Team 38 Smart Phone Repair Manual Code Review

Soren Andersen, Nowlen Webb, Claire Swanson

WalkThru

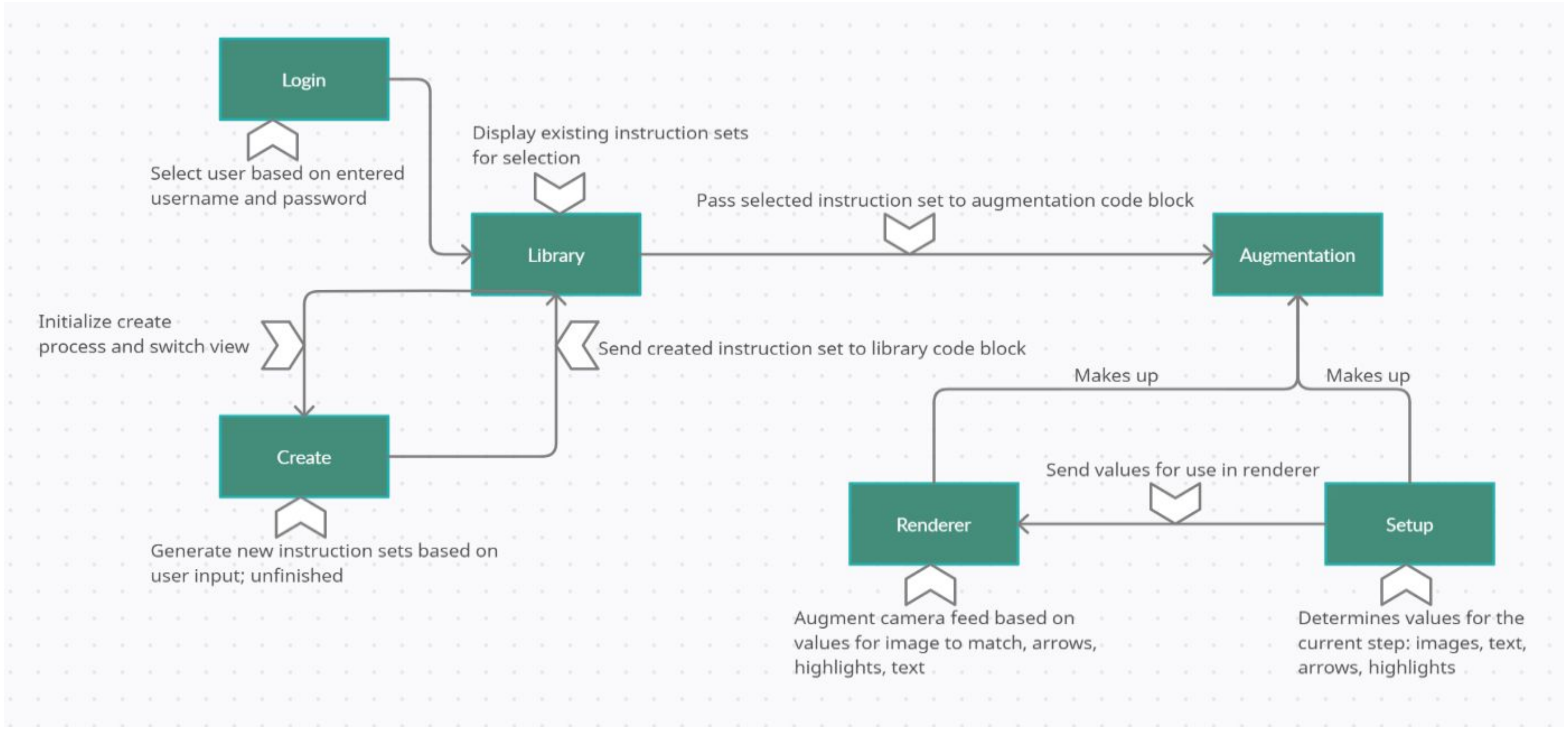
What It Is

Step-by-step how to guides using augmented reality to show users what they should be interested in.

