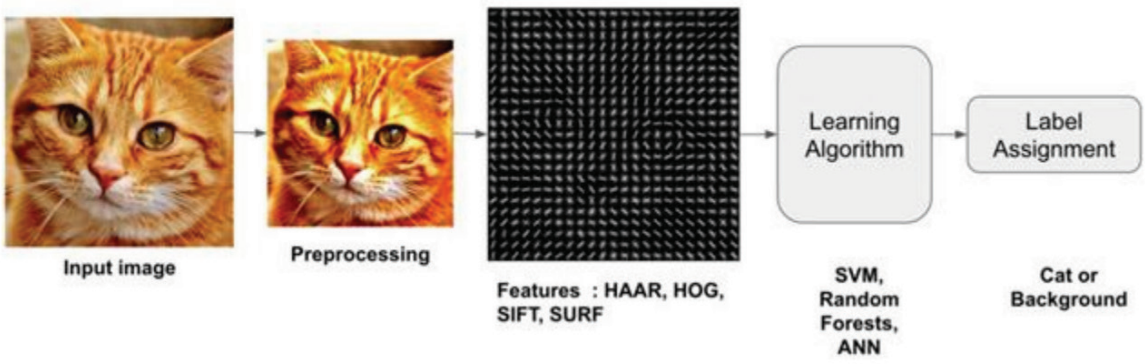

Figure 16: Image Classification Process [A. Neetu 2015]

.

The first step is to preprocess input images with operations such as crop, resize, and change brightness or contrast. In this process, we make sure that the training data is well balanced to get better results. For example, some of the input images have different image sizes. If we do not preprocess them so that they have the same size, it will result in lower accuracy of the final image classifier.

Next, we need to extract the features of the preprocessed input images. Usually, there is redundancy in the classification information of some input images. So, we need to extract only the data or features that can effectively represent the object. Suppose an image classifier is trying to identify an animal, then we need to extract critical features such as the animal's appearance in images rather than the background pixels in images. In this step, standard features such as the histogram of oriented gradients (HOG) shown in Figure 16 will be extracted for the final step.

Once the features have been successfully extracted, learning algorithms will be applied to help the model learn the extracted features. When the image classifier model finishes its training process, it can make predictions in a similar pattern: preprocess the image that needs to be predicted or classified, extract its features, compare such features with the existing features and corresponding labels that have already been learned by the model, and finally assign a label to predict or classify the image. Even though there are many libraries and frameworks available, such as TensorFlow, that can be used to build image classifiers or other ML models, it is much more convenient to use the Core ML framework to create an iOS / iPadOS application.

Core ML is a framework developed by Apple to integrate ML models into apps on Apple platforms [Apple 2020d]. Each model results from applying an ML algorithm to a set of training data. Then, we can use the complete model to make predictions about new input data. In Core ML, there are two ways to create a Core ML model. The first way is to use the Create ML tool to train a Core ML model. The other way is to use other ML libraries and use Core ML tools to transform the models with other formats into Core ML models so that we can use them for Swift programming. We can effortlessly use a Core ML model by dragging it into Xcode's project navigator.

Create ML allows developers to quickly train and build Core ML models on Macs compared to methods that convert models to the Core ML format. Since Apple has lots of experience in training sophisticated ML models, such as the ML model for Siri, they can apply transfer learning on top of their pre-trained complex models to efficiently train similar models. Figure 17 illustrates how transfer learning takes advantage of prior models and quickly build new models based on them.

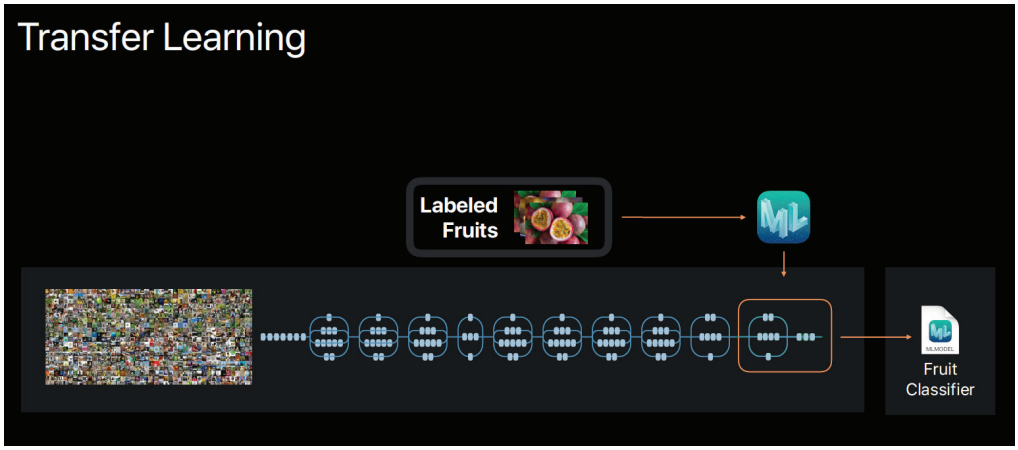

Figure 17: Transfer Learning in Create ML [Apple 2018]

By augmenting the existing model and retraining the last few layers of the new data, we no longer need an extensive image set to train our custom model. Instead, only a few hundred images are sufficient for the image classifier to generate a satisfactory model. As a result, building classifier models in Create ML reduces training time and reduces the final model size from megabytes to kilobytes [Apple 2018]. Because of this, the models built in Create ML are

usually small but reliable. In addition, by providing a simple and straightforward user interface, Create ML lowers the barriers for beginners to design machine learning models. In this thesis, we decided to use Create ML as the technology to build our image classifier to identify campus buildings.

Core ML model prepares a classifier, but we still need to combine it with Vision API to feed the model with screenshot images and make predictions in real-time. Vision is a framework for applying computer vision algorithms to perform various tasks on input images and videos. An image classifier predicts the best result if input images are in a fixed aspect ratio. However, the devices have different screen sizes, resulting in an image classifier that may receive different aspect ratios of the input images. Therefore, we use Vision to scale, crop, and rotate the input image for the classifier.

4.1.2 AR Recognition Literature Review

In the AR Recognition feature, SpriteKit is the rendering technology framework we use in conjunction with ARKit for rendering 2D graphics in AR. In order to establish a correspondence between real and virtual space, ARKit uses a technique called visual inertial odometry [Apple 2020e]. This technique takes advantage of the Apple device's motion sensing hardware to analyze the scene visible to its camera. By doing so, ARKit creates a virtual 3D coordinate system of the world that can be seen when users change the orientation of the phone's camera. Virtual objects can be rendered on this coordinate system, and objects will overlay the live camera view to create AR experiences [Apple 2020e].

ARKit requires an ARSession to render AR experiences to users. An ARSession is the main object to control AR experiences and illustrate how the coordinates system is defined in ARKit. It coordinates the major processes, including reading data from devices' motion hardware, controlling the device's camera, and performing image analysis [Apple 2020f].

In an ARSession, ARConfiguration determines how ARKit tracks the position and movement of a device relative to the real world. For instance, one of the most common ARConfiguration types is named ARWorldTrackingConfiguration, which helps monitor the position and orientation of the device and enhances the AR environment in front of the user.

Besides, we need to assign each ARConfiguration a WorldAlignment type, which has three choices and determines how ARKit builds the scene coordinate system based on real-world device motion and position. In this thesis, we find that gravityAndHeading best suits our purpose. The gravityAndHeading tells the ARSession to make the y-axis of the coordinate system parallel to gravity, with its x- and z-axes pointing in the direction of the compass. Its origin is the initial position of the device [Apple 2020g], as shown in Figure 18. By using gravityAndHeading, we can assign AR objects to 3D space with latitude and longitude based on the world coordinate system, instead of rendering the AR experience based on the device position alone without knowing four cardinal directions.

