



UNIVERSITÀ DEGLI STUDI DI ROMA "SAPIENZA"
FACOLTÀ DI INGEGNERIA
CORSO DI LAUREA IN INGEGNERIA INFORMATICA
DIPARTIMENTO DI INFORMATICA E SISTEMISTICA

Tesi di Laurea

Bonsai Due

Smart Growing Assistant for Bōnsai Lövers

iPhone App

Federico Paliotta

Relatore
Prof. Luigi Laura

A volte ci preoccupiamo tanto perché le foglie cadono, ma non abbiamo tenuto conto dell'autunno.

Indice

1. Introduzione	5
2. Specifiche dell'applicazione.....	9
2.1. Cura del bonsai e principali problemi.....	10
2.1.1. Dove e come disporre le piante.....	11
2.1.2. Parassiti e malattie.....	11
2.1.3. Manutenzione.....	12
2.2. Requisiti di progetto	12
2.2.1. Funzionalità di base	13
2.2.2. Funzionalità avanzate.....	13
3. Tecnologie utilizzate.....	15
3.1. Cenni sull'linguaggio Objective-C.....	15
3.1.1. Sintassi.....	16
3.1.2. Messaggi	18
3.1.3. Tipizzazione dinamica.....	19
3.1.4. Objective-C 2.0.....	20
3.2. Panoramica su iOS	21
3.2.1. Cocoa Touch.....	22
3.2.2. Media.....	23
3.2.3. Core Services.....	24
3.2.4. Core OS	25
3.3. Libreria OpenCV	26
4. Realizzazione dell'App.....	30
4.1. Classi principali.....	31
4.1.1. Model.....	31
4.1.2. DAO	34
4.1.3. View Controllers	35
4.1.4. Storyboard	36
4.2. Image processing	37
4.2.1. Camera Controller.....	38
4.2.2. Bonsai Due e OpenCV	39
4.2.3. Haar Classifier Cascade for Object Detection	40
4.2.4. Processamento dell'immagine	Error! Bookmark not defined.
4.3. Haar training	45

5. Risultati.....	54
5.1. Haar-feature like leavesCascade.xml.....	54
5.2. Tests.....	56
6. Manuale utente	60
6.1. Bonsai List.....	61
6.2. Add New Bonsai.....	63
6.3. My Bonsai	67
6.4. Funzioni avanzate.....	69
6.5. Nota sul nome.....	73
7. Futuri Sviluppi	75
8. Bibliografia.....	77
<i>iOS Apps Developping</i>	77
<i>OpenCV</i>	77
9. Ringraziamenti	79

1. Introduzione

Già qualche anno, fa ispirato dal lavoro di una collega di corso, mi ero interessato all'argomento, iscrivendomi al programma Apple Developpers.

A quell'epoca mi trovavo immerso nello studio e la preparazione del esame finale del corso Reti di Calcolatori, tenuto dal professor Andrea Vitaletti.

Essendomi trovato a utilizzare intensivamente uno strumento di analisi di rete come Wireshark, un noto packet sniffer, mi ero reso conto della grossa utilità che avrebbe comportato la possibilità di implementare tale strumento come un'applicazione per dispositivi mobili. Essendo in possesso di un iPhone, quello sarebbe chiaramente stato il dispositivo mobile, target di una tale applicazione.

Successivamente, le mie ambizioni si sono presto infrante contro l'impossibilità di accedere alle configurazioni di basso livello su un dispositivo con piattaforma iOS. Per poter implementare un packet sniffer infatti, si rende necessario arrivare a manipolare le funzioni di sistema che interagiscono molto vicino al livello hardware in modo da impostare, ad esempio, la modalità passiva del ricevitrasmettitore WiFi. Tali opzioni purtroppo, ad oggi non vengono ancora rese disponibili agli sviluppatori iOS.

Dovendo ripiegare su un'alternativa, l'ipotesi seguente è stata quella legata alla domanda: "Riusciremmo a spostarci più velocemente, se potessimo avere delle informazioni sui flussi del traffico stradale, ottimizzando il percorso fino alla destinazione desiderata tramite un elaborazione delle informazioni real-time oltre a quelle su base statistica?"

Effettivamente, tenendo presente il fatto che ogni dispositivo mobile viene oramai equipaggiato con un apparecchio di rilevamento della posizione GPS, la risposta a tale domanda si basa sulla possibilità di interpolare ogni segnale GPS inviato come un feedback, ad una certa frequenza di tempo, dal dispositivo mobile dell'utente in viaggio. In tal modo sarebbe possibile ricollocare gli utenti - i veicoli in movimento - all'interno di una mappa e soprattutto determinarne i flussi associando i segnali geograficamente vicini fra loro provenienti da dispositivi in movimento.

Questo tipo d'informazioni, unite alle statistiche sul traffico, permetterebbero quindi di valutare il percorso probabilmente più breve nel tempo per raggiungere una destinazione qualsiasi. Inoltre le informazioni in tempo reale potrebbero servire come calibratura di statistiche passate per formularne di nuove e con un'accuratezza maggiore.

Ahimè, purtroppo, nemmeno quest'idea si è rivelata, seppure di fondo genuina, immediatamente traducibile in un'applicazione pratica. Per poter effettuare un elaborazione di così tante informazioni in tempo reale si sarebbe reso indispensabile investire nell'infrastruttura e nel network necessario al funzionamento della parte client.

In più, date le intensioni finali di progetto, non ho potuto non tener conto del livello concorrenziale molto alto che vede coinvolte, mediamente, le App che riguardano la mobilità.

“Waze”, un esempio su tutte, è un’applicazione molto simile, nel concetto, all’idea appena descritta.

Alla fine ho dovuto desistere dal provare a percorrere anche questa strada e di nuovo mi sono ritrovato a caccia di un’idea...

Molto frustrante. Più andavo avanti nella conoscenza dell’objective-C, dell’iOS programming, dei pattern di progettazione GUI dell’iOS Human Interface, e più ero a corto d’idee effettivamente realizzabili e alla portata dei miei mezzi.

Quello che stavo cercando era una buona idea di base, ma che non comportasse grossi oneri, come vastità del sistema lato server o dal punto di vista dell’infrastruttura sottostante. Un’App “stand alone”, utile veramente almeno a qualcuno degli utenti iPhone e altri dispositivi simili, che risultasse compatibile con le limitazioni imposte da Apple riguardanti lo sviluppo di applicazioni su piattaforma iOS.

Da circa cinque anni avevo cominciato ad interessarmi alla coltivazione e manutenzione di alberi bonsai come hobbie.

In particolare sono in possesso di un esemplare della specie Ficus Retusa, comunemente chiamato Ficus Ginseng a causa della similitudine fra il tronco di quest’arbusto e la radice del Ginseng.

Da quando l’ho acquistata non ho mai smesso di combattere contro la caduta repentina del fogliame di questa pianta tropicale così delicata. Spesso cercando le cause di un tanto spiacevole effetto sui più disparati forum esistenti sul Web. A volte facendo delle comparazioni con le fotografie delle foglie

malate che altri utenti avevano pubblicato, altre scattando personalmente delle istantanee e “postandole” a mia volta.

L’ultima volta che mi è capitato di farlo è stato qualche mese fa, così il risultato di tale addizione mi è balzato in testa quasi automaticamente.

Questo è ciò che mi ha portato a decidere di intraprendere il progetto trattato in questa tesi.

Avevo finalmente trovato un’idea adeguata per un’applicazione, senza un grosso overhead determinato dallo sviluppo collegato di un eventuale lato server, né il bisogno di un’infrastruttura di rete o di un cospicuo numero di utenze per il corretto funzionamento.

Soprattutto, cosa di fondamentale importanza, sarebbe valsa come utile per qualcuno degli utenti di dispositivi iOS, almeno per me.

Da qui nasce l’idea “kernel” di un assistente alla gestione dei bonsai come Bonsai Due.

2. Specifiche dell'applicazione

Come il preambolo al nome suggerisce, Bonsai Due è un'applicazione ideata e sviluppata pensando agli

appassionati dell'hobbie del bonsai – termine giapponese costituito da due ideogrammi. Il primo significa contenitore (bon), mentre il secondo (sai) significa educare e, in senso lato, coltivare, crescere.

In realtà più che di hobbie sarebbe il caso di parlare di una vera e propria arte, che risale a tempi lontanissimi nel passato. Inizialmente nata in Cina e perfezionata in Giappone, è legata a quello che gli orientali chiamano "seishi", l'arte di dare una forma, di coltivare e praticare le tecniche più svariate sempre nel rispetto della pianta.

Lo scopo di quest'utility pertanto è, forse, un po' ambizioso.

Il tentativo, infatti, è proprio quello di rendere semplice e alla portata di tutti una delle arti più complesse e antiche conosciute dall'uomo. Chi ha mai provato a cimentarsi in questo, ben sa come sia complicato riuscire anche soltanto a mantenere in vita e in buona salute questi piccoli alberi.

2.1. Cura del bonsai e principali problemi

Molto spesso il problema principale, quando acquistiamo un bonsai, le prime volte, e lo portiamo a casa, è che non abbiamo un'idea chiara di quale pianta sia o della regione da qui originariamente provenga, né del tipo di cure di cui abbia bisogno. Se vada tenuta all'esterno o dentro le nostre abitazioni. Se occorra spostare le piante in base alle temperature e ai periodi dell'anno, o magari in che modo e dove andrebbero poste. A volte ci preoccupiamo tanto perché le foglie cadono, ma non abbiamo tenuto conto dell'autunno. Altre abbiamo a che fare con delle sempreverdi e magari

stiamo sbagliando qualche parametro di un'equazione, spesso abbastanza complessa, che risulta in una pianta in salute e bella, bellissima a vedersi.

2.1.1. Dove e come disporre le piante

Le piante tropicali vanno solitamente tenute all'interno in ambienti chiusi e non troppo areati, in modo da garantire un tasso d'umidità e temperatura sufficientemente alti, ma ben illuminati. In estate, se troppo caldo, dovremo mettere la pianta al riparo da raggi di luce solare diretta, ma sempre mantenendo un alto grado di luminosità dell'ambiente.

Altri piccoli alberi, originari in latitudini più nordiche, staranno meglio all'esterno, mentre è un errore comune considerare tutti i bonsai delle piante da interno solo perché, date le ridotte dimensioni, vi entrano comodamente... Stupendo!

2.1.2. Parassiti e malattie

Quando siamo ormai diventati quasi degli esperti su come e dove tenerli arriva il momento in cui serve il dottore. Alcune piante sono molto delicate, specialmente se in natura vivrebbero in un ambiente dal clima diverso da quello cui si sono adattate. Se il lettore ha mai avuto esperienze simili, saprà con quale affanno spesso, ci si ritrovi a caricare delle fotografie dei nostri bonsai malaticci sui più disparati forum,

in cerca disperata di un aiuto. Aspettando a volte anche dei mesi per ottenere risposta, o non ottenendola affatto, in una similitudine inequivocabile coll'odierno servizio sanitario nazionale.

Tutto tempo utile a svariati parassiti, buono per proliferare, sino a lasciar morire la pianta. Con buona pace dei rivenditori, vivai specializzati, e del mercato dei bonsai.

2.1.3. Manutenzione

Ammettendo finalmente di avere ricevuto, "honoris causa", una laurea in botanica, potremmo ancora aver bisogno di fare annotazioni, immagazzinare dati circa la temperatura, l'umidità, le ore di luce, il numero e lo storico delle potature effettuate, o sulla frequenza delle innaffiature nei diversi periodi dell'anno. O magari semplicemente collezionare fotografie di com'era la pianta, crescendo nel tempo, e tenere traccia di come man mano è andata costituendosi la forma attuale.

2.2. Requisiti di progetto

Questi appena trattati sono alcuni dei motivi per cui, in tale ambito, quest'applicazione potrebbe risultare, forse non indispensabile davvero, ma certamente piuttosto utile.

Di seguito si riportano quali funzionalità sono richieste attualmente per quest'App.

2.2.1. Funzionalità di base

L'applicazione deve come prima cosa costituire una sorta di registro dei bonsai di cui l'utente si prende cura.

Quest'ultimo deve avere quindi la possibilità di inserire i dati necessari per avviare la manutenzione assistita dall'App, che di contro deve poter aiutare l'utente a orientarsi, fornendo informazioni utili riguardo alla pianta che egli ha registrato in precedenza.

Inoltre è richiesta la possibilità di scattare istantanee, tramite applicazione stessa, che l'utente può eventualmente collezionare in un album fotografico relativo a ciascun bonsai.

2.2.2. Funzionalità avanzate

Qualora fosse necessario, l'utente deve poter "domandare" a Bonsai Due se qualcosa non va, nella salute della pianta. L'applicazione quindi, dovrà essere in grado di capirlo. Tramite l'elaborazione dell'immagine che l'utente è chiamato a fornire tramite la funzione di "embedded camera", l'applicazione risponderà – se nell'immagine è presente una foglia – cosa potrebbe essere andato storto nella gestione, o se la pianta è affetta da eventuali malattie o parassiti.

3. Tecnologie utilizzate

Di seguito viene fatta una panoramica sulle due principali tecnologie utilizzate durante lo sviluppo dell'applicazione. Il linguaggio di programmazione Objective-C nella programmazione su piattaforma iOS e, in ultimo, la libreria opensource per la computer vision science, OpenCV.

3.1. Cenni sull'linguaggio Objective-C

Objective C, noto anche come Objective-C, o Obj-C, è un linguaggio di programmazione riflessivo orientato agli oggetti, sviluppato da Brad J. Cox nella prima metà degli anni ottanta.

