UNIVERSITA' DEGLI STUDI DI PADOVA

Corso di Laurea Magistrale in Ingegneria Elettronica

Ottimizzazione delle prestazioni real time dell'algoritmo SURF di OpenCV su piattaforma i.MX51 tramite NEON

Relatore:

Chiar.mo Prof. Lorenzo Vangelista

Candidato:

Marco Di Vivo IL\607340

A. A. 2010/2011

INDICE

Ai miei genitori

e al mio amore

Introduzione

Attualmente la parola “sistema embedded” viene utilizzata per tantissimi dispositivi presenti sul mercato. Innanzitutto, bisogna fare un po’ di chiarezza e chiedersi : che cos’è un sistema embedded?

Un sistema embedded è un dispositivo sviluppato e programmato opportunamente per eseguire un numero limitato di operazioni e funzioni. A volte la peculiarità di un sistema embedded è quella di avere specifiche *real-time*, ossia di svolgere tutti i suoi compiti entro dei tempi (detti anche *deadline),* prestabiliti. Queste caratteristiche mettono in risalto una prima differenza con i cosiddetti “general-purpose system”: per esempio un PC è un sistema molto flessibile che ci permette di effettuare numerosissime operazioni senza alcun vincolo temporale sulla durata di ciascuna di esse. Uno dei motivi principali che hanno affermato l’utilizzo dei sistemi embedded sul mercato è la possibilità di realizzarli come *system-on-chip* (SoC). Infatti, realizzando un SoC si riesce ad ottenere oltre alla riduzione delle dimensioni fisiche del dispositivo ed un consumo minore in termini di potenza, anche la produzione di massa dello stesso con conseguente abbattimento del prezzo di mercato. Ovviamente ciascuno di questi chip viene programmato in modo che svolga un compito preciso in modo tale che, mettendo insieme più dispositivi di questo tipo (detti anche special purpose), si dia vita ad un'architettura più complessa e sofisticata. Un esempio lampante di sistema complesso contenente più dispositivi embedded è il *modem wi-fi*: esso al suo interno è composto da più chip, ciascuno dei quali implementa una determinata funzione(ad esempio funzioni di networking, power managment, ecc.), ed essi sono in grado di comunicare tra loro e scambiarsi delle informazioni utili per completare la funzione principale che il dispositivo complessivo deve realizzare.

In questo lavoro di tesi è stata sviluppata un’applicazione che permette il riconoscimento di un’immagine all’interno di una fotografia scattata, ad esempio, da un visitatore di un museo, di un palazzo antico oppure di una semplice mostra. L’idea nasce dalla possibilità di voler fornire, nel momento in cui viene riconosciuto l’oggetto, delle informazioni caratteristiche riguardo all’anno in cui è stato creato, l’autore, il contesto storico e tutte le tipiche informazioni che si forniscono a dei visitatori di mostre o esposizioni artistiche.

L'applicazione utilizza gli algoritmi e le funzioni generali per l'analisi ed elaborazione delle immagini contenuti nella libreria OpenCV, sviluppata dalla Intel ma attualmente sotto licenza *free BSD*. In particolare tra tutti gli algoritmi contenuti nella suddetta libreria, si è scelto di utilizzare le potenzialità nell'estrazione delle *features* di un'immagine dell'algoritmo SURF (Speeded Up Robust Features), sia in termini di robustezza che in termini di calcolo computazionale. Lo scopo principale di questa tesi non è stato quello di testare le potenzialità già ben note in letteratura del SURF[3], ma bensì quello di valutare le sue prestazioni su un particolare dispositivo embedded. Infatti, negli ultimi due anni sono stati effettuati alcuni porting della libreria OpenCV sia su architettura multiprocessore[1] che su *smart cameras*[2]. Si tenga però presente che in entrambe i target utilizzati, la componente di Digital Signal Processing (DSP), svolge un ruolo predominante. Infatti, basti notare che nella [1] viene utilizzato il *Cell Broadband Engine*®, il quale è stato sviluppato principalmente per applicazioni di tipo grafico, non a caso risulta essere l'attuale core della piattaforma Playstation 3.

Figura I.1: La console Playstation 3 di casa Sony.

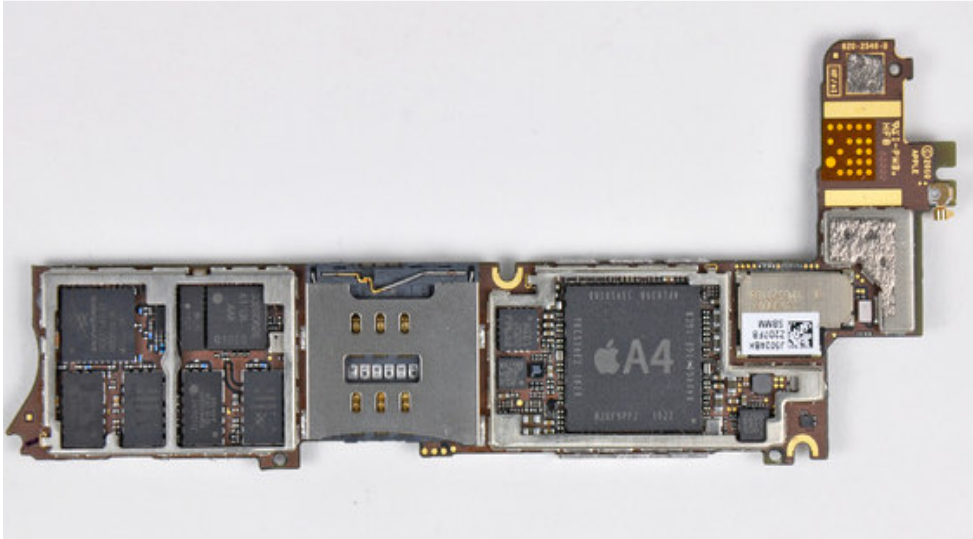
Per la nostra applicazione abbiamo scelto come dispositivo target il multimedial processor Freescale i.MX51, il quale presenta come core principale un processore ARM A8 con frequenza di clock pari a 600 MHz. In questa architettura non troviamo un DSP specifico, ma un coprocessore chiamato NEON il quale esegue istruzioni SIMD a 128 bit. La scelta di questo processore non è casuale, anzi, molti dispositivi in commercio fanno uso di tale processore. Infatti, se andiamo ad esplorare

l'architettura interna del celeberrimo smartphone IPhone 4, scopriamo quanto riportato nella seguente figura:

Figura I.2: Struttura interna dell' IPhone 4 della Apple.

Lo scopo di questa operazione è quello di mostrare qual è il processore presente sullo smartphone, che possiamo vedere ancor meglio nell'immagine successiva:

Figura I.3: Il processore A4 ed altra componentistica presente all'interno dell' IPhone4.



Dalla figura precedente scopriamo la presenza del processore Apple A4 che non è nient'altro che l'ARM A8 prodotto da Samsung e montato anche su altri smartphone in commercio. Riassumendo la scelta del processore i.MX51 ricade su due fattori principali: il primo è quello di aver a disposizione uno dei processori più popolari sul

mercato e quindi assicurare una portabilità elevata all'applicazione che andremo a sviluppare. Il secondo motivo consiste nel poter sfruttare la presenza dell'architettura NEON SIMD per l'accelerazione delle operazioni grafiche. Infatti, l'obiettivo principale è stato proprio quello di cercare di velocizzare l'applicazione sviluppata attraverso l'utilizzo di questa unità in modo tale da poter realizzare un sistema che sia capace allo stesso tempo di lavorare in real time oltre che ad essere robusto ed affidabile dal punto di vista funzionale.

La tesi è stata organizzata nel seguente modo: il primo capitolo descrive la libreria OpenCV con particolare attenzione all'algoritmo di estrazione dei descrittori utilizzato. Infatti, viene descritta in maniera dettagliata la metodologia di indagine ed estrazione dei punti salienti da parte dell'algoritmo SURF. Nel secondo capitolo viene presentata la scheda di sviluppo utilizzata ed in particolar modo il processore utilizzato per il test dell'applicazione sviluppata. Il terzo capitolo è stato dedicato all'ambiente di sviluppo: descrizione della toolchain, configurazione dell'ambiente, procedura di boot del sistema e analisi del compilatore utilizzato. Segue poi il quarto capitolo in cui oltre a descrivere tutte le problematiche affrontate per sviluppare correttamente l'applicazione, vengono riportate anche i commenti a vari frammenti di codice scritti per realizzare il riconoscimento delle immagini. Infine, nel quinto capitolo vengono riportati tutti i risultati significativi, le conclusioni e gli sviluppi futuri proposti per migliorare eventualmente l'applicazione proposta.

Capitolo 1

1. La libreria OpenCV

1.1 Introduzione

OpenCV è una libreria scritta in C++ ideata dalla Intel per risolvere le tipiche problematiche della "Visione Artificiale" o *computer vision* . Lo scopo principale della visione artificiale è quello di ottenere un modello approssimato della realtà tridimensionale, utilizzando immagini bidimensionali tentando di simulare in qualche modo il comportamento dell'occhio umano.

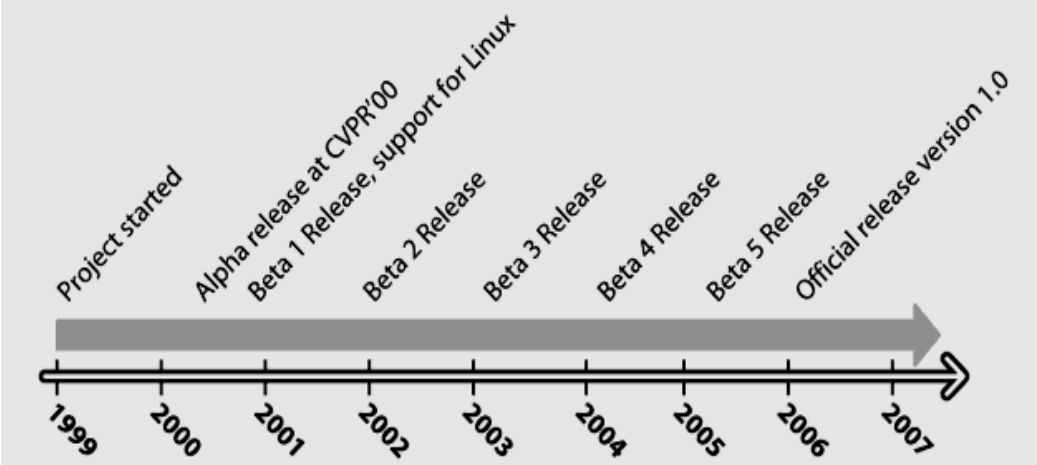


Figura 1.1: Evoluzione temporale di OpenCV

La libreria OpenCV[10] nasce nel 1999 nei laboratori di ricerca della Intel, come strumento di test delle CPU in fase di sviluppo. In quel periodo uno degli autori di OpenCV che lavorava per Intel, stava visitando alcune università e notò in particolare che alcuni gruppi universitari, come l'MIT Media Lab, avevano sviluppato delle strutture di codice basilari che venivano passate da studente a studente in modo tale che ognuno potesse sviluppare la propria applicazione senza dover partire ogni volta da zero. Infatti, OpenCV è stata concepita proprio partendo da questa idea: mettere a disposizione dei ricercatori/sviluppatori un insieme di strutture e algoritmi basilari, utili per sviluppare progetti più complessi in ambito computer vision.

Durante gli anni di sviluppo antecedenti alla prima release ufficiale, la libreria non solo è stata ottimizzata grazie all'ausilio delle Intel Performance Primitives (IPP: migliorano le prestazioni real time su processori con architettura Intel), ma sono state rilasciate anche versioni compatibili con sistemi operativi UNIX. In realtà la libreria OpenCV è scritta completamente in codice C e C++ ottimizzato per le operazioni video, per questo la presenza delle IPP influisce solo sulle prestazioni del codice ma non sul suo effettivo funzionamento. Infatti, se le IPP sono presenti nel nostro sistema, OpenCV utilizzerà automaticamente i link dinamici delle IPP, aumentando le prestazioni in termini di velocità.

Dato che la libreria OpenCV è stata realizzata inizialmente per scopi didattici e di ricerca, la sua licenza è stata sempre mantenuta *free*, e quindi il suo codice può essere utilizzato completamente o in parte sia per applicazioni commerciali che di ricerca.

Nel 2006 nasce la prima versione ufficiale di OpenCV 1.0 e fino ad oggi sono state rilasciate altre due versioni la 2.1 e la 2.2. Quest'ultima presenta nuovi miglioramenti sia in termini di costi computazionali (alcuni algoritmi di identificazione sono stati migliorati, risulta possibile utilizzare sistemi multi-core, ecc..), sia in termini di compatibilità verso sistemi operativi di ultima generazione.

Ciò dimostra come durante questi ultimi anni OpenCV abbia sviluppato a pieno una delle sue principali prerogative, ossia quella di essere indipendente dalla piattaforma e dal sistema operativo da utilizzare. Infatti, mentre la versione 2.1 è compatibile solo con sistemi FreeBSD, Unix, Windows e Mac, la versione di OpenCV 2.2 espande la sua compatibilità anche con i sistemi operativi per dispositivi mobili: iOS e Android[1]. Infine, in rete è possibile trovare tutto il necessario per scaricare[2] ed effettuare il porting di OpenCV sulla piattaforma desiderata, il che risulta di enorme beneficio quando si bisogna compilare su un PC host avente architettura differente dal dispositivo target (*cross-compiling*).

Tra le numerose librerie e funzioni utili offerte da OpenCV troviamo: il riconoscimento di uno o più oggetti attraverso la "feature detection" che permette di caratterizzare un immagine attraverso un numero limitato di punti salienti (*keypoints*), l' identificazione di un volto, la ricostruzione di una scena 3D partendo da due o più immagini 2D, l'acquisizione ed elaborazione di immagini in diversi

[1] http://opencv.willowgarage.com/wiki/Android
[2] http://sourceforge.net/projects/opencvlibrary/

formati, ecc. Tutte queste funzioni sono integrate all'interno della libreria Computer Vision di OpenCV e suddivise in 4 principali librerie come mostra la figura 1.2.

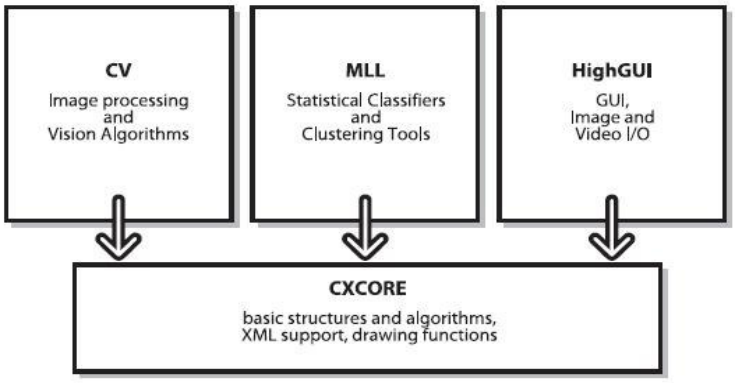


Figura 1.2: Struttura interna delle librere di OpenCV

Per poter eseguire le funzioni elencate precedentemente, la libreria CV sfrutta le proprietà della cosiddetta "Machine Learning Library"(MLL), la quale si basa principalmente su funzioni di classificazione statistiche. La libreria HighGUI contiene le routine e le funzioni per le operazioni di I/O utili per l'elaborazioni video. Le tre librerie appena descritte, si appoggiano sulla CXCORE la quale contiene tutte le strutture dati basilari.

Le principali operazioni che vengono effettuate al'interno di un generico programma che utilizzi le librerie di OpenCV sono le seguenti:

- Acquisizione di Immagini di diverse estensioni(.jpeg , .png, .bmp, ecc.) da file o da fotocamera.
- Acquisizione di frame da uno stream video (.avi).
- Elaborazione grafica delle immagini (filtraggio, rotazione, riduzione o espansione delle dimensioni, conversione in scala di grigi, ecc …).
- Costruzione di un'immagine partendo da immagini pre-elaborate.

Tutte queste operazioni(e molte altre che non riportiamo per semplicità), possono essere realizzate semplicemente chiamando delle opportune funzioni di OpenCV. Quindi, come vedremo anche nei capitoli successivi, grazie alla struttura con cui sono organizzate le funzioni di OpenCV è possibile realizzare elaborazioni complesse in poche righe di codice.

La descrizione fatta in questo paragrafo presenta molto genericamente quali sono le principali potenzialità e capacità che OpenCV mette a disposizione attualmente. Resta comunque da considerare che la computer vision si sta espandendo notevolmente e ciò fa sì che questa libreria ricopra, col passare del tempo, un numero considerevole di applicazioni via via sempre più complesse ed all'avanguardia.

1.2 La scelta dell'algoritmo SURF

In questo paragrafo si vuole chiarire quali sono state le cause principali che ci hanno fatto decidere di optare, tra i vari algoritmi di estrazione dei descrittori contenuti nella libreria OpenCV, proprio il Speed Up Robust Features (SURF). Come accennato nell'introduzione, lo scopo della nostra applicazione è quello di poter rilevare la presenza di un particolare oggetto, indagando all'interno di una generica immagine. Per fare ciò è necessario estrarre sia dall'immagine da riconoscere che da quella generica, degli opportuni punti di interesse (che chiameremo anche *keypoints*). La proprietà principale che deve avere ciascuno di questi punti è quella della ripetibilità, ossia di poter essere identificato anche in condizioni di visualizzazione diverse dell'immagine. Quindi, idealmente ogni punto di interesse non dovrebbe confondersi con altri punti che rappresentano immagini diverse, ne tanto meno dovrebbe cambiare la sua struttura al variare di trasformazioni dell'immagine come: rotazione dell'immagine, variazione del contrasto o della luminosità, ecc. Infatti, attualmente nessun algoritmo di estrazione di features è in grado di garantire una ripetibilità pari al 100%. Una volta trovati i *keypoints*, l'analisi si concentra sul descrivere tutti i punti in un intorno di ciascun punto di interesse, attraverso un opportuno vettore: il descrittore. Anche il descrittore deve essere costruito in modo che goda delle proprietà descritte in precedenza. Quindi, al fine di poter identificare in maniera corretta un'immagine è necessario che da essa vengano estratti un numero significativo sia di *keypoints* che di descrittori In questo modo ogni immagine avrà un set di ciascuno di che la caratterizzano univocamente.

Le fasi realizzate da ogni algoritmo di estrazione sono essenzialmente le seguenti:

1. **Detection**: in questa fase si rilevano all'interno dell'immagine i possibili punti candidati ad essere i *keypoints* principali e si effettuano delle operazioni per rendere tali punti *scale-invariant*.

2. **Description**: qui vengono calcolati i descrittori in modo tale da rendere l'algoritmo invariante alle rotazioni.

Ovviamente, qualora si volesse realizzare non solo l'estrazione dei descrittori ma anche il riconoscimento di oggetti, è necessario implementare un'ulteriore procedura che realizzi il *matching* tra i descrittori. Ossia stabilire se l'immagine di riferimento è presente o meno nell'immagine da analizzare, effettuando opportunamente il confronto tra i descrittori di entrambe le suddette immagini. Ogni algoritmo di estrazione dei descrittori si differenzia dall'altro per come vengono realizzate la fase di "*Detection*" e di "*Description*". I principali algoritmi presenti all'interno della libreria sono i seguenti[3]:

- SIFT: Scale Invariant Feature Transform. Quest'algoritmo utilizza il metodo di estrazione sviluppato da David Lowe[4].
- STAR: Basato sull'algoritmo CenSurE [5] ma modificato in modo da aumentare le prestazioni in termini di velocità ed affidabilità.
- SURF: Speed Up Robust Feature[3]: rimandiamo l'analisi di questo algoritmo al paragrafo 1.3.

I parametri di riferimento per la valutazione degli algoritmi di estrazione sono:

- Ripetibilità: definita come il rapporto tra il numero di punti rilevati più volte diviso il numero totale di *keypoints* trovati.
- Matching Ratio: è il rapporto tra il numero di coppie di descrittori corrispondenti, diviso il numero totale di descrittori.
- Precisione: viene misurata attraverso la valutazione dell'errore di riproduzione dei descrittori.

Recentemente uno studio comparativo tra questi algoritmi [6], ha evidenziato che l'algoritmo SIFT presenta la peggiore ripetibilità ma risulta essere migliore degl'altri in termini di matching ratio e precisione. Il SURF invece risulta essere migliore dello STAR in tutti e tre i parametri. Quindi, in prima battuta l'algoritmo SIFT sembra essere il migliore di tutti. In realtà bisogna effettuare un'ulteriore analisi che riguarda non più le prestazioni "grafiche" dell'algoritmo, ma bensì quelle temporali. Infatti, volendo realizzare un'applicazione real time bisogna privilegiare il tempo

[3] http://opencv.willowgarage.com/documentation/cpp/feature_detection.html

totale di estrazione delle keypoints a discapito dei fattori di merito elencati in precedenza.

A tal scopo si riporta di seguito la tabella 1.1 che elenca le prestazioni in termini di efficienza computazionale ottenute dai tre algoritmi[6]:

Tabella 1.1: Confronto dei tempi di elaborazione tra gli algoritmi di features extraction

methods	Number of points	Detection time /s	Description time /s	Total time /s
SIFT	100	0.374	0.203	0.577
	1000	0.733	1.81	2.543
SURF	100	0.359	0.109	0.468
	1000	0.452	1.108	1.56
Star	100	1.03	0.093	1.123
	1000	1.061	0.873	1.934

Come si nota dalla tabella 1, l'algoritmo più lento in fase di "*Detection*" è lo Star dato che analizza ogni singolo pixel dell'immagine senza effettuare operazioni di scaling, cosa che invece accade sia per il SURF che per il SIFT. In fase di "*Description*" il più lento risulta essere il SIFT a causa dell'elevata dimensione del singolo vettore utilizzato come descrittore del *keypoints*. SURF risulta essere mediamente il più veloce dei tre, anche se risulta essere più lento dello Star in fase di costruzione dei descrittori.

Riassumendo:

- SIFT è in grado di rilevare punti salienti anche di sala ridotta ma presenta costi computazionali gravosi.
- Star è più veloce ma meno accurato di SIFT. La sua tecnica di analisi delle immagini lo rende più lento di SURF.
- SURF è il migliore in termini di costi computazionali ma presenta una precisione minore sull'estrazione dei descrittori, per questo necessita immagini di scala più larga rispetto a SIFT.

Come già detto in precedenza, il criterio di maggior importanza in ambito dell'elaborazione delle immagini in tempo reale è la velocità. Per avvicinarsi al funzionamento real time, il tempo di elaborazione del singolo frame dovrebbe avvicinarsi ai 30 ms. Tale risultato è abbastanza semplice da ottenere se si utilizzano processori di ultima generazione come il multi-core i7 con frequenza di clock superiore ai 2 GHz. Raggiungere i 30 ms per frame diventa un obiettivo molto arduo

invece, quando si utilizzano processori presenti ad esempio nella maggior parte degli attuali smartphone[4] in commercio, i quali presentano frequenze di lavoro non superiori al GHz. Per questo la scelta scelta non poteva che ricadere sull'algoritmo SURF il quale come vedremo, non solo ci permetterà di ottenere ottime prestazioni in termini di tempo, ma anche in termini di affidabilità del matching effettuato.

1.3 Proprietà del SURF

In questo paragrafo viene mostrata con dettaglio quali sono le tecniche utilizzate dal SURF in modo tale da ricavare i descrittori partendo dall'estrazione e caratterizzazione dei *keypoints* .

1.3.1 Immagine Integrale

Uno dei concetti principali che hanno consentito al SURF di migliorare le sue perfomance in termini di costi computazionali, deriva sicuramente dall'utilizzo delle immagini integrali. Infatti, esse permettono di utilizzare delle implementazioni molto veloci di particolari filtri per l'analisi delle immagini. Data un'immagine in ingresso I ed un punto di coordinate (x,y) l'immagine integrale risulta essere definita dalla seguente sommatoria:

$$I_{\Sigma}(x,y) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(x,y) \qquad (1.1)$$

In cui l'elemento della sommatoria I(x,y) è l'intensità dei pixel compresi nella regione rettangolare che ha per convenzione, come vertice superiore sinistro l'origine dell'immagine I e come vertice inferiore destro il punto (x,y). Usando l'immagine integrale, il calcolo della somma delle intensità di una regione rettangolare qualsiasi si riduce a quattro operazioni.

[4] Si veda ad esempio le caratteristiche del processore Snapdragon presente su un'elevata molteplicità di smartphone: http://en.wikipedia.org/wiki/Snapdragon_%28system_on_chip%29

Infatti, se consideriamo un rettangolo limitato dai vertici A, B, C e D, come quello di colore verde nella Figura 1.3, tale valore si calcola come:

$$S = A + D - (C + B)$$

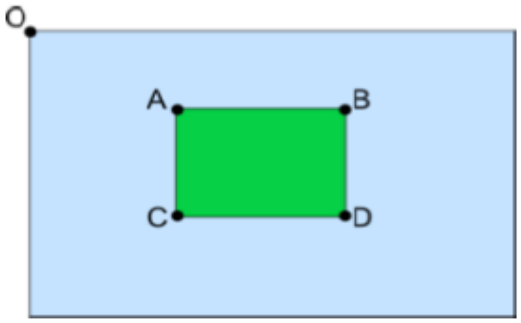


Figura 1.3: Esempio di calcolo di un'imagine integrale.