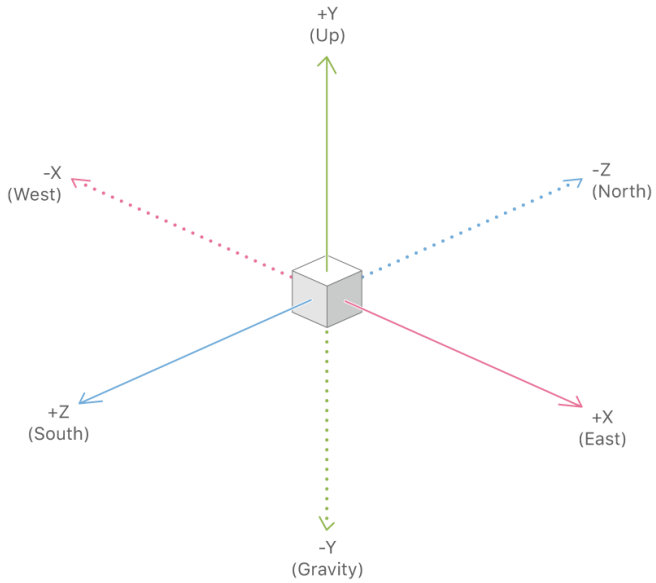

Figure 18: GravityAndHeading Configuration [Apple 2020g]

Also, we still need an ARAnchor that contains information about the position and orientation of virtual objects to be added to the ARSession. We can then specify materials such as images or colors to visualize the virtual objects since the default anchor object's appearance is transparent.

4.2 Design Specification

4.2.1 User Interface Design

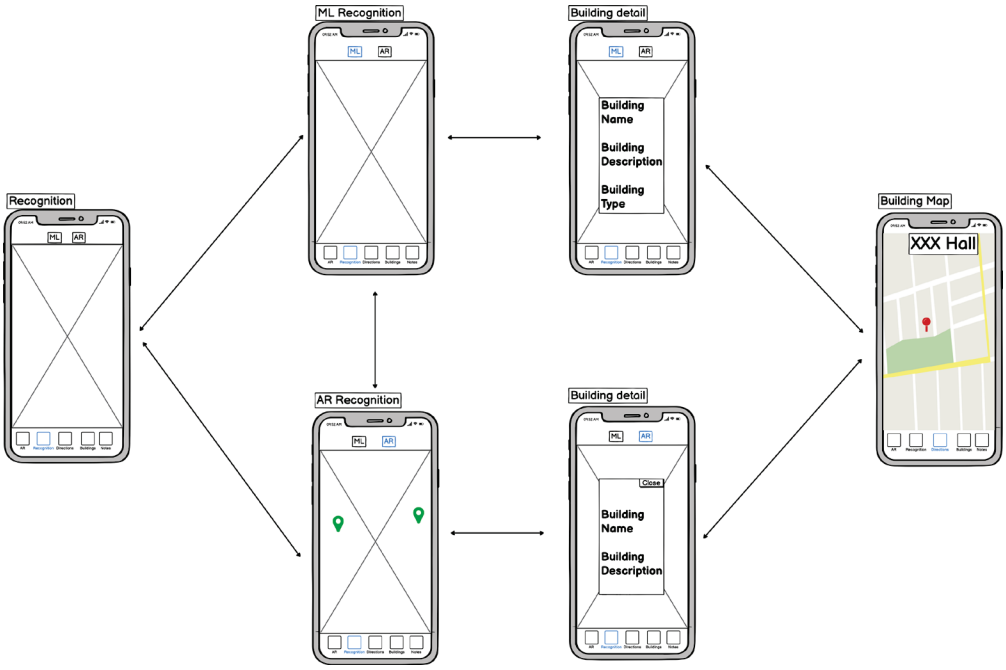

Figure 19: Prototyping of Image Recognition

The prototyping of Image Recognition is shown in Figure 19. Users can easily switch between ML and AR recognition without any effort. Moreover, we use an overlay view to display the building information page instead of a full-screen view because we want users to be aware of their surroundings when using the Image Recognition functionality.

4.3 Implementation

Campus visitors encounter problems, such as wanting to know the name of a building but not knowing what to do. Usually, they will choose to check Google Maps or Maps. However, campus buildings are very close to each other, and it is sometimes difficult to find building information by using Google Maps or Maps. Besides, some people do not have a good sense of direction, especially on a two-dimensional map. Even if the user finds the building, most map applications do not have detailed building information, such as the building's history. Therefore, we created this feature to help users identify buildings and check building information with ease.

4.3.1 ML Recognition

The ML Recognition feature was developed with the aid of ML to create an image classifier capable of recognizing campus buildings. If the user uses ML Recognition outside the campus area, *VTQuestAR* will not run the image classifier model but will display a "Not on Campus" error message (Figure 20). Here, we have used Google Map API to check whether the user is within the campus area by defining a polygon containing the Virginia Tech campus and comparing it to the user's current location.

Figure 20: Not on Campus Error Message

If the user is on campus, but the camera is pointing at an object other than buildings, the application will display the error message: "Not a Campus Building," as Figure 21 shows.

Figure 21: Not a Campus Building Error Message

Suppose the user is on campus and points the phone camera at one of the 31 buildings that support ML recognition. In that case, the ML Recognition scene will display an overlay that shows the details of the building being pointed at. The overlay contains information such as the building name, building photo, building name abbreviation, building category, year of construction, map type to view the building location on a map, building description, and building address (Figure 22). Since the image classifier runs in real-time, a user can thereby identify the building by merely pointing the rear camera on the phone at any building supported by the classifier.

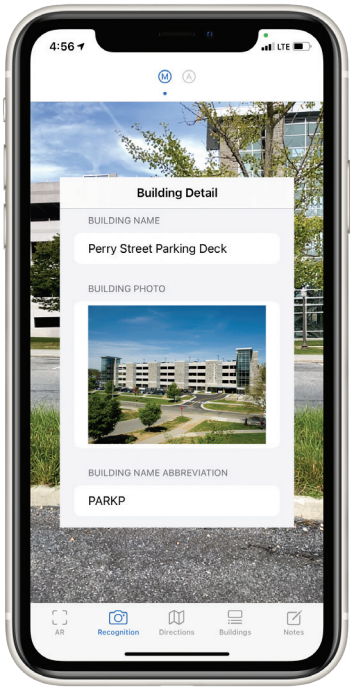

Figure 22: Building Overlay Detail Page for ML Recognition

The building classifier will run and make predictions in real-time by using current screenshot images as input. Once the classifier detects that the current frame image contains a campus building, *VTQuestAR* prepares information about the building and displays it in the overlay. If the user changes the phone's orientation, the overlay view may disappear because the current frame could have objects different from the building, and the classifier will make another prediction. If the user leaves the current ML Recognition scene, the classifier will stop running to reduce energy consumption.

In *VTQuestAR*, only 31 buildings can be classified by the model. This is because some of the buildings look very similar. If we train the model to identify all the campus buildings, the recognition accuracy will be reduced unless we train the model with more images to adapt to different conditions, such as light, weather. Nevertheless, this requires more time, so we have decided to implement it in future work.

4.3.2 AR Recognition

Even though we filtered the buildings so that the model identified only 31 buildings for the sake of accuracy, the classifier is still expected to make erroneous predictions due to state of the art. Reasons for such misidentification include, but are not limited to, inappropriate scanning angles of buildings, different weather, and lighting conditions. Therefore, we would like to use another feature named AR Recognition as a complement to ML Recognition in order to obtain more accurate building identification results.

AR Recognition collects the user's geographic location and building coordinates to help identify buildings through AR. In this scene, buildings within 100 meters from the user are given a virtual green pin in the AR scene. If the user wants to know more about a building, they can simply click on the green pin belonging to the building to be identified in the AR scene to see the building information overlay, as Figure 23 displays.

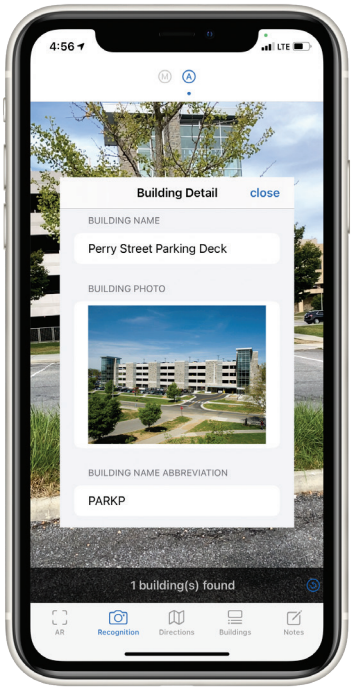

Figure 23: Building Overlay Detail Page for AR Recognition

To add a virtual green pin to the AR scene, we filter all campus buildings by the user's current location and only use buildings within 100 meters of the user for rendering. Then, we calculate the distance and bearing between the user and the filtered buildings. In navigation, a bearing is an angle between an object and another object relative to the true north. For our purposes, we need to calculate the bearing between the user and the buildings.

We store the distance, orientation, and height information for each building in a transformation matrix, then apply the transformation matrix to an ARAnchor and add such an ARAnchor to the ARSession. In this way, we can place a green pin icon on the ARAnchor with the coordinates of the building it represents. We repeat the same process for all nearby buildings to add clickable green pins to the AR scene.