Come suggerito dal nome, l'Objective C è un sovra insieme proprio del linguaggio C del quale rappresenta un'estensione che supporta il paradigma di programmazione a oggetti. Questa caratteristica arricchisce il modello semantico di C utilizzando un sistema di comunicazione a scambio di messaggi realizzato su impronta della sintassi del linguaggio Smalltalk.

3.1.1. Sintassi

Objective-C mantiene piena compatibilità con i linguaggi C e C++, pertanto all'interno di un programma scritto in tecnologia Objective-C è possibile utilizzare indifferentemente anche porzioni di codice nativo C o C++.

A differenza di C, Objective-C richiede che l'interfaccia (contenente la dichiarazione dei prototipi delle funzioni, delle costanti e delle direttive del preprocessore) e l'implementazione di una classe siano dichiarati in blocchi di codice separati: la prima deve essere inclusa all'interno di un file di header con suffisso ".h" mentre la seconda in un file con suffisso ".m"(che sta per implementation).

Esempio d'interfaccia:

```
#import <Foundation/Foundation.h>

@interface ClassName : SuperClassName
{
    // Instance Variables
    int _myInt;
    float _myFloat;
}

// Declared Properties
@property (nonatomic) NSDate * myDateProperty;
@property (nonatomic, readonly) NSString * myReadOnlyStringProperty;

// Class Methods
+ firstClassMethod;
+ secondClassMethodWithThis: (id)Parameter;

// Instance Methods
- firstInstanceMethod;
- secondInstanceMethodWithOne: (NSString *) Parameter;
- thirdInstanceMethodWithOne: (NSInteger *) ParameterOne andAlsoWithAnother: (NSObject *) ParameterTwo;

@end
```

Il segno - denota i metodi d'istanza, mentre il segno + quelli di classe (analoghi alle funzioni statiche del C++ e ai metodi statici di Java).

3.1.2. Messaggi

Come già detto, Objective-C utilizza un meccanismo particolare per effettuare tutte le operazioni che coinvolgono gli oggetti: il sistema dei messaggi.

Un messaggio è simile alla chiamata di un metodo Java, o di una funzione C, tuttavia in Objective-C, tutte le operazioni che in altri linguaggi a oggetti sono svolte ad esempio da operatori (come l'operatore new in Java) vengono effettuate inviando messaggi a oggetti.

- Message expression
[receiver **method:argument**]
- Message
[receiver **method:argument**]
- Selector
[receiver **method:** argument]

Si dice che un oggetto *object*, la cui classe implementa il metodo *doSomething*, risponde al messaggio *doSomething*.

L'invio del messaggio `doSomething` all'oggetto `object` (che lo riceve) è espresso in questo modo:

[object doSomething];

Mentre l'azione equivalente in C++ o in Java sarebbe espressa da:

object.doSomething();

In questo modo è possibile inviare messaggi ad un oggetto anche se questo non è capace di rispondere. Questo concetto mette in luce un'altra differenza di Objective C rispetto ad altri linguaggi ad oggetti tipizzati staticamente come C++ e Java, nei quali tutte le chiamate devono essere effettuate verso metodi predefiniti.

3.1.3. Tipizzazione dinamica

Objective C è un linguaggio a tipizzazione dinamica o loosely typed poiché posticipa a run-time, invece che a tempo di compilazione, la maggior parte delle azioni di riconoscimento degli oggetti come ad esempio l'appartenenza ad una specifica classe, i metodi e le sue variabili di istanza.

Questo compito è svolto proprio dalla libreria run-time, che viene staticamente "linkata" ad ogni programma Objective-C, agendo come una macchina virtuale in cui "vivono" le istanze Objective-C.

Grazie a questa caratteristica di flessibilità del linguaggio, è possibile dichiarare oggetti di tipo id, un tipo di dato generico simile a void, fornito dal compilatore Objective-C, che può contenere un puntatore a un oggetto di una classe qualsiasi.*

- Dynamically-typed object
`id anObject`
 - Just id
 - Not id * (unless you really, really mean it...)
- Statically-typed object
`Person *anObject`

3.1.4. Objective-C 2.0

Nel 2006 durante la Worldwide Developers Conference (WWDC), Apple ha annunciato l'uscita di Objective-C 2.0: le principali modifiche introdotte sono l'introduzione della garbage collection, alcune migliorie nella sintassi (introduzione delle properties, enumerazione veloce), miglioramento delle prestazioni del run-time e il supporto per i processori a 64-bit.

Le declared properties sono un'espediente sintattico di Objective-C 2.0 che permette di dichiarare esplicitamente quali sono le proprietà delle variabili d'istanza di una classe: queste proprietà permetteranno al programmatore di omettere l'implementazione dei metodi accessori (getters e setters) per le variabili, poiché ne definiscono già implicitamente e in modo non ambiguo il comportamento.

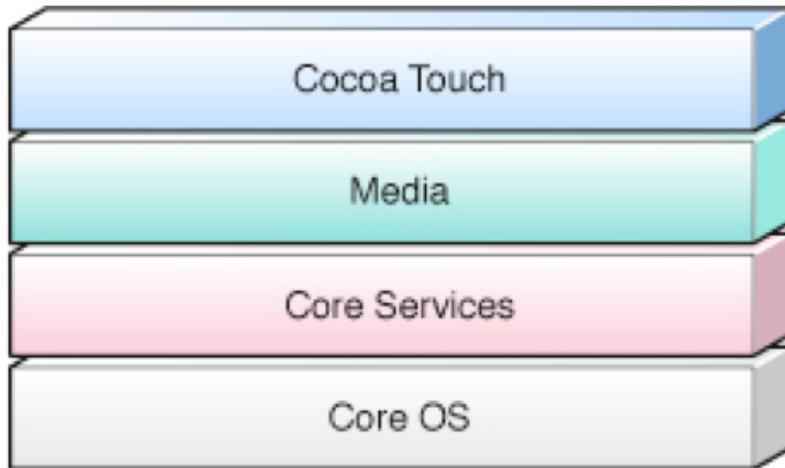
Esse permettono ad esempio di specificare se una variabile ha accesso in sola lettura (non sarà implementato un setter, oppure lettura e scrittura, e specificano le policies di gestione della memoria).

3.2. Panoramica su iOS

iOS (iPhone OS) è un sistema operativo UNIX-based pensato per girare su dispositivi mobili dotati di schermo multitouch (al momento iPhone, iPod Touch e iPad).

E' basato sul kernel XNU di Darwin, il cuore del sistema operativo MacOSX: XNU è un kernel ibrido, risultato dell'unione del codice del microkernel Mach e del kernel monolitico FreeBSD. Su di esso si appoggiano una serie di servizi e API, organizzati per livelli di astrazione.

Nell'analizzarli si è seguito un metodo top-down, considerando l'ottica del programmatore, il quale preferisce trovare soluzioni nei livelli a maggior astrazione piuttosto che lavorare a livello più basso.



La pila degli strati di iOS mostrata alla iPhone SDK Press Conference tenutasi nella sede principale di Apple a Cupertino il 6 marzo del 2008.

3.2.1. Cocoa Touch

Cocoa touch è un set di API scritto principalmente in Objective-C, ottenuto estendendo e modificando Cocoa, la sua versione utilizzata all'interno di OSX. I Framework principali che si trovano all'interno di questo strato sono UIKit e Foundation che supportano direttamente le applicazioni basate su iPhone OS in quanto forniscono le strutture dati per implementare l'interfaccia grafica delle applicazioni e consentono di interagire con i servizi di sistema più ad alto livello. Il primo, UIKit permette la creazione e la gestione delle GUI (campi di testo, buttoni, etichette, font, ecc.), gestisce il ciclo di vita delle applicazioni e gli eventi (ad es. l'interazione touch), si occupa della presentazione di contenuti web e testo, della gestione dei dati, dell'integrazione tra applicazioni e

sistema, del servizio di notifiche Push (servizio che permette di inviare all'utente notifiche relative ad applicazioni installate anche se queste non sono in esecuzione), fornisce supporto di accessibilità e API per interagire ad alto livello con accelerometro, batteria, sensore di prossimità e fotocamera.

Il framework Foundation invece regola i comportamenti di base degli oggetti, stabilisce i meccanismi per la loro gestione e fornisce gli oggetti per i tipi di dato primitivi, collezioni e strutture dati. Contiene la definizione di NSObject, la root class da cui ereditano tutte le classi presenti in Cocoa touch (e in Cocoa), e più in generale tutto quello che non riguarda l'interazione con l'utente o la visualizzazione.

Nonostante Cocoa Touch rappresenti il livello dal più alto grado di astrazione, tipicamente, esiste sempre un metodo UIKit oppure una funzione Foundation corrispondente per una chiamata a basso livello.

Gli altri framework appartenenti al layer Cocoa Touch ma qui non approfonditi sono: MapKit (visualizzazioni di mappe), MessageUI (supporto all'invio di email all'interno delle applicazioni), AddressBookUI (accesso in lettura e scrittura ai contatti della rubrica di iPhone e iPad) e GameKit (connettività peer-to-peer e voip, supporto alle applicazioni per giochi multiplayer).

3.2.2. Media

Il framework e i servizi in questo strato dipendono dai servizi di base (Core Services) e forniscono allo strato Cocoa Touch funzionalità multimediali quali la riproduzione audio e

video o la possibilità di effettuare plotting di grafici 2D e 3D e il rendering di animazioni complesse.

Il framework più importante all'interno dello strato Media è Core Graphics (conosciuto anche come Quartz 2D). Essenzialmente racchiude al suo interno un motore grafico bidimensionale molto leggero e potente, al quale sono demandati compiti quali la creazione e la presentazione dei documenti PDF, il disegno vettoriale, la gestione della trasparenza all'interno delle interfacce grafiche, l'antialiasing, la manipolazione dei colori, rendering di immagini e gradienti.

Altrettanto rilevante è OpenGL ES (OpenGL for Embedded Systems), una versione ultraleggera dell'implementazione dell'engine opensource OpenGL pensata appositamente per dispositivi mobili di piccole dimensione e dalle caratteristiche hardware ridotte come iPhone.

Fanno parte dello strato Media anche iPhone Audio Support e Core Audio (registrazione e processamento degli stream audio) e Media Player Framework.

3.2.3. Core Services

Il layer Core Services contiene al suo interno numerosi framework di sistema.

Core Foundation, che fornisce funzionalità quali la manipolazione dei tipi di dato, gestione dei thread, dei run-loop e socket. Pur essendo un framework C, la maggior parte dei servizi offerti da Core Foundation sono disponibili anche come wrappers Objective-C all'interno del Foundation Framework.

Il framework Core Location invece permette di ottenere la posizione geografica della periferica (latitudine e longitudine): questi dati vengono reperiti tramite il GPS, la connessione WiFi o la triangolazione tra celle (o una combinazione delle tre). Fornisce inoltre supporto all'uso della bussola.

Il framework StoreKit ha il compito di facilitare le transazioni tra le applicazioni e l'App Store. L'iPhone OS 3.0 ha introdotto il concetto di "In App Purchase", ovvero la possibilità da parte dell'utente di acquistare direttamente all'interno dell'applicazione funzionalità aggiuntive o contenuti ulteriori (si pensi alle versioni "pro" di un'applicazione o ad un nuovo livello di un videogioco).

Un altro framework che merita menzione è Core Data, il quale si occupa di fornire API molto semplici per la creazione e l'utilizzo di database basati su SQLite

3.2.4. Core OS

Questo livello contiene il kernel, le facilities per l'allocazione della memoria, il file system, le primitive di rete e di sicurezza, la gestione delle risorse energetiche e una serie di interfacce a basso livello verso l'hardware sottostante (device drivers).

Esso contiene, inoltre, la libreria libSystem, che supporta le specifiche POSIX/BSD 4.4 ed include primitive di sistema per svariati servizi.

Il framework CFNetwork fornisce un'interfaccia C alla pila di protocolli TCP/IP e accesso a basso livello verso i socket locali BSD, External Accessory permette di interagire con accessori esterni connessi fisicamente al dispositivo attraverso il

connettore dock a 30-pin di iPhone, iPhone Touch e iPad o via Bluetooth.

Il Security Framework di iPhone OS fornisce primitive e strutture dati relative alla sicurezza necessarie per un dispositivo connesso a network esterni quali certificati, cifratura, trust policies, portachiavi, ecc.

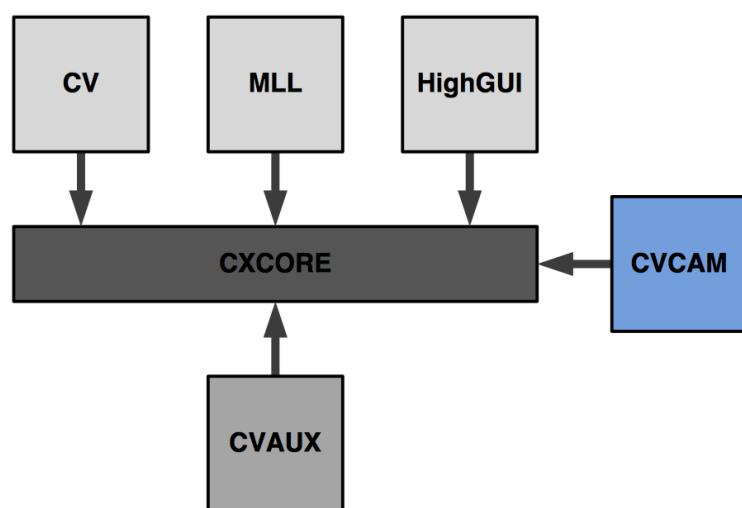
3.3. Libreria OpenCV

Il framework OpenCV nasce da una iniziativa dell'Intel, mentre lavorava su miglioramenti delle loro CPU per applicazioni intensive, ad esempio ray-tracing in tempo reale e proiezione 3D. Uno degli addetti della Intel, aveva notato che in molte università, tra cui il MIT Media Lab, avevano messo su un'infrastruttura per il computer vision, il cui codice era passato da studente a studente. A tal proposito si decise di iniziare, partendo da questo codice, a dar vita a un framework per la computer vision per i processori Intel. Il primo avvio di tale progetto, con la collaborazione di un team Intel russo, fu nel 1999. La prima release ufficiale di OpenCV risalve, invece, al 2006.

