Documentation for myObjDetector App

1. Introduction

This camera component allows integration of a real-time video feed with a React Native application. The video feed comes from the device's camera, and this component can detect objects within the feed, outputting bounding box coordinates for each detected object.

2. Architecture

The camera component makes use of Swift and Objective-C for the native part of the code, interacting with the AVFoundation and Vision libraries for camera access and object detection. It uses React Native for the JavaScript part of the code, where it interfaces with the native code to display video feed and detected objects.

The main components of the camera component are:

- **CameraManager.swift**: A Swift class that manages the custom camera view. It inherits from RCTViewManager which is part of the React Native Bridge.
- **CameraView.swift**: This Swift class handles the actual camera functionality, including starting the capture session, processing video frame data, and detecting objects in the video feed.
- **CameraManagerBridge.m**: An Objective-C file that exposes the CameraManager to the React Native code.
- **myObjDetector-Bridging-Header.h**: The Objective-C header file that is used to bridge between Swift and Objective-C.
- **React Native code**: This JavaScript code creates and manages the native camera view, as well as the bounding boxes for each detected object.

3. Workflow

When the component is loaded, the CameraManager creates a new instance of CameraView. The CameraView starts a capture session on the device's back camera, with video data output being sent to a DispatchQueue for processing. Each frame of

video data is then processed to detect objects within the frame, and the bounding box data for each detected object is sent back to the JavaScript part of the code.
The JavaScript component MyCameraComponent renders the CameraView and for each bounding box received, a BoundingBox component is rendered. The BoundingBox component is a simple View with absolute positioning, based on the coordinates of the bounding box. This BoundingBox is overlayed on top of the CameraView, effectively outlining detected objects in the video feed.

4. File Components

- **CameraManager.swift**

This is the main class that manages the camera view. It overrides the view method to return our custom CameraView and requiresMainQueueSetup to ensure that the module setup is performed on the main thread.

- **CameraView.swift**

This class handles the camera capture session. It gets the back camera of the device, sets it as the input to the AVCaptureSession, and then sets up a AVCaptureVideoPreviewLayer to display the video feed. It then sets up an AVCaptureVideoDataOutput object and sets itself as the delegate, so that it can receive video frames for processing. When a video frame is received, it uses the Vision library to detect objects within the frame, and then calls the onBoundingBox function with the bounding box data for each detected object.

- **CameraManagerBridge.m**

This Objective-C file is used to expose the CameraManager to the React Native code, allowing JavaScript to use the native functionality. It exports the onBoundingBox property to the React Native side.

- **myObjDetector-Bridging-Header.h**

This Objective-C header file is used to bridge between Swift and Objective-C, allowing the two languages to interact with each other. It imports the necessary headers for bridging.

- **React Native code**

This JavaScript code handles the rendering of the native view and the bounding boxes. It uses the CameraView imported from the native code and sets the onBoundingBox property to a function that updates the state whenever a new bounding box is received. The state consists of an array of bounding boxes, and for each bounding box in the array, a BoundingBox component is rendered. The BoundingBox component is a View

that is styled based on the bounding box data, and when rendered over the CameraView, creates the effect of outlining the detected object.

5. 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.