The user can click the refresh button shown in Figure 24 while walking around the campus to reload the green pins into the nearby buildings. By doing so, the user will be able to look up the information of all campus buildings for a campus visit.

Figure 24: AR Recognition

By offering both AR Recognition and ML Recognition, visitors can identify all campus buildings and look up the information they need while visiting.

Chapter 5: Directions

5.1 Literature Review

We utilized three technologies in the Directions feature: Core Location Framework, Mapbox API, as well as the ARCL open-source project [ProjectDent 2020].

Core Location is a framework for obtaining the geographic location and orientation of a device. The framework collects location data using all available components on the device, such as Wi-Fi, GPS, Bluetooth, magnetometer, barometer, and cellular hardware [Apple 2020h]. To develop location-based applications, developers need to enable core location services on the device using the CLLocationManager class. In order to embed the location service, a location authorization request needs to be present to the user first. Once the permission is granted, developers can use location data to build location features and map applications.

However, using CLLocationManager alone is not enough. Developers often need the help of other technologies, such as MapKit, to perform complex map functions. MapKit is a framework that helps display coordinates on a map, call out points of interest, and determine placemark information [Apple 2020i]. In MapKit, MKDirections can compute directions with travel-time information. Users will be given the route and approximate travel time from the origin to the destination. However, a route without real-time navigation does not allow users to get directions quickly and efficiently. Hence, we want to design a map containing both the route and the real-time instructions to help the user get directions on the Virginia Tech campus.

To make *VTQuestAR* support real-time route updates, we switched to Mapbox, an open-source mapping platform for custom designed maps. Mapbox Navigation SDK provides professionally designed map styles, different transport type directions, and traffic avoidance and rerouting options. Hence, we use Mapbox in our app to implement the 2D real-time direction feature with turn-by-turn instructions.

Besides, we also wanted to develop an AR direction feature to address the needs of visitors better. We have used an open-source library called ARCL, which combines the high accuracy of AR and the scale of GPS data. Here, AR stands for ARKit, and CL is the Core Location. This library utilizes the user's locations, routes generated by MapKit, and ARKit to render a virtual path in the AR scene. We added this feature to the application because sometimes visitors are confused by a 2D map route due to a poor sense of direction or a poor GPS signal. In contrast, the AR map is straightforward: the user just needs to follow the route in the AR scene until they reach their destination, regardless of east, west, north, south, or south.

5.2 Design Specification

5.2.1 UI Design

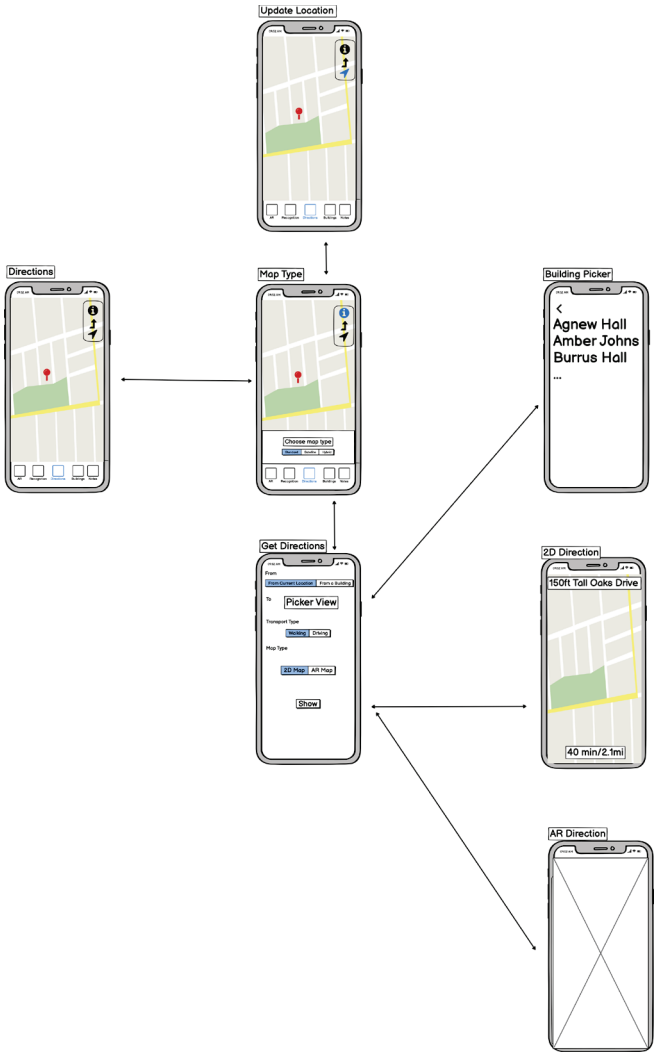

Figure 25: Prototyping of Directions

The prototyping of Directions is shown in Figure 23. In the Get Directions view, instead of providing a search bar, we use a picker view to allow the user to select the destination of the navigation. Since visitors are usually visiting the school for the first time and are unfamiliar with the school buildings, our design choice helps reduce the frequency of typos.

5.3 Implementation

We have designed three sub-features under the Directions feature. After brainstorming in the early design phase, we decided to implement three functionalities for campus visitors, including change map type, get directions, and monitor the user's real-time position on the map.

5.3.1 Default Map View

The default view of Directions is shown in Figure 26. In the beginning, the user can see their current location on a standard style map with three vertically stacked buttons in the upper right corner representing three different functions.

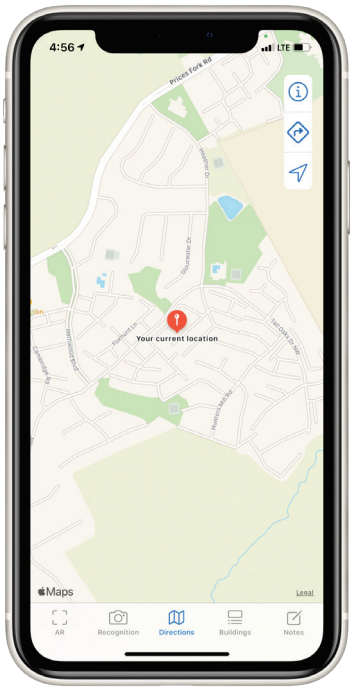

Figure 26: Directions

5.3.2 Map Type Selection

When the user taps on the information icon, our app will present a custom view. The view allows the user to select different map types, including the standard map, satellite map, and hybrid map. When the selection is complete, the map view will be updated with the user's selected type. After that, the user can tap on the horizontal bar in gray or drag it down manually to dismiss the custom map type selection view.

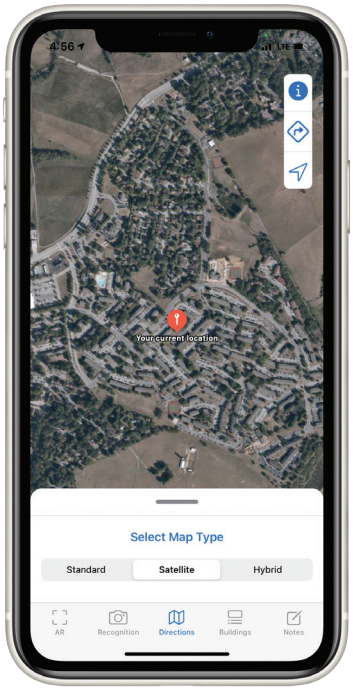

Figure 27: Map Type Selection

5.3.3 Location Tracking

The user can see their location on the map, but by default, the user's location is not updated in real-time. However, the user can select the button with the paper plane icon to enable the real-time location tracking feature, as shown in Figure 28. By tapping it again, the user will stop the real-time location tracking service such that the user’s location on the map will stop updating.

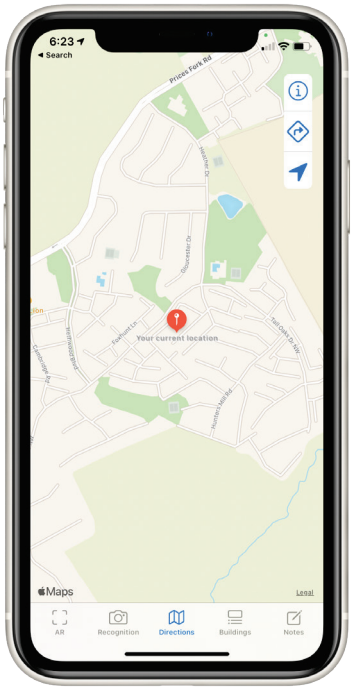

Figure 28: Location Tracking

5.3.4 Get Directions

When users are on campus, they may want to go from one building to another. Hence, we designed the Get Directions scene. By tapping the right turn arrow icon, users will be navigated to the Get Directions scene (Figure 29).