Dove nella formula A, B, C e D indicano i valori integrali corrispondenti alle coordinate dei vertici. Infine, si noti che, una volta fissata l'area totale della singola immagine integrale, il tempo di computazione della somma S non dipende dalla dimensione totale dell'immagine analizzata. Ovviamente il tempo complessivo di elaborazione in questo caso sarà dato dalla somma dei tempi dovuti al calcolo di tutte le immagini integrali: per questo talvolta è necessario trovare il giusto compromesso tra la dimensione dell'area della singola immagine integrale e il tempo finale di elaborazione che si vuole ottenere.

1.3.2 Fast Hessian Detector

Il detector, ossia l'elemento che permette di individuare i punti salienti dell'immagine, si basa sul determinante della matrice Hessiana. Di seguito riportiamo la definizione che troviamo in [3] della matrice Hessiana calcolata per un'immagine I nel punto x(x,y) e con scala σ mostrata di seguito:

$$H(x, \sigma) = \begin{bmatrix} L_{xx}(x, \sigma) & L_{xy}(x, \sigma) \\ L_{xy}(x, \sigma) & L_{yy}(x, \sigma) \end{bmatrix} \qquad (1.2)$$

Dove $\mathbf{L_{xx}(x,\sigma)}$ è la convoluzione tra le derivate di secondo ordine della Gaussiana $\mathbf{g(\sigma)}$ e l'immagine **I** nel punto $\mathbf{x}(x,y)$ e in modo analogo si definiscono anche le altre

convoluzioni $\mathbf{L_{xy}(x,\sigma)}$ e $\mathbf{L_{yy}(x,\sigma)}$, conosciute anche come *Laplacian of Gaussian* (LOG).

Dal calcolo del determinante della matrice $\mathbf{H(x,\sigma)}$ per ogni singolo pixel dell'immagine, saranno ricavati tutti i punti di interesse dell'immagine stessa.

Infatti, siccome il discriminante corrisponde al prodotto degli autovalori della matrice Hessiana, è possibile classificare se i punti sono estremanti o meno, basandosi sul segno del risultato.

Se gli autovalori sono entrambi non nulli e il discriminante risulta essere negativo, allora gli autovalori hanno segno differente e quindi il punto non è un estremo locale; se invece il discriminante è positivo, allora entrambi gli autovalori o sono entrambi positivi o sono entrambi negativi ma in entrambi i casi il punto è classificato come punto estremante.

L'uso delle Gaussiane permette di variare l'effetto di smooth durante la fase di convoluzione in modo che il determinante sia calcolato a differenti scale. Inoltre, poiché la Gaussiana è una funzione isotropica (cioè ad asimmetria circolare), la convoluzione col suo nucleo è invariante alla rotazione. In pratica le Gaussiane vanno discretizzate e campionate in maniera opportuna. A tal fine Bay[3] utilizza un'approssimazione delle LOG realizzata attraverso i cosiddetti *box filters*.

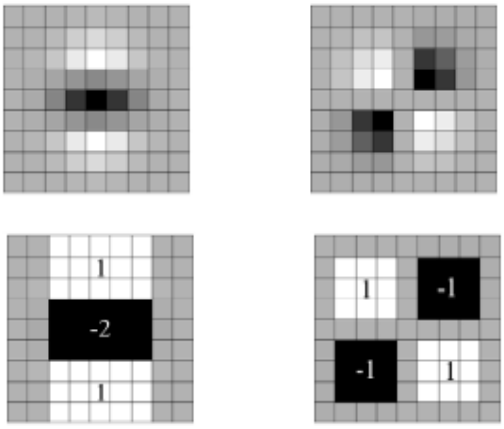


Figura 1.4: Approssimazione del LOG.

La figura 1.4 mostra la somiglianza tra i filtri originali con i box filters utilizzati per l'approssimazione, infatti la riga superiore riporta le derivate Gaussiane del secondo ordine in direzione x e y discretizzate e approssimate (L_{xx},L_{yy}), mentre nella riga

inferiore troviamo le approssimazioni utilizzate nel SURF, dove le parti in nero hanno valore negativo e le parti in bianco valore positivo. I pesi applicati a ciascuna sezione del filtro sono volutamente semplici in modo da effettuare un calcolo dell'area più veloce. Una volta applicata l'approssimazione con i box filters, la matrice Hessiana ha come quattro parametri le convoluzioni approssimate che indichiamo con D_{xx} , D_{xv} e D_{yy}. Utilizzando questi parametri, Bay[3] propone la seguente formula per un'accurata approssimazione del determinante dell'Hessiana:

$$\det(H_{approx}) = D_{xx}D_{yy} - (0.9D_{xy})^2 \qquad (\mathbf{1.3})$$

L'utilizzo congiunto dell'approssimazione appena detta e delle immagini integrali, permettono di diminuire notevolmente i tempi necessari per compiere la fase di "*Detection*" dei punti salienti. Infatti, la perdita di accuratezza dovuta all'approssimazioni utilizzate, risulta ampiamente ripagata da un aumento di efficienza e velocità. Come vedremo in seguito, la ricerca dei massimi locali nello spazio dell'immagine opportunamente ridimensionato al variare della scala, condurrà all'individuazione dei punti di interesse dell'immagine analizzata.

1.3.3 La funzione Scale-Space

Al fine di individuare i punti di interesse usando il determinante dell'Hessiana è inizialmente necessario introdurre la nozione di *scale-space*. Tale funzione può essere usata per trovare punti estremanti attraverso il cambiamento della scala dell'immagine da analizzare. Nella computer vision, la *scale-space* viene tipicamente implementata come una piramide nella quale l'immagine di input è iterativamente convoluta con il nucleo Gaussiano e ripetutamente ridimensionata. Questo metodo è usato in SIFT dove però ogni livello analizzato dipende dal precedente e le immagini necessitano di essere continuamente ridimensionate, motivo per il quale risulta essere non particolarmente efficiente dal punto di vista computazionale. Per quanto riguarda il SURF, il costo di elaborazione dei nuclei risulta essere indipendente dalla dimensione consentendo di computare simultaneamente livelli multipli della piramide scale-space. Si riesce quindi ad evitare il continuo ridimensionamento dell'immagine al variare della scala adottata, riducendo il numero di operazioni da effettuare, guadagnandone quindi in termini computazionali.

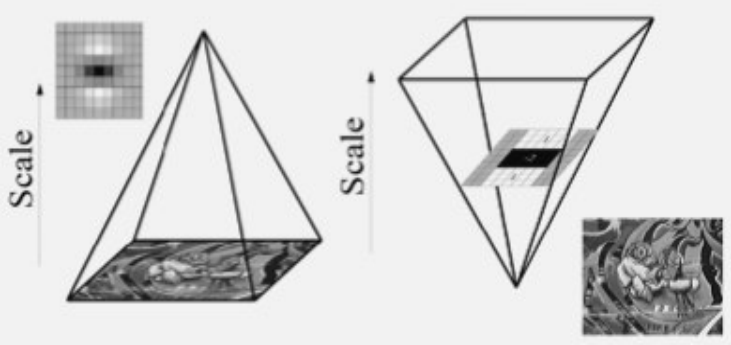


Figura 1.5: Comparazione dei due filtri a piramide realizzabili.

In figura 1.5 viene mostrata la differenza tra l'approccio tradizionale (a sinistra) per costruire la funzione scale space a piramide mentre (a destra), si mostra la tecnica di scale space utilizzata in SURF: a differenza dell'approccio tradizionale, la dimensione originale dell'immagine resta immutata mentre si varia soltanto la dimensione del box filter utilizzato in virtù della proprietà di invarianza allo scaling.

Il livello inferiore della scale-space nel SURF viene ottenuto come output dei filtri di dimensione 9x9 (mostrati in Figura 1.4 di pagina 19). Tali filtri si riferiscono alla convoluzione descritta precedentemente dove la Gaussiana ha deviazione standard σ=1.2. I livelli successivi sono ottenuti incrementando il numero di pixels dei filtri base e mantenendo le proporzioni come si mostra nella Figura 1.6. Affinchè vengano rispettate tutte le proporzioni tra l'immagine e il filtro Gaussiano utilizzato, l'incremento di ciascun filtro rispetto all'altro dev'essere massimo di 6 pixels. La dimostrazione di questo risultato viene mostrata in [8] ma non viene riportata per non appesantire troppo la nostra trattazione.

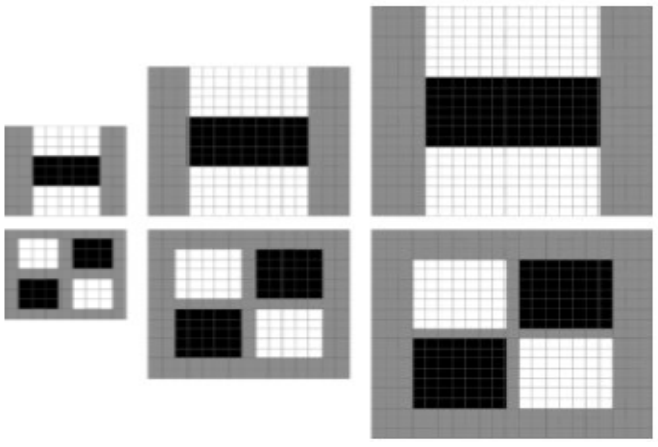


Figura 1.6: Struttura dei filtri usati nella funzione scale-space.

Poiché la dimensione dei filtri aumenta, anche la relativa scala di riferimento deve aumentare in modo da mantenere inalterate le proporzioni. Affinché ciò avvenga calcoliamo la scala con la seguente formula:

$$\boldsymbol{\sigma_{approx}} = \text{FilterSize} \cdot \frac{\text{BaseFilterScale}}{\text{BaseFilterSize}} = = \text{FilterSize} \cdot \frac{1.2}{9} \quad (1.4)$$

1.3.4 Localizzazione dei Keypoints

Per localizzare correttamente l'insieme dei *keypoints* da utilizzare per descrivere l'immagine analizzata, bisogna realizzare essenzialmente le seguenti operazioni:

1. Filtraggio dei punti.
2. Applicazione del metodo di "Non- Maximal Suppression".
3. Interpolazione dei punti trovati.

Il primo punto consiste nel filtrare opportunamente tutti i punti rilevati dal *detector* attraverso la scelta di un valore di soglia (valore minimo del discriminante dell'Hessiana), al di sotto del quale i pixel vengono scartati. Aumentando la soglia diminuisce il numero di punti individuati, i quali però sono i punti più "forti" cioè più caratteristici e originali. Successivamente, viene attuata la cosiddetta operazione di "*Non-Maximal Suppression*" per trovare un set di punti candidati. Questa operazione consiste nel confrontare ciascun pixel della scale-space, con i suoi 26 punti più vicini, compresi 8 punti della scala nativa e i 9 della scala superiore ed inferiore.

Nella pagina successiva si mostra la figura 1.7 dove il pixel indicato con "X" viene selezionato come possibile punto saliente se il discriminante della matrice Hessiana (calcolato dalla 1.3), risulta avere valore più elevato di tutti i discriminanti delle Hessiane dei punti circostanti ad esso, sia nella scala nativa che nelle scale dei filtri adiacenti.

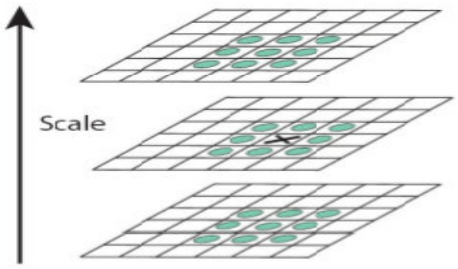


Figura 1.7: Non Maximal Suppression.

A questo punto si ha a disposizione un set di punti di massimo o di minimo pre-filtrati in base al valore di soglia scelto. Il passo conclusivo è quello di interpolare opportunamente i punti in modo da localizzarli con accuratezza ottenendo i *keypoints* finali. Come vedremo in seguito, la determinazione della soglia sarà uno delle operazioni fondamentali per l'applicazione svolta in questa tesi. Infatti, non è semplice stabilire a priori qual è il valore da applicare data una qualsiasi immagine da analizzare. Resta comunque di fondamentale importanza la possibilità di poter effettuare un'operazione di filtraggio che sia invariante sia alla scala che alla rotazione dell'immagine, in quanto la riduzione del numero di punti salienti comporta come conseguenza anche la riduzione del numero di confronti da effettuare in fase di matching. Tutto ciò va ovviamente a vantaggio delle prestazioni computazionali globali dell'applicazione che ricordiamo necessita di operare in real-time.

1.3.5 Tipologie dei descrittori utilizzati dal SURF

Conclusa la fase di "*Detection*" si procede con la fase di "*Description*" nella quale si passa alla costruzione dei descrittori tramite la conoscenza a priori dei *keypoints*.

Come già detto in precedenza, i descrittori sono dei vettori utili a rappresentare in qualche maniera l'intorno di ciascun punto di interesse. SURF effettua questa caratterizzazione andando a valutare come si distribuisce la luminosità dei pixel che si trovano nell'intorno del punto di interesse individuati dal "*Fast-Hessian Detector*". A tale risultato si giunge attraverso l'ausilio semplici filtri per il calcolo del gradiente lungo le direzioni x e y, conosciuti in letteratura anche come "*Haar wavelet*".

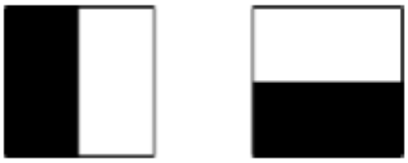


Figura 1.8: " *Haar Wavelet* ": Il filtro a sinistra calcola la risposta nella direzione x mentre quello a destra nella direzione y. La regione bianca ha peso -1 mentre quella nera ha peso 1.

In figura 1.8 troviamo la rappresentazione grafica dei due filtri utilizzati per ricavare la distribuzione di luminosità dei pixel e rendere i descrittori invarianti ai possibili cambiamenti di luminosità. Si fa presente che quando questi filtri vengono utilizzati sulle immagini integrali, sono necessarie solo 6 operazioni per ottenere il risultato finale dell'operazione di filtraggio, a vantaggio sia della robustezza che della velocità computazionale dell'operazione. Si procede all'estrazione dei descrittori seguendo essenzialmente 2 fasi. La prima consiste nell'individuare per ogni punto di interesse una orientazione e successivamente si costruisce una finestra centrata sul punto di interesse, la cui dimensione dipende dalla scala alla quale il punto stesso è stato trovato. Da tale area, e con l'uso congiunto sia dei filtri di Haar che dell'immagine integrale, si estrarrà un vettore di 64 componenti.

In questa fase l'algoritmo SURF propone di assegnare ad ogni punto di interesse un'orientazione riproducibile, in modo da ottenere l'invarianza per rotazione(oltre all'invarianza per scala già ottenuta durante la fase di Detection). Infatti, l'estrazione del vettore descrittore, viene fatta seguendo l'orientazione assegnata ed è per questo molto importante che quest'ultima sia ripetibile ai fini di garantire un algoritmo più robusto. Per determinare l'orientazione di ciascun descrittore, vengono calcolate le risposte dei filtri di Haar con dimensione 4σ per ogni set di pixel contenuti in un'area circolare centrata nel punto saliente e di raggio pari a 6σ, dove σ è la scala relativa al rilevamento del punto stesso. Campionando all'interno dell'area suddetta con un passo di dimensione σ, si ricavano i punti da analizzare. Successivamente le risposte dei filtri vengono poi pesate con una Gaussiana centrata nel punto di interesse. Una volta pesate le risposte vengono rappresentate come elementi di un vettore bidimensionale, dove le risposte lungo x saranno le ascisse e quelle lungo y le ordinate.

Infine, l'orientazione dominante viene selezionata ruotando di 60° l'area circolare costruita in precedenza e centrata nel punto di interesse. A questo punto tutte le

risposte lungo le direzioni x ed y vengono sommate e riutilizzate per creare un nuovo vettore. Ruotando più volte l'area si ottengono diversi vettori, andando a valutare il modulo di ciascuno di essi si sceglierà come direzione dominante quella del vettore che avrà il modulo maggiore.

Il procedimento appena descritto viene mostrato graficamente nella figura 1.9.

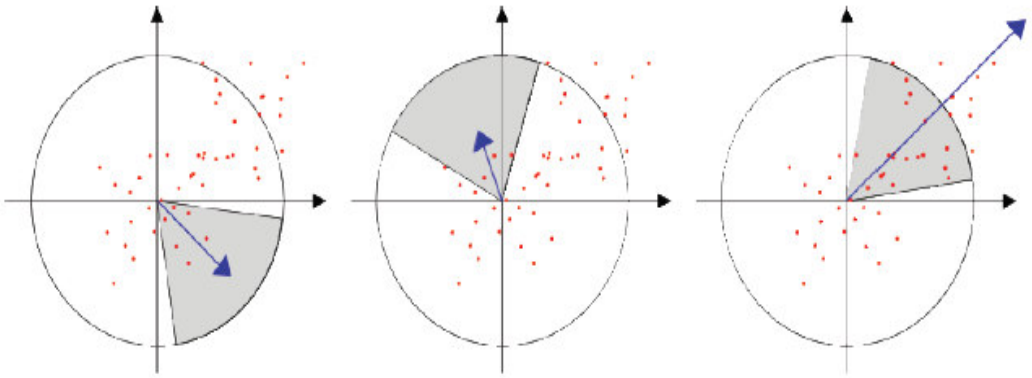


Figura 1.9: Man mano che si ruota la finestra di interesse intorno all'origine vengono sommate le componenti delle risposte dei filtri, in modo da ottenere il vettore indicato in blu.

Esiste anche una versione del SURF che non è invariante per rotazione, in quanto tutte le operazioni appena descritte non vengono implementate. Tale versione è stata chiamata Upright-SURF (U-SURF), e può essere utilizzata in tutte quelle applicazioni in cui non risulta necessario acquisire immagini che devono essere riconosciute qualsiasi sia l'angolo di rotazione rispetto all'originale. Come sarà mostrato nei risultati finali, l'applicazione sviluppata utilizza la versione completa del SURF in quanto la proprietà di invarianza per rotazione risulta essere una componente fondamentale per garantire un'affidabilità maggiore in fase di matching.

1.2 Estrazione dei descrittori

Il primo passo da eseguire per l'estrazione dei descrittori è quello di costruire una finestra quadrata centrata nel punto di interesse rilevato dal detector. Questa finestra ha una dimensione pari a 20σ e contiene al suo interno i pixel che produrranno il vettore descrittore. La finestra risulta essere allineata parallelamente all'orientazione principale ricavata come descritto nel paragrafo precedente. Successivamente la finestra del descrittore viene divisa in una matrice 4 per 4 di regioni regolari.

All'interno di ciascuna regione vengono campionati 25 punti in modo che siano distribuiti regolarmente ai quali poi si applicano i filtri di Haar di dimensione 2σ.

Figura 1.10: Finestre dei vari descrittori individuati. In verde è possibile notare i vari vettori che indicano l'orientazione principale.

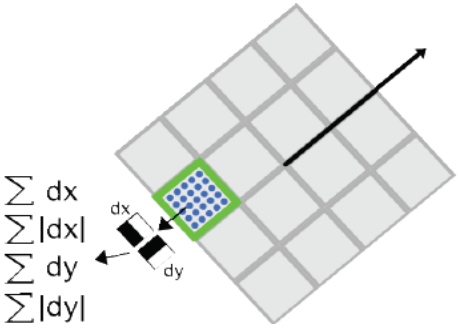


Figura 1.11: Creazione delle sottoregioni (una viene evidenziata in verde) e campionamento uniforme dei punti (in blu) a cui applicare i filtri di Haar.

Se indichiamo con dx e dy le risposte ai filtri lungo le direzioni x ed y per tutti i punti della regione, si ottiene il seguente vettore quadrimensionale che rappresenta il descrittore della singola regione:

$$\mathbf{v_{subregion}} = \left(\sum \mathrm{dx}, \sum \mathrm{dy}, \sum |\mathrm{dx}|, \sum |\mathrm{dy}|\right) \qquad (1.5)$$

Quindi ogni sottoregione fornisce quattro valori al vettore complessivo che avrà una dimensione totale di 4x4x4=64 elementi.

Riassumendo il descrittore ottenuto seguendo le tecniche descritte nei paragrafi precedenti presenta sia caratteristiche di invarianza alla scala, dovute all'opportuna costruzione del Fast- Hessian Detector e all'ausilio dell'Immagine Integrale, sia di invarianza alla rotazione e alla luminosità dell'immagine, proprio grazie alle operazioni viste nel paragrafo precedente. Inoltre, grazie al SURF le performance in termini computazionali risultano essere le migliori tra i vari algoritmi di estrazione analizzati e discussi in precedenza. Quindi, forti delle prestazioni ottimali che il SURF garantisce, ci siamo cimentati in una vera e propria sfida implementativa, ossia quella di eseguire un algoritmo complesso di elaborazione grafica su uno dei più popolari e potenti processori per sistemi embedded di ultima generazione. L'obiettivo primario è quello di eseguire dei test significativi in modo da verificare la possibilità di sviluppare un'applicazione che sia allo stesso tempo affidabile e in grado di lavorare in *real time*.

Capitolo 2

2. La board i.MX51 BBG

In questo capitolo sarà presentato l'hardware utilizzato per effettuare la valutazione delle prestazioni dell'applicazione sviluppata in questa tesi, con un focus particolare sull'application processor i.MX51 e il coprocessore NEON.

2.1 Application Processor Freescale i.MX51

La Freescale Semiconductor è attualmente una delle società leader mondiali per lo sviluppo e la commercializzazione di dispositivi embedded per applicazioni in campo industriale, auto motive e commerciale. Tra i vari prodotti messi sul mercato da Freescale come micro controllori, sensori, DSP, troviamo gli application processor destinati ad essere utilizzati sia in ambito industriale e automotive che in ambito commerciale. Attualmente, la Freescale presenta diverse famiglie di processori utilizzabili in svariati ambiti, nel nostro caso ci interessano tutti i multimedia processor etichettati sotto la sigla i.MX. In particolare si riporta la famiglia i.MX51 in figura 2.1 per analizzare meglio le sue potenzialità.

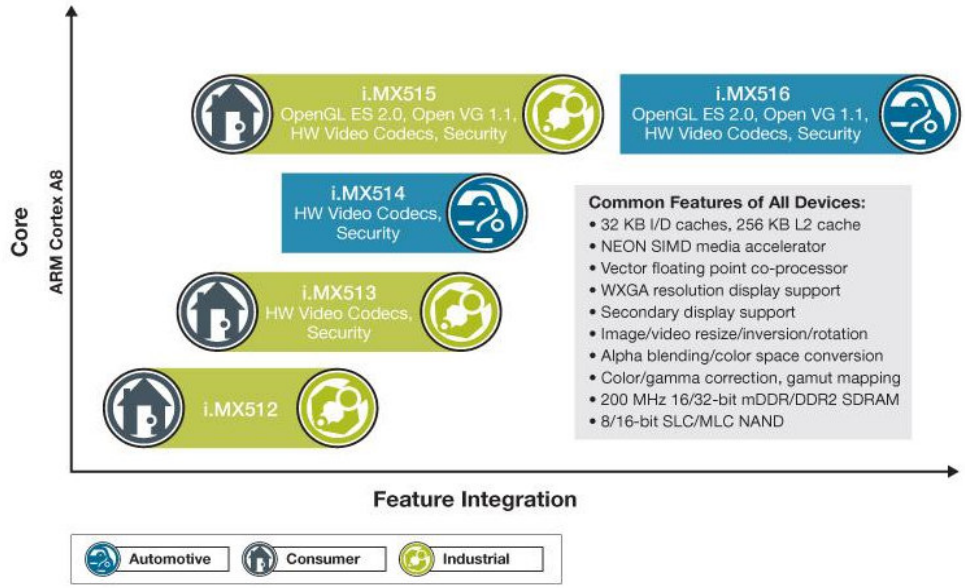


Figura 2.1: i.MX51 Family.