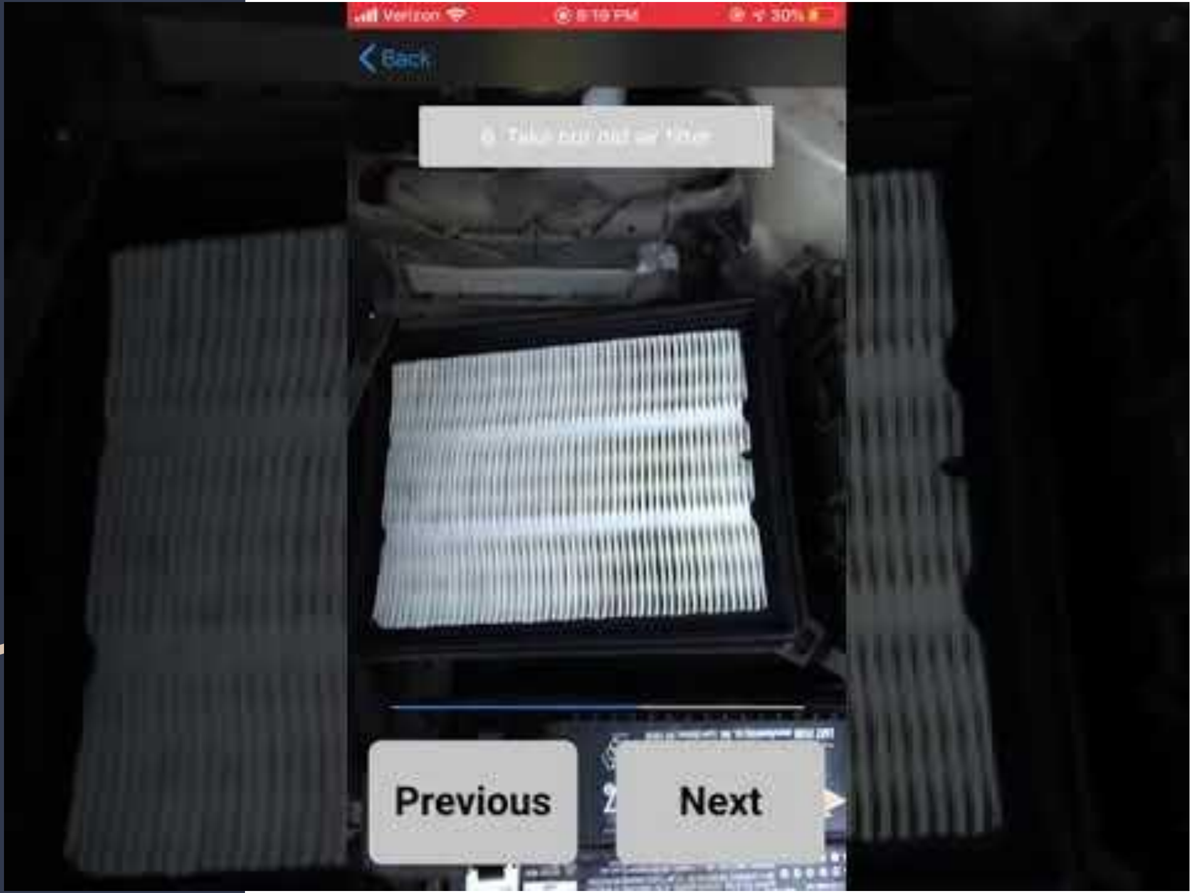
Tools Used Overview

- iOS Platform
    a. Apple devices with camera and ARKit support
        i. Tested on iPhone 6s and iPhone 8
            1. Compatible with iPhone 6s and newer
    b. MacinCloud
        i. Testflight
- Programming
    a. Swift/Xcode 12
        i. ARKit 3
            1. Image recognition
    b. Parse
        i. Swift SQL Package
    c. Began with Python SciKit