La struttura di OpenCV consta delle seguenti 6 parti:

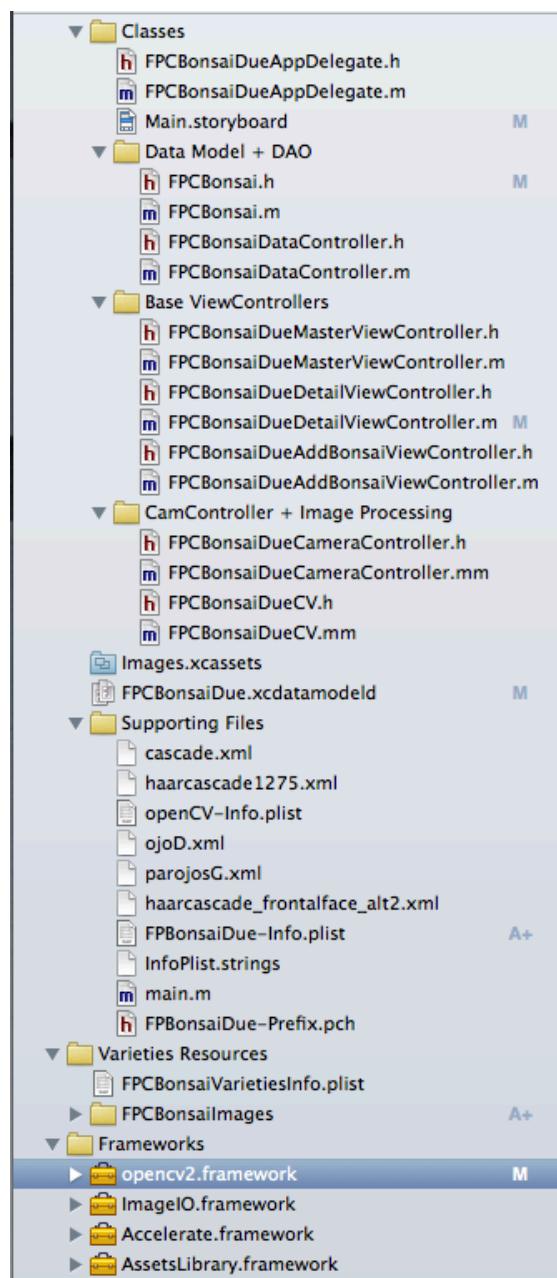
1. *CXCORE: contiene la definizione di tutte le strutture dati e le funzioni per gestire immagini e video.*
2. *CV: contiene tutte le funzioni per il processamento e l'analisi delle immagini, la calibrazione, il tracking e il pattern recognition.*
3. *ML (Machine Learning): contiene molte funzioni sul Machine Learning, quali il clustering e la classificazione.*
4. *HighGUI: contiene le definizioni delle interfacce utenti (GUI).*
5. *CVCAM: contiene le interfacce per le webcam.*
6. *CVAUX: contiene algoritmi sperimentali per scopi diversi, ad esempio: segmentazione, sottrazione del background, modelli HMM, ecc.*

La struttura di OpenCV precedente può essere riassunta nella schematizzazione seguente.



4. Realizzazione dell'App

Struttura del progetto:



4.1. Classi principali

4.1.1. Model

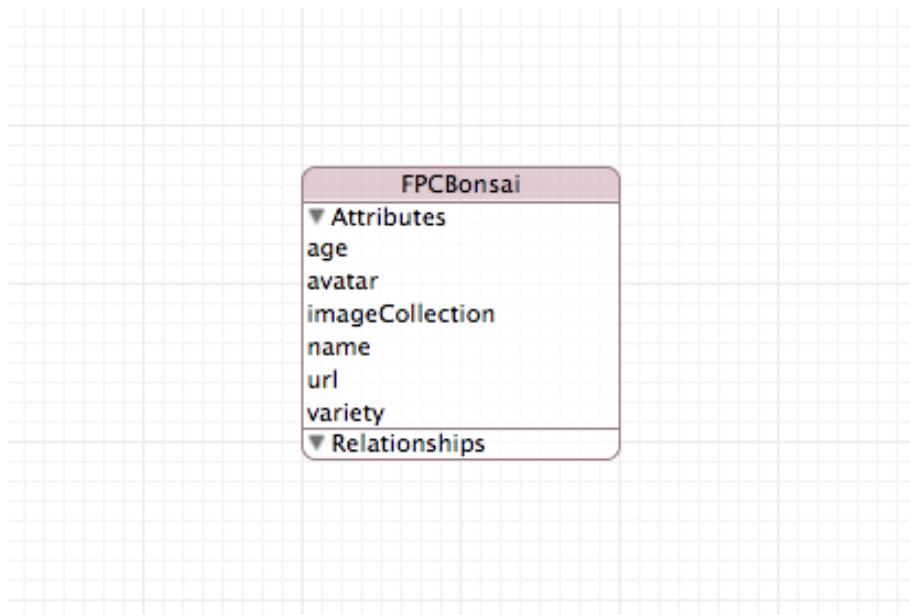
*L'unica “entity” o *modello*, nella terminologia iOS programmino che segue il template di progettazione Model View Controller (MVC), è costituito dalla classe “FPCBonsai”.*

Per distinguerla da un'altra eventuale classe che modelli un'entità omonima, sono state aggiunte tre lettere in maiuscolo come prefisso, in accordo a quanto suggerisce Apple, seguendo la convenzione di tre lettere per le classi “custom”. Per le classi fornite nei framework proprietari se ne utilizzano soltanto due.

NSString o UIImage e CGImage, ad esempio, sono tre classi diverse, anche se le ultime due riguardano lo stesso oggetto. UI, infatti, si riferisce a User Interface, mentre CG sta per Core Graphics. Si tratta comunque di classi che modellano immagini, ma da punti di vista differenti. Per i diversi set di attributi e reattività ai messaggi, a cui le istanze di tali classi rispondono, che sono i metodi.

Il prefisso NS, invece, ha anche un valore storico oltre che puramente identificativo. Infatti, quando il framework Cocoa nacque, tali classi erano utilizzate in applicazioni basate sul sistema operativo NeXTStep. Quando Apple nel 1996 acquisì NeXT, gran parte del suo sistema fu incorporato nel noto OS X, compresi i nomi delle classi preesistenti. Coll'avvento dei sistemi iOS e del framework Cocoa Touch, versione per i dispositivi mobili, molte classi furono rese disponibili direttamente per entrambi gli ambienti. Altre, tuttavia, subirono delle modifiche obbligate ovviamente dalla diversità

dei dispositivi per i quali dovevano servire. Così, per distinguerle, fu introdotto tale sistema di nomenclatura con prefisso.



La classe che modella l'oggetto bonsai ha sei attributi.

```

// FPCBonsai.h
// FPCBonsaiDue
//
// Created by Federico Paliotta on 04/10/13.
// Copyright (c) 2013 Federico Paliotta. All rights reserved.
//

#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>

@interface FPCBonsai : NSObject

@property (nonatomic, copy) NSString * name;
@property (nonatomic, copy) NSString * variety;
@property (nonatomic) NSInteger age;
@property (nonatomic, copy) UIImage * avatar;
@property (nonatomic, copy) NSURL *url;
@property (nonatomic, copy) NSMutableDictionary * imagesCollection;

- (id)initWithName:(NSString *)name variety: (NSString *)variety age:(NSInteger)years;
- (id)initWithName:(NSString *)name variety: (NSString *)variety age:(NSInteger)years image:(UIImage *) avatar;
- (id)initWithDictionary:(NSDictionary *)data;

@end

```

Nome, età e varietà di pianta, ma anche un'immagine “avatar” che viene assegnata in fase di inizializzazione dell'oggetto in base alla varietà scelta dall'utente.

Inoltre, un URL, nel quale è salvato l'indirizzo web della pagina wikipedia rispettiva alla varietà del bonsai, e un dizionario che serve per immagazzinare una collezione di coppie <key, value>, dove il valore è costituito da un'immagine scattata dall'utente in un determinato momento nel tempo, la chiave.

I metodi esplicitamente dichiarati per questa classe sono tre inizializzatori, di cui uno è il metodo inizializzatore designato. I metodi getter e setter sono definiti implicitamente grazie all'utilizzo delle “properties” con le quali sono stati dichiarati gli attributi.

4.1.2. DAO

“FPCBonsaiDueDataController”, come si può intuire dal nome, è la classe che funge da Data Access Object (DAO) e nel pattern di progettazione MVC assume il ruolo di Controller. Questa classe serve alla mediazione fra i dati e i View Controller che si occupano di formattare ulteriormente i dati che sono poi presentati all’utente nelle rispettive View.

```
#import <Foundation/Foundation.h>

@class FPCBonsai;

@interface FPCBonsaiDataController : NSObject

@property (nonatomic, copy) NSMutableArray *masterFPCBonsaiList;
@property (nonatomic, readonly) NSMutableArray *varietiesList;

- (NSUInteger)countOfList;
- (FPCBonsai *)objectInListAtIndex:(NSUInteger)theIndex;
- (void)addFPCBonsai:(FPCBonsai *)aFPCBonsai;

- (UIImage *)varietyAvatarImage:(NSString *)imageFileName;
- (NSArray *)varietiesArrayList:(NSString *)propertyListFileName;

@end
```

Nel suo file d’implementazione inoltre, è stata definita un’estensione per la classe, nella quale è dichiarato un metodo ausiliario. Tale metodo si occupa di inizializzare la “masterFPCBonsaiList” un oggetto della classe NSMutableArray che mantiene l’elenco dei bonsai gestiti dall’applicazione e che viene poi passato al ViewController della vista principale dell’App, quella dal titolo Bonsai List.

```

- (NSUInteger)countOfList{
    return [self.masterFPCBonsaiList count];
}

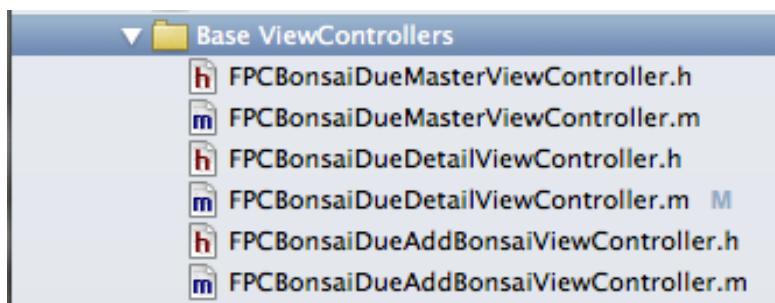
- (FPCBonsai *)objectInListAtIndex:(NSUInteger)theIndex{
    return [self.masterFPCBonsaiList objectAtIndex:theIndex];
}

- (void)addFPCBonsai:(FPCBonsai *)aFPCBonsai{
    [self.masterFPCBonsaiList addObject:aFPCBonsai];
}