Tutti i prodotti di questa famiglia presentano come caratteristiche comuni un'architettura basata sul core ARM Cortex A8 con frequenza di clock di 800 MHz. Inoltre, troviamo la presenza del coprocessore NEON basato sulle Single Instruction Multiple Data (SIMD) il quale permette di accelerare le operazioni di elaborazione grafica e tante altre caratteristiche fondamentali per tutti quei processori che vengono utilizzati in ambito multimediale. Tra tutti i dispositivi presenti in figura 2.1 abbiamo utilizzato l' i.MX513 che oltre alle caratteristiche suddette presenta anche la possibilità di sfruttare i codec di ultima generazione per l'HDMI**.**

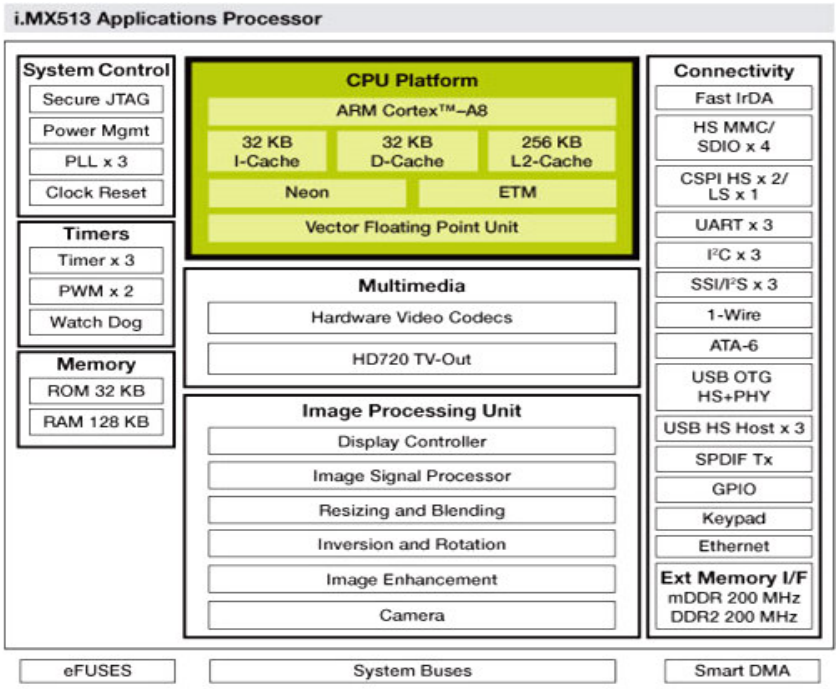


Figura 2.2: i.MX513

Come si evince dalla figura 2.2 nel processore presenta anche la possibilità di sfruttare i codec di ultima generazione per visualizzare immagini con il formato HD. Inoltre, vi è la presenza dell'unita di elaborazione grafica IPU che permette di gestire il display LCD tramite un opportuno controller e di effettuare alcune operazioni di elaborazione delle immagini come il ridimensionamento o la rotazione delle stesse. Attualmente sul mercato è possibile trovare anche la famiglia successiva all'i.MX51 e cioè la i.MX53 la quale eredità tutte le caratteristiche della i.MX51 ma è sviluppata su un core ARM A8 ad 1 GHz. Le applicazioni finali a cui vengono destinati questi application processor sono molteplici e vanno da quelle tipicamente di uso comune fino a quelle di auto motive e industriali. Infatti, come già detto in precedenza, il processore ARM A8

trova largo uso negli smartphone di ultima generazione, nei televisori LED, nei tablet PC, navigatori satellitari, console di gioco e tanti altri prodotti tipici del mercato consumer. Il vero e proprio passo in avanti per il mondo dei processori embedded (che si riuscirà a fare soltanto nel Q4 del 2011), è quello di aver a disposizione sul mercato il primo processore con architettura quad-core. Infatti, Freescale ha appena annunciato la nuova famiglia di application processor i.MX6 la quale presenta tre prodotti basati su architettura singole core, dual core e quad core in cui l'elemento principale è il core ARM A9 .

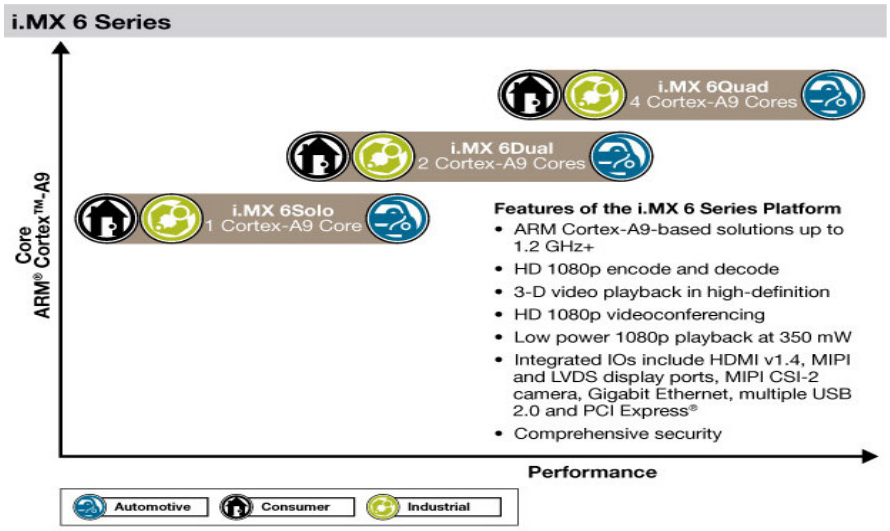


Figura 2.3: i.MX6 Family.

La famiglia **i.MX 6Quad** permetterà di realizzare prodotti mobile con performance elevate come smartbook e hiegh-end tablet, i quali richiedono alte prestazioni multimediali e allo stesso tempo dei consumi di potenza ridotti. Tutto ciò è fattibile grazie all'utilizzo dell' ARM Cortex A9 che lavora a 1.2 GHz integrato con un unità grafica 3D e motori di codifica/decodifica a 1080p. Il consumo di questo application processor viene stimato essere pari a 350 mW anche in fase di playback dei video in HD.

2.2 NEON SIMD Media Accelerator

Uno degl'elementi di forza del processore i.MX51 è sicuramente quello di poter sfruttare tutte le potenzialità del processore ARM A8 nel quale è stata implementata per la prima volta la tecnologia SIMD utilizzata dal NEON. Innanzitutto, cerchiamo di capire qual è il vantaggio di utilizzare questo dispositivo in modo tale da saper come e

quando esso agisce generando un vero e proprio aumento delle prestazioni computazionali. Alcuni software moderni utilizzati nel campo multimediale, fanno uso di particolari acceleratori e codec grafici che lavorano su un'enorme quantità di istruzioni che talvolta hanno dimensioni inferiori rispetto alla dimensione totale della word che il processore ha a disposizione. Ad esempio in alcune applicazioni audio vi sono delle istruzioni che operano a 16 bit o ancora, in tipiche applicazioni video vengono utilizzate di solito istruzioni a 8 bit. Quando queste operazioni vengono realizzate su un microprocessore a 32 bit, l'unità di calcolo continua a consumare energia come se venisse utilizzata interamente mentre invece è sfruttata solo parzialmente. Quindi, per migliorare l'uso delle risorse disponibili, la tecnologia SIMD utilizza singole istruzioni in parallelo per realizzare la stessa operazione e lo fa su dati che hanno lo stesso tipo e la medesima dimensione. Ad esempio, l'hardware che tipicamente aggiunge due dati a 32 bit viene realizzato in modo tale che esegua quattro addizioni parallele ad 8 bit, impiegando lo stesso tempo dell'hardware che fa la somma standard dei due elementi presi singolarmente a 32 bit.

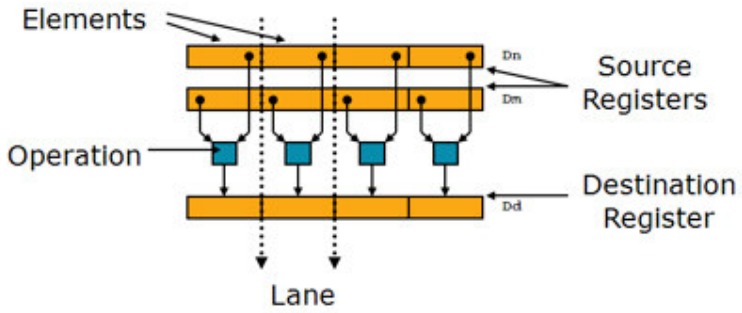


Figura 2.4: Esempio di istruzione SIMD realizzata col NEON.

Le architetture ARMv7 hanno introdotto l'utilizzo della tecnologia Advanced-SIMD che estende il concetto della SIMD tramite l'utilizzo di gruppi di istruzioni che operano su vettori immagazzinati nelle cosiddette *double-word* ed *quad-word* da 64 e 128 bit rispettivamente. L'implementazione dell' Advanced SIMD usata nei processori ARM viene chiamata NEON. Attualmente tale tecnologia è implementata per tutti i processori della serie ARM Cortex-A. Le istruzioni del NEON vengono eseguite come parti dei tipici flussi di istruzioni ARM o Thumb, in modo da semplificare lo sviluppo software e le fasi di debug. Mentre le istruzioni tradizionali ARM o Thumb gestiscono l'intero flusso di un programma e la sua sincronizzazione, le istruzioni NEON consentono di:

- Effettuare accessi in memoria.
- Convertire tipi di dati.
- Elaborazione dei dati.
- Copia di dati tra i registri NEON e registri di uso generico.

Per utilizzare correttamente il NEON è necessario disporre degl'ultimi compilatori GNU: in particolar modo nel prossimo capitolo vedremo meglio questo aspetto quando andremo a descrivere la toolchain utilizzata. Vi sono due modi principali di scrivere del codice per il NEON. Il primo è quello di sviluppare del codice direttamente in linguaggio C tramite il Gnu Compiler Collection (GCC) o tramite il RVCT (ossia il *RealView Compilation Tools*[5] che utilizza un compilatore proprietario realizzato dalla ARM). Un altro modo di utilizzare il NEON, meno diretto ma comunque efficace, è quello di utilizzare delle funzioni intrinseche o *intrinsic*. Questa particolari funzioni vengono scritte in C o C++ direttamente nel codice in cui si realizza l'applicazione, seguendo delle istruzioni opportune. Dopodiché esse vengono sostituite da specifiche sequenze di istruzioni assembler in fase di compilazione. Ciò permette allo sviluppatore di rappresentare delle operazioni a basso livello attraverso un linguaggio ad alto livello. In questo modo il programmatore stesso ha la possibilità di scrivere direttamente delle funzioni a basso livello che di solito non vengono tradotte bene dal compilatore, ottimizzando le operazioni che esso svolge in fase di di compilazione e aumentando quindi le performance complessive. GCC ed RVCT supportano la sintassi delle NEON intrinsic, in modo da realizzare un codice C o C++ indipendente dalle toolchain che si utilizzano.

In questo lavoro di tesi si è scelto di utilizzare il NEON per ottimizzare il codice scritto in C/C++ in modo tale da migliorare le prestazioni computazionali complessive. Come vedremo, il NEON è stato abilitato settando opportunamente alcune opzioni del compilatore GCC in fase di compilazione, in modo tale che solo alcune porzioni di codice vengano tradotte in linguaggio di basso livello e quindi senza l'ausilio delle *intrinsics*. Infatti, l'obiettivo primario è stato sicuramente quello di valutare un effettivo incremento delle prestazioni abilitando semplicemente il NEON. Una volta constatato ciò si potrebbe anche svolgere un lavoro più approfondito che sviluppi

[5] Questo compilatore commerciale è sviluppato per ottimizzare il codice C/C++ scritto per i processori ARM: http://www.arm.com/products/tools/software-tools/rvds/arm-compiler.php

tramite le *intrinsic* delle opportuni routine che mirino ad ottimizzare ancora di più il codice dell'applicazione. In questo lavoro di tesi tale approfondimento non risulta necessario in quanto, come dimostreranno i risultati finali raggiungi, è molto più che sufficiente la sola abilitazione del NEON per avere elaborazioni real time. Ciò non toglie che si potrebbero sviluppare in futuro porzioni di codice che utilizzino anche le istruzioni intrinsic in modo da migliorare ancor di più le prestazioni computazionali dell'algoritmo.

2.3 Kit di sviluppo per i.MX51

Al fine di poter sviluppare nuove applicazioni con gli application processor descritti nel paragrafo precedente, la Freescale mette a disposizione per i propri clienti degli opportuni kit di sviluppo. Nel nostro caso il kit di sviluppo fornisce all'utente una scheda contenente sia l'i.MX51 che una serie di utili periferiche per poter programmare correttamente il processore. Inoltre, abbiamo a disposizione anche un touch screen LCD e una scheda di espansione provvista di fotocamera digitale per l'acquisizione di immagini. La scheda viene mostrata in figura 2.5 dove possiamo notare tra le principali periferiche di I/O il connettore ethernet, UART, USB e mini USB. Inoltre, grazie ai connettori MMC/SD sia su lato anteriore e posteriore della scheda è possibile utilizzare delle schede di memoria SD per immagazzinare le informazioni utili per esempio al booting locale del sistema operativo.

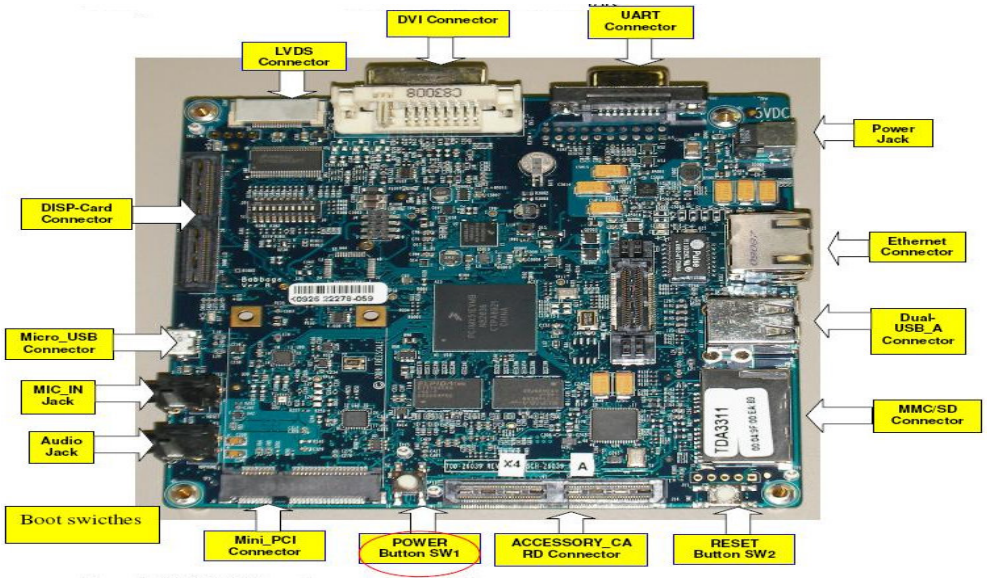


Figura 2.5: Facciata superiore della scheda di sviluppo i.MX51 BBG.

La board presenta una memoria RAM da 512 MB SDRAM DDRII che lavora ad una frequenza di 200 MHz con un bus dati a 32 bit. Come memoria non volatile si ha a disposizione di una flash SPI NOR di dimensione pari a 4 MB che può essere utilizzata in fase di booting settando opportunamente degli switch presenti sulla scheda. La dimensione della memoria flash non è sicuramente da sottovalutare in quanto è possibile ad esempio salvare in essa il bootloader e l'immagine compressa del kernel linux (qualora si volessero utilizzare sistemi Unix-based ovviamente). Per quanto riguarda l'uscita video è possibile utilizzare sia quella DVI per visualizzare l'output su un classico monitor LCD che il connettore DISP-Card sul quale montare il display LCD fornito nel kit. Infine, il connettore Accessory_card viene utilizzato per montare la scheda su cui è montata la fotocamera digitale. Il montaggio completo di tutte le parti disponibili nel kit i.MX51 viene mostrato in figura 2.6.

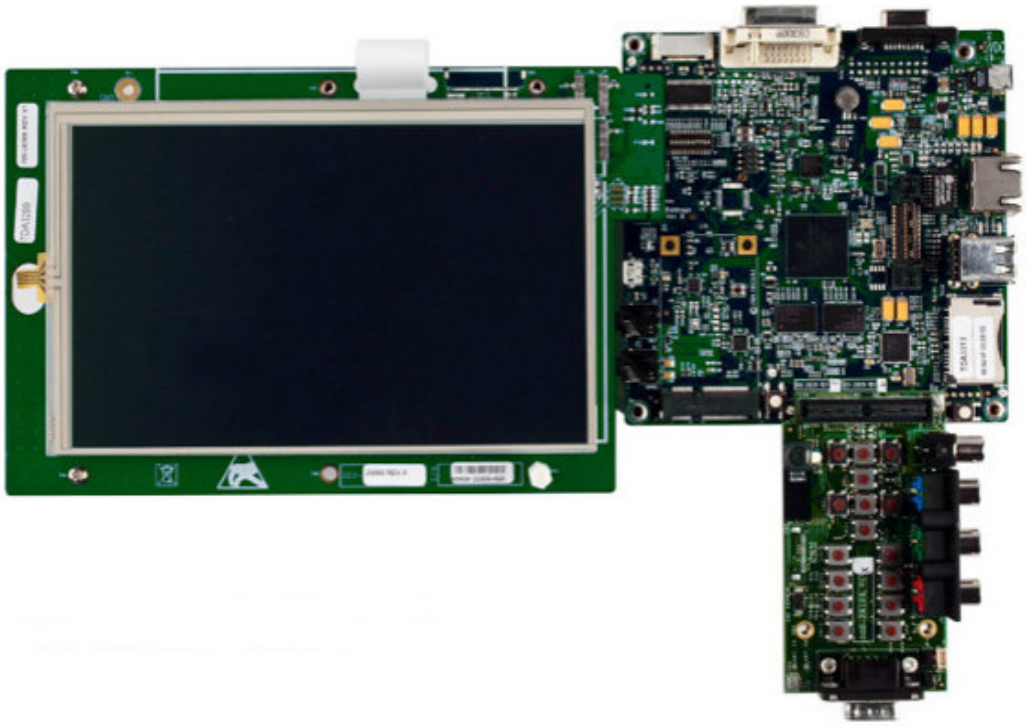


Figura 2.6: Evaluation Kit completo per l'i.MX51.

Infine, all'interno del kit è possibile trovare anche tre schede di memoria SD ciascuna delle quali contiene un sistemi operativo Android R7, Ubuntu e Windows CE. Per il nostro lavoro di tesi non abbiamo sfruttato nessuno di questi tre sistemi operativi, ma abbiamo lavorato su un sistema operativo Linux creato opportunamente da noi tramite i tool di sviluppo che andremo a descrivere nel capitolo successivo.

Capitolo 3

3. Ambiente di sviluppo

3.1 Freescale SDK per i.MX51

Oltre all'hardware messo a disposizione dal kit di sviluppo, la Freescale fornisce sul proprio sito i cosiddetti Software Development Kit (SDK) per effettuare le operazioni di installazione, debugging e cross-compilazione delle applicazioni sviluppate per le piattaforme i.MX. A seconda del sistema operativo sul quale si decide di operare, è possibile trovare una SDK appropriata: nel nostro caso abbiamo scelto di sviluppare tutto il lavoro su un sistema operativo Linux. Quindi, il primo passo da effettuare è quello di scaricare la SDK dal sito della Freescale nella sezione relativa agli application processor i.MX51[6]. Di seguito si riporta la descrizione degli elementi principali che costituiscono l'SDK e come essi sono stati utilizzati per poter creare tutto l'ambiente di sviluppo utilizzato per progettare correttamente l'applicazione oggetto di questa tesi.

3.2 LTIB

Il Linux Target Image Builder [7] è un progetto Open Source che permette di sviluppare tutto ciò che serve per avviare un sistema operativo Linux su diverse tipologie di target. In questo paragrafo andremo a descrivere essenzialmente come sia possibile creare un'immagine personalizzata del kernel linux tramite LTIB. Tale immagine rappresenterà il nostro ambiente di lavoro per la scheda i.MX51.

3.2.1 Prerequisiti per l'host PC

Al fine di poter completare correttamente la procedura di costruzione del kernel linux tramite LTIB bisogna assicurarsi di aver i seguenti prerequisiti minimi:

- Ubuntu 9.04 (versione a 32 bit).

[6] http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=i.MX513&nodeId=018rH3ZrDR633B&fpsp=1&tab=Design_Tools_Tab : Scegliere poi il file L2.6.31_10.07.11_ER_SOURCE. Si ricordi che è necessario essere registrati al sito della Freescale per poter scaricare il tutto.

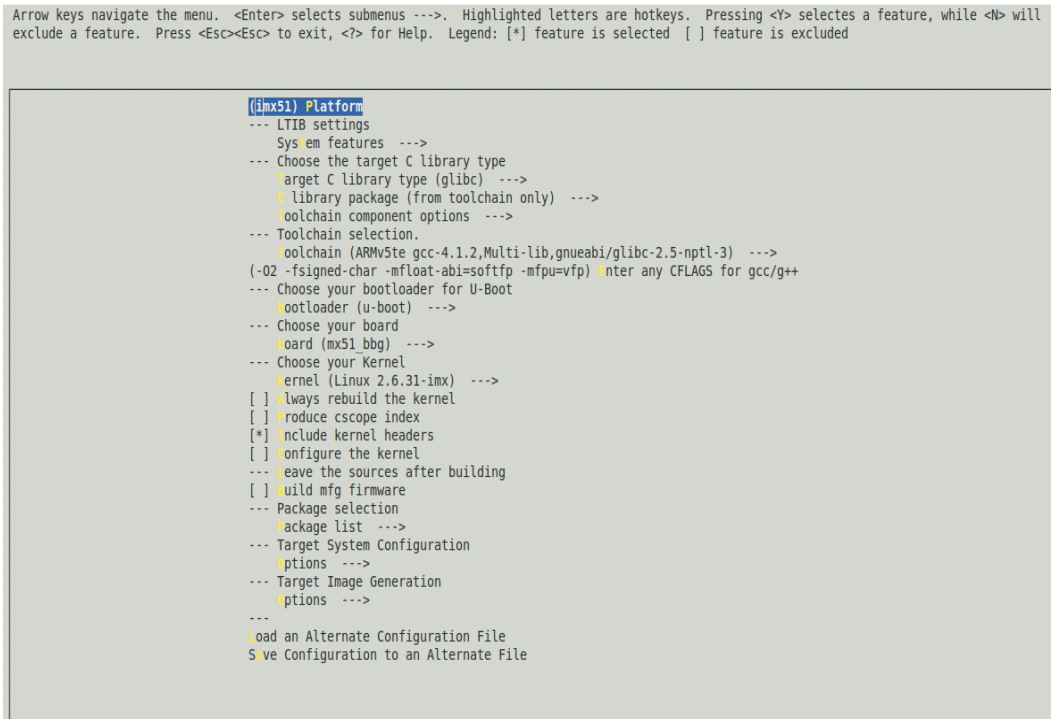
[7] Per ulteriori informazioni sul progetto LTIB andare sul sito: http://www.bitshrine.org/ltib/ .

- Installare i pacchetti liblzo2-dev e uuid-dev con apt-get.
- Ulteriori pacchetti mancanti saranno indicati in fase di installazione di LTIB.
- Server NFS.

3.2.2 Procedura per l'installazione di LTIB

Una volta scaricata l'SDK, estrarre il contenuto di L2.6.31_10.07.11_ER_source.tar.gz in una cartella di lavoro, tale cartella è temporanea e va poi rimossa al termine dell' installazione. Entrare nella cartella di lavoro e dare il comando **./install** (è conveniente non avere i diritti di root in fase di installazione di LTIB). Durante l'esecuzione dello script install viene chiesto dove installare LTIB, questa è la cartella definitiva di installazione a cui si farà riferimento nel seguito con LTIB_DIR. Terminata l'installazione, entrare nella cartella creata e dare il comando **./ltib**. Tale comando genera tutto l'ambiente di sviluppo a partire dall'installazione della toolchain alla compilazione di boot-loader e kernel alla compilazione dei componenti necessari per il rootfs. La toolchain ed i packages rpm con i sorgenti vengono copiati in /etc/freescale mentre LTIB viene installato nella directory specificata LTIB_DIR.

Figura 3.1 : Schermata iniziale di LTIB per la configurazione principale del kernel Linux.



Durante l'esecuzione del comando **./ltib**, viene presentato un menù di selezione nel quale è possibile definire le opzioni di generazione del SDK, in particolare è necessario selezionare la piattaforma i.MX51. Inoltre, all'interno della schermata di configurazione è possibile selezionare anche alcune librerie opzionali. Nel nostro caso abbiamo importato la libreria OpenCV 1.1 in modo tale da poterla sfruttare successivamente nella fase di realizzazione del codice C/C++ dell'applicazione sviluppata. Ovviamente ciò comporta anche l'importazione di ulteriori librerie da cui OpenCV dipende. In questa fase abbiamo scelto di creare un kernel che contenesse soltanto lo stretto necessario per eseguire e verificare le prestazioni dell'applicazione che andremo a realizzare. Per questo abbiamo preferito evitare di appesantire il kernel con l'aggiunta dei tipici tool di cui esso di solito è costituito (come gedit, vim, ecc.). Al termine dell'esecuzione di **./ltib** la directory LTIB_DIR conterrà la cartella rootfs la quale a sua volta contiene la cartella boot. Se il processo è stato correttamente eseguito troveremo nella cartella LTIB_DIR/rootfs/boot sia boot loader (**u_boot.bin**) che il kernel (**uImage**).

La struttura dell'ambiente di sviluppo è stata organizzata nel seguente modo:

- Il boot loader uBoot viene scritto in SD
- Il kernel generato viene scritto in SD
- Il rootfs viene montanto via server nfs

3.3 Preparazione dell'ambiente di sviluppo

3.3.1 Configurazione della scheda SD

Abbiamo scelto di utilizzare la SD per memorizzare u-boot e il kernel, ma nulla vieta in un secondo momento di utilizzarla anche per il rootfs. E' stata creata un'unica partizione riservando uno spazio libero iniziale di 8192 settori da 512 byte pari a 4 Mbyte. I 4 Mbyte iniziali servono per ospitare il settore di avvio o il cosiddetto *master boot record* (MBR), u-boot e il kernel. Qualora si voglia memorizzare il rootfs in SD allora si può utilizzare la partizione che occupa tutto lo spazio eccedente i 4 Mbyte iniziali. Per prima cosa bisogna creare la partizione inizializzando il MBR, per far questo eseguiamo la seguente procedura:

1. Inserire la scheda SD su un PC con distribuzione Linux.
2. Dare il comando **“** $ sudo fdisk /dev/xxx**”** dove xxx è il device che rappresenta la SD utilizzata (rilevabile con il comando *dmesg* da eseguire dopo l’inserimento della SD).
3. Dare il comando *u* per mettere le unità di misura in blocchi.
4. Creare la nuova partizione con il comando *n*, specificare blocco iniziale = *8192*.
5. Salvare con il comando *w*.