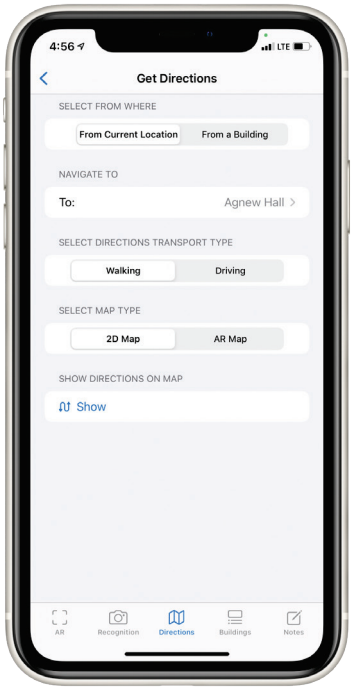

Figure 29: Get Directions

Users can select the origin and destination for getting directions. They can select their current location or a campus building as the starting point and then select any campus building as the destination. Besides, they can choose the type of transportation: walking or driving. Finally, they can choose the type of map: 2D map or AR map. Once the selection is complete, the user can click the Show button to see the route.

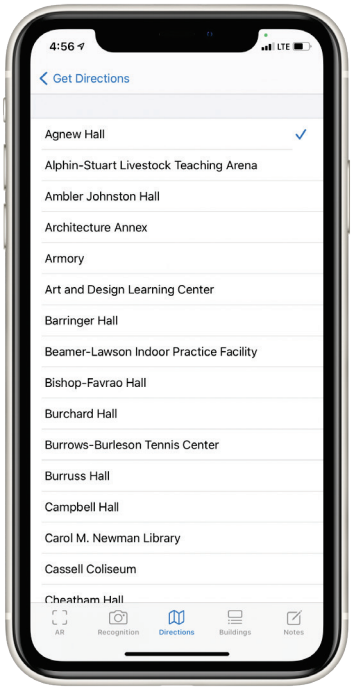

Figure 30: Building Selection Picker View

The UI design of the 2D map and AR map is shown in Figure 31. For the 2D map, the user can follow the real-time instructions to reach the destination. For the AR map, the user needs to follow the virtual blue line in the AR scene to navigate. The blue line starts at the starting point and ends at the destination, so the user will reach the destination if the blue line ends.

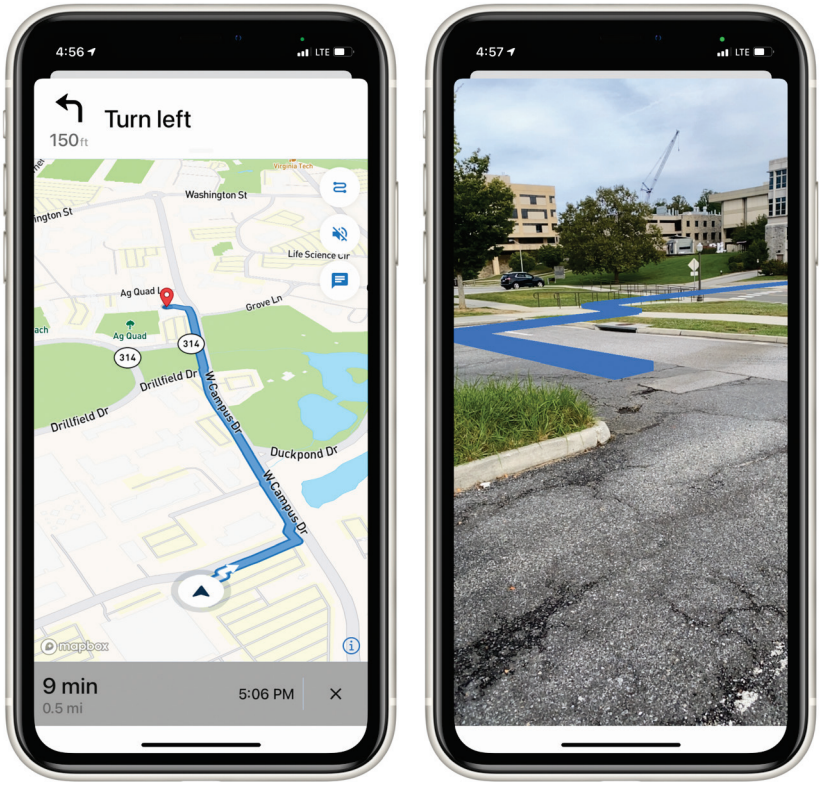

Figure 31: 2D Directions (left) and AR Directions (right)

Chapter 6: Building Information

6.1 Literature Review

Core Data is a framework that persists or caches data and supports undo on a single device [Apple 2020j]. In a Core Data model, developers are able to define data types and relationships between them and use the model to save the application's data, retrieve the application's data, and search for the application's data. In the building information function, we use the core data model to store, retrieve, and search for information for a total of 122 buildings.

6.2 Design Specification

6.2.1 User Interface Design

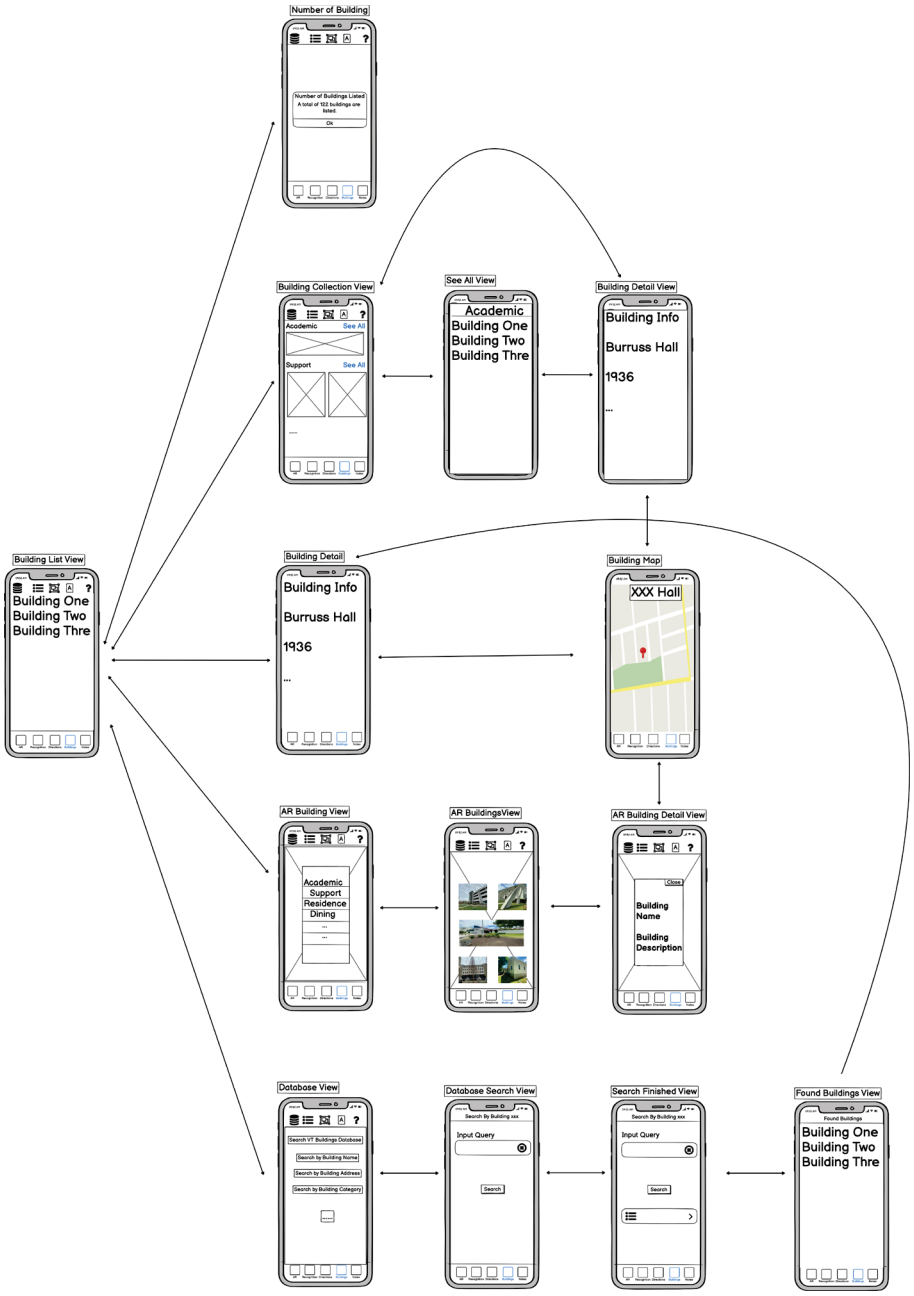

Figure 32: Prototyping of Buildings

Figure 32 shows the prototyping of the Building Information view. Users will be navigated to different views by different navigation bar buttons, and the UI design strictly follows the tab bar navigation pattern.

