Roberto Bosco

Navigazione indoor basata su iPhone mediante sensori inerziali e codici posizionali

Tesi di Laurea Magistrale

Relatore: Prof.ssa Maria-Gabriella Di Benedetto

Università degli Studi di Roma La Sapienza
Facoltà di Ingegneria
Ingegneria delle Comunicazioni
Gennaio 2012

Dedicato a ValeMia,
senza te tutto questo non sarebbe possibile.

Ringraziamenti

Ringrazio tutti quelli che si sono sforzati a leggere dietro le mie parole. Ringrazio coloro che sono riusciti, nell'ardua impresa, di tirar fuori il meglio di me. Ringrazio i miei Genitori perché, nel bene o nel male, mi hanno portato dove sono. Ringrazio il mio Maestro sempre in grado di farmi leggere le situazioni dal giusto punto di vista. Ringrazio quelli che mi hanno sostenuto senza nessun doppio fine e coloro che mi hanno aiutato. Un grazie anche a chi mi ha messo i 'bastoni fra le ruote', perché mi ha reso più forte e ha reso tutto questo ancora più soddisfacente. Grazie a Danilo, Zi Pietro, Omone, Accio, Robertino e Alessietto che sono sempre riusciti a strapparmi un sorriso. Grazie alla Prof. Di Benedetto e a tutti "Quelli" del laboratorio ACTS per il modo in cui mi hanno 'contaminato'. Infine un grazie immenso a ValeMia, che mi ha supportato e sopportato in ogni momento.

Roma, gennaio 2012 R. B.

Indice

1 Il Progetto SPinV **1**

2 Tecniche di localizzazione indoor **3**

2.1 Introduzione 3

2.2 Approccio Geometrico 5

2.2.1 Time of Arrival 8

2.2.2 Time Different of Arrival 11

2.2.3 Risoluzione temporale 12

2.2.4 Angle of Arrival 13

2.3 Proximity System 15

2.4 Received Signal Strenght Indication 17

2.5 Errori di localizzazione 19

2.6 I sistemi di localizzazione esistenti 21

2.6.1 Sistemi Time Based 23

2.6.2 Sistemi RSSI 29

2.7 Conclusioni 41

3 Dead Reckoning **43**

3.1 Introduzione 43

3.2 Principi del Dead Reckoning 47

3.3 Sensori Inerziali 49

3.3.1 Accelerometro 50

3.3.2 Giroscopio 53
3.3.3 Sensore di campo magnetico 54
3.3.4 Orientamento 57
3.3.5 Errori nelle misure inerziali 61
3.4 Calcolo della distanza 66
3.5 Calcolo della direzione 77
3.6 Stima della nuova posizione 80
3.7 Dead Reckoning Test 82
3.8 Conclusioni 92

4 Autolocalizzazione **93**
4.1 Introduzione 93
4.2 Codici Bidimensionali 94
4.3 Positional Code 100
4.4 Nozioni di Geometria Proiettiva 103
4.4.1 Coordinate Omogenee 103
4.4.2 Omografia 104
4.5 Modelli di fotocamera 106
4.5.1 Modello semplificato 107
4.5.2 Modello generale 111
4.6 Conclusioni e SPinV Code 118

5 Il Processo di Decodifica **121**
5.1 Introduzione 121
5.2 La piattaforma di sviluppo e le Applicazioni 122
5.3 Lo Standard Jpg 123
5.3.1 Codifica Jpeg 124
5.3.1.1 Nozioni preliminari 125
5.3.1.2 Trasformata Discreta del Coseno (DCT) 129
5.3.1.3 Quantizzazione e preparazione alla codifica 134

5.3.1.4 Codifica Entropica 138
5.3.1.5 Scrittura e Struttura del File 146
5.3.2 Decodifica Jpeg 156
5.4 Decodifica SPinV Code 167

6 Conclusioni e Sviluppi Futuri **175**

A Il codice Matlab: Dead Reckoning **179**

B Il codice C: Il Processo di Decodifica **183**

Bibliografia **261**

Elenco delle figure

2.1 Classificazione delle tecniche di localizzazione 6
2.2 Trilateration: Caso tipico della tecnica ToA La posizione dell'utente è data dal punto di intersezione delle tre circonferenze 8
2.3 Ambiguità presente nella localizzazione 2D con due misure a disposizione 10
2.4 Multilateration: caso tipico della tecnica TDoA in 2D. La posizione dell'utente è data dal punto di intersezione fra le due iperboli 12
2.5 Triangulation: caso tipico della tecnica AoA La posizione dell'utente è data dal punto di intersezione fra le due rette 14
2.6 Proximity System: La posizione dell'utente è calcolata come il baricentro del poligono 16
2.7 Caso reale ToA: non si ha un punto, ma un area di intersezione . . . 21
2.8 Caso reale ToA: calcolo della posizione dell'utente come intersezione fra rette costruite a partire dalle intersezioni tra le circonferenze . . . 22
2.9 Sensori e Tag Ubisense 25
2.10 Bat System 29
2.11 Mappa degli esperimenti del progetto RADAR 31
2.12 Algoritmo NNSS-AVG 32
2.13 cumulative distribution function dell'errore sulla stima della posizione - confronto fra i tre metodi 33
2.14 Prestazioni del sistema RADAR al variare del numero k di nearest neighbors considerati nei casi di 25° e 50° percentile 34
2.15 RSS percepito su un access point Bluetooth e su un access point Wi-Fi 36
2.16 SSD percepito tra una coppia di access point Bluetooth e access point Wi-Fi 36
2.17 Esempio di posizionamento di 9 readers che presentano due differenti regioni di copertura 38
2.18 Sistema Landmarc: la mappa di riferimento con il posizionamento dei 4 readers, dei 16 reference tags e degli 8 tracking tags 39

2.19 Sistema Landmarc: Errore percentile cumulativo sul posizionamento al variare delle k distanze euclidee scelte 40
2.20 Sistema Landmarc: Errore percentile cumulativo sul posizionamento per n=3 e n=4 readers 40
2.21 Sistema Landmarc: Errore percentile cumulativo sul posizionamento,valutato durante le ore diurne e notturne 41

3.1 Play Station Move (sulla sinistra), Wii Remote e WiiMotionPlus . . 44
3.2 Rappresentazione geometrica del dead reckoning 48
3.3 Ellisse di confidenza che contiene al 95% l'errore commesso 49
3.4 Assi di riferimento in iPhone 4 - sdr body per iPhone 50
3.5 Applicazione "XSensor Pro": Accelerazioni misurate lungo i 3 assi durante uno scuotimento "lieve" dell'iPhone 52
3.6 Schema di un giroscopio meccanico 53
3.7 Applicazione "Bussola": indicazione del nord magnetico 54
3.8 Applicazione "XSensor Pro": Misurazioni del campo magnetico al variare delle condizioni 56
3.9 Schermata di calibrazione della bussola nei dispositivi Apple e movimento "a 8" 57
3.10 Orientamento basato sulla terna $(Roll, Pitch, Yaw)$ 58
3.11 Orientamento usato in aviazione 59
3.12 xSensor Pro: misurazioni giroscopio e stima orientamento. Pocket IMU: misurazioni sensoriali e orientamento 60
3.13 Errori legati al fattore di scala 62
3.14 Errori legati alla non ortogonalità degli assi reali dei sensori 63
3.15 Livella di superficie, filo a piombo e livella classica nell'applicazione Carpentiere iHandy 64
3.16 Pedometer: Applicazione per il conteggio dei passi 67
3.17 Cammino rettilineo: valori di accelerazioni misurati lungo l'asse x . . 68
3.18 Cammino rettilineo: valori di accelerazioni misurati lungo l'asse y . . 69
3.19 Cammino rettilineo: valori di accelerazioni misurati lungo l'asse z . . 69
3.20 Cammino rettilineo: accelerazione totale 69
3.21 Cammino rettilineo: Doppia Soglia per valutare il numero di passi effettuati 70
3.22 Cammino con curva a destra e terminale impugnato con la mano destra: valori di accelerazioni misurati lungo i tre assi 71
3.23 Cammino con curva a destra e terminale impugnato con la mano destra: accelerazione totale 71

3.24 Cammino con curva a destra e terminale impugnato con la mano sinistra: valori di accelerazioni misurati lungo i tre assi 73
3.25 Cammino con curva a destra e terminale impugnato con la mano sinistra: accelerazione totale 73
3.26 Scala in salita: valori di accelerazioni misurati lungo i tre assi 74
3.27 Scala in salita: accelerazione totale 75
3.28 Conteggio numero dei passi con metodo Zero-Crossing ([34]) 76
3.29 Frequenza della camminata versus lunghezza del passo. Varianza dell'accelerazione versus lunghezza del passo 77
3.30 Terna (yaw,roll,pitch) durante cammino rettilineo 78
3.31 yaw, media istantanea e valore medio durante cammino rettilineo . . 79
3.32 Terna (yaw,roll,pitch) durante curva a destra di 90° 79
3.33 Terna (yaw,roll,pitch) durante curva a sinistra di 90° 80
3.34 Stima della nuova posizione - sullo sfondo la mappa del piano terra della facoltà di Ingegneria in San Pietro in Vincoli 81
3.35 I quattro percorsi 82
3.36 Risultati del Test 1.b: percorso rettilineo 85
3.37 Risultati del Test 2.b: percorso con curva a destra 86
3.38 Risultati del Test 3.b: percorso con doppia curva a sinistra 87
3.39 Risultati del Test 4.b: percorso misto con 4 curve 88
3.40 Stime di posizione ottenute nelle tre prove del Test 1 - numero di passi contati e coordinate finali 89
3.41 Stime di posizione ottenute nelle tre prove del Test 2- numero di passi contati e coordinate finali 89
3.42 Stime di posizione ottenute nelle tre prove del Test 3- numero di passi contati e coordinate finali 90
3.43 Stime di posizione ottenute nelle tre prove del Test 4- numero di passi contati e coordinate finali 90

4.1 Esempio di codice monodimensionale e bidimensionale 94
4.2 Evoluzione dei codici bidimensionali: codici a barre multipli, codici stackable e codici a matrice 95
4.3 Struttura del Qr Code - nell'immagine un QR code versione 4 97
4.4 Struttura del Data Matrix 98
4.5 Esempi di Bee Tagg 99
4.6 High Capacity Color Barcode della Microsoft 100
4.7 Il problema dell'autolocalizzazione 102
4.8 La proiezione da un punto dello spazio mappa punti e linee del piano π in punti e linee del piano π' 105

4.9 Esempi pratici di omografia 105
4.10 Distorsione a barilotto e distorsione a cuscinetto 107
4.11 Modello geometrico semplice della fotocamera 108
4.12 Vista lungo il piano X Z del modello geometrico semplice della fotocamera 109
4.13 Sistemi di riferimento nel modello generale 111
4.14 Punti di riferimento e relative rette da inserire per attuare la correzione prospettica mediante programma 116
4.15 Correzione prospettica dell'immagine di figura 4.14 mediante PTGui 117
4.16 Modello semplice: calcolo della distanza 117
4.17 Easymeasure: Calcolo della distanza, dell'altezza e della larghezza sfruttando l'angolo α valutato mediante l'accelerometro 118
4.18 SPinV Code che codifica la sequenza 0101010101 120

5.1 Schema a blocchi concettuale del codificatore jpeg 125
5.2 Ordine di analisi dei blocchetti 8x8 al variare del sottocampionamento effettuato 128
5.3 Ordine di analisi dei blocchetti in immagini con un unica componente 129
5.4 Completamento dei blocchetti in immagini con dimensione non multipla del blocchetto elementare 129
5.5 Rappresentazione in scala di grigi della base DCT 132
5.6 Immagine e relativa trasformata DCT-2 133
5.7 Codifica differenziale dei coefficienti DC 137
5.8 Scansione a zig-zag 138
5.9 Visualizzazione multipla delle immagini in iPhone 4 utilizzando le miniature 150
5.10 Struttura in un file Exif con miniatura 151
5.11 Schema a blocchi concettuale del decodificatore jpeg 156
5.12 Struttura di un file jpeg - visualizzazione mediante hex-fiend 157
5.13 Debugger Consolle: dati iniziali 159
5.14 Debugger Consolle: matrici di quantizzazione 160
5.15 Debugger Consolle: segmento Frame Header 160
5.16 Debugger Consolle: segmento Codici di Huffman 162
5.17 Debugger Consolle: segmento Scan Header e analisi preliminare sull'ordinamento dei blocchetti 163
5.18 Esempio di decodifica entropica dei primi coefficienti nel file Sedicix-Sedici.jpeg 164
5.19 Esempio di decodifica entropica nel file SedicixSedici.jpeg - presenza della corsa di zeri 165

