Documentation for myObjDetector App

1. Overview

The provided code shows how to implement a custom camera view for an iOS app that uses the AVFoundation framework to capture frames from the device's camera, performs rectangle detection on each frame using the Vision framework, and displays the detected rectangles using a custom overlay.

The code is composed of native Swift files, a native Objective-C file, and a React Native JavaScript file. The Swift files define classes responsible for setting up and managing the camera capture, while the Objective-C file acts as a bridge to expose these classes to React Native. The JavaScript file defines a React Native component that uses the custom camera view and renders the bounding boxes overlay.

2. Architecture

The architecture of the code is divided into three main sections: Swift, Objective-C, and JavaScript.

Swift

In the Swift code, there are two main classes defined:

- **CameraManager**: This is a wrapper class that provides a view method to return the **CameraView** instance and a **requiresMainQueueSetup** method to specify the setup necessity on the main queue.
- **CameraView**: This class manages the camera view. It sets up the capture session, preview layer, input and output devices, and processes the frames from the capture. It detects rectangles in the frames and calls a provided closure for each detected rectangle.

Objective-C

The Objective-C code defines **CameraManagerBridge**, an Objective-C class that acts as a bridge to expose CameraManager to React Native. It uses macros to declare the view and requiresMainQueueSetup methods, and the **onBoundingBox** property, to be accessible from JavaScript.

JavaScript

The JavaScript file imports **CameraManager** from native modules and uses it to create a custom camera view in a React Native component. It handles the **onBoundingBox** event to update the state with new bounding boxes and renders an overlay view for each bounding box.

3. Workflow

- **CameraManager's view** method is called to create a new **CameraView** instance.
- In the **CameraView's** initializer, **startCaptureSession** is called to start the camera capture.
- In **startCaptureSession**, a capture session is created, a device input is created from the device's back camera, an output is created to process the frames, and a preview layer is created to display the capture.
- **captureOutput(_:didOutput:from:)** is called every time a new frame is captured. It extracts the pixel buffer from the frame, performs rectangle detection on it, and calls the **onBoundingBox** closure for each detected rectangle.
- **onBoundingBox** closure updates the boxes state with the detected rectangles.
- The **MyCameraComponent** React Native component renders a **BoundingBox** component for each rectangle in the **boxes** state.

4. Components Detail

Swift

- **CameraManager**: This class inherits from NSObject and provides a **view ()** method to create a new instance of **CameraView.** It also provides **requiresMainQueueSetup** to specify the setup necessity on the main queue. This class is later exposed to React Native through a bridging header and an Objective-C file.
- **CameraView**: Inherits from UIView and conforms to AVCaptureVideoDataOutputSampleBufferDelegate protocol. It is responsible for the camera view setup and management. It provides a **startCaptureSession** function that sets up the capture session, adds inputs, and prepares the outputs. The class also implements a delegate method **captureOutput(output: AVCaptureOutput,**

didOutput sampleBuffer: CMSampleBuffer, from connection: AVCaptureConnection) from AVCaptureVideoDataOutputSampleBufferDelegate that gets called every time a new video frame was written.

Objective-C

- **CameraManagerBridge.m:** This file is responsible for bridging between Objective-C and Swift. It uses RCT_EXTERN_REMAP_MODULE to expose **CameraManager** class to JavaScript, and RCT_EXPORT_VIEW_PROPERTY to expose the **onBoundingBox** property to JavaScript.

JavaScript

- **MyCameraComponent**: This is the main React component that users will interact with. It initializes the **CameraManager** component and handles the **onBoundingBox** event. It updates its state with each new detected bounding box and renders a **BoundingBox** component for each of them.

5. Implementation Details

- **startCaptureSession** function in **CameraView** sets up the capture session, adds input from the back camera, prepares the output with a sample buffer delegate, and starts the session.
- When a new frame is captured, **captureOutput** gets called. It extracts the pixel buffer from the frame and performs rectangle detection on it using **VNDetectRectanglesRequest**.
- For each detected rectangle, it calls the **onBoundingBox** closure with the rectangle's bounding box.
- In **MyCameraComponent**, the **onBoundingBox** event handler updates the boxes state with the new bounding box. The component's render method renders a **BoundingBox** component for each bounding box in the **boxes** state.
- **BoundingBox** is a functional component that renders an absolute positioned view with the properties of the bounding box passed as props.

6. Note: -Apple's Vision over Google's ML Kit for object detection

I choose Apple's Vision over Google's ML Kit for object detection in our camera component due to several reasons:
Native Integration: Being an Apple product, Vision offers seamless integration and performance optimization on iOS devices.
Offline Processing: Vision can perform all processing on-device, ensuring functionality even without internet connection.
Privacy: On-device processing means data isn't sent to the cloud, enhancing user privacy.
Performance: Vision provides high performance, especially for real-time image analysis.
Rectangle Detection: Vision's VNDetectRectanglesRequest is ideal for our specific requirement of detecting rectangular objects, providing a more targeted solution than the general object detection in ML Kit.