cover_logos.tifcover_logos.tifFacial-Object Recognition System

# Problem & Solution

## Dietary Intake Problem

Fact, astronauts need to take track of their dietary intake to be studied. The current way to track it is using the FFQ (Food Frequency Questionnaire) where they declare the food they have eaten. The main issues with this method are the time it takes to be filled, the difficulty to remember what they have eaten certain times, among others.

This generates the need to create a solution that replaces and improves this method through an iPad application considering the problem key factors and constraints, such as low lightning conditions, in-app image/video processing, body changes due to encounter with micro-gravity and iPad’s accelerometer sensitivity on space (hard to detect where it is up or down).

## Facial-Object Recognition Idea Overview

The idea of having facial and object recognition in the application is edgy, it’s something unique, very helpful because it reduces the interaction paths and *makes the data registry process (FFQ) smooth* by pointing out the camera to objects instead of regular UI interaction*.* The application would provide a view tool where the astronaut can easily recognize him/her (user login) and the food he/she is having.

This functionality should be a primary asset of the application only if it’s implemented with an acceptable level of efficiency when detecting objects and faces, and this implies to run tests on real or similar environment where the application is going to be used; otherwise it could be an obstacle to the user experience if there are too many failures trying to recognize objects, it could become annoying and we don’t want that. However, it would make a very useful asset.

Experienced Source

According to Dr. Pettit – experienced NASA astronaut [interviewed by Clinton Bonner](http://www.youtube.com/watch?v=bmWQRFVSf-8) – there are important conditions we need to consider; the light level, packages surface may provide difficulties for image recognition. **This may cause problems and frustration**, we don’t want that for the users (we’ll see how to handle this below in “Facial-Object Recognition System Features”).

## Facial-Object Recognition System Features

Considering the important conditions, the image recognition system should offer certain features in order to successfully perform visual operations.

* **User identification**; the application should provide an option for log in, where it associates faces with users (astronauts) by comparing selected facial features from the camera view and a facial database.
* **Food scanning**; the camera view should work as a scanner so the user doesn’t have to take a picture; it’s in a sort of video mode. If there are any matches the UI should properly react and let the user continue with the FFQ registry.
* **Auto-training**; we need to accept there could be difficulties for proper scanning in the environment, also with probably inaccuracy of the scanning matching system. There are some SDKs that provide auto-training capabilities where the database is populated with images automatically associated to a food the user is trying to scan. Next time it will have more angles and perspectives from that product to successfully provide a match.
* **Barcode system link**; since both barcode and object recognition systems use the same resource (camera) they should be integrated into one single option. Instead of having options such as “scan package barcode” and “recognize food“ there should be only “scan food”, which will perform an object recognition scan but if there’s any chance it finds a barcode it should quickly match the product, either by barcode or object recognition, the first thing that happens. There are other approaches of course, but this could be a good start.
* **Hard conditions capability**; it’s a known fact the environment challenge is quite considerable. Low lightning, shiny packages, transparent packages, among others. This goes for the SDK; it must support this kind of conditions.
* **Failure smart response**; from the same hard conditions scenario, the application must provide a mechanism to counter attack this issue. It’s very likeable some scanning may fail and if this happens continuously the application needs to respond back. The application will allow three continuous failures maximum, at the third time a message will be prompted, showing an apology with options to continue, these might be: speak, explore the UI or save the product (added as new match item to images database). Saving a new product is not that open thought, the user would pick or search the product from the UI (we would need to handle a set of space food) so there is consistency and data integrity.
* **In-app processing**; there is not access to internet for this application. The framework should be able to provide an off-line performance.
* **Fixed and auto focus camera support**; there are known issues with several SDKs not working properly on fixed-focus cameras (iPad 2) because they can’t process blurry pictures properly. The new iPad has an auto-focus camera, so it’s not a problem.
* **System bulk pre-load**; the proposed auto-training feature could do this work, however there should be a formal option for administrative staff to pre-load the images database that will be used as match parameter (faces and food).

# Barcode Scanning Frameworks-iOS Review

There is a wide variety of frameworks for image recognition, open source and proprietary. In this study I’m focusing the most suitable ones for the proposed behavior of the facial-object recognition system.