5.20 Valori finali del primo blocchetto di luminanza del file SedicixSedici.jpeg166
5.21 Aree di analisi dello slot e problema di rotazione dello SPinV Code . 169
5.22 Rotazione della barra rettangolare 170
5.23 Alcuni scatti fotografici dello SPinV Code correttamente decodificati 174

Elenco delle tabelle

2.1 Specifiche tecniche del sensore Ubisense serie 7000 26
2.2 Specifiche tecniche del tag Ubisense Slim 27
2.3 25°, 50° e 75° percentile dell'errore in distanza sulla localizzazione . 33

3.1 Valor medio e varianza dei valori di accelerazione misurati nel test di calibrazione 65
3.2 Valor medio e varianza dei valori delle velocità angolari misurate nel test di calibrazione 65
3.3 Valor medio e varianza dei valori di orientamento stimati nel test di calibrazione (CoreMotion API) 65

4.1 Caratteristiche di alcuni tipi di codici bidimensionali 96

5.1 Valori di luminanza di un blocchetto 8x8 134
5.2 DCT bidimensionale del blocchetto in tabella 5.1 134
5.3 Passi di quantizzazione per la luminanza suggeriti nello standard . . 135
5.4 Passi di quantizzazione per le crominanze suggeriti nello standard . . 135
5.5 Quantizzazione dei coefficienti DCT di tabella 5.2 136
5.6 Valori di *Size* in relazione all'ampiezza del coefficiente *Amplitude* . . 139
5.7 Codifica di Huffman proposta nello standard per il SIMBOLO 1 dei coefficienti ΔDC 142
5.8 Alcuni valori della codifica di Huffman proposta nello standard per il SIMBOLO 1 dei coefficienti AC 143
5.9 Marker di interesse in JFIF nella modalità Baseline 146
5.10 Marker e relativi segmenti di interesse nel file JFIF nella modalità Baseline 147
5.11 Struttura dell' Application Segment 0 149
5.12 Struttura del segmento Matrice di Quantizzazione 151
5.13 Struttura del segmento Frame Header 152
5.14 Struttura del segmento Codici di Huffman 153
5.15 Struttura del segmento Scan Header 155

Capitolo 1

Il Progetto SPinV

SPinV è l'acronimo che sta per San Pietro in Vincoli, antica Basilica che sorge accanto alla sede principale della facoltà di Ingegneria de La Sapienza in Roma, facoltà in cui è nato e si sta sviluppando il progetto SPinV; il nome San Pietro in Vincoli è spesso utilizzato anche per indicare la zona in cui è situata la facoltà.

Il progetto nasce dall'esigenza di colmare il gap esistente, in ambito di localizzazione, fra l'ambiente outdoor e indoor. Di fatti in ambiente non indoor, la localizzazione è effettuata sfruttando il sistema GPS (Global Positioning System); in particolare, a partire dal Maggio 2000, è stata eliminata l'interferenza volontaria da parte del Governo degli Stati Uniti (si vedano [1], [2]) consentendo di fatto l'utilizzo del GPS in ambito civile, e ciò ha determinato, nell'ultima decade, un fortissimo sviluppo di tutte le tecnologie che ruotano attorno al GPS al fine di migliorare la stima della posizione e tutti i servizi annessi. Il sistema GPS però, a causa della degradazione che subisce il segnale satellitare, non può essere usato in ambito indoor (si veda El Rabbany [3]). Il progetto SPinV nasce proprio da quest'ultima considerazione. Per la navigazione in ambito outdoor, sia veicolare che pedonale, si può dunque affermare che non c'è più molto da innovare in relazione a quanto presenta l'attuale panorama di mercato.

L'obiettivo di SPinV è quello di realizzare un sistema di localizzazione e navigazione utilizzabile in ambito indoor, in cui l'utente sia in grado di visualizzare la propria posizione così che possa usufruire di servizi basati sulla posizione acquisita (LSB - Location Based Service). Un progetto di questo tipo trova applicazione in tutte quelle strutture pubbliche e attività commerciali in cui può essere utile guidare gli utenti e fornire loro informazioni aggiuntive: ospedali, centri commerciali, università, aeroporti, musei, uffici pubblici...

Nel *Capitolo 2* si descriveranno le principali tecniche di localizzazione indoor, affrontando tematiche sia concettuali che pratiche. In particolare, nella prima parte del capitolo verranno descritti gli approcci base di un sistema di localizzazione e

le tecniche teoriche utilizzabili per stimare la posizione dell'utente. In seguito si analizzeranno degli aspetti più reali, con la descrizione di alcuni sistemi di localizzazione esistenti sul mercato. Si cercherà di accennare a gran parte delle problematiche presenti in un sistema di localizzazione, così da individuarne le caratteristiche essenziali.

Nei *Capitoli 3 e 4* si descriverà il metodo che si è deciso di adottare in questo lavoro di tesi basato sui concetti di dead reckonig e autolocalizzazione. A questi due concetti sono direttamente legati i sensori inerziali e i codici bidimensionali e per questo motivo si farà una descrizione di questi argomenti.
Nel Capitolo 3 si adatteranno i principi del dead reckoning all'approccio proposto e, sfruttando i sensori inerziali dell'iPhone, verranno mostrati i risultati dei test sulla localizzazione, sottolineando i punti di forza e i limiti dell'approccio proposto.
Nel Capitolo 4 si descriverà un approccio alternativo ai codici bidimensionali, che si collega direttamente ai concetti di dead reckoning; verranno descritte le linee guide per risolvere il problema dell'autolocalizzazione e infine, sulla base delle considerazioni effettuate, verrà proposto un codice (SPinV Code).

Nel *Capitolo 5* si affronteranno le tematiche legate al processo di decodifica del file nel formato jpg e dello SPinV Code. Per questo motivo verrà trattato in maniera approfondita il processo di codifica e decodifica dello standard jpeg. In seguito, verranno descritte le tecniche che sono state usate per la decodifica/lettura dello SPinV Code. In *Appendice B* è stato inserito lo script, relativo al processo di decodifica. Questo è scritto interamente in linguaggio C e permette, entro certe condizioni, di decodificare correttamente il file jpeg e leggere lo SPinV Code a partire da una fotografia scattata realmente con il dispositivo di riferimento usato (iPhone 4).

Nel *Capitolo 6* si parlerà infine dei limiti offerti dall'approccio proposto e di tutte le possibili evoluzioni del sistema.

Capitolo 2

Tecniche di localizzazione indoor

Indice

2.1 Introduzione 3
2.2 Approccio Geometrico 5
2.2.1 Time of Arrival 8
2.2.2 Time Different of Arrival 11
2.2.3 Risoluzione temporale 12
2.2.4 Angle of Arrival 13
2.3 Proximity System 15
2.4 Received Signal Strenght Indication 17
2.5 Errori di localizzazione 19
2.6 I sistemi di localizzazione esistenti 21
2.6.1 Sistemi Time Based 23
2.6.2 Sistemi RSSI 29
2.7 Conclusioni 41

2.1 Introduzione

Il seguente capitolo descrive le tecniche di localizzazione utilizzate in ambito indoor; le tematiche sono descritte con un approccio volutamente generico, così da applicare i concetti base alle molteplicità di tecnologie disponibili. Per eventuali approfondimenti si riporta la letteratura (ampiamente presente nel settore) e le singole tecnologie. Prima di iniziare la trattazione, è utile sottolineare alcuni aspetti e classificazioni.

Lo *scenario* che si considera è costituito da due elementi base: il *terminale di utente* o *apparato di utente* e gli *apparati di rete* che sono i nodi che costituiscono l'*infrastruttura di rete*. In generale, il processo di localizzazione si divide in due fasi distinte: una prima fase prevede delle misurazioni (tempi, angoli o potenze), e in una

seconda fase, si utilizza un determinato algoritmo, che utilizza i parametri misurati, per determinare la posizione del terminale di utente. Sono possibili tre situazioni:

- il terminale mobile riceve il segnale irradiato dai nodi di rete, effettua le misurazioni di uno o più *parametri* e stima la propria posizione; in tal caso si parla di *localizzazione interna* o localizzazione *terminal based*;
- gli apparati di rete (due o più) ricevono il segnale irradiato dal terminale di utente, stimano uno o più parametri mediante opportune misurazioni e determinano la posizione dell'utente. Questa situazione è indicata con il termine di *localizzazione esterna* o localizzazione *network based*;
- approcci ibridi:
  - *terminal-based, network-assisted*: Gli apparati di rete inviano dei segnali ausiliari, quali coordinate di riferimento, informazioni temporali per aiutare il terminale mobile nel processo di localizzazione;
  - *network-based, terminal-assisted*: Il terminale mobile effettua le misurazioni e le comunica alla rete che provvede al calcolo della posizione dell'apparato di utente.

É importante aggiungere, che un ruolo fondamentale, in un sistema di localizzazione è svolto dal *protocollo di rete*, essenziale per la comunicazione tra gli apparati coinvolti nel processo di localizzazione. Un altro elemento di primaria importanza di qualsiasi sistema di localizzazione è il sistema di riferimento, proprio perché il "risultato" della localizzazione deve essere espresso su un determinato riferimento; in particolare, si parla di coordinate geografiche e coordinate locali (si veda Mao e Fidan [4]). Le *coordinate geografiche* sono utilizzate identificare univocamente la posizione di un punto sulla superficie terrestre mediante l'utilizzo di una terna di valori (esempio classico è il GPS). Le *coordinate locali* invece si basano su un sistema cartesiano, la cui origine degli assi è fissata in maniera locale ed arbitraria. Per il progetto SPinV, l'attenzione ricade sulle coordinate locali, proprio perché ci si riferisce ad applicazioni indoor e non si ha interesse ad un posizionamento rispetto a un punto sulla superficie terrestre. In definitiva, è possibile riassumere gli elementi essenziali di un sistema di localizzazione:

- terminale di utente;
- infrastruttura di rete costituita da un insieme di apparati di rete ed eventualmente da un unità centrale di gestione ed elaborazione;
- uno o più parametri ottenuti mediante opportune misurazioni;

- metodo di posizionamento che sfrutta i parametri di localizzazione e che fornisce la posizione del terminale di utente in un opportuno sistema di riferimento;
- protocollo di rete.

Una distinzione che può essere fatta è relativa al tipo di infrastruttura di rete utilizzata, si distinguono due grandi categorie: infrastruttura dedicata o infrastruttura indipendente.
Un *infrastruttura dedicata*, anche chiamata infrastruttura stand-alone, è costituita da un insieme di apparati di rete che sono utilizzati per sole finalità di posizionamento. In sistemi di questo tipo, spesso sono richiesti apparati di utente specifici; l'esempio classico è quello del GPS, progettato e ottimizzato per consentire la localizzazione degli utenti.
Un *infrastruttura integrata* è invece un infrastruttura che "nasce" inizialmente per finalità di comunicazione, e successivamente viene integrata così da poter essere adoperata anche per finalità di posizionamento. Per cui, il grosso vantaggio di un sistema che usa un infrastruttura integrata è che la maggior parte degli apparati di rete di cui è costituita, così come i protocolli, possono essere riutilizzati per il positioning, inoltre non sono necessari apparati di utente specifici; bisogna però considerare che il progetto iniziale dell'infrastruttura di rete non è stato ottimizzato per la localizzazione e dunque l'integrazione finalizzata al positioning, risulta in alcuni casi complessa.

Possono essere effettuate diverse classificazioni delle tecniche di localizzazione indoor, basandosi su diversi "criteri", in questo lavoro di tesi si è considerata la classificazione espressa dalla figura 2.1.