```

4.1.3. View Controllers

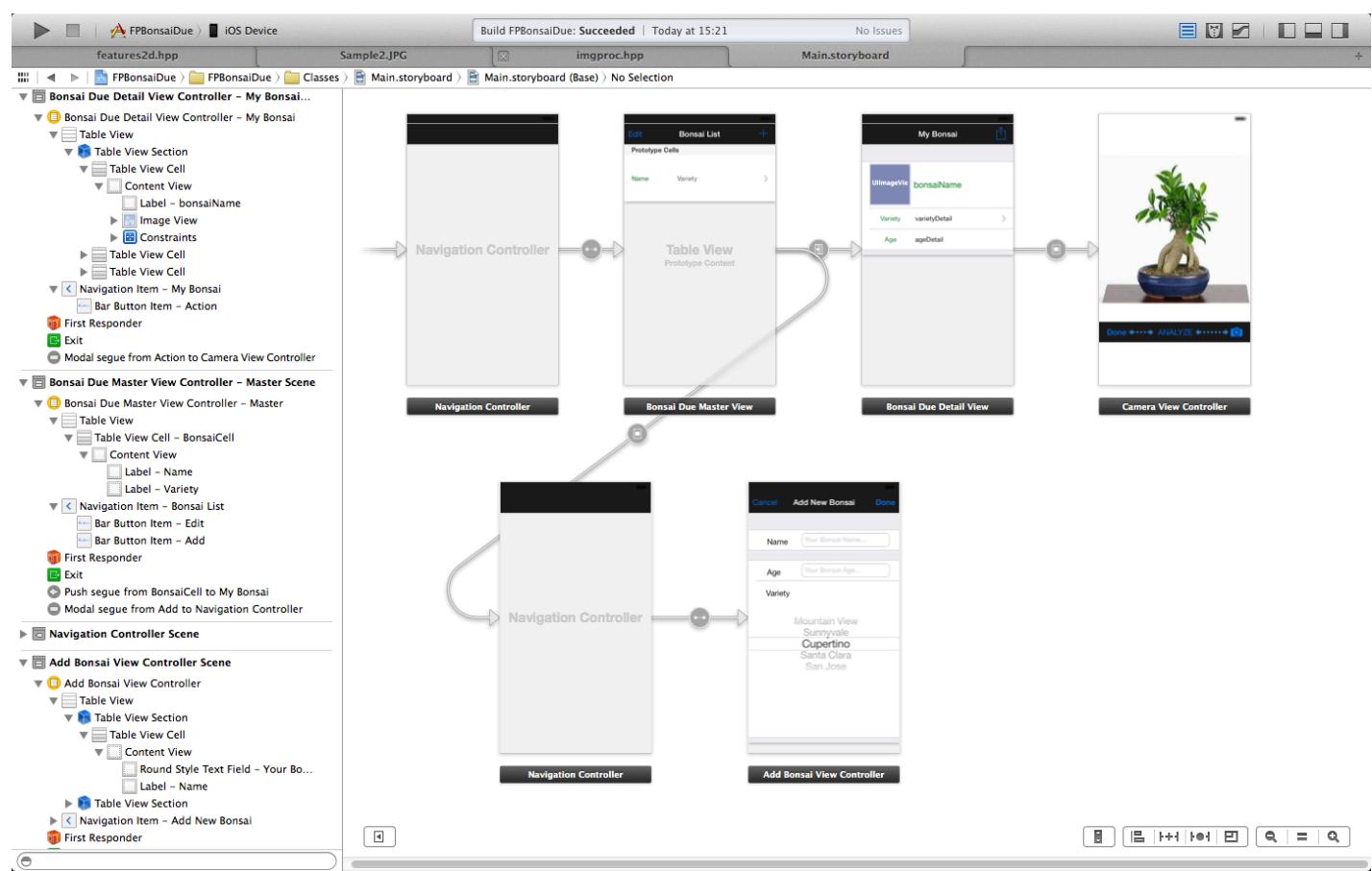
Altre classi riguardano invece i ViewController che gestiscono le viste nell'applicazione – le schermate – e contengono il codice da eseguire in occasione delle varie interazioni dell'utente sull'App.



La classe “FPCBonsaiDueMasterViewController”, nei suoi file di header e implementation, si occupa di gestire la view principale. “FPCBonsaiDueDetailViewController” riguarda quella dei dettagli, mentre quando l’utente si troverà di fronte alla schermata per aggiungere un nuovo bonsai, questa sarà controllata dalla classe “FPCBonsaiDueAddBonsaiViewController”.

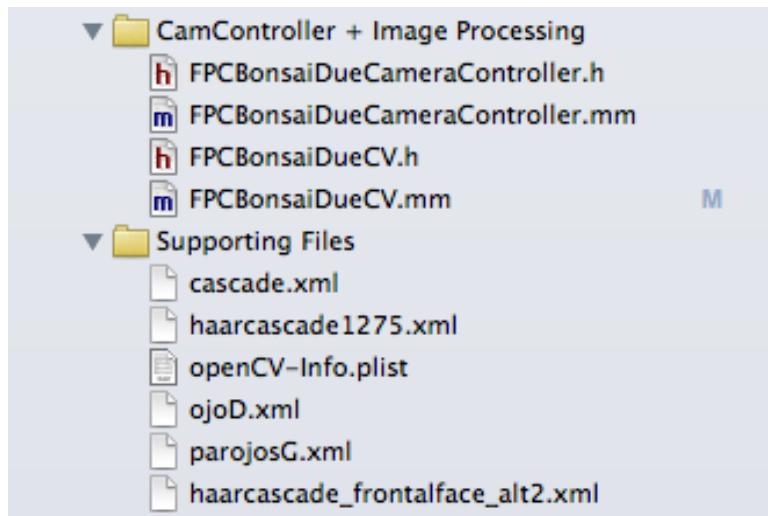
4.1.4. Storyboard

Per rendere un’idea di com’è stata strutturata l’applicazione, dal punto di vista dell’interfaccia grafica, di seguito si riporta il file “Main.storyboard”, costruito tramite lo strumento Interface Builder integrato in Xcode.



4.2. Image processing

Per quanto riguarda la parte di processamento immagini, le classi e i file in questione sono “FPCBonsaiDueCV”, “FPCBonsaiDueCameraController” e qualche xml.



4.2.1. Camera Controller

La seconda menzionata è semplicemente la classe associata al ViewController che gestisce la schermata più a destra, nella figura di pagina precedente.

Da notare, tuttavia, che la chiamata al metodo che eseguirà l'elaborazione dell'immagine è assegnata a un nuovo thread. In questo modo l'utente può continuare a interagire con la parte dell'applicazione che viene schedulata sul main thread, anche quando un elaborazione è in corso.

```

- (void)doCalculation
{
    dispatch_queue_t backgroundQueue = dispatch_queue_create("processing_images_queue", 0);

    // E stacca un thread!
    dispatch_async(backgroundQueue,
    ^{
        //UIImage *grayImg = [self.cv toGrayscale: self.imageView.image];
        dispatch_async(dispatch_get_main_queue(),
        ^{
            [self.imageView setImage:[self.cv bgRemove:self.imageView.image]];
            //[[self.imageView setImage:grayImg];
            //self.processingImageIndicator.color = [UIColor colorWithRed:0 green:255 blue:0 alpha:1];

            [self.processingImageIndicator stopAnimating];
            self.processingImageIndicator.hidden = YES;

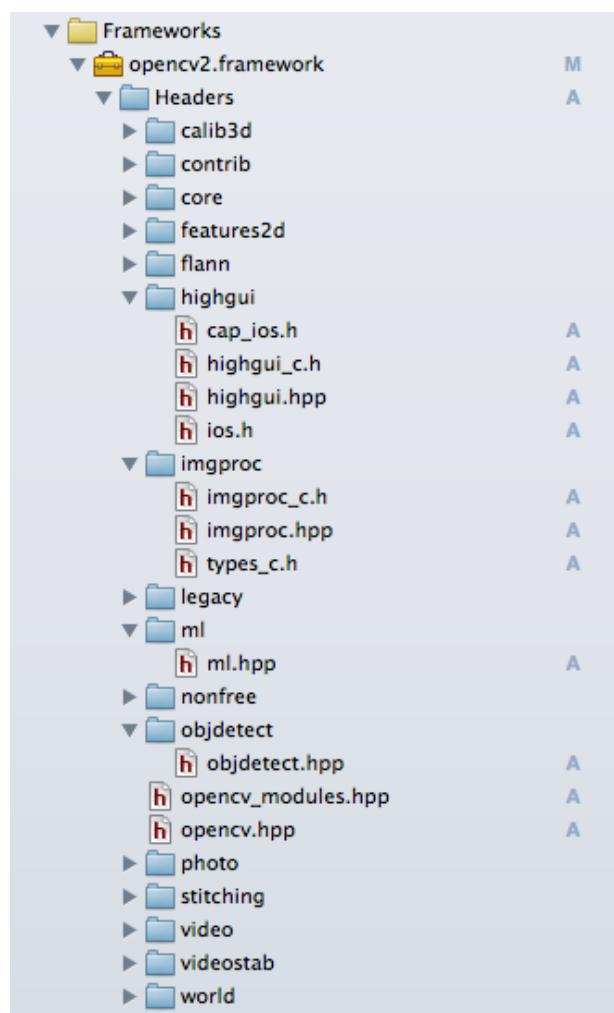
        });
    });
}

```

4.2.2. Bonsai Due e OpenCV

“FPCBonsaiDueCV” è invece una classe particolare rispetto alle altre. Il file d’implementazione porta l’estensione “.mm” che indica che il sorgente contenuto non è solo Objective-C.

In questa classe infatti, sono definiti metodi che poggiano sul framework opencv basato sull’omonima libreria, scritta prevalentemente in C++.



4.2.3. Haar Classifier Cascade for Object Detection

Nell'implementazione viene dapprima chiamato il metodo `viewDidLoad`, nel quale sono impartite le impostazioni per la

videocamera e, in ultimo, si trova la chiamata load per il classifier cascade.

```
//  
// FPCBonsaiDueCV.m  
// FPCBonsaiDue  
//  
// Created by Federico Paliotta on 13/10/13.  
// Copyright (c) 2013 Federico Paliotta. All rights reserved.  
  
#import "FPCBonsaiDueCV.h"  
#import <opencv2/opencv.hpp>  
#import <Foundation/Foundation.h>  
#import <CoreGraphics/CoreGraphics.h>  
  
NSString* const CascadeFilename = @"cascade";  
const int HaarOptions = CV_HAAR_FIND_BIGGEST_OBJECT | CV_HAAR_DO_ROUGH_SEARCH ;  
  
@implementation FPCBonsaiDueCV  
  
-(void)viewDidLoad  
{  
    [super viewDidLoad];  
  
    self.videoCamera = [[CvVideoCamera alloc] initWithParentView:self.imageView];  
    self.videoCamera.defaultAVCaptureDevicePosition = AVCaptureDevicePositionBack;  
    self.videoCamera.defaultAVCaptureSessionPreset = AVCaptureSessionPreset352x288;  
    self.videoCamera.defaultAVCaptureVideoOrientation = AVCaptureVideoOrientationPortrait;  
    self.videoCamera.defaultFPS = 30;  
    self.videoCamera.grayscaleMode = NO;  
    self.videoCamera.delegate = self;  
  
    NSString *CascadePath =[[NSBundle mainBundle] pathForResource:CascadeFilename ofType:@"xml"];  
    cascade.load( [CascadePath UTF8String]);  
}
```

Quando viene invocata la chiamata al metodo load, il file xml ottenuto durante il training del cascade è caricato in memoria.

A questo punto nel metodo processImage, quando invocato, è presente la chiamata alla funzione detectMultiScale definita nell'header file objdetect.hpp.

```

#pragma mark - Protocol CvVideoCameraDelegate

#endif __cplusplus
-(void)processImage:(cv::Mat &)image
{
    Mat greyscaleFrame;
    cv::cvtColor(image, greyscaleFrame, CV_BGR2GRAY);
    cv::equalizeHist(greyscaleFrame, greyscaleFrame);

    std::vector<cv::Rect> faces;

    //Haar-feaature like object detection
    cascade.detectMultiScale(greyscaleFrame, faces, 1.1, 2, HaarOptions, cv::Size(30, 80));

    for(int i = 0; i < faces.size(); i++)
    {
        cv::Point pt1(faces[i].x + faces[i].width , faces[i].y + faces[i].height);
        cv::Point pt2(faces[i].x, faces[i].y);

        //cv::rectangle(image, pt1, pt2, cvScalar(0, 255, 0, 0), 1, 8, 0);

        cv::Point cntr((faces[i].x + (faces[i].width)/2), faces[i].y + (faces[i].height/2));
        //cv::circle(image, cntr, (faces[i].x + faces[i].width)/3, cvScalar(0, 255, 0, 0), 1, 8, 0);

        cv::Size axes(2*(faces[i].height)/3, (faces[i].width)/2);
        cv::ellipse(image, cntr, axes, 90, 360, 0, cvScalar(0, 255, 0, 0), 1, 8, 0);
    }
}
#endif

```

Tale funzione rileva gli oggetti di differenti grandezze nell'immagine di input. Gli oggetti trovati sono ritornati come una lista di rettangoli e accetta i seguenti parametri:

- **cascade** - l'Haar classifier cascade, il file xml o yaml caricato tramite load()
- **image** - una matrice in formato CV_U8 che contiene l'immagine dove si deve rilevare la presenza dell'oggetto
- **objects** - un vettore di rettangoli dove ognuno di essi contiene l'oggetto trovato
- **scaleFactor** - che specifica quanto deve essere ridimensionata ad ogni passaggio l'immagine
- **minNeighbors** - quanti vicini dovrebbe avere ogni candidato selezionato
- **minSize** - la minore dimensione possibile per l'oggetto
- **maxSize** - la maggiore dimensione possibile per l'oggetto

(oggetti con taglie rispettivamente minore e maggiore di minSize e maxSize sono ignorati)

4.2.4. Processamento dell'immagine

Sempre nel file d'implementazione della classe FPCBonsaiDueCV è stato definito il metodo bgRemove, utile a scontornare la foglia, in primo piano, dal resto dell'immagine.

```

#pragma mark BACKGROUND REMOVAL

#endif __cplusplus
-(UIImage *)bgRemove:(UIImage *) image
{
    cv::Mat present = [self cvMatFromUIImage:image];
    cv::Mat src = [self cvMatGrayFromUIImage:image];
    cv::Mat foreground;
    vector< vector <cv::Point>> contours;

    threshold(src,foreground,179,255,THRESH_BINARY);
    medianBlur(foreground,foreground,9);
    erode(foreground,foreground,cv::Mat());
    dilate(foreground,foreground,cv::Mat());

    findContours(foreground, contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);
    drawContours(present, contours, -1, CV_RGB(0, 0, 255), 16);

    return [self UIImageFromCVMat:present];

}
#endif

```

In questo metodo le funzioni più importanti sono `findContours` e `drawContours`, definite nel file di header `imgproc.hpp`, sono deputate rispettivamente a rilevare i contorni dell'immagine data input, opportunamente convertita in oggetto `Mat`, e a disegnarli mettendoli in evidenza.

