Università degli studi di Padova

Facoltà di Ingegneria

Tesi di Laurea in

Ingegneria Biomedica

# Implementazione in hardware di un sistema di pattern recognition

Relatore

Prof. Maria Pia Saccomani

Correlatore

Nicola Codogno

Candidato

Antonio Maria Bovo

Anno Accademico 2012/2013

Possiamo perdonare un bambino quando ha paura del buio. La vera tragedia della vita è quando un uomo ha paura della luce

Platone

# Indice

# Sommario

Il progetto di cui fa parte questo lavoro, è quello di sperimentare se è possibile trasferire in un sistema embedded alcune delle operazioni che, al momento, il topografo corneale Magellan Mapper$^{TM}$ svolge con l'ausilio di un PC connesso tramite USB.

Tale obbiettivo è raggiunto se il tempo di acquisizione e di elaborazione dei frame delle immagini catturate, utilizzando il medesimo algoritmo che già gira su x86 nel prodotto Magellan Mapper$^{TM}$, è in linea con il target prestabilito di 25 frame al secondo.

Scopo di questa tesi è il porting di tale algoritmo sulla piattaforma Beagle Board e il setup della toolchain necessaria a svolgere questa attività.

# Introduzione

Il presente lavoro di tesi è relativo al tirocinio da me svolto presso Nidek Technologies S.R.L. di Albignasego.

L'obiettivo principale e la missione di Nidek Technologies all'interno del Gruppo NIDEK consiste nella ricerca all'avanguardia e nello sviluppo di attrezzature innovative e tecnologicamente avanzate di strumenti per l'industria dell'Eye care. In questa sua missione Nidek Technologies collabora attivamente con le istituzioni accademiche e i centri di ricerca internazionale per lo sviluppo e la commercializzazione del know-how e della tecnologia.

Il progetto di cui fa parte questo lavoro, ha lo scopo di sperimentare se è possibile trasferire in un sistema embedded alcune delle operazioni che, al momento, il topografo corneale Magellan Mapper$^{TM}$ svolge con l'ausilio di un PC connesso tramite USB. La realizzazione di tale progetto aumenterebbe così la portabilità e la versatilità di utilizzo del macchinario oftalmico, diminuendone anche i costi.

Le principale componenti utilizzate nel lavoro sono state la Beagle Board, che si può brevemente descrivere come un vero e proprio mini computer single-board sviluppato da Texas Instruments dotato di tutte le funzioni necessarie per far funzionare in condizioni ottimali un sistema operativo, la piattaforma di sviluppo Eclipse e le librerie OpenCV per la computer vision.

Per valutare se la piattaforma costituita dalla parti precedentemente citate è adeguata allo scopo, si considera il tempo di acquisizione e di elaborazione dei frame delle immagini catturate utilizzando il medesimo algoritmo che già gira su x86 nel prodotto Magellan Mapper$^{TM}$: il target prestabilito è di 25 frame al

secondo. Scopo di questa tesi è il porting di tale algoritmo sulla piattaforma Beagle Board e il setup della toolchain necessaria a svolgere questa attività.

L'algoritmo viene scritto con il linguaggio di programmazione C++ ed oltre ad appoggiarsi alle librerie OpenCV per l'acquisizione e l'elaborazione delle immagini saranno utilizzate le librerie necessarie per sfruttare due componenti hardware molto interessanti presenti nella Beagle Board: gli acceleratori NEON e DSP.

Grazie al loro utilizzo si conta infatti di riuscire a ridurre in modo molto marcato operazioni di trasformazione, analisi e elaborazione dell'immagine, come ad esempio la trasformata di Fourier, che richiedono grande lavoro da parte del calcolatore per la complessità del calcolo e l'elevata mole di dati.

Di grande rilevanza è che il tutto viene svolto con strumenti Open Source, senza licenza, e quindi senza costi per l'azienda, che in fase di valutazione della tecnologia non è costretta ad investimenti onerosi.

Il fatto che non siano a pagamento non deve indurre il lettore a pensare che i mezzi utilizzati siano di basso livello anzi, la licenza libera oltre ad essere un vantaggio dal punto di vista economico lo è anche poiché vede orbitare intorno a se una comunità molto ampia che consente un continuo sviluppo e miglioramento del software.

La successione e il contenuto dei capitoli sono così strutturati:

- Il capitolo 1 fornisce una descrizione dell'azienda in cui si è svolto il tirocinio, quale ruolo occupa la sede di Padova all'interno della multinazionale e una panoramica sui prodotti sviluppati e commercializzati dalla Nidek.

- Il capitolo 2 ha lo scopo di fornire quelle che sono le conoscenze di base necessarie per comprendere come si è operato nello sviluppo del progetto, spiegando il funzionamento dei mezzi e delle proprietà utilizzate.

- Il capitolo 3 fornisce una guida per la preparazione della postazione, Eclipse ed OpenCV, e vengono descritte le caratteristiche della BeagleBoard.

- Nel capitolo 4 sono illustrati i procedimenti e i test eseguiti per valutare se i mezzi utilizzati sono adeguati allo scopo.
- Nel capitolo delle Conclusioni vengono discussi i risultati ottenuti.

# Capitolo 1

# L'azienda

Nidek è una multinazionale con sedi in Giappone e negli Stati Uniti. Anche in Italia è presente un reparto di ricerca e sviluppo, Nidek Technologies, dove è stato svolto il presente lavoro di tesi.

Nidek Technologies si occupa di apparecchiature diagnostiche per oculisti e sistemi di chirurgia oftalmica. Inoltre si sta orientando con buon successo nella progettazione di sistemi telematici per l'utilizzo dei propri strumenti.

Ubicata nel comune di Albignasego, cintura urbana del comune di Padova, è una società specializzata nella ricerca, sviluppo e produzione di strumenti diagnostici ad alto contenuto tecnologico e di sistemi informatici per l'oftalmologia. Incorporata nel gruppo giapponese Nidek nel 2001, Nidek Technologies si dedica alla progettazione e allo sviluppo di strumentazioni innovative e tecnologicamente avanzate, e di sistemi software per la diagnostica in oftalmologia. Nidek Techno-

Figura 1.1: L'azienda

logies svolge un importante ruolo all'interno del gruppo poiché presso la propria sede si svolge la ricerca e sviluppo di nuovi prodotti. Grazie a tale lavoro e allo stretto contatto con il Dipartimento di Ingegneria dell'Informazione dell'Università di Padova, con la Clinica Oculistica dell'università di Padova e con alcune tra le più prestigiose università statunitensi, l'azienda si colloca attualmente nel suo campo tra le più importanti a livello mondiale.

## 1.1 L'innovazione

Attualmente la ricerca è orientata allo sviluppo di un'ampia gamma di tecnologie afferenti all'oculistica e all'ottica. L'obbiettivo che Nidek intende raggiungere è di permettere lo screening di patologie oculari per un elevato numero di persone semplificando l'utilizzo della strumentazione, al fine di eliminare ogni barriera burocratica e impedimento psicologico o tecnologico.

Il tutto per permettere di rilevare, con notevole anticipo, l'insorgenza di patologie che minacciano la funzione visiva e di conseguenza rendere efficace e tempestiva la prevenzione mediante le terapie.

Per raggiungere tale traguardo Nidek ha puntato fortemente sulla ricerca ed in particolare sulla evoluzione dei propri prodotti, filosofia che si può riassumere in tre concetti chiave:

- facilità di esecuzione;
- affidabilità dei dati acquisiti;
- contenimento dei costi.

## 1.2 I prodotti

### 1.2.1 Navis (Nidek Advanced Vision Information System)

Sistema operativo completo per la gestione integrata in rete del database dei pazienti e di tutti gli esami strumentali.


Consente la gestione in tempo reale delle informazioni diagnostiche provenienti dai diversi strumenti utilizzati in oftalmologia. I dati e le informazioni acquisiti vengono automaticamente memorizzati all'interno di un database elettronico dedicato al paziente, prevenendo la perdita accidentale di informazioni tipica delle cartelle paziente cartacee e gli errori di trascrizione, riducendo il tempo dell'esame ed aumentando di conseguenza il flusso di pazienti.

### 1.2.2 Orion Autoretinal Imaging

Orion è un retinografo in grado di esaminare il fondo

dell'occhio e scoprire eventuali segni di patologie retinica. Questo strumento non necessita di un operatore dedicato, identifica la presenza di un paziente ed acquisisce immagini dei differenti campi retinici compensando il difetto ottico del paziente.

### 1.2.3 Confoscan 4

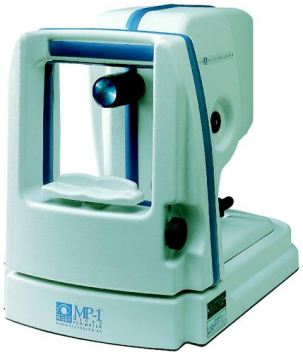
Microscopio corneale confocale.

È uno strumento diagnostico che integra un microscopio confocale, un microscopio non a contatto con l'endotelio e un preciso pachimetro.

### 1.2.4 Micro Perimeter

Il Microperimetro MP-1 integra in un unico strumento i dati soggettivi della perimetria computerizzata ed i dati

oggettivi delle immagini retiniche ottenendo misurazioni precise, ripetibili e completamente automatiche delle funzioni retinica e maculare.



### 1.2.5 Magellan Mapper

Topografo corneale ad altissima risoluzione (oltre 21.000 punti effettivi) semplicissimo da usare e dotato di un

innovativo sistema di intelligenza artificiale per il riconoscimento di otto diversi tipi di anomalie corneali. Nel seguito vedremo più in dettaglio le caratteristiche tecniche di questa apparecchiatura.

## 1.3 Nidek Magellan Mapper$^{TM}$

## MM1

1. Appoggio Frontale
2. Proiettore di anelli
3. Testa ottica
4. Joystick
5. Regolazione mentoniera


Figura 1.2: Nidek Magellan Mapper

Configurazione del sistema:

- Illuminazione allo stato solido
- Telecamera CCD
- Sistema Operativo per l'Oftalmologia NAVIS
- Classe 1B

- Alimentazione: 100/240 VAC 50/60 Hz
- Dimensione: (LxAxP) 36x48x49 cm / 14.2" x18.9" x19.3"
- Peso: 12 Kg / 26.5 Lbs
- Temperatura utilizzo: 15 / 30 °C - 59 / 86 F
- Umidità relativa 30 ÷ 75 %

Nidek Magellan Mapper$^{TM}$ è un topografo utilizzato nell'industria oftalmologica per rilevare e analizzare i difetti della cornea.

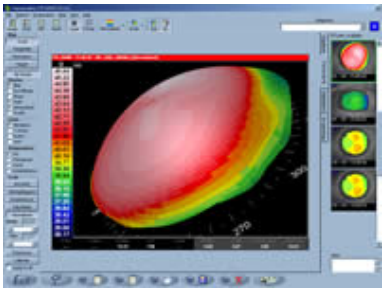
Questo potente macchinario usa il Corneal Navigator$^{TM}$ ideato da Stephen Klyce e Michael Smolek, estremamente versatile, facile da utilizzare e veloce nell'analisi è in grado di diagnosticare ben otto differenti tipi di difetti della cornea grazie alla sua alta definizione.

La sua versatilità è dovuta alla possibilità di utilizzare il MagellanMapper$^{TM}$ semplicemente collegandolo via USB ad un qualsiasi notebook o desktop computer operante con Wndows 2000 o XP e che utilizzi la nuova applicazione software di topografia corneale Nidek.

La facilità dell'utilizzo e la velocità nell'analisi si devono invece al disegno ergonomico, la chiara proiezione degli anelli ed il semplice sistema di allineamento che riducono la durata dell'esame e richiedono una minima cooperazione da parte del paziente. Entrando più nel dettaglio, il Magellan Mapper$^{TM}$ a differenza dei topografi convenzionali sfrutta una proiezione degli anelli "sharp angle" tramite un cono di ridotte dimensioni.

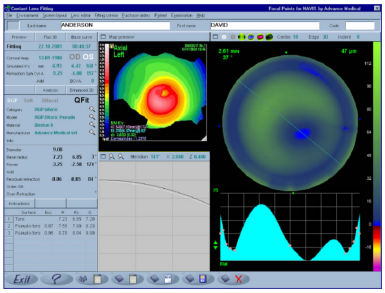
L'alta densità di anelli proiettati sulla cornea aumenta la qualità delle misure ed evita interferenze ed artefatti indesiderati. Si parla di un cono costituito da 30 anelli la cui proiezione corneale è analizzata dal "ring tracking software" ottenendo un totale di 60 bordi anulari e 26.600 punti corneali rilevati tramite telecamera ad alta risoluzione, garantendo un'assoluta precisione.

Le caratteristiche dell'innovativo software di topografia corneale si possono riassumere con le seguenti caratteristiche:



- Interfaccia grafica semplice e veloce.
- Visualizzazione singola, multipla e tridimensionale.
- Scala assoluta, standard, normalizzate e personalizzabile.
- Calcolo e visualizzazione della mappa topografica estremamente veloce.

Un'altra caratteristica di grande pregio è la presenza di un software di applicazione lenti corneali:



- Mappe

  topografiche fluoresceiniche, valutazione del film lacrimale, analisi del profilo della lente a contatto e visualizzazione 3D dell'applicazione della lente corneale.

- Database di lenti corneali rigide e morbide, di produzione, personalizzate ed adattive.
- Elevata flessibilità nella definizione delle lenti corneali.
- Criteri e procedure di applicazione completamente personalizzabili.
- Analisi di Fourier per componenti sia di alto che di basso ordine.

Il Sistema Operativo per Oculistica Navis "Nidek Advanced Vision Information System", incluso in ogni Magellan Mapper$^{TM}$ , consente inoltre una completa gestione dei dati paziente, il trasferimento automatico dei dati dalla strumentazione oftalmica e la necessaria connettività in telemedicina.

# Capitolo 2

# Concetti di base

## 2.1 Organi di senso

Sono composti da strutture più o meno complesse specializzate nella ricezione di stimoli provenienti dall'esterno o dall'interno per poi trasformarli in impulsi nervosi e infine trasmetterli al sistema nervoso centrale. Si possono distinguere sette diversi organi di senso:

1. vista;
2. udito;
3. tatto;
4. olfatto;
5. gusto;
6. propriocezione[1];
7. apparato vestibolare o equilibrio.

[1] Capacità di percepire e riconoscere la posizione del proprio corpo nello spazio e lo stato di contrazione dei propri muscoli, anche senza il supporto della vista.

Gli organi di senso si basano su i recettori sensoriali (recettori della retina, recettori delle terminazioni tattili... ) che sono dei veri e proprio trasduttori, ovvero trasformano una variazione di energia in potenziale d'azione (es. cellule della retina trasformano l'energia dei fotoni in potenziale d'azione o l'equilibrio traduce variazioni di energia cinetica in potenziale d'azione).

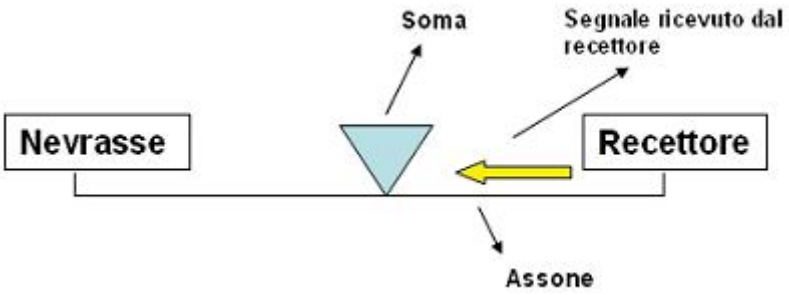
Dal tipo di stimolo si possono distinguere:

- Chemiorecettori: ricevono stimoli chimici come ad esempio il gusto e l'olfatto (ma anche concentrazione di CO2 e pH nel sangue).
- Meccanorecettori: ricevono variazioni di pressione. Comprendono l'udito e il tatto.
- Termorecettori: ricevono stimoli termici.
- Fotorecettori: ricevono stimoli luminosi come ad esempio la vista.

Si possono distinguere due tipi di sensibilità:

1. sensibilità somatica: coinvolge tutti i recettori che raccolgono informazioni dall'esterno. Viene ulteriormente suddivisa in:
    - sensibilità somestesica: sensibilità in generale gusto, tatto, vista, olfatto;
    - sensibilità propriocettiva: raccolta di segnali dai muscoli e dalle articolazioni.
2. Sensibilità viscerale: coinvolge tutti i recettori che raccolgono informazioni dall'interno.

**Per quanto riguarda la sensibilità somestesica...** I segnali accolti dai recettori sensitivi vengono veicolati al SNC tramite i neuroni sensitivi.




I neuroni sensitivi si trovano in particolari strutture vicine al nostro SNC chiamate gangli sensitivi. Sono bipolari, cioè presentano un assone e un solo dendrite. I due prolungamenti si dipartono dagli antipodi del soma: uno dal recettore e l'altro dalla nevrasse (SNC). Quindi siamo in presenza di un assone che si è modificato sia per ricevere segnali sensoriali sia per veicolarli poi al SNC.

I neuroni sensitivi prendono contatto con dei secondi neuroni presenti nel SNC organizzati in nuclei sensitivi; questi mandano le loro proiezioni ad una stazione intermedia: il talamo dal quale i segnali vengono poi inoltrati alla corteccia sensitiva.

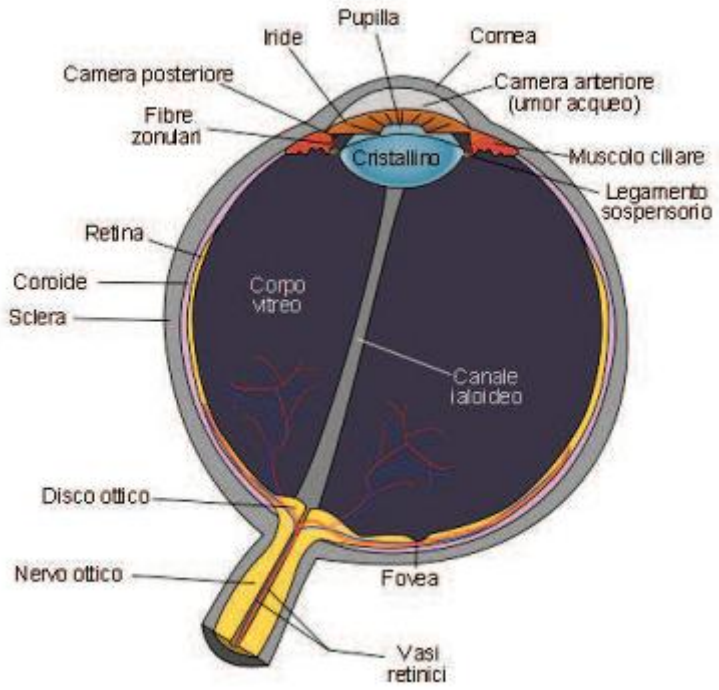
**Per quanto riguarda la sensibilità propriocettiva...** Anche la sensibilità propriocettiva funziona a grandi linee come quella somestesica. Un'importante differenza tra le due è che le proiezioni (relative alla sensibilità propriocettiva) vengono inoltrate oltre che al nevrasse anche al cervelletto.

### 2.1.1 Vista

La vista è il senso mediante il quale è possibile percepire gli stimoli luminosi e quindi la forma, il colore, la dimensione e la posizione degli oggetti. Tale percezione avviene per mezzo dell'occhio.

### 2.1.2 Occhio

L'occhio ha una forma ovoidale pesa 6-8 grammi ed è formato da tre tuniche fibrose che, procedendo dall'esterno verso l'interno, sono: la sclera, la coroide , la retina. Si chiude sul davanti con una membrana trasparente, la cornea, che




si incastra come un vetro d'orologio alla sclera e che ne è la sua prosecuzione in avanti.

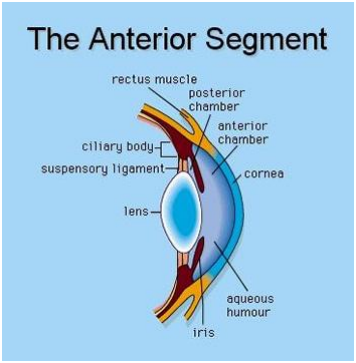
Nel suo interno sono contenuti due liquidi che oltre a provvedere al nutrimento delle varie strutture, danno consistenza e volume al globo oculare.

Il primo liquido che troviamo è l'umore acqueo che occupa lo spazio della camera anteriore, cioè lo spazio compreso tra superficie posteriore della cornea e superficie anteriore del cristallino.

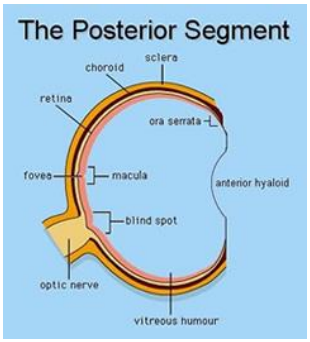
Il secondo liquido, di consistenza gelatinosa, è l'umore vitreo che occupa la cavità vitrea, spazio compreso tra la superficie posteriore del cristallino e la retina.

### Il segmento anteriore e posteriore

Il cosiddetto segmento anteriore del bulbo oculare è la parte visibile dell'occhio e comprende la cornea, la camera anteriore, l'iride, il forame centrale pupillare, il cristallino con le strutture che lo sostengono, cioè l'apparato zonulare, il corpo ciliare che prosegue anteriormente con l'iride e che si continua con la coroide posteriormente.




In oftalmologia la suddivisione del bulbo nei due segmenti anteriore e posteriore è molto importante sia per la chirurgia che per la diagnostica. Anche i chirurghi sono di solito dedicati o al segmento anteriore o a quello posteriore e raramente sono ugualmente eccellenti in entrambi i settori.




Il cosiddetto segmento posteriore del bulbo si estende in quell'area anatomica che va dalla superficie posteriore del cristallino alla retina con il corpo vitreo che riempie la cavità omonima.

La diagnostica di tale segmento e ancor più la chirurgia si avvalgono di strumenti e di tecniche tra le più sofisticate ed elettive, idonee per osservare ed intervenire su strutture molto delicate ed estremamente vulnerabili. La chirurgia della retina e del vitreo sono sempre molto evolute ed in continuo cambiamento.

La parte anteriore è delimitata verso l'esterno dalla cornea (I lente), caratteristicamente trasparente e costituita principalmente da cellule connettivali. Dietro la cornea troviamo una camera contenente l'umor acqueo e dopo di esso è presente una seconda lente (II lente): il cristallino. Davanti al cristallino si trova l'iride, mentre dietro è presente l'umor vitreo che costituisce il più vasto mezzo diottrico del bulbo. Le membrane che avvolgono il bulbo sono (dalla più interna alla più esterna):

1. La retina o superficie altamente differenziata. Ha la forma di una rete neuronale costituita da più strati di neuroni ed è ricca di fotorecettori. Tra

le cellule che la compongono si devono ricordare: i coni, responsabili della visione a colori ma sensibili solo a luci piuttosto intense; i bastoncelli, che sono particolarmente sensibili a basse intensità di luce ma non ai colori.