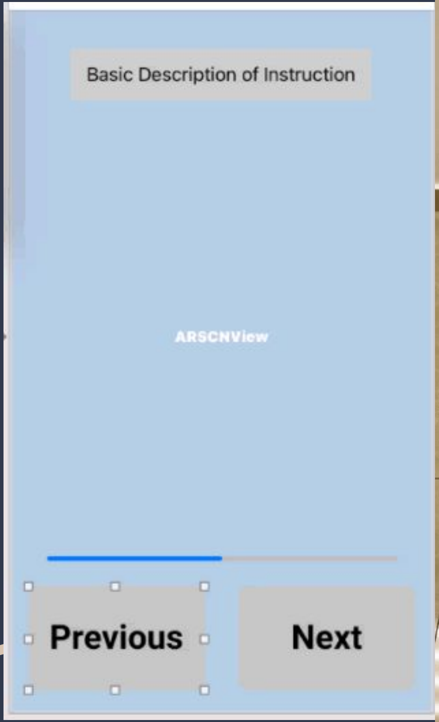
Functional Diagram

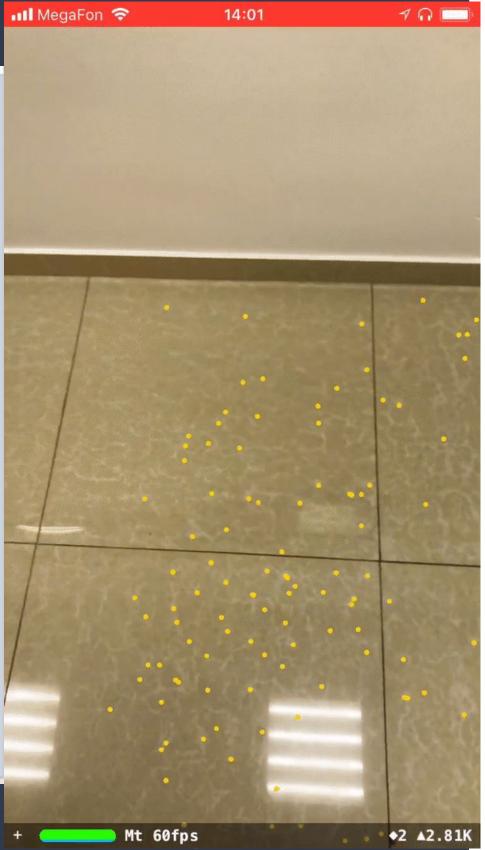

Login
Select user based on entered username and password
Display existing instruction sets for selection
Library
Pass selected instruction set to augmentation code block
Augmentation
Initialize create process and switch view
Send created instruction set to library code block
Makes up
Makes up
Create
Generate new instruction sets based on user input; unfinished
Send values for use in renderer
Renderer
Setup
Augment camera feed based on values for image to match, arrows, highlights, text
Determines values for the current step: images, text, arrows, highlights