2.2 Approccio Geometrico

Con L'approccio geometrico è possibile individuare la posizione del terminale di utente mediante l'intersezione fra curve nel piano; il tipo di curva ottenuta dipende dal tipo di misurazione (*parametro di localizzazione*) che viene effettuata:

- Misure temporali ⇒ metodi time-based:
  - Time of Arrival (ToA);
  - Time Different of Arrival (TDoA);

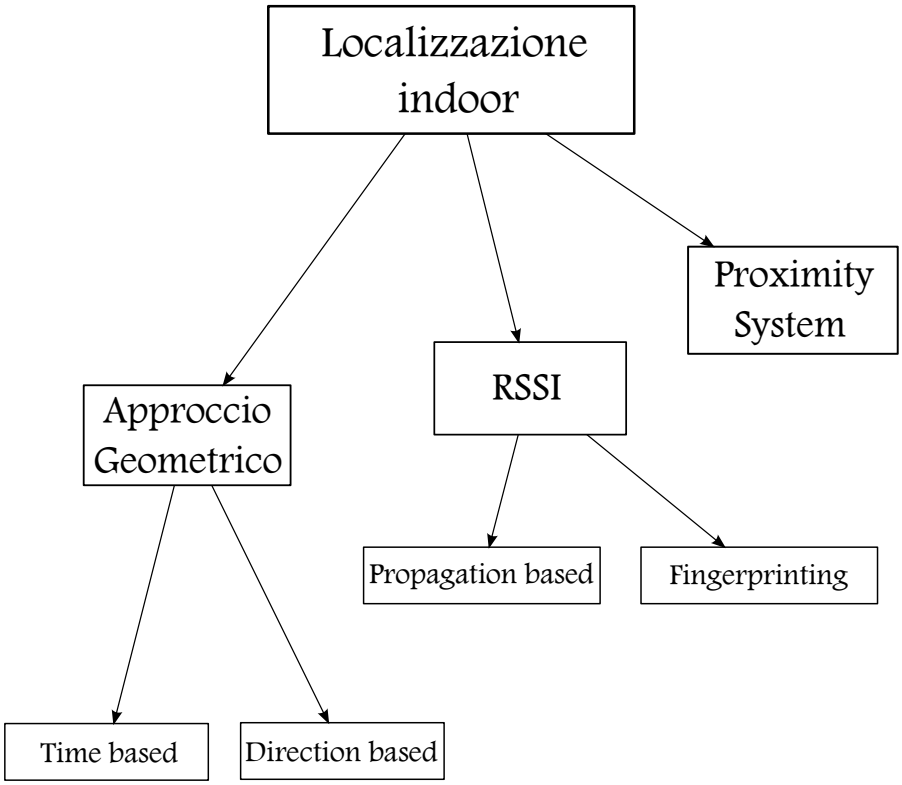


Figura 2.1: Classificazione delle tecniche di localizzazione

- Misure angolari ⇒ metodi direction-based:
  - Angle of Arrival (AoA).

Per quanto riguarda le misurazioni temporali, è importante sottolineare la differenza fra sistemi a una e a due vie. Per esprimere questi concetti, considero momentaneamente uno scenario in cui sono presenti un trasmettitore e un ricevitore posti a distanza r. Si vuole calcolare tale distanza sfruttando il ritardo di propagazione dell'onda e.m. Indico con t_1 il tempo di arrivo al ricevitore e con t_0 il tempo di partenza dal trasmettitore.

$$r = c\,(t_1 - t_0) \tag{2.1}$$

L'equazione 2.1 è rispettata se gli orologi del trasmettitore e del ricevitore sono perfettamente sincronizzati tra loro; nelle situazioni reali, fra il trasmettitore e il ricevitore è sempre presente un offset δt sulla temporizzazione che determina un calcolo errato della distanza; indico con ρ la distanza valutata realmente, mentre r è la distanza vera.

$$\rho = c\,(t_1 - t_0) + c\,\delta t = r + c\,\delta t \tag{2.2}$$

anche un errore "piccolo" sulla temporizzazione, determina un errore "considerevole" sulla della distanza:

$$\delta t = 1\mu s \;\mapsto\; c\,\delta t = 300m$$

La situazione appena descritta è caratteristica dei *sistemi a una via*. Nei *sistemi a due vie*, invece, si sfrutta il segnale di andata e ritorno, in tal caso si parla di *Round Trip Time - RTT*, ovvero il segnale trasmesso dal trasmettitore, viene rilanciato indietro dal ricevitore. Quindi la differenza temporale è valutata dal trasmettitore. In tal caso non è necessario il sincronismo tra trasmettitore e ricevitore, proprio perché si tiene in considerazione solo l'orologio del trasmettitore. Nei casi pratici, deve essere noto il tempo di latenza $\Delta_{RX/TX}$ del ricevitore, ovvero il tempo necessario alla ricezione e alla ritrasmissione del segnale da parte del ricevitore. Nel caso di misura a due vie, la distanza r è valutata secondo la 2.3:

$$r = \frac{c}{2}\,(t_1 - t_0 - \Delta_{RX/TX}) \tag{2.3}$$

Per quanto detto, si capisce l'importanza della sincronizzazione temporale in apparati che utilizzano segnali in radiofrequenza per effettuare misurazioni temporali. In particolare, ottenere un elevata sincronizzazione, in alcuni casi, può significare utilizzare degli orologi estremamente precisi (aumento dei costi), oppure aumentare la complessità del sistema utilizzando degli algoritmi molto sofisticati (si veda Mao e Fidan [4]). Un alternativa, per ridurre gli effetti causati dall'offset tra gli orologi, è quella di utilizzare gli *ultrasuoni* piuttosto che segnali RF. Infatti,la velocità di propagazione di un onda sonora è molto più piccola della velocità di propagazione di un onda em in RF e un eventuale non sincronia si traduce in un errore in distanza notevolmente più basso; nei casi pratici, si sfrutta la differenza temporale tra la propagazione del segnale in RF e l'ultrasuono, ma è importante sottolineare che tale tecnica di differenza temporale è diversa dal TDoA che verrà descritto nel sottoparagrafo 2.2.2. Una volta effettuato il calcolo di uno dei parametri di localizzazione (tempo o angolo), si utilizza un *algoritmo di localizzazione* che stima la posizione dell'apparato di utente a partire dai parametri di localizzazione calcolati.

2.2.1 Time of Arrival

L'approccio **ToA** (Kolodziej e Hjelm [5], Samama [6] Eladio et al. [7]) sfrutta un principio simile a quello che è alla base del posizionamento con il sistema GPS. Con il ToA si misura l'intervallo di tempo che intercorre fra la trasmissione del segnale da parte dei nodi di rete e la ricezione da parte dell'apparato di utente (misura del *Time of flight*). Tali differenze temporali si traducono in calcoli della distanza, considerando la velocità di propagazione c (velocità della luce nel vuoto).
Dal punto di vista geometrico il problema dell'individuazione della posizione del terminale di utente si riconduce all'intersezione delle tre circonferenze nel piano (si veda la figura 2.2). Il problema è noto come *Circular Lateration* o *Trilateration*.

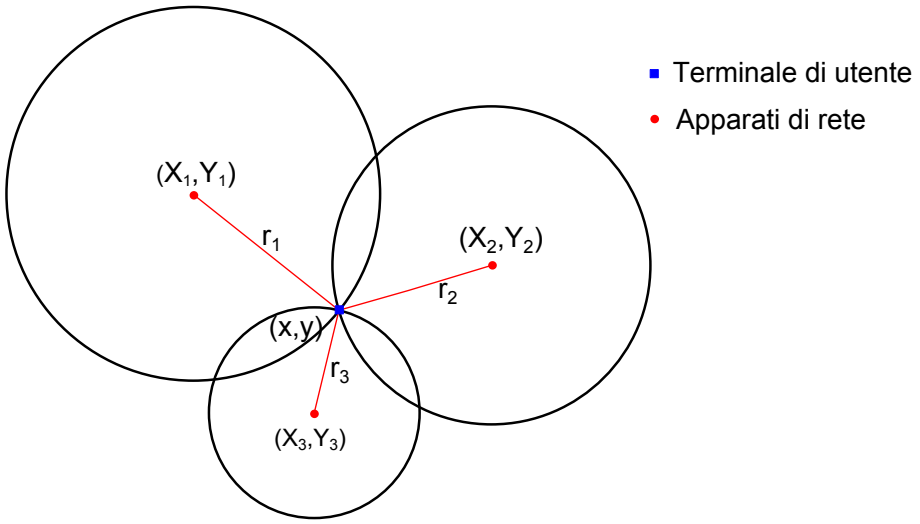


Figura 2.2: Trilateration: Caso tipico della tecnica ToA La posizione dell'utente è data dal punto di intersezione delle tre circonferenze

Dal punto di vista analitico il problema è il seguente:

$$c\,(t_i - t_0) = r_i = \sqrt{(X_i - x)^2 + (Y_i - y)^2} \qquad i = 1, 2, 3 \tag{2.4}$$

Dove t_i è l'istante temporale in cui viene ricevuto il segnale dal terminale di utente proveniente dall'i-esimo nodo di rete. t_0 è l'istante di trasmissione da parte degli apparati di rete. $(t_i - t_0)$ è l'intervallo di tempo tra la trasmissione e la ricezione

del segnale. r_i è la distanza tra l' i-esimo apparato di rete e il terminale di utente. (X_i, Y_i) sono le coordinate locali dell' $i-esimo$ apparato di rete. (x, y) sono le coordinate locali del terminale di utente.
Nel caso 3D si aggiungono le coordinate Z_i e z che rappresentano rispettivamente le altitudini dell' i-esimo apparato di rete e del terminale di utente. Nel caso di localizzazione su un sistema tridimensionale, sono necessarie quattro equazioni (ovvero quattro misurazioni temporali) per risolvere il sistema. Questa situazione è descritta nell'equazione 2.5.

$$c\,(t_i - t_0) = r_i = \sqrt{(X_i - x)^2 + (Y_i - y)^2 + (Z_i - z)^2} \qquad i = 1, 2, 3, 4 \qquad (2.5)$$

È importante sottolineare alcune considerazioni che scaturiscono direttamente da tale tecnica:

- Il problema che si è considerato è un problema di localizzazione nel piano, ovvero di ricerca di una coppia di coordinate locali, quindi sono necessarie tre misure temporali per individuare univocamente, nel caso ideale, la posizione dell'utente. Se il problema della localizzazione fosse nello spazio, sarebbero state necessarie 4 misure temporali (ovvero 4 distanze), e in tal caso la posizione del terminale è il risultato dell'intersezione tra sfere nello spazio. Più in generale servono n+1 equazioni (n+1 misurazioni temporali) e n+1 coordinate di riferimento per determinare la posizione in un sistema n-dimensionale. Nel caso di localizzazione 2D, la "terza misura" è necessaria per eliminare l'ambiguità derivante dall'intersezione delle due circonferenze, che produce due diversi punti di posizionamento. Questa situazione è rappresentata in figura 2.3. In questa situazione ho considerato una perfetta sincronizzazione trasmettitori/ricevitore.