2. La coroide o uvea, superficie impiegata essenzialmente all'irrigazione vasale del bulbo.

3. La sclera che compare anche all'esame esterno dell'occhio, di colore biancastro costante su tutta la superficie dell'occhio, di natura fibrosa.

### La cornea

La cornea è la prima lente naturale che la luce incontra, è determinante che sia trasparente e di forma più o meno sferica. La diversa curvatura della cornea nei diversi archi per i 360° della sua circonferenza determina astigmatismo, cioè un difetto visivo che non permette una nitida visione. Il suo potere medio è di circa 43 Diottrie. La cornea è una membrana trasparente priva di vasi ma ricchissima di fibre nervose. Essa è bagnata continuamente dal film lacrimale che aderisce alla sua superficie anteriore ed è nutrita dall'umore acqueo cui trae contatti nella sua superficie posteriore.

L'interfaccia film lacrimale-superficie corneale costituisce la lente convergente più potente dell'occhio umano. La stabilità del film lacrimale e la trasparenza della cornea sono essenziali per la visione.

La cornea ha uno spessore di circa 550 micron ed è composta dall'esterno all'interno da 5 strati:

- epitelio Pavimentoso (stratificato);
- membrana di Bowman;
- lo Stroma;
- la membrana di Descemet;
- l'Endotelio.

### Iride e pupilla

L'iride è la parte più anteriore dell'uvea che dà il colore ai nostri occhi e circonda un piccolo foro centrale di ampiezza variabile da 2 a 8 mm : la pupilla.


L'iride è composta da uno stroma, un foglietto pigmentato posteriore, da vasi e da 2 muscoli: il muscolo radiale (dilatatore) ed il muscolo sfintere (costrittore) dell'iride. Può essere chiara (dal blu al verde) o bruna (dal marrone al nero) ma in realtà la sua colorazione dipende sia dalla quantità di pigmento che da fenomeni ottici di riflessione e di diffrazione della luce nello stroma irideo.

Nelle iridi chiare poco pigmentate la luce passa fino agli strati profondi dove viene riflessa assumendo un colore chiaro. Al contrario nelle iridi brune, ricche di pigmento, la luce non penetra fino agli strati profondi e non viene riflessa né diffratta.

L'iride circonda la pupilla che si allarga o si restringe a seconda della quantità di luce che la raggiunge, agendo così come il diaframma di una macchina fotografica che regola la quantità di luce che deve raggiungere la retina.

Dietro l'iride c'è il cristallino o lente che delimita il segmento anteriore e che costituisce, quando diventa opaca, la struttura che è oggetto dell'intervento più frequente in oculistica : l'intervento di cataratta.

### Cristallino

Il cristallino è una lente biconvessa che viene tenuta in sede grazie a dei muscoli tensori del cristallino e legamenti: legamenti sospensori del cristallino.

È una struttura trasparente collocata posteriormente all'iride e anteriormente al corpo vitreo e alla fossa jaloidea, bagnata dall'umore acqueo della camera posteriore e tenuto in sede dalle fibre zonulari che si attaccano all'apice dei corpi ciliari. È una struttura priva di vascolarizzazione e innervazione e questo le permette di rimanere otticamente perfetta. Il cristallino è formato da tre strati, dal più superficiale al più profondo sono: la capsula, l'epitelio e le fibre del cristallino.

A seconda della convessità del cristallino il potere di convergenza cambia e cambia quindi anche il grado di focalizzazione (o messa a fuoco). Questi cambiamenti di convessità vengono fatti in modo riflesso e prendono il nome di riflessi di accomodazione. Mentre in una macchina fotografica il fotografo mette a fuoco l'immagine variando la distanza focale fra lente e pellicola, nell'occhio la distanza tra il cristallino e la retina rimane fissa.

L'occhio mette a fuoco a distanze variabili con una strategia diversa: il cristallino ha la capacità di modificare continuamente la sua forma e di variare la sua curvatura in modo da aumentare o diminuire il suo potere di convergenza.

Quando l'occhio guarda un oggetto in lontananza il cristallino si appiattisce e diminuisce la sua curvatura. Al contrario quando guarda un oggetto vicino diventa più convesso ed aumenta la sua curvatura.

L'invecchiamento fa perdere sia al cristallino che al corpo ciliare il potere di accomodazione cosicché si diventa presbiti e non si è capaci di leggere a 30 cm. In questo caso si ricorre alla correzione con lenti per vicino, bifocali o multifocali.

### Retina

La retina, che rappresenta l'equivalente della pellicola fotografica, riveste la superficie interna del globo oculare.

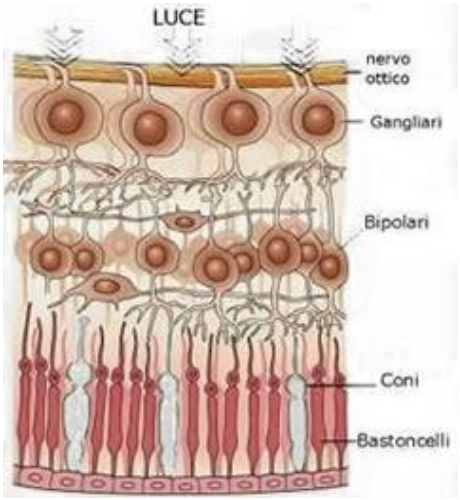
Essa appare come una sottile membrana trasparente suddivisa in due aree:

- un'area centrale chiamata macula che contiene la fovea centrale, ricca di coni;

- un'area media e periferica, dove prevalgono le cellule dei bastoncelli, che serve a mediare la visione crepuscolare e notturna.

Dopo aver attraversato la cornea, la camera anteriore, la pupilla, il cristallino ed il vitreo, i raggi luminosi vengono fatti convergere sulla retina ed in particolare in quella piccolissima area chiamata fovea centrale: una struttura altamente specializzata che presiede, in condizioni di alta luminosità, alla massima acuità visiva per lontano e per vicino, alla percezione dei colori ed alla sensibilità al contrasto.

Nella retina avvengono i meccanismi più complessi della visione. La luce passa l'intero spessore della retina e colpisce immediatamente i fotorecettori, essenzialmente di due tipi: coni e bastoncelli.




I coni sono cellule fotorecettori presenti nella retina oculare che funzionano maggiormente in condizione di luce relativamente intensa. Sono meno sensibili alla luce dei bastoncelli ma permettono la percezione dei colori. Esistono tre tipi diversi di coni a seconda dei pigmenti che contengono. I pigmenti sono codificati da geni che si trovano sul cromosoma X, ad esempio il daltonismo è dovuto ad un deficit del pigmento rosso. I coni sono anche in grado di percepire dettagli con maggiore risoluzione e cambiamenti di immagine più rapidi perché i loro tempi di risposta agli stimoli sono più veloci di quelli dei bastoncelli.

Sono localizzati principalmente nella fovea (parte centrale della retina), e permettono la cosiddetta visione fotopica.

I bastoncelli, di forma più allungata, sono molto più numerosi dei coni e risiedono maggiormente nella parte periferica della retina.

Essi sono più specializzati a raccogliere stimoli luminosi di bassa intensità e

quindi sfruttati nella visione in condizioni di scarsa luminosità. Più bastoncelli traggono contatti, cioè scaricano il proprio impulso, con una sola cellula neuronale sottostante venendo a creare un collegamento di molte cellule con una bipolare e ganglionare. Ne consegue che lo stimolo generato non è così strutturato e preciso come quello dei coni. Infatti l'acutezza visiva cala notevolmente in condizioni di scarsa luminosità.

È un recettore visivo fotosensibile che ha una struttura particolare formata da una porzione esterna, una interna e un terminale sinaptico dal quale vengono rilasciati i neurotrasmettitori.

Il bastoncello, essendo un trasduttore, converte l'energia luminosa in variazione di potenziale elettrico delle membrana cellulare. Questo avviene sfruttando un particolare pigmento presente sulla membrana di bastoncello chiamato rodopsina. La rodopsina è una molecola che colpita dalla luce isomerizza, subisce cioè un cambiamento conformazionale, attivando a sua volta la trasducina (proteina) che lega GTP (stretto parente dell'ATP) attivando la fosfodiesterasi , la quale trasforma il GNP ciclico in GNP linearizzato.

I bastoncelli hanno una funzione complementare ai coni, infatti operano in condizioni di scarsa visibilità e forniscono la cosiddetta visione scotopica.

## 2.2 Trasformata di Fourier

La trasformata di Fourier è una delle più utili formule che siano mai state scritte, alla base del progresso tecnologico degli ultimi 200 anni!

Pochi ne sono consapevoli, ma questa formula (e le sue varianti), sono usate di continuo in decine di campi a prima vista inconciliabili tra loro, come ad esempio l'elettronica, la musica, la medicina, la fisica, la chimica...

Usate il cellulare? Guardate la TV? Ascoltate la musica? Allora questa formula (in realtà è più corretto parlare di una coppia di formule) merita di essere compresa. Iniziamo a vedere di che tipo di equazioni si sta parlando:

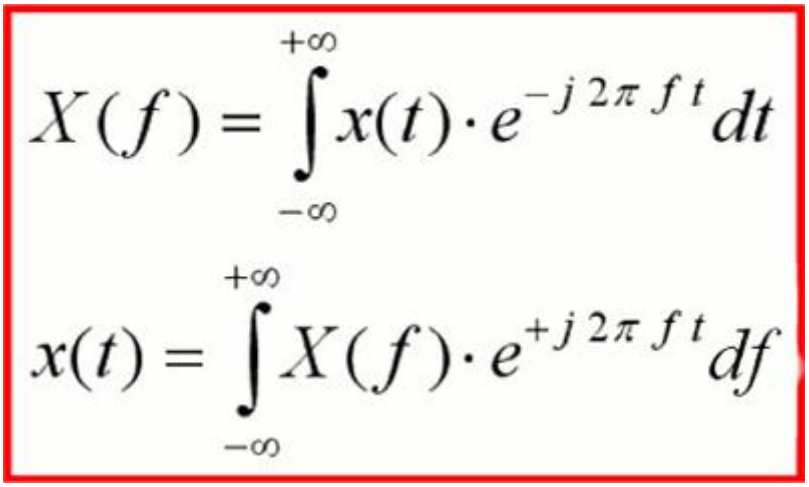



Figura 2.1: Trasformata e antitrasformata di Fourier

### 2.2.1 Un po' di storia

Queste due equazioni sono nate dalla brillante mente di Jean Baptiste Joseph Fourier quasi 200 anni fa, nel 1822.

Fourier, rimasto orfano a 10 anni, ebbe una vita incredibilmente avventurosa e impegnata. Dapprima poliziotto segreto durante la rivoluzione francese, poi prigioniero politico, fu anche professore universitario, governatore d'Egitto, prefetto di Francia e amico di Napoleone.

Il suo lavoro burocratico come prefetto in Egitto lo costrinse per anni a vivere lontano dal mondo scientifico parigino, costantemente impegnato a risolvere problemi amministrativi. Fu solo con la caduta di Napoleone che cominciò la sua vita da scienziato a tempo pieno.

Le sue lotte scientifiche con Lagrange, Laplace, Cauchy e Poisson lo aiutarono a sviluppare nel dettaglio le teorie per cui è conosciuto ancora oggi.

### 2.2.2 Di cosa stiamo parlando quindi?

In parole povere la trasformata di Fourier consente di scomporre un'onda di qualsiasi tipo, anche molto complessa e "rumorosa", in più sotto-componenti, un po' come attraverso la chimica si può scomporre un cibo nei suoi sotto elementi così da carpirne la reale composizione.

Più precisamente la trasformata di Fourier permette di calcolare le diverse componenti (ampiezza, fase e frequenza) delle onde sinusoidali che, sommate tra loro, danno origine al segnale di partenza.

La funzione $X(f)$ viene chiamata trasformata di Fourier del segnale $x(t)$ e il segnale $x(t)$ viene chiamato antitrasformata di Fourier di $X(f)$.

Si tratta quindi di una trasformazione che associa ad ogni segnale $x(t)$, $t \in R$ una funzione $X(f)$, $f \in R$ detta appunto trasformata di Fourier. Dalla funzione $X(f)$, a sua volta, può essere ricostruito il segnale originario $x(t)$, attraverso l'antitrasformata di Fourier che corrisponde ovviamente alla trasformazione inversa.

Dato che alla trasformata di Fourier di un segnale si può attribuire significato fisico, la conoscenza di $X(f)$ consente di meglio interpretare la natura del segnale stesso. Inoltre, considerate le strette analogie esistenti tra l'insieme dei segnali e l'insieme delle loro trasformate, lo studio dei segnali nel dominio della frequenza consente di ottenere risultati che non sarebbe agevole ottenere nel dominio del tempo.

Con la trasformata di Fourier si ha quindi una completa simmetria fra dominio del tempo, $t \in R$, e dominio delle frequenze, $f \in R$; può essere allora vista come un'operazione che trasforma l'insieme delle funzioni complesse di variabile reale (la variabile $t$), o meglio quante tra loro sono dotate di trasformata di Fourier, in altre funzioni complesse di variabile reale (la variabile $f$); e si noti che le trasformate hanno la stessa natura dei segnali nel dominio del tempo, di modo che esse possono essere interpretate come segnali nel dominio della frequenza.

### 2.2.3 Perché è utile?

Una volta capito che la trasformata di Fourier scompone un'onda nelle sue sottoparti sinusoidali, possiamo cercare di vedere a cosa serve. L'utilità come già accennato all'inizio è molto vasta, tra cui:

1. Osservando le sotto componenti possiamo "filtrare" l'onda eliminando quelle frequenze che noi reputiamo di disturbo. Una volta ripulito il segnale si può riportarlo nel dominio del tempo (cioè nella situazione iniziale) usando l'antitrasformata ed ottenere un segnale più pulito.

2. Nel dominio della frequenza una funzione di spazio e tempo viene scomposta in frequenza, ampiezza e fase di più onde sinusoidali, abbiamo quindi più informazioni su cui lavorare e in cui mettere dei dati.

3. Lavorare con tante sinusoidi a diversa frequenza e ampiezza è molto meglio che lavorare con un'onda che continua a variare nel tempo, perché nel primo caso la matematica ci fornisce il calcolo tramite numeri complessi che rende molto più facile e veloce effettuare certe operazioni sull'onda iniziale (es: la convoluzione di più segnali nel tempo diventa un semplice prodotto nel dominio della frequenza).

### 2.2.4 Simmetrie fondamentali

Molto importante nel lavoro svolto per ridurre il tempo di calcolo della trasformata di Fourier è stata la conoscenza delle simmetrie fondamentali che i segnali possono presentare, e in particolare la simmetria hermitiana.

#### Segnali pari e segnali dispari

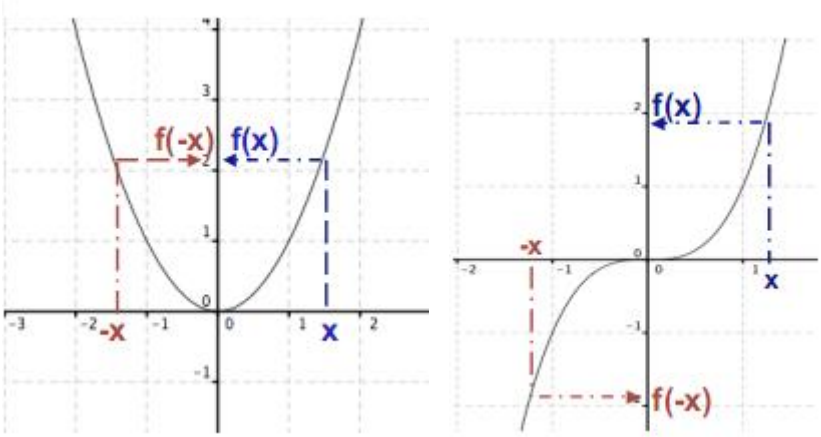
Un segnale $x(t)$ si dice pari se per ogni t risulta:

$$x(-t) = x(t),$$

mentre si dice dispari:

$$x(-t) = -x(t),$$

Si osservi che un segnale pari è simmetrico rispetto all'asse delle ordinate, mentre un segnale dispari è simmetrico rispetto all'origine. Ne consegue che per




(a) Simmetria pari (b) Simmetria dispari

Figura 2.2: Simmetrie

segnale pari o dispari è sufficiente conoscere l'andamento per $t \geq 0$ (oppure per $t \leq 0$).

### Simmetria Hermitiana

Si dice che un segnale $x(t)$, $t \in R$ è Hermitiano se soddisfa la condizione:

$$x(-t) = x^*(t),$$

mentre si dice che antihermitiano se:

$$x(-t) = -x^*(t).$$

Il significato della simmetria Hermitiana può essere esplicitato ponendo in evidenza parte reale parte immaginaria di $x(t) = x_r(t) + ix_i(t)$. Si ha allora:

$$x_r(-t) + ix_i(-t) = x_r(t) - x_i(t)$$

da cui seguono:

$$x_r(-t) = x_r(t), \qquad x_i(-t) = -x_i(t).$$

### 2.2.5 Segnali reali

Quanto appena visto può essere sfruttato nella trasformata di Fourier. Infatti se il segnale $x(t)$ è complesso, la trasformata di Fourier $X(f)$ non ha in generale alcuna simmetria; invece , se il segnale è reale (cosa che si presenterà nel progetto), si ha la simmetria Hermitiana:

$$X(-f) = X^*(f),$$

risultando da $x(t) = x^*(t)$ :

$$X(f) = \int_{-\infty}^{+\infty} x^*(t) e^{-i2\pi ft}\,\mathrm{d}t = \left\{ \int_{-\infty}^{+\infty} x(t) e^{i2\pi ft}\,\mathrm{d}t \right\}^* = X^*(-f).$$

Quindi la conoscenza di $X(f)$ per $f \geq 0$ dà la completa informazione sul segnale $x(t)$ e, posto

$$X(f) = R(f) + iW(f) = A_x(f) e^{i\beta_X(f)}$$

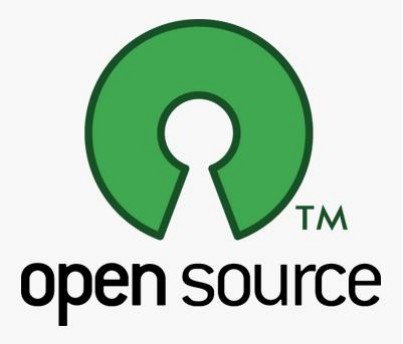
da $X(-f) = X^*(f)$ si trova

$$R(f) = R(-f), \qquad W(f) = -W(-f)$$

ossia: la parte reale è una funzione pari di $f$ e il coefficiente dell'immaginario è una funzione dispari. Analogamente si ottiene per il modulo e l'argomento:

$$A_X(f) = A_X(-f), \qquad \beta_X(f) = -\beta_X(-f).$$

## 2.3 Software Open Source




Con Software Open Source si intende software a codice sorgente aperto che, attraverso questa disponibilità, consente sia la sua libera circolazione sia processi di modifica, produzione, ridistribuzione, evoluzione e riuso.

L'idea di base nasce dal concetto che maggiore è la circolazione dell'informazione, maggiore sarà il numero delle persone coinvolte. Di conseguenza sarà più rapido lo sviluppo del codice, gli errori e le incompatibilità verranno evidenziate in tempi più brevi, infine si collezioneranno molti più feedback e richieste degli utenti.

Nel 1998 nasce OSI(Open Source Initiative), si tratta di un'organizzazione no profit che promuove gli aspetti positivi del software libero senza le licenze, ma fissando i criteri generali per il riconoscimento.

Le condizioni che definiscono la metodologia Open Source sono dieci, le prime tre garantiscono le libertà fondamentali, le altre l'assenza di discriminazioni e i diritti degli autori.

Principi fondamentali:

1. distribuzione aperta (non è impedita la vendita o la donazione),
2. accesso aperto (codice sorgente disponibile al prezzo di distribuzione),
3. modificabilità aperta,
4. integrità del codice sorgente dell'autore,
5. nessuna discriminazione contro le persone o gruppi,

6. nessuna discriminazione contro campi applicativi,

7. i diritti offerti dalla licenza si applicano autonomamente a tutti i destinatari,

8. la licenza non deve essere specificata ad un prodotto,

9. la licenza non deve porre vincoli su altro software allegato a quello licenziato,

10. la licenza non deve richiedere particolari tecnologie di accesso.


### 2.3.1 Licenza GNU GPL

La GNU (General Public License) è una delle licenze per software libero più conosciute ed usate. È stata scritta da Richard Stallman e Eben Molpen nel 1989, permette all'utente libertà di utilizzo, copia, modifica e distribuzione.

La licenza GNU GPL stabilisce che l'accettazione delle sue condizioni dia la possibilità di modificare il software, di copiare e redistribuire con o senza modifiche, sia gratuitamente che a pagamento. Rispetto alle altre licenze di software libero, la GPL è classificabile come "persistente" e "propagativa".

È "persistente" perché impone un vincolo alla redistribuzione: se l'utente distribuisce copie del software, deve farlo secondo i termini della GPL stessa. In pratica, deve distribuire il testo della GPL assieme al software e corredarlo del codice sorgente o di istruzioni per poterlo ottenere ad un costo nominale. Questa è la caratteristica principe della GPL, il concetto ideato da Richard Stallman e da lui battezzato copyleft. Il suo scopo è di mantenere libero un programma una volta che esso sia stato posto sotto GPL, anche se viene migliorato correggendolo e ampliandolo.

È "propagativa" perché definisce nel testo una particolare interpretazione di "codice derivato", tale che in generale l'unione di un programma coperto da GPL con un altro programma coperto da altra licenza può essere distribuita sotto

GPL, o in alternativa non essere distribuita affatto. Nel primo caso si dice che l'altra licenza è "compatibile con la GPL"; nel secondo caso, che non lo è. Questa caratteristica è indicata come strong copyleft nella terminologia della FSF. Il suo scopo è evitare che la persistenza venga via via indebolita apportando modifiche coperte da un'altra licenza meno libera, inficiando così lo scopo di mantenere libero il software coperto dalla GPL.

# Capitolo 3

# Postazione di lavoro

## 3.1 Eclipse

Eclipse è un ambiente di sviluppo integrato multi-linguaggio e multipiattaforma. Ideato da un consorzio di grandi società chiamato Eclipse Foundation sullo stile dell'open source, può essere utilizzato per la produzione di software di vario genere. Si passa infatti da un completo IDE[1] per il linguaggio Java (JDT, Java Development Tools) a un ambiente di sviluppo per il linguaggio C++ (CDT, C/C++ Development Tools) e a plug-in che permettono di gestire XML, Javascript, PHP.

Principalmente consiste in un editor di codice sorgente, un compilatore o un interprete, un tool di building automatico e di un debugger.

### 3.1.1 Installazione di Eclipse

Una fase importante nella preparazione della postazione è stata l'installazione di Eclipse per cross-compilare le applicazioni per sistemi integrati.

Le istruzioni successive forniscono una linea guida per usare Eclipse e compilare programmi in Linux con linguaggio C/C++ per "BeagleBoard-xM embedded system".