6.2.2 Database Design

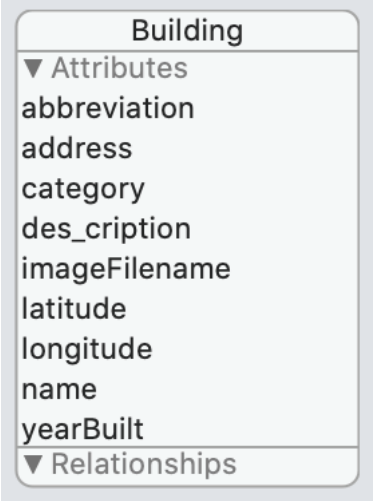

Figure 33: Database Design of Buildings

The database design is shown in Figure 33. In each Building entity of the Core Data model, there are attributes that represent building information, including building abbreviations, building address, building category, building description, building image file name, building latitude, building longitude, building full name, and also year built. We use this model to display building information and provide complex searches that allow users to search for buildings by building attributes.

6.3 Implementation

6.3.1 Standard Building List

A standard style list is displayed containing a total of 122 campus buildings. Each row has a list item that shows an image thumbnail, the building name, building type, and year of construction of a single building (Figure 34).

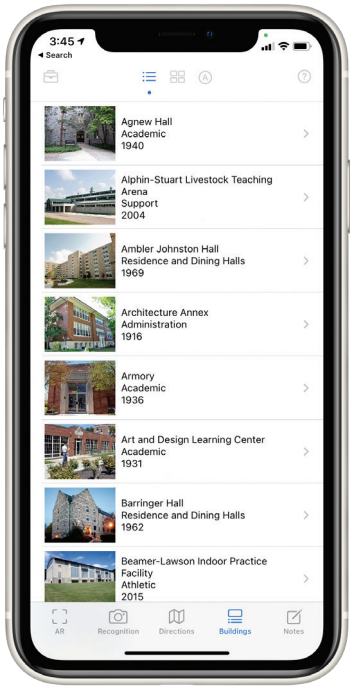

Figure 34: Building List View

After tapping on a row, the user is navigated to a detail page that includes the building details shown in both Figure 35 and Figure 36. In the Building Detail scene, the user will see sections including the building name, building photo, building name abbreviation, building type, the year building was built, displaying the building on a map, and building description, arranged from top to bottom.

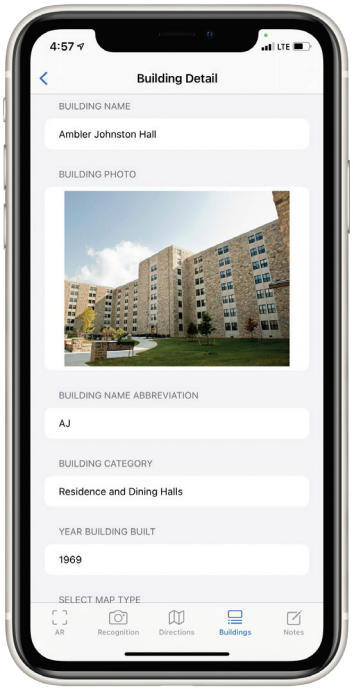

Figure 35: Building Detail View

If the user wants to view the building's location on the map, they can select a map type and click the Show on Map button (Figure 36).

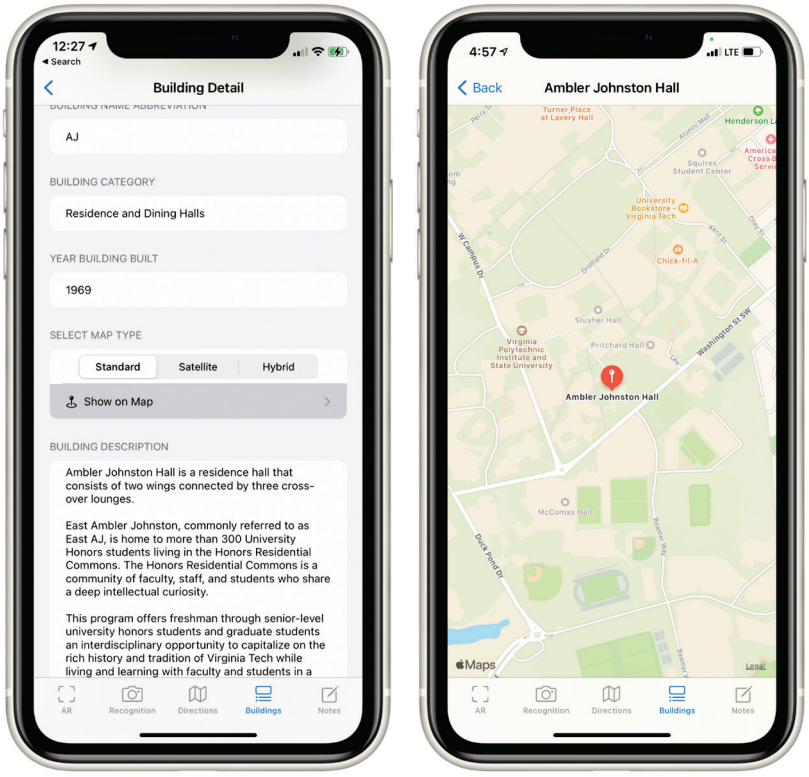

Figure 36: Show Building on Map

6.3.2 Building Collection View

We also designed a custom collection view organized by building category. Each section represents a building category, and each section has a header view with building category labels and a See All button (Figure 37). See All buttons have a blue background color to indicate that they are clickable.

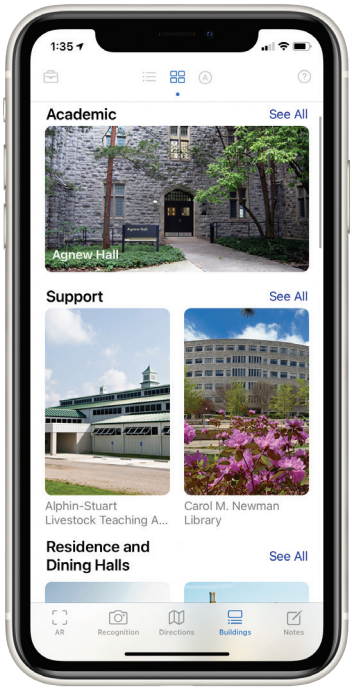

Figure 37: Building Collection View

In each building category section, cells can be scrolled from right to left. If the user wants to see the details of a building, they can click on the picture of the building, and the details of the building will be displayed in a modal view (Figure 38). Similarly, it contains the same building information as the detail page that is shown in Figure 34.

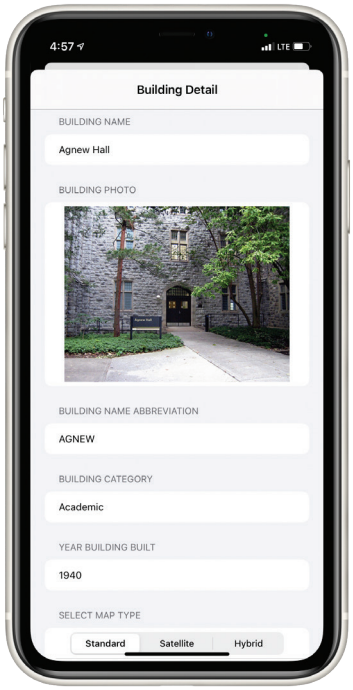

Figure 38: Building Detail Modal View

If users would like to get a list of buildings in a specific category, the See All button serves the requirement. Whenever the button is tapped, a list of buildings will be listed in a modal view (Figure 39). In this view, rows are clickable and will navigate users to a view that contains exactly the same information as the view shown in Figure 38.