- Nel sistema GPS, sono i satelliti che trasmettono il segnale "circa" allo stesso istante temporale e si valuta l'intervallo di tempo che intercorre fra la trasmissione dei segnali satellitari e la ricezione da parte del ricevitore GPS. Ciò implica che i satelliti debbano essere *sincronizzati* tra loro, ovvero che inizino a trasmettere nello stesso istante. Questo è possibile mediante l'utilizzo di orologi atomici e di tecniche supplementari che prevedono l'ausilio del control segment (si veda Samama [6]) per correggere eventuali errori di sincronizzazione tra satelliti. Inoltre, è necessaria la sincronizzazione tra gli orologi satellitari e l'orologio del ricevitore. Infatti, come detto all'inizio del capitolo, in assenza di tale sincronizzazione in sistemi a una via, si commetterebbe un

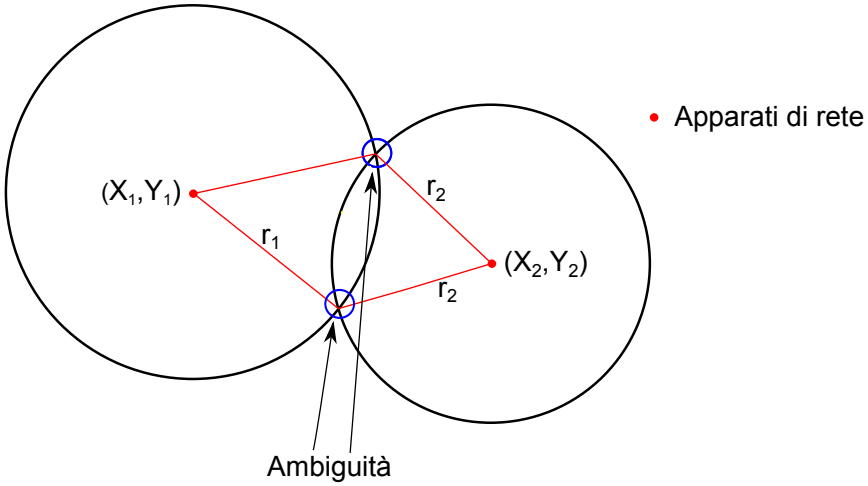


Figura 2.3: Ambiguità presente nella localizzazione 2D con due misure a disposizione

errore inaccettabile nel calcolo delle distanza (che poi si ripercuoterebbe nella valutazione della posizione dell'utente). In definitiva, nel caso GPS (caso tridimensionale), la quarta equazione è sfruttata, sia per eliminare l'ambiguità (l'intersezione tra sfere non genera un unico punto) che per valutare l'offset δt fra gli orologi satellitari e l'orologio del ricevitore. Tornando all'approccio ToA nel caso bidimensionale, valgono delle considerazioni analoghe: servono tre misurazioni, ovvero tre equazioni, così da eliminare l'ambiguità e valutare l'offset δt fra l'orologio del terminale di utente e gli orologi dei nodi di rete.

- Per il calcolo del time of flight è necessario conoscere l'istante di trasmissione t_0.

- Nel sistema ToA descritto, la posizione viene stimata dal terminale di utente; per cui la localizzazione è interna.

- L'approccio ToA può essere anche applicato inversamente: il terminale di utente trasmette e il segnale viene ricevuto da tre o più apparati di rete. In tal caso la localizzazione è esterna.

2.2.2 Time Different of Arrival

L'approccio **TDoA** (Kolodziej e Hjelm [5], Samama [6], Eladio et al. [7]), può essere applicato nei due versi. "Lato utente": l'apparato di utente deve essere in grado di misurare le differenze dei ritardi di propagazione fra gli apparati di rete e il terminale di utente. "Lato rete": tre o più nodi di rete devono calcolare il TDoA di un segnale inviato dal terminale di utente. Per meglio comprendere questo concetto, considero il caso lato utente: il terminale inizia a trasmettere al tempo t_0. Indico con t_1, t_2, t_3 gli istanti di arrivo del segnale a tre apparati di rete:

$$\begin{cases} t_1 = t_0 + \Delta t_1 \\ t_2 = t_0 + \Delta t_2 \\ t_3 = t_0 + \Delta t_3 \end{cases} \tag{2.6}$$

dove Δt_i è data dalla 2.7:

$$\Delta t_i = \frac{\sqrt{(X_i - x)^2 + (Y_i - y)^2}}{c} \qquad i = 1, 2, 3 \tag{2.7}$$

Sono noti gli istanti di arrivo t_1, t_2, t_3, e le coordinate degli apparati di rete (X_i, Y_i); non è noto l'istante di inizio trasmissione t_0 e , ovviamente, le coordinate (x, y). É possibile semplificare il sistema espresso nella 2.7 operando le differenze $(t_2 - t_1)$ e $(t_3 - t_1)$. Si ottengono così delle equazioni indipendenti da t_0, come espresso nella 2.8:

$$\begin{cases} t_2 - t_1 = \Delta t_2 - \Delta t_1 \\ t_3 - t_1 = \Delta t_3 - \Delta t_1 \end{cases} \tag{2.8}$$

Dalla 2.8, si capisce perché nei sistemi TDoA. la sincronizzazione tra il terminale e i nodi di rete, non sia necessaria. Considerando la 2.7, si ottiene la 2.9:

$$\begin{cases} c\,(t_2 - t_1) = \sqrt{(X_2 - x)^2 + (Y_2 - y)^2} - \sqrt{(X_1 - x)^2 + (Y_1 - y)^2} \\ c\,(t_3 - t_1) = \sqrt{(X_3 - x)^2 + (Y_3 - y)^2} - \sqrt{(X_1 - x)^2 + (Y_1 - y)^2} \end{cases} \tag{2.9}$$

Ovvero si ottengono le equazioni di due iperboli che presentano i fuochi nei punti degli apparati di rete. Dall'intersezione fra le due iperboli, ovvero dalla risoluzione del sistema espresso nella 2.9, risultano le coordinate (x, y) che rappresentano appunto la posizione dell'utente. Questo problema è noto come *Hyperbolic Lateration* o

Multilateration. Anche in questo caso è possibile estendere al caso 3D, considerando la quota.
La situazione del caso bidimensionale, è rappresentata nella figura 2.4.

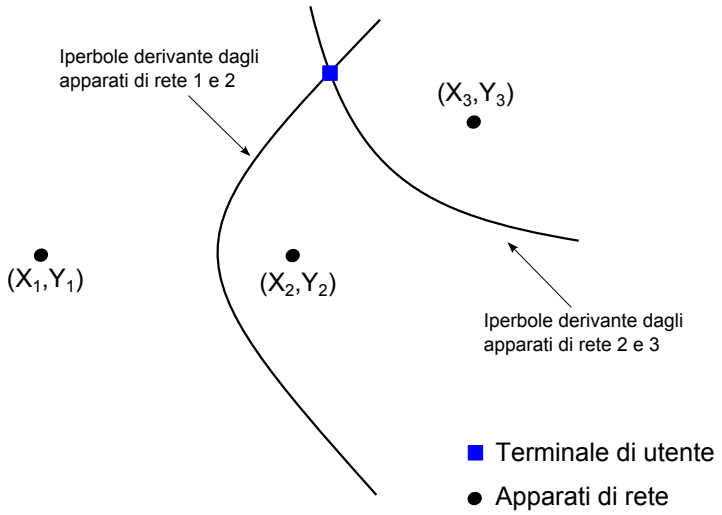


Figura 2.4: Multilateration: caso tipico della tecnica TDoA in 2D. La posizione dell'utente è data dal punto di intersezione fra le due iperboli

Analogamente a quanto fatto per il ToA, è possibile fare alcune considerazioni:

- Il terminale di utente deve essere in grado di comunicare con almeno tre apparati di rete, nel caso di posizionamento 2D, e con almeno quattro nodi di rete, nel caso di posizionamento 3D.
- Non è necessario conoscere l'istante di inizio trasmissione t_0 che viene "filtrato" dalla differenza.
- É necessaria la sincronizzazione temporale tra tutti gli apparati di rete.
- La localizzazione, nel caso TDoA considerato, è esterna.

L'aggiunta di misurazioni supplementari, consentirebbe di migliorare l'affidabilità o ottenere una maggiore precisione attraverso metodi statistici (si veda Loschmidt, Gaderer e Sauter [8]).

2.2.3 Risoluzione temporale

Molti sistemi di localizzazione sono time based (ToA o TDoA), per cui è importante caratterizzare questi sistemi in termini di *capacità di risoluzione*, ovvero in termini della minima variazione temporale che può essere percepita dal sistema,

proprio perché tale risoluzione temporale determina l'errore introdotto nel processo di misurazione temporale, ovvero l'errore nel calcolo delle distanze e quindi l'errore sulla posizione dell'utente. Si possono avere sostanzialmente tre categorie di sistemi:

- Narrowband
- Wideband
- Ultra Wideband

Nei sistemi *narrowband*, si utlizzano trasmissioni a banda stretta, e la stima del ritardo è effettuata stimando lo sfasamento che subisce il segnale durante la propagazione tra la trasmissione e la ricezione. Per cui la risoluzione temporale dipende dalla precisione sulla misura dello sfasamento.

Nei sistemi *wideband*, si utilizzano segnali a banda larga sfruttando la tecnica del Direct Sequence Spread Spectrum (DS-SS); il ritardo di propagazione viene stimato effettuando la cross-correlazione fra il segnale ricevuto e delle sue repliche che sono generate localmente. In tal caso, la risoluzione temporale dipende approssimativamente dal tempo di chip che coincide circa all'inverso della banda utilizzata dal segnale. Questo è il caso delle reti wi-fi.

I sistemi *ultrawideband* utilizzano dei segnali impulsivi non modulati (a differenza dei segnali narrowband). In questi sistemi la risoluzione temporale è data approssimativamente dalla durata dell'impulso (che è pari a circa l'inverso della banda).

2.2.4 Angle of Arrival

Nei casi ToA e TDoA si sono utilizzate tre misurazioni temporali (per il posizionamento 2D) e la conoscenza dei tre punti di riferimento, ovvero delle coordinate degli apparati di rete, per determinare la posizione dell'utente. Il problema, in termini geometrici, è quello dell'intersezione tra circonferenze (ToA) e iperboli (TDoA), e tali curve sono il risultato della "traduzione" da misure temporali in distanze.

Con l'approccio **AoA** ([5–7]) si "sfrutta" sempre un principio geometrico, ma si considerano le direzioni di arrivo, ovvero il sistema è basato su misurazioni angolari. Il problema geometrico è l'intersezione tra due rette con angolazioni di Φ_1 e Φ_2, rispetto all'asse delle x (vedi figura 2.5). In questo caso si parla di *Triangulation*.

Analiticamente il problema è quello della risoluzione del sistema di equazioni espresso dalla 2.10:

$$\begin{cases} y = x \tan \Phi_1 + Y_1 - X_1 \tan \Phi_1 \\ y = x \tan \Phi_2 + Y_2 - X_2 \tan \Phi_2 \end{cases} \tag{2.10}$$

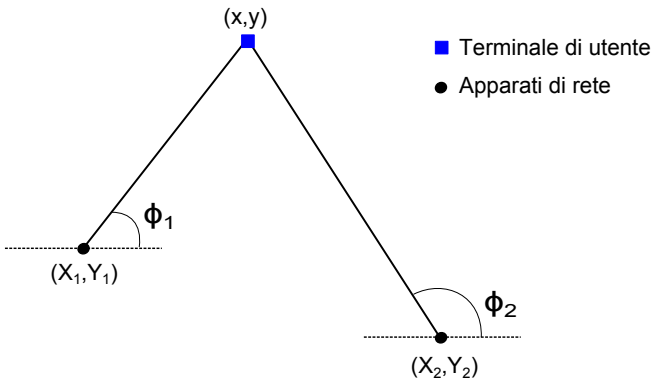


Figura 2.5: Triangulation: caso tipico della tecnica AoA La posizione dell'utente è data dal punto di intersezione fra le due rette

É possibile fare le seguenti considerazioni riguardo il sistema AoA:

- Le misurazioni angolari sono normalmente svolte dagli apparati di rete; ovvero, una coppia di apparati di rete stima la direzione da cui proviene il segnale irradiato dal terminale di utente e, sulla base di tali misurazioni, risolve l'algoritmo di stima della posizione e comunica la posizione al terminale di utente.
- In linea teorica, sono sufficienti due sole misure di angolo per identificare univocamente nel piano la posizione del terminale di utente. Nel caso di misurazioni temporali, il terminale di utente, doveva invece "vedere" almeno tre apparati di rete.
- Non è richiesta alcuna sincronizzazione tra gli apparati di rete.
- Analogamente a quanto detto per la risoluzione temporale dei sistemi TDoA e ToA, anche i sistemi AoA possono essere caratterizzati in termini di risoluzione angolare; in particolare tale valore coincide alla minima distanza angolare rilevabile. A partire dalla risoluzione angolare è possibile capire la precisione nella misura dell'angolo e quindi l'eventuale errore nella stima della posizione dell'utente.
- La maggiore complessità di tale tecnica risiede nella misurazione degli angoli, ovvero nella complessità delle antenne che si traduce in costi elevati. Esistono molte tecniche di misurazione angolare, ma in pratica è possibile ricondursi a due categorie (Mao e Fidan [4]): tecniche che sfruttano la risposta in ampiezza

dell'antenna ricevente e quelle che usano la risposta in fase. Nella prima categoria rientrano, le antenne che effettuano una scansione (rotazione meccanica o elettronica). Tali antenne presentano un certo diagramma di irradiazione caratterizzato da una direzione di massimo, quando tale direzione coincide con la direzione di arrivo del segnale, si ha la misura dell'angolo. L'accuratezza che si vuole ottenere dipende fortemente dalle dimensioni dell'antenna L e dalla lunghezza d'onda λ (infatti la larghezza del fascio dipende dal rapporto λ/L). Alternativamente è possibile utilizzare più antenne e la direzione di arrivo del segnale è valutata confrontando l'intensità del segnale ricevuto su ogni antenna (con la conoscenza a priori dei diagrammi di irradiazione delle antenne). Per quanto riguarda la seconda categoria, si sfruttano due o più antenne (array di antenne) poste a distanza nota e si valuta lo sfasamento del segnale alle varie antenne. Se si volesse integrare un infrastruttura esistente basata su WLAN, sarebbe necessario utilizzare dell'hardware aggiuntivo, ovvero un opportuno array di antenne presso l'AP.