[1] Integrate development enviroment, in italiano ambiente di sviluppo integrato (conosciuto anche come integrated design enviroment o integrated debugging enviroment, rispettivamente ambiente integrato di progettazione e ambiente integrato di debugging) è un software che aiuta i programmatori nello sviluppo del codice.

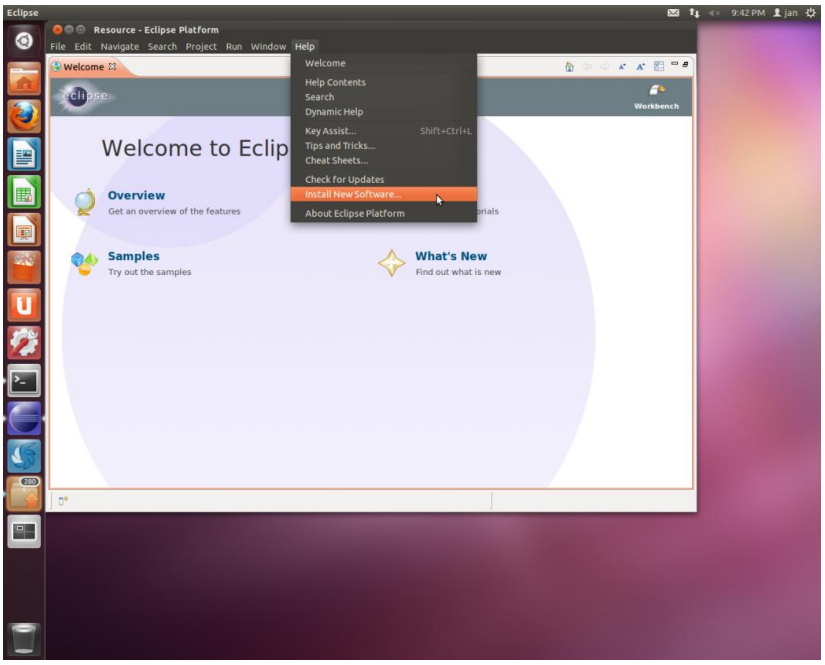
### 3.1.2 Installazione della toolchain

La prima cosa necessaria da verificare di possedere e nel caso da scaricare è la toolchain[2] con cross-compilatore adatto alla propria architettura, indispensabile per creare i programmi che verranno eseguiti sulla BeagleBoard-xM. La CodeSourceryToolchain per ARM Cortex A8 è ciò che fa al caso nostro.

### 3.1.3 Installazione di Eclipse e C/C++ Development Tools

Si può ora procedere all'installazione di Eclipse, scaricandolo da "Ubuntu Software Center", presente nella finestra delle "Applications". Una volta aperto Eclipse da terminale andremo ad installare i C/C++ Development Tools (CDT), attraverso i seguenti passi:

1. Da Eclipse selezionare "Help → Install New Software..."

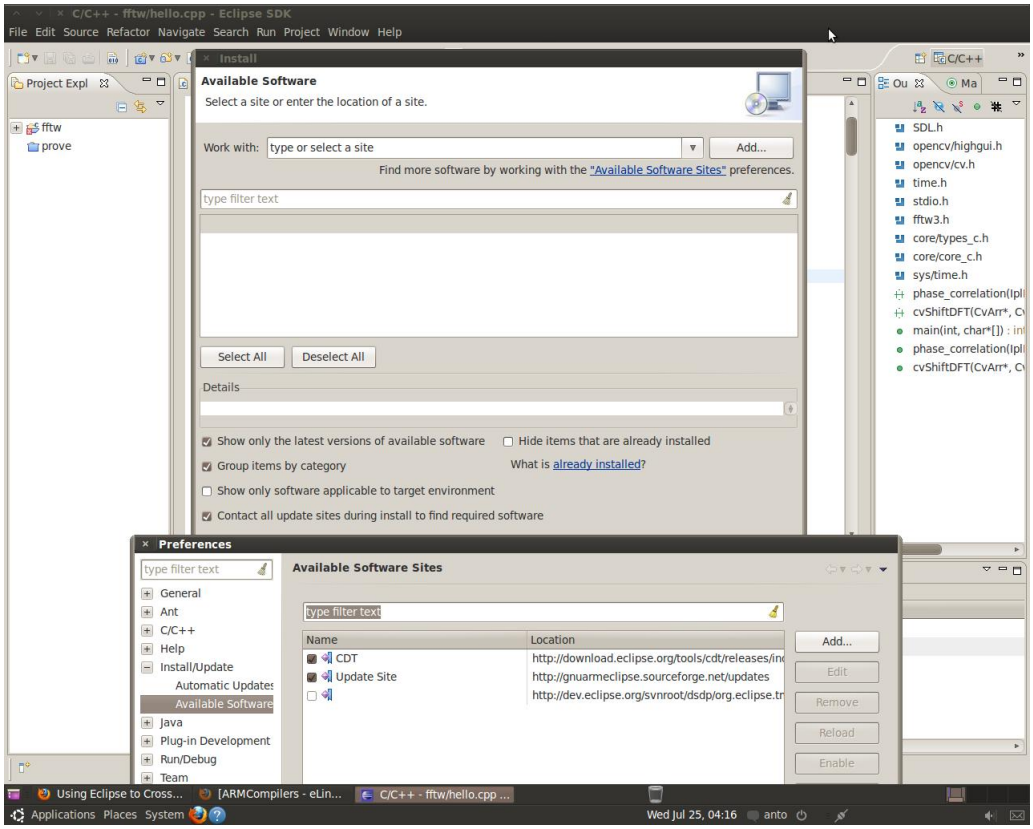



[2]In ambito software una toolchain è l'insieme dei programmi (tools) usati nello sviluppo di un prodotto (tipicamente un altro programma o sistema di programmi). I tool possono essere utilizzati in catena in moda tale che l'output di ciascun tool rappresenti l'input per il successivo, ma il termine è utilizzato in maniera più estesa per riferirsi, più in generale, a qualunque insieme di tool di sviluppo collegati tra loro.

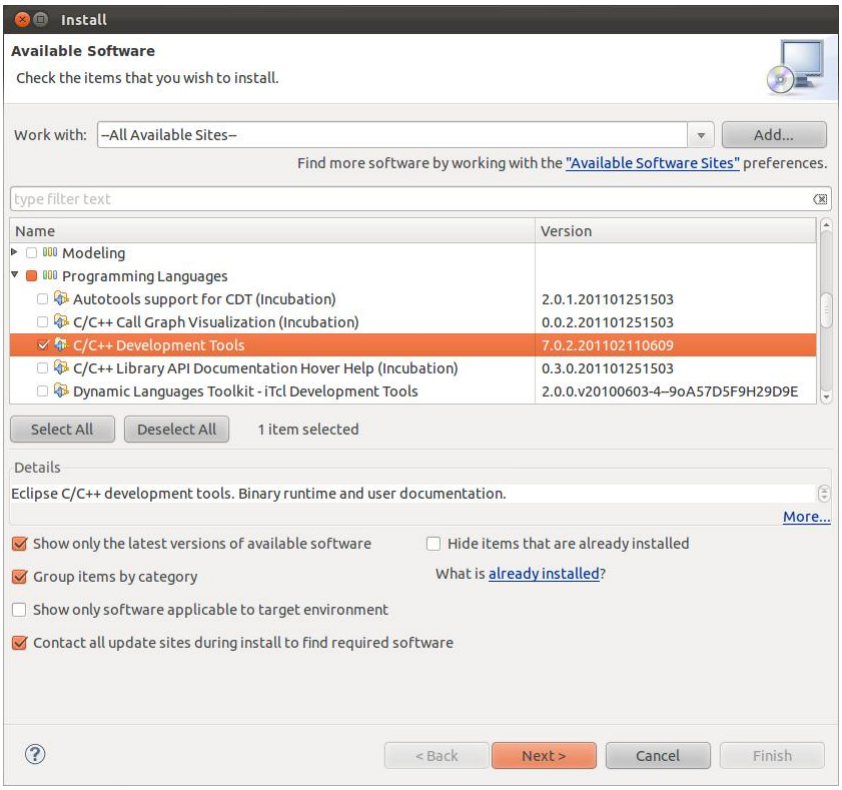
Un semplice esempio di toolchain per lo sviluppo del software è rappresentato da un editor di testo per l'inserimento del codice sorgente, un compilatore ed un linker per la trasformazione del codice sorgente in un programma eseguibile e le librerie che forniscono l'interfaccia col sistema operativo.

2. Nella finestra di installazione sotto la casella di testo “Work with”, click su “Available Software Sites” e selezionare:

   - CDT
   - Update Site



3. Ritornare nella finestra di installazione e sulla casella di testo “Work with” selezionare “All Available Sites”. Nella main list box, su “Programming Languages” scegliere “C/C++ Development Tools”, infine pulsante “Next >” e seguire le istruzioni accettando le licenze per installare il software.




### 3.1.4 Creazione e configurazione di un progetto

Ora che si è installato Eclipse siamo pronti per creare e configurare un progetto.

Per prima cosa aprire Eclipse e selezionare “File → New → C++ Project” ed inserire il nome del progetto.

Devo ora definire le “Build Configuration” per spiegare ad Eclipse come cross-compilare il progetto per la BeagleBoard-xM, ed è a questo punto che ci torna utile la CodeSourceryToolchain per ARM Cortex A8 precedentemente scaricata.

Tasto destro del mouse sul nome del progetto appena creato, scorrere sulla finestrella andando su “Build Configurations → Manage..” e creare una nuova configurazione per la cross-compilazione.

Inserire il il nome della configurazione nella “Name box” e selezionare “Existing configuration”.

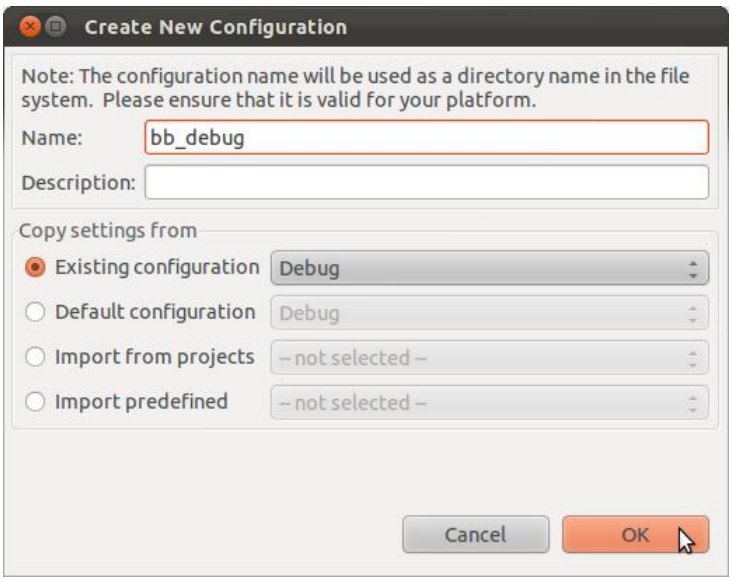
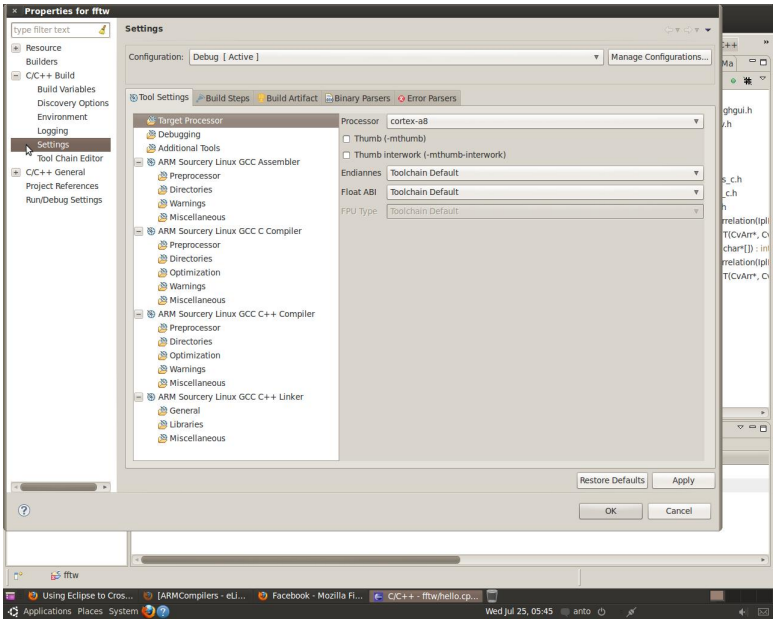



Figura 3.1: Creazione della nuova configurazione per la cross-compilazione

Si procederà ora con il settare le proprietà del compilatore, indicando i vari percorsi delle librerie del CodeSourcery; tasto destro del mouse sul nome del progetto e alla fine della finestra trovo “Properties”.

Una volta aperta la finestra “Settings” in “Configuaration” seleziono bb_debug (nome della nuova configurazione per la cross-compilazione).



Bisogna ora indicare i percorsi per i vari path sul command box nelle Configuration delle ARM Sourcery Linux GCC :

- ARM Sourcery Linux GCC Assembler: /.../CodeSourcery/arm-2011.03/bin /arm-none-linux-gnueabi-as

- ARM Sourcery Linux GCC C Compiler: /.../CodeSourcery/arm-2011.03/bin /arm-none-linux-gnueabi-gcc

- ARM Sourcery Linux GCC C++ Compiler: /.../CodeSourcery/arm-2011.03/bin /arm-none-linux-gnueabi-g++

- ARM Sourcery Linux GCC C++ Linker: /.../CodeSourcery/arm-2011.03/bin /arm-none-linux-gnueabi-g++

Come ultima fatica si procede andando nella voce "Directories" sottostante "ARM Sourcery Linux GCC C++ Compiler" e nella finestra "Include paths (-l)" aggiungere:

/.../CodeSourcery/arm-2011.03/arm-none-linux-gnueabi/include

Operazione simile va fatta per "ARM Sourcery Linux GCC C++ Linker" alla voce "Libraries" nella parte bassa della finestra in "Library search path (-L)" aggiungere:

/.../CodeSourcery/arm-2011.03/arm-none-linux-gnueabi/libc/usr/lib

Ora Ecliplse è configurato e pronto per essere utilizzato.

## 3.2 Elaborazione di immagini con la libreria OpenCV

### 3.2.1 Un po' di storia


OpenCV nasce nel 1999 nei laboratori di Intel grazie a Gary Bradski, come strumento di test e "stress" per l'analisi delle CPU in fase di sviluppo. L'elaborazione di immagini è ben noto essere una delle applicazioni che sfrutta al 100% la potenza delle unità di calcolo per l'enorme quantità di dati da elaborare in tempi brevissimi

Nel 2000 è stata rilasciata al pubblico la prima versione Alpha durante la conferenza internazionale "Conference on Computer Vision and Pattern Recognition" organizzata dalla IEEE. Inizialmente OpenCV aveva tre scopi principali:

1. Ricerca avanzata sulla Visione Artificiale grazie ad un codice non solo "open", ma soprattutto ottimizzato, come strumento di base per un'infrastruttura più avanzata. (Da ricordare l'integrazione con le librerie IPP di Intel, ancora oggi "attiva").

2. Diffondere conoscenza sulla Visione Artificiale fornendo un'infrastruttura comune agli sviluppatori in modo da poter sviluppare codice più leggibile e facilmente trasferibile.

3. Creazione di applicazioni commerciali avanzate grazie a codice portabile ottimizzato e disponibile gratuitamente, senza l'obbligo di distribuire il codice scritto sfruttando OpenCV stessa.

Tra il 2001 e il 2005 alla prima Alpha sono seguite cinque versioni Beta fino ad arrivare alla sofferta Release 1.0 nel 2006.

Vadim Pisarevsky si è unito a Gary Bradsky nei laboratori russi di Intel per portare avanti lo sviluppo della libreria. Nel susseguirsi degli anni e delle versioni il Team di OpenCV lasciò Intel e si spostò dalla compagnia in centri di ricerca, fino ad arrivare a Willow Garage, attuale "manutentore".

Nel 2008 Willow Garage si è resa conto della necessità di dotare rapidamente la robotica di strumenti di percezione seguendo la filosofia "open", in modo da accrescere le potenzialità delle comunità scientifiche e commerciali e iniziò a supportare attivamente OpenCV mantenendo Gary e Vadim a capo del team di sviluppo della libreria.

Nell'ottobre 2009, con l'uscita della Release 2.0, alla "classica" interfaccia in linguaggio C di OpenCV si è affiancata la più moderna interfaccia C++, che ha aggiunto alla già potente libreria tutti i vantaggi e i pregi della programmazione orientata agli oggetti, ma soprattutto una migliore gestione della memoria dinamica grazie ad una gestione degli oggetti simile alla "Garbage Collection" tipica del linguaggio Java.

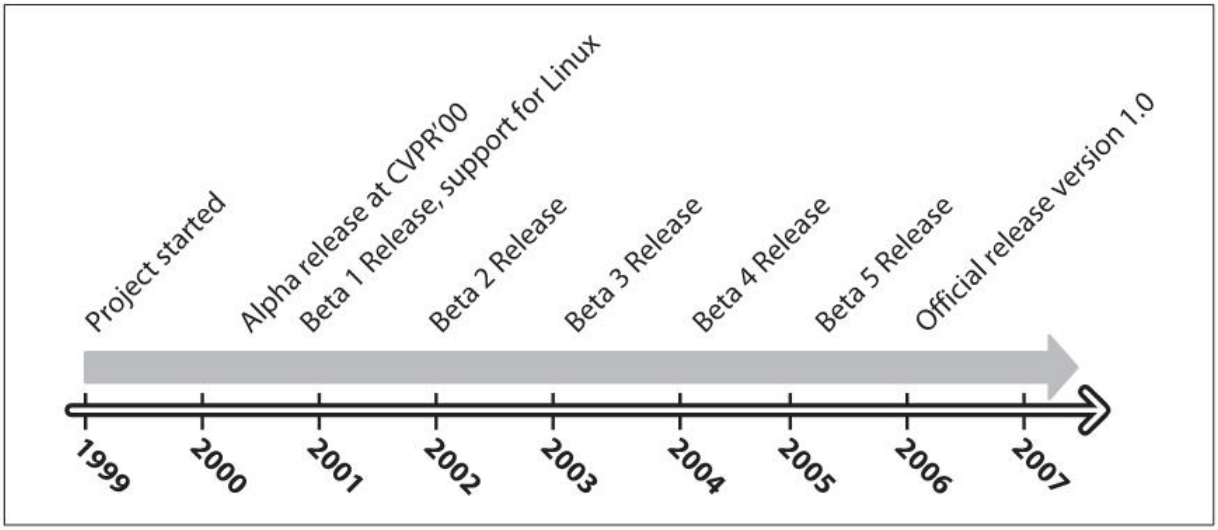



Figura 3.2: OpenCV time-line

### 3.2.2 Introduzione

Le tecniche di image processing sono impiegate nel miglioramento della qualità delle immagini, nel controllo di prodotti industriali, nell'analisi di immagini satellitari, nell'elaborazione di immagini biomedicali, e più in generale come passo di pre-elaborazione nei procedimenti di computer vision.

La OpenCV Source Computer Vision Library è una libreria scritta in C e C++, costituita da oltre 500 funzioni utili nel campo dell'image processing e

della computer vision. Punti di forza della libreria sono la completezza e il suo essere open source. Il fatto di essere liberamente distribuita garantisce sicurezza al codice e la possibilità di apportare modifiche al codice stesso, assicurando così una continua evoluzione. La licenza di distribuzione è priva di royalty e ciò consente il suo utilizzo anche in prodotti commerciali a condizione di mantenere le note di copyright.

OpenCV costituisce quindi una infrastruttura aperta e gratuita, compatibile con la Intel Image Processing Library (IPL). Quest'ultima è una libreria non open source che esegue solo operazioni di image processing e dalla quale OpenCV ha ereditato originariamente alcune strutture dati.

La portabilità della libreria OpenCV è completa, sono infatti disponibili le versioni per i sistemi MS-Windows, Linux, BSD, Unix e MacOSX.

OpenCV è stato progettato per un'efficiente funzionamento a livello computazionale e nelle applicazioni real-time. Uno degli obbiettivi di OpenCV è quello di fornire un mezzo semplice da usare nella computer vision aiutando così le persone a costruire, abbastanza rapidamente e agevolmente, sofisticate applicazioni di visione.

### 3.2.3 Cos'è la computer vision?

La computer vision è la trasformazione di dati da un'immagine o da un video in un nuovo tipo di rappresentazione. Tutte queste trasformazioni sono eseguite per conseguire alcuni particolari obbiettivi, come ad esempio la necessità di contare il numero di persone presenti in una scena o l'individuazione di cellule tumorali.

La computer vision non è affatto un meccanismo semplice da far funzionare; in un sistema di visione artificiale infatti, il computer riceve una griglia di numeri dalla fotocamera, e queste sono le uniche informazioni che si hanno. Non è presente un riconoscimento delle forme, nessun controllo automatico di messa a fuoco, non una cross-associazione con anni di esperienza, come avviene invece nella visione umana nella comunicazione occhio-cervello, e che avrebbe potuto

illudere il lettore, portandolo a immaginare la computer vision come una "passeggiata" e chiedersi "cosa ci vuole ad individuare in un'immagine il numero di persone?".

Avvincente è quanto riportato da Microsoft sulla sezione del sito dedicato alla ricerca sulle tecnologie visive: "Lo scopo della ricerca sulla computer vision è di dotare il computer dell'abilità di comprendere le immagini ferme e in movimento. Anche se noi, come esseri umani, possiamo dare un senso alle fotografie e ai video, per un computer esse sono solo una matrice di numeri rappresentante la luminanza e il colore di un pixel. Come possiamo ottenere da questa matrice di numeri la comprensione che c'è una ragazza che sta giocando a pallone davanti all'edificio? Quanto alto sta gettando la palla? Qual è la struttura di fondo della casa? Questi sono gli argomenti che ci interessano!".

Per capire meglio prendiamo ad esempio l'immagine di figura 3.3, e ci concentriamo sullo specchietto dell'auto, ciò che il computer vede è unicamente una griglia di numeri e inoltre, una griglia di numeri costituita da una forte componente di rumore, insomma non molte informazioni. Il nostro compito diventa quindi trasformare questa rumorosa griglia di numeri nella percezione: "specchietto laterale".