Demo of functionality



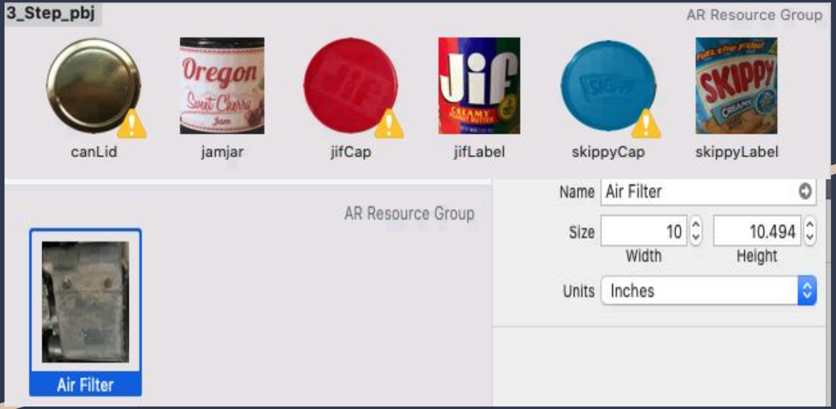
AR Overview





To understand the bulk of the app, we first need to explain how the image recognition system through ARKit 3 works. When correctly configured with an AR screen view such as the one on the left here, the view is essentially just a view of the phone’s camera. While we do not see everything that is going on in the background, the camera is no ordinary camera as it is creating key tracking points to judge the positional relationship between each object in frame. This idea is best exhibited by the GIF currently playing in which the user has what Apple calls “feature points” which are just inflection points turned on. In our case because we are using ARWorldTracking as our AR configuration our camera is simply waiting something it recognizes.

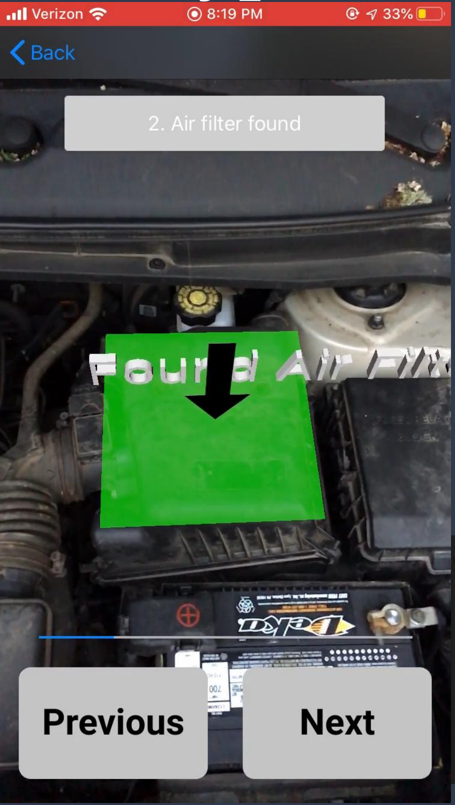
AR Configuration



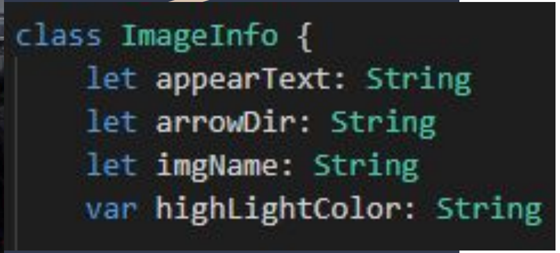
Name: Air Filter
Size: 10 (Width) 10.494 (Height)
Units: Inches

What can it recognize? Good question, after declaring the type of AR config we then bind a group of reference images in the form of an AR Resource group. An AR resource group is a folder of photos with measurements tied to each photo. These folders sometimes contain a single photo or many, however these resource groups are important to manage as Apple has a hard and fast cap on these folders with the limit being 24 photos. The larger these folders get the more time the phone spends judging each picture against its current image input and it quickly takes a huge toll on performance. With all that being said we are simply giving the AR config photos and measurements for it to recognize, and when it does we get sent to the renderer function.

3 Types of Node



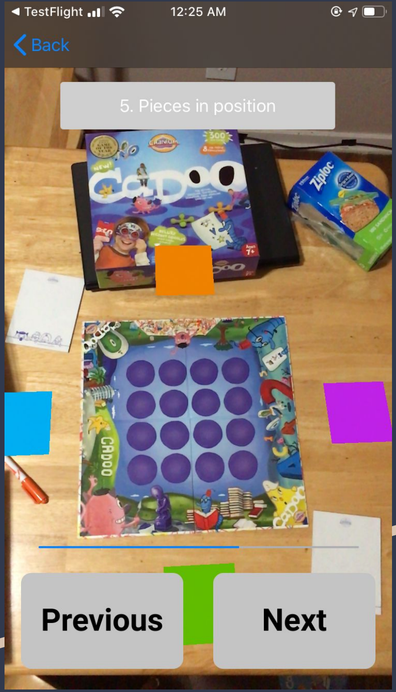
- Nodes Used
  - Text
  - Image (Arrow)
  - Plane



```
class ImageInfo {
    let appearText: String
    let arrowDir: String
    let imgName: String
    var highLightColor: String
```