- É richiesta la condizione di line-of-sight (LOS) fra terminale e apparati di rete per ottenere una stima sufficientemente accurata. Inoltre, tale tecnica è fortemente influenzata da multipath e shadowing.

2.3 Proximity System

Con il termine "Proximity System" si fa riferimento a tutti quei sistemi di localizzazione che sono in grado di determinare qual'è l'apparato di rete che si trova più vicino al terminale di utente, e questa, è l'unica informazione che viene utilizzata per la localizzazione dell'utente. Ogni nodo di rete può essere visto come un sensore che copre una certa zona mediante segnale radio, infrarossi o ultrasuoni e quando, terminale e sensore di rete si vedono, si assegna al terminale la posizione in relazione al nodo in vista. In particolare, le coordinate dell'apparato di utente coincideranno o con quelle del nodo di rete più vicino, oppure con un punto che si trova nella sua zona di copertura. I nodi di rete emettono periodicamente dei segnali includendo il loro ID. Per cui con il "metodo di prossimità" gli apparati di rete o i terminali di utente non devono calcolare nessun parametro e non ci saranno quindi problemi legati alla sincronizzazione e precisione nelle misurazioni. Con il procedimento, appena descritto si prende in considerazione un unico nodo di rete.

Data la semplicità dei proximity system, negli ultimi anni si sono sviluppate molte tecnologie basate sul concetto di prossimità (si vedano Yunhao e Zheng [9], Eladio et al. [7]), estendendo questo principio di base e considerando il caso in cui una certa zona è "coperta" da più nodi di rete. In generale, in un sistema di prossimità,

le ”variabili” sono: l’area di copertura di ogni nodo di rete (legame con il diagramma di irradiazione dell’antenna) e la disposizione di tali nodi; dal dimensionamento di questi due parametri, scaturisce la sovrapposizione fra le aree di copertura dei vari nodi, e a questo punto si utilizzerà un certo algoritmo per stimare la posizione dell’utente, che meglio si adatta alla situazione. Considero ad esempio la tecnica del centroide: la posizione dell’utente (nel caso 2D) può essere calcolata come il baricentro del poligono i cui vertici sono i nodi di riferimento, questa situazione è rappresentata in figura 2.5.

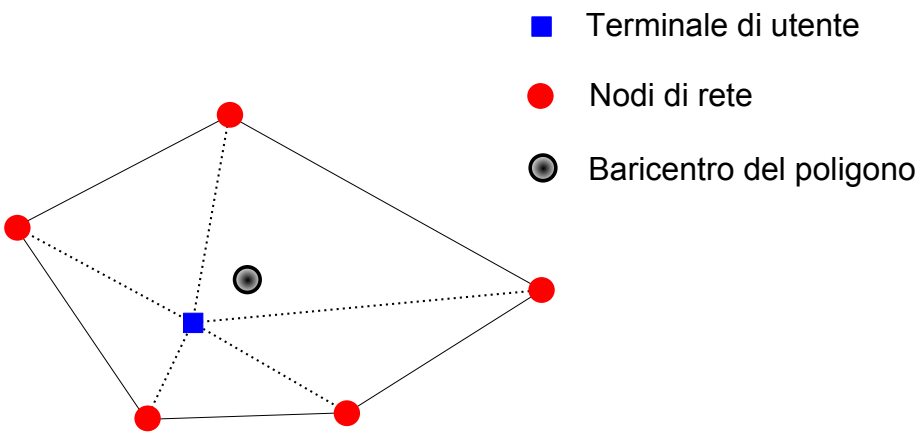


Figura 2.6: Proximity System: La posizione dell’utente è calcolata come il baricentro del poligono

Esistono molte altre strategie: intersezioni fra triangoli, uso del centro di gravità tra aree sovrapposte o tecniche probabilistiche. Alcune di queste tecniche necessitano di un hardware specifico , ma sono in grado di ottenere una buona precisione. Il grosso vantaggio delle tecniche basate sulla prossimità è la semplicità, che in alcune tecnologie si traduce in costi bassi.

É intuitivo quindi, che l’accuratezza sul posizionamento sarà tanto maggiore quanto più ”fitta” sarà la copertura, ovvero l’errore medio sulla posizione dell’utente diminuisce al crescere dei nodi di rete.

2.4 Received Signal Strenght Indication

Con l'approccio geometrico, si è visto che i parametri utilizzati per la localizzazione sono le differenze temporali e gli angoli. Un approccio alternativo è quello che prevede l'utilizzo dell'intensità della potenza come parametro di localizzazione, in particolare si parla di tecniche *Received Signal Strenght Indication* (RSSI) che si basano sul fatto che l'intensità del segnale radio si attenua durante la propagazione e per questo motivo è possibile legare tale attenuazione ad una distanza e quindi alla posizione di un terminale (si vedano [9],[7],[10]). Sono possibili due diversi approcci basati sull'intensità della potenza:

- Propagation Based
- Fingerprinting

L'approccio *Propagation Based* si basa su dei modelli statistici, che cercano appunto di modellizzare la propagazione di un onda elettromagnetica, così da valutare la potenza ricevuta. Il caso più generale, è quello di propagazione nello spazio libero, in cui la potenza ricevuta da un terminale posto a distanza d dal terminale trasmittente vale:

$$P_r = \left(\frac{\lambda}{4\pi d}\right) P_t \, G_t \, P_r \tag{2.11}$$

Dove P_t è la potenza trasmessa, G_t e G_r sono i guadagni dell'antenna trasmittente e ricevente e λ è la lunghezza d'onda. Un modello di questo tipo è valido nello spazio libero e non rispecchia la situazione reale in molti ambienti, soprattutto in ambito indoor, in cui fenomeni quali shadowing, riflessioni, interferenze ed altri effetti modificano fortemente la propagazione rispetto al caso nello spazio libero, e quindi la potenza ricevuta. Possono essere utilizzati altri modelli che descrivono meglio l'attenuazione del segnale, che utilizzano distribuzioni log-normali o variabili calcolate empiricamente; alcuni modelli noti (si veda Mao e Fidan [4]) sono: il modello Longley-Rice, il modello di Durkin, Okumura, Hata... Non ci interessa trattare nel dettaglio nessun modello, ma a titolo di esempio si riportano i risultati ottenuti con un modello descritto in Yunhao e Zheng [9]:

$$P_r = P_0(d_0) - \eta 10 \log_{10}\left(\frac{d}{d_0}\right) + X_\sigma \tag{2.12}$$

$P_0(d_0)$ indica la potenza ricevuta ad una distanza di riferimento d_0, η indica l'esponente della path-loss, X_σ è una variabile che tiene conto degli effetti dovuti al

fading ed è una variabile aleatoria log-normale con varianza σ^2. Per il calcolo della distanza d si utilizzerà la stima a massima verosomiglianza. Si riporta direttamente il risultato:

$$\hat{d} = d_0 \left(\frac{P_r}{P_0(d_0)} \right)^{-1/\eta} e^{-(\alpha^2/2)(\sigma^2/\eta^2)} \tag{2.13}$$

Dove $\alpha = \ln 10/10$.

In molti casi, le tecniche che fanno uso dell' RSSI utilizzano un approccio completamente diverso, che viene indicato con il termine *fingerprinting*. Per poter applicare questo metodo è necessaria una analisi preventiva dell'ambiente. Si distinguono due fasi: la fase offline e la fase online.

Nella *fase offline*, la zona che si vuole coprire viene suddivisa da una griglia, caratterizzata da un certo numero di punti. Per ognuno di questi punti, viene creato un vettore di valori, chiamato appunto fingerprint, i cui elementi sono i valori RSS misurati rispetto ai diversi nodi di rete. Per ogni punto della mappa, le misurazioni vanno effettuate considerando diversi orientamenti tra trasmettitore e ricevitore, questo a causa dell'attenuazione prodotta dal corpo umano, che determina risultati diversi. Un parametro di cui tener conto in fase di misurazione è il tempo di osservazione o tempo di calibratura, che è l'intervallo di tempo in cui viene "osservata" l'intensità del segnale ricevuto in un certo punto, in generale al crescere del tempo di osservazione si riescono a raccogliere un maggior numero di campioni, e si avrà una migliore caratterizzazione statistica del fenomeno. Il tempo di calibrazione è dell'ordine di decine di secondi. I dati così ottenuti per ogni punto vanno memorizzati in un database.

Nella *fase online*, il terminale di utente misura i valori di RSSI. A tal proposito un altro parametro di interesse è il tempo impiegato per la misurazione da parte del ricevitore (tempo di analisi); questo parametro deve essere adeguatamente valutato, perché se risulta essere troppo basso, si avranno a disposizione un numero di campioni di intensità di potenza ricevuta troppo basso per eseguire un confronto "efficace" con i valori di potenza memorizzati nella fase offline e ciò potrebbe determinare un degrado delle prestazioni. Tempi di analisi tipici sono dell'ordine del secondo. In seguito verrà applicato un determinato algoritmo che sfrutta le misurazioni effettuate e la mappa dei valori RSSI registrati per determinare la posizione dell'utente. Gli algoritmi possono essere di natura deterministica o probabilistica. L'approccio fingerprinting presenta il grosso svantaggio, che le misurazioni offline effettuate sono adatte solo all'area studiata, per cui ad ogni nuovo progetto, e ad ogni aggiunta o

modifica dei dispositivi di rete le misurazioni andranno nuovamente svolte, inoltre questa tecnica risulta poco efficace in ambienti radio con caratteristiche altamente dinamiche. Comunque l'approccio fingerprinting trova ampio utilizzo perché è una soluzione dai costi non elevati dato che non è richiesto alcun hardware particolare. In generale le tecniche fingerprinting sono in grado di fornire delle prestazioni migliori, sotto il profilo dell'accuratezza sulla stima della posizione, rispetto alle tecniche propagation-based.

2.5 Errori di localizzazione

Nelle sezioni precedenti, si sono trattati argomenti quali sincronizzazione, risoluzione temporale e principi di funzionamento dei diversi sistemi di localizzazione, considerando solo in parte quelle che sono le problematiche che si riscontrano in un ambiente indoor reale e gli errori di varia natura che si commettono. In generale, i fattori che contribuiscono ad una errata stima della posizione dell'utente sono i seguenti:

1. errori causati dal multipath;
2. errori causati da imprecisioni nel sistema;
    - misura dei parametri di localizzazione;
    - algoritmi utilizzati;
    - posizione degli apparati di rete;
3. errori causati dalla propagazione NLos;
4. errori dovuti alla riduzione del fattore geometrico.