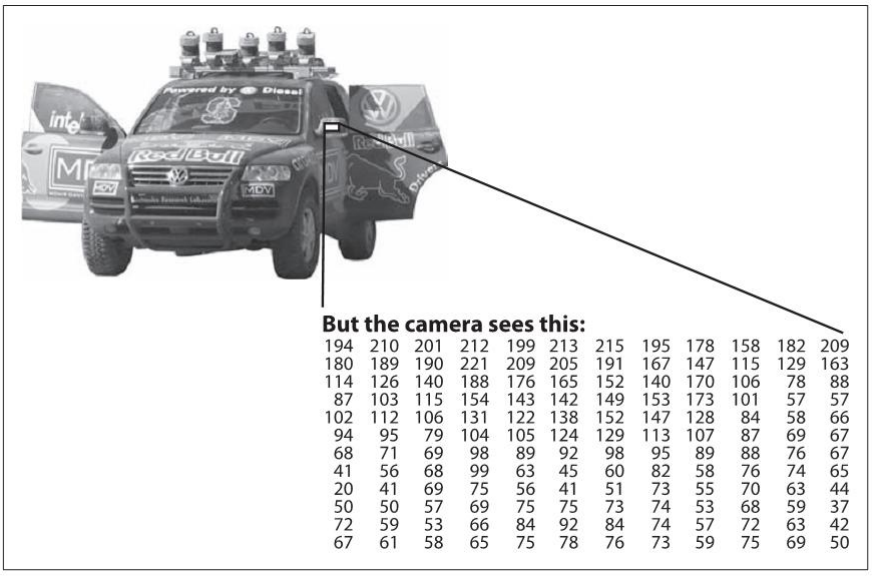



Figura 3.3: Lo specchietto della macchina è una matrice di numeri

OpenCV è volto a fornire gli strumenti di base necessari per risolvere i problemi della computer vision. In alcuni casi saranno sufficienti le funzioni di alto livello presenti nella libreria a risolvere anche i più complessi problemi, ma anche quando ciò non avverrà, le componenti di base all'interno della libreria consentiranno la creazione di una completa soluzione per qualsiasi tipo di problema della computer vision.

### 3.2.4 Installazione

I successivi step sono stati utilizzati per l'installazione di OpenCV in Ubuntu 10.04, ma possono essere utilizzati anche per altre distribuzioni.

#### Installazione di OpenCV utilizzando Cmake da riga di comando

L'installazione di OpenCV per ARM Cortex-A8 nel sistema operativo Ubuntu 10.04, è avvenuta seguendo i seguenti passi, appoggiandosi alla guida internet presente in questo sito: http://processors.wiki.ti.com/index.php/Building_OpenCV_for_ARM_Cortex-A8.

In questa guida sono presenti le istruzioni di base per installare OpenCV nella piattaforma ARM Cortex-A8 utilizzando Cmake nella distribuzione Linux. Le istruzioni sottostanti sono state utilizzate per la distribuzione di OpenCV2.3.1 cross-compilandola nel sistema operativo Ubuntu 10.04. La toolchain è la Codesourcery 2011.03.

Per prima cosa scarico OpenCV dal seguente link http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/; continuano ad uscire nuove versioni, io personalmente ho scelto la 2.3.1 (anche se non è l'ultima) perché è quella che, a parere di molti "smanettoni" e anche mia, più affidabile e meglio sviluppata al momento. Da terminale scompatto il file attraverso il comando:

$$tar - jxfOpenCV - 2.3.1.tar.bz2$$

Prima di iniziare con l'installazione vera e propria di OpenCV è necessario verificare di avere tutti i seguenti pacchetti installati all'interno del proprio computer: *build-essential, cmake, pkg-config, libpng12-0 libpng12-dev libpng++-dev libpng3, libpnglite-dev libpngwriter0-dev libpngwriter0c2, zlib1g-dbg zlib1g zlib1g-dev, pngtools libtiff4-dev libtiff4 libtiffxx0c2 libtiff-tools, libjpeg8 libjpeg8-dev libjpeg8-dbg libjpeg-progs, ffmpeglibavcodec-dev libavcodec52 libavformat52 libavformat-dev, libg streamer0.10-0-dbg libgstreamer0.10-0 libgstreamer0.10-dev, libxine1-ffmpeg libxine-dev libxine1-bin, libunicap2 libunicap2-dev, libdc1394-22-dev libdc1394-22 libdc1394-utils, swig, libv4l-0 libv4l-dev, python-numpy, libpython2.6 python-dev python2.6-dev, libgtk2.0-dev pkg-config.*

Nel caso non ne sia presente uno o più di essi si può facilmente risolvere il problema installandolo/i attraverso il comando, da terminale, sudo apt-get install nome del pacchetto, ad esempio sudo apt-get install cmake, nel caso non sia presente il pacchetto cmake.

Arrivati a questo punto si eseguono i seguenti passaggi trovati seguendo la guida al link precedente citato:

1. Creare una cartella che io chiamo "release" , dove metteremo i Makefiles generati, i projectfiles, objectfiles e output binaries con le seguenti istruzioni:

```
cd OpenCV-2.3.1
mkdir release
cd release
```

2. Creare e salvare all'interno della cartella release un file che chiamo "codesourcery.cmake" all'interno del quale scrivo il seguente testo:

```
set(toolchain_dir /opt/CodeSourcery/arm-2011.03) # SET THIS TO YOUR TOOLCHAIN PATH
set(toolchain_bin_dir ${toolchain_dir}/bin)
set(toolchain_libc_dir ${toolchain_dir}/arm-none-linux-gnueabi/libc)
set(toolchain_inc_dir ${toolchain_libc_dir}/usr/include) # was /include
set(toolchain_lib_dir ${toolchain_libc_dir}/usr/lib)

set(CMAKE_SYSTEM_NAME Linux)# CACHE INTERNAL "system name")
set(CMAKE_SYSTEM_PROCESSOR arm)# CACHE INTERNAL "processor")
set(CMAKE_C_COMPILER ${toolchain_bin_dir}/arm-none-linux-gnueabi-gcc)
set(CMAKE_CXX_COMPILER ${toolchain_bin_dir}/arm-none-linux-gnueabi-g++)
set(CMAKE_C_FLAGS "-isystem ${toolchain_inc_dir}") #CACHE INTERNAL "c compiler flags")
set(CMAKE_CXX_FLAGS "-isystem ${toolchain_inc_dir}") #CACHE INTERNAL "cxx compiler flags")
SET(EXTRA_C_FLAGS "-pipe -Os -mtune=cortex-a8 -march=armv7-a -mabi=aapcs-linux -msoft-float -fPIC")

set(link_flags -L${toolchain_lib_dir})

set(CMAKE_EXE_LINKER_FLAGS ${link_flags}) #CACHE INTERNAL "exe link flags")
set(CMAKE_MODULE_LINKER_FLAGS ${link_flags}) #CACHE INTERNAL "module link flags")
set(CMAKE_SHARED_LINKER_FLAGS ${link_flags}) #CACHE INTERNAL "shared link flags")
set(CMAKE_FIND_ROOT_PATH ${toolchain_libc_dir}) #CACHE INTERNAL "cross root directory")
set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
```

3. È ora arrivato il momento di generare il "MakeFile", digitando da terminale all'interno della cartella "release": cmake -DCMAKE_TOOLCHAIN_FILE = codesourcery.cmake ../OpenCV-2.3.1/ (percorso della cartella opencv dove è presente il file CMakeLists.txt)

4. Attraverso il comando "ccmake." posso aprire un'interfaccia sul terminale che mi permette di resettare alcuni parametri e rigenerare il MakeFile.

5. Ora per completare l'installazione è necessario e sufficiente digitare i comandi make e successivamente make install.

OpenCV è installato e pronto per essere utilizzato.

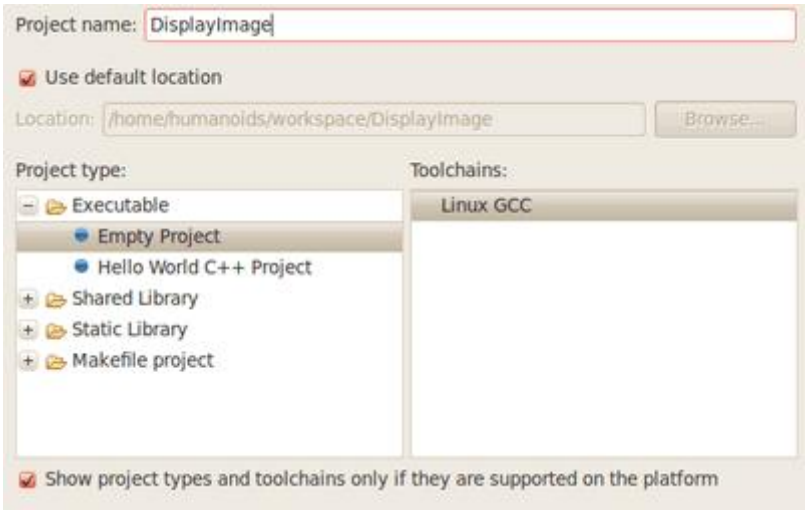
### 3.2.5 Utilizzo di OpenCV con Eclipse

Con Eclipse installato e pronto all'uso seguendo i passaggi visti nel capitolo precedente, possiamo sfruttare questo ambiente di sviluppo per utilizzare le librerie messe a disposizione da OpenCV. Una volta aperto Eclipse si procede con la creazione di un nuovo progetto:

1. Per la creazione del progetto con le impostazioni necessarie per la cross-compilazione con la BeagleBoard-xM basterà seguire le istruzioni esposte nel capitolo precedente in "Creazione e Configurazione di un Progetto".

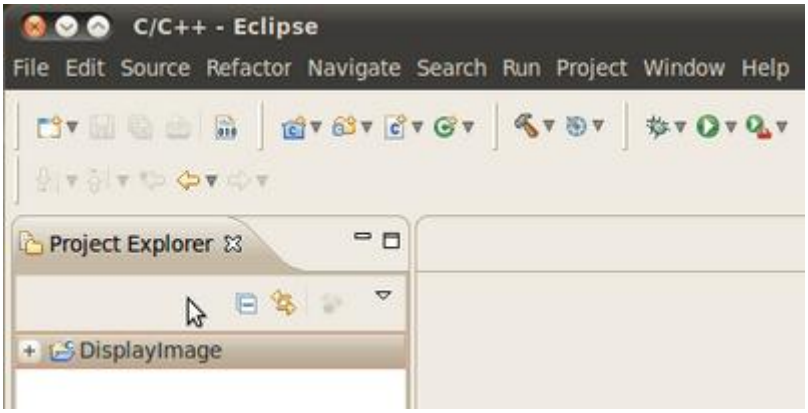

Figura 3.4: Creazione del nuovo progetto da Eclipse

2. Scegliere il nome del progetto (DisplayImage in questo caso)




3. Lasciare i parametri selezionati per default e premere "Finish".

4. Il progetto appena creato (DisplayImage) apparirà nella parte sinistra della finestra chiamata “Project Explorer”:




5. Ora si può procedere ad aggiungere un “source file”:

   - Testo destro del mouse su DisplayImage e procedere andando in “New→Source File”.
   - Lo chiamo DisplayImage.cpp e premo “Finish”.


6. Si può iniziare a scrivere un semplice codice.

7. L’ultimo passo consiste nel “dire” a OpenCV dove sono presenti i suoi headers e librerie. Per fare ciò si procede in questo modo:

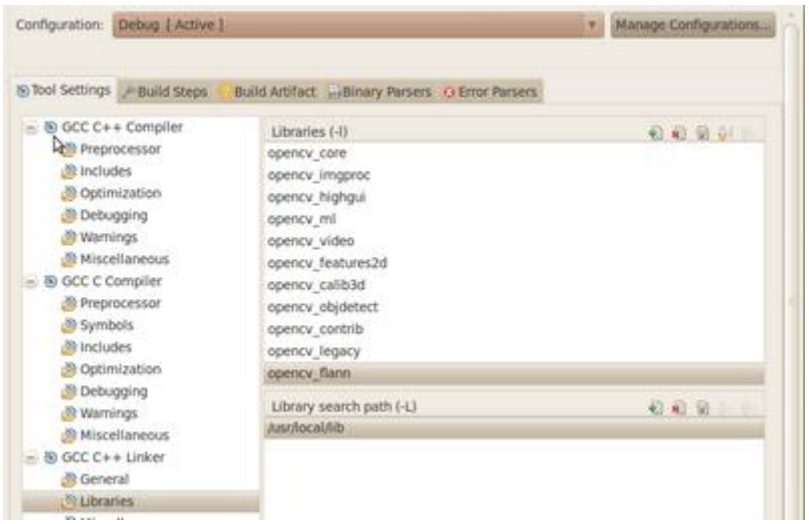
   - Tasto destro del mouse sul mio progetto e andare su “Properties”.
   - In “C/C++ Build” selezionare “Settings” arrivando così nella sezione dedicata all’immissione delle informazioni sulla posizione degli headers e delle librerie:

(a) In “GCC C++ Compiler”, andare in “Includes” e nella sezione “Include paths(-l)” bisogna includere le path della cartella dove è installato OpenCV

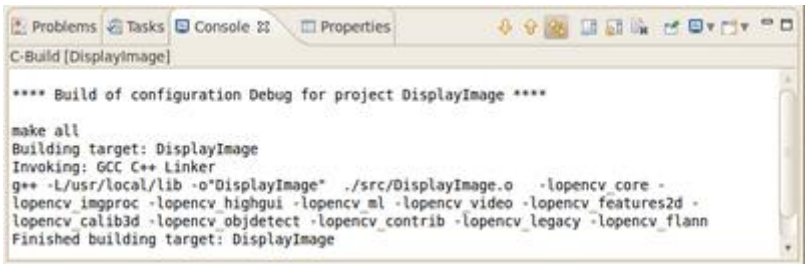

(b) In “GCC C++ Linker” sono presenti due finestre:

nella prima “Libraries(-l)” bisogna aggiungere le librerie di OpenCV che saranno necessarie per il funzionamento del programma: *opencv_core, opencv_imgproc, opencv_highgui, opencv_ml, opencv_video, opencv_features2d, opencv_calib3d, opencv_objdetect, opencv_contrib, opencv_legacy, opencv_flann.*

Nella seconda “Library search path (-L)” bisogna scrivere il percorso dove sono presenti le librerie di OpenCV.



Ora è tutto pronto e si può procedere, andando su "Project → Build All" posso vedere se il codice è corretto, e se non ci sono errori nella console apparirà qualcosa di simile a questo:




### 3.2.6 Convenzioni

Tutte le funzioni della libreria sono caratterizzate dal prefisso cv, mentre i nomi delle strutture hanno prefisso Cv ad eccezione della IplImage, la quale è eredita dalla libreria IPL. Poiché le funzioni operano con immagini e con matrici numeriche, le dimensioni seguono rispettivamente la convenzione di larghezza-altezza (X, Y) comune nella grafica e l'origine righe-colonne (R, C).

In genere i nomi delle funzioni hanno la forma cv<ActionName><Object>, dove ActioName rappresenta l'azione principale della funzione, ad esempio cvCreate<Object>, cvRelease<Object>, mentre Object è l'oggetto su cui avviene l'azione, ad esempio cvReleaseImage. Altre funzioni assumono invece il nome dell'algoritmo che implementano, ad esempio cvCanny.

### 3.2.7 Organizzazione delle librerie

La libreria è strutturata in cinque sottolibrerie (dette anche moduli) ciascuna con funzionalità specifiche.

Il modulo CXCORE è quello principale nonché indispensabile. Contiene le strutture dati di base con le rispettive funzioni di inizializzazione, le funzioni matematiche, le funzioni di lettura, scrittura e memorizzazione dati, le funzioni di sistema e di gestione degli errori.

Il modulo CV contiene le funzioni relative all'image processing, le funzioni di analisi strutturale del moto, di objectdetection e ricostruzione 3D.

HIGHGUI contiene le funzioni GUI e quelle di salvataggio e caricamento immagini, nonché le funzioni di acquisizione video e di gestione delle telecamere.

Il modulo ML (Machine Learning) contiene le classi e funzioni relative all'implementazione e gestione di reti neurali, in particolare di tipo multilayerperceptions (MPL), di classificazione statistica e clustering di dati.

Infine vi è il modulo CVAUX che contiene sia le funzioni basate su algoritmi ancora in fase di sperimentazione, il cui futuro è quello di migrare nel modulo CV, e sia le funzioni considerate obsolete e quindi non più supportate. Le funzionalità di tale modulo sono rivolte alla corrispondenza stereo, al tracking 3D, al riconoscimento degli oggetti.

### 3.2.8 Strutture dati

Introduciamo alcune delle strutture dati e funzioni base utili per iniziare ad operare con la libreria.

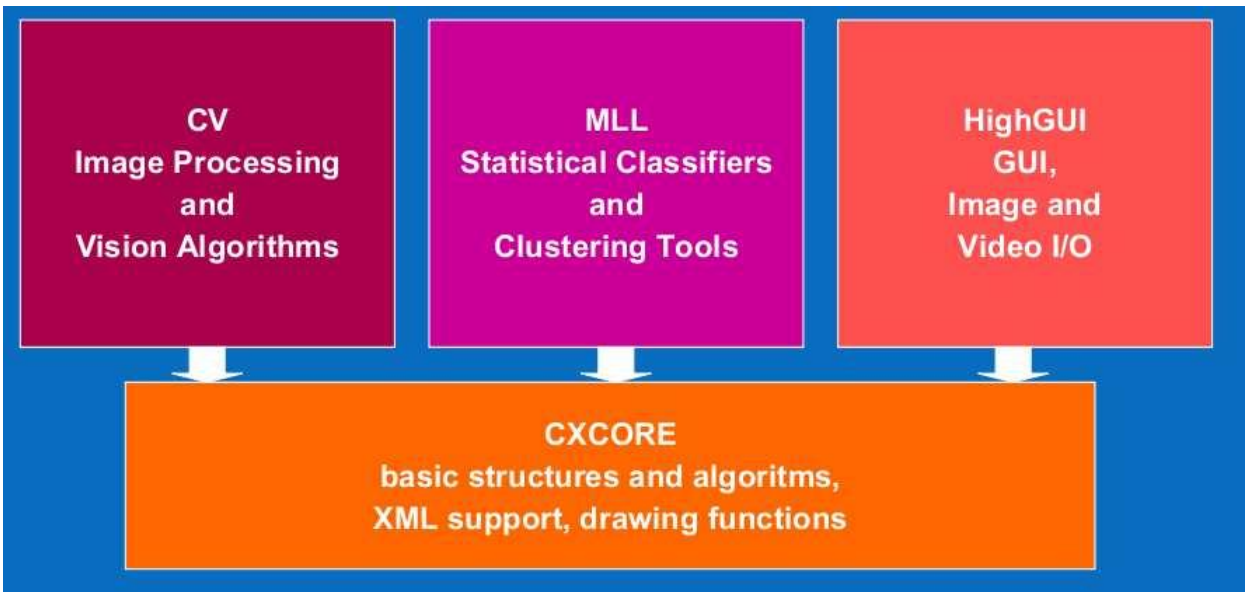



Figura 3.5: Organizzazione delle librerie

### CvArr

La maggior parte delle funzioni eseguono operazioni su strutture dati diverse tra loro, come le immagini (IplImage), le matrici (CvMat) e le liste dinamiche (CvSeq).

Le funzioni che eseguono operazioni su questi tipi di dati ammettono sempre un virtuale oggetto array CvArr.

La relazione esistente tra le strutture CvArr, CvMat e IplImage è di tipo Object-Oriented, dove la IplImage deriva dalla CvMat la quale a sua volta deriva dalla CvArr.

### IplImage

IplImage è l'acronimo di Image Processing Library Image, ossia il formato e standard utilizzato da Intel per rappresentare un'immagine nelle API IPL e OpenCV.

Tramite questa struttura dati è possibile gestire completamente tutto ciò che ruota intorno alle immagini, come caricamento e salvataggio, conversione di formato, elaborazione, filtraggio e visualizzazione. La struttura definisce sia immagini con origine top-left che immagini avente origine bottom-left, inoltre è costituita dai seguenti campi che nel loro insieme prendono il nome di Header:

typedef struct _IplImage {

int nSize;

int ID;

int nChannels;

int alphaChannel;

int depth;

char colorModel;

char channelSeq;

int dataOrder;

int origin;

int align;

int width;

int height;

struct _IplROI* roi;

struct _IplImage* maskROI;

void* imageId;

struct _IplTileInfo* tileInfo;

int imageSize;

char *imageData;

int widthStep;

int BorderMode;

int BorderConst;

char* imageDataOrigin;

} IplImage;

La libreria OpenCV, a differenza della IPL, utilizza solo alcuni campi della struttura.

Tra questi abbiamo nChannels che indica il numero di canali, e il campo depth che indica la profondità in bit. Per quest'ultimo i valori possibili sono : IPL_DEPTH_8U intero a 8-bit unsigned, IPL_DEPTH_8S intero a 8-bit con segno, IPL_DEPTH_16S e IPL_DEPTH_16U interi a 16-bit con e senza segno, IPL_DEPTH_32S intero a 32-bit con segno, IPL_DEPTH_32F floating point a 32-bit e IPL_DEPTH_64F floating point a 64-bit.

Il campo origin assume valore 0 quando l'origine delle coordinate è top-left, viceversa assume il valore 1 quando l'origine è bottom-left; i campi width e height indicano rispettivamente l'ampiezza e l'altezza in pixel dell'immagine, mentre il campo struct _IplROI* roi specifica una regione dell'immagine da elaborare (Region Of Interest). ImageSize contiene le dimensioni in bytes dell'intera immagine.

La funzione che consente di creare e allocare un'immagine è la seguente:

*cvCreateImage(CvSize size, int depth, int channels);*

Il parametro size permette di assegnare le dimensioni relative all'ampiezza e a all'altezza, il parametro depth assegna la profondità in bit ed infine channels che assegna il numero di canali. Ad esempio, volendo allocare un'immagine (img) di dimensione 640 $\times$ 320 con profondità a 8-bit con un solo canale, possiamo scrivere:

*CvSize size = cvSize(640, 320);*
*IplImage* img = cvCreateImage(size, IPL_DEPTH_8U, 1);*

La struttura CvSize è quindi costituita da due membri di tipo intero utili per definire le dimensioni, espresse in pixel, dell'immagine. La funzione costruttore corrispondente è:

*cvSize (int width, int height);*

e l'accesso ai campi si ottiene tramite l'operatore punto " . ":

*size.width = 640;*
*size.height = 320;*

### 3.2.9 Immagini e spazi colore

Si può, per semplicità, considerare un'immagine come una distribuzione bidimensionale di intensità luminosa:

$$f = f(x, y).$$

Poiché essa è acquisita tramite sensori, l'uscita è normalmente una grandezza continua le cui variazione spaziali di ampiezza seguono una legge dipendente dal tipo di sensore. Per poter essere trattata con un computer è necessario convertirla

in formato numerico. La creazione dell'immagine digitale a partire dal segnale analogico richiede un processo di campionamento e di quantizzazione.

Il campionamento è la discretizzazione dei valori delle coordinate spaziali, mentre la quantizzazione è la discretizzazione dei valori di intensità.

Un semplice modello consiste nel rappresentare l'intensità mediante il prodotto di due termini, l'illuminazione $i(x, y)$ e la riflettanza $r(x, y)$. L'immagine è quindi costituita da una componente dovuta alla luce proveniente dalla sorgente di illuminazione e da una componente dovuta alla luce riflessa dagli oggetti presenti sulla scena:

$$f(x, y) = i(x, y) \cdot r(x, y)$$

$$con \quad 0 < f(x, y) < \infty, \qquad 0 < r(x, y) < 1.$$

Tenuto conto del campionamento spaziale, che rende discreti gli intervalli di variazione di x e y, e assumendo che l'immagine continua sia approssimata mediante M × N campioni equispaziati lungo le dimensioni, si ha la sua rappresentazione classica sotto forma di matrice.