Prima di procedere con la procedura successiva, copiamo il boot loader e il kernel che si trovano nella cartella LTIB_DIR/rootfs/boot all’interno della cartella di lavoro. Ora occorre scrivere in SD il boot loader, copiando in SD il file u_boot.bin a partire dall’ offset 0 in modo da sovrascrivere il MBR che si trova nel primo blocco da 512 byte. Per questo è necessario salvare il MBR in un file binario prima che venga sovrascritto utilizzando il comando seguente:

- $ sudo dd if=/dev/xxx of=./mbr.bin bs=512 count=1

Dopodichè andiamo a copiare il boot loader in SD a partire dall’ indirizzo “0”:

- $ sudo dd if=./u-boot.bin of=/dev/xxx bs=512

Quindi, procediamo alla scrittura del kernel a patire dall’ indirizzo “0x1000”(ossia pari ad 8192 blocchi da 512):

- $ sudo dd if=./uImage of=/dev/xxx bs=512 seek=8192

Adesso ripristiniamo il MBR dal file binario salvato in precedenza:

- $ sudo dd if=./mbr.bin of=/dev/xxx bs=512

Infine, per sicurezza diamo un comando *sync* prima di rimuovere la scheda SD.

3.3.2 Configurazione di Minicom

Minicom è un programma che permette alla macchina host di comunicare con la macchina target e viceversa, attraverso la porta seriale. Per installarlo digitare su una shell di comando:

```
$ sudo apt-get install minicom
```

I parametri basilari da impostare per comunicare correttamente via seriale sono riportati nella tabella seguente:

Baud Rate	115200
Data Bits	8
Parity	None
Stop Bits	1
Flow Control	None

3.3.3 Procedura di avviamento iniziale della board.

Una volta preparata la scheda SD viene inserita nello slot inferiore della board, posizionando i dip-switch S1 per avviare il boot tramite SD/MMC (tutti gli switch a off tranne 7 e 8 come indicato nei data sheet della board). Viene collegato opportunamente il cavo seriale tra PC host e target e si alimenta la board. Adesso inseriamo i seguenti comandi da shell per avviare la comunicazione seriale e il processo di boot della scheda:

```
$ TERM=linux minicom
```

All'avvio di minicom è possibile inserire i vari comandi per il set up della comunicazione seriale. Inoltre, è possibile abilitare alcuni opzioni utili come il *line wrapping* tramite il comando *Ctrl+A* e poi *w*. In generale per accedere alla lista di tutti i comandi eseguibili basta eseguire *Ctrl+A* e poi *z*. Il comando TERM=linux consente di eliminare un piccolo baco che è stato rilevato nelle distribuzioni di linux meno recenti quando utilizzano minicom come terminale seriale. Praticamente permette di ridimensionare senza problemi la finestra della shell dopo aver lanciato la comunicazione seriale tramite minicom. Una volta configurato opportunamente minicom, possiamo avviare la fase di boot della scheda tramite l'opportuno tasto di accensione. Se la scheda SD è stata scritta correttamente, dopo alcuni secondi dovrebbe comparire il prompt di u-boot sulla seriale. Adesso occorre configurare le variabili d'ambiente di u-boot, per far questo dare in sequenza tutti i comandi sotto elencati al prompt di u-boot:

- Configurazione base:

```
$ setenv bootargs_nfs

$ setenv bootcmd_net

$ setenv bootargs_base

$ setenv uboot_addr

$ setenv uboot

$ setenv kernel

$ setenv load_uboot

$ setenv serverip 192.168.0.2

$ setenv ipaddr 192.168.0.100

$ setenv nfsroot /tftpboot/fs_imx51
```

Nella configurazione base i primi comandi *setenv* servono per dichiarare delle variabili d'ambiente che poi saranno opportunamente settate successivamente a seconda delle opzioni che si vogliono assegnare per il boot del sistema operativo. Le ultime tre righe assegnano alle variabili *serverip*, *ipaddr* e *nfsroot* i valori indicati. Questo perché abbiamo scelto di caricare il root file system via server nfs, quindi è necessario assegnare alla scheda target un suo ip (*ipaddr*) che utilizzeremo in seguito per la fase di avvio della stessa. Il *serverip* invece è l'ip del nostro PC host mentre *nfsroot* indica il percorso in cui si trova il file system per la nostra board i.MX51. Una volta eseguita la configurazione base è possibile aggiungere i comandi di una delle seguenti configurazioni a seconda che si voglia proiettare l'output sullo schermo LCD o su un ulteriore display connesso tramite cavo DVI.

- Configurazione per utilizzare lo schermo **LCD** in dotazione:

```
$ setenv nfsroot '/home/marco/Scrivania/work/ltib/rootfs'

$ setenv bootargs_base 'setenv bootargs console=ttymxc0,115200
  ip=${ipaddr} wvga'

$ setenv bootargs_nfs 'setenv bootargs ${bootargs} root=/dev/nfs
  nfsroot=${serverip}:${nfsroot},v3,tcp'
```

```
$ setenv bootcmd_nfs 'run bootargs_base bootargs_nfs ;
  nfs${loadaddr} ${serverip}:${nfsroot}/boot/uImage;bootm'

$ setenv bootcmd 'run bootcmd_nfs'

$ saveenv
```

- Configurazione per utilizzare l'**uscita DVI**:

```
$ setenv nfsroot '/home/marco/Scrivania/work/ltib/rootfs'

$ setenv bootargs_base 'setenv bootargs console=ttymxc0,115200
ip=${ipaddr} video=mxcdi0fb:RGB24,1024x768M-16@60 '

$ setenv bootargs_nfs 'setenv bootargs ${bootargs} root=/dev/nfs
nfsroot=${serverip}:${nfsroot},v3,tcp'

$ setenv bootcmd_nfs 'run bootargs_base bootargs_nfs ; nfs
${loadaddr} ${serverip}:${nfsroot}/boot/uImage;bootm'

$ setenv bootcmd 'run bootcmd_nfs'

$ saveenv
```

Attenzione: il path di nfsroot dev'essere adattato all'occorrenza a seconda della cartelle ad esso riservata in fase di installazione di ltib. Alla fine di ogni configurazione verifichiamo tramite il comando *printenv* che l'effettivo cambiamento delle variabili d'ambiente sia avvenuto correttamente.

Come si nota in entrambe le configurazioni vengono richiamate le variabili definite in precedenza con il comando setenv. Le funzioni principali che vengono effettuate in questa fase sono quelle di specificare le periferiche principali di I/O al sistema operativo(tastiera e video nel nostro caso), indicare la modalità di caricamento dell'immagine del kernel (via nfs) e infine indicare la posizione dell'immagine stessa. Infine, si lancia il comando di boot di default. L'analisi approfondita di questi comandi esula dallo scopo di questo paragrafo, pertanto si rimanda ad una lettura più approfondita al sito [9] degli sviluppatori di uBoot.

3.3.4 Configurazione del server NFS

Il Network File System è un protocollo che permette a più target di condividere file, cartelle o un intero file system con altre macchine host tramite la rete. Nel nostro caso

settiamo il server NFS in modo tale che la scheda i.MX51 possa montare via ethernet il root file system contenuto sul PC-host.

Creiamo un link simbolico al rootfs nell'ambiente di sviluppo creato in precedenza:

```
$ cd /tftpboot
$ ln -s LTIB_DIR\rootfs fs_imx51
```

Rendiamo disponibile la rootfs agendo sul file etc/exports. Infine, riavviamo il server nfs tramite il seguente comando da shell:

```
$ /etc/init.d/nfs_kernel_server restart
```

3.3.5 Boot del sistema operativo

A questo punto è possibile lanciare il nostro sistema operativo sulla scheda target. Una volta controllata la connessione della porta ethernet del PC-host alla porta ethernet del target tramite un opportuno cavo ethernet "*crossover"*, vengono eseguiti i seguenti comandi dalla shell del PC-host :

```
$ sudo ifconfig eth1 192.168.0.2
$ sudo service nfs-kernel-server restart
$ TERM=linux minicom
```

In questo modo viene settato opportunamente l'ip dell'host pc con quello indicato come *serverip* nelle variabili d'ambiente di uBoot. Infine, si riavvia il server nfs (per evitare problemi qualora fosse utilizzato da altre applicazioni) e viene lanciata la comunicazione seriale come già visto in precedenza. A questo punto ci sarà un count-down di tre secondi utile per poter accedere al prompt di uBoot premendo un tasto qualsiasi dalla console dell' host-Pc (qualora si voglia modificare le variabili d'ambiente di uBoot). Al termine di questo tempo parte la vera e propria fase di boot. Se la procedura è andata a buon fine resta solo da inserire nella shell Linux come login e password la parola ***root***.

3.3.6 Modifiche al File System per avviare il Server X

Per lanciare correttamente il server X[8] dal target bisogna modificare opportunamente il file */bin/startx* lasciando attivo solo l'avvio del server e disattivando l'avvio del *window manage twm* e tutto ciò che segue. Per maggiore chiarezza riportiamo di seguito il contenuto del file startx modificato per le nostre esigenze:

```
#!/bin/sh
Xfbdev -ac -mouse mouse -keybd keyboard &
#sleep 3
#DISPLAY=:0 twm
#killall Xfbdev
```

Adesso sempre dalla shell target lanciare il server X ed esportare la variabile d'ambiente DISPLAY tramite i seguenti comandi:

```
$ startx
$ export DISPLAY=127.0.0.1:0.0
```

Trascurare gli eventuali warning forniti dalla shell dopo aver inserito il comando *startx*. Da questo momento in poi è possibile lanciare i client X.

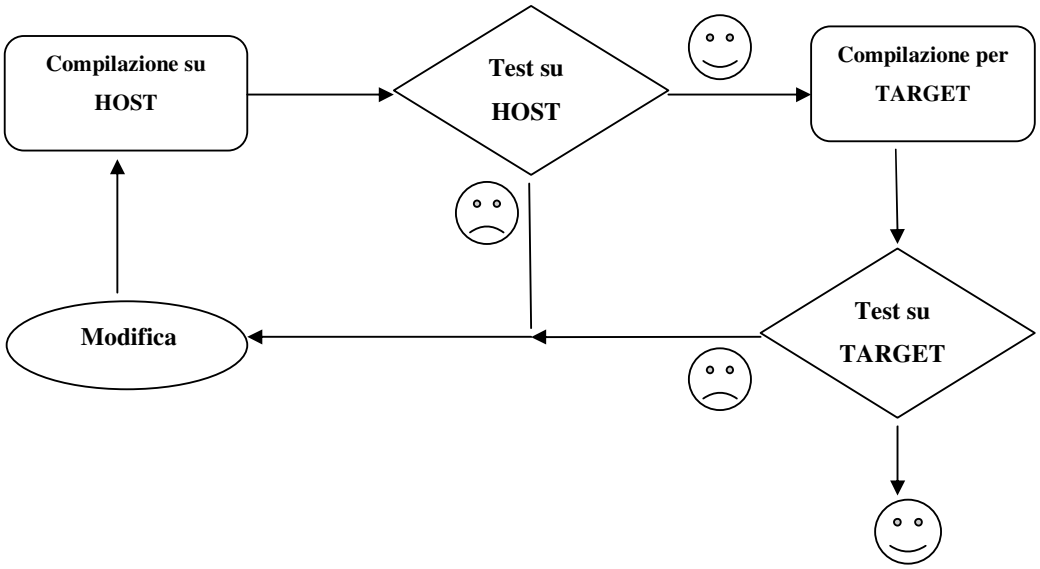
Queste ultime modifiche ci hanno permesso di poter visualizzare sul display LCD, montato sulla evaluation board, le immagini ottenute durante le varie fasi di test dell'algoritmo implementato. Ovviamente facendo questa scelta, bisogna configurare anche opportunamente il bootloader in modo tale che venga abilitato lo schermo LCD della scheda anche in fase di boot. Per questo basta scegliere come configurazione di uBoot quella che abbiamo indicato con "*Configurazione per utilizzare lo schermo LCD in dotazione*" nel paragrafo 3.3.3 .

3.4 Cross compilazione

Al fine di poter sviluppare correttamente la nostra applicazione è stato necessario installare opportunamente la libreria OpenCV sul PC-host e configurare il compilatore gcc in modo tale che generasse codice eseguibile per l'architettura ARM della nostra

[8] http://it.wikipedia.org/wiki/X_Window_System

scheda target. Questa operazione viene anche detta di *cross compilazione*. Essenzialmente il vantaggio di eseguire tale operazione consiste nello sfruttare un host-PC la cui potenza di calcolo è sicuramente maggiore rispetto ai tipici processori utilizzati dalle schede target. Inoltre, il sistema operativo che abbiamo installato sulla nostra i.MX51 non ha a disposizione il compilatore, il debugger e tanti altri strumenti tipici di un qualsiasi sistema operativo. Questo per i motivi già detti in precedenza inerenti alla realizzazione di un sistema operativo più semplice possibile. Quindi, la scrittura e il debug del codice viene eseguito sul PC-host dal quale si ricava il codice eseguibile dal processore del target mediante l'utilizzo di un opportuno cross compilatore fornito nella toolchain di Freescale. Infatti, una volta ottenuto l'eseguibile per ARM ci basterà eseguirlo sul target per valutarne le effettive prestazioni della nostra applicazione. Lo schema sottostante riporta il ciclo delle operazioni effettuate in fase di scrittura e test dell'algoritmo sviluppato.



Come si nota dal grafico la maggior parte del lavoro viene svolto sull'host-PC in quanto soltanto la fase di "*Test su TARGET*" viene eseguita sulla scheda i.MX51. Ciò evidenzia come sia possibile, al giorno d'oggi, sviluppare applicazioni anche per più schede target, lavorando da un singolo host-PC dotato degl'opportuni strumenti di cross compilazione e senza aver bisogno necessariamente, in prima battuta, dell'hardware della scheda target. Questo modo di lavorare riduce sicuramente i tempi di arrivo e i costi complessivi dell'applicazione sul mercato (*time to market*). Nel nostro caso abbiamo a disposizione come host-PC un Intel i7 che presenta ben otto core, ognuno dei quali lavora a frequenze superiori ai 2 GHz e sul quale abbiamo

installato come sistema operativo Ubuntu 9.04. Quindi gli strumenti utilizzati per la cross compilazione ci permetteranno di poter generare tramtie l' architettura x86 dei file eseguibili non su x86, ma bensì sull'architettura ARM del nostro target.

Avendo già a disposizione nel kernel creato tramite LTIB la libreria OpenCV1.1, non ci resta che configurare opportunamente il cross compilatore e procedere successivamente all'esecuzione dei file eseguibili attraverso la scrittura di opportuni bash script che saranno richiamati opportunamente da shell in user space.

3.4.1 Download della toolchain e configurazione del cross compilatore

Innanzitutto abbiamo bisogno di una toolchain che contiene un compilatore in grado di poter abilitare il NEON in fase di compilazione. Per questo è stato necessario scaricare dal sito di Codesourcery [9] la toolchain *lite* per GNU/Linux all'interno della quale troviamo il compilatore desiderato. Una volta scaricata la toolchain, andiamo ad installarla nel seguente path:

```
$ opt/freescale/usr/local/gcc-4.1.2-glibc-2.5-nptl-3
```

Successivamente è necessario eseguire un comando di "make clean" nella cartella build di opencv :

```
$ cd Scrivania/work/ltib/rpm/BUILD/opencv-1.1.0
$ make clean
```

Il passaggio successivo consiste nel modificare il file "opencv.spec" contenuto all'interno della directory di opencv1.1 costruita da LTIB. All'interno di questo file andiamo ad inserire le opzioni da passare al compilatore per abilitare il NEON. Per fare ciò eseguiamo i seguenti comandi:

```
$ sudo gedit <LTIB_DIR>/dist/lfs-5.1/opencv/opencv.spec &
```

Con questo commando apriamo il file "opencv.spec" con l'editor di testo **gedit**. All'interno di tale file andiamo ad inserire le seguenti righe:

[9] http://www.codesourcery.com/sgpp/lite/arm/portal/package7853/public/arm-none-linux-gnueabi/arm-2010.09-50-arm-none-linux-gnueabi.bin

\CFLAGS="-mcpu=cortex-a8 -mfloat-abi=softfp -mfpu=neon -ftree-vectorize" \

CPPFLAGS="-mcpu=cortex-a8 -mfloat-abi=softfp -mfpu=neon -ftree-vectorize"

Adesso è necessario rilanciare il comando **.\ltib** per rigenerare tutto il sistema creato in precedenza in modo tale che le librerie vengano ricompilate opportunamente e il compilatore venga configurato con la possibilità di abilitare le opzioni per l'attivazione delle istruzioni per il NEON.

3.4.2 Realizzazione degli script utilizzati in fase di compilazione

Una volta scritto il codice C della nostra applicazione è necessario compilarlo da shell tramite l'ausilio del cross compilatore descritto nei paragrafi precedenti. Per questo si è deciso di realizzare un opportuno script che contenesse tutte i comandi da eseguire ogni qual volta si vuole compilare un file con estensione .c o .cpp. In appendice A.1 è possibile trovare lo script "*build_arm.sh*" realizzato in linguaggio *Bash*[10] ed utilizzato per la compilazione del file eseguibile per architettura ARM. All'inizio del file vengono definiti le variabili "CPP" e "prefix" che rappresentano rispettivamente il nome del cross compilatore della toolchain e il path da cui caricare tutte le librerie che troviamo elencate nelle opzioni del compilatore (-lgtk, –lx11, ecc.). Infine, dopo l'aggiunta di tutte le librerie necessarie alla compilazione, troviamo le opzioni che riguardano l'attivazione del NEON e come esso viene utilizzato in fase di compilazione. In particolare abbiamo attivato le seguenti opzioni :

- *-mcpu=cortex-a8*: specifica il core utilizzato.
- *-mfloat-abi=softfp*: indica quale floatin point ABI utilizzare. In questo caso con softfp si consente la generazione di codice usando istruzioni floating point hardware, nel nostro caso quelle relative al NEON.
- *-mfpu=neon* : utile per abilitare l'uso delle NEON intrinsic
- *-ftree-vectorize* : è l'opzione che permette al compilatore gcc di sfruttare le operazioni di vettorizzazione fatte dal NEON.
- *-ftree-vectorize-verbose=1* : stampa sull'output video quali loop sono stati vettorizzati in modo da verificare se qualche ottimizzazione è stata realmente

[10] http://wiki.ubuntu-it.org/Programmazione/LinguaggioBash

effettuata. Inoltre indica anche quale codice non è stato possibile vettorizzare e quale problema si è verificato.

- -O3: permette di introdurre alcune particolari ottimizzazioni dei loop.

Una volta scritto lo script build_arm.sh è necessario effettuare un export delle variabili d'ambiente nella shell in cui stiamo compilando. Precisamente bisogna eseguire da ogni shell in cui si vuole compilare, il seguente comando:

```
$ export  PKG_CONFIG_PATH=/home /marco/Scrivania /work /ltib/rpm/
BUILD/opencv-1.1.0
```

Per comodità copiamo il nostro script build_arm.sh all'interno della cartella /usr/bin e rendiamolo eseguibile con il solito comando chmod 777. Grazie a queste due semplici operazioni siamo in grado di richiamare lo script da qualsiasi path ci troviamo in fase di compilazione.

Adesso siamo pronti a compilare il nostro file "*esempio.cpp"* eseguendo lo script da shell in questo modo:

```
$build_arm.sh   esempio.cpp
```

Verifichiamo di aver prodotto il file eseguibile dando il comando ls. L'eseguibile ha lo stesso nome del file con entente il codice ma privo dell'estensione .c o .cpp. Un'ultima operazione che possiamo fare per assicurarci che l'eseguibile ottenuto sia compatibile con l'architettura ARM è la seguente:

```
$ readelf -h esempio
```

Il risultato prodotto da tale comando è simile a quanto riportato di seguito:

```
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           ARM
  Version:                           0x1
```

```
Entry point address:               0x90d0
Start of program headers:          52 (bytes into file)
Start of section headers:          89380 (bytes into file)
Flags:                             0x5000002,has entry point,Version5
(…)
```

Il comando **readelf** visualizza informazioni su uno o più file oggetto in formato ELF (Executable and Linking Format). Le opzioni controllano quale particolare informazione visualizzare, nel nostro caso con l'opzione –h andiamo a visualizzare le informazioni contenute all'inizio del file ELF (ossia l'header). In particolare siamo interessati al campo "**Machine**" dove la sigla ARM sta ad indicare l'architettura per il quale l'eseguibile è stato compilato. Qualora fossero stati commessi degl'errori in fase di configurazione del cross compilatore, potremmo trovarci che al posto della sigla ARM ci ritroviamo X86. Ciò significa che stiamo compilando per l'architettura del nostro host-PC e quindi bisogna rivedere i passaggi svolti in precedenza.

Una volta terminate correttamente le fasi precedenti è possibile sviluppare il codice C/C++ per architettura ARM. Nel momento in cui andiamo ad avviare la fase di boot della nostra scheda i.MX51, tutto ciò che si trova nella cartella seguente del host-PC:

```
/<LTIB_DIR>/rootfs/
```

Verrà caricato nel sistema operativo della nostra scheda target. Se eseguiamo il comando ls dalla shell del nostro dispositivo target, ci troveremo il tipico "linux tree" composto dalle principali cartelle come /home , /bin, /lib ecc. Come ultima operazione si consiglia di realizzare una propria cartella di lavoro "work" nell'ambiente di lavoro dell'utente. Ad esempio :

```
$cd /home/user

$mkdir work
```

In questo modo ci siamo riservati uno spazio in cui andare a inserire tutti i file eseguibili ottenuti dall cross compilatore sul PC-host.

Capitolo 4

4. L'applicazione sviluppata

4.1 Obiettivo principale

In questo capitolo analizzeremo con cura quali sono state le fasi realizzative e i problemi riscontrati nel corso dello sviluppo dell'applicazione oggetto di questa tesi. Innanzitutto, cerchiamo di capire qual è l'idea principale sulla quale si basa questo lavoro di tesi. Come già detto nei capitoli precedenti, si vuole sfruttare a pieno le potenzialità di invarianza dei descrittori SURF estratti da un'immagine generica. Inoltre, uno dei primi dubbi da chiarire è stato quello di verificare se tale algoritmo permettesse di riconoscere un insieme di immagini campione, prese come riferimento, a partire da una serie di immagini scattate da una qualsiasi fotocamera di uso comune. In particolare, questo riconoscimento viene effettuato in un contesto molto comune come quello di mostre ed esposizioni di quadri.

Si immagini di essere ad una mostra di quadri ad esempio di arte contemporanea, o di Picasso, oppure di Monet o di un qualsiasi altro artista più o meno famoso. Di tutti i quadri che ci sono nella sala abbiamo a disposizione tutte le immagini "originali" che ci saranno utili successivamente per il riconoscimento. A tutti i visitatori che vengono ad ammirare le opere in mostra, viene fornita la possibilità, ad esempio, di scaricare sul proprio cellulare, un'applicazione che è in grado di riconoscere e fornire informazioni dettagliate su tutti i quadri della sala. Questa è soltanto una delle soluzioni possibili, infatti una seconda soluzione potrebbe essere quella di realizzare un dispositivo sul quale installare l'applicazione sviluppata. Ad ogni modo in questo lavoro di tesi non ci siamo preoccupati di questo aspetto sicuramente fondamentale tanto quanto complesso da realizzare.

Arrivati a questo punto, chiunque potrebbe chiedersi legittimamente: "Ma non è vietato fare foto ai quadri nelle mostre?". Giusta osservazione, ma perché si applica questo divieto?. In realtà le foto sono proibite in quanto il vero pericolo al quale i quadri incombono sono le cosiddette lampade ad incandescenza e i flash. Questo perché ciò che viene reputato dannoso è il calore prodotto da queste fonti luminose.

Ad ogni modo, i comportamenti che i musei assumono nei confronti dei visitatori sono tutti dettati dall'attuale decreto in vigore, ossia il Decreto Legislativo 22 gennaio 2004, n. 42- "Codice dei beni culturali e del paesaggio"[11] entrato in vigore a partire dal 1 Maggio 2004. Ciò che accade è che nella maggior parte dei casi risulta proibito effettuare riprese amatoriali quando, tali riprese, possono essere in concorrenza con gli interessi *economici* di una struttura (anche privata), che stia pagando una concessione per vendere in esclusiva cartoline, immagini o libri all'interno della struttura stessa. In sostanza: se fotografare non rappresenta né un pericolo per le opere, né un potenziale elemento di concorrenza a qualche cessionario esclusivo, allora la direzione non proibisce le riprese, il che significa che le autorizza. In caso contrario, laddove venga indicato un espresso divieto alla realizzazione di riprese, risulta *sempre* possibile fare un'esplicita richiesta per chiedere un'eventuale autorizzazione individuale e gratuita ad eseguire riprese amatoriali. Concludendo, una volta avuta l'autorizzazione da parte dell'istituto che possiede le opere è lecito effettuare riprese anche amatoriali ammesso che non siano dannose per le opere stesse.