Il multipath è un fenomeno che va attentamente considerato, soprattutto in ambito indoor: in ricezione giungeranno tante repliche ognuna caratterizzata da un certo ritardo e attenuazione; si avrà quindi un interferenza fra i diversi cammini ricevuti e un oscillazione sulla potenza ricevuta, infatti si ha un attenuazione supplementare, che si aggiunge all'attenuazione in spazio libero, soggetta ad aleatorietà, che viene indicata con il termine fading. Il fading causa degli errori sulle misure di potenza, di distanza e angolo: una determinata misurazione non può essere effettuata se il segnale ricevuto è troppo basso, inoltre in presenza di una molteplicità di contributi ricevuti, si commettono degli errori sulle misure di tempi e angoli, poiché la potenza ricevuta è distribuita su un intervallo temporale/angolare più ampio. In generale, è possibile asserire che la precisione ottenibile con un generico sistema di localizzazione degrada, in misura che dipende dalla capacità del sistema di sopportare e gestire

cammini multipli. Tali capacità dipendono, dal tipo di sistema utilizzato: i sistemi narrowband sono particolarmente affetti da multipath; con i sistemi wideband, la banda cresce ed è possibile risolvere parzialmente il problema dei cammini multipli, con i sistemi ultra wideband si ha una buona immunità al multipath.

Ogni sistema è affetto da imprecisioni che dipendono da più fattori. Un fattore è ovviamente l'errore sulle misurazioni temporale, angolare o di potenza. Un altro errore è causato dall'algoritmo utilizzato per calcolare la posizione dell'utente; ad esempio nel sistema GPS il ricevitore non risolve in forma chiusa il sistema di quattro equazioni, ma utilizza una risoluzione numerica che sfrutta delle approssimazioni, che inevitabilmente generano degli errori. Inoltre, ci sono degli errori già "in partenza", come ad esempio, quelli relativi alle coordinate locali degli apparati di rete. Tali errori si ripercuoteranno, in maniera più o meno rilevante, sulla posizione dell'utente.

In condizioni NLos, ovvero in assenza di visibilità diretta fra trasmettitore/ricevitore, si commetterà sempre un errore sulle misurazioni. I metodi basati sulle misurazioni di direzioni risultano particolarmente inefficaci, proprio perché può succedere che nessun contributo ricevuto abbia una direzione di arrivo prossima alla direzione del collegamento. Anche sui metodi time-based si può commettere un errore grossolano sulla stima delle distanze, in particolare, in condizione NLos si avrà sempre una sovrastima delle misurazioni temporali (ovvero delle distanze) proprio perché tutti i cammini hanno percorso distanze maggiori rispetto al cammino LOS.

L'errore di posizione, oltre a dipendere fortemente dall'errore commesso sulla valutazione delle distanze (pseudo-distanze), dipenderà anche dalla geometria, ovvero dalla posizione dei diversi apparati di rete rispetto al terminale di utente

Considero, a titolo di esempio, un sistema di localizzazione bidimensionale che utilizza un approccio ToA, considero che sia presente un errore sul calcolo delle misure temporali, causato da uno o più fattori. La situazione è quella descritta nella figura 2.7.

La distanza vera tra i tre apparati di rete e il terminale di utente è pari a r_i, ma a causa di errori temporali di varia natura, si andranno a misurare delle pseudo-distanze pari a ρ_i. Da un punto di vista geometrico, non esiste nessun punto di intersezione fra le tre circonferenze, ma ci possono essere fino a un massimo di sei punti di intersezione fra coppie di circonferenze, per cui non è possibile stimare la posizione sulla base delle intersezioni fra cerchi. A questo punto è necessario applicare un certa strategia, per stimare una posizione univoca. Un approccio potrebbe essere quello di considerare che il punto di interesse sia nella regione comune a tutte e tre le circonferenze (questa zona è stata chiamata area di incertezza) e scegliere poi come posizione di stima, il baricentro di tale zone. Un altro possibile approccio è descritto nella figura 2.8: il punto di localizzazione, coincide con l'intersezione fra i tre segmenti di colore verde; ogni segmento è costruito a partire dai due punti di

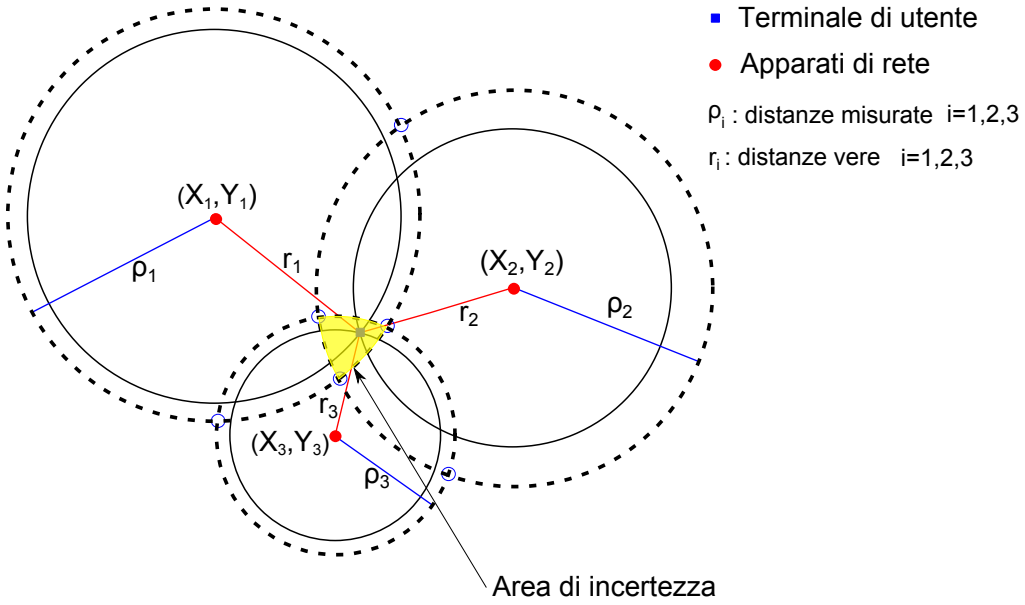


Figura 2.7: Caso reale ToA: non si ha un punto, ma un area di intersezione

intersezione fra le diverse coppie di circonferenze.

2.6 I sistemi di localizzazione esistenti

In questa sezione si descriveranno alcuni dei sistemi di positioning esistenti in ambito indoor; esistono sul mercato svariati sistemi che si basano su diverse tecnologie: suoni, ultrasuoni, infrarossi, Wi-Fi, ZigBee, Bluetooth, UWB, reti di sensori... Nel caso in cui si utilizzino delle infrastrutture dedicate, il terminale di utente deve essere equipaggiato con badge o tag proprietari in grado di "trattare" con i diversi tipi di segnali; questo è ad esempio il caso, delle tecnologie RF-ID, UWB o ultrasuoni. Se invece la localizzazione è realizzata usando delle reti Wi-fi o bluetooth, l'utente potrà utilizzare il proprio smartphone e specifiche applicazioni per finalità di positioning.

A questo punto è importante fare una distinzione in base agli *obiettivi finali del processo di localizzazione*, ovvero ha senso svolgere la localizzazione dell'utente, per due diverse finalità:

- legate al controllo di varco e al controllo di aree riservate;
- legate alla navigazione;

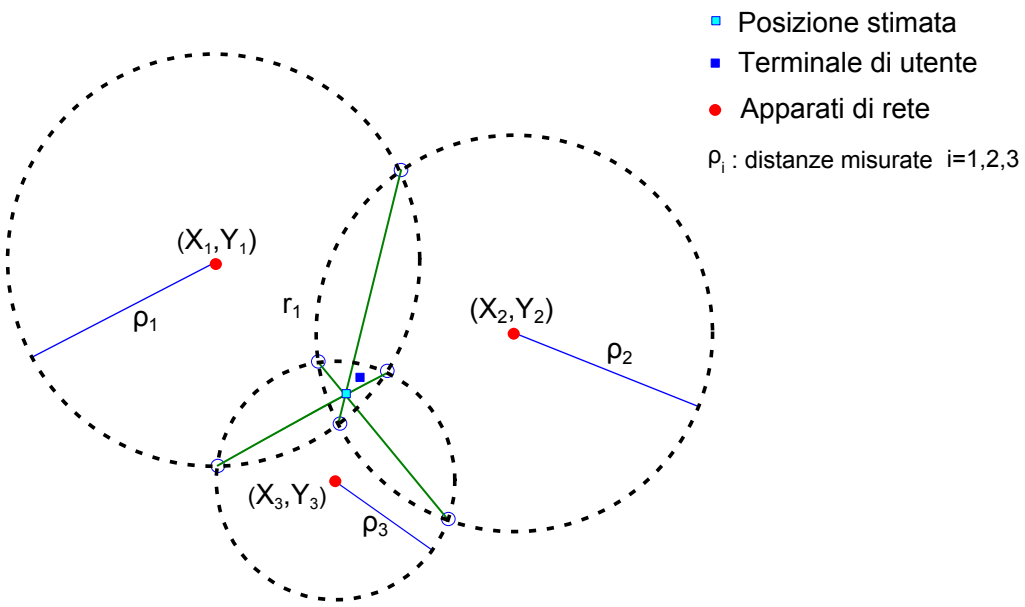


Figura 2.8: Caso reale ToA: calcolo della posizione dell'utente come intersezione fra rette costruite a partire dalle intersezioni tra le circonferenze

Il *controllo di varco* è effettuato per finalità di controllo accessi in zone riservate: un esempio pratico è all'interno di un ospedale, in cui si desidera che un medico abbia accesso in certi reparti, perciò non appena il medico si trova nei pressi dell'ingresso, l'apparato di legge il tag, indossato dall'utente e comanda l'apertura della porta. Più in generale, il controllo di varco può essere adoperato in tutte quelle sedi dove ci sono delle aree accessibili solo da alcuni utenti; altre possibili applicazioni sono l'identificazione veicolare (il sistema più conosciuto è telepass) o l'accesso ai magazzini. Il *controllo di aree riservate* invece è utilizzato per monitorare gli utenti o le cose in zone riservate; questo è ad esempio il caso di applicazioni aeroportuali, in cui si vogliono localizzare passeggeri e bagagli. Nei due casi descritti, l'utente non ha l'obiettivo di "posizionarsi", ma è il sistema che vuole localizzare l'utente; nel caso di controllo di varco, si vuole localizzare l'utente per permettergli di usufruire di servizi legati alla propria posizione, ovvero dell'identificazione a "mani libere", è possibile però anche svolgere dei controlli sugli accessi. Nel caso di controllo di aree riservate, si cerca di monitorare lo spostamento di cose e persone in strutture molto estese o complesse. In applicazioni di questo tipo, la *sicurezza* è un aspetto di fondamentale importanza. É diretto il legame con i proximity system.

Quando la finalità del sistema di localizzazione è la *navigazione*, è l'utente che

vuole posizionarsi, ovvero vuole sapere qual è la propria posizione all'interno della struttura, interagendo ad esempio con una mappa. Si possono aggiungere servizi di navigazione: assistono l'utente indicando interattivamente il percorso da seguire per raggiungere una destinazione impostata dall'utente; in questi sistemi gli aspetti legati alla sicurezza sono secondari. Si parla in questi casi di Real Time Location - RTL.

Nel seguito si descriveranno alcuni sistemi di localizzazione presenti sul mercato, dividendo la trattazione in questo modo:

- sistemi time based (ToA, TDoA)
- sistemi angle based
- sistemi RSSI

La divisione appena fatta è solo indicativa, nel senso che molti sistemi usano *approcci ibridi*, adoperando ad esempio sia approcci time based che angle based, oppure basati sulla potenza e sulle differenze temporali. Infatti, la tendenza è quella di sfruttare contemporaneamente diversi approcci, per aumentare al massimo la precisione della localizzazione e minimizzare gli eventuali effetti negativi di ogni approccio.

2.6.1 Sistemi Time Based

In linea teorica, con qualsiasi tecnologia è possibile misurare il Time of flight (una via) o il Round Trip Time (due vie), ma per finalità di positionig, si rendono indispensabili alcuni requisiti al fine di ottenere delle misurazioni con una precisione sufficiente per consentire un accurata stima delle distanze. In particolare, nei paragrafi precedenti si è sottolineata l'importanza della sincronizzazione: si dovrebbero avere precisioni dell'ordine dei ns per avere delle misure accettabili, inoltre nel pararagrafo 2.5 si sono descritte le problematiche in un ambiente reale. É possibile affiancare alle tecnologie a RF, dei segnali a ultrasuoni sfruttando la loro ridotta velocità di propagazione e quindi i minori problemi legati alla sincronizzazione, evitando quindi di usare costosi orologi (si veda Eladio et al. [7]). La tecnologia a ultrasuoni però, oltre a necessitare dell'hardware dedicato, comporta dei rischi dal punto di vista della sicurezza.