Un'immagine digitale è quindi una matrice di valori discreti di intensità luminosa costituita da M × N pixel ognuno dei quali ha valore compreso nell'intervallo $[0, 2^k - 1]$, dove k è il numero di bit usato per codificarlo.

Un pixel è una regione rettangolare coincidente con una cella della griglia di campionamento, e il valore ad esso associato è l'intensità media nella cella. Dal campionamento si determina la risoluzione spaziale, la quale si indica come risoluzione lungo x e lungo y, e definisce il numero di pixel da usare; maggiore è il numero, maggiori sono i particolari visibili.

I formati generalmente usati nell'elaborazione di immagini sono: a due livelli e in scala di grigi. Nelle immagini a due livelli (binarie) ciascun pixel può assumere solo il valore 1 (bianco) o il valore 0 (nero). In quelle monocromatiche ciascun elemento della matrice è un valore intero compreso tra 0 e 255, e rappresenta uno dei 256 livelli di grigio.

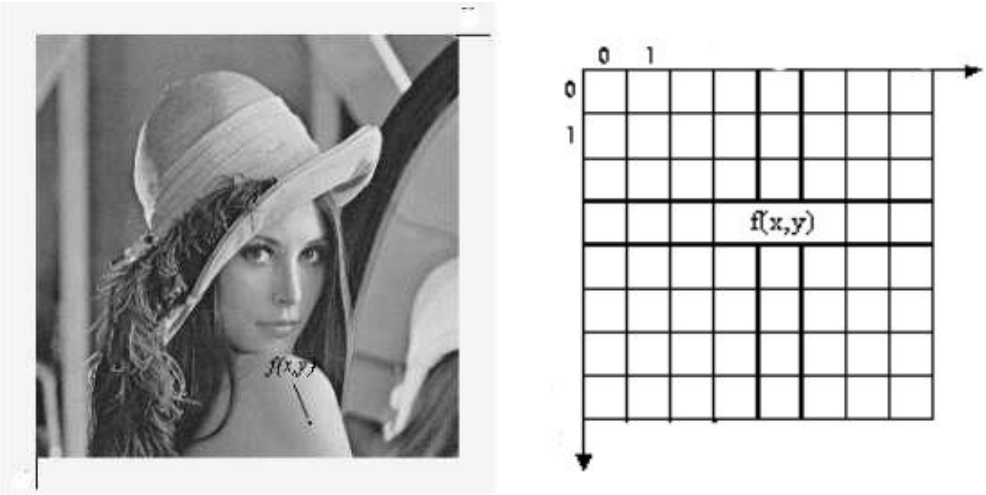



Figura 3.6: Rappresentazione di un'immagine digitale

Nelle immagini a colori ogni elemento della matrice corrisponde a una terna ordinata di interi (R, G, B), dove R, G, B indicano rispettivamente le intensità di rosso, verde e blu e assumono valori compresi tra 0 e 255, corrispondenti ai 256 diversi livelli di intensità delle tre sorgenti di colore.

Lo spazio colore è generalmente tridimensionale in modo da poter essere rappresentato da una terna di numeri, e i colori fondamentali RGB sono additivi, ossia i contributi individuali di ciascuno sono sommati insieme per produrre il risultato. La forma dello spazio che contiene al suo interno i colori visibili è un cubo unitario, dove la diagonale principale rappresenta i livelli di grigio.

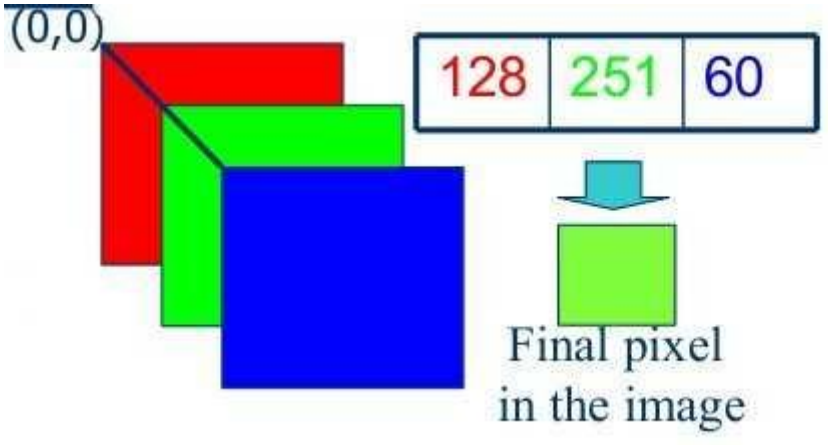



Figura 3.7: Spazio colore

OpenCV è in grado di gestire numerosi spazi colore, consentendo il passaggio

da uno spazio all'altro tramite la funzione:

*void cvCvtColor(const CvArr* src, CvArr* dst, int code);*

Le due immagini src e dst devono essere dello stesso tipo (8U, 16U e 32F ) ma possono avere differente numero di canale. Il parametro code specifica il tipo di conversione che deve essere eseguita utilizzando l'espressione CV_<spazio_colore_sorgente>2<spazio_
Ad esempio, con il codice di conversione CV_RGB2GRAY la funzione converte un'immagine a colori (RGB) in una a toni di grigio. Le intensità dell'immagine a toni di grigio sono calcolate secondo la formula:

$$Y = (0.299)R + (0.587)G + (0.144)B$$

Viceversa, nella conversione di un'immagine a toni di grigio in una a colori (CV_GRAY2RGB) le componenti sono prese tutte con lo stesso valore:

$$R = Y, \;\; G = Y, \;\; B = Y$$

## 3.3 Beagle Board

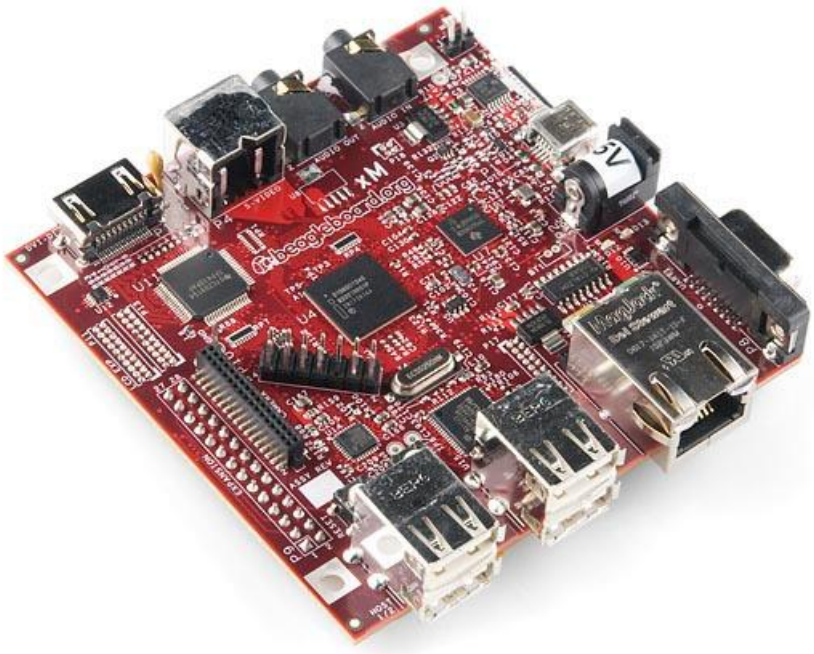


Figura 3.8: Beagle Board xM

La Beagle Board è un computer su singola scheda, basso costo e consumi in grado di offrire prestazioni e connettività prossime a quelli di un laptop. Infatti la Beagle Board si basa su un potente SoC[3] della Texas Instuments[4], l'OMAP3530

[3]Il termine System on a Chip (o System-on-a-chip spesso abbreviato in SoC) viene utilizzato per indicare quei particolari circuiti integrati che in un solo chip contengono un intero sistema, o meglio, oltre al processore centrale, integrano anche un chipset ed eventualmente altri controller come quello per la memoria RAM, la circuiteria input/output o il sotto sistema video.

Un singolo chip può contenere componenti digitali, analogici e circuiti in radiofrequenza in un unico integrato. Questa tipologia di integrati viene utilizzata comunemente nelle applicazioni embedded, date le dimensioni ridotte che essi raggiungono con l'integrazione di tutti i componenti.

[4]Texas Instruments Incorporated, meglio conosciuta nel mondo della industria elettronica con l'acronimo TI, è una compagnia statunitense con sede a Dallas in Texas, famosa per lo sviluppo, la produzione e la vendita di dispositivi elettronici a semiconduttori e di tecnologia informatica in genere. La TI, preceduta dalla Intel e dalla Samsung, risulta attualmente essere il terzo produttore mondiale di dispositivi elettronici e possiede centri di produzione, sviluppo e commercializzazione in 3 diversi continenti.

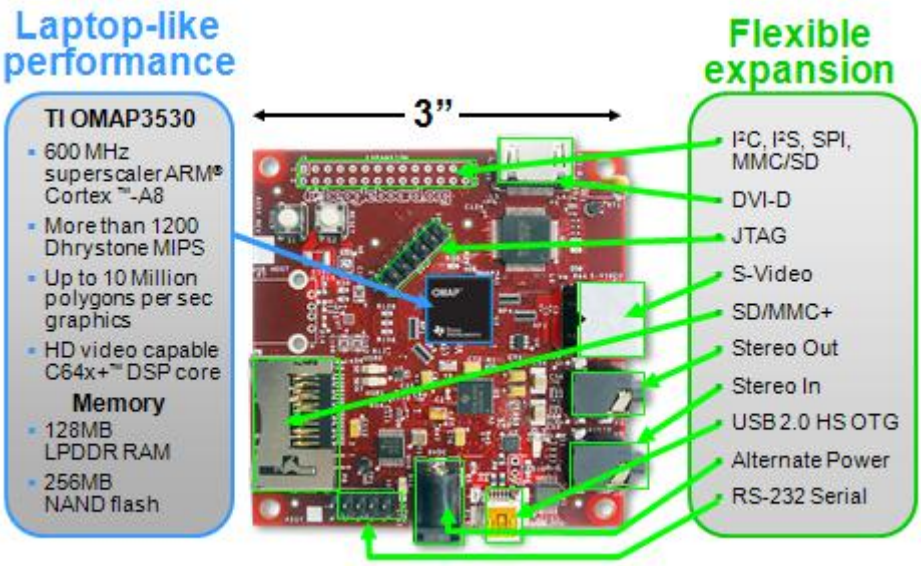



Figura 3.9: Beagle Board xM

che offre prestazioni eccezionali grazie al potente processore superscaler ARM Cortex TM-A8 da 600 MHz con coprocessore NEON TM SIMD e alle elevate performance audio-video garantite dal DSP TMS320C64x+ da 430 MHz e dall'acceleratore grafico 2D/3D OpenGLES 2.0 in grado di generare 10 milioni di poligoni al secondo. Il vantaggio principale nell'usare la Beagle Board è costituito dal carattere open dell'intero progetto: ricca documentazione hardware accessibile a tutti, una comunità open-source in forte crescita e ampia possibilità di innovare.

| | Feature | |
|---|---|---|
| **Processor** | Texas Instruments Cortex A8 1GHz processor | |
| **POP Memory** | Micron 4Gb MDDR SDRAM (512MB) 200MHz | |
| **PMIC TPS65950** | Power Regulators | |
| | Audio CODEC | |
| | Reset | |
| | USB OTG PHY | |
| **Debug Support** | 14-pin JTAG | GPIO Pins |
| | UART | 3 LEDs |
| **PCB** | 3.1" x 3.0" (78.74 x 76.2mm) | 6 layers |
| **Indicators** | Power, Power Error | 2-User Controllable |
| | PMU | USB Power |
| **HS USB 2.0 OTG Port** | Mini AB USB connector | |
| | TPS65950 I/F | |
| **USB Host Ports** | SMSC LAN9514 Ethernet HUB | |
| | 4 FS/LS/HS | Up to 500ma per Port if adequate power is supplied |
| **Ethernet** | 10/100 | From USB HUB |
| **Audio Connectors** | 3.5mm | 3.5mm |
| | L+R out | L+R Stereo In |
| **SD/MMC Connector** | MicroSD | |
| **User Interface** | 1-User defined button | Reset Button |
| **Video** | DVI-D | S-Video |
| **Camera** | Connector | Supports Leopard Imaging Module |
| **Power Connector** | USB Power | DC Power |
| **Overvoltage Protection** | Shutdown @ Over voltage | |
| **Main Expansion Connector** | Power (5V & 1.8V) | UART |
| | McBSP | McSPI |
| | I2C | GPIO |
| | MMC2 | PWM |
| **2 LCD Connectors** | Access to all of the LCD control signals plus I2C | 3.3V, 5V, 1.8V |
| **Auxiliary Audio** | 4 pin connector | McBSP2 |
| **Auxiliary Expansion** | MMC3 | GPIO,ADC,HDQ |

Figura 3.10: Tabella riassuntiva caratteristiche Beagle Board xM

### 3.3.1 Architettura ARM e tecnologia Neon

Oggi Arm sta per Advanced RISC Machine, ma alla nascita significava Acorn RISC Machine dal nome dell'azienda fondatrice la Acorn Computers Ltd.


Si tratta di una famiglia di microprocessori RISC[5] a 32-bit sviluppata da ARM Holdings e utilizzata in una moltitudine di sistemi embedded[6]. Grazie alle sue caratteristiche a basso consumo (rapportato alle prestazioni) l'architettura ARM domina il settore dei dispositivi mobili dove il risparmio energetico delle batterie è fondamentale.

Attualmente la famiglia ARM copre il 75% del mercato mondiale dei processori 32 bit per applicazioni embedded, ed è una delle più diffuse architetture a 32 bit del mondo. I processori ARM vengono utilizzati in PDA (Personal Digital Assistant più comunemente palmare), cellulari, tablet, lettori multimediali, videogiochi portatili e periferiche computer (come router, hard disk di rete ecc.).

[5]RISC, acronimo dell'inglese Reduced Instruction Set Computer, indica una filosofia di progettazione di architetture per microprocessori che predilige lo sviluppo di un'architettura semplice e lineare. Questa semplicità di progettazione permette di realizzare microprocessori in grado di eseguire il set di istruzioni in tempi minori rispetto a una classica architettura CISC. La definizione attualmente più diffusa in ambito informatico parla di architettura load-store dato che le architetture RISC permettono di accedere alla memoria unicamente tramite delle istruzioni specifiche (load e store) che provvedono a leggere, scrivere i dati nei registri del microprocessore mentre tutte le altre istruzioni manipolano i dati contenuti all'interno dei microprocessori.

[6]Con il termine sistema embedded (generalmente tradotto in italiano con sistema immerso o incorporato) si identificano genericamente tutti quei sistemi elettronici di elaborazione a microprocessore progettati appositamente per una determinata applicazione (special purpose) ovvero non riprogrammabili dall'utente per altri scopi, spesso con una piattaforma hardwaread ad hoc, integrati nel sistema che controllano ed in grado di gestirne tutte o parte delle funzionalità richieste. Contrariamente ai computer generici riprogrammabili (general purpose), un sistema embedded ha dei compiti noti già durante lo sviluppo, che eseguirà dunque grazie ad una combinazione hardware/software specificamente studiata per la tale applicazione. Grazie a ciò l'hardware può essere ridotto ai minimi termini per ridurne lo spazio occupato riducendo così anche i consumi, i tempi di elaborazione (maggiore efficienza) e il costo di fabbricazione. Inoltre l'esecuzione del software è spesso in tempo reale (real-time) per permettere un controllo deterministico dei tempi di esecuzione.

Importanti rami della famiglia ARM sono i processori Xscale e i processori OMAP[7] prodotti dalla Texas Instruments.

Nel 2005 è stata presentata una nuova famiglia processori ARM: l'ARM Cortex basati sul set di istruzioni ARMv7, formata da una serie di blocchi funzionali che possono essere collegati tra loro al fine di soddisfare le esigenze del cliente, quindi uno specifico processore Cortex non ha necessariamente tutte le funzionalità della famiglia.

Rispetto alla versione 6 la famiglia Cortex introduce le seguenti novità:

- L'unità NEON sviluppata per eseguire operazioni SIMD su vettori di 64 o 128 bit. L'unità è dotata di registri dedicati ed i vettori possono contenere numeri interi a 16 o 32 bit o numeri in virgola mobile a singola precisione a 32 bit. L'unità opera in parallelo alla pipeline principale, essa interviene solo durante il caricamento delle istruzioni da eseguire.

- L'unità in virgola mobile VFPv3 raddoppia i registri della precedente versione portandoli a 32 e introduce alcune nuove operazioni.

- Il set di istruzioni Thumb-EE è un derivato del set di istruzioni Thumb-2 ed è nato per sostituire le istruzioni Jazelle. Queste istruzioni vengono utilizzate per accelerare l'esecuzione di codice eseguito da macchine virtuali come quello richiesto dal linguaggio Java.

- TrustZone è una modalità di esecuzione sicura nata per permettere l'esecuzione di codice sicuro o per eseguire meccanismi di digital rights management (DRM)[8].

[7] Gli OMAP, acronimo di Open Multimedia Application Platform (Piattaforma per applicazioni multimediali aperte), sono dei processori RISC specificatamente progettati per applicazioni multimediali, costruiti dalla Texas Instruments sotto licenza ARM.

[8] Con Digital Rights Management (DRM), il cui significato letterale è gestione dei diritti digitali, si intendono i sistemi tecnologici mediante i quali i titolari di diritto d'autore (e dei cosiddetti diritti connessi) possono esercitare ed amministrare tali diritti nell'ambiente digitale, grazie alla possibilità di rendere protette, identificabili e tracciabili le opere di cui detengono i diritti, quindi scongiurarne la copia e altri usi non autorizzati. Si tratta di misure di sicurezza incorporate nei computer, negli apparecchi elettronici e nei file digitali.

La famiglia Cortex è suddivisa nella serie A (Application), la serie R (Real-time) e la serie M (Microcontroller).

La serie A è quella indirizzata ai computer, telefoni cellulari evoluti e più in generale le applicazioni che necessitano di potenza di calcolo e flessibilità. Questa è la serie più completa e oltre al set di istruzioni classiche ARM gestisce le istruzioni Thumb-2, Thumb-EE, include le unità VectorFloating Point (unità di calcolo in virgola mobile e NEON (unità SIMD).

La serie R è sviluppata per applicazioni realtime, il set di istruzioni Thumb-2 è presente, le istruzioni VectorFloating Point sono opzionali e la cache è configurabile.

La serie M è la serie più ridotta, implementa solamente il set di istruzioni Thumb-2, la cache non è presente e la MPU è opzionale.

**Cos'è Neon?**

Neon è un'estensione dell'instruction-set del processore, questo sta a significare che aggiunge alle normali istruzioni RISC di una tipica architettura ARM delle nuove che possono lavorare in parallelo al processore in maniera più ottimizzata e quindi veloce.

Queste nuove istruzioni si occupano di eseguire operazioni su dati dello stesso tipo che necessitano delle medesime azioni, per questo quando ci si riferisce a queste particolari architetture si sente parlare di SIMD (Single Instruction Multiple Date).

Le operazioni che ne beneficeranno saranno tutte quelle dove è possibile avere una gran quantità di dati da elaborare in maniera analoga, e queste sono ad esempio:

- giochi;
- riconoscitore vocale,
- effetti per l'interfaccia del dispositivo,

- manipolazione di segnali audio Hi-fi e surround,
- codifica e decodifica di segnali video.

Per capire meglio di come un'architettura di questo genere può apportare dei benefici dobbiamo addentrarci leggermente più a fondo nell'argomento.

Una normale CPU è un'architettura SISD (Single Instruction Single Data) questo significa che ogni istruzione vale solo per un dato, come ad esempio la somma dei numeri.

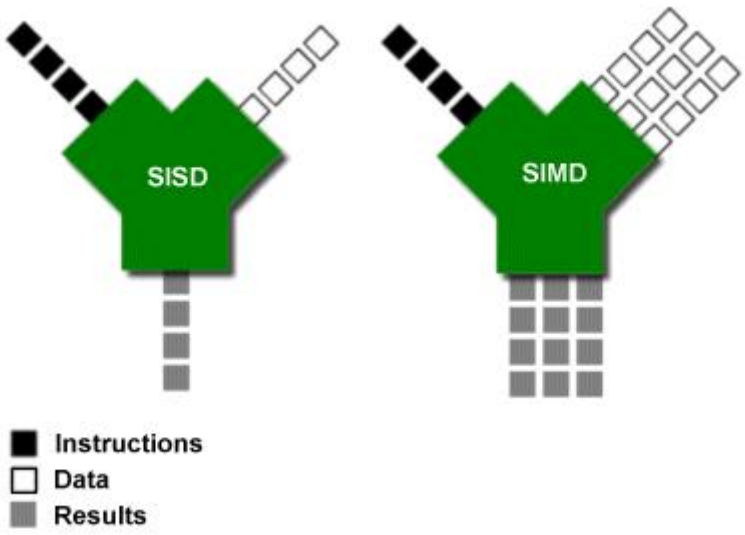



Figura 3.11: Architettura SISD e SIMD

L'architettura SIMD si applica, come detto prima, quando si hanno dati dello stesso tipo, pensiamo all'esempio molto comune della lettura dei pixel, può capitare di cambiare la luminosità allo schermo, trasformare l'immagine in una in bianco e nero ecc. tutte queste operazioni sono semplici operazioni iterate su ogni singolo pixel.

Per semplicità prendiamo l'esempio del cambio di luminosità, a ogni componente rgb di ogni pixel deve essere sommato un valore proporzionale all'incremento di luminosità, se prendiamo un normale schermo di uno smartphone attuale basta modificare il numero di pixel per tre $480 \times 320 \times 3$ che risulta essere 460800 e quindi sono necessarie 460800 somme. Da questo esempio con un'immagine di

dimensioni ridicole e una semplice somma per ogni componente possiamo capire come può essere gravoso il lavoro di una CPU ad esempio nella codifica e la decodifica di segnali video magari HD.

La tecnologia NEON è una combinazioni di istruzioni che operano su vettori a 64 e 128 bit di lunghezza per accelerare e standardizzare il trattamento e l'elaborazione di segnali multimediali; essa poggia su un set di istruzioni separate, registri indipendenti e esecuzione del codice separata, gestendo dati a 8/16/32/64 bit di tipo intero, a singola precisione e in virgola mobile.

Questi vettori sono quindi composti da dati che possono avere diversa lunghezza 8,16,32 o 64 bit come mostrato in figura:


Figura 3.12: Possibili costruzioni di array Q da 128 bit

Immaginiamoci il caso precedente, ogni valore rgb di ogni pixel è composto normalmente da 16 bit (65536 colori), se dobbiamo aumentare la luminosità possiamo costruirci dei vettori da 8 elementi ($8 \times 16 = 128$ bit) e processarli con le istruzioni NEON. In questo modo in un ciclo del processore avremmo 8 valori mentre con un normale processore dovremmo aspettare 8 cicli per avere la stessa sequenza.