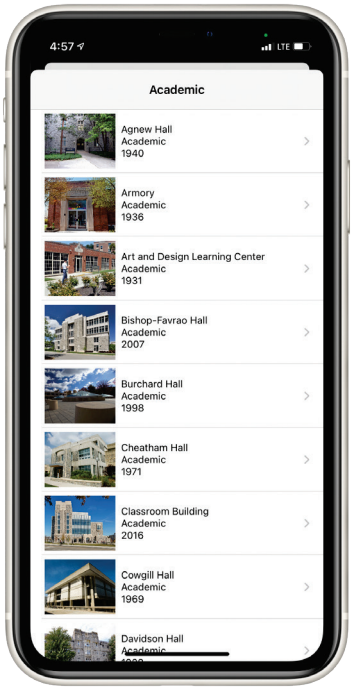

Figure 39: Building List by Category

We used the UICollectionView with UIViewRepresentable to replicate the behavior and UI of the collection view in SwiftUI since SwiftUI does not have a built-in collection view so far.

6.3.3 Building Number Alert Message

When the question mark icon in the upper right corner gets tapped, a message appears telling the user the total number of VT buildings that are shown in this view (Figure 40). The user can select OK to dismiss the message. This feature is intended to let the user know how many buildings are included in the Core Data model.

Figure 40: Number of Buildings

6.3.4 AR Building

Furthermore, we wanted to allow users to see the campus buildings not only in a standard view but also in an AR view. Therefore, we designed the AR Building that allows users to view building information in an AR scene. We provide a total of three ways for users to view the building information. The user can view the building through the standard style list, the collection view, and the AR view. However, the purpose of these views is the same. It gives the user detailed information about campus buildings, and all three views contain the same building information.

As shown in Figure 41, the user can select a building category to view all buildings of that type in this view.

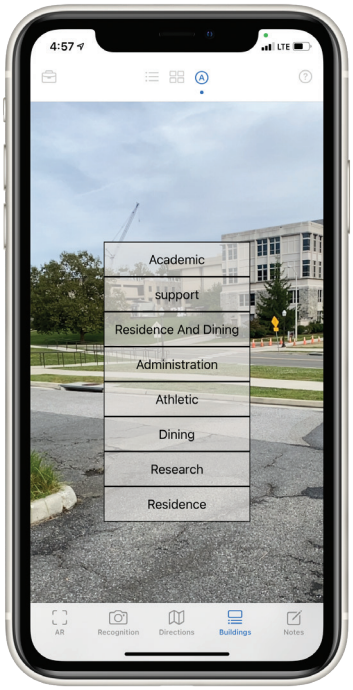

Figure 41: AR Building Type

Once a category is selected, images of all buildings of the selected type will be displayed in the AR scene (Figure 42). Building images in this scene will rotate 360 degrees around the user at a constant speed. This feature allows the user to learn about campus buildings through an immersive AR experience from anywhere.

Figure 42: AR Building

Likewise, the user can click on the building image to see its details. After that, the user can see the building detail overlay (Figure 43). If the user wishes to reselect the building category, they can click on the left arrow button shown in Figure 43 to go back to the AR Building Type scene to reselect the category.

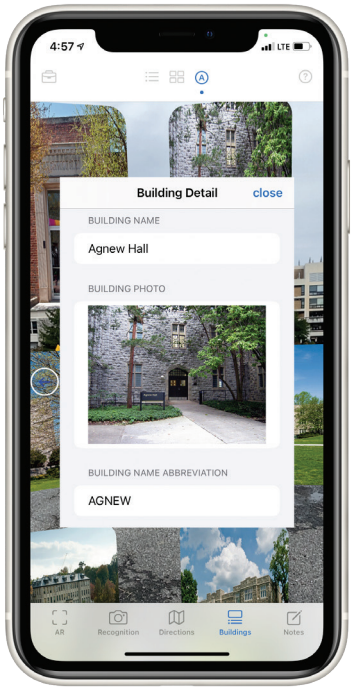

Figure 43: AR Building Detail Overlay

6.3.5 Building Database

Besides, users can search for buildings through the Core Data database. It is challenging to find a building in a list of 122 campus buildings, so we have implemented a search function on the Core Data model to find buildings by various attributes. As shown in Figure 44, the user can search the database by name, name or description, name abbreviation, category, address, year built, year built range, and category with year built range.
.

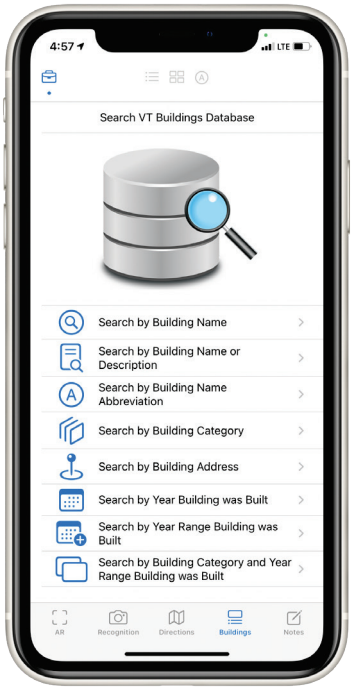

Figure 44: Search VT Buildings Database

By clicking on any of the search options in Figure 44, the user will be navigated to the corresponding search page. The user will then need to complete a search query to find a list of campus buildings that match the search criteria. Clicking on any row of the building results will redirect the user to the details page for that single building.

In this Buildings Information tab view, the user can be redirected to the Building Detail page in many sub-functions. For example, clicking on a row in the building list and clicking on a building picture in the building collection view will both navigate the user to the Building Detail scene. This design choice does not lead to redundancy but allows the user to learn about campus buildings thoroughly.

Chapter 7: Multimedia Notes

7.1 Design Specification

When visitors visit the campus, a tool to capture the moments of their trip can be beneficial. However, a camera app or a memo app with only basic functionality is not enough. So, we decided to create Multimedia Notes for campus visitors to capture their memories. This feature allows users to record text, pictures, and voice memos. Also, we used different views to visualize the stored notes.

7.1.1 User Interface Design

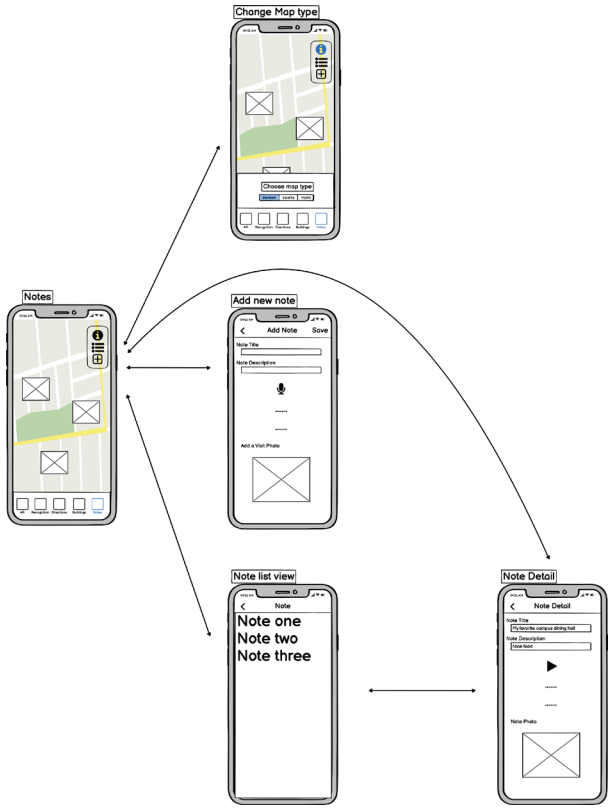

Figure 45: Prototyping of Note

The prototyping of Multimedia Notes is shown in Figure 45. We designed the UI and navigation pattern to be straightforward and efficient, similar to the Directions scene.