Le tecnologie maggiormente usate con i metodi time-based (ToA e TDoA) sono RFID, Wi-Fi e UWB, si possono però anche usare ultrasuoni, infrarossi e segnali laser.

I sistemi basati su **Ultra Wideband** stanno trovando negli ultimi anni un grande

sviluppo; infatti, come accennato nel paragrafo 2.2.3, l'impulso di brevissima durata, offre i seguenti vantaggi:

1. elevata immunità al multipath: infatti a causa della breve durata dell'impulso, i cammini multipli sono potenzialmente "risolvibili";

2. alta efficienza energetica che implica praticamente una lunga durata della batteria nei tag;

3. buone capacità di penetrazione dei materiali, almeno rispetto a sistemi a banda più stretta e ciò implica un aumento di probabilità, che in ricezione si abbia il contributo diretto;

4. alta risoluzione temporale che si traduce in errori sulle misure delle pseudo-distanze molto piccole;

5. elevata capacità dovuta alla larghezza di banda;

6. difficile da intercettare: infatti l'elevata larghezza di banda determina una densità spettrale di potenza molto bassa, per cui il segnale UWB risulta avere un intensità analoga a quella del rumore.

Il principio di funzionamento di un sistema di localizzazione UWB, basato su metodi time based, è il seguente: i terminali di rete, sono dei *tag attivi* che emettono dei segnali molto brevi di durate comprese tra i $100\,ps$ e $1.2\,ns$, su una banda operativa molto larga, ad esempio tra i $5,8$ e i $7.2\,Ghz$, (comunque su una banda maggiore di 500 Mhz), inoltre gli impulsi non sono modulati su una portante a RF in modo "classico", ma sono comunque modulati: in particolare si sfrutta una modulazione Pulse Amplitude Mode (variando l'ampiezza degli impulsi trasmessi) e Pulse Position Mode (variando la posizione sull'asse temporale degli impulsi trasmessi). Gli impulsi trasmessi dai tag, contengono l'identificativo del singolo tag ed altre informazioni accessorie. I tag trasmettono periodicamente impulsi di breve durata, e ciò implica un basso consumo energetico (durata della batteria per molti anni). Se (almeno) tre ricevitori UWB (sensori) ricevono gli impulsi irradiati dal tag, è possibile misurare il time of flight e mediante opportuni algoritmi, viene stimata la posizione dell'utente. A tal proposito, è possibile raggiungere risoluzioni temporali di $40\,ps$, che si traducono in errori sul calcolo del range di appena $12\,mm$. I tag possono inoltre disporre, di una convenzionale trasmissione radio in banda ISM (sui $2,4\,GHz$) che può essere utilizzata come canale di controllo e telemetria. Oltre ai tag e ai ricevitori UWB, è necessaria un unità centrale, chiamata hub, a cui sono connessi i ricevitori UWB e che deve poter connettere e gestire un elevato numero di ricevitori e deve essere in grado di calcolare la posizione dell'utente sulla base

delle informazioni fornite dai ricevitori. La tecnologia UWB si presta bene ad essere impiegata in applicazioni indoor, visto anche il corto raggio d'azione a causa delle limitazioni sulla potenza emessa.

Le grosse limitazioni di tale tecnologia sono legati principalmente agli elevati costi dell'infrastruttura di rete.

Alcune sistemi di localizzazione UWB sono i seguenti:

- Ubisense (www.ubisense.net)
- Uraxs (www.uraxs.com)
- Timedomain (www.timedomain.com)

Vediamo nel dettaglio alcune peculiarità del Sistema Ubisense UWB (si veda Srl [11]). In figura 2.9 sono mostrati i sensori e i tag Ubisense: sulla sinistra il sensore serie 7000 a destra Ubisense Slim Tag.

Figura 2.9: Sensori e Tag Ubisense

Per quanto riguarda i sensori Ubisense 7000:

- Utilizzano le tecnologie AoA e TDoA combinate, così da garantire una stima più affidabile sulla posizione dell'utente . Ogni sensore contiene un array di 5 antenne UWB e permette di calcolare sia l'azimut che l'elevazione del segnale UWB (tramite AOA), ottenendo un vettore per ciascun tag. Grazie all'array di antenne è possibile avere la stima della posizione anche utilizzando la ricezione su un unico sensore; ciò riduce i requisiti dell'infrastruttura, dato che

sarà possibile usare un minor numero di sensori. Inoltre si sfutta il TDoA: i sensori sono collegati fisicamente tra loro, tramite un cavo di sincronizzazione, il tag emette l'impulso che viene rilevato dai sensori, che possono usare la Multilateration per la stima.

- I sensori sono connessi tra loro, o con cavi standard Ethernet o tramite adattatori wireless Wi-Fi, utilizzando infrastrutture preesistenti.

- I tag e i sensori Ubisense sono in grado di comunicare tra loro sfruttando la banda ISM a 2.45 Ghz. Questa comunicazione permette di gestire in maniera ottimale i tag (informazioni di controllo e telemetria).

- ogni sensore utilizza una frequenza continua di aggiornamento di 160 Hz, ciò implica che ogni tag può essere visto ogni $1/160 = 6.25\,ms$

- I sensori includono algoritmi di filtering per i dati spaziali così da migliorare la precisione della localizzazione;

- Alcune specifiche tecniche dei sensori sono descritte in tabella 2.1.

Ubisense serie 7000	
Dimensioni	20cm x 13 cm x 6 cm
Peso	650 g
Range Operativo	>=160 m
Frequenze Radio UWB	6 GHz ÷ 8 GHz
Frequenze Radio Telemetria	2.45 GHz
Condizioni Operative	-20°C - 60 °C

Tabella 2.1: Specifiche tecniche del sensore Ubisense serie 7000

Vediamo adesso alcune caratteristiche dei tag Ubisense:

- il tag trasmette impulsi radio in modalità UWB; tale comunicazione è monodirezionale.

- I tag sono provvisti di pulsanti e led che permettono all'utente di interagire direttamente con il sistema.

- In una tipica applicazione in cui un tag viene utilizzato per individuare un lavoratore ogni 3 secondi, la batteria ha una durata media di 4 anni.
- Alcune specifiche tecniche dei tag sono descritte in tabella 2.2.

Ubisense serie 7000	
Dimensioni	83 mm x 42 mm x 11 mm
Peso	32 g
Frequenze Radio UWB	6 GHz ÷ 8 GHz
Frequenze Radio Telemetria	2.45 GHz
Update rate	0.00225 Hz ÷ 33.75 Hz
Condizioni Operative	-20°C - 60 °C

Tabella 2.2: Specifiche tecniche del tag Ubisense Slim

Altre caratteristiche del sistema Ubisense sono le seguenti:

- Possibilità di effettuare le simulazioni sul comportamento del sistema (tag e sensori), in ambienti Windows e Linux, così da progettare ed ottimizzare la copertura;
- In quegli ambienti in cui i costi di cablaggio sono elevati o là dove il cablaggio è difficile da eseguire, è possibile adoperare degli appositi moduli: Network e Timing Conbiner (NTC). Questi dispositivi, connessi ai sensori, permettono di ridurre il numero di cavi.
- Gli strumenti di calibrazione Ubisense consentono di visualizzare graficamente tutte le letture dei sensori in 2D e 3D.
- Con il sistema proposto da Ubisense è possibile raggiungere una precisione di rilevazione di 10 cm in 3D.

Un'altra tecnologia che è stata oggetto di molti studi negli ultimi anni è il **wi-fi** basato sullo standard IEEE 802.11. Gli apparati appartenenti a questa tecnologia sono sostanzialmente dispositivi a basso costo. Sono stati fatti molti studi a riguardo nell'ultima decade, vista la massiccia diffusione del Wi-Fi. In Gunther e Hoene [12], viene mostrato un algoritmo che consente di migliorare l'accuratezza sulle misure. In particolare, questo approccio utilizza la misura del Round Trip Time e gli esperimenti sono stati eseguiti su schede 802.11g e 802.11e; si è ottenuto un errore sulle misure

di range minore di 8 m. Un differente approccio è descritto in Loschmidt, Gaderer e Sauter [8] in cui si utilizza una tecnica TDoA e dell'hardware dedicato per la sincronizzazione tra gli apparati di rete, ottenendo un elevata accuratezza. In ogni caso, le soluzioni di localizzazione basate su tecnologia wi-fi, sfruttano di solito approcci ibridi, ad esempio RSSI+time-based e per incrementare la precisione delle misurazioni si utilizzano soluzioni software.

Altra tecnologia utilizzata per scopi di positioning sono gli **ultrasuoni** (US). I segnali a ultrasuoni permettono di raggiungere un elevata precisione e non interferiscono con i segnali in RF. Però, come accennato in precedenza, il grosso problema è legato alla sicurezza e alla necessità di un infrastruttura dedicata. Nel seguito verranno descritti i due principali sistemi di localizzazione basati su US: Bat System e Cricket.

Nel *Bat System* (si veda Cambridge [13]) l'infrastruttura di rete, così come il terminale di utente (dedicato) utilizza i segnali a ultrasuoni su una frequenza di $433\,Mhz$ e le misure del time of fligth. L'apparato di utente è un trasmettitore US contraddistinto da un codice identificativo e che presenta una batteria con durata media di 18 mesi. L'infrastruttura di rete è invece costituita da una griglia di sensori US posti sul soffitto e da trasmettitori in grado di emettere segnali in RF. Il funzionamento è il seguente: l'unità centrale trasmette, per finalità di sincronizzazione, il segnale in RF che viene ricevuto sia dal terminale di utente, che dai sensori. Il terminale di utente non appena riceve il segnale RF, emette l'impulso US. Il tempo che intercorre fra la trasmissione del segnale US da parte del terminale di utente e la ricezione da parte dei sensori di rete è utilizzato per il calcolo della posizione dell'utente. Bat System è in grado di raggiungere accuratezze di 3 cm per il 95% del tempo, i maggiori problemi di localizzazione si hanno in corrispondenza di aree ad alto assorbimento, come ad esempio gli angoli di una stanza. Un aspetto di interesse è che possono essere utilizzati fino a un massimo di 15 sensori per la stima della posizione del terminale, ovvero 15 misurazioni. Il principio di funzionamento e gli elementi del Bat System sono visibili in figura 2.10.

Il sistema *Cricket* è analogo al Bat System, ma sono i sensori a trasmettere i segnali US. Perciò nel sistema Cricket i sensori sono trasmettitori, mentre nel Bat System i sensori agivano solo da unità riceventi. Ciò implica che nel sistema Cricket la localizzazione è eseguita a livello locale dal terminale di utente, mentre nel Bat System, la localizzazione era esterna. Grazie a questo approccio è richiesto un minor numero di sensori. Esistono diverse versioni del sistema Cricket: Grillo e Compass,

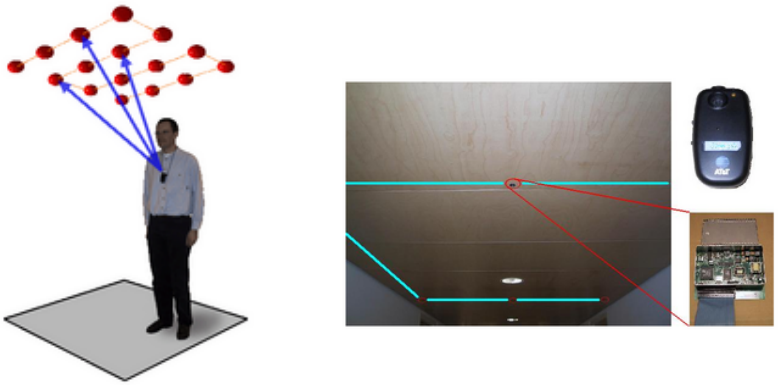


Figura 2.10: Bat System

ed è possibile raggiungere accuratezze fino a 1 cm.