Nella seguente figura è rappresentato un esempio analogo al nostro con l'unica differenza che sono stati utilizzati i registri "D" che nella terminologia ARM NEON stanno per registri a 64 bit.

Esasperando questo ragionamento se dovessimo processare dati da 8 bit (ad esempio caratteri ASCII) si potrebbero ottenere 16 risultati per ogni ciclo di processore.

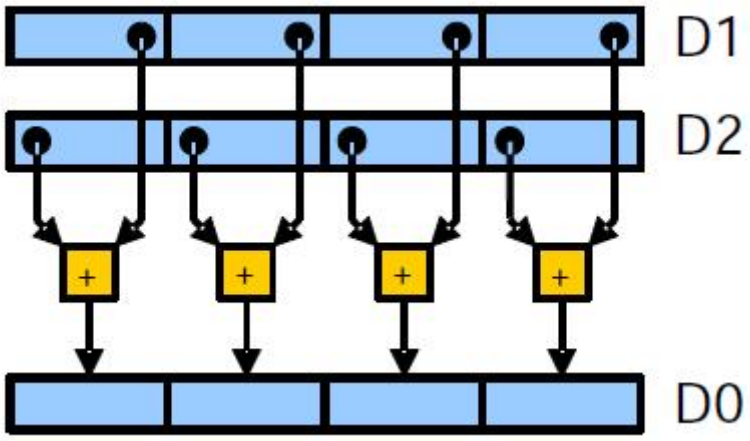



Figura 3.13: Esempio di somma multipla

Quindi per riassumere, un normale processore opera su valori singoli (scalari), un'architettura SIMD opera su vettori che sono un insieme di valori; ciò che è più interessante è che queste operazioni vengono eseguite nelle stesso tempo perché ci sono strutture parallelizzate che eseguono singole operazioni contemporaneamente.

L'architettura NEON può operare sia su dati interi che a virgola mobile, dati allineati e non allineati, caratteristica quest'ultima molto importante perché in questo modo è più facile poter costruire dei vettori anche da dati da allocazioni non allineate di memoria.

### 3.3.2 Cos'è il DSP?

DSP è un acronimo che sta per Digital Signal Processing, ed esplica compiti di elaborazione digitali di segnali, dedicato e ottimizzato per eseguire in maniera estremamente efficiente sequenze di istruzioni ricorrenti (come ad esempio somme, moltiplicazioni e traslazioni).

Nell'ambito del Signal Processing, i DSP offrono il supporto ottimizzato a ben precisi algoritmi di riferimento; in quanto tali, i DSP possono esser visti come componenti dedicati alla classe di algoritmi numerici per l'elaborazione di segnali digitali. Un DSP unisce velocità e potenza di calcolo per elaborare i dati

in tempo reale. Questa sua capacità lo rende perfetto per applicazioni che non possono tollerare ritardi nelle risposte del sistema.

Non è sempre vero che un DSP è meno potente di un microprocessore: infatti, anche se la sua frequenza di clock (≈200MHz) è notevolmente inferiore a quella dei microprocessori classici, il DSP per le applicazioni per cui è stato pensato si rivela più veloce di microprocessori come il Pentium. In generale, si dice che il DSP è un componente dedicato, application dependent, mentre il microprocessore è un general purpose.

Infatti un microprocessore general purpose può essere estremamente performante nel processo di database e nell'elaborazione di fogli elettronici, ma avere invece delle difficoltà nell'elaborare segnali audio e video in real time. Questo compito può essere svolto efficacemente da un DSP, che non ripone le sue potenzialità in una frequenza di clock elevata, ma in un'architettura diversa da quella di un microprocessore.

### Che cosa fa un DSP: caratteristiche operative

**MAC: moltiplicazione + addizione simultanea:** La maggior parte degli algoritmi per l'elaborazione dei segnali richiede un'operazione di moltiplicazione ed addizione del tipo:

$$A = ( B \times C ) + D$$

Questa operazione, multiply and accumulate, in un DSP viene effettuata in un solo ciclo di clock.

L'operazione di MAC è utile in algoritmi che prevedono il calcolo di prodotti fra vettori, come accade per i filtri digitali e per le trasformate di Fourier. Per poter disporre di questa caratteristica, i DSP presentano moltiplicatore ed accumulatore integrati nella stessa unità aritmetica, chiamata datapath.

**Accesso multiplo in memoria:** Un'altra caratteristica importante dei DSP è la capacità di compiere più accessi in memoria in uno stesso ciclo di istruzione.

Questo permette al processore di caricare simultaneamente un'istruzione ed i suoi operandi, oppure di salvare in memoria il risultato dell'operazione precedente. Tutto questo grazie alla presenza di due bus indirizzi e due bus dati: aumenta il throughput, ovvero il tasso di emissione di dati in uscita rispetto al flusso di dati in ingresso.

**Set di istruzioni specifico:** Per far sì che le operazioni aritmetiche siano calcolate alla massima velocità e che in una singola istruzione siano specificati più operandi, i DSP fanno ricorso ad un set di istruzioni specifico.

Questo set riguarda i modi di indirizzamento (addressing modes). I DSP incorporano unità di generazione degli indirizzi, address generation units, che gestiscono la formazione del nuovo indirizzo in parallelo all'esecuzione delle istruzioni aritmetiche.

**Loop control in hardware:** Gli algoritmi dei DSP contengono frequentemente istruzioni che devono essere ripetute. Invece di aggiornare un contatore o di eseguire un salto di istruzione all'inizio del loop, i DSP risolvono questo problema via hardware. Esistono infatti delle strutture hardware che effettuano il controllo dei loop: alcune permettono di ripetere per un numero fissato di volte una singola istruzione, altre ripetono un blocco di istruzioni un certo numero di volte (Single- e Multi-Instruction Hardware Loops).

**I/O interface:** Per conseguire gli obiettivi di costi contenuti ed alte performance in I/O, molti DSP incorporano una o più interfacce seriali o parallele; inoltre essi utilizzano il direct memory access (DMA). Spesso le interfacce sono direttamente collegate ai convertitori A/D e D/A.

I DSP servono all'industria di consumer electronics; queste sono alcune delle applicazioni: audio production, audio consumer, wireless communications, motor control, radar, sonar, biomedical, voice over internet, satellite communica-

tions, digital cell phones, digital cameras, digital TV, speech processing, seismic processing, instruments, music effects, music synthesis.

I maggiori produttori mondiali di DSP sono Texas Instruments, Analog Devices, Motorola.

ST Microelectronics, presente in Italia, produce DSP per un mercato advanced ma in un certo qual modo di nicchia.

### Caratteristiche hardware

**Architettura classica di un DSP:** Tutti i processori digitali sono costituiti da alcuni moduli fondamentali:

- Unità di elaborazione per operazioni aritmetiche e logiche (ALU).
- Memorie per conservare i dati e le istruzioni dei programmi.
- Bus per trasferire in modo efficiente i dati e le istruzioni da una memoria ad una ALU e viceversa.

Le usuali operazioni compiute da un processore sono quelle di prelevare dati ed istruzioni dalla memoria, elaborarli ed infine ritornare in memoria per salvarvi il risultato dell'elaborazione.

L'equivalente di una ALU in un DSP è il DATAPATH, che può essere definito come il percorso logico aritmetico compiuto dai dati.

**Overflow:** Operando in fixed point, ad ogni operazione bisogna che il programmatore consideri l'eventualità che si verifichi il caso di overflow o di underflow. Il tipico strumento per rimediare a questa eventualità è un accumulatore esteso, ma si deve avere anche dell'hardware dedicato: vi devono essere unità che gestiscono tutto in hardware.

Tipicamente i DSP non possono permettersi di segnalare overflow senza curarsene e prendere degli accorgimenti, perché in caso di applicazioni critiche non si può interrompere l'esecuzione di un'elaborazione. Nei DSP se un numero

va in overflow l'Hardware Saturation Overflow vi sostituisce il massimo numero rappresentabile.

### Architettura di memoria

Tradizionalmente nei microprocessori è utilizzata un'architettura di tipo Von Neumann. Questo tipo di architettura contiene una singola memoria ed un singolo bus per il trasferimento dei dati da e verso la CPU. Lo schema di Von Neumann è abbastanza soddisfacente se bisogna eseguire le operazioni richieste in modo seriale: questo è il caso più comune nei microprocessori.

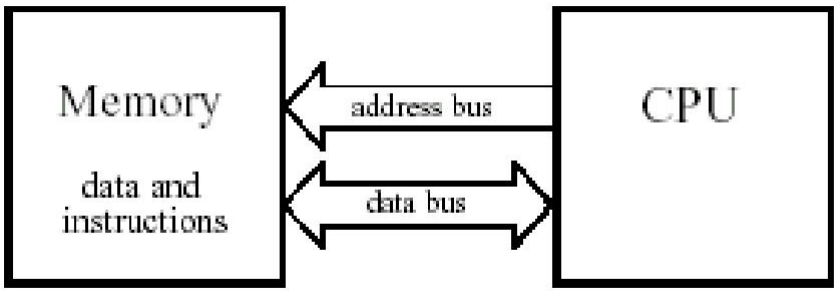



Figura 3.14: Architettura di Von Neumann

Quando però si ha bisogno di incrementare la velocità dei processi, si deve far ricorso ad un altro tipo di architettura, denominata Harvard. Un'architettura di tipo Harvard dispone sia di memorie che di bus separati per dati ed istruzioni. Un'istruzione di programma ed un dato possono essere caricati simultaneamente, incrementando le prestazioni rispetto alla configurazione a singolo bus del caso Von Neumann. La maggior parte dei DSP odierni implementa questa architettura dual bus:

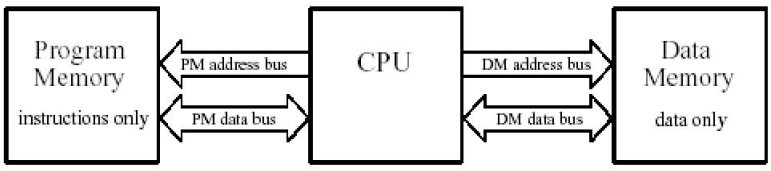



Figura 3.15: Architettura di Harvard

Il nome di Super Harvard Architecture è associato ad un ulteriore miglioramento in termini di throughput dell'architettura Harvard. Queste maggiori prestazioni si ottengono tipicamente aggiungendo una cache per le istruzioni nella CPU, il che virtualmente rende disponibili al core del DSP tre bus; naturalmente il termine Super Harvard si può applicare anche al caso di tre effettivi bus disponibili.

Nel processore SHARC di Analog Devices le prestazione dell'architettura sono ulteriormente potenziate inserendo un controller di I/O connesso alla data memory:

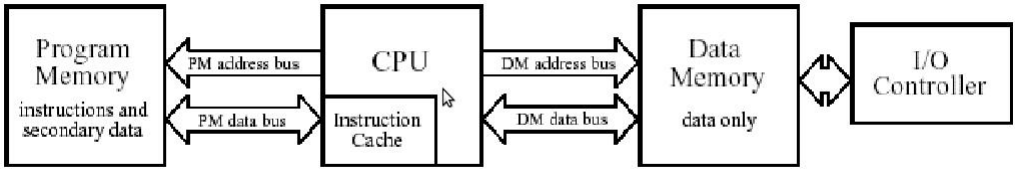



Figura 3.16: Architettura SHARC

Nell'architettura Harvard l'uso di due memorie, una per i dati ed una per le istruzioni, determina un incremento dei pin del processore. Si deve infatti indirizzare uno spazio di memoria maggiore. Per risparmiare pin a bordo di un DSP può essere implementata una memoria, mentre il collegamento alla memoria esterna è demandato ad un unico bus.

**Multiple acces memory:** La struttura Harvard permette di compiere accessi multipli in memoria. Questi accessi possono essere sequenziali: in un ciclo di clock si compiono due fetch. Se il DSP-core accede simultaneamente a due banchi di memoria usando due bus indipendenti si ottengono quattro accessi in memoria per ciclo di clock.

Un'altra tecnica per implementare una struttura Harvard, al fine di incrementare il numero di accessi in memoria, è quella di usare memorie multiporte. Una memoria multiporte dispone di più set di indirizzi e connessioni indipendenti,

in modo da permettere un accesso parallelo in memoria. Il tipo più comune di memoria multiporte è la dual-port memory, che permette due accessi simultanei.

**Acceleratore hardware:** La memoria è composta da registri e da cache: le cache di programma sono memorie veloci che contengono le istruzioni più probabili. Esse vengono utilizzate per diminuire il numero di accessi in memoria.

I DSP supportano a livello hardware diversi tipi di cache di programma:

- Repeat buffer
  - single instruction;
  - multiple instruction;

  Il fatto che le istruzioni (for, ...) siano implementate in hardware nella cache di programma fa in modo che si liberi il program bus. Si ha un vantaggio architetturale: è possibile mettere dei dati nella program memory (usarla come data memory).

  Un limite del repeat buffer è che non ci possono essere jump, goto, chiamate a subroutine.

- Single-sector cache: un blocco di memoria che contiene un pezzo di programma (qualunque pezzo, non per forza un loop: ovviamente un pezzo di programma autoconsistente).

- Multiple-sector cache: a differenza della single-sector cache, questa cache può memorizzare due o più pezzi indipendenti di programma. Quando bisogna caricare nella cache un nuovo settore, l'algoritmo utilizzato per scegliere quale settore sostituire è l'LRU (least-recent-used).

# Capitolo 4

# Il Progetto

Si passa ora alla descrizione dei programmi scritti nel linguaggio C++ nell'ambiente di sviluppo Eclipse e alla visualizzazione dei risultati ottenuti.

Su un'immagine campione caricata da me di dimensione 640x480 viene eseguita la DFT sfruttando prima i mezzi messi a disposizione dalla sola libreria OpenCV e poi con quelli della libreria FFTW in modo da poter utilizzare l'unità Neon presente nella Beagle Board.

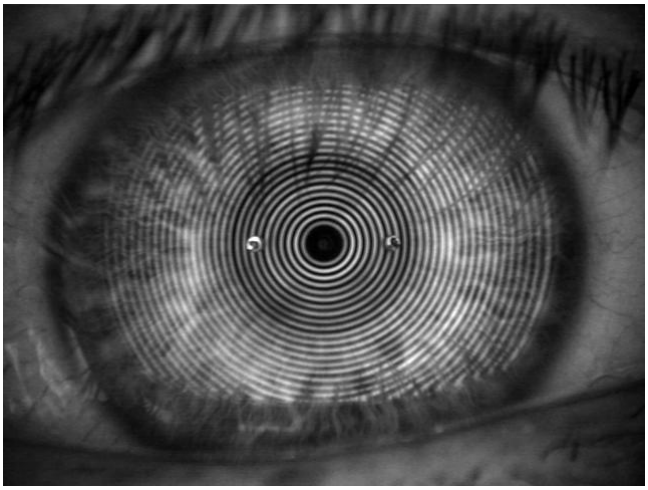


Figura 4.1: Esempio di immagine catturata dal topografo

## 4.1 Trasformata di Fourier utilizzando la libreria OpenCV

Nel primo programma si è scritto il codice utilizzando unicamente le funzioni messe a disposizione da OpenCV e si è analizzato se il target prestabilito veniva già raggiunto. OpenCV nel calcolo della trasformata di Fourier mette a disposizione la funzione cvDFT() così composta:

```
1 void cvDFT(   const CvArr* src,
                CvArr* dst,
3               int   flags,
                int nonzero_rows = 0);
```

Attraverso la funzione cvDFT() si può calcolare la trasformata discreta di Fourier per N numeri complessi $x_0, ...., x_{N-1}$, di array a una e due dimensioni. Gli array monodimensionale vengono così definiti:

$$f_k = \sum_{n=0}^{N-1} x_n exp(-\frac{2\pi i}{N} kn), \quad k = 0, ..., N-1$$

e analogamente quelli bidimensionali:

$$f_{k_x k_y} = \sum_{n_x=0}^{N_x-1} \sum_{n_y=0}^{N_y-1} x_{n_x n_y} exp(-\frac{2\pi i}{N_x} k_x n_x) exp(-\frac{2\pi i}{N_y} k_y n_y)$$

In generale, ci si può aspettare che la computazione di N differenti termini $f_k$ possa richiedere $O(N^2)$ operazioni. Di fatto, ci sono diversi algoritmi chiamati *Fast Fourier Transform* (FFT) capaci di computare questi valori in un tempo di $O(NlogN)$, e la funzione cvDFT() di OpenCV implementa uno di questi. Gli array di input e output devono essere di tipo floating-point e a singolo o doppio canale. Nel caso di un singolo canale, si assume l'ingresso costituito da numeri reali e l'uscita viene memorizzata in uno "speciale formato salva spazio" (ereditata dalla libreria IPL come la struttura IplImage). Se la sorgente è una matrice a due canali o un'immagine, allora i due canali saranno interpretati come

i componenti reali e immaginari dei dati di ingresso. In questo caso, non ci sarà uno speciale formato di salvataggio per i risultati, e dello spazio sarà sprecato con un gran numero di 0 sia nel array di ingresso che di uscita[1].

La rappresentazione di un array monodimensionale è la seguente:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $\mathrm{Re}\,Y_0$ | $\mathrm{Re}\,Y_1$ | $\mathrm{Im}\,Y_1$ | $\mathrm{Re}\,Y_2$ | $\mathrm{Im}\,Y_2$ | $\cdots$ | $\mathrm{Re}\,Y_{(N/2-1)}$ | $\mathrm{Im}\,Y_{(N/2-1)}$ | $\mathrm{Re}\,Y_{(N/2)}$ |

Per un array bidimensionale:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $\mathrm{Re}\,Y_{00}$ | $\mathrm{Re}\,Y_{01}$ | $\mathrm{Im}\,Y_{01}$ | $\mathrm{Re}\,Y_{02}$ | $\mathrm{Im}\,Y_{02}$ | $\cdots$ | $\mathrm{Re}\,Y_{0(Nx/2-1)}$ | $\mathrm{Im}\,Y_{0(Nx/2-1)}$ | $\mathrm{Re}\,Y_{0(Nx/2)}$ |
| $\mathrm{Re}\,Y_{10}$ | $\mathrm{Re}\,Y_{11}$ | $\mathrm{Im}\,Y_{11}$ | $\mathrm{Re}\,Y_{12}$ | $\mathrm{Im}\,Y_{12}$ | $\cdots$ | $\mathrm{Re}\,Y_{1(Nx/2-1)}$ | $\mathrm{Im}\,Y_{1(Nx/2-1)}$ | $\mathrm{Re}\,Y_{1(Nx/2)}$ |
| $\mathrm{Re}\,Y_{20}$ | $\mathrm{Re}\,Y_{21}$ | $\mathrm{Im}\,Y_{21}$ | $\mathrm{Re}\,Y_{22}$ | $\mathrm{Im}\,Y_{22}$ | $\cdots$ | $\mathrm{Re}\,Y_{2(Nx/2-1)}$ | $\mathrm{Im}\,Y_{2(Nx/2-1)}$ | $\mathrm{Re}\,Y_{2(Nx/2)}$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $\mathrm{Re}\,Y_{(Ny/2-1)0}$ | $\mathrm{Re}\,Y_{(Ny-3)1}$ | $\mathrm{Im}\,Y_{(Ny-3)1}$ | $\mathrm{Re}\,Y_{(Ny-3)2}$ | $\mathrm{Im}\,Y_{(Ny-3)2}$ | $\cdots$ | $\mathrm{Re}\,Y_{(Ny-3)(Nx/2-1)}$ | $\mathrm{Im}\,Y_{(Ny-3)(Nx/2-1)}$ | $\mathrm{Re}\,Y_{(Ny-3)(Nx/2)}$ |
| $\mathrm{Im}\,Y_{(Ny/2-1)0}$ | $\mathrm{Re}\,Y_{(Ny-2)1}$ | $\mathrm{Im}\,Y_{(Ny-2)1}$ | $\mathrm{Re}\,Y_{(Ny-2)2}$ | $\mathrm{Im}\,Y_{(Ny-2)2}$ | $\cdots$ | $\mathrm{Re}\,Y_{(Ny-2)(Nx/2-1)}$ | $\mathrm{Im}\,Y_{(Ny-2)(Nx/2-1)}$ | $\mathrm{Re}\,Y_{(Ny-2)(Nx/2)}$ |
| $\mathrm{Re}\,Y_{(Ny/2)0}$ | $\mathrm{Re}\,Y_{(Ny-1)1}$ | $\mathrm{Im}\,Y_{(Ny-1)1}$ | $\mathrm{Re}\,Y_{(Ny-1)2}$ | $\mathrm{Im}\,Y_{(Ny-1)2}$ | $\cdots$ | $\mathrm{Re}\,Y_{(Ny-1)(Nx/2-1)}$ | $\mathrm{Im}\,Y_{(Ny-1)(Nx/2-1)}$ | $\mathrm{Re}\,Y_{(Ny-1)(Nx/2)}$ |

Il terzo argomento, *flags*, indica l'operazione che si vuole fare, nel caso della trasformata diretta di Fourier il comando è *CV_DXT_FORWARD*.

Per comprendere l'ultimo argomento, *nonzero_rows*, bisogna fare un breve inciso. Nella maggior parte degli algoritmi DFT, le dimensioni più adatte sono potenze di 2 ($2^n$ per qualsiasi n intero). Nel caso dell'algoritmo utilizzato da OpenCV, le prestazioni saranno migliori con vettori di lunghezza, o array di dimensioni, $2^p\ 3^q\ 5^r$, per qualsiasi numero intero p, q e r. La procedura più comune è quella di creare un array leggermente più grande del necessario (a tal fine, è presente la funzione *cvGetOptimalDFTSize()*, che acquisisce la dimensione del

[1] Quando sarà utilizzato questo metodo, nella rappresentazione a due canali, bisogna essere sicuri di assegnare il valore 0 a tutti i componenti della parte immaginaria. Una semplice via per farlo è quella di creare una matrice riempita con zeri usando la funzione cvZero() per la parte immaginaria e attraverso la chiamata di cvMerge() con la matrice dei numeri reali, formare un array di numeri complessi temporaneo.

vettore e restituisce la dimensione appropriata pari o maggiore a quella iniziale) e successivamente si copierà la matrice in un array più grande "imbottito" di zeri utilizzando *cvGetSubRect()*. Quindi nonostante la necessità di aggiungere righe contenenti zeri per far raggiungere alla matrice le dimensioni più adatte all'operazione di trasformazione, attraverso il parametro *nonzero_rows* è possibile indicare il numero di righe che può essere ignorato; ciò consentirà un risparmio nel tempo di calcolo.