The renderer function allows us to place 3d objects around the scene. Once the renderer function is called a central node called an anchor is placed at where the camera recognizes the photo. This anchor acts as the center of a new system of coordinates, which allows us to have a reference point to place nodes from. A node is simply an object in 3d space relative to the anchor. Nodes have a few simple characteristics: they require a point in 3d space relative to the anchor, a geometry type, and a material type. The paradigm behind these nodes is to separate by geometry as that represents the 3d object being presented. For our project we found the most useful types to be text, image, and plane. Although there are some more visually impressive geometries that come in ARKit, we found that along with the constant image recognition larger 3d shapes worsen performance. Ultimately the renderer function is the core of the AR tracking as it allows us to interact with the world through placing objects.

Nodes Improved



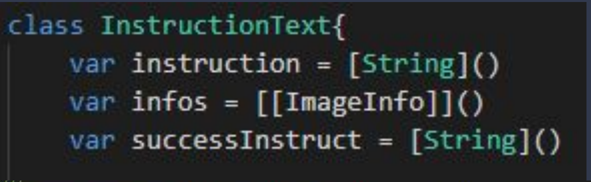
Although AR Kit 3 is doing a lot of the work through its calculations, right of the box image recognition and placing nodes is cumbersome and time consuming and with our plans to build an instructional app we would have to do this frequently. To make things more efficient we designed the ImageInfo class to describe a single image in an AR resource folder and the corresponding 3d objects to be placed around it. Then we added functions to the renderer to allow for standardization of these 3d objects such as the text node function you see on screen which simply takes in the width of the object recognized that the renderer function provides and the text we want displayed.

```
func addSpecialPlaneNode(shapeCheck: String, widthOfObject: CGFloat, heightOfObject: CGFloat, highlighCheck: String) -> [SCNNode] {
```

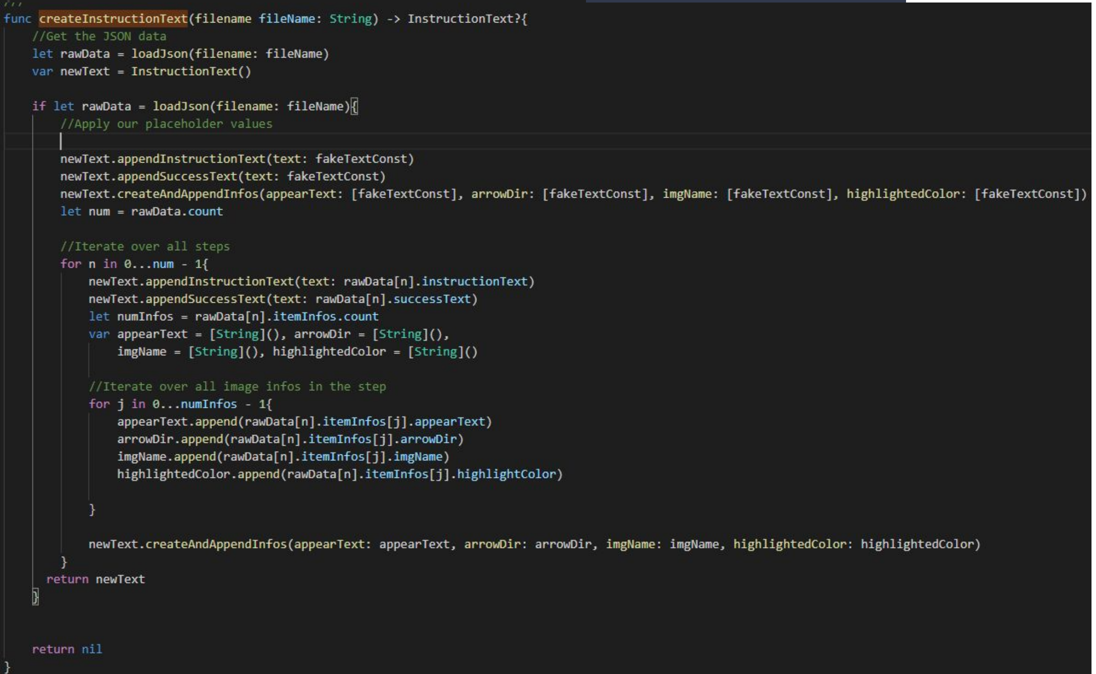
```
func addArrowNode(arrowName: String, widthOfObject: CGFloat, heightOfObject: CGFloat) -> SCNNode {
```

```
func addPlaneNode(shapeCheck: String, widthOfObject: CGFloat, heightOfObject: CGFloat, highlighCheck: String) -> SCNNode {
```