4.3. Haar training

Durante la prima fase, essendomi occupato di vedere inizialmente come funzionasse l'object detection nelle immagini, avevo utilizzato un noto classificatore in cascata haar-feature like per il riconoscimento del volto, haarcascade_frontalface_alt2.xml, e solo in seguito ho iniziato a studiare il modo di creare uno apposta per il rilevamento delle foglie.



In seguito, seguendo un famoso tutorial sul web¹ sono riuscito a creare un classificatore a cascata, haar-feature like, che fosse orientato al conoscimento degli oggetti desiderati in questa trattazione, le foglie.

Come prima cosa ho dovuto collezionare gli esempi positivi e negativi (97 e 2380 per l'esattezza) – immagini qualsiasi questi ultimi, ma che non devono assolutamente contenere l'oggetto cercato.

Tramite la funzione opencv_createsample, invece, si provvede a creare i file .vec, che serviranno nella fase di training, formattando correttamente le immagini che contengono gli esempi positivi.

```
$ createsamples -img face.png -num 10 -bg negatives.dat -vec samples.vec -maxxangle 0.6 -  
maxyangle 0 -maxzangle 0.3 -maxidev 100 -bgcolor 0 -bgthresh 0 -w 20 -h 20
```

Finalmente, quando si è ripetuto il procedimento per le immagini volute, si è pronti per passare montagne di ore nella fase di training vera e propria.

¹ <http://note.sonots.com/SciSoftware/haartraining.html>

```

MacBook-Pro-di-Federico-Paliotta:Haar_Training_Imp Aleph$ opencv_traincascade -data traincascade -vec samples/samples1501z.vec -bg ./negative.dat -numPos 6 -numNeg 2000 -numStages 12 -featureType LBP -w
80 -h 80 -bt DAB -minHitRate 0.999 -maxFalseAlarmRate 0.6
PARAMETERS:
cascadeDirName: traincascade
vecFileName: samples/samples1501z.vec
bgFileName: ./negative.dat
numPos: 6
numNeg: 2000
numStages: 12
precalcValBufSize[Mb] : 256
precalcIdxBufSize[Mb] : 256
stageType: BOOST
featureType: LBP
sampleWidth: 80
sampleHeight: 80
boostType: DAB
minHitRate: 0.999
maxFalseAlarmRate: 0.6
weightTrimRate: 0.95
maxDepth: 1
maxWeakCount: 100

===== TRAINING 0-stage =====
<BEGIN
POS count : consumed 6 : 6
NEG count : acceptanceRatio 2000 : 1

```

Come si riesce a capire da questo screenshot, dopo quasi due ore e trenta minuti di tempo CPU (in realtà il tempo effettivamente trascorso è anche di più), il processo di training si trova ancora allo stage iniziale. Attualmente, il classifier cascade che è in uso dall'applicazione per il riconoscimento delle foglie, è stato ottenuto arrivando all'undicesimo stage di training.

Di seguito, si riporta un log del training effettuato.

```
MacBook-Pro-di-Federico-Paliotta:Haar_Training_Imp Aleph$ opencv_traincascade -data traincascade -vec samples/samples1501z.vec -bg ./negative.dat -numPos 30 -numNeg 300  
-numStages 11 -featureType HAAR -w 80 -h 80 -bt LB -minHitRate 0.999 -maxFalseAlarmRate 0.5  
-baseFormatSave  
  
PARAMETERS:  
cascadeDirName: traincascade  
vecFileName: samples/samples1501z.vec  
bgFileName: ./negative.dat  
numPos: 30  
numNeg: 300  
numStages: 16  
precalcValBufSize[Mb] : 256  
precalcIdxBufSize[Mb] : 256  
stageType: BOOST  
featureType: HAAR  
sampleWidth: 80  
sampleHeight: 80  
boostType: LB  
minHitRate: 0.999  
maxFalseAlarmRate: 0.5  
weightTrimRate: 0.95  
maxDepth: 1  
maxWeakCount: 100  
mode: BASIC  
  
===== TRAINING 0-stage =====  
<BEGIN  
POS count : consumed 30 : 30  
NEG count : acceptanceRatio 300 : 1  
Precalculation time: 10  
+-----+-----+  
| N | HR | FA |  
+-----+-----+  
| 1| 1| 1|  
+-----+-----+  
| 2| 1| 0.0633333|  
+-----+-----+  
END>  
  
===== TRAINING 1-stage =====  
<BEGIN  
POS count : consumed 30 : 30  
NEG count : acceptanceRatio 300 : 0.239425  
Precalculation time: 18  
+-----+-----+  
| N | HR | FA |  
+-----+-----+  
| 1| 1| 1|  
+-----+-----+  
| 2| 1| 1|  
+-----+-----+  
| 3| 1| 1|  
+-----+-----+  
| 4| 1| 1|  
+-----+-----+  
| 5| 1| 1|  
+-----+-----+  
| 6| 1| 1|
```

```

+-----+
| 7|   1|   1|
+-----+
| 8|   1|   1|
+-----+
| 9|   1|   1|
+-----+
| 10|   1|   1|
+-----+
| 11|   1| 0.92|
+-----+
| 12|   1| 0.92|
+-----+
| 13|   1| 0.926667|
+-----+
| 14|   1| 0.726667|
+-----+
| 15|   1| 0.586667|
+-----+
| 16|   1| 0.653333|
+-----+
| 17|   1| 0.673333|
+-----+
| 18|   1| 0.48|
+-----+
END>

```

```

===== TRAINING 2-stage =====
<BEGIN
POS count : consumed 30 : 30
NEG count : acceptanceRatio 300 : 0.171723
Precalculation time: 9
+-----+
| N | HR | FA |
+-----+
| 1|   1|   1|
+-----+
| 2|   1|   1|
+-----+
| 3|   1|   1|
+-----+
| 4|   1|   1|
+-----+
| 5|   1|   1|
+-----+
| 6|   1| 0.113333|
+-----+
END>

```

```

===== TRAINING 3-stage =====
<BEGIN
POS count : consumed 30 : 30
NEG count : acceptanceRatio 300 : 0.0806235
Precalculation time: 9
+-----+
| N | HR | FA |
+-----+
| 1|   1|   1|
+-----+
| 2|   1|   1|
+-----+
| 3|   1|   1|
+-----+
| 4|   1|   1|
+-----+

```

```

| 5| 1| 1|
+---+-----+
| 6| 1| 0.55|
+---+-----+
| 7| 1| 0.553333|
+---+-----+
| 8| 1| 0.26|
+---+-----+
END>

===== TRAINING 4-stage =====
<BEGIN
POS count : consumed 30 : 30
NEG count : acceptanceRatio 300 : 0.0314895
Precalculation time: 9
+---+-----+
| N | HR | FA |
+---+-----+
| 1| 1| 1|
+---+-----+
| 2| 1| 1|
+---+-----+
| 3| 1| 1|
+---+-----+
| 4| 1| 1|
+---+-----+
| 5| 1| 1|
+---+-----+
| 6| 1| 0.466667|
+---+-----+
END>

===== TRAINING 5-stage =====
<BEGIN
POS count : consumed 30 : 30
NEG count : acceptanceRatio 300 : 0.0245902
Precalculation time: 8
+---+-----+
| N | HR | FA |
+---+-----+
| 1| 1| 1|
+---+-----+
| 2| 1| 1|
+---+-----+
| 3| 1| 1|
+---+-----+
| 4| 1| 1|
+---+-----+
| 5| 1| 0.5|
+---+-----+
END>

===== TRAINING 6-stage =====
<BEGIN
POS count : consumed 30 : 30
NEG count : acceptanceRatio 300 : 0.012088
Precalculation time: 8
+---+-----+
| N | HR | FA |
+---+-----+
| 1| 1| 1|
+---+-----+
| 2| 1| 1|
+---+-----+
| 3| 1| 1|

```

```

+-----+
| 4|   1|   1|
+-----+
| 5|   1|   1|
+-----+
| 6|   1|   1|
+-----+
| 7|   1|   1|
+-----+
| 8|   1| 0.936667|
+-----+
| 9|   1|  0.52|
+-----+
| 10|   1| 0.216667|
+-----+
END>

===== TRAINING 7-stage =====
<BEGIN
POS count : consumed 30 : 30
NEG count : acceptanceRatio 300 : 0.00439123
Precalculation time: 8
+-----+
| N | HR | FA |
+-----+
| 1|   1|   1|
+-----+
| 2|   1|   1|
+-----+
| 3|   1|   1|
+-----+
| 4|   1|   1|
+-----+
| 5|   1|   1|
+-----+
| 6|   1|  0.06|
+-----+
END>

===== TRAINING 8-stage =====
<BEGIN
POS count : consumed 30 : 30
NEG count : acceptanceRatio 300 : 0.000478015
Precalculation time: 9
+-----+
| N | HR | FA |
+-----+
| 1|   1|   1|
+-----+
| 2|   1|   1|
+-----+
| 3|   1|   1|
+-----+
| 4|   1|   1|
+-----+
| 5|   1| 0.446667|
+-----+
END>

===== TRAINING 9-stage =====
<BEGIN
POS count : consumed 30 : 30
NEG count : acceptanceRatio 300 : 0.00023649
Precalculation time: 9
+-----+

```

```

| N | HR | FA |
+---+-----+-----+
| 1| 1| 1|
+---+-----+-----+
| 2| 1| 1|
+---+-----+-----+
| 3| 1| 1|
+---+-----+-----+
| 4| 1| 0.51|
+---+-----+-----+
| 5| 1| 0.51|
+---+-----+-----+
| 6| 1| 0.07|
+---+-----+-----+
END>

===== TRAINING 10-stage =====
<BEGIN
POS count : consumed 30 : 30
NEG count : acceptanceRatio 300 : 4.50311e-05
Precalculation time: 8
+---+-----+-----+
| N | HR | FA |
+---+-----+-----+
| 1| 1| 1|
+---+-----+-----+
| 2| 1| 1|
+---+-----+-----+
| 3| 1| 1|
+---+-----+-----+
| 4| 1| 1|
+---+-----+-----+
| 5| 1| 1|
+---+-----+-----+
| 6| 1| 1|
+---+-----+-----+
| 7| 1| 0.976667|
+---+-----+-----+
| 8| 1| 0.976667|
+---+-----+-----+
| 9| 1| 0.99|
+---+-----+-----+
| 10| 1| 0.606667|
+---+-----+-----+
| 11| 1| 0.606667|
+---+-----+-----+
| 12| 1| 0.616667|
+---+-----+-----+
| 13| 1| 0.606667|
+---+-----+-----+
| 14| 1| 0|
+---+-----+-----+
END>

===== TRAINING 11-stage =====
<BEGIN
POS count : consumed 30 : 30
NEG count : acceptanceRatio 300 : 3.88939e-06
Required leaf false alarm rate achieved. Branch training terminated.

```


5. Risultati

5.1. Haar-feature like leavesCascade.xml

Dopo svariati giorni di piccoli passi avanti e grandi frustrazioni sono riuscito finalmente nel creare il mio classificatore a cascata haar-feature like, impacchettandolo in un file xml.

A seguire un estratto del file xml.

```
1  <?xml version="1.0"?>
2  <opencv_storage>
3  <haarcascade type_id="opencv-haar-classifier">
4    <size>
5      80 80</size>
6    <stages>
7      <_>
8        <trees>
9          <_>
10         <feature>
11           <rects>
12             <_>
13               28 30 2 10 -1.</_>
14             <_>
15               29 30 1 10 2.</_></rects>
16             <tilted>0</tilted></feature>
17             <threshold>-1.9172584870830178e-04</threshold>
18             <left_val>3.8095238804817200e-01</left_val>
19             <right_val>-1.9305555820465088e+00</right_val></_></_>
20           <_>
21           <_>
22           <feature>
23             <rects>
24               <_>
25                 2 22 78 53 -1.</_>
26               <_>
27                 28 22 26 53 3.</_></rects>
28             <tilted>0</tilted></feature>
29             <threshold>-3.5390490293502800e-01</threshold>
30             <left_val>2.5030303001403809e+00</left_val>
31             <right_val>-1.0376452207565308e+00</right_val></_></_></trees>
32           <stage_threshold>-6.5669280290603638e-01</stage_threshold>
33         <parent>-1</parent>
34         <next>-1</next></_>
35       <_>
36       <trees>
37         <_>
38         <_>
39         <feature>
40           <rects>
41             <_>
42               20 17 4 30 -1.</_>
43             <_>
44               20 17 2 15 2.</_>
45             <_>
46               22 32 2 15 2.</_></rects>
47             <tilted>0</tilted></feature>
48             <threshold>-3.3544981852173805e-03</threshold>
49             <left_val>-1.8426229953765869e+00</left_val>
50             <right_val>8.7999999523162842e-01</right_val></_></_>
51           <_>
52         <_>
53       <_>
```