4.2 La struttura di OpenCV 1.1

Una volta completate le operazioni di sviluppo del kernel Linux e terminata la fase di booting, all'interno della cartella:

/<LTIB_DIR>/rpm/BUILD/opencv-1.1.0

troveremo tutto ciò che la libreria OpenCV 1.1[11] mette a disposizione dello sviluppatore. In particolare, per la scrittura del codice della nostra applicazione, è stato di fondamentale importanza aver a disposizione gli esempi di piccole applicazioni già sviluppate da coloro che hanno sviluppato la libreria. Infatti, il codice che andremo ad analizzare nei prossimi paragrafi utilizza delle funzioni scritte nell'esempio *find_obj.cpp* contenuto all'interno della seguente cartella :

/<LTIB_DIR>/rpm/BUILD/opencv-1.1.0/sample/c

Tali funzioni permettono di implementare l'algoritmo SURF sfruttando le strutture basilari della libreria OpenCV. Il nostro obiettivo principale è stato quello di

[11] Pubblicato sulla Gazzetta Ufficiale n. 45 del 24 Febbraio 2004 - Supplemento Ordinario n.28

analizzare tali funzioni, modificarle ed usarle opportunamente per scrivere il nostro codice. Ovviamente, al fine di poter ottenere un'applicazione real time robusta è stato necessario verificare che l'implementazione dell'algoritmo realizzato presenti le seguenti caratteristiche:

1. Tempi di acquisizione dei descrittori della singola immagine inferiori al secondo
2. Dimensioni del codice complessivo ridotte
3. Affidabilità e robustezza in fase di matching
4. Tempo totale di matching ridotto

Queste proprietà saranno ricordate man mano durante l'analisi del codice in modo tale da giustificare l'utilizzo di alcune porzioni di codice.

4.3 Il problema dell'affidabilità del matching

Prima di iniziare l'analisi della vera e propria fase implementativa dell'applicazione, ci siamo posti alcuni quesiti di non semplice risoluzione: quali sono i punti cruciali su cui l'algoritmo deve basarsi per ottenere correttamente il matching? Su quali parametri bisogna intervenire per ridurre al minimo la probabilità che l'algoritmo fornisca un matching errato? Per fornire una giustificazione adeguata a queste perplessità, è stato necessario raccogliere un numero significativo di immagini sia di riferimento che da identificare. In poche parole, la prima fase di questo lavoro è stata quella di creare un database di immagini di riferimento. Dopodiché per ciascuna immagine di riferimento trovare delle foto in cui il quadro veniva ripreso in contesti e condizioni diverse.

Convenzione utilizzata: Per evitare equivoci, da ora in poi chiameremo con "**Immagine di riferimento**" quella contenuta all'interno del database e che bisogna identificare. Indicheremo invece con "**Immagine Utente**" la foto scattata dal visitatore nella quale bisogna andare ad effettuare la scansione per verificare la presenza o meno di una o più immagini di riferimento.

Per esprimere meglio il concetto trattato in questo paragrafo, nella pagine successive mostriamo alcune foto che sono state utilizzate in fase di test dell'algoritmo. Queste foto, insieme a quelle riportate in Appendice C, rappresentano l'insieme completo di tutte le foto utilizzate per testare l'applicazione. Come è possibile notare dalle immagini e a conferma di quanto già detto in precedenza, abbiamo scelto come contesto di riferimento quello di un museo o di una mostra di quadri.

Al fine di poter verificare l'affidabilità dell'algoritmo non abbiamo scelto delle foto molto professionali, anzi, abbiamo prediletto tutte quelle foto in cui il quadro non era l'unico oggetto di interesse. Infatti, come si nota dalle foto riportate troviamo anche la presenza dei visitatori della mostra. Notiamo in particolare nella foto 2.1 un piccolo dettaglio: fate caso al libro della donna, sulla copertina troviamo la stessa immagine del quadro che vogliamo identificare. Quindi, l'algoritmo in questo caso dovrebbe riconoscere sia il quadro sulla copertina che quello in esposizione. Quanto meno potremmo aspettarci che vengano trovati dei descrittori in comune sia sul quadro che si trova sulla parete che quello raffigurato sulla copertina del libro. Il comportamento ideale da parte dell'applicazione sarebbe quello di identificare entrambe le immagini.

Figure 4.1: A sinistra troviamo le immagini di riferimento(x.0) mentre a destra le foto scattate da semplici visitatori di un museo(x.1).

Foto 1.0 e 1.1: "Dottor Gachet", V. Van Gogh.

Foto 2.0 e 2.1: "Dans la praire" , C. Monet.


Foto 3.0 e 3.1: "Bacino delle ninfee", C. Monet.

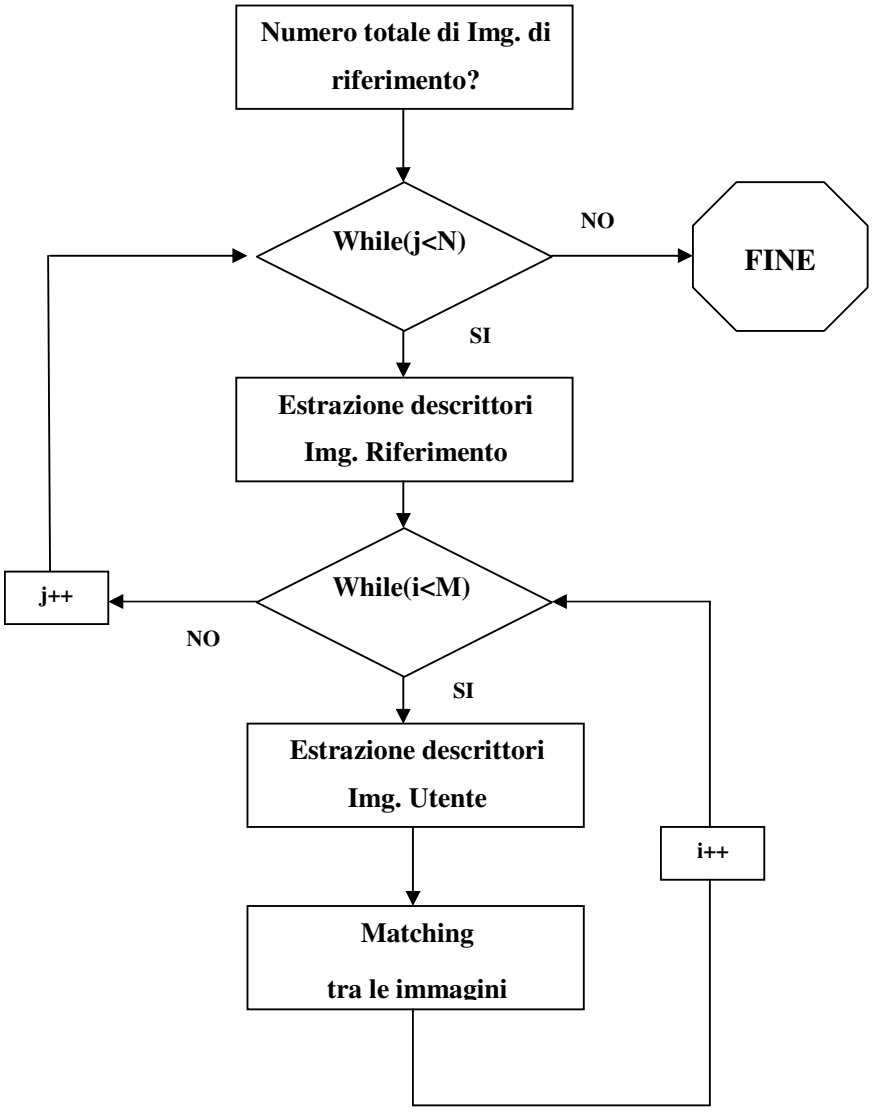
Il risultato del matching di questo quadro sarà presentato nel capitolo successivo insieme a tutti i gli altri risultati ottenuti. Infine, mettiamo in evidenza un ultimo particolare che riguarda la coppia di foto 3.0 e 3.1. L'immagine di riferimento di questo quadro è stata scelta per tentare di ingannare l'algoritmo. Infatti, guardando attentamente i due quadri, si nota che in realtà i colori dei due quadri sono abbastanza diversi tra loro. Ciò non è dovuto semplicemente ad una differenza di luminosità tra le due immagini, la quale potrebbe giustificare le differenze che ci sono tra il quadro di riferimento e quello fotografato dai visitatori della mostra. In realtà il primo è soltanto un'imitazione del quadro originale di Monet e quindi non rappresenta realmente il quadro che ci interessa identificare. In questo caso il corretto funzionamento della nostra applicazione dovrebbe garantire che non vengano associati descrittori in comune tra i due quadri, mettendo a dura prova le caratteristiche di robustezza dell'algoritmo SURF.

4.4 Struttura del codice sviluppato

L'obiettivo principale di questo paragrafo non è tanto quello di vedere nel dettaglio come sono scritte le funzioni che implementano il SURF ma quanto capire come utilizzare tale codice per realizzare l'applicazione di nostro interesse. Di seguito riportiamo le fasi principali che il codice sviluppato implementa:

1. Acquisizione delle immagini di riferimento e utente.
2. Estrazione dei descrittori SURF di entrambe le immagini.
3. Confronto tra i descrittori dell'immagine utente con quelli di riferimento.
4. Verifica del matching ed esposizione del risultato.

Per chiarire meglio qual è la procedura che viene eseguita dal nostro codice riportiamo il seguente diagramma di flusso dove con N ed M rappresentano rispettivamente il numero totale di immagini di riferimento e utente:



Come si nota dal diagramma di flusso il codice è stato diviso in due cicli principali. Possiamo riassumere tutta la procedura attraverso le seguenti fasi:

1. Si inseriscono tutte le immagini da elaborare (sia riferimento che utente).
2. Si fornisce il numero totale delle immagini di riferimento.
3. Vengono estratti i descrittori della j-esima immagine di riferimento.
4. Vengono estratti i descrittori della i-esima immagine utente.
5. Per ogni immagine utente viene effettuato il confronto con i descrittori della j-esima immagine di riferimento.
6. Una volta terminati i confronti tra tutte le immagini utente e l'immagine j-esima di riferimento si passa all'immagine di riferimento successiva.
7. Tutti i passi precedenti vengono ripetuti per tutte le immagini di riferimento, dopodiché l'applicazione termina.

Come si nota dalla realizzazione dell'algoritmo sia il numero di operazioni relative all'estrazione dei descrittori che il numero di confronti effettuati è pari al prodotto N*M. Tale scelta non è sicuramente la migliore in termini di prestazioni. Questo perché non si è voluto ottimizzare l'algoritmo in base a come esso effettua i confronti e come viene realizzata l'acquisizione dei descrittori. Infatti, questa tesi si concentra sulla riduzione dei singoli tempi di estrazione e di matching delle immagini acquisite, in modo tale da garantire le migliori prestazioni da cui partire per un'eventuale sviluppo futuro dell'applicazione (organizzazione del database dei descrittori, miglioramento della modalità di confronto, ecc…).

4.4.1 Inserimento delle immagini da shell

Il punto di partenza della nostra applicazione è quello di inserire le immagini che saranno soggette all'elaborazioni successive. Per eseguire questa operazione sulla nostra applicazione bisogna sapere a priori qual è il numero totale delle immagini di riferimento e quali sono tutte le immagini utente. Infatti, il ***main*** [12] della nostra applicazione è stato realizzato in modo tale da acquisire da shell i riferimenti di tutte le immagini da elaborare. Di seguito riportiamo un esempio di avvio della nostra applicazione in cui vengono inserite 3 immagini di riferimento e 2 utente:

[12] Vedi Appendice B

```
$ ./my_project    rif_1.jpg rif_2.jpg rif_3.jpg  /home/user/image/img_A.jpg
/home/user/image/img_B.jpg
```

Dove *my_project* è il nome del nostro file eseguibile seguito dai path delle foto da analizzare. Si ricorda che per eseguire correttamente questa operazione il path della shell deve far riferimento alla cartella contenente il file *my_project* che a sua volta è stato opportunamente reso eseguibile attraverso il comando ***chmod***.

4.4.2 Acquisizione delle immagini di riferimento e utente

Passiamo adesso ad analizzare con più dettaglio il codice sviluppato. Una delle prime operazioni effettuate è l'acquisizione e conversione delle immagini in scala di grigi. Di seguito riportiamo alcune righe di codice che ci permettono di mostrare la funzione per acquisire e convertire in scala di grigi un'immagine (contenuta all'interno del package "cv.h"):

```
IplImage* object = cvLoadImage( object_filename,CV_LOAD_IMAGE_GRAYSCALE );
```

Questa funzione permette di creare un puntatore alla struttura di tipo IplImage (una delle strutture basilari per descrivere un immagine in OpenCV). Come parametri vengono passati il nome dell'immagine e un'opportuna flag che nel nostro caso ci permette di caricare l'immagine direttamente in bianco e nero. Grazie a questa funzione riusciamo sia ad effettuare l'operazione di creare in memoria un riferimento alla nostra immagine, sia ad effettuare la conversione dell'immagine nella scala di colori desiderata. Questa operazione è necessaria in quanto per l'acquisizione dei descrittori il SURF lavora con immagini in scala di grigi e non a colori. OpenCV mette a disposizione anche una funzione che effettua solo la conversione dei colori che ci è tornata utile per riconvertire le immagini nel formato iniziale una volta terminata l'estrazione dei descrittori SURF. La funzione in questione è la seguente:

```
cvCvtColor( object, object_color, CV_GRAY2BGR );
```

Dove object è l'immagine iniziale in scala di grigi, object_color è l'immagine da creare alla fine della conversione. Il flag CV_GRAY2BGR indica la conversione da

scala di grigi a quella a colori (Blue Green Red). Ovviamente queste sono solo alcune delle funzioni messe a disposizione da OpenCV per l'elaborazione delle immagini che quindi, mostra di poter real realizzare in poche righe di codice anche elaborazioni piuttosto complesse.

4.4.3 Estrazione dei descrittori SURF

Passo successivo è quello di estrarre i descrittori SURF dalle immagini di riferimento e utente. Anche per questa operazione la libreria OpenCV mette a nostra disposizione una funzione apposita nella quale sono stati implementate tutte le funzioni matematiche che abbiamo descritto nel capitolo 2 (ossia tutte quelle operazioni matematiche che permettono di giungere alla corretta estrazione dei descrittori SURF). Riportiamo anche in questo caso alcune righe di codice per conoscere come è stata utilizzata la funzione suddetta:

```
CvSeq  *objectKeypoints = 0, *objectDescriptors = 0;
CvMemStorage*  storage = cvCreateMemStorage(0);
CvSURFParams  params = cvSURFParams(soglia, 1);
cvExtractSURF( object, 0, &objectKeypoints, &objectDescriptors, storage,
params );
```

Questa operazione richiede alcune considerazioni iniziali da fare per capire correttamente come si realizza l'estrazione dei descrittori. Innanzitutto vediamo come viene definita la struttura di tipo **CvSURFParams** all'interno del package "cvsurf":

```
typedef struct CvSURFParams
{
  int extended;
  double hessianThreshold;
  int nOctaves;
  int nOctaveLayers;

}CvSURFParams;
```

Dove i parametri indicati sono:

- **Extended**: Se vale 0 si utilizzano i descrittori basilari a 64 elementi, se invece vale 1 si utilizzano i descrittori da 128 elementi.

- **hessianThreshold**: è il valore di soglia minimo dell'Hessiano. Tutti i descrittori che hanno Hessiano inferiore a tale soglia non saranno considerati utili.
- **nOctaves**: numero di ottave utilizzate per l'estrazione, di default è pari a 3.
- **nOctavesLayer**: è il numero di layer per ogni ottava, di default è pari a 4.

Nel nostro caso utilizzeremo i descrittori basilari a 64 elementi e i valori di default per il numero di ottave e il numero di piani per ogni ottava. Per quanto riguarda il valore della soglia di default si consiglia un valore compreso tra 300-500. In realtà l'ottimizzazione delle prestazioni della nostra applicazione dipende fortemente dal valore della soglia. Per ora non entriamo troppo nel dettaglio in merito a come è stato ricavato il valore ottimale della soglia. Soltanto mostrando i risultati ottenuti in fase di testing dell'algoritmo, riusciremo a dimostrare in maniera significativa il valore numerico scelto per questo parametro.

Una volta descritta la struttura **params**, non ci resta che analizzare la funzione che estrae i descrittori: **cvExtractSURF()** [11]. Come già detto in precedenza non ci addentriamo nel codice che implementa questa funzione in quanto esula dagli scopi di questa tesi. I parametri di ingresso di questa funzione sono i seguenti:

- "***object***": è l'immagine convertita in scala di grigi.
- "***0***": indica che non viene utilizzata una maschera opzionale per ricavare i descrittori ma si utilizza quella di default prevista dall'algoritmo.
- "***storage***": Indica l'area di memoria precedentemente allocata dove saranno memorizzati i descrittori e i keypoints ricavati.
- "***params***": è la struttura che abbiamo analizzato in precedenza da cui la funzione preleva i parametri per implementare l'algoritmo SURF.

I parametri di output sono:

- "***objectKeypoints***": E' la sequenza di punti caratteristici individuati nell'immagine. Ogni punto ha una struttura di tipo CvSURFPoint riportata in appendice A.2 .
- "***objectDescriptors***": E' la sequenza dei descrittori estratti dall'immagine utilizzando la tecnica SURF. Il numero di descrittori soprattutto dal valore di soglia impostato nella struttura "***params***".

Quindi, una volta inseriti opportunamente i parametri troveremo i descrittori di nostro interesse all'interno della sequenza **objectDescriptors**. Quest'operazione di estrazione va effettuata sia per le immagini di riferimento che quelle utente. Di seguito esponiamo un'unica volta questo procedimento, in quanto è possibile iterarlo cambiando semplicemente il riferimento dell'immagine da analizzare.

Infine, ricordiamo ancora una volta che, per ora, non ci siamo addentrati nel dettaglio su come va dimensionato il valore della soglia dell'Hessiano. Questo perché, per capire come scegliere correttamente la soglia, bisogna mostrare il comportamento complessivo dell'applicazione in funzione di essa, cosa che sarà mostrata meglio in seguito quando riporteremo i risultati ottenuti al termine dei test effettuati.

4.4.4 Funzione per il matching dei descrittori : findPairs()

Arrivati a questo punto abbiamo a nostra disposizione sia i descrittori dell'immagine di riferimento che quelli dell'immagine utente e quindi non ci resta che eseguire il confronto. Per effettuare questa operazione abbiamo sfruttato la funzione **findPairs()** riportata con dettaglio in appendice A. Come si può notare dal codice in realtà l'operazione di confronto tra descrittori viene realizzata attraverso una serie di funzioni che vengono chiamate ricorsivamente. La funzione principale tra tutte quelle che vengono chiamate è la seguente: **naiveNearestNeighbor()**, chiamata all'interno di **findPairs().** Di seguito riportiamo la parte principale del codice che implementa questa funzione:

```
if( laplacian != kp->laplacian )
            continue;

        d = compareSURFDescriptors( vec, mvec, dist2, length );

if( d < dist1 )
        {
            dist2 = dist1;
            dist1 = d;
            neighbor = i;
        }
        else if ( d < dist2 )
            dist2 = d;
    }
    if ( dist1 < 0.6*dist2 )
        return neighbor;
    return -1;
}
```

Si analizzano le sequenze dei descrittori dell'immagine di riferimento e utente. Si prende l'i-esimo descrittore della sequenza di riferimento e l'i-esimo descrittore della sequenza utente. Su questa coppia di descrittori vengono confrontati i *laplaciani* di ciascuno di essi: se non corrispondono allora sicuramente i due descrittori non sono uguali e la funzione termina l'analisi di questi descrittori passando poi ai successivi. Dopodiché, viene chiamata la funzione **compareSURFDescriptors()** la quale calcola essenzialmente la distanza euclidea tra i due descrittori. Una volta calcolato questo parametro la funzione effettua dei confronti tra le distanze calcolate tra i vari descrittori. Questi confronti si basano su risultati empirici ottenuti da parte degli autori che hanno sviluppato e inserito nella libreria OpenCV il codice analizzato. Infatti, la funzione **naiveNearestNeighobr()** considera come coppie di descrittori significative, tutte quelle che presentano una distanza euclidea inferiore al 60% della più grande di tutte le distanze euclidee calcolate (per ulteriori dettagli si veda **[12]** a pag. 20).

Una volta terminata l'esecuzione dei confronti tra tutti i descrittori, la funzione **findPairs()** fornisce come parametro di uscita il vettore di interi **ptpairs** il quale contiene gli indici della sequenza dei descrittori trovati in comune (e non i descrittori stessi). In particolare, troviamo: nelle posizioni pari l'indice del descrittore dell'immagine di riferimento e nelle posizioni dispari l'indice del descrittore dell'immagine utente. Se viene trovato almeno un descrittore in comune tra le due immagini, allora la dimensione del vettore sarà pari a 2 (dimensione minima). All'interno del codice riportato in appendice troviamo implementata anche la scrittura su file delle caratteristiche dei descrittori(dove il file viene chiamato "descrittori.txt"). Tale funzionalità risulterà utile in fase di analisi dei risultati e viene effettuata se solo se la dimensione del vettore **ptpairs** è maggiore di uno, in modo da garantire che contenga almeno una coppia di descrittori.

4.4.5 Ulteriori proprietà e presentazione delle immagini

In quest'ultimo paragrafo riportiamo tutte quelle funzioni che non sono state usate in fase di testing dell'algoritmo ma che possono essere abilitate qualora desiderato.

Una funzione che troviamo implementata è la **locatePlanarObject()**. Questa viene utilizzata per localizzare l'immagine di riferimento all'interno di quella utente. Essenzialmente esegue delle operazioni matematiche per poi tracciare un rettangolo intorno all'area localizzata, quindi è una funzione puramente grafica. Una seconda

operazione che abbiamo disabilitato è quella che prevedeva di disegnare su entrambe le immagini dei cerchi che individuassero la posizione di ciascun descrittore. Infine, vi è la possibilità di abilitare una terza funzione che permette di disegnare una linea che ha come estremi i descrittori comuni trovati nelle due immagini. Tutte e tre queste operazioni sono di tipo grafico e tornano utili per capire se i descrittori raccolti sono in posizioni particolari dell'immagine (ad esempio se il descrittore viene raccolto nei pressi della cornice del quadro e non sul quadro stesso). Come tutte le funzioni grafiche anche queste richiedono delle elaborazioni non indifferenti e per questo, abbiamo preferito disabilitarle nel momento in cui siamo andati a misurare i tempi che ci interessano (nell'appendice è possibile identificarle attraverso i commenti riportati). Questo perché in questo lavoro di tesi siamo interessati ai soli tempi di estrazione e di matching dei descrittori, tutte le ulteriori elaborazioni porterebbero soltanto ad un appesantimento del codice e ad ulteriori ritardi temporali indesiderati.

L'applicazione si conclude mostrando sia l'immagine di riferimento che quella utente nell'ipotesi in cui esiste almeno una coppia di descrittori in comune tra le due immagini. Se invece il matching non ha riportato nessuna coppia di descrittori, allora viene mostrato un messaggio di testo che riporta l'esito negativo del confronto. La scelta di riportare un esito positivo del matching basato sul rilevamento di almeno una coppia di descrittori sarà descritto nel prossimo capitolo.

Capitolo 5

5. Risultati Finali

5.1 Introduzione

Il capitolo precedente ci ha permesso di evidenziare diversi aspetti che hanno portato alla nascita dei seguenti quesiti:

- Qual è la soglia minima da scegliere in fase di estrazione? Da cosa dipende il suo valore?
- E' possibile stabilire l'esito positivo del matching basandosi sul fatto che viene trovata soltanto un'unica coppia di descrittori in comune tra le due immagini?
- Quanto è affidabile questa scelta?

Lo scopo di questo capitolo è quello di giustificare le scelte effettuate attraverso i risultati ottenuti in fase di test. Per questo riportiamo di seguito tutte le procedure seguite che ci hanno permesso di individuare quali sono i parametri su cui agire per ottimizzare sia le prestazioni real time che l'affidabilità del matching.

Come vedremo, le prime fasi mirano ad individuare quali sono i parametri da settare in modo tale da minimizzare i tempi necessari per effettuare il singolo confronto tra le due immagini. Successivamente si passerà all'analisi in cui si effettua la ricerca dell'immagine utente all'interno del set di immagini di riferimento a disposizione.

5.2 Elaborazione iniziale delle immagini

Inizialmente abbiamo testato la nostra applicazione con le immagini riportate sia in Appendice C che nel paragrafo 4.3, nelle loro effettive dimensioni e senza alcuna elaborazione grafica su di esse. Inoltre, abbiamo compilato il nostro codice sorgente senza abilitare il NEON in modo tale da capire quali sono le prestazioni iniziali della nostra applicazione. Possiamo dire subito che sotto queste condizioni, le prestazioni ottenute risultano essere davvero scadenti. Infatti, i tempi medi di elaborazione delle immagini sia in fase di estrazione dei descrittori che in fase di matching degli stessi,

superano addirittura i 30 secondi. Per questo abbiamo deciso di non riportare i risultati ottenuti in questa fase e di passare direttamente all'analisi delle prestazioni ottenute con NEON abilitato.