### 4.1.1 Spiegazione del programma

Come detto la trasformata di Fourier separa le componenti seno e coseno di un'immagine, in altri termini, trasforma l'immagine dal dominio spaziale a quello delle frequenze. L'idea di base è che ogni funzione può essere esattamente approssimata con la somma infinita di seni e coseni. Questo è ciò che fa la trasformata di Fourier e che per un'immagine a due dimensioni può essere matematicamente scritto in questo modo:

$$F(k,l) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j) e^{-i2\pi \left( \frac{ki}{N} + \frac{li}{N} \right)}$$

$$e^{ix} = cos\ x\ +\ i\ sin\ x$$

Dove f è il valore dell'immagine nel dominio spaziale e F nel dominio delle frequenze. Il risultato della trasformazione è un numero complesso, e la visualizzazione può avvenire tramite un'immagine reale e una complessa o attraverso la *"magnitude"* . In questo programma viene inoltre mostrato come trovare e visualizzare il *"magnitude"* dell'immagine, non necessario al momento ma utile per il futuro, poiché la sua immagine contiene tutte le informazioni sulla struttura geometrica.

Vediamo nel dettaglio i passaggi principali del programma; l'intero codice è presente all'interno dell'Appendice A.

**Adattamento dell'immagine alla dimensione ottimale.** Come detto le prestazioni della DFT dipendono dalla grandezza dell'immagine, a tale scopo si usano i seguenti comandi:

```
dft_M = cvGetOptimalDFTSize( img->height - 1);
dft_N = cvGetOptimalDFTSize( img->width - 1);
dft_A = cvCreateMat( dft_M, dft_N, CV_64FC2);
cvGetSubRect(dft_A, &tmp, cvRect(0,0,img->width, img->height));
cvCopy(complexInput, &tmp, NULL);
```

**Creazione della matrice complessa.** Si crea una matrice reale copiando e scalando i valori dell'immagine di origine e una matrice immaginaria che sarà riempita di zeri. Le due matrici andranno a comporre un unico array di numeri complessi a due canali:

```
realInput = cvCreateImage (cvGetSize(img), IPL_DEPTH_64F, 1);
imaginaryInput = cvCreateImage( cvGetSize(img), IPL_DEPTH_64F, 1);
complexInput = cvCreateImage( cvGetSize(img), IPL_DEPTH_64F, 2);
cvScale(img, realInput, 1.0 , 0.0);
cvZero(imaginaryInput);
cvMerge(realInput, imaginaryInput, NULL, NULL, complexInput);
```

**Eseguo la Trasformata Discreta di Fourier e visualizzo il tempo di calcolo.** Si può immettere lo stesso parametro in input e output:

```
        gettimeofday(&start, NULL);
        cvDFT( dft_A, dft_A, CV_DXT_FORWARD, complexInput->height);
        gettimeofday(&end, NULL);
        seconds = end.tv_sec - start.tv_sec;
        useconds = end.tv_usec - start.tv_usec;
        mtime = ((seconds) * 1000 + useconds/1000.0) + 0.5;
        printf("Tempo impiegato: %ld [msec]\n", mtime);
```

**Calcolo il *"magnitude"*.** Il calcolo del *"magnitude"* avviene attraverso la formula:

$$M = \sqrt{Re^2 + Im^2}$$

Divido quindi il risultato della DFT in parte reale e immaginaria, per poi calcolare il *"magnitude"*:

```
cvSplit (dft_A, image_Re, image_Im, 0, 0);
cvPow( image_Re, image_Re, 2.0);
cvPow( image_Im, image_Im, 2.0);
cvAdd( image_Re, image_Im, image_Re, NULL);
cvPow( image_Re, image_Re, 0.5);
```

**Converto in scala logaritmica.** Poiché risulta che la gamma dinamica di valori dei coefficienti di Fourier è troppo ampia per essere visualizzata sullo schermo, per utilizzare la scala di grigi per la visualizzazione possiamo trasformare la nostra scala lineare in una logaritmica:

```
cvAddS( image_Re, cvScalarAll(1.0), image_Re, NULL);
cvLog(image_Re, image_Re);
```

**Riorganizzazione dei quadranti.** Dato che inizialmente l'immagine di origine è stata ingrandita per renderla di dimensioni multiple dei valori 2,3,5 è ora arrivato il momento di farla tornare alle dimensioni iniziali e riorganizzare i quadranti in modo tale che l'orine (0,0) corrisponda al centro dell'immagine:

```
cvShiftDFT(image_Re, image_Re);
```

**Normalizzo.** Anche questo passaggio è puramente per scopi di visualizzazione. Poiché siamo ancora fuori dal range di visualizzazione, è necessario normalizzare i nostri valori:

```
cvMinMaxLoc(image_Re, &m, &M, NULL, NULL, NULL);
cvScale(image_Re,image_Re, 1.0/(M−m),1.0∗(−m)/(M−m));
```

### 4.1.2 Visualizzazione dei risultati

Una volta compilato e verificato il corretto funzionamento, si può eseguire il programma nella Beagle Board e visualizzare nel monitor la *"magnitude"* e il tempo impiegato per il calcolo della DFT.

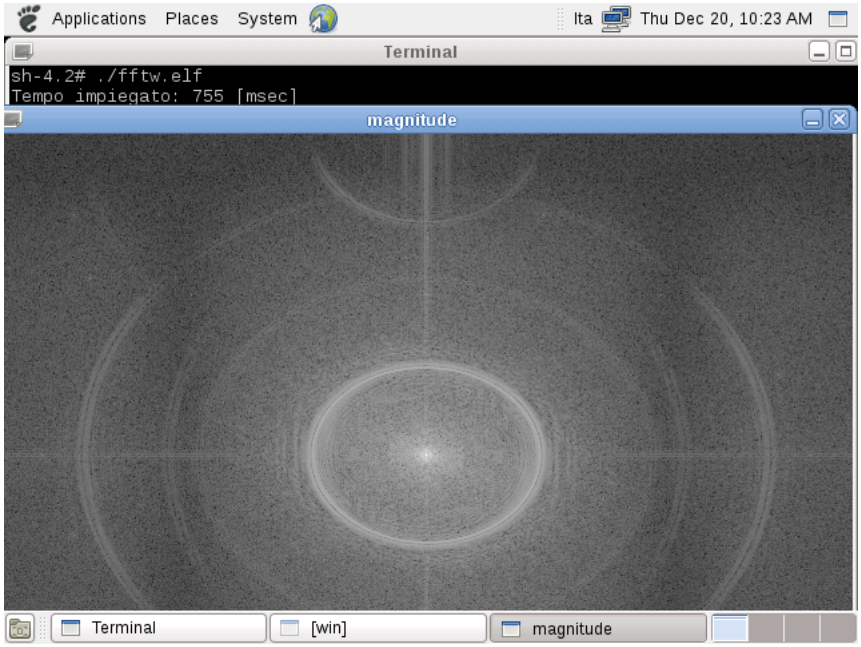



Figura 4.2: Visualizzazione del tempo di calcolo della DFT e della *"magnitude"*

Come è evidente 755 msec sono eccessivi per il calcolo della trasformata di Fourier su un'unica immagine e del resto, come ci si aspettava, l'utilizzo della sola libreria OpenCV che mette a disposizione la funzione cvDFT() per un calcolo così complesso non è sufficiente a raggiungere il target prestabilito di 25 frame al secondo.

## 4.2 Trasformata di Fourier utilizzando la libreria FFTW

Per sfruttare l'architettura Neon e le sue potenzialità nel calcola della trasformata di Fourier, importo la libreria FFTW.

La Fast Fourier Transform in the West (FFTW), è una libreria per la computazione della trasformata discreta di Fourier, ed è stata sviluppata da Matteo

Frigo e Steven G. Johnson nel Massachusetts Institute of Technology. FFTW è conosciuta come l'implementazione software gratuita più veloce per il calcolo della FFT. È in grado di calcolare le trasformazioni di array reali e immaginari di dimensione arbitraria in un tempo di $O(NlogN)$.

Il codice di base per utilizzare la FFTW nel calcolo della DFT di un array a due dimensioni è il seguente:

```
#include <fftw3.h>
...
{

        fftw_complex *in, *out;
        fftw_plan p;
        ...
        // N righe e M colonne
        in = (fftw_complex*) fftw_malloc (sizeof(fftw_complex)*(N*M));
        out = (fftw_complex*) fftw_malloc (sizeof(fftw_complex)*(N*M));
        p = fftw_plan_dft_2d(N,M,in,out,FFTW_FORWARD,FFTW_ESTIMATE);
        ...
        fftw_execute(p); //si può ripetere il comando finchè necessario
        ...
        fftw_destroy_plan(p);
        fftw_free(in);
        fftw_free(out);
}
```

Nel codice come prima azione vengono creati gli array di ingresso (in) e uscita (out). Si possono allocare come si preferisce, ma per sfruttare l'architettura SIMD è necessario l'utilizzo del comando *fftw_malloc* che garantisce un corretto allineamento dei dati trasmessi.

I dati vengono inseriti in un array di tipo *fftw_complex*, che è per default *double*[2] e composto da una parte reale ($in[i][0]$) e una immaginaria ($in[i][1]$)

che formano un numero complesso.

Il passo successivo consiste nel creare un *piano*, esso è un oggetto che contiene tutti i dati di cui ha bisogno la FFTW per poter calcolare la FFT. Nella funzione che crea il piano N e M sono le dimensioni della trasformata, i successivi due argomenti sono i puntatori alle matrici di ingresso e di uscita.

Il terzo è il *sign* e può essere o *FFTW_FORWARD* (-1) o *FFTW_BACKWARD* (+1) che indica se si tratta di trasformata diretta o inversa e tecnicamente esso indica il segno dell'esponente. Infine, *flags*, usualmente o *FFTW_MEASURE* o *FFTW_ESTIMATE*. *FFTW_MEASURE* esegue e misura il tempo di esecuzione di FFT diverse, al fine di trovare il modo migliore per calcolare la trasformata. Questo processo richiede del tempo (di solito pochi secondi), a differenza di *FFTW_ESTIMATE* che costruisce unicamente un piano che probabilmente è quello ottimale.

Ora che il piano è stato creato, lo si può utilizzare tutte le volte che si vuole attraverso il comando *fftw_execute(p)*. Vanno poi liberate le memorie attraverso i comandi *fftw_destroy_plan(p)* per il piano e *fftw_free()* per gli array.

### 4.2.1 Utilizzo della simmetria Hermitiana

In molti casi pratici, i dati di input sono costituiti unicamente da numeri reali, e in questo caso la DFT soddisfa la simmetria Hermitiana. È possibile sfruttare queste circostanze per ottenere circa un fattore due di miglioramento riguardanti velocità e utilizzo della memoria.

In cambio dei vantaggi in termini di tempo e spazio, l'utente dovrà sacrificare alcune delle semplicità di utilizzo della trasformazione complessa di FFTW. Per prima cosa, gli array di ingresso e uscita sono di diverse dimensioni e tipo: l'input è formato da $n$ numeri reali, mentre l'output da $n/2+1$ numeri complessi. In secondo luogo, la trasformata inversa (da complesso a reale) ha l'effetto collaterale di sovrascrivere gli array di input per default. Nessuno di questi inconvenienti costituisce un serio problema, ma è bene ed importante esserne consapevoli.

La sintassi per la creazione del piano a due dimensioni è la seguente:

```
fftw_plan fftw_plan_dft_r2c_2d(int n0, int n1, double *in, fftw_complex *out,
unsigned flags);
fftw_plan fftw_plan_dft_c2r_2d(int n0, int n1, fftw_complex *in, double *out,
unsigned flags);
```

per la trasformata da reale a complesso si usa $r2c$ e per l'antitrasformata da complesso a reale $c2r$. Non è necessario l'argomento *sign* poiché $r2c$ corrisponde sempre a *FFTW_FORWARD* e $c2r$ a *FFTW_BACWARD*.

Qui n è la dimensione "logica" della DFT, non necessariamente quella fisica della matrice. In particolare l'array reale (double) ha n elementi, mentre l'array complesso (fftw_complex) ha $n/2 + 1$ elementi (la divisione viene arrotondata per difetto).

Supponiamo che i dati reali abbiano dimensione $n_0 \times n_1 \times ... \times n_{d-1}$, e dopo la trasformazione $r2c$, l'output è un array di dimensione $n_0 \times n_1 \times ... \times (n_{d-1}/2+1)$ con valori di tipo *fftw_complex*, di poco maggiore di metà della grandezza dell'array di output nella DFT complessa vista precedentemente.

Nel caso di trasformazioni "in-place" poiché l'array complesso è più grande di quello reale è necessaria un padding di quest'ultimo con valori extra (due valori se l'ultima dimensione è pari e uno se è dispari). Ovvero la dimensione dell'array reale deve poter contenere fisicamente $2 \times (n_{d-1}/2 + 1)$ valori di tipo double (sufficienti a contenere i dati complessi).

Per esempio, consideriamo la trasformata di un array reale di dimensioni da $n0$ a $n1$. L'output della r2c è un array complesso a due dimensioni di grandezza da $n0$ a $n1/2 + 1$, le colonne y della matrice sono state tagliate quasi a metà e questo perché l'output di tipo fftw_complex è il doppio dell'input di tipo double. Quindi se vogliamo eseguire trasformazioni "in-place", è necessario completare la matrice di input in modo che che abbia dimensioni da $n0$ a $2 \times (n1/2 + 1)$. Se n1 è pari allora ci sono due elementi di imbottitura alla fine di ogni riga che non devono essere inizializzati in quanto vengono utilizzati unicamente per l'output.

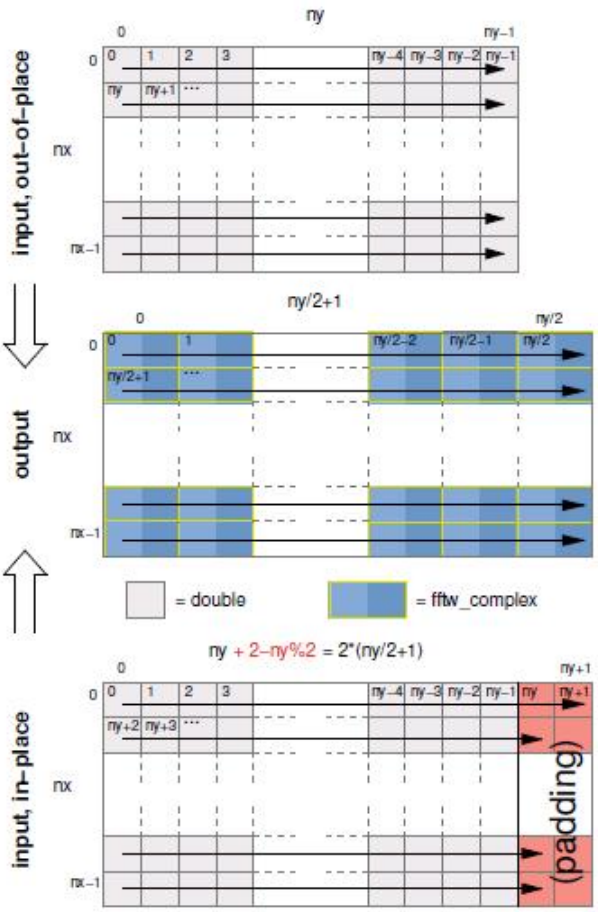



Figura 4.3: Trasformazione Real-to-Complex

### 4.2.2 Visualizzazione dei risultati

Di seguito verranno illustrati i tempi necessari per eseguire la trasformazione utilizzando la libreria FFTW prima senza sfruttare e poi sfruttando la simmetria Hermitiana.

Nel primo caso il tempo necessario ad eseguire una singola trasformazione è di 325 msec evidentemente ancora troppo alto per il target richiesto. Con l'utilizzo della simmetria Hermitiana le cose migliorano:

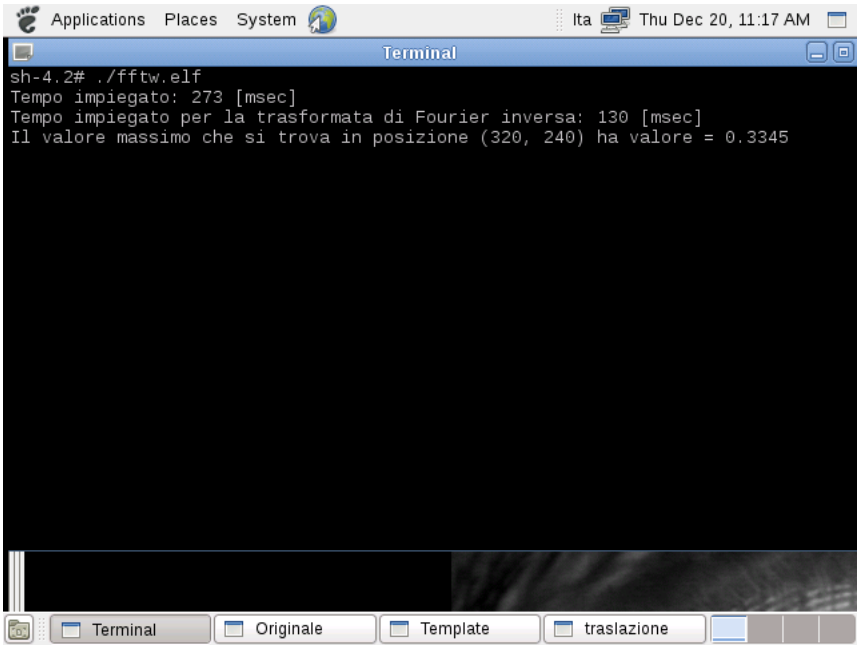



Figura 4.4: Visualizzazione dei tempi di calcolo della trasformata $r2c$ e dell'antitrasformata $c2r$

273 msec è un ottimo tempo ed è evidente il miglioramento che l'utilizzo dell'unità Neon ha portato rispetto a prima, ma non è ancora sufficiente per poter stare all'interno dei 25 frame al secondo. Non bisogna disperarsi però, perché l'immagine può essere sotto-campionata ed eseguire la DFT su una figura che ha dimensioni 320x240. Come spiegato precedentemente la grandezza dell'immagine ha una grande importanza sui tempi di calcolo della trasformata come evidenziano i nuovi risultati ottenuti:

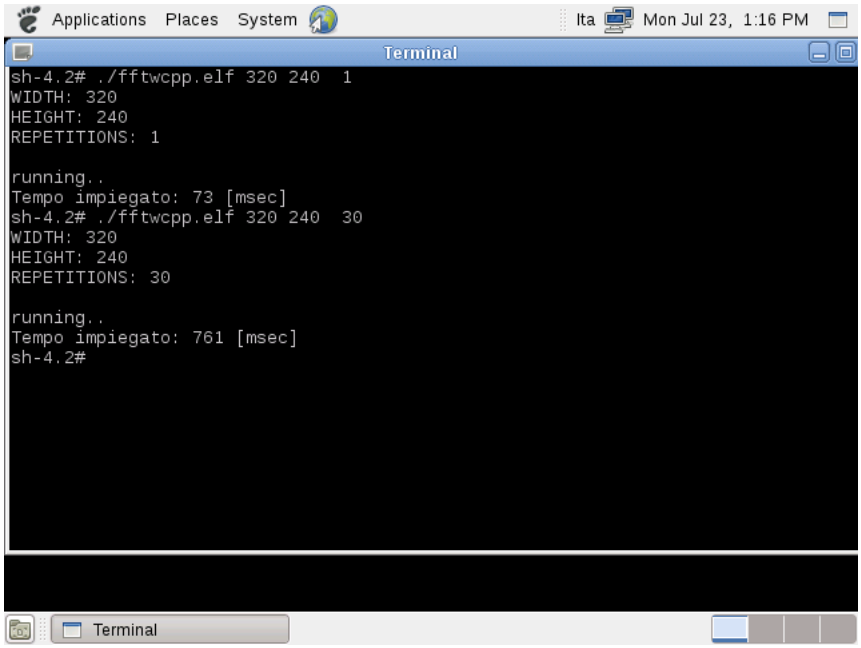



Figura 4.5: Visualizzazione dei tempi di calcolo della trasformata $r2c$ su un'immagine di dimensioni 320x240

da essi si può notare come con le nuove dimensioni della nostra immagine si è ben al di sotto del nostro target; 73 msec sulla singola immagine e 761 msec per eseguire 30 trasformazioni.

# Conclusioni

Questo lavoro di tesi rappresenta un punto di inizio per l'integrazione di alcune funzioni all'interno dell'apparecchiatura oftalmica Nidek Magellan Mapper$^{TM}$ in modo da aumentare la sua portabilità e versatilità di utilizzo, diminuendo anche i costi. L'obbiettivo prefissato era quello di sperimentare se attraverso strumenti open source il tempo di acquisizione e di elaborazione dei frame delle immagini catturate rientrava all'interno dei 25 frame al secondo.

Seguendo la linea della libera distribuzione si è proceduto con l'installazione della piattaforma di sviluppo Eclipse, delle librerie per la computer vision e calcolo della trasformata di Fourier OpenCV e FFTW. Si sono svolte varie prove attraverso la scrittura di algoritmi nel linguaggio di programmazione C++ per l'acquisizione e l'elaborazione delle immagini; inizialmente con il solo utilizzo delle librerie OpenCV e successivamente adoperando anche le FFTW in grado di sfruttare l'architettura Neon presente all'interno della Beagle Board e in generale in tutti i processori ARM Cortex. Tale tecnologia è sviluppata per eseguire operazioni SIMD lavorando su dati dello stesso tipo, questo significa che i calcoli vengono svolti nello stesso momento perché ci sono strutture parallelizzate che eseguono singole operazioni contemporaneamente.

Neon consente inoltre di sfruttare il fatto che i miei dati in input sono costituiti unicamente da numeri reali, quindi la DFT soddisfa la simmetria hermitiana e si ottiene in questo modo circa un fattore due di miglioramento riguardanti velocità e utilizzo della memoria.

Infine il DSP non è stato utilizzato sia a causa dei tempi ristretti per svolgere

la tesi sia perché il supporto in ambiente open source non si presentava stabile. L'uso di questa tecnologia necessita infatti di software costituiti da tool originali provocando problemi di integrazione con Linux e la libreria FFTW. Il fatto che l'azienda produttrice, la Texas Instruments, abbia chiuso all'inizio del 2013 la divisione OMAP costituita dalle famiglie di processori ARM+DSP rivelatesi poco appetibili agli sviluppatori a causa delle elevate problematiche di integrazione nei sistemi Linux delle funzionalità DSP, dimostra le sue difficoltà di utilizzo.

I programmi compilati sono stati eseguiti nella Beagle Board e si è proceduto con la visualizzazione dei risultati. Sfruttando tutte le potenzialità e le proprietà degli strumenti open source a disposizione si è riusciti a raggiungere l'obbiettivo prefissato arrivando ad impiegare solo 73 msec per il calcolo della trasformata di Fourire sulla singola immagine e 761 msec per eseguire 30 trasformazioni.

Come detto, quindi, questo lavoro costituisce un valido punto di partenza per trasferire in un sistema embedded alcune delle operazioni che, al momento, il topografo corneale Magellan Mapper$^{TM}$ svolge con l'ausilio di un PC connesso tramite USB.