///

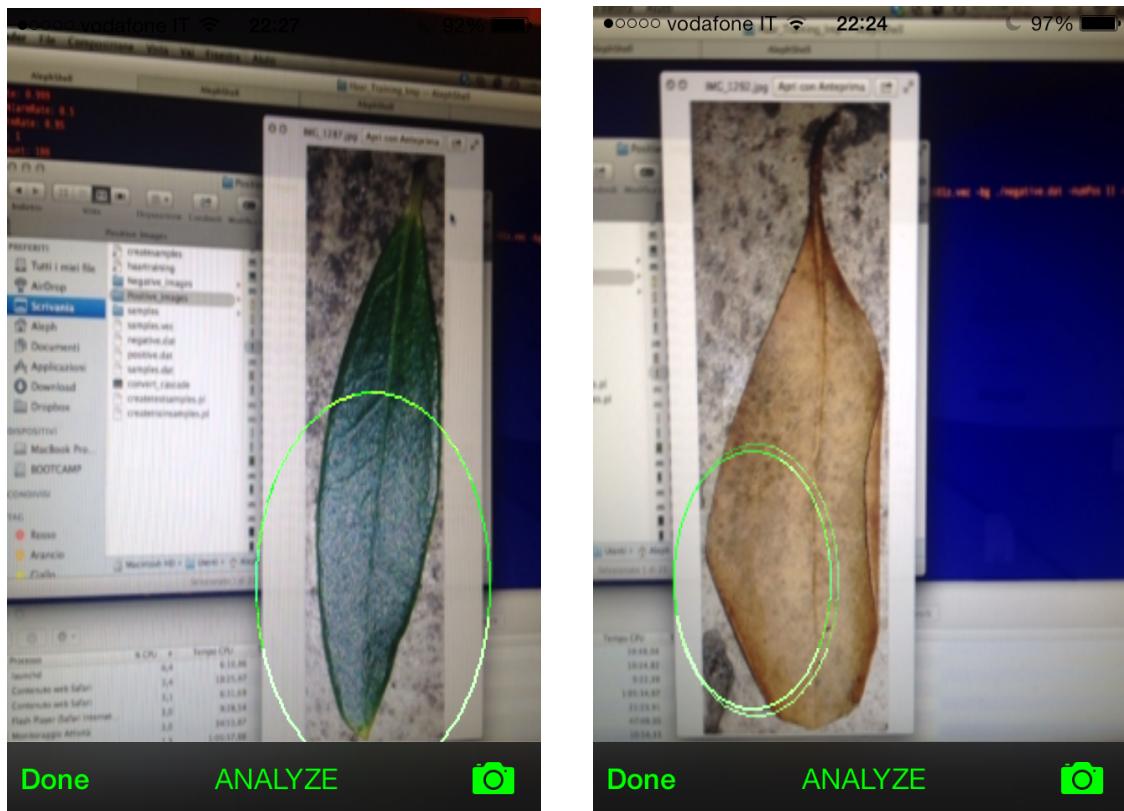
```
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959

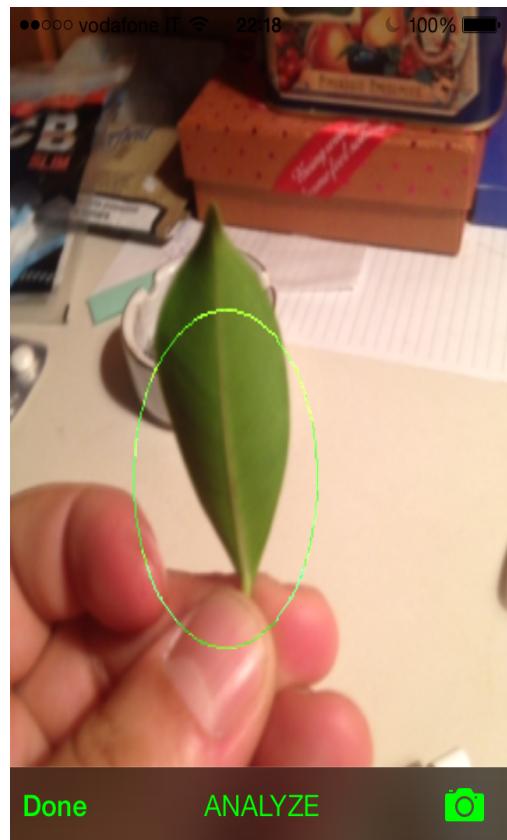
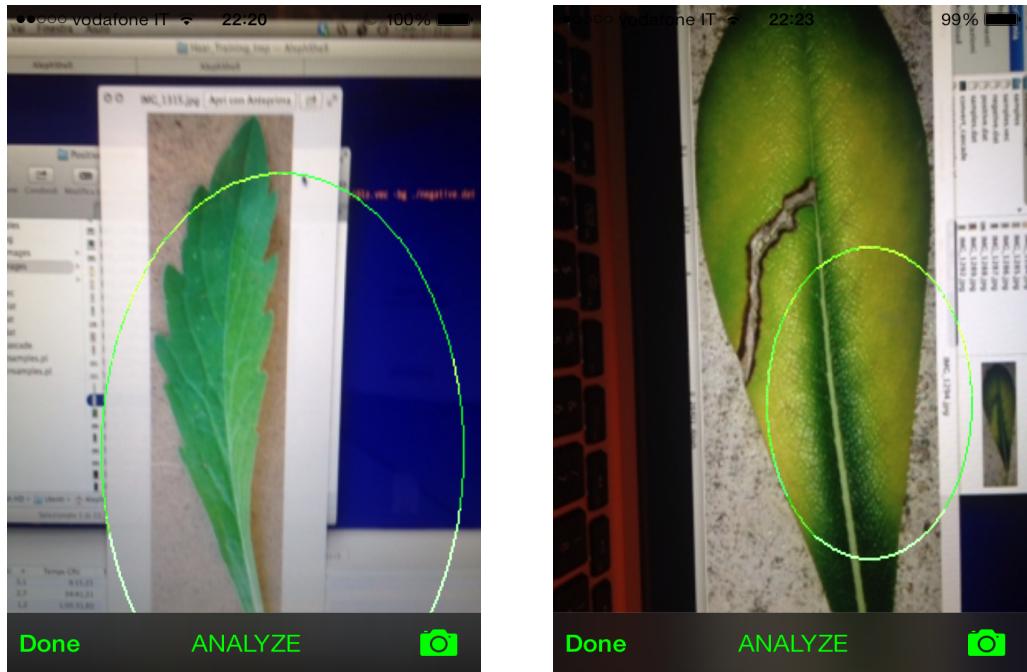
    <feature>
        <rects>
            <_>
                39 33 2 18 -1.</_>
            <_>
                39 42 2 9 2.</_></rects>
        <tilted>0</tilted></feature>
        <threshold>2.2750105927116238e-05</threshold>
        <left_val>6.5732628107070923e-01</left_val>
        <right_val>-1.4627612829208374e+00</right_val></_></_>
    <_>
    <_>
        <feature>
            <rects>
                <_>
                    12 71 1 6 -1.</_>
                <_>
                    12 74 1 3 2.</_></rects>
            <tilted>0</tilted></feature>
            <threshold>7.6147273648530245e-04</threshold>
            <left_val>-3.0567952990531921e-01</left_val>
            <right_val>10.</right_val></_></_>
        <_>
        <_>
            <feature>
                <rects>
                    <_>
                        0 64 62 10 -1.</_>
                    <_>
                        0 69 62 5 2.</_></rects>
            <tilted>0</tilted></feature>
            <threshold>-6.5560098798394203e-03</threshold>
            <left_val>1.2184363441467285e+00</left_val>
            <right_val>-6.4077866077423096e-01</right_val></_></_></trees>
        <stage_threshold>-1.8642107248306274e+00</stage_threshold>
        <parent>8</parent>
        <next>-1</next></_></stages>
    </haarcascade>
</opencv_storage>
```

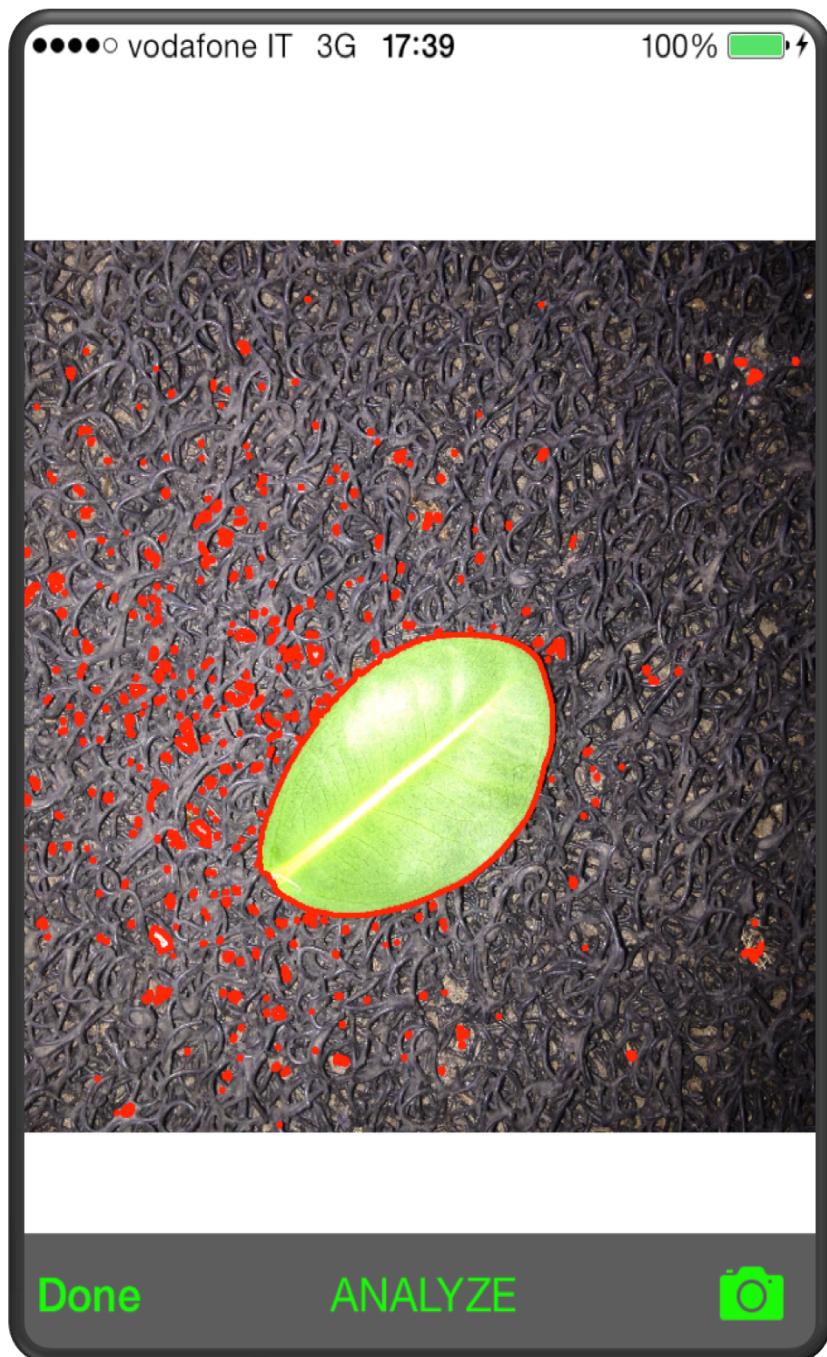
Il file xml in cui è salvato il classificatore è lungo circa 960 righe per un training a undici stages, cominciato il 31 Ottobre alle ore 20:24 e terminato alle ore 13:18 del giorno successivo.

5.2. Tests

Anche se il riconoscimento oggetti non è ancora accuratissimo, si può affermare che funzioni sufficientemente bene. Specialmente se l'oggetto appare chiaramente in primo piano e sullo sfondo non vi sono elementi che possano confondere il classificatore. Questo effettivamente è valido anche per quanto riguarda la funzione di rimozione dallo sfondo descritta precedentemente (*bgRemoval*).







Done

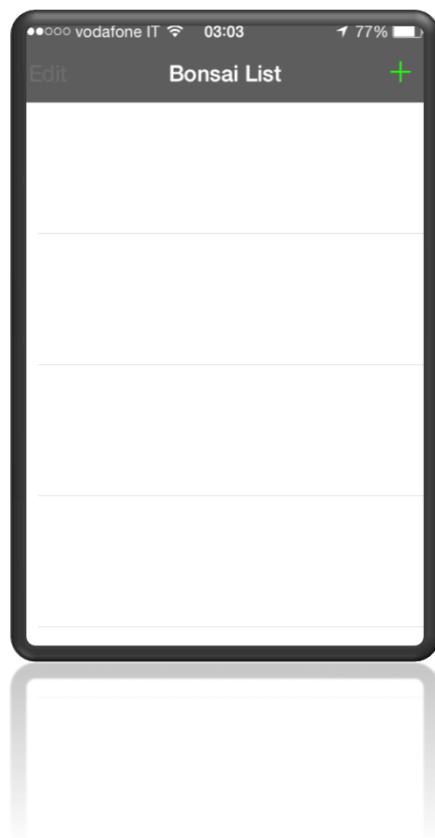
ANALYZE



6. 6. 6.