5.2.1 Elaborazione delle immagini con NEON Abilitato.

Abilitando le opzioni opportune del compilatore GCC in modo da attivare il NEON in fase di compilazione, siamo riusciti ad ottenere dei risultati che ci hanno permesso di ridurre, rispetto al caso precedente, di circa un terzo il tempo di matching e di estrazione. I risultati sono riportati nella seguente tabella:

Immagini	**8.0/8.1**	**9.0/9.1**	**10.0/10.1**
Tempo di estrazione [ms]	29982.4	30143.6	20401.0
Tempo di Matching [ms]	56646.3	13370.3	1982.7
Coppie di Descrittori	7960	5217	975

I risultati riportati in questa tabella si riferiscono alle immagini riportate in Appendice C. Il "tempo di estrazione" si riferisce all'operazione di estrazione dei descrittori SURF relativi alla sola immagine utente trascurando il tempo necessario per estrarre i descrittori dell'immagine di riferimento. Questo perché in realtà i descrittori delle immagini di riferimento vengono estratti e memorizzati a priori in un apposito database. Mentre per quanto riguarda i descrittori dell'immagine utente, essi devono essere estratti ogni volta che viene scattata una foto (ad esempio dal cellulare). Il tempo di matching invece, si riferisce al solo tempo che viene impiegato dalla funzione **findPairs()** per ricavare il vettore **ptpairs** così come descritto nel capitolo precedente. Per misurare questi tempi sono state usate delle particolari funzioni della libreria OpenCV che permettono di "contare" il numero di colpi di clock trascorsi.

Di seguito riportiamo le seguenti righe di codice come esempio di misura del tempo di matching, implementato nel codice complessivo dell'applicazione e consultabile nell' appendice B:

```
t_match = (double)cvGetTickCount();//start
findPairs( objectKeypoints, objectDescriptors, imageKeypoints, imageDescriptors, ptpairs );
t_match = ((double)cvGetTickCount()-t_match)/(cvGetTickFrequency()*1000.);//stop
```

Nella prima riga troviamo un casting a double del numero di colpi di clock (*tick*) ricavati tramite la funzione **cvGetTickCount**(). Dopo aver chiamato la funzione **findPairs** e ricavato le coppie di descrittori in comune tra le due immagini, troviamo il calcolo vero e proprio del tempo di match: ricaviamo di nuovo il numero di colpi di clock e lo sottraiamo al numero di tick precedenti. Il tutto viene riportato in millisecondi attraverso la funzione **cvGetTickFrequency**() che ci permette di ricavare la frequenza di lavoro della nostra cpu.

Per questo primo test abbiamo scelto una soglia pari a **1000**. Una primo dettaglio che vale la pena sottolineare, è che i tempi di elaborazione sono tanto più elevati quanto più grande è il numero delle coppie di descrittori trovati in comune tra le due foto. Sebbene le prestazioni siano migliori rispetto al caso in cui il NEON non veniva abilitato, i tempi sono ancora troppo alti. Infatti, se vogliamo che la nostra applicazione consenta un elaborazione real time delle immagini bisogna ridurre drasticamente questi tempi di elaborazione.

5.3 Scaling delle Immagini

In virtù della proprietà del SURF di invarianza alla scala delle immagini, abbiamo deciso di effettuare uno scaling delle immagini da analizzare. Di seguito si mostrano tre tabelle in cui si riportano le performance ottenute scalando di un fattore 4, 8 e 16 le dimensioni in pixel delle immagini con valore della soglia fissato a **1000**. Lo scopo ovviamente è quello di capire come variano i tempi di elaborazione e il numero di coppie di descrittori quando la dimensione delle immagini si riduce.

Tabella 5.1: Fattore di Scala=4 , Soglia=1000, Dimensione Immagini=764x573 pixel.

	8.0/8.1	**9.0/9.1**	**10.0/10.1**
Tempo di estrazione [ms]	3178.77	3669.12	1856.12
Tempo di matching [ms]	4999.57	4006.7	577.815
# Descrittori Img. Riferimento	1158	698	461
# Descrittori Img. Utente	1471	1959	445
#Coppie Descrittori	818	116	144

Tabella 5.2: Fattore di Scala=8, Soglia=1000, Dimensione Immagini=382x287 pixel.

	8.0/8.1	**9.0/9.1**	**10.0/10.1**
Tempo di estrazione [ms]	996.391	1101.13	569.035
Tempo di matching [ms]	1038.23	711.263	120.2
# Descrittori Img. Riferimento	565	315	193
# Descrittori Img. Utente	630	750	213
#Coppie Descrittori	416	64	44

Tabella 5.3:Fattore di Scala=16, Soglia=1000, Dimensioni Immagini=191x143 pixel.

	8.0/8.1	**9.0/9.1**	**10.0/10.1**
Tempo di estrazione [ms]	98.98	130.73	76.58
Tempo di matching [ms]	7.47	4.09	1.34
# Descrittori Img. Riferimento	32	24	15
# Descrittori Img. Utente	40	35	20
#Coppie Descrittori	23	19	8

Come possiamo notare dalle tabelle 5.1, 5.2 e 5.3 la semplice operazione di scaling delle immagini ha permesso di passare da tempi di elaborazione superiori ai 30s a tempi inferiori ai 10s. Precisamente, applicando un fattore di scala pari a 4, i tempi di elaborazione non superano i 5s e il numero di coppie di descrittori è ancora elevato. Riducendo ancora le dimensioni di un fattore 8 si ottiene che sia il tempo di matching che quello di estrazione sono prossimi al secondo.

Ricordiamo che per adesso abbiamo impostato il valore della soglia a 1000 ma come si è già detto nel capitolo precedente, il valore della soglia permette di variare il numero di descrittori utili al fine del matching. Per ora abbiamo preferito lasciarlo ad un valore che ci permettesse di raccogliere un numero di descrittori abbastanza elevato in modo da capire come varia la velocità dell'elaborazione al variare di tale numero di punti.

Di seguito riportiamo diverse tabelle in cui elaboriamo i risultati ottenuti in diverse fasi di elaborazione:

FS	8.0	riduzione	8.1	riduzione
1	2598		7960	
2	1768	1,47	3639	2,18
4	1158	2,24	1471	5,41
8	565	4,6	630	12,64
16	**32**	**81,19**	**40**	**199**

Tabella 5.4 A: Numero di descrittori delle immagini 8.0 e 8.1 al variare del fattore di scala (FS)

FS	9.0	Riduzione	9.1	riduzione
1	1016		5217	
2	946	1,07	3869	1,35
4	698	1,46	1959	2,66
8	315	3,22	750	6,96
16	**35**	**29,03**	**24**	**217,37**

Tabella 5.4 B: Numero di descrittori delle immagini 9.0 e 9.1 al variare del fattore di scala (FS)

FS	10.0	Riduzione	10.1	riduzione
1	724		975	
2	630	1,15	719	1,36
4	461	1,57	445	2,19
8	193	3,75	213	4,58
16	**15**	**48,27**	**20**	**48,75**

Tabella 5.4 C: Numero di descrittori delle immagini 10.0 e 10.1 al variare del fattore di scala (FS)

Nelle tabelle 5.4 viene evidenziata la riduzione del numero di punti che si ha passando dalla foto originale (FS=1) a quelle scalate, indicando i vari fattori di scala utilizzati. Nella colonna "riduzione" si riporta il rapporto tra il numero di punti della foto originale diviso il numero di punti della foto con FS indicato. Come si nota, i valori migliori di riduzione si verificano sempre in corrispondenza del **FS=16** e per questo sono stati evidenziati in grassetto. Premesso che il numero di descrittori dipende dal *contenuto* dell'immagine analizzata (per questo non calcolabile a priori), è possibile comunque fare un'osservazione interessante sui dati ottenuti: ci sono delle immagini come la 8.1 e la 9.1 non scalate (FS=1), che presentano un elevato numero di descrittori ed altre come la 10.0 originale che ne hanno molti di meno. Per altro la 10.0 e la 10.1 originali presentano un numero di descrittori profondamente diverso anche se in realtà dovrebbero rappresentare lo stesso oggetto.

Ciò mette in evidenza una peculiarità dell'algoritmo: parte dei descrittori raccolti risultano essere **non significativi** per cui sarebbe bene filtrarli opportunamente.

Un'ulteriore osservazione che possiamo fare è la seguente: riducendo le dimensioni delle immagini il numero di descrittori diminuisce. Questo significa che le immagini con risoluzione maggiore hanno un numero di descrittori meno significativi e che vengono filtrati una volta effettuata l'operazione di scaling. Inoltre, se guardiamo la tabella 5.3 notiamo che il numero di coppie di descrittori risulta essere più vicino al numero minimo di descrittori raccolti tra le due immagini. Ciò ovviamente dimostra come l'operazione di scaling non influisca sull'affidabilità del corretto matching, anzi permette di ridurre il numero di confronti effettuati ai soli descrittori più significativi.

Tale riduzione comporta di conseguenza un aumento della velocità in termini di tempo di matching. In realtà questo miglioramento era del tutto prevedibile in quanto il numero di confronti che effettua la funzione **findPairs()** è tanto più piccolo quanto più piccola è la dimensione dei descrittori delle due immagini analizzate. Come vedremo nel prossimo paragrafo, è invece meno prevedibile scegliere i descrittori più significativi e basare soltanto su di essi il corretto matching dell'immagini.

Nella pagina successiva viene riportata la tabella 5.5 in cui viene calcolato il tempo di elaborazione della singola coppia di descrittori al variare del FS (riportato nella quinta colonna), dividendo il tempo di matching per il numero totale di coppie ricavate.

immagini	FS	#coppie	t. match[ms]	t.match/coppia	fat_riduzione
8	4	818	4999,57	6,11	1
9	4	116	4006,7	34,54	1
10	4	144	577,81	4,01	1
8	8	416	1038,23	2,49	2,45
9	8	64	711,263	11,12	3,11
10	8	44	120,2	2,73	1,47
8	16	23	7,47	**0,32**	**18,82**
9	16	19	4,09	**0,22**	**160,46**
10	16	8	1,34	**0,17**	**23,96**

Tabella 5.5: Rapporto tra i tempi di matching e numero di coppie di descrittori e fattore di riduzione di tale rapporto al variare del FS.

Nell'ultima colonna è stato calcolato di quanto si riduce il tempo suddetto se si passa da un FS pari a 4 ad un FS pari a 16. Ad esempio per le immagini 9 il tempo di elaborazione della singola coppia si riduce di un fattore pari a 160,46 se si passa da uno FS di 4 a un FS di 16. Guardando i tempi di elaborazione riportati nella quarta colonna della tabella 5.5 possiamo notare che: mentre per FS=4 e FS=8 i tempi delle tre immagini risultano essere profondamente diversi, per FS=16 i tempi tendono ad essere più vicini tra loro. Ciò mette ancor più in risalto l'utilità dell'operazione di scaling, mostrando effettivamente che le prestazioni ottimali si ottengono proprio in corrispondenza di un FS=16. Ai fini pratici dell'applicazione, ciò che interessa realmente è la dimensione vera e propria delle immagini ottenute a posteriori dell'operazione di scaling. Riportiamo di seguito le dimensioni delle immagini ottenute per ogni FS applicato:

FS	pixel
1	2292x3056
4	764x573
8	382x287
16	**191x143**

Tabella5.6: Dimensioni in pixel delle immagini 8, 9 e 10 in corrispondenza del FS applicato. L'immagine originale è stata ottenuta da una fotocamera con risoluzione di 7 Mpixel.

Ovviamente le dimensioni indicate sono dei riferimenti approssimativi. Nulla vieta di scegliere delle dimensioni che variano leggermente rispetto a quelle riportate. In tal caso le prestazioni risultano essere sicuramente prossime a quelle mostrate precedentemente.

Nasce quindi un concetto fondamentale su cui possiamo basare l'ottimizzazione della nostra applicazione:

- "*Ridurre il numero di descrittori da utilizzare per il matching ottimizzando le dimensioni delle immagini*"

5.3.1 Analisi dell'Hessiano dei descrittori

Abbiamo appena detto che la riduzione del numero di descrittori porta all'incremento della velocità di elaborazione delle immagini, ma esistono altri parametri, oltre alla dimensione delle immagini, che ci permettono di ridurre questo numero? E' possibile effettuare in qualche modo un *ranking* dei descrittori? Quali sono i descrittori che bisogna ritenere utili ai fini di ottenere un matching corretto?

Cerchiamo di rispondere a tutte queste domande partendo da un elemento che abbiamo descritto nel capitolo 1. Sappiamo che ogni descrittore presenta tra i vari parametri che lo caratterizzano, il seguente parametro fondamentale: l'Hessiano. Abbiamo visto che possiamo decidere di imporre il valore di soglia di estrazione dei descrittori in modo da ricavare solo quelli che hanno un valore dell'Hessiano superiore a tale soglia. Quindi, possiamo utilizzare questa soglia per filtrare tutti quei descrittori che non presentano tale proprietà e che porterebbero soltanto ad un aumento del numero di descrittori da analizzare con conseguente aumento del tempo di matching.

Quindi, oltre all'ottimizzazione delle dimensioni, abbiamo a disposizione un secondo parametro fondamentale che ci consentirà di ridurre il tempo di elaborazione dei descrittori:

- ✓ *"aumentare la soglia dell'Hessiano in fase di estrazione"*

Dall'analisi del paragrafo precedente abbiamo ottenuto la dimensione ottimale delle immagini. Invece, adesso cerchiamo di capire sotto quali condizioni è possibile

ottimizzare la nostra applicazione agendo sul valore di soglia dell'Hessiano dei descrittori estratti.

A tale scopo riportiamo nelle tabelle 5.7 l'analisi dell'Hessiano delle coppie di descrittori ricavati dalle immagini 10 con FS pari ad 8 (si è scelto di riportare i risultati di una sola coppia di immagini per evitare di riportare un'enormità di dati).

In queste tabelle viene stilato un vero e proprio *ranking* dei descrittori in base al valore del loro Hessiano.

Tabelle 5.7: Analisi dei descrittori delle immagini **10.0** e **10.1** con **FS=8** al variare della soglia.

Soglia 3000			
Desc_10.0	**Desc_10.1**	**H_10.0**	**H_10.1**
(93,33 ; 130,67)	**(112,00 ; 168,00)**	**9162,6**	**14165,8**
(174,00 ; 138,00)	**(147,00 ; 162,00)**	**6723,6**	**9189,2**
(246,00 ; 252,00)	**(178,00 ; 226,00)**	**6004,66**	**7376,47**
(246,67 ; 253,33)	**(178,00 ; 226,00)**	**5796,17**	**7376,47**
(170,00 ; 162,00)	**(143,00 ; 177,00)**	**4488,96**	**6127,44**
(60,00 ; 310,00)	**(91,00 ; 261,00)**	**3561,69**	**5346,08**
(256,00 ; 244,00)	**(183,00 ; 220,00)**	**3285,9**	**3693,2**
(60,00 ; 312,00)	**(91,67 ; 263,33)**	**3265,3**	**4304,31**
(256,67 ; 242,67)	**(183,00 ; 220,00)**	**3131,97**	**3693,2**
~~(144,00 ; 306,00)~~	~~(123,33 ; 260,00)~~	~~2666,26~~	~~4579,14~~
~~(60,00 ; 313,33)~~	~~(91,67 ; 263,33)~~	~~2626,98~~	~~4304,31~~
~~(143,33 ; 323,33)~~	~~(123,00 ; 269,00)~~	~~2541,57~~	~~2117,83~~
~~(140,00 ; 306,67)~~	~~(123,33 ; 260,00)~~	~~2419,92~~	~~4579,14~~
~~(190,00 ; 358,33)~~	~~(142,00 ; 289,00)~~	~~2332,73~~	~~3089,73~~
~~(178,00 ; 168,00)~~	~~(146,00 ; 180,00)~~	~~2191,52~~	~~3712,18~~
~~(242,67 ; 240,33)~~	~~(176,67 ; 218,33)~~	~~2036,81~~	~~2062,38~~

Soglia 4000			
Desc_10.0	**Desc_10.1**	**H_10.0**	**H_10.1**
(93,33 ; 130,67)	**(112,00 ; 168,00)**	**9162,6**	**14165,8**
(174,00 ; 138,00)	**(147,00 ; 162,00)**	**6723,6**	**9189,2**
(246,00 ; 252,00)	**(178,00 ; 226,00)**	**6004,66**	**7376,47**
(246,67 ; 253,33)	**(178,00 ; 226,00)**	**5796,17**	**7376,47**
(150,00 ; 176,67)	**(133,33 ; 186,67)**	**4590,36**	**9584,19**
(170,00 ; 162,00)	**(143,00 ; 177,00)**	**4488,96**	**6127,44**

Soglia 6000			
Desc_10.0	**Desc_10.1**	**H_10.0**	**H_10.1**
(93,33 ; 130,67)	**(112,00 ; 168,00)**	**9162,6**	**14165,8**
(174,00 ; 138,00)	**(147,00 ; 162,00)**	**6723,6**	**9189,2**
(246,00 ; 252,00)	**(178,00 ; 226,00)**	**6004,66**	**7376,47**

Nelle tabelle 5.7 troviamo nelle prime due colonne le coordinate che rappresentano la posizione dei descrittori comuni all'interno delle rispettive immagini. Nelle ultime due colonne invece troviamo l'Hessiano di ciascun descrittore. La tabella è stata ordinata partendo dalla coppia di descrittori con Hessiano maggiore fino ad arrivare a quella con Hessiano minore. A colpo d'occhio notiamo subito l'effettiva riduzione del numero delle coppie di descrittori in comune che si ottiene passando da una soglia all'altra. Inoltre, sono state evidenziate in azzurro tutte le coppie che "sopravvivono" all'operazione di filtraggio mentre quelle che vengono filtrate dal passaggio di una soglia all'altra sono state cancellate. In particolare troviamo una riga di colore rosso nella tabella con soglia 4000: stranamente questa coppia di descrittori non è stata trovata con soglia pari a 3000, cosa che invece accade per tutte le altre coppie. Non siamo riusciti a comprendere il motivo per il quale si verifica questa anomalia e per questo non possiamo che attribuirla ad un comportamento erroneo, anche se raro, dell'estrazione dei descrittori. Concludendo possiamo dire che l'aumento del valore della soglia ci ha permesso di ridurre drasticamente il numero di descrittori nonostante le foto analizzate non avessero la dimensione ottimale. Infatti, siamo riusciti a raccogliere soltanto tre coppie di descrittori (riportate in verde), avendo a disposizione immagini con FS pari ad 8 (e non 16), con soglia però pari a 6000.

5.3.2 Procedura per l'ottimizzazione finale

In quest'ultimo paragrafo si riassumono i parametri fondamentali ricavati precedentemente e qual è la procedura d'indagine da seguire per il dimensionamento finale di ciascuno di essi.

Essenzialmente si può agire su due gradi di libertà:

- Dimensione dell'immagine
- Soglia dell'Hessiano dei descrittori

La procedura da seguire per il dimensionamento di questi due parametri è la seguente:

- Verifica dei tempi di elaborazione al variare delle dimensioni.
- Dimensionare tutte le immagini con la dimensione che minimizzi tali tempi.
- Analizzare l'Hessiano dei descrittori al variare della soglia.

- Dimensionare il valore della soglia in modo tale da disporre di un numero piccolo ma significativo di descrittori.

Nel nostro caso abbiamo già visto che la dimensione che minimizza i tempi di estrazione e di matching è quella relativa ad un FS pari a 16, quindi dalla tabella 5.7 ricaviamo che tale dimensione è pari a 191x143 pixel. Dettò ciò abbiamo adottato tale dimensione come standard per tutte le immagini a disposizione e utilizzate per il test. L'operazione di ridimensionamento delle immagini è stata eseguita utilizzando l'editor grafico GIMP (software Open Source compatibile con tutti i sistemi UNIX based).

Successivamente abbiamo indagato sul valore di soglia da impostare andando ad effettuare l'estrazione dei descrittori di tutte le immagini agendo sia sulla soglia stessa sia controllando il numero di coppie di descrittori ottenuti come nel paragrafo precedente. Per semplicità riportiamo nelle tabelle successive soltanto i descrittori con valori di Hessiano superiori a 3000:

Tabelle 5.8: Verifica del numero di coppie di descrittori per la scelta del valore di soglia da applicare.

1.0/1.1			
pt1	**pt2**	**Hessian_1**	**Hessian_2**
(78,00 ; 80,00)	(80,00 ; 60,00)	15591,05	10420,97
(43,33 ; 100,00)	(65,00 ; 71,67)	11453,56	7922,11
(64,00 ; 84,00)	(73,33 ; 61,67)	7922,07	12025,42
(74,67 ; 70,00)	(79,33 ; 53,67)	4022,06	4511,62

2.0/2.1			
pt1	**pt2**	**Hessian_1**	**Hessian_2**
(130,00 ; 23,33)	(82,00 ; 28,00)	6926,61	9044,58
(118,00 ; 44,00)	(76,00 ; 38,00)	6129,63	8043,94
(116,67 ; 44,33)	(76,00 ; 38,00)	6023,42	8043,94
(96,00 ; 40,00)	(65,00 ; 36,00)	5614,45	7589,81

4.0/4.1			
pt1	**pt2**	**Hessian_1**	**Hessian_2**
(21,00 ; 45,00)	(66,00 ; 54,00)	16646,08	5453,49
(86,67 ; 55,00)	(66,00 ; 54,00)	16645,8	5453,49
(69,00 ; 42,00)	(80,00 ; 53,33)	14658,02	9452,86
(135,00 ; 60,00)	(80,00 ; 53,33)	11837,15	9452,86

5.0/5.1			
pt1	**pt2**	**Hessian_1**	**Hessian_2**
(90,00 ; 108,00)	(141,00 ; 21,00)	6659,92	13902,06
(90,00 ; 106,67)	(141,00 ; 21,00)	6973,99	13902,06

7.0/7.1			
pt1	**pt2**	**Hessian_1**	**Hessian_2**
(61,67 ; 173,33)	(69,00 ; 154,00)	3834,22	3663,6

7.0/7.2			
pt1	**pt2**	**Hessian_1**	**Hessian_2**
(11,67 ; 136,67)	(20,00 ; 131,67)	18136,74	18382,31
(31,00 ; 99,00)	(38,00 ; 100,00)	11924,67	12896,15
(37,33 ; 149,33)	(42,00 ; 142,00)	11375,94	12785,49
(37,00 ; 87,00)	(43,00 ; 90,00)	11317,4	10024,68
(38,00 ; 150,00)	(43,33 ; 141,67)	9814,95	12989,45
(44,00 ; 80,00)	(48,00 ; 87,00)	8247,55	8319,41
(43,33 ; 83,33)	(48,00 ; 87,00)	7894,35	8319,41
(21,00 ; 121,00)	(29,00 ; 119,00)	6895,66	7015,54
(39,00 ; 104,00)	(44,00 ; 104,00)	6017,55	8087,86

8.0/8.1			
pt1	**pt2**	**Hessian_1**	**Hessian_2**
(13,00 ; 72,00)	(18,00 ; 78,00)	12682,63	13171,99
(42,00 ; 64,00)	(46,67 ; 70,00)	11050,2	8909,46
(41,67 ; 63,33)	(46,67 ; 70,00)	10762,43	8909,46
(122,00 ; 50,00)	(126,00 ; 58,00)	10182,5	11459,18
(74,00 ; 63,00)	(78,00 ; 70,00)	10135,69	10427,75
(47,00 ; 69,00)	(51,00 ; 76,00)	9319,78	8475,4
(100,00 ; 48,33)	(105,00 ; 55,00)	8179,92	5364,75
(137,00 ; 52,00)	(140,00 ; 60,00)	8019,29	7420,58
(113,33 ; 78,33)	(116,67 ; 85,00)	6381,25	6520,7
(66,00 ; 55,00)	(70,00 ; 62,00)	6261,33	5754,91
(117,00 ; 55,00)	(120,00 ; 62,00)	6236,85	9394,15
(124,00 ; 44,00)	(127,00 ; 52,00)	6038,96	5635,44
(126,67 ; 60,00)	(130,00 ; 66,67)	5806,56	7927,07
(45,00 ; 108,00)	(48,00 ; 114,00)	5779,05	5318,1
(34,00 ; 110,00)	(38,00 ; 117,00)	5300,15	5481,63
(70,00 ; 51,00)	(74,00 ; 58,00)	5018,4	5594,33

9.0/9.1			
pt1	**pt2**	**Hessian_1**	**Hessian_2**
(46,67 ; 65,33)	(56,00 ; 84,00)	9246,66	14243,3
(44,00 ; 64,00)	(56,00 ; 84,00)	8364,74	14243,3
(124,00 ; 128,00)	(89,00 ; 113,00)	6124,01	7162,46

10.0/10.1			
pt1	**pt2**	**Hessian_1**	**Hessian_2**
(65,00 ; 51,67)	(75,00 ; 62,00)	4187,12	4654,25
(101,67 ; 53,33)	(100,00 ; 63,00)	4011,78	6630,2

10.0/10.2			
pt1	**pt2**	**Hessian_1**	**Hessian_2**
(151,67 ; 80,00)	(88,33 ; 45,00)	10640,33	5368,51
(101,67 ; 45,00)	(107,00 ; 50,00)	7353,37	6821,41

Come possiamo notare, mancano alcune foto tra tutte quelle analizzate. In particolare per le immagini 3.0/3.1 e 6.0/6.1 non è stato possibile individuare nessuna coppia di descrittori anche per valori di soglia molto bassi (300-500). Questo perché nel caso della 3.0 abbiamo preso come originale un'imitazione del vero quadro rappresentato nella 3.1. Ciò a conferma di quanto avevamo detto anche nel paragrafo 4.3 riguardo l'affidabilità del matching. Invece, per la 6.1 il quadro del ritratto di Van Gogh è stato fotografato con una rotazione eccessiva che non permette di ottenere correttamente i descrittori da identificare. In questo caso si mette in risalto un'ulteriore limite del SURF: l'invarianza per rotazione è limitata e non ricopre tutto l'intervallo che va da 0° a 360° rispetto alla posizione standard dell'immagine di riferimento[3][8].