I have to confess this choice was straight forward and simple after reviewing the existing frameworks. The potential and strength of **OpenCV** is the best suit for the project, no doubt at all. It’s open source, it’s powerful and provides documentation for iOS development. There is another open source library, Tessetact, it is not as powerful as OpenCV neither some commercial ones but it’s useful though (with a low percentage of accuracy). The top three are marked with a start. Let’s study *the why* of these particular picks.

comparison.tif

*Key Features Comparison*

## OpenCV

No doubt this is the top choice, it's very complete and powerful; it has everything!

OpenCV is an open source library (free for academic and commercial use) that includes several hundreds of computer vision algorithms. It can be used to object recognition, motion tracking, image processing, video processing, among other things. It has a wide community and documentation. For our purposes, it provides official resources for iOS development.

OpenCV has trained collection of templates in XML files for detecting faces for example. OpenCV contains utility for training thus you are able to generate cascades for other kinds of objects.

**OpenCV for iOS**

There are several compiled libraries ready to use on iOS applications from many sources. The most solid is one version released by [eosgarden](http://www.eosgarden.com/en/opensource/opencv-ios/overview/) and the recommended to use which supports multiple versions of the iPad.

In details:

* Supports any iOS version. Works better with iOS 4 and higher though.
* Fixed focus camera support.
* Provides training system.
* Provides facial recognition system.
* 2D and 3D feature toolkits.
* Egomotion estimation.
* Gesture recognition.
* Motion understanding.
* Object identification.
* Free to use. You need to give credits to OpenCV if you use it according to their [license](http://www.eosgarden.com/en/opensource/licenses/bsd/).
* [Website](http://opencv.org/).
* [Tutorials](http://docs.opencv.org/doc/tutorials/ios/table_of_content_ios/table_of_content_ios.html#table-of-content-ios).
* [XCode project](https://github.com/macmade/OpenCV-iOS/archive/master.zip).

**Open CV application samples**

* <http://www.youtube.com/watch?v=xzVXyrIRm30>
* <http://www.youtube.com/watch?v=zXHXusO8eGw>
* <http://www.youtube.com/watch?v=CigGvt3DXIw>

TopCoder Implementation Advice

This is an excellent choice; documentation is very organized and easy. However, if you decide to implement this you should know that the learning curve takes time. If competitors have to learn to use this during the competition you should consider setting long deadlines.

## IQ Engines SmartCamera SDK

From commercial libraries, Smartcamera SDK is the first best for recommendation for the project needs. The only limitation is it works on-line for doing object recognition, however it can work with up to 100 images in a local database, if you need more it has to be done with IQ Engines cloud help.

VisionIQ software resides on mobile phones and in the cloud. It provides an iOS SDK that is designed to handle on-device visual scans for image recognition, QR codes and barcodes. On-device scans are fast, less than 1 second.

See this cool working sample: <http://vimeo.com/58229429>.

In Details:

* Supports training system.
* Supports object recognition.
* Supports iOS 4 and higher.
* Commercial license with moderated prizing.
* [Website](https://www.iqengines.com/smartcamera/).
* [Tutorial](https://www.iqengines.com/tutorial/).

## Moodstocks

This is my second favorite SDK from paid licenses. It’s technically gifted with a clean architecture for implementation (very easy to implement). However it’s a bit expensive and depends from internet connectivity, and again however, the SDK performance and quality is lot better than the second best non-commercial library (Tessetact).

See working sample: <http://www.youtube.com/watch?v=4V6Nd2TS5n8>.

In Details:

* Supports object recognition.
* Supports iOS 4 and higher.
* Commercial license with moderated prizing.
* [Website](http://www.moodstocks.com/).

## Other Frameworks

Since there a lot of other frameworks that could probably be used in the application in an acceptable way I won’t keep them out so I’ll just group them quickly here, in the middle-reliability level / average performance for hard conditions group.

[Tessetact](https://github.com/ldiqual/tesseract-ios) »

* Simple but limited.
* Open source.
* Supports iOS 4 and higher.
* Provides facial and object recognition.

[Metaio SDK](http://www.metaio.com/products/sdk/) »

* Technically powerful, very powerful.
* Extremely expensive.
* Supports iOS 5 and higher.
* Provides facial and object recognition.

[Recognize IM](https://www.recognize.im/) »

* Well documented.
* On-line dependant.
* Provides easy way to integrate to projects.
* Commercial license.
* Supports iOS 5 and higher.
* Provides facial and object recognition.