7.1.2 Database Design

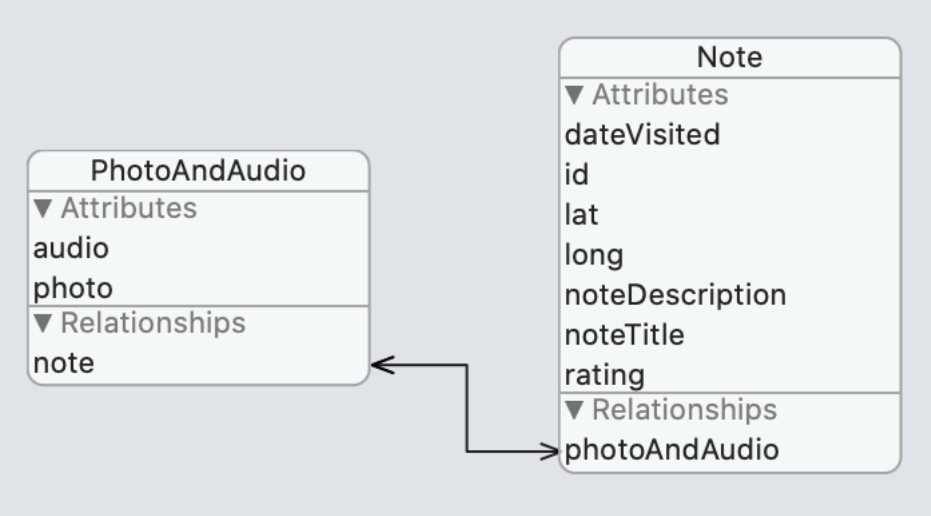

Figure 46: Database Design of Multimedia Notes

In the Core Data model, we designed two entities: PhotoAndAudio and Note, to store new notes and fetch notes. The Note entity contains attributes such as the date users have visited the campus, the location coordinates, the note description, the note title, and the place rating. The PhotoAndAudio entity contains the audio and image data of a note. We have also set up a relationship between two entities. By doing so, whenever we fetch a note, we will be able to retrieve the multimedia data of the note.

7.2 Implementation

7.2.1 Default Map View

The default scene of Multimedia Notes is a map with a set of buttons stacked in the upper right corner (Figure 47).

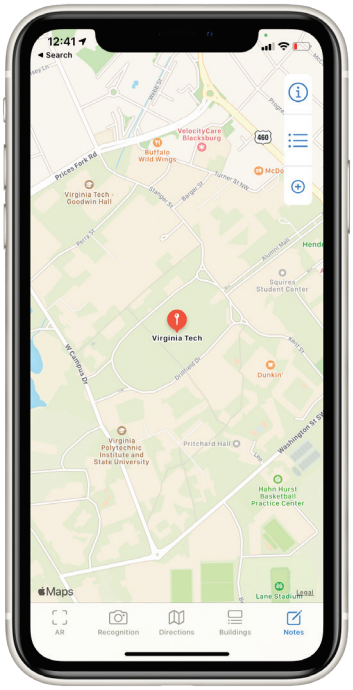

Figure 47: Multimedia Notes

7.2.2 Map Type Selection

Similar to the Map Type Selection function in the Directions feature, the user can change map types among three choices: Standard, Satellite, Hybrid by tapping the information button (Figure 48).

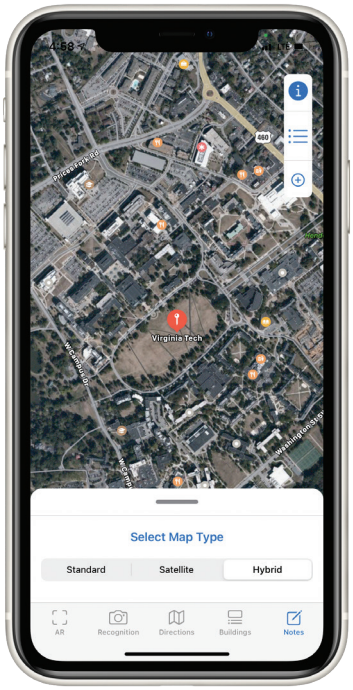

Figure 48: Map Type Selection

7.2.3 Add New Note

By tapping on the button with a plus symbol in the upper right corner, the user will be navigated to a view where they can fill in the data of a new note and save the note, as shown in Figure 49.

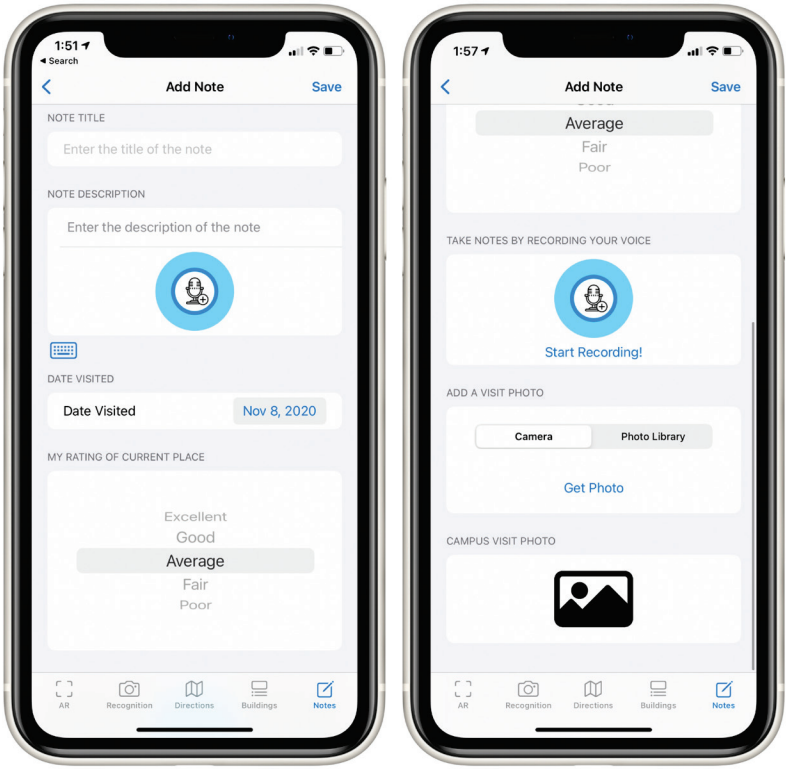

Figure 49: Add New Multimedia Note

In the Add Note scene, the user first needs to enter the title of the note. The user can then fill in the note description by typing on the keyboard or converting the speech to text. After that, the user needs to select the date of the visit. Then, the visitor can rate the places visited in a picker view. Users can also record voice memos. Finally, the user can select an image from the photo gallery or take a photo with phone cameras as the note image. If any of the input sections are missing, the user receives an error message (Figure 50).

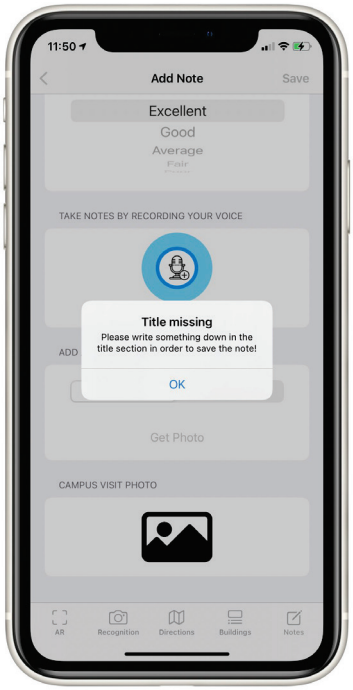

Figure 50: Input Missing Alert

7.2.4 Note List

If the user wants to view the saved notes, they can click on the button with the list icon in Figure 47 to access the list view. The list view displays all the saved notes in a list (Figure 51).

Figure 51: Note List View

If the user wants to check a note's details, they can click on the row where the note is located, which will take them to the note information page shown in Figure 52.

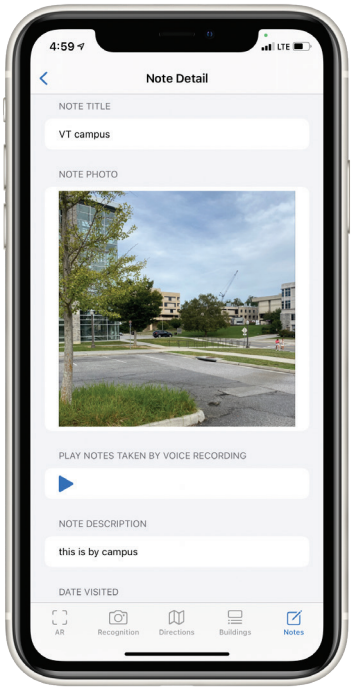

Figure 52: Note Detail View

7.2.5 Note on Map

Also, there is another way that the user can view the saved notes. As Figure 53 shows, whenever the user saves a new note, its photo is automatically placed in the map view according to the location of the note. After clicking on the note thumbnail on the map, a callout with the note title and date information will be displayed to the user. If the user clicks on the callout, the user will be redirected to the same note details page as the view shown in Figure 52.