Loading Instruction Data



```
class InstructionText{
    var instruction = [String]()
    var infos = [[ImageInfo]]()
    var successInstruct = [String]()
```



```
'''
func createInstructionText(filename fileName: String) -> InstructionText?{
    //Get the JSON data
    let rawData = loadJson(filename: fileName)
    var newText = InstructionText()

    if let rawData = loadJson(filename: fileName){
        //Apply our placeholder values

        newText.appendInstructionText(text: fakeTextConst)
        newText.appendSuccessText(text: fakeTextConst)
        newText.createAndAppendInfos(appearText: [fakeTextConst], arrowDir: [fakeTextConst], imgName: [fakeTextConst], highlightedColor: [fakeTextConst])
        let num = rawData.count

        //Iterate over all steps
        for n in 0...num - 1{
            newText.appendInstructionText(text: rawData[n].instructionText)
            newText.appendSuccessText(text: rawData[n].successText)
            let numInfos = rawData[n].itemInfos.count
            var appearText = [String](), arrowDir = [String](),
                imgName = [String](), highlightedColor = [String]()

            //Iterate over all image infos in the step
            for j in 0...numInfos - 1{
                appearText.append(rawData[n].itemInfos[j].appearText)
                arrowDir.append(rawData[n].itemInfos[j].arrowDir)
                imgName.append(rawData[n].itemInfos[j].imgName)
                highlightedColor.append(rawData[n].itemInfos[j].highlightColor)

            }

            newText.createAndAppendInfos(appearText: appearText, arrowDir: arrowDir, imgName: imgName, highlightedColor: highlightedColor)
        }
      return newText
    }


    return nil
}
```

Once we had a standard way of placing objects by creating nodes based on the image found, we could move onto to creating full instruction sets. Each instruction set is simply a .json file of type InstructionText which consists of all the ImageInfo objects in the relevant AR resource folder and 2 different text variables. Each of these objects represent an entire step in our WalkThru. Each time a user clicks on a WalkThru on the library screen, the relevant .json is read into the AR view controller. This setup drastically reduced the amount of time it takes to create one WalkThru while also decreasing the complexity from needing to create a new AR view controller for each WalkThru to simply needing a formatted .json and the corresponding AR resource groups. Although we tried to make it as simple as possible, creating a single WalkThru still takes a while as the requirement of creating AR resource groups with cropped photos and measurements is time consuming.

User Stories

- As a mechanic, I need a simple user interface so that I can follow the guide as I complete the job.
- As a user, I need the AR coloring to be clear enough to see each part I am supposed to manipulate so I can quickly complete the project.
- As a user, I need the object I am working on to be recognized so I can work on the project from most angles I want.
- As a user, I need to see markings on video of parts so that I can know what the next step is and avoid confusion.