Nella tabella successiva riportiamo le performance ottenute settando opportunamente la soglia per ogni immagine in modo da raccogliere un numero non troppo elevato di descrittori (ad esempio meno di 20 descrittori in questo caso).

Tabella 5.9:Fattore di Scala=16, Soglia variabile, Dimensioni Immagini=191x143p

	8.0/8.1	**9.0/9.1**	**10.0/10.1**
Soglia	5000	5000	3000
Tempo di estrazione [ms]	98.98	130.73	76.58
Tempo di matching [ms]	7.47	4.09	1.34
# Descrittori Img. Riferimento	25	35	16
# Descrittori Img. Utente	30	45	20
Coppie di Descrittori	15	3	2

Come si nota dalla tabella i tempi di estrazione non superano i 150 ms mentre i tempi di matching sono addirittura inferiori ai 10 ms in tutte e tre i casi. Inoltre, notiamo

come nella terza colonna la dimensione il numero di coppie di descrittori (che coincide con la dimensione del vettore **ptpairs**), sia pari al minimo consentito ossia una sola coppia di descrittori in comune. Quindi, in questo caso riusciamo a discriminare l'immagine 10.0 anche con una sola coppia di descrittori.

Se effettuiamo una stima del tempo totale medio di elaborazione come la media dei 3 tempi di elaborazioni ottenuti otteniamo il seguente risultato:

$$T_{av} = \frac{T_{tot1} + T_{tot2} + T_{tot3}}{3} = \frac{319{,}19}{3} = 106{,}4\ ms$$

Dove con T_{tot} si è indicato la somma del tempo di matching e tempo di estrazione di ciascuna coppia di immagini. Il risultato a cui siamo giunti ci permette di affermare che, una volta effettuate le ottimizzazioni seguendo la procedura suggerita, mediamente la nostra applicazione elabora ogni singola coppia di immagini in un tempo prossimo ai 100 ms.

5.4 Test di riconoscimento con database di immagini

Nel paragrafo precedente ci siamo limitati ad analizzare le prestazioni dell'applicazione ponendo come input le sole coppie di immagini. Ovviamente questo è il caso in cui sappiamo a priori dell'esistenza di un riscontro positivo nel matching, salvo casi particolari di cui abbiamo già parlato in precedenza.

Invece, adesso ci proponiamo di verificare le prestazioni e l'affidabilità del matching quando si dispone di un database di immagini di riferimento. In questo caso l'applicazione dovrà estrarre i descrittori dell'immagine scattata dall'utente ed effettuare un confronto multiplo con tutte le immagini di riferimento, presentando infine il risultato finale del riconoscimento.

Per questa analisi abbiamo scelto di utilizzare come database tutte le immagini di riferimento viste in precedenza (ossia tutte quelle che denominate come " **.0** "). In realtà non abbiamo costruito un vero e proprio database contenente i descrittori di tutte le immagini. Infatti, abbiamo semplicemente passato in ingresso al codice utilizzato prima, tutte le immagini di riferimento e la sola immagine utente che si voleva

riconoscere. Ciò significa che l'algoritmo estrarrà ogni volta prima i descrittori dell'immagine di riferimento poi quelli dell'immagine utente e dopodiché effettuerà il confronto. La procedura si ripete fin quando non terminano tutte le immagini di riferimento.

Ovviamente questa non è una soluzione ottimizzata. Infatti, un approccio migliore sarebbe quello di estrarre a priori tutti i descrittori delle immagini di riferimento, stabilendo il valore della soglia come indicato nella procedura di ottimizzazione del paragrafo precedente. Fatto ciò non resta che estrarre soltanto una volta i descrittori dell'immagine utente e confrontarli tramite **findPairs()** con tutti i descrittori del database. Ciò nonostante i tempi sono stati calcolati tenendo conto solo dell'operazione di estrazione dei descrittori dell'immagine utente e solo dell'operazione di matching tra le varie immagini, *simulando* effettivamente il comportamento ottimizzato descritto in precedenza. Inoltre, per poter misurare correttamente il tempo totale di elaborazione sono state disabilitate le funzioni di scrittura su file e di visualizzazione grafica.

Tabella 5.10: Confronto delle varie immagini utente con tutte le immagini di riferimento.

IMMAGINE UTENTE	t_match [ms]	t_tot [ms]	soglia	ptpairs
7.1	7,441	**3559,7**	3000	2
7.2	5,122	**3236,7**	5000	18
8.1	8,113	**3171,6**	5000	32
9.1	4,07	**3516,7**	5000	6
10.2	3,63	**3350,2**	4000	4
10.3	13,75	**6716,2**	1000	8

Nella tabella 5.10 abbiamo confrontato le varie immagini utente indicate nella prima colonna, con le immagini di riferimento come descritto in precedenza. La tabella riporta oltre ai parametri di soglia e numero di coppie, anche i seguenti tempi:

- **t_match**: è il tempo di matching corrispondente al solo caso in cui il match produce un numero utile di coppie di descrittori.
- **t_tot**: tiene conto solo del tempo di estrazione dell'immagine utente e del suo confronto con tutte le immagini di riferimento.

Nell'estrazione di questi risultati si sono verificate diverse anomalie rispetto ai casi affrontati in precedenza. Infatti, per le figure 8.1, 9.1, 10.2 impostando una soglia

inferiore rispetto a quella indicata in tabella 5.10 (in particolare con valori di soglia tra 2000 e 3000), l'applicazione individuava delle coppie in comune anche con quadri diversi dai loro riferimenti reali. La presenza di queste ambiguità è imputabile sicuramente alle non idealità dell'algoritmo presenti sia in fase di caratterizzazione del descrittore che in fase di individuazione delle coppie dei descrittori stessi. Inoltre, abbiamo effettuato un controllo sul numero di coppie individuate erroneamente, esso infatti è risultato essere sempre inferiore al numero di coppie ottenute con il vero quadro di riferimento.

Quest'ultimo problema riscontrato ha messo a dura prova l'affidabilità dell'algoritmo nel momento in cui il matching corretto viene stabilito in base al numero di coppie di descrittori ricavate. Infatti, abbiamo già detto che questo numero deve risultare piccolo in modo da ridurre il numero di confronti effettuati. In questi casi nel momento in cui si verifica che il numero di coppie non significative eguagli o addirittura, superi il numero di coppie utili prestabilito, si ottenga un riconoscimento errato (oltre ad un ulteriore aumento dei tempi di elaborazione totali).

Concludendo, questa ulteriore analisi ci conferma che bisogna ottimizzare la scelta della soglia non solo come già visto nei paragrafi precedenti, ma anche in base al confronto tra ogni immagine di riferimento con tutte le immagini che andranno a costituire il database effettivo dell'applicazione. In questo modo si possono evitare le problematiche relative alle ambiguità mostrate in precedenza scegliendo, ancora una volta, opportunamente il valore di soglia di estrazione.

5.4.1 Presentazione grafica dei risultati

Dopo aver effettuato tutte le misure dei tempi di elaborazione, abbiamo deciso di riabilitare le funzioni grafiche disabilitate in precedenza. In questo modo è stato possibile vedere graficamente la posizione dei descrittori ricavati dall'algoritmo SURF all'interno dell'immagine. Inoltre, è stata abilitata anche la funzione che implementa una semplice locazione planare dell'oggetto identificato (A.3: *locatePlanarObject()*). Questa funzione non fa altro che disegnare un rettangolo intorno all'area dell'immagine in cui è stato individuato il quadro desiderato(si veda ad esempio la

figura 5.2). Di seguito riportiamo alcune immagini relative ai risultati grafici ottenuti al variare della soglia di estrazione.

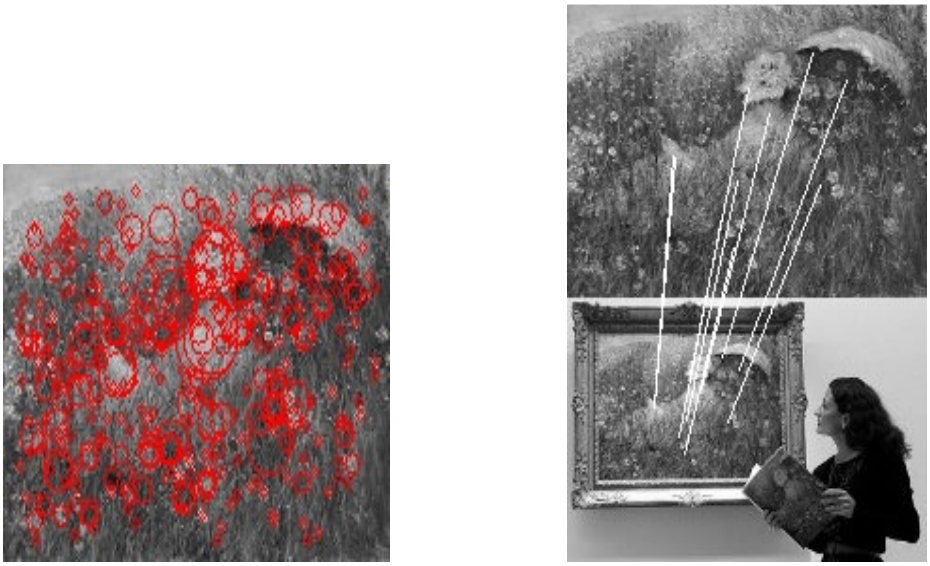


Figura 5.1: 2.0 e corrispondenze tra 2.0 e 2.1: Soglia=500.

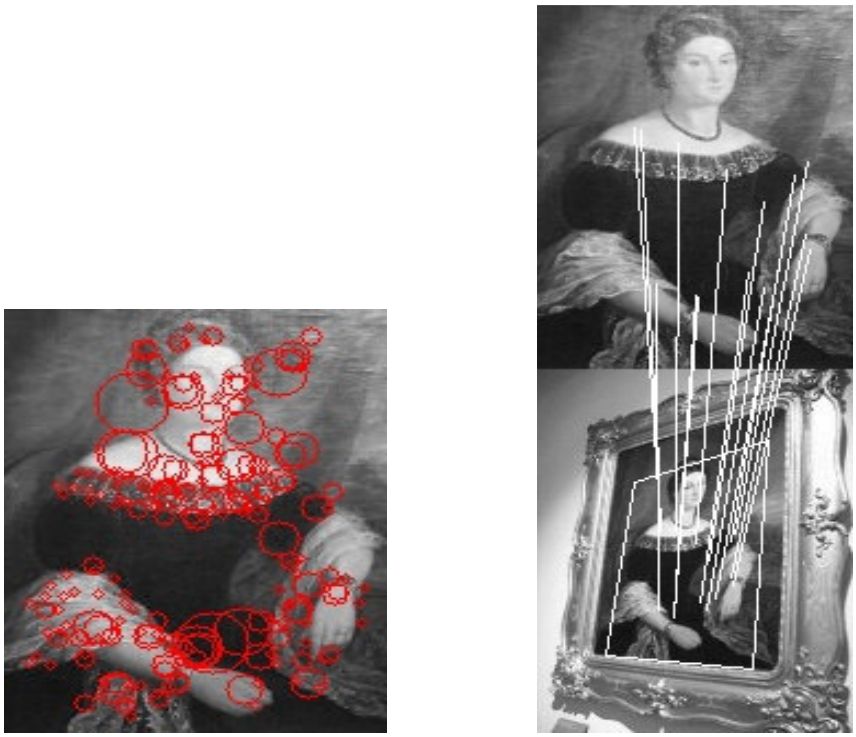


Figura 5.1: 9.0 e corrispondenze tra 9.0 e 9.1: Soglia=500

Figura 5.2: 10.0 e corrispondenze tra 10.0 e 10.1: Soglia=500

Figura 5.4: corrispondenze tra 1.0 e 1.1: Soglia=2000

Figura 5.5: corrispondenze tra 3.0 e 3.1: Soglia=2000

Figura 5.6: corrispondenze tra 8.0 e 8.1: Soglia=2000

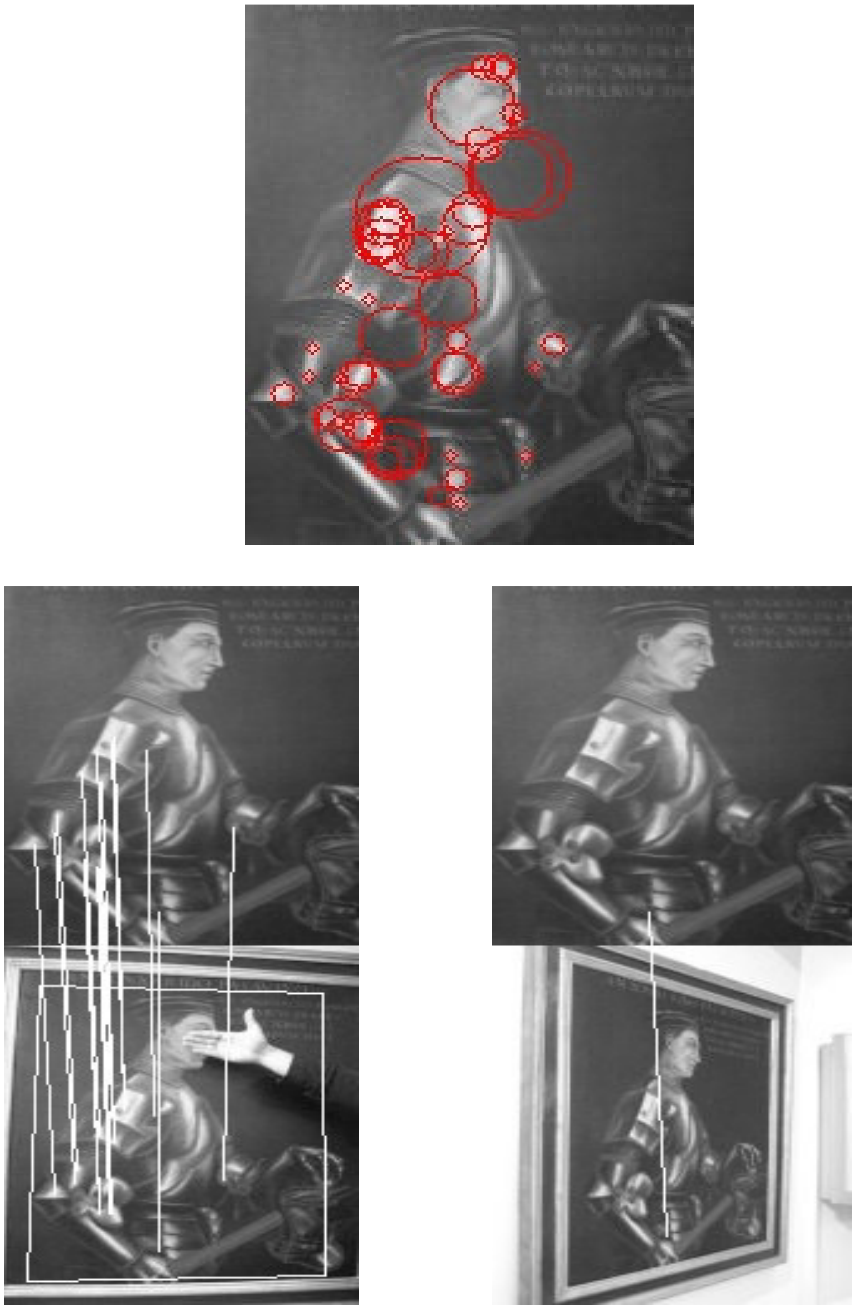


Figura 5.7:In alto 7.0 con evidenziati i descrittori estratti in aree circolari rosse, mentre in basso si mostrano le corrispondenze tra la 7.0 e 7.2 (a sinistra) e 7.1 (a destra). Soglia=2000

Dalla figura 5.7 è possibile notare che per la 8.2, nonostante vi sia la presenza di una mano che "oscura" in parte alcuni descrittori, il quadro viene comunque riconosciuto e la funzione locatePlanarObject() riesce a tracciare un rettangolo nell'area in cui viene individuato l'originale. Per quanto riguarda le corrispondenze tra 8.0 e 8.1 viene trovata la presenza di un solo descrittore in comune. Ciò ovviamente è imputabile alla rotazione e alle non ideali condizioni di luminosità e contrasto dell'immagine utente.

In questo caso la funzione citata in precedenza, non ha un numero sufficienti di descrittori per poter tracciare i quattro vertici del rettangolo.

Figura 5.8: corrispondenze tra 9.0 e 9.1: Soglia=2000

Figura 5.9: corrispondenze tra 10.0 e 10.1: Soglia=2000

Come è possibile notare dalla figura 5.9, si verifica un comportamento simile a quanto descritto per la figura 5.7: anche in questo caso la presenza di persone tra l'obiettivo della fotocamera e il quadro produce un ostacolo per la raccolta dei descrittori dell'immagine utente. In questo caso il risultato finale del matching produce soltanto due coppie di descrittori in comune, e quindi la funzione locatePlanarObject() non può funzionare correttamente.

5.5 Conclusioni e sviluppi futuri

I risultati ottenuti in questo lavoro di tesi tramite l'applicazione sviluppata, hanno permesso di mettere in risalto sia le ottime prestazioni dell'algoritmo SURF in termini di tempi di estrazione dei descrittori, sia i suoi limiti principali. Nonostante ciò, e sebbene l'applicazione non presenti un accurato metodo di ricerca e selezione dei descrittori, i tempi di elaborazione finali ottenuti e presentati nella tabella 5.10 risultano essere sempre inferiori ai 7 secondi. Infatti l'applicazione presenta delle caratteristiche non del tutto real time, in linea con gl'obiettivi prefissati e pertanto risulta essere un ottimo punto di partenza per gli sviluppi futuri.

Pertanto riportiamo alcuni accorgimenti suggeriti, non sviluppati in questa tesi, al fine di poter migliorare ancor più le prestazioni globali:

1. Ottimizzare il codice C/C++ attraverso l'utilizzo delle istruzioni *"intrinsic"* del NEON.
2. Creare un database in cui vengono inseriti opportunamente tutti i descrittori delle immagini di riferimento.
3. Migliorare la modalità di ricerca e confronto dei descrittori in modo da ridurre il tempo di matching tra le immagini.
4. Aumentare il numero di casi test per singola immagine, in modo da garantire maggiore affidabilità dell'algoritmo in condizioni differenti da quelle già testate.
5. Porting dell'applicazione su Android.

Per quanto riguarda l'ultimo punto dell'elenco riportato, si ricordi che in questo lavoro di tesi ci siamo limitati ad un'analisi composta da soltanto una decina di immagini di riferimento e confrontate con una sola immagine utente per volta, ed abbiamo ristretto il campo della nostra analisi soltanto a mostre ed eventi simili. Concludendo, visto il grande successo di Android nel mercato degli *smartphone*, sarebbe di notevole interesse effettuare un porting della nostra applicazione su tale sistema operativo.

Ringraziamenti

Desidero ringraziare fortemente il mio Professore, nonché Relatore Lorenzo Vangelista che in questi ultimi due anni di corsi qui a Padova è sempre riuscito a far nascere in me un forte interesse nello studio e nei progetti da realizzare, partendo non solo dai concetti teorici, ma anche dalla curiosità, dalla passione, dal continuo confronto col mondo extra-universitario e dalla voglia di affrontare nuove sfide durante questo percorso formativo. Scrivo percorso formativo e non semplicemente universitario, proprio perché ho imparato da lui che non bisogna fermarsi ad apprendere soltanto la teoria o la pratica in facoltà, ma bisogna vivere l'università anche da un lato sociale ed umano.

Per questo lavoro di tesi voglio ringraziare per l'infinita disponibilità anche l'Ing. Michele Bevilacqua che mi ha aiutato molto durante tutta l'attività di tesi. Allo stesso modo ringrazio anche Ivan Vecchiato.

Durante questi cinque anni e mezzo di studi e di spostamenti che mi hanno portato a vivere lontano dalla mia terra natale, sono riuscito a mantenere sempre i miei obiettivi e ad ottenere i risultati sperati soltanto grazie alla serenità e alla determinazione che la mia famiglia è riuscita a trasmettermi.

In particolare mio Padre: sei sempre in grado di tirar fuor il meglio di me anche se talvolta con mezzi poco convenzionali ma sicuramente efficaci e che ai giorni nostri, soltanto i papà migliori e più lungimiranti, sono in grado di applicare nella maniera più affettuosa verso i propri figli.

Mia Madre: sei unica, speciale e irrazionale allo stesso tempo, forse l'unica persona che con il solo sguardo e due parole riesce a capire tutto ciò che provo in quel momento o in quella determinata situazione. So bene che a volte la distanza un po' ci affligge, e sai bene che purtroppo per ora dovrò stare ancora lontano in modo tale da poter imparare quel qualcosa di più per crescere professionalmente. Già proprio quel qualcosa in più che purtroppo oggi, la nostra realtà campana non mi riesce a dare a me e a molti altri miei colleghi/amici.

Ma non preoccupatevi Papà e Mamma un giorno torneremo a stare un po' più vicini perché anche voi mi mancate, e anche se non ve lo dico o non ce lo diciamo, lo sappiamo tutti… ci vogliamo bene, io vi voglio bene più di ogni altra cosa, così come

voglio bene a Fabio e ad Alessandra… e come gl'amori e i sentimenti più forti… anche questo è un sentimento segreto e silenzioso di cui solo noi conosciamo l'esistenza!!

Colgo l'occasione non solo di ringraziare mio fratello e mia sorella, ma anche per fargli il mio più sentito augurio di poter crescere e imparare tutto ciò che più vi aggrada, che vi appassiona e che un giorno vi porterà ad essere delle splendide figure professionali, oltre alle già splendide persone che siete. Spero che la mia esperienza e il mio consiglio siano stati e possano essere sempre di buon esempio per tutti e due.

Come dicevo prima, sono davvero fortunato ad avere una famiglia eccezionale ed essa lo è infatti perché oltre ai miei genitori e fratelli vi sono i miei zii che a loro volta mi hanno dapprima cresciuto insieme ai miei genitori fin da bambino, e successivamente ognuno di essi mi ha sostenuto e consigliato nel suo piccolo.

In particolare mi riferisco allo Zio Stefano: uno zio davvero formidabile, giovane d'animo e di pensiero, sempre pronto a darmi utili consigli sul futuro e a farmi visita ogni qualvolta ne avevo bisogno, in questi ultimi due anni sei stato davvero un punto di riferimento per me e che mi ha aiutato fin da subito a stare tranquillo anche lontano da casa !

Tra le mie tante zie voglio ricordare per prima la Zia Gina: nonostante questi due anni siamo stati lontani, sapevo che il tuo pensiero era e sarà sempre con me. Poi quando ci siamo visti mi riempivi d'amore come solo tu sai fare, uno sguardo, un abbraccio e il tuo intramontabile sorriso (tanto da far invidia anche a donne più giovani zia.. solo tu ce l hai cosi !!), mi facevano stare bene.

Come non posso poi ringraziare tutti i miei altri zii Paolo, zio Salvatore, zio Antonio, zio Gennaro e zio Ernesto tutti sempre pronti a sostenermi, spronarmi e consigliarmi, una "squadra di zii" davvero unica e che mi fa sentire sempre importante per la mia famiglia… e le zie? Ma certo non vi ho dimenticate…

La mitica zia Patrizia: lo sai che ti voglio molto bene e so che anche tu me ne vuoi tanto perché ogni volta che vengo a trovarti me lo fai capire, anche tu come mamma, con pochi sguardi e poche parole!

Zia Assunta: che ultimamente mi sta "sopportando" e coccolando insieme allo zio Nino e Simona nel mio andirivieni da Monza a Varese, accompagnandomi in questa mia nuova esperienza lavorativa.

Zia Teresa, zio Salvatore, Francesco, Nunzio e Rosanna anche loro sono sempre nei miei pensieri e quando ci sentiamo mi mostrano sempre grande affetto e sostegno in tutto ciò che faccio! Soprattutto la mia zia e Nunzio con i suoi lamenti per gl'ultimi esami da sostenere ☺ e Francesco con le nostre lunghissime chiamate skype passate a parlare ….tant' pe parlà ☺ sei grande frà !!Vi voglio bene tanto!

Zia Rosaria, zio Aniello poi Ester e Rosalba con voi ultimamente ci sentiamo meno ma so che anche voi mi volete bene così come ne voglio anch'io a voi. Sia ad Ester che a Rosalba spero di vederle presto per poterci raccontare un po' la nostra storia di quest'ultimi anni in modo anche da farci qualche risata come facevamo un po' di tempo fa tra pizze sfornate dalla zia e scemenze sfornate dallo zio ☺ !!

E ancora la zia Mariapia, Rosa e Nunzia: anche a voi va il mio pensiero e il mio grazie per l'affetto che mi dimostrate ogni volta che ci vediamo con le cose più semplici, anche Rosa ormai è quasi giunta al suo primo traguardo e le auguro che sia soltanto il primo dei tanti successi, mentre a Nunzia dico di non mollare e continuare a cercare la sua strada guardando soprattutto a ciò che ti piace fare, tante volte conta solo quello…il resto vien da se !!

Restano i miei cugini Nunzio e Francesca ai quali voglio davvero molto bene e che so che riusciranno anche loro man mano crescendo ad avere un futuro pieno di soddisfazione!

In realtà restano…i più piccoli..i miei cuginetti Marco, Rossella e Stefano i quali non vedo l'ora di vederli crescere e fargli rivedere tutte le foto di quando erano più piccoli e giocavano e si divertivano insieme a me e alla mia famiglia!

E questo era solo il lato di mia madre… La famiglia Cancelliere…!