# Appendice A

# Codice per calcolare la trasformata di Fourier utilizzando la libreria OpenCV

```
#include <SDL.h>
#include <opencv/highgui.h>
#include <opencv/cv.h>
#include <time.h>
#include <stdio.h>
#include <fftw3.h>
#include <core/types_c.h>
#include <core/core_c.h>
#include <sys/time.h>

void cvShiftDFT(CvArr * src_arr, CvArr * dst_arr )
        {
            CvMat * tmp;
            CvMat q1stub, q2stub;
            CvMat q3stub, q4stub;
            CvMat d1stub, d2stub;
```

```
CvMat d3stub, d4stub;
CvMat * q1, * q2, * q3, * q4;
CvMat * d1, * d2, * d3, * d4;

CvSize size = cvGetSize(src_arr);
CvSize dst_size = cvGetSize(dst_arr);
int cx, cy;
    if(dst_size.width != size.width ||
   dst_size.height != size.height){
    cvError( CV_StsUnmatchedSizes,
    "cvShiftDFT", "Source and Destination arrays must have
    equal sizes", __FILE__, __LINE__ );
}//if
if(src_arr==dst_arr){
    tmp = cvCreateMat(size.height/2, size.width/2,
    cvGetElemType(src_arr));
}//if
cx = size.width/2;
cy = size.height/2; // image center
q1 = cvGetSubRect( src_arr, &q1stub, cvRect(0,0,cx, cy) );
q2 = cvGetSubRect( src_arr, &q2stub, cvRect(cx,0,cx,cy) );
q3 = cvGetSubRect( src_arr, &q3stub, cvRect(cx,cy,cx,cy) );
q4 = cvGetSubRect( src_arr, &q4stub, cvRect(0,cy,cx,cy) );
d1 = cvGetSubRect( src_arr, &d1stub, cvRect(0,0,cx,cy) );
d2 = cvGetSubRect( src_arr, &d2stub, cvRect(cx,0,cx,cy) );
d3 = cvGetSubRect( src_arr, &d3stub, cvRect(cx,cy,cx,cy) );
d4 = cvGetSubRect( src_arr, &d4stub, cvRect(0,cy,cx,cy) );
if(src_arr!=dst_arr){
    if( !CV_ARE_TYPES_EQ( q1, d1 )){
        cvError( CV_StsUnmatchedFormats, "cvShiftDFT",
        "Source and Destination arrays must have the same
```

```
48                      format'', __FILE__, __LINE__ );
                    }//if
50                  cvCopy(q3, d1, 0);
                    cvCopy(q4, d2, 0);
52                  cvCopy(q1, d3, 0);
                    cvCopy(q2, d4, 0);
54              }
                else{
56                  cvCopy(q3, tmp, 0);
                    cvCopy(q1, q3, 0);
58                  cvCopy(tmp, q1, 0);
                    cvCopy(q4, tmp, 0);
60                  cvCopy(q2, q4, 0);
                    cvCopy(tmp, q2, 0);
62              }//else
        }
64  int main(int argc, char* argv[])
    {
66          const char* filename = argc >=2 ? argv[1] : ''esame.bmp'';
            IplImage* img;
68          IplImage* realInput;
            IplImage* imaginaryInput;
70          IplImage* complexInput;
            int dft_M, dft_N;
72          CvMat* dft_A, tmp;
            IplImage* image_Re;
74          IplImage* image_Im;
            double m,M;
76          struct timeval start, end;
        long mtime, seconds, useconds;
78          img = cvLoadImage (filename, CV_LOAD_IMAGE_GRAYSCALE);
```

```
if(!img)
        return -1;
realInput = cvCreateImage (cvGetSize(img), IPL_DEPTH_64F, 1);
imaginaryInput = cvCreateImage( cvGetSize(img), IPL_DEPTH_64F, 1);
complexInput = cvCreateImage( cvGetSize(img), IPL_DEPTH_64F, 2);
cvScale(img, realInput, 1.0 , 0.0);
cvZero(imaginaryInput);
cvMerge(realInput, imaginaryInput, NULL, NULL, complexInput);
dft_M = cvGetOptimalDFTSize( img->height - 1);
dft_N = cvGetOptimalDFTSize( img->width - 1);
dft_A = cvCreateMat( dft_M, dft_N, CV_64FC2);
image_Re = cvCreateImage( cvSize (dft_N, dft_M), IPL_DEPTH_64F, 1);
image_Im = cvCreateImage( cvSize (dft_N, dft_M), IPL_DEPTH_64F, 1);
cvGetSubRect(dft_A, &tmp, cvRect(0,0,img->width, img->height));
cvCopy(complexInput, &tmp, NULL);
if (dft_A->cols > img->width)
{
        cvGetSubRect(dft_A, &tmp, cvRect(img->width, 0 , dft_A->cols
        - img->width, img->height));
        cvZero(&tmp);
}//if
gettimeofday(&start, NULL);
cvDFT( dft_A, dft_A, CV_DXT_FORWARD, complexInput->height);
gettimeofday(&end, NULL);
seconds = end.tv_sec - start.tv_sec;
useconds = end.tv_usec - start.tv_usec;
mtime = ((seconds) * 1000 + useconds/1000.0) + 0.5;
printf("Tempo impiegato: %ld [msec]\n", mtime);
cvNamedWindow("win", CV_WINDOW_AUTOSIZE);
cvNamedWindow("magnitude", CV_WINDOW_AUTOSIZE);
cvShowImage("win", img);
```

```
110         //split Fourier in parte reale e immaginaria
            cvSplit (dft_A, image_Re, image_Im, 0, 0);
112         //calcolo la agrandezza dello spettro Mag = sqrt(Re^2+ Im^2)
            cvPow( image_Re, image_Re, 2.0);
114         cvPow( image_Im, image_Im, 2.0);
            cvAdd( image_Re, image_Im, image_Re, NULL);
116         cvPow( image_Re, image_Re, 0.5);
            //calcolo log(1+Mag)
118         cvAddS( image_Re, cvScalarAll(1.0), image_Re, NULL); //1+Mag
            cvLog(image_Re, image_Re); //log(1+Mag)
120         //Riorganizzo i quadranti dell'immagine di Fourier in modo
            //tale che l'origine sia al centro dell'immagine
122         cvShiftDFT(image_Re, image_Re);
            cvMinMaxLoc(image_Re, &m, &M, NULL, NULL, NULL);
124         cvScale(image_Re,image_Re, 1.0/(M-m),1.0*(-m)/(M-m));
            cvShowImage("magnitude",image_Re);
126         cvWaitKey(-1);
            cvReleaseImage(&img);
128         cvReleaseImage(&image_Re);
            cvReleaseImage(&image_Im);
130         cvReleaseImage(&realInput);
            cvReleaseImage(&imaginaryInput);
132         cvReleaseImage(&complexInput);
            cvDestroyWindow("win");
134         cvDestroyWindow("magnitude");
            return 0;
136 }
```

# Appendice B

# Codice per calcolare la trasformata di Fourier utilizzando la libreria FFTW

Il codice che viene riportato è quello in cui si è sfruttata la simmetria Hermitiana dei dati di input.

Listing B.1: FFTW con simmetria Hermitiana

```
2  #include <SDL.h>
   #include <opencv/highgui.h>
4  #include <opencv/cv.h>
   #include <time.h>
6  #include <stdio.h>
   #include <fftw3.h>
8  #include <core/types_c.h>
   #include <core/core_c.h>
10 #include <sys/time.h>


12 void phase_correlation( IplImage *ref, IplImage *tpl, IplImage *poc );
```

```
14 int main(int argc, char* argv[])
   {
16         const char* filename = argc >=2 ? argv[1] : ''esame.bmp'';
           IplImage* img;
18         IplImage* tpl;
           IplImage *poc;
20         img = cvLoadImage (filename, CV_LOAD_IMAGE_GRAYSCALE);
           if(!img)
22                 return -1;
           tpl = cvLoadImage (filename, CV_LOAD_IMAGE_GRAYSCALE);
24         if(!tpl)
                   return -1;
26         if( ( tpl->width != img->width ) || ( tpl->height != img->height ) ) {
                   fprintf( stderr, ''Le immagini devono avere stessa
28                 larghezza e altezza!\n'' );
                   return -1;
30         }//if
           cvNamedWindow(''Originale'',CV_WINDOW_AUTOSIZE);
32         cvNamedWindow(''Template'',CV_WINDOW_AUTOSIZE);
           cvShowImage(''Originale'',img);
34         cvShowImage(''Template'',tpl);
           //traslazione dell'immagine
36         IplImage *traslo;
           traslo = cvCreateImage(cvGetSize(img),IPL_DEPTH_8U, 1);
38         //spostamento in pixel lungo l'asse X, Y
           int x = img->width/2;
40         int y = img->height/2;
           //matrice di trasformazione
42         CvMat *Trasl = cvCreateMat(2,3,CV_32FC1);
           //assegna il valore 1 all'elemento (0,0)
44         cvmSet(Trasl,0,0,1);
```

```
        //assegna il valore 0 all'elemento (0,1)
46      cvmSet(Trasl,0,1,0);
        //assegna il valore 0 all'elemento (1,0)
48      cvmSet(Trasl,1,0,0);
        //assegna il valore 1 all'elemento (1,1)
50      cvmSet(Trasl,1,1,1);
        //assegna il valore x all'elemento (0,2)
52      cvmSet(Trasl,0,2,x);
        //assegna il valore y all'elemento (1,2)
54      cvmSet(Trasl,1,2,y);
        //traslazione
56      cvWarpAffine(img, traslo,Trasl, CV_INTER_LINEAR +
         CV_WARP_FILL_OUTLIERS,cvScalarAll(0));
58      cvNamedWindow(''traslazione'', CV_WINDOW_AUTOSIZE);
        cvShowImage(''traslazione'', traslo);
60      //creo una nuova immagine per visualizzare la phase correlation
        poc = cvCreateImage(cvSize(tpl->width, tpl->height),
62      IPL_DEPTH_64F,1);
        //eseguo la phase correlation dell'immagine di input
64      phase_correlation(img,traslo,poc);
        //cerco il minmax valore e la sua posizione
66      CvPoint minloc,maxloc;
        double minval, maxval;
68      cvMinMaxLoc(poc, &minval,&maxval, &minloc, &maxloc, 0);
        //lo stampo a video
70      fprintf(stdout, ''Il valore massimo che si trova in posizione (%d, %d) ha
        valore = %2.4f\n'',maxloc.x,maxloc.y,maxval);
72      cvReleaseImage(&poc);
        cvReleaseImage(&tpl);
74  cvWaitKey( 0 );
    cvDestroyWindow( ''Template'' );
```

```
76      cvDestroyWindow( ``Originale'' );
        cvReleaseImage( &tpl );
78      cvReleaseImage( &img );
        cvReleaseImage( &poc );
80  }
    //phase correlation da due immagini e salvataggio del risultato in una terza immagine
82  void phase_correlation (IplImage *ref, IplImage *tpl, IplImage *poc){
            int i, j, k;
84          double tmp;
            struct timeval start, end;
86          long mtime, seconds, useconds;
            //setto le proprietà delle immagini
88          int width = ref->width;
            int height = ref->height;
90          int step = ref-> widthStep;
            int nyh = (width / 2) + 1;
92          int fft_size = nyh*height;
            int fft_sizec = width*height;
94          uchar *ref_data = (uchar*) ref->imageData;
            uchar *tpl_data = (uchar*) tpl->imageData;
96          double *poc_data = (double*) poc->imageData;
            float *orig, *second,*ris;
98          //alloco gli array di input e output della FFTW
            fftwf_complex *img1 =
100          (fftwf_complex*)fftwf_malloc(sizeof(fftwf_complex)*nyh*height);
            fftwf_complex *img2 =
102         (fftwf_complex*)fftwf_malloc(sizeof(fftwf_complex)*nyh*height);
            fftwf_complex *res =
104         (fftwf_complex*)fftwf_malloc(sizeof(fftwf_complex)*nyh*height);
            orig = (float*) fftwf_malloc(sizeof(float)*width*height);
106         second = (float*)fftwf_malloc(sizeof(float)*width*height);
```

```
ris = (float*)fftwf_malloc(sizeof(float)*width*height);
//setup FFTW plans
fftwf_plan fft_img1 = fftwf_plan_dft_r2c_2d(height, width, orig, img1,
FFTW_ESTIMATE);
fftwf_plan fft_img2 = fftwf_plan_dft_r2c_2d(height, width, second, img2,
FFTW_ESTIMATE);
fftwf_plan ifft_res = fftwf_plan_dft_c2r_2d(height, width, res, ris,
FFTW_ESTIMATE);
//carico i dati delle immagini negli array di FFTW input
for (i =0, k=0; i<height; i++){
        for(j=0; j<width; j++, k++){
                orig[k] = (float) ref_data[i*step+j];
                second[k] = (float) tpl_data[i*step+j];
        }
}
gettimeofday(&start, NULL);
fftwf_execute(fft_img1);
gettimeofday(&end, NULL);
fftwf_execute(fft_img2);
seconds = end.tv_sec - start.tv_sec;
useconds = end.tv_usec - start.tv_usec;
mtime = ((seconds) * 1000 + useconds/1000.0) + 0.5;
printf("Tempo impiegato: %ld [msec]\n", mtime);
for (i=0;i<fft_size;i++){
        res[i][0] = ( img2[i][0] * img1[i][0] ) - ( img2[i][1] * ( -img1[i][1]
        ) );
        res[i][1] = ( img2[i][0] * ( -img1[i][1] ) ) + ( img2[i][1] * img1[i]
        [0] );
        //set tmp valore assoluto (reale) dei numeri complessi res[i]
        tmp = sqrt(pow(res[i][0],2.0)+pow(res[i][1],2.0));
        //normalizzazione
```

```
138                 if(tmp == 0){
                            res[i][0] = 0;
140                         res[i][1] = 0;
                    }
142                 else{
                            res[i][0]/=tmp;
144                         res[i][1]/=tmp;
                    }
146         }//for
            //ottengo l'array della phase correlation
148         gettimeofday(&start, NULL);
            fftwf_execute(ifft_res);
150         gettimeofday(&end, NULL);
            seconds = end.tv_sec - start.tv_sec;
152         useconds = end.tv_usec - start.tv_usec;
            mtime = ((seconds) * 1000 + useconds/1000.0) + 0.5;
154         printf("Tempo impiegato per la trasformata di Fourier inversa: %ld
            [msec]\n", mtime);
156         //normalizzo e copio nell'immagine il risultato
            for (i=0;i<fft_sizec;i++)
158         poc_data[i] = (ris[i]/(double)fft_sizec);
            fftwf_destroy_plan( fft_img1 );
160         fftwf_destroy_plan( fft_img2 );
            fftwf_destroy_plan( ifft_res );
162         fftwf_free( img1 );
            fftwf_free( img2 );
164         fftwf_free( res );
}
```

## B.1 Phase correlation

Come si può notare, in quest'ultimo programma viene sviluppata la funzione di phase correlation. Questo metodo è utile per verificare la somiglianza fra due immagini delle stesse dimensioni. Per ottenere la phase correlation di due immagini gli step da seguire sono i seguenti:

- caricare due immagini, f e g;
- eseguire la FFT sulle due immagini, trovando F e G;
- otteniamo lo spettro mediante la seguente formula:

$$R = \frac{FG^*}{|FG^*|}$$

  dove $G^*$ è il complesso coniugato di G;

- ottengo la phase correlation attraverso la IFFT di R.

La posizione del valore massimo corrisponde alla traslazione avvenuta dall'immagine 1 all'immagine 2.

Per testare se la phase correlation funziona correttamente ho eseguito una traslazione dell'immagine originale di 320 pixel lungo l'asse delle ascisse e 240 pixel lungo quella delle ordinate:

Listing B.2: Traslazione dell'immagine

```
IplImage *traslo;
traslo = cvCreateImage(cvGetSize(img),IPL_DEPTH_8U, 1);
//spostamento in pixel lungo l'asse X, Y
int x = img->width/2;
int y = img->height/2;
//matrice di trasformazione
CvMat *Trasl = cvCreateMat(2,3,CV_32FC1);
//assegna il valore 1 all'elemento (0,0)
```

```
        cvmSet(Trasl,0,0,1);
10      //assegna il valore 0 all'elemento (0,1)
        cvmSet(Trasl,0,1,0);
12      //assegna il valore 0 all'elemento (1,0)
        cvmSet(Trasl,1,0,0);
14      //assegna il valore 1 all'elemento (1,1)
        cvmSet(Trasl,1,1,1);
16      //assegna il valore x all'elemento (0,2)
        cvmSet(Trasl,0,2,x);
18      //assegna il valore y all'elemento (1,2)
        cvmSet(Trasl,1,2,y);
20      //traslazione
        cvWarpAffine(img, traslo,Trasl, CV_INTER_LINEAR +
22       CV_WARP_FILL_OUTLIERS,cvScalarAll(0));
        cvNamedWindow("traslazione", CV_WINDOW_AUTOSIZE);
24      cvShowImage("traslazione", traslo);
```

L'immagine traslata che ne risulta è la seguente:

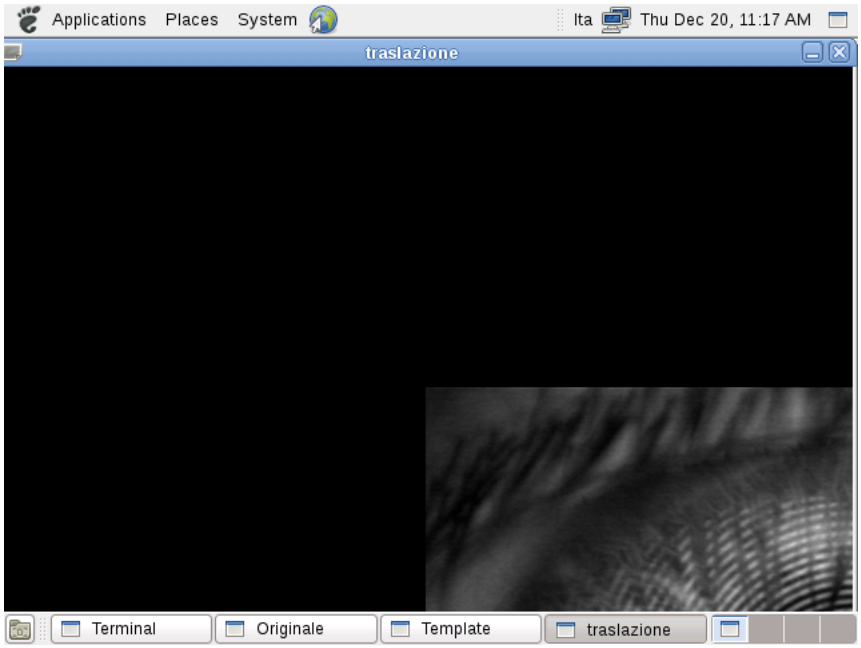



Figura B.1: Immagine traslata

Il calcolo dello spettro avviene all'interno della funzione "phase_correlation()":

Listing B.3: Calcolo dello spettro

```
for (i=0;i<fft_size;i++){
        res[i][0] = ( img2[i][0] * img1[i][0] ) - ( img2[i][1] * ( -img1[i][1]
        ) );
        res[i][1] = ( img2[i][0] * ( -img1[i][1] ) ) + ( img2[i][1] * img1[i]
        [0] );
        //set tmp valore assoluto (reale) dei numeri complessi res[i]
        tmp = sqrt(pow(res[i][0],2.0)+pow(res[i][1],2.0));
        //normalizzazione
        if(tmp == 0){
                res[i][0] = 0;
                res[i][1] = 0;
        }
        else{
                res[i][0]/=tmp;
                res[i][1]/=tmp;
        }
}//for
```

Tornando al main, ora che la variabile poc contiene il risultato della phase correlation, trovo il valore massimo e la sua posizione attraverso la funzione "cvMinMaxLoc()"; infine stampo il risultato a video.

Listing B.4: Ottengo il valore massimo e visualizzo il risultato

```
//cerco il minmax valore e la sua posizione
CvPoint minloc,maxloc;
double minval, maxval;
cvMinMaxLoc(poc, &minval,&maxval, &minloc, &maxloc, 0);
//lo stampo a video
fprintf(stdout, "Il valore massimo che si trova in posizione (%d, %d) ha
valore = %2.4f\n",maxloc.x,maxloc.y,maxval);
```

```
    cvReleaseImage(&poc);
9   cvReleaseImage(&tpl);
```

Come mi aspettavo il massimo viene trovato in posizione (320,240), esattamente la traslazione eseguita sull'immagine originale:

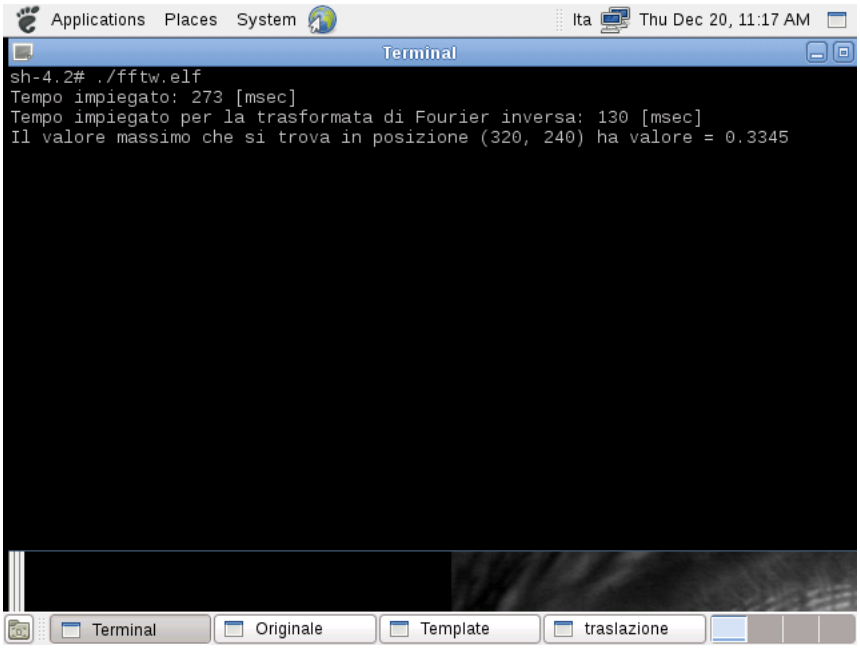



Figura B.2: Visualizzazione dei risultati

# Bibliografia


[1] Gary Bradski, Adrian Kaehler (2008), *Learning OpenCV*, O'Reilly Media.

[2] G. Cariolaro, G. Pierobon, G. Calvagno (2004), *Segnali e Sistemi*, McGraw-Hill.

[3] http://opencv.willowgarage.com/wiki/.

[4] http://it.wikipedia.org/.

[5] http://en.wikipedia.org/.

[6] http://www.fftw.org/.

[7] http://www.lvr.com/eclipse1.htm.

[8] http://blog.galemin.com/.

[9] http://processors.wiki.ti.com/index.php/Building_OpenCV_for_ARM_Cortex-A8.