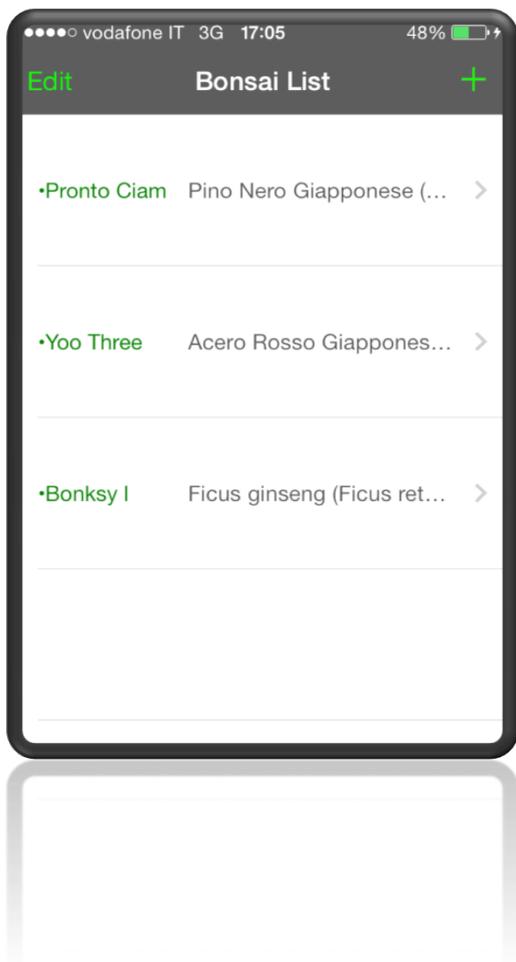
6. Manuale utente

Inizialmente, quando l'utente avvia, toccando con un dito l'icona dell'App, si troverà di fronte un elenco di tutti i bonsai di cui si prende cura. Se quella fosse la prima volta che venisse utilizzata l'applicazione, ovviamente l'elenco non conterebbe alcuna informazione, ma sarebbe sempre possibile, toccando il bottone '+' in alto a destra, aggiungere una nuova pianta all'elenco.



6.1. Bonsai List

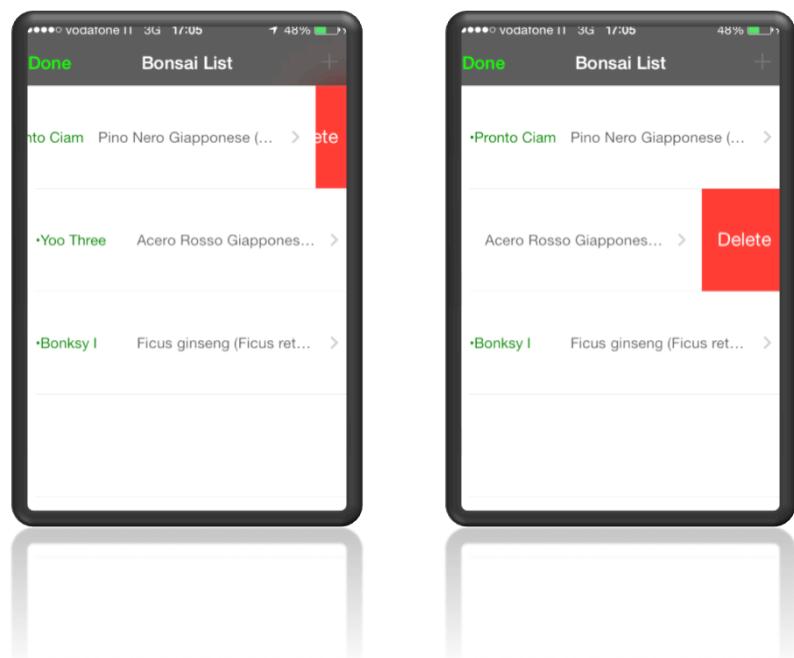
Su ogni riga della tabella in questione, si trovano due campi.



In primis, a sinistra, il nome che abbiamo scelto per identificare il bonsai e, di fianco, il tipo di varietà cui afferisce pianta originale.

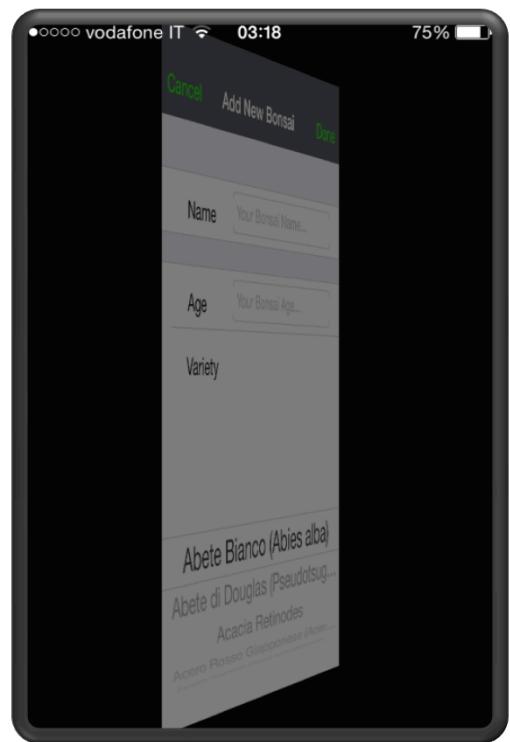
A questo punto il bottone 'Edit' verrà reso disponibile, rendendo così possibile la cancellazione, dall'elenco, della

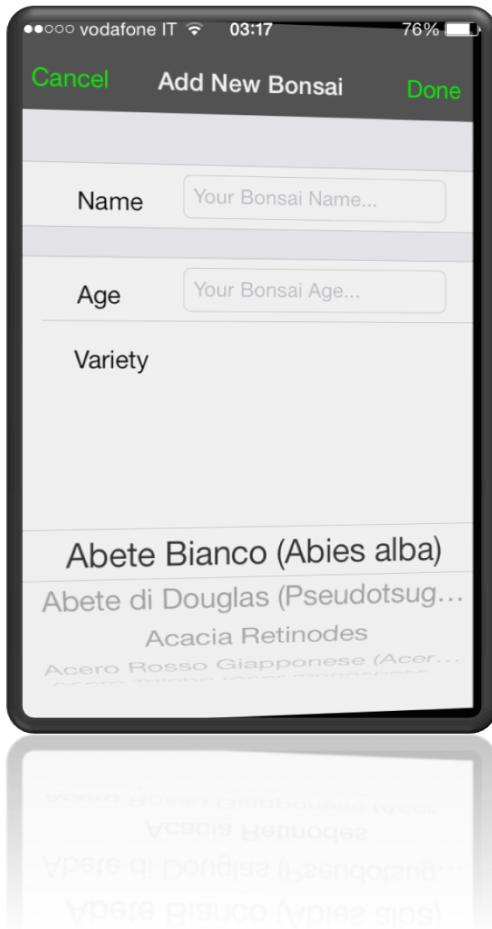
riga corrispondente a quel bonsai. Inoltre, per aumentare il grado di consistenza con le altre applicazioni e il sistema in generale, nel senso inteso dall'iOS programming, è sempre possibile passare nella modalità di editing della tabella performando la gestione di “swipe” sulla cella – che è una riga dell’elenco – ossia scorrendo con un solo dito parallelamente da destra verso sinistra. In questo modo la entry e tutto il suo contenuto slitterà coerentemente nella direzione sopradetta e apparirà un bottone, riempiente il vuoto creato, fornendo così un modo ulteriore di eliminazione del contenuto selezionato. Premuto tale bottone si presenterà all’utente un “action sheet”, in maniera modale – vale a dire dal fondo della schermata verso l’alto – con la scelta di annullare o confermare la cancellazione. Se l’elenco è vuoto, il tasto descritto prima è visibile ma disabilitato. E ragionevolmente, la modalità di editing non è disponibile.



6.2. Add New Bonsai

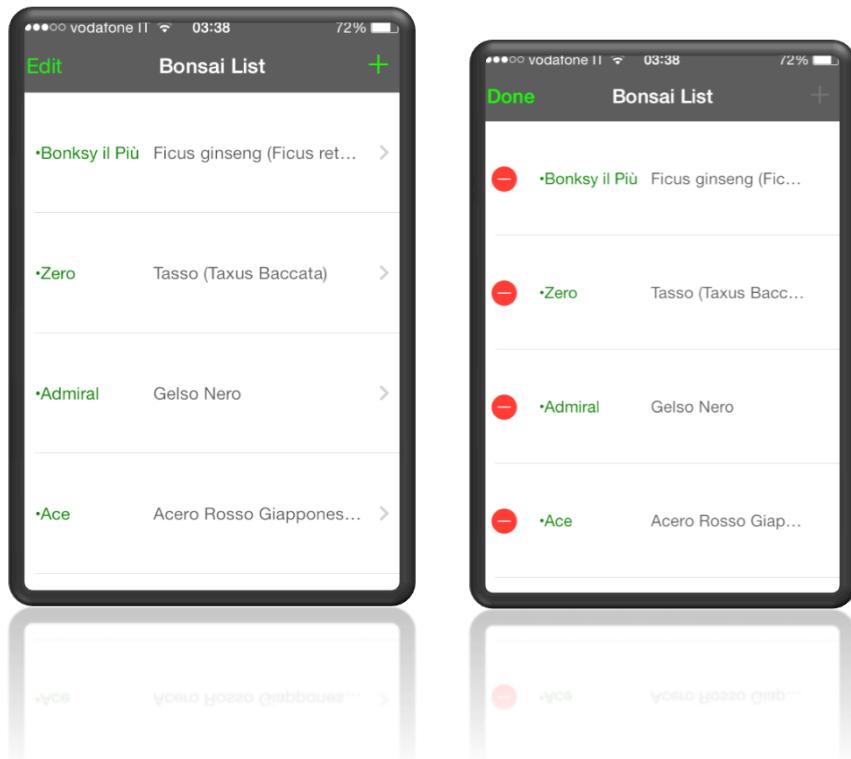
Quando il nostro utente vorrà inserire una nuova pianta nel sistema di gestione guidata, non dovrà fare altro che pigiare sul tasto di aggiunta. A quel punto la schermata iniziale ruoterà, di 180° rispetto al suo asse verticale, presentando la sua faccia posteriore. Questa scelta progettuale è anch'essa in linea coi dettami e i pattern di progettazione applicazioni suggeriti da Apple. Il motivo alla sua base è quello di aumentare la percezione di fisicità che l'utente riscontra durante la sua esperienza d'interazione col dispositivo, dando quanto più possibile la sensazione di maneggiare uno strumento materiale e tangibile come può esserlo un taccuino o una sezione di uno schedario.





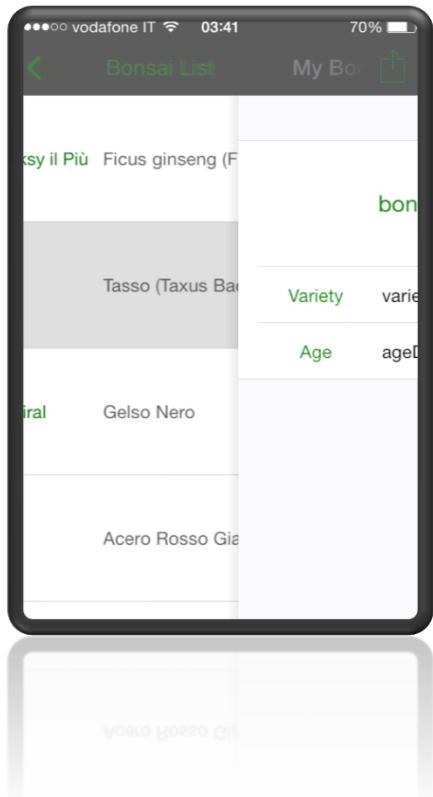
Sulla schermata di aggiunta si forniranno le informazioni iniziali necessarie nei rispettivi campi. Il nome con il quale vogliamo riferirci al nuovo bonsai di cui cominciamo a prenderci cura, l'età della pianta in questione – informazione utile per regolare i parametri di gestione della stessa – e il tipo di varietà. Quest'ultima verrà fornita selezionandone una fra quelle attualmente supportate nell'App, mentre nome ed età saranno scritti manualmente e non sarà, per scelta, più possibile modificare i valori selezionati.

Premendo il tasto 'Done' si innescherà una rotazione della schermata uguale è contraria alla precedente che riporterà in primo piano la schermata iniziale, ma con in più una nuova entry in cima a quelle già esistenti che corrisponde alla pianta appena aggiunta. È stato assunto che dei nuovi bonsai assumano maggiore rilevanza rispetto ai precedenti. Quindi, per aumentare la qualità dell'esperienza utente, sono aggiunti in ordine LIFO (Last In First Out), cosicché l'interazione coll'App sia ancora più efficiente e sbrigativa. Evitando che, non appena aggiunte le informazioni sulla nuova pianta, si renda necessario scorrere sino in fondo tutto l'elenco per poterle reperire.

