Sample Code

Tracking the User’s Face in Real Time

Detect and track faces from the selfie cam feed in real time.

Download

iOS 12.0+ | iPadOS 12.0+ | Xcode 11.3+

Overview

The Vision framework can detect and track rectangles, faces, and other salient objects across a sequence of images.

This sample shows how to create requests to track human faces and interpret the results of those requests. In order to visualize the geometry of observed facial features, the code draws paths around the primary detected face and its most prominent features.

The sample app applies computer vision algorithms to find a face in the provided image. Once it finds a face, it attempts to track that face across subsequent frames of the video. Finally, it draws a green box around the observed face, as well as yellow paths outlining facial features, on Core Animation layers.

To see this sample app in action, build and run the project on iOS 11. Grant the app permission to use the camera.

Configure the Camera to Capture Video

This section shows how to set up a camera capture session using delegates to prepare images for Vision. Configuring the camera involves the following steps.

1. First, create a new `AVCaptureSession` to represent video capture. Channel its output through `AVCaptureVideoDataOutput`.

2. Query the user's input device and configure it for video data output by specifying its resolution and camera.
3. Next, create a serial dispatch queue. This queue ensures that video frames, received asynchronously through delegate callback methods, are delivered in order. Establish a capture session with AVMediaType video and set its device and resolution.
4. Finally, designate the video's preview layer and add it to your view hierarchy, so the camera knows where to display video frames as they are captured.

Most of this code is boilerplate setup that enables you to handle video input properly. Tweak the values only if you choose a different camera arrangement.

Parse Face Detection Results

You can provide a completion handler for a Vision request handler to execute when it finishes. The completion handler indicates whether the request succeeded or resulted in an error. If the request succeeded, its `results` property contains data specific to the type of request that you can use to identify the object's location and bounding box.

For face rectangle requests, the `VNFaceObservation` provided via callback includes a bounding box for each detected face. The sample uses this bounding box to draw paths around each of the detected face landmarks on top of the preview image.

```
let faceDetectionRequest = VNDetectFaceRectanglesRequest(completionHandler: { (reque

    if error != nil {
        print("FaceDetection error: \(String(describing: error)).")
    }

    guard let faceDetectionRequest = request as? VNDetectFaceRectanglesRequest,
          let results = faceDetectionRequest.results else {
            return
    }
    DispatchQueue.main.async {
        // Add the observations to the tracking list
        for observation in results {
            let faceTrackingRequest = VNTrackObjectRequest(detectedObjectObservatio
            requests.append(faceTrackingRequest)
        }
        self.trackingRequests = requests
    }
})
```

In addition to drawing paths on `CALayer` to visualize the feature, you can access specific facial-feature data such as eye, pupil, nose, and lip classifications in the face observation's `landmarks` property. Your app can leverage this information to track the user's face and apply custom effects. For a face landmarks request, the face rectangle detector will also run implicitly.

Preprocess Images

Perform any image preprocessing in the delegate method `captureOutput:didOutputSampleBuffer:fromConnection:`. In this delegate method, create a pixel buffer to hold image contents, determine the device's orientation, and check whether you have a face to track.

Before the Vision framework can track an object, it must first know which object to track. Determine which face to track by creating a `VNImageRequestHandler` and passing it a still image frame. In the case of video, submit individual frames to the request handler as they arrive in the delegate method `captureOutput:didOutputSampleBuffer:fromConnection:`.

```
let imageRequestHandler = VNImageRequestHandler(cvPixelBuffer: pixelBuffer,
                                                orientation: exifOrientation,
                                                options: requestHandlerOptions)

do {
    guard let detectRequests = self.detectionRequests else {
        return
    }
    try imageRequestHandler.perform(detectRequests)
} catch let error as NSError {
    NSLog("Failed to perform FaceRectangleRequest: %@", error)
}
```

The `VNImageRequestHandler` handles detection of faces and objects in still images, but it doesn't carry information from one frame to the next. For tracking an object, create a `VNSequenceRequestHandler`, which can handle `VNTrackObjectRequest`.

Track the Detected Face

Once you have an observation from the image request handler's face detection, input it to the sequence request handler.

```
try self.sequenceRequestHandler.perform(requests,
                                         on: pixelBuffer,
                                         orientation: exifOrientation)
```

If the detector hasn't found a face, create an image request handler to detect a face. Once that detection succeeds, and you have a face observation, track it by creating a `VNTrackObjectRequest`.

```swift
// Setup the next round of tracking.
var newTrackingRequests = [VNTrackObjectRequest]()
for trackingRequest in requests {

    guard let results = trackingRequest.results else {
        return
    }

    guard let observation = results[0] as? VNDetectedObjectObservation else {
        return
    }

    if !trackingRequest.isLastFrame {
        if observation.confidence > 0.3 {
            trackingRequest.inputObservation = observation
        } else {
            trackingRequest.isLastFrame = true
        }
        newTrackingRequests.append(trackingRequest)
    }
}
```

Then call the sequence handler's `perform(_:)` function. This method runs synchronously, so use a background queue to avoid blocking the main queue as it executes, and call back to the main queue only if you need to perform UI updates such as path drawing.

See Also

Object tracking

{} Tracking Multiple Objects or Rectangles in Video

Apply Vision algorithms to track objects or rectangles throughout a video.

`class VNTrackingRequest`

The abstract superclass for image-analysis requests that track unique features across multiple images or video frames.

`class VNTrackRectangleRequest`

An image-analysis request that tracks movement of a previously identified rectangular object across multiple images or video frames.

`class VNTrackObjectRequest`

An image-analysis request that tracks the movement of a previously identified object across multiple images or video frames.

`class VNDetectedObjectObservation`

An observation that provides the position and extent of an image feature that an image-analysis request detects.