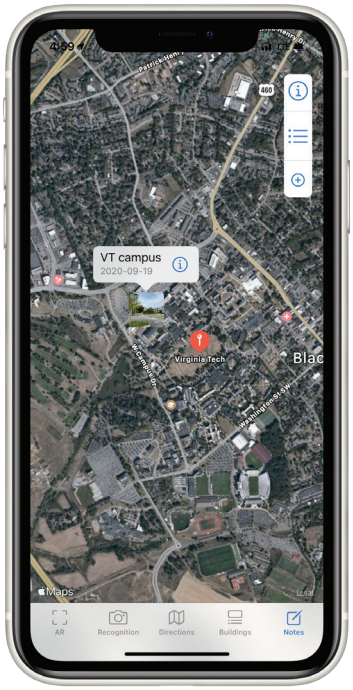

Figure 53: Note on Map

We customized the map annotation and changed it to a photo with a callout view. To embed MKMapView in *VTQuestAR,* we used UIViewRepresentable with the Coordinator.

Chapter 8: VTQuestAR Evaluation

This chapter gives a self-evaluation of the *VTQuestAR* application introduced from Chapter 3 to Chapter 7. Software Quality Assurance is a means of monitoring the software engineering processes and examining the quality of software. Specifically, quality characteristics are usually being used in the Software Quality Assurance process. Here, we analyze several quality indicators that fit our app best according to ISO 9126 and ISO 25010.

8.1 Functionality

Functionality is the essential purpose of any software. The functionality of applications and software is determined by whether all expected functionalities are present in the final product. We followed the software life cycle [Balci 2020] and discussed the desired functionalities of *VTQuestAR* in the early development stage. Furthermore, we examined how well our app has satisfied the expected functional requirements when we have finished the *VTQuestAR* development. The conclusion is that we have adequately addressed all the requirements and expected behaviors in our app.

8.2 Reliability

8.2.1 Maturity

The maturity is examined by how stable the app is during everyday use. We tested the app under different devices, including iPhones, iPads, and the simulators in Xcode. Moreover, we observed that the app was stable without unexpected behaviors such as crashing issues.

8.2.2 Recoverability

Recoverability examines whether the application can recover the data in the event of interruptions or failures. To keep the data safe and efficient for the data storing and retrieving process, we created entities to be stored in the Core Data model. Therefore, the recoverability of *VTQuestAR* is outstanding.

8.3 Usability

8.3.1 Learnability

Learnability refers to how easy it is for users to learn an application. Our applications provide straightforward functionality and user interface without the need for a user manual. For example, the names and icons of each tab bar item and button reveal their functions clearly. Besides, navigation bar titles and section headers in *VTQuestAR* allow users to understand and learn the features of our app quickly.

8.3.2 Operability

Operability determines whether the app is easy to control and operate. In *VTQuestAR*, we use the tab bar navigation pattern rather than the Hamburger Menu to ensure operability. Users always know which view they are controlling and can easily navigate among views.

8.4 Efficiency

Efficiency determines whether an app uses a reasonable amount of resources, such as battery and memory.

We used the Instrument developer tool provided by Xcode to examine the memory performance of *VTQuestAR*. By using the Instrument tool, we can check and analyze the allocation and deallocation of memory. After we run the analysis multiple times, *VTQuestAR* is proven to perform well regarding memory performance.

We also tested the app on the iPhone and iPad to examine the battery efficiency. The results are expected that our app does drain the battery faster than apps without AR and ML features because AR and ML require a considerable amount of calculations and resources, resulting in relatively high power consumption. In general, *VTQuestAR* consumes the battery at a reasonable rate.

8.5 Compatibility

Compatibility determines whether the app can be deployed on universal platforms. For *VTQuestAR*, though it only runs on Apple platforms, it can be deployed on the two most common Apple platforms: iPhone and iPad. The compatibility of our app meets our initial expectations, as the majority of the people who visit campus will only use iPhones or iPad due to the portability concern.

Chapter 9: Conclusions and Future Research

9.1 Conclusions

With the assistance of Virginia Tech building data database, our exploratory development research successfully resulted in an iPhone / iPad mobile application named *VTQuestAR*. The app helps resolve problems that Virginia Tech campus visitors face, such as recognizing a campus building, finding a building, obtaining directions from one building to another, and getting information about a building by using the augmented reality (AR) technology and machine learning (ML) technology. Compared to traditional campus visit tools, *VTQuestAR* breaks the limitation such as time and demonstrates the feasibility of using AR and ML in providing much more effective assistance to campus visitors of any organization.

Since *VTQuestAR* is designed to be a universal app that runs on iOS and iPadOS devices, the implementation of the user interface is essential. In this thesis, the SwiftUI framework is the technology we have used to build the user interface. Compared to Storyboards UI design, it is more efficient and compatible with most Apple devices. Besides, we discussed the reasons for using the tab bar navigation pattern instead of using the side menu navigation.

During the development of *VTQuestAR,* we strictly follow the software life cycle [Balci 2020]. We also apply incremental development principles to design and develop our app rather than the build-and-fix approach. By using ISO 9126 and ISO 25010 software quality indicators, we conduct the self-evaluation of *VTQuestAR* after we have finished the development of our app. The evaluation results suggest that *VTQuestAR* can benefit campus visitors and Virginia Tech enrollment management.

VTQuestAR is not only a helpful tool for campus visitors but also a project that demonstrates the efficiency of embedding AR and ML in mobile software. Our exploratory research can serve as a resource for other developers and organizations to build their own mobile software based on the development experiences of *VTQuestAR*.

9.2 Future Research

While we mainly focused on the achievement of functionalities, there are several aspects that can be improved to make *VTQuestAR* more powerful and effective.

The *VTQuestAR* application can be improved with the following:

1. Make the Core ML model to recognize all campus buildings.
2. Enable real-time turn-by-turn instructions for AR directions.
3. Support offline mode for 2D and AR directions.
4. Provide more AR object options in AR Camera.

5. Explore the feasibility of utilizing the LiDAR scanner to improve AR experiences.

In the future, we would like to build a system that supports multi-user AR experiences such that users can share the same AR scenes and AR objects in *VTQuestAR*. Also, we want to make *VTQuestAR* support social media features. For instance, when users visit the campus, they can post pictures, rate a place, and comment on other visitors' posts.

REFERENCES

Apple (2014), “Designing Intuitive User Experiences”
https://developer.apple.com/videos/wwdc2014/#211-video

Apple (2018), “Introducing Create ML”
https://developer.apple.com/videos/play/wwdc2018/703/?time=102

Apple (2020a), “Apple Developer Document: ARKit”
https://developer.apple.com/documentation/arkit

Apple (2020b), “Apple unveils new iPad Pro with breakthrough LiDAR Scanner and brings trackpad support to iPadOS”
https://www.apple.com/newsroom/2020/03/apple-unveils-new-ipad-pro-with-lidar-scanner-and-trackpad-support-in-ipados/

Apple (2020c), “Apple Developer Document: SceneKit”
https://developer.apple.com/documentation/scenekit/scnscene

Apple (2020d), “Apple Developer Document: Core ML”
https://developer.apple.com/documentation/coreml

Apple (2020e), “Apple Developer Document: Understanding World Tracking”
https://developer.apple.com/documentation/arkit/world_tracking/understanding_world_tracking

Apple (2020f), “Apple Developer Document: ARSession”
https://developer.apple.com/documentation/arkit/arsession

Apple (2020g), “Apple Developer Document: ARConfiguration.WorldAlignment.gravityAndHeading”
https://developer.apple.com/documentation/arkit/arconfiguration/worldalignment/gravityandheading

Apple (2020h), “Apple Developer Document: Core Location”
https://developer.apple.com/documentation/corelocation

Apple (2020i), “Apple Developer Document: MapKit”
https://developer.apple.com/documentation/mapkit

Apple (2020j), “Apple Developer Document: Core Data”
https://developer.apple.com/documentation/coredata

Boland, M. (2017), “ARCore + ARkit = 3.4 Billion Devices by 2022”
https://arinsider.co/2017/10/12/arcore-arkit-4-25-billion-devices-by-2020/

Balci, O. (2020), “CS3714 Course Website: Software Life Cycle” https://manta.cs.vt.edu/cs3714/Index.html

Minnich, G. (2019), “Blacksburg campus exceeds 50,000 visits for the year” https://vtnews.vt.edu/articles/2019/12/planes--trains--automobiles-.html

Merriam-Webster (2020), “Augmented reality” https://www.merriam-webster.com/dictionary/augmented%20reality/

Neetu, A. (2015), “Image Recognition Process through Human Eye, Computer and Artificial Intelligence,” *International Journal of Science and Research (IJSR) 6,* 3, 1449-1454

ProjectDent (2020), “ARKit + CoreLocation” https://github.com/ProjectDent/ARKit-CoreLocation

Swift.org (2019), “About Swift” https://swift.org/