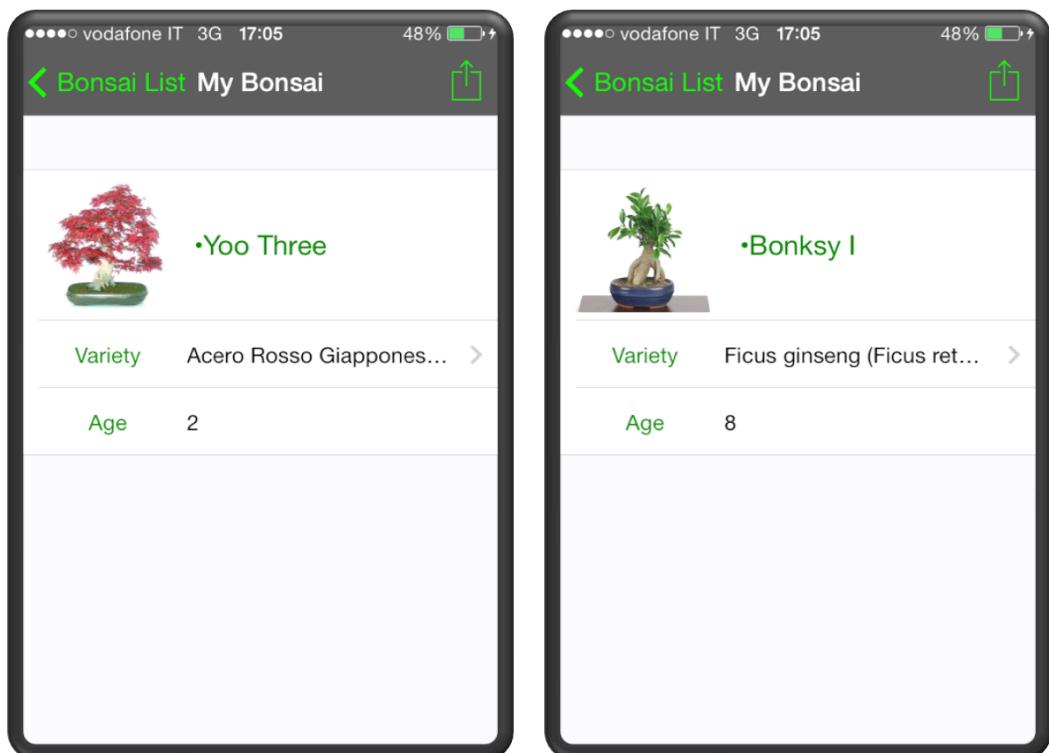


6.3. My Bonsai

Facendo "tap" su un'entry della tabella, nella schermata iniziale, si potrà accedere alla sezione dettagliata che riguarda il bonsai selezionato. Nella schermata successiva l'utente avrà dunque la possibilità di visualizzare altre informazioni utili sulla pianta e la transizione fra le due schermate avviene in modalità "push". Questo, come sempre, nel rispetto della logica che regola l'utilizzo delle applicazioni iOS che garantisce semplicità e rende intuitivo, da parte degli utenti, capire come meglio avvantaggiarsi di tutte le funzioni dell'App.



Nella schermata in dettaglio, nella parte superiore sinistra, si trova un'immagine thumbnail che corrisponde alla varietà di albero bonsai. Al suo fianco, alla stessa altezza, il nome per esteso della nostra pianta e, al di sotto, il tipo e l'età.



Premendo sulla riga che contiene la descrizione delle varietà, si arriverà alla pagina wiki che descrive il tipo di pianta, potendo facilmente apprendere molte cose utili e interessanti a riguardo.

Mentre l'età dell'albero, dal momento in cui è stata fornita dall'utente, viene aggiornata di anno in anno.

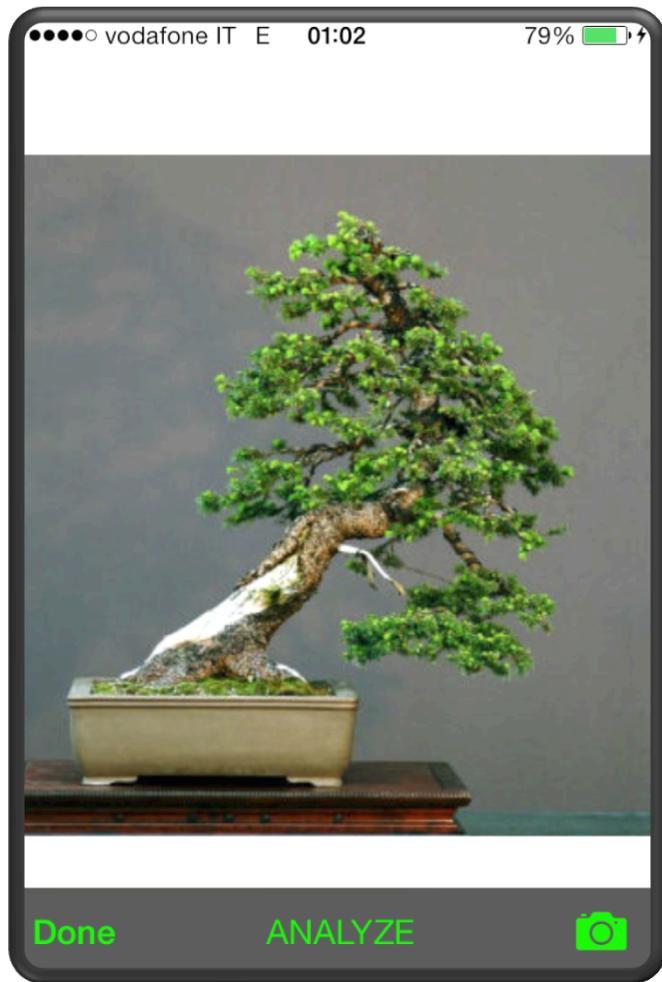
Nella barra di navigazione alla sommità dello schermo, notiamo il titolo della schermata – “My Bonsai” – che è cambiato da quella precedente, e due pulsanti.

Quello più a sinistra, se premuto, ha come effetto lo slittamento a destra della sezione dettagli e il conseguente ritorno alla “Bonsai List”, sotto gli occhi dell’utente.

Il secondo pulsante  serve invece per accedere alle funzioni più avanzate dell’applicazione.

6.4. Funzioni avanzate

Se l’utente vorrà, pigiando sul bottone appena descritto, causerà la presentazione di un’ulteriore schermata, in maniera modale, sul cui sfondo è posta inizialmente l’immagine thumbnail precedentemente menzionata, ma in scala adeguata.



In quest'ultima sezione dell'App sarà possibile scattare delle fotografie da aggiungere alla collezione per il nostro bonsai.

Nello schermo in basso, una barra degli strumenti contiene tre pulsanti. 'Done' anche in questo caso, causa il ritorno alla

schermata iniziale “Bonsai List”, mentre nell’angolo destro il tasto  permette di accedere all’utilizzo della video camera per effettuare gli scatti desiderati. Una volta messo a fuoco e fermata l’immagine avremo la possibilità di utilizzarla o catturarne un’altra.

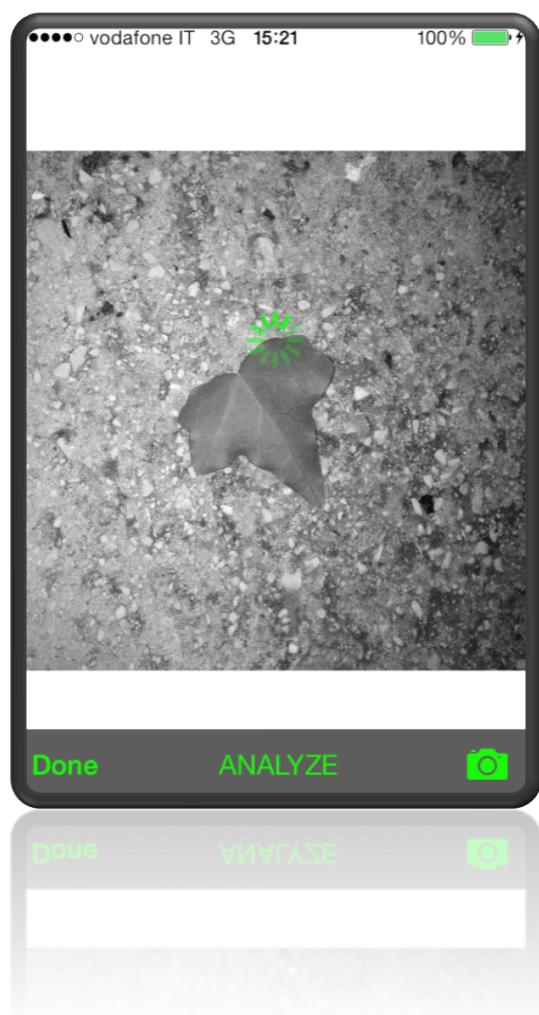
Sarebbe possibile, tal volta, decidere di scattare una fotografia di una foglia per effettuarne un’analisi. La funzionalità più all’avanguardia, e insieme nucleo, di quest’App infatti, è proprio questa.

Se l’utente riscontra dei problemi nella manutenzione della pianta, potrà trarre un’enorme vantaggio “chiedendo” direttamente a Bonsai Due, cos’ha che non va il proprio bonsai.

Se una pianta ha delle carenze, o sta subendo un attacco da parte di qualche parassita, l’indice principale dei sintomi che ne conseguono si trova proprio nelle foglie. Troppo chiare o che ingialliscono, che presentano delle macchioline, o sono più o meno accartocciate. Dalle foglie, facendo un’elaborazione dell’immagine, è possibile determinare con buona approssimazione, nei casi più comuni, che cosa sta tediando la pianta cui appartengono.

In tal modo l’utente non dovrà far altro, dopo aver scattato una foto della foglia, che toccare il tasto ‘ANALIZE’ al centro della barra degli strumenti. A quel punto l’applicazione avvierà l’elaborazione, tenendo conto di alcuni dati forniti dall’utente in precedenza come il tipo di pianta, e ricavati dal sistema come la data e quindi in che periodo dell’anno ci si trovi, o il clima in un determinato range di tempo.

Dopo pochi istanti, all'utente, verrà presentato il responso e una serie eventuale di suggerimenti su come correre ai ripari e curare al meglio il suo caro bonsai.



6.5. Nota sul nome

Magari qualcuno, leggendolo, si sarà chiesto se questa fosse una seconda versione dell'applicazione precedente. In un certo senso sì. Quando ero agli inizi della progettazione creai un novo progetto Xcode chiamandolo "Bonsai". Dopo qualche tempo, a causa di alcune variazioni sostanziali alle scelte di progetto, invece di modificare l'esistente, mi sembrò più semplice e veloce creare, ex novo, un secondo progetto. Lo chiamai quindi Bonsai Due, pensando che da ultimo avrei trovato un nome più esplicativo per un'applicazione del genere.

Quello che accadde poi, procedendo nella lettura di centinaia di pagine fra manuali di programmazione obj-C ed iOS, framework references e commenti sui forum in cerca di soluzioni ai problemi riscontrati, tutti rigorosamente in inglese, fu abbastanza divertente. Mi resi conto che, effettivamente, Bonsai Due non poteva essere più appropriato. Infatti, tradotto dall'inglese, "due" significa adatto, adeguato, programmato per, ma anche doveroso. Così decisi di mantenere esattamente "Bonsai Due" come nome dell'App, come a dire, indispensabile!

Ho sviluppato quest'App coll'intento di renderla uno strumento "indispensabile" a chiunque si voglia avvicinare alla conoscenza delle tecniche del seishi in modo semplice, facilitando la curva d'apprendimento, e assistito, cercando di farlo quasi sembrare, spero, una sciocchezza: un vecchio gioco tascabile degli anni novanta. Un Tamagotchi.

7. Futuri Sviluppi

Sicuramente, in futuro, vorrei perfezionare la parte legata all'object detection – e più in generale all'utilizzo della libreria openCV – ma ci sono alcune funzionalità che andrebbero implementate, come il calendario della coltivazione e il sistema che manda avvisi all'utente quando è necessario. Altre funzioni potrebbero certamente essere migliorate e anche l'applicazione in generale, prima di poterla distribuire, dovrà superare molti test per migliorarne le prestazioni.

Delle icone adatte saranno l'oggetto di studio della fase successiva, mentre contemporaneamente punterò ad internazionalizzare l'applicazione aggiungendo altri tipi di linguaggi supportati oltre l'inglese e l'italiano.

Di seguito, ancora, si potrà pensare di estendere l'App agli altri dispositivi come iPad, iPad Mini o iPhone5.

Tuttavia, conseguite le prossime tappe, è mia intensione sottoporla alla convalida di Apple e in tal modo, venderla sull'Apple Store.

Speriamo.

8. Bibliografia

iOS Apps Developping

- Start Developping iOS Apps Today, © 2013 Apple Inc.
- Programming with Objective-C, © 2013 Apple Inc.
- iOS App Programming Guide, © 2013 Apple Inc.
- iOS Technology Overview, © 2013 Apple Inc.
- iOS Human Interface Guidelines, © 2013 Apple Inc.
- App Distribution Guide, © 2013 Apple Inc.
- Concepts in Objective-C Programming, © 2013 Apple Inc.
- Cocoa Core Competences, © 2013 Apple Inc.
- View Programming Guide for OS, © 2013 Apple Inc.
- View Controller Programming Guide for iOS, © 2013 Apple Inc.
- Table View Programming Guide for iOS, © 2013 Apple Inc.
- Camera Programming Topics for iOS, © 2013 Apple Inc.

OpenCV

- OpenCV 2.4.7.0 Documentation.
- OpenCV User's Guide.
- Learning OpenCV, *Gary Bradski & Adrian Kaehler*, © 2008 O'Reilly Media Inc.
- Face And Facial Feature Detection Evaluation: Performance Evaluation of Public Domain Haar Detectors for Face and Facial Feature Detection, *M. Castrillo 'n-Santana, O. De'niz-Sua'rez, L. Anto'n-Canal'is and J. Lorenzo-Navarro*, Institute of Intelligent Systems and Numerical Applications in Engineering.

- Rapid Object Detection Using a Boosted Cascade of Simple Features, *Paul A. Viola, Michael J. Jones*, 2004 Mitsubishi Electric Research Laboratories.
- Learning A Face Detector. Learning A Pose Estimator, *Paul A. Viola, Michael J. Jones*, 2001 Robust Real-Time Face Detection IEEE International Conference on Computer Vision.

9. Ringraziamenti

Ringrazio mio Padre e mia Madre, che hanno continuato a darmi fiducia.

La mia Famiglia tutta e in particolar modo mia Nonna Anna, che non ha mai smesso di chiamarmi insistentemente ogni volta che era pronto il pranzo.

In fine vorrei ringraziare il mio collega e amico Bruno, che mi ha dato degli ottimi suggerimenti, nonché è stato di grande aiuto e supporto, anche morale, in questi ultimi mesi.

FINE