Per questo adesso passo a ringraziare anche tutti gl'altri componenti della mia famiglia e cioè quelli del lato di mio padre… La famiglia Di Vivo:

La zia Angela, zio Rocco e i miei cugini Mimmo, Carmela, Marianna e la splendida Valentina, anche voi mi avete visto crescere e ciò ci fa voler bene davvero tanto,

soprattutto alla zia Angela: voglio dirti davvero grazie perché so che anche tu mi vuoi bene e mi sostieni anche se da lontano, il sostegno più forte è col cuore e col pensiero!

Lo zio Antonio, zio Pasquale e tutto il resto della famiglia che ho conosciuto, apprezzato e a cui voglio bene.

In questi due anni la mia vita mi ha fatto anche un altro splendido regalo…ho conosciuto una donna di cui mi sono innamorato e con la quale ho avuto la fortuna di passare quest'ultimi due splendidi anni… Si sei tu Anna… amore mio! Quanti giorni e quante notti mi son lamentato di qualcosa o di qualcuno, quante volte siamo stati lontani e quante volte ancor più siamo stati vicini… Nonostante le sofferenze che il vero amore ti infligge, e nonostante le difficoltà di tutti i giorni, io e te siam riusciti a superarle e ad affrontarle come solo la forza del vero amore può farti fare. Ed è per questo che ti dico grazie, per avermi fatto cresce e diventare sicuramente più uomo di quello che ero, per avermi fatto capire quali sono le cose più belle della vita, quelle per cui vale la pena essere al mondo e che talvolta da solo non le vedi, hai bisogno di una mano. Ma più che di una mano tu mi hai concesso il tuo cuore e per questo io spero che ogni prossimo giorno insieme, io sarò in grado di meritarmi il tuo affetto e la tua complicità … per questo ti dico che tu per me…sei unica amore!

Insieme a te amore, ho conosciuto anche la tua splendida famiglia: tua madre Rosanna e tuo padre Elio e Antonio tuo fratello: a tutti e tre voglio dire grazie per tutto l'affetto e il sostegno che mi dimostrano continuamente e quotidianamente con gesti semplici ma per me importanti. Lo stesso vale anche per zio Corrado e zia Lorenzina, vi ringrazio per tutto l'affetto che mi avete dimostrato e per tutte le volte che avete guidato me ed Anna nel nostro percorso di vita !

"Last but not least", voglio ringraziare tutti i miei cari amici che mi sono stati vicini durante quest'ultimi anni e non solo: Paolo e Luca, fedeli amici di serate piene di risate e di discorsi talvolta sensati e a volte pazzoidi, fatte di scherzi improvvisi e pensate strambe messe in opera all'ultimo secondo…. grandi davvero grandi!

I miei amici di corso e di vita: Vincenzo e il fratello Antimo grandi amici tanto quanto gran belle persone con cui avrei sicuramente passare più tempo in questi ultimi due anni. Poi l'incorregibile Antonio il mio "mast", che mi supporta sempre anche da lontano, Ivano e Davide con cui abbiamo passato indimenticabili serate in giro per la Sicilia e non solo … dove io davo sempre il meglio di me nel fare la stroxxxx..oops

scemenza di turno...!!Chiudo la carrellata di amici con Vito e Alfredo amici di residenza nonché “muse” ispiratrici di gavettoni fenomenali, lo stesso vale anche per Alessandro mio ex-coinquilino di stanza…le risate che ci siam fatti a mezza notte con “Exstralibur expingibur… Excalibur Imbecille”…troppo forti! Poi il mio carissimo amico il dott. Ing. Cap. de Capi Nicola (scusa mi son fatto prendere dalla vena fantozzistica haha), al quale va un mio forte grazie per tutte le volte che mi hai sopportato durante le serate di acchiappanza (amò non ti spaventare…niente di che..giusto Nicò? Hahaha). Della residenza voglio ancora ringraziare Alessandra e Sara pazienti amiche che sono riuscite a sopportarmi anche nei momenti in cui ero davvero insopportabile sempre con simpatia e gioia. Infine ringrazio Tommaso, Denis e Isabella compagni di studi in quest’avventura indimenticabile padovana che sono stati sempre al mio fianco e mi hanno aiutato a superare le difficoltà nell’impatto con la nuova facoltà. Soprattutto ringrazio la mitica Isa che con il suo impegno e con la sua generosità e semplicità riesce sempre a darti una mano come solo un vero amico sa fare!

Come vedete mi sono divertito parecchio a scrivere i ringraziamenti, e si, mi andava proprio di farlo !

Concludendo, voglio dire che sono davvero contento di aver avuto una famiglia così bella e tanti amici su cui contare, ciò mi ha fatto rendere conto di essere stato davvero fortunato… Ed è per questo che ogni giorno ringrazio il Signore di tutto quello che mi ha donato e prego ancor più allo stesso modo, per far sì che tutti i giorni io possa onorare al meglio la mia vita e la mia famiglia, seguendo tutti gl’insegnamenti e i valori che mi hanno trasmesso i miei genitori e la mia fede Cristiana.

Marco

Appendice A

A.1 Script build_arm.sh

#build_arm.sh:

prefix1=<LTIB_DIR> /rpm/BUILD/opencv-1.1.0

CPP=arm-none-linux-gnueabi-g++

$CPP -ggdb `pkg-config --cflags opencv` -o `basename $1 .cpp` $1 -L$prefix -lgtk-x11-2.0 -lgdk_pixbuf-2.0 -lgdk-x11-2.0 -lpangocairo-1.0 -lX11 -latk-1.0 -lcairo -lgio-2.0 -lpangoft2-1.0 -lpango-1.0 -lfreetype -lfontconfig -lgobject-2.0 -lgmodule-2.0 -lglib-2.0 -lXext -lXrender -lXrandr -lxcb-xlib -lxcb -lhighgui -lcv -lcxcore -lml -lcvaux -lz -lpng12 -lpixman-1 -lexpat -lXau -lgstreamer-0.10 -lgthread-2.0 -lxml2 -lgstbase-0.10 -ljpeg -ltiff -I$prefix1/cv/include/ -I$prefix1/otherlibs/highgui/ -I$prefix1/cvaux/include/ -I$prefix1/cxcore/include/ -I$prefix1/ml/include/ -mcpu=cortex-a8 -mfloat-abi=softfp -mfpu=neon -ftree-vectorize -ftree-vectorizer-verbose=1 -O3

A.2 Contenuto dell'header file find_obj.h:

```
typedef struct CvSURFPoint{

        CvPoint2D32f pt;
        int laplacian;
        int size;
        float dir;
        float hessian;

  }CvSURFPoint;
```

```
#include <cv.h>
#include <highgui.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <vector>


using namespace std;
```

```
CvMemStorage* storage = cvCreateMemStorage(0);

static CvScalar colors[] = //Colori dal rosso al bianco...
    {
        {{0,0,255}},
        {{0,128,255}},
        {{0,255,255}},
        {{0,255,0}},
        {{255,128,0}},
        {{255,255,0}},
        {{255,0,0}},
        {{255,0,255}},
        {{255,255,255}}
    };
```

// int locatePlanarObject():

```
int
locatePlanarObject( const CvSeq* objectKeypoints, const CvSeq*
objectDescriptors,
                    const CvSeq* imageKeypoints, const CvSeq*
imageDescriptors,
                    const CvPoint src_corners[4], CvPoint
dst_corners[4]) {

double h[9];
CvMat _h = cvMat(3, 3, CV_64F, h);
vector<int> ptpairs;
vector<CvPoint2D32f> pt1, pt2;
CvMat _pt1, _pt2;
int i, n;


findPairs( objectKeypoints, objectDescriptors, imageKeypoints,
imageDescriptors, ptpairs );
n = ptpairs.size()/2;

if( n < 4 )

    return 0;

pt1.resize(n);
pt2.resize(n);
for( i = 0; i < n; i++ )
{
pt1[i] = ((CvSURFPoint*)cvGetSeqElem(objectKeypoints,ptpairs[i*2]))->pt;

pt2[i] = ((CvSURFPoint*)cvGetSeqElem(imageKeypoints,ptpairs[i*2+1]))->pt;
}

_pt1 = cvMat(1, n, CV_32FC2, &pt1[0] );
_pt2 = cvMat(1, n, CV_32FC2, &pt2[0] );
if( !cvFindHomography( &_pt1, &_pt2, &_h, CV_RANSAC, 5 ))
    return 0;

for( i = 0; i < 4; i++ )
{
    double x = src_corners[i].x, y = src_corners[i].y;
    double Z = 1./(h[6]*x + h[7]*y + h[8]);
    double X = (h[0]*x + h[1]*y + h[2])*Z;
    double Y = (h[3]*x + h[4]*y + h[5])*Z;
```

```
    dst_corners[i] = cvPoint(cvRound(X), cvRound(Y));
}

return 1;
}
```

//void findPairs():

```
void
findPairs( const CvSeq* objectKeypoints, const CvSeq* objectDescriptors,
           const CvSeq* imageKeypoints, const CvSeq* imageDescriptors,
           vector<int>& ptpairs )
   {
int i;
CvSeqReader reader, kreader;
cvStartReadSeq( objectKeypoints, &kreader );
cvStartReadSeq( objectDescriptors, &reader );
ptpairs.clear();

for( i = 0; i < objectDescriptors->total; i++ )
{
    const CvSURFPoint* kp = (const CvSURFPoint*)kreader.ptr;
    const float* descriptor = (const float*)reader.ptr;
    CV_NEXT_SEQ_ELEM( kreader.seq->elem_size, kreader );
    CV_NEXT_SEQ_ELEM( reader.seq->elem_size, reader );



int nearest_neighbor= naiveNearestNeighbor( descriptor, kp->laplacian,
imageKeypoints, imageDescriptors );


    if( nearest_neighbor >= 0 )
    {
        ptpairs.push_back(i);
        ptpairs.push_back(nearest_neighbor);
            }
}
}
```

//int naiveNearestNeighbor():

```
  int
  naiveNearestNeighbor( const float* vec, int laplacian,
                  const CvSeq* model_keypoints,
                  const CvSeq* model_descriptors )
  {
int length = (int)(model_descriptors->elem_size/sizeof(float));
int i, neighbor = -1;
double d, dist1 = 1e6, dist2 = 1e6;
CvSeqReader reader, kreader;
cvStartReadSeq( model_keypoints, &kreader, 0 );
cvStartReadSeq( model_descriptors, &reader, 0 );

for( i = 0; i < model_descriptors->total; i++ )
{
    const CvSURFPoint* kp = (const CvSURFPoint*)kreader.ptr;
    const float* mvec = (const float*)reader.ptr;
    CV_NEXT_SEQ_ELEM( kreader.seq->elem_size, kreader );
```

```
    CV_NEXT_SEQ_ELEM( reader.seq->elem_size, reader );


        if( laplacian != kp->laplacian )
        continue;

    d = compareSURFDescriptors( vec, mvec, dist2, length );

if( d < dist1 )
    {
        dist2 = dist1;
        dist1 = d;
        neighbor = i;
    }
    else if ( d < dist2 )
        dist2 = d;
}
if ( dist1 < 0.6*dist2 )
    return neighbor;
return -1;
 }
```

//double compareSURFDescriptors():

```
double

compareSURFDescriptors( const float* d1, const float* d2, double
best, int length )

{
    double total_cost = 0;
    assert( length % 4 == 0 );
    for( int i = 0; i < length; i += 4 )
    {   double t0 = d1[i] - d2[i];
        double t1 = d1[i+1] - d2[i+1];
        double t2 = d1[i+2] - d2[i+2];
        double t3 = d1[i+3] - d2[i+3];
        total_cost += t0*t0 + t1*t1 + t2*t2 + t3*t3;
        if( total_cost > best )
            break;
    }
    return total_cost;
}
```

Appendice B

B.1 Codice del file my_fobj.cpp

```
#include "find_obj.h"

int main(int argc, char** argv)
{

FILE * fx;
char* object_filename;
int i,j,object_tot=0;
double t_match,t_extract,t_complessivo=0;
vector<int> ptpairs;

CvPoint dst_corners[4];
CvSeq *objectKeypoints = 0, *objectDescriptors = 0;


printf("\nInizio fase di acquisizione delle Imm. Di Riferimento \n ");

printf("\n \n Inserire il numero Imm. Di Riferimento da acquisire: \t");

scanf("%d",&object_tot);

j=1;//indice relativo alle immagini di riferimento

i=object_tot+1 ; //indice relativo alle immagini utente

fx=fopen("descrittori.txt","wa");

//***FASE 1 : ACQUISIZIONE E CONVERSIONE IN SCALA DI GRIGI ***


while(j<=object_tot){// Ciclo per tutte le immagini di riferimento

printf("\nInizio acquisizione descrittori dell'OGGETTO %s \n ",argv[j]);

object_filename=argv[j];
//Elaborazione Immagini di riferimento “object”

IplImage* object=cvLoadImage(object_filename, CV_LOAD_IMAGE_GRAYSCALE );

CvPoint src_corners[4] = {{0,0}, {object->width,0}, {object->width,
object->height}, {0, object->height}};//utilizzati in seguito

  if( !object  )//In caso di errore..
    {
        fprintf( stderr, "Can not load %s \n",argv[1]);
              usage();
              exit(-1);
         }
//Riconverto in scala a colori

  IplImage* object_color = cvCreateImage(cvGetSize(object), 8, 3);

  cvCvtColor( object, object_color, CV_GRAY2BGR );
```

```
//Elaborazione Immagini Utente   “image”

   CvSeq *imageKeypoints = 0, *imageDescriptors = 0;


   printf("\n\n\n****Comincia l'analisi delle imagini****\n\n\n");

//Definizione della struttura params con livello di soglia scelto

printf("\n\n Inserire soglia da usare per l'immagine %s:\t ",argv[i]);

scanf("%f",&soglia);

  CvSURFParams params = cvSURFParams(soglia, 1);


//***Estrazione dei keypoints e descrittori dell'OGGETTO ******

cvExtractSURF( object, 0, &objectKeypoints, &objectDescriptors, storage,
params );

//Calcolo del tempo
t_complessivo=(double)cvGetTickCount();

  // ***************** SCORRO LE IMMAGINI UTENTE********

   while(argv[i]!= NULL){        //lavoriamo su un'immagine alla volta...

IplImage* image = cvLoadImage( argv[i], CV_LOAD_IMAGE_GRAYSCALE);

//riscrivo ogni volta l'immagine utente succesiva su image

   if( !image  )
   {
     fprintf( stderr, "Can not load %s \n",argv[i]);
     usage();
    exit(-1);
   }

printf("\n Descrittori totali per %s : %d \n\n",argv[j],
objectDescriptors->total);

   t_extract=(double)cvGetTickCount();//tempo di estrazione

cvExtractSURF(image,0,&imageKeypoints,&imageDescriptors,storage,params);

t_extract=((double)cvGetTickCount()-t_extract)/(cvGetTickFrequency()*1000.);

printf("Totale Descrittori di %s= %d ",argv[i],imageDescriptors->total);

printf("\n\n ******Tempo extraction = %g ms \n\n",t_extract);
IplImage* correspond;

if(image->width >= object->width){

correspond = cvCreateImage( cvSize(image->width, object->height+image->height),
8, 1 );

}else//se l'oggetto ha larghezza maggiore dell'immagine....
{
```

```
correspond=cvCreateImage(cvSize(object->width,object->height+image->height),8,1
);

}


cvSetImageROI( correspond,cvRect(0,0,object->width, object->height) );

//Setto la Region Of Interest

cvCopy( object, correspond );
cvSetImageROI( correspond, cvRect( 0, object->height, image->width,
correspond->height ) );
cvCopy( image, correspond );
cvResetImageROI( correspond );


/* Funzione per il tracciamento del Rettangolo (NON ABILITATA)

if( locatePlanarObject( objectKeypoints, objectDescriptors, imageKeypoints,
        imageDescriptors, src_corners, dst_corners ))
    {
        for( i = 0; i < 4; i++ )
        {
            CvPoint r1 = dst_corners[i%4];
            CvPoint r2 = dst_corners[(i+1)%4];
            cvLine( correspond, cvPoint(r1.x, r1.y+object->height ),
            cvPoint(r2.x, r2.y+object->height ), colors[8] );
        }
    }
*/


//Ricavo le coppie di descrittori in comune tra le due immagini...

  t_match = (double)cvGetTickCount();//tempo di matching start

findPairs( objectKeypoints, objectDescriptors, imageKeypoints, imageDescriptors,
ptpairs );

t_match = ((double)cvGetTickCount()-t_match)/(cvGetTickFrequency()*1000.);//stop


printf("IL NUMERO DI COPPIE IN COMUNE TRA %s ED %s E' \n \t
%d",argv[j],argv[i],ptpairs.size());

  printf("\n\n ******Tempo matching FindPairs= %gms \n\n\n",t_match);


//ADESSO SCRIVO SU FILE I PTPAIRS TROVATI TENENDO TRACCIA SIA
DELL'OGGETTO CHE DELL'IMMAGINE UTILIZZATI NELLA FINDPAIRS:

  if(ptpairs.size()/2 >1){

cvNamedWindow(argv[i], 1);//nomino le finestre
cvNamedWindow(argv[j], 1);

vector<CvPoint2D32f> pt1, pt2;
int n;

float h1,h2; //h1=Hessiano riferimento , h2=Hessiano utente

n=ptpairs.size()/2;
```

```
pt1.resize(n);
pt2.resize(n);
//Inizio scrittura su file

fprintf(fx,"\n \n************** ptpairs di Oggetto: %s Immagine: %s ,
Soglia=%.1f \n\n",argv[j],argv[i],soglia);

fprintf(fx,"\n\n**extraction time: %f ms\n**matching time: %f ms \n **
global time: %f ms \n\n",t_extract,t_match,t_glob);

fprintf(fx,"\n \n*** Numero di coppie: %d  \n\n\n",n*2);

fprintf(fx," pt_originale -- pt_user -- hessiano_pt_originale --
hessiano_pt_user \n\n\n");


    for( int m = 0; m < n; m++ )
    {
pt1[m]=((CvSURFPoint*)cvGetSeqElem(objectKeypoints,ptpairs[m*2]))->pt;

h1=((CvSURFPoint*)cvGetSeqElem(objectKeypoints,ptpairs[m*2]))->hessian;

pt2[m]=((CvSURFPoint*)cvGetSeqElem(imageKeypoints,ptpairs[m*2+1]))->pt;

h2=((CvSURFPoint*)cvGetSeqElem(imageKeypoints,ptpairs[m*2+1]))->hessian;

fprintf(fx,"( %.2f ; %.2f ) -- ( %.2f ; %.2f ) --  %.2f  --  %.2f --
\n",pt1[m].x,pt1[m].y,pt2[m].x,pt2[m].y,h1,h2);

    }

//fine scrittura su file


if(ptpairs.size()>1){  //se ptpairs ha dimensione non nulla

/* Disegno delle linee che hanno come estremi i descrittori in comune
trovati tra le due immagini (NON ABILITATO)

for( int k = 0; k < (int)ptpairs.size(); k += 2 )
                    {
CvSURFPoint* r1 = (CvSURFPoint*)cvGetSeqElem( objectKeypoints,
ptpairs[k] );
CvSURFPoint* r2 = (CvSURFPoint*)cvGetSeqElem( imageKeypoints,
ptpairs[k+1] );
cvLine( correspond, cvPointFrom32f(r1->pt),
cvPoint(cvRound(r2->pt.x), cvRound(r2->pt.y+object->height)), colors[8]
);
}
*/

              cvShowImage( argv[i], correspond );

/* Disegno delle aree circolari dei descrittori (NON ABILITATO)

for( int x = 0; x < objectKeypoints->total; x++ )
{
CvSURFPoint* r = (CvSURFPoint*)cvGetSeqElem( objectKeypoints, x );
CvPoint center;
int radius;
center.x = cvRound(r->pt.x);
```

```
center.y = cvRound(r->pt.y);
radius = cvRound(r->size*1.2/9.*2);
cvCircle( object_color, center, radius, colors[0], 1, 8, 0 );
}
*/

            cvShowImage( argv[j], object_color );

            cvWaitKey(1000);

            cvDestroyWindow(argv[i]);
            cvDestroyWindow("Object SURF");
            cvDestroyWindow(argv[j]);

        }

  }//fine if n>4

else printf(" \n\n L'immagine %s ....NON contiene l'oggetto %s \n\n
",argv[i],argv[j]); //ptpairs.size() = 0 !! Nessuna coppia trovata!

        i++;

        }//fine while delle immagini..

t_complessivo=((double)cvGetTickCount()-t_complessivo)
/(cvGetTickFrequency()*1000.)-1000;//stop

printf(" \n \n******** Tempo complessivo=%.2f  \n\n ",t_complessivo);

//esente dal tempo di estrazione dell'oggetto, ma ingloba solo i
//confronti delle varie immagini con la reference indicata

fprintf(fx," \n \n******** Tempo complessivo=%.2f ",t_complessivo);

i=object_tot+1; //reset della variabile i

j++;          // mi sposto all'immagine successiva...


}//fine while sugli oggetti



fclose(fx);

return 0;



}//fine main
```

Appendice C

Immagini Utilizzate

Nella colonna sinistra troviamo tutte le immagini di riferimento mentre in quella destra (o al centro) le immagini utente.

I quadri riportati nelle figure 7.0, 7.1, 7.2, 8.0, 8.1, 9.0, 9.1, 10.0, 10.1 e 10.2 sono stati fotografati presso alcuni musei di Padova. Abbiamo ottenuto il consenso per la pubblicazione di queste foto tramite un'apposita autorizzazione la quale ci permette di poter pubblicare queste foto per il congresso "Globecomm 2011" che si terrà ad Houston il 5 dicembre 2011.

Le foto 1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 4.0, 4.1, 5.0, 5.1, 5.2, 6.0, 6.1 e 6.2 sono state prelevate da vari siti internet di pubblico accesso in cui non risulta nessun diritto di copyright applicato su di esse.

Figura 4.0 e 4.1: "Campo di Papaveri" V. Monet


Figura 5.0 e 5.1: “Soil Levant” V. Monet.

Figura 5.2 “Soil Levant” V. Monet

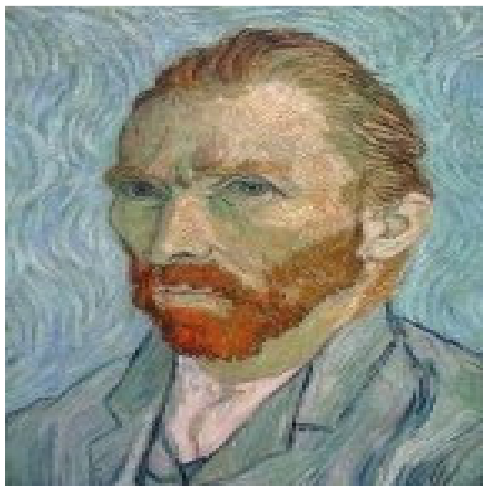



Figure 6.0 e 6.1: “Ritratto di Van Gogh”


Figure 7.0 e 7.1: “Antonio da Rio”

Figura 7.2: “Antonio da Rio”.


Figure 8.0 e 8.1: “Il banchetto di Erode”.


Figure 9.0 e 9.1: ” Ritratto di Elena Martinelli” A. Astolifi.



Figure 10.0,10.1 e 10.2: “Ritratto di Gentildonna con figlioletta”

Bibliografia

[1] H. Sugano and R. Miyamoto:
"Opencv implementation optimized for a cell broadband engine processor"
in Digital Signal Processing Workshop and 5th IEEE Signal Processing Education Workshop, 2009. DSP/SPE 2009. IEEE 13th, 2009, pp. 182 –187.

[2] O. Sidla, N. Braendle, W. Benesova, M. Rosner, and Y. Lypetskyy,
"Towards complex visual surveillance algorithms on smart cameras,"
in Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on, 2009, pp. 847 –853.

[3]H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool,
"Speededup robust features (surf)"
Computer Vision and Image Understanding, vol. 110, no. 3, pp. 346 – 359, 2008, similarity Matching in Computer Vision and Multimedia.
[Online] Available at: http://www.sciencedirect.com/science/article/B6WCX-4RC2S4T-2/2/c2c03b6165996e30312e5b7c7b681155

[4]"Object recognition from local scale-invariant features" Lowe, D. G. .
Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference.

[5]"CenSurE: Center Surround Extremas for Realtime Feature Detection and Matching"by .Motilal Agrawal, Kurt Konolige, and Morten Rufus Blas.

[6] "Detailed Analysis and Evaluation of Keypoint Extraction Methods".
Xianliang Wu, Zongying Shi, Yisheng Zhong. 20IO International Conference on Computer Application and System Modeling (ICCASM 2010).

[7] Viola, P. & Jones, M. (2001). "Rapid object detection using a boosted cascade of simple Features". In IEEE Computer Vision and Pattern Recognition (pp. I:511–518).

[8] "Notes on the OpenSURF Library", Christopher Evans(2009).

[9]Sito sviluppatori di U-Boot: http://www.denx.de/wiki/U-Boot

[10] "Learning OpenCV, Computer Vision with the OpenCV Library"

By Gary Bradski, Adrian Kaehler Publisher: O'Reilly Media .Released: September 2008

[11] Per informazioni dettagliate consultare il manuale in rete : http://opencv.jp/opencv-1.1.0/document/opencvref_cv.htm

[12] "Distinctive Image Features from Scale-Invariant Keypoints"

by David G. Lowe. Computer Science Department University of British Columbia