Un altra possibile tecnica di localizzazione è quella che fa uso della **tecnologia laser**. In questi casi si parla di LIDAR, acronimo che sta per LIght Detection And Ranging. La tecnologia Laser è simile alla tecnologia radar, per cui la distanza dell'oggetto è determinata misurando il tempo trascorso fra l'emissione dell'impulso e la ricezione del segnale retrodiffuso. In realtà, questa tecnologia è maggiormente utilizzata in robotica piuttosto che in ambito di positioning indoor.

2.6.2 Sistemi RSSI

Nella prima parte di questa sezione ci soffermeremo su alcuni sistemi di localizzazione basati su wi-fi, in particolare che sfruttano l'approccio fingerprintg. Verrà descritto ampiamente, dapprima, il sistema RADAR, che rappresenta il capostipite dei sistemi wi-fi che sfruttano l'approccio fingerprintng e in seguito il sistema Ekahau, che attualmente rappresenta una delle soluzioni più complete sul mercato basate su wi-fi. In seguito verrà descritto un approccio ibrido che sfrutta sia la tecnologia Wi-Fi che la tecnologia Bluetooth. Infine, verrà descritto il sistema Landmarc che fa uso di un di un approccio propagation based.

RADAR (si vedano [14],[15]) è stato uno dei primi progetti completi di localizzazione e tracciamento basati su tecnologia wi-fi. Si basa sul fingerprinting. Merito dei progettisti non è stato solo quello di assicurare una buona accuratezza nella

localizzazione dell'utente, ma anche quello di fornire una documentazione chiara e completa che affronta tutte le problematiche di interesse. Nel progetto RADAR gli esperimenti sono stati condotti in un area abbastanza estesa, pari a $980\,m^2$ con più di 50 stanze e coperto da tre Base Station, come mostrato in figura 2.11. Il terminale utilizzato per le misurazioni è dotato di scheda NIC (Network Interface Card) Digital RoamAbout basata su tecnologia WaveLAN RF che permette di avere informazioni sulla potenza del segnale (SS - Signal Strength) e sul rapporto Signal-to-Noise Ratio SNR. La rete opera in banda ISM a 2.4 Ghz e presenta una velocità di 2 Mbps. Il clock del terminale e delle base station è sincronizzato (entro i 5 ms).

Le misurazioni caratteristiche della fase off-line sono state eseguite in questo modo: In determinati punti (bidimensionali) di interesse (x, y) vengono inviati da parte del terminale mobile 4 pacchetti UDP al secondo di 6 byte ciascuno di payload. Ogni BS registra l'intensità del segnale ss (signal strength) insieme al timestamp (t). Verrà memorizzata una terna di valori (t, BS, ss). É utile sottolineare che le misure relative al rapporto SNR non sono in pratica utilizzate a causa delle forti fluttuazioni dovute al rumore, quindi l'unica misurazione presa in considerazione è il ss. Inoltre, sono state registrati in 70 punti diversi i segnali che giungono al ricevitore dalle tre base station. In particolare, per ognuno dei punti di analisi (x, y), il terminale è stato considerato in 4 diverse direzioni (nord, sud, ovest, est), la direzione si è indicata con d. Infatti si è visto che la potenza del segnale in un determinato punto può variare fino a $5\,dBm$ in base alla direzione in cui è disposto il terminale. Ciò è dovuto alla posizione del corpo, che quando si interpone tra BS e terminale, genera un attenuazione supplementare che incide notevolmente sulle potenze ricevute. Per ognuna delle 70 posizioni (x, y) e per ognuno dei quattro possibili orientamenti t sono stati raccolti 20 campioni di potenza del segnale, tutte queste informazioni sono legate al timestamp t. In seguito si sono unite le informazioni raccolte durante la fase off-line: sono state calcolate per ogni terna di valori (x, y, d) la media, deviazione standard e mediana dei valori di ss per ogni BS. Per il calcolo della posizione dell'utente, si è considerato dapprima l'algoritmo *Nearest Neighbor in Signal Space (NNSS)*: Con l'algoritmo NSSS viene valutata la distanza, nello spazio del segnale, tra l'insieme di misure osservate (SS_1, SS_2, SS_3) e tutti gli altri ss registrati $(ss_{i1}, ss_{i2}, ss_{i3})$ (70 posizioni), e poi scegliere la posizione (x, y), legata ai ss registrati, che minimizza tale distanza. Quanto detto è indicato nella 2.14.

$$D_i = \sqrt{(SS_1 - ss_{i,1})^2 + (SS_2 - ss_{i,2})^2 + (SS_3 - ss_{i,3})^2} \qquad i = 1, 2, ..., 70 \quad (2.14)$$

Alla distanza D_i più piccola risultante, sono "legate" le coordinate (x, y), legate a loro

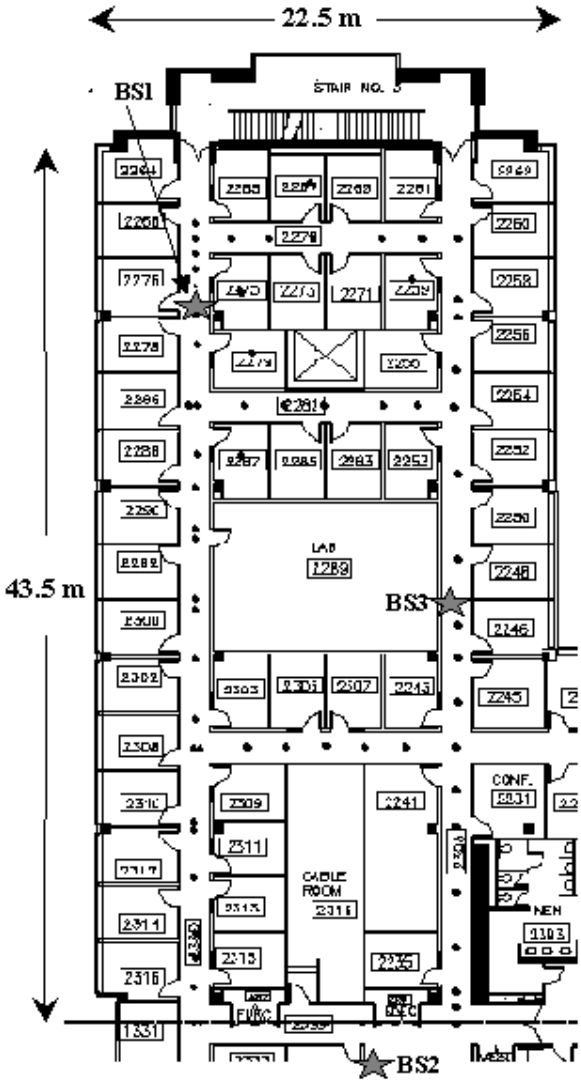


Figura 2.11: Mappa degli esperimenti del progetto RADAR

volta ai valori $(ss_{i1}, ss_{i2}, ss_{i3})$. Con l'approccio descritto, la posizione scelta per la localizzazione dell'utente può essere solo tra quelle della griglia, per cui l'accuratezza che si ottiene dipenderà dalla granularità della griglia utilizzata nella fase off-line. In seguito si è considerato una sorta di "evoluzione" di questo algoritmo, chiamato *Multiple Nearest Neighbors*; questo algoritmo è anche indicato in letteratura con il nome di *Nearest Neighbor in Signal Space Average (NNSS-AVG)* (si veda [16]): al posto di considerare solo la distanza più piccola, ovvero solo una coppia di coordinate (x, y) che genera la distanza D_i minore, si considerano le k coordinate più vicine nello spazio del segnale, che generano le k distanze più piccole. Operando la media di queste k coordinate (equazione 2.15) si ottiene il posizionamento finale. Un esempio è mostrato in figura 2.12 in cui si considerano $k = 3$ nearest neighbors.

$$(x, y) = \frac{1}{k} \sum_{i=1}^{k} (x_i, y_i) \tag{2.15}$$

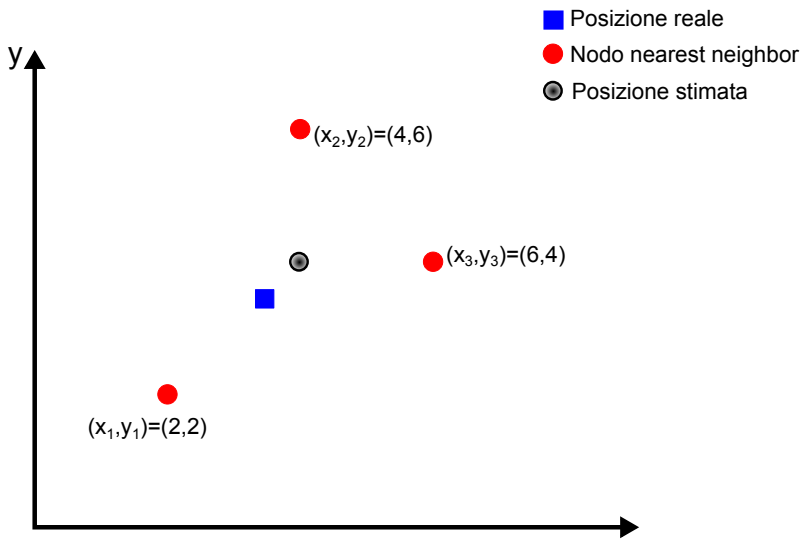


Figura 2.12: Algoritmo NNSS-AVG

I risultati ottenuti con questo approccio empirico sono stati confrontati con altri due metodi:

- Strongest BS: si sceglie la posizione dell'utente considerando solo il segnale della BS più forte.

- Random: La posizione dell'utente è scelta casualmente, indipendentemente dai valori *ss*.

Le prestazioni ottenute sono mostrate in figura 2.13 in cui si confrontano la cumulative distribution function (CMD) nei tre casi. In tabella 2.3, sono invece riassunte le informazioni contenute nella figura in termini di valori del 25°, 50° (mediana) e 75° percentile dell'errore in distanza con i tre approcci considerati.

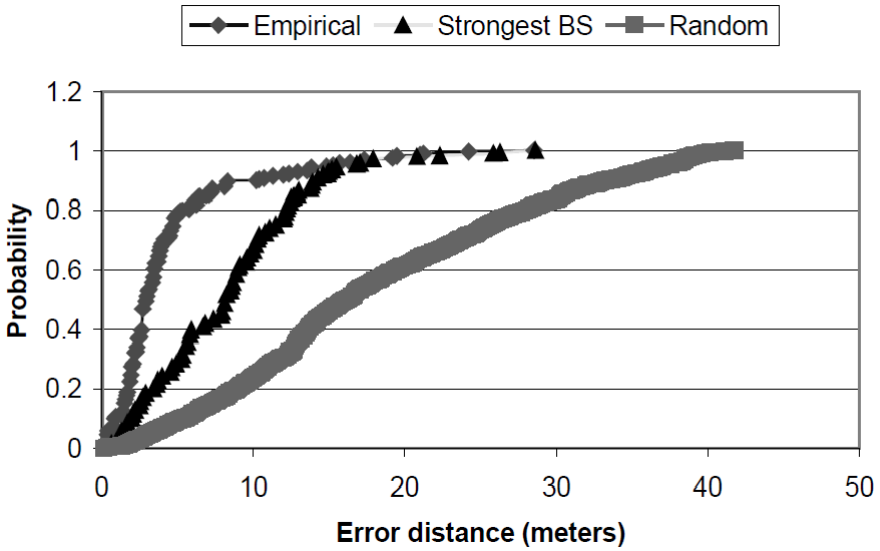


Figura 2.13: cumulative distribution function dell'errore sulla stima della posizione - confronto fra i tre metodi

Metodo	25^{th} (metri)	50^{th} (metri)	75^{th} (metri)
Empirico	1.92	2.94	4.69
Strongest BS	4.54	8.16	11.5
Random	10.37	16.26	25.63

Tabella 2.3: 25°, 50° e 75° percentile dell'errore in distanza sulla localizzazione

Un risultato di notevole interesse è quello mostrato in figura 2.14: gli errori minori in distanza, relativi al 25° e 50° percentile si hanno applicando l'algoritmo NNSS-AVG rispettivamente con k=2 e k=3 nearest neighbors. Inoltre, risulta che l'errore

in distanza che si ottiene decresce al crescere del numero di campioni considerati per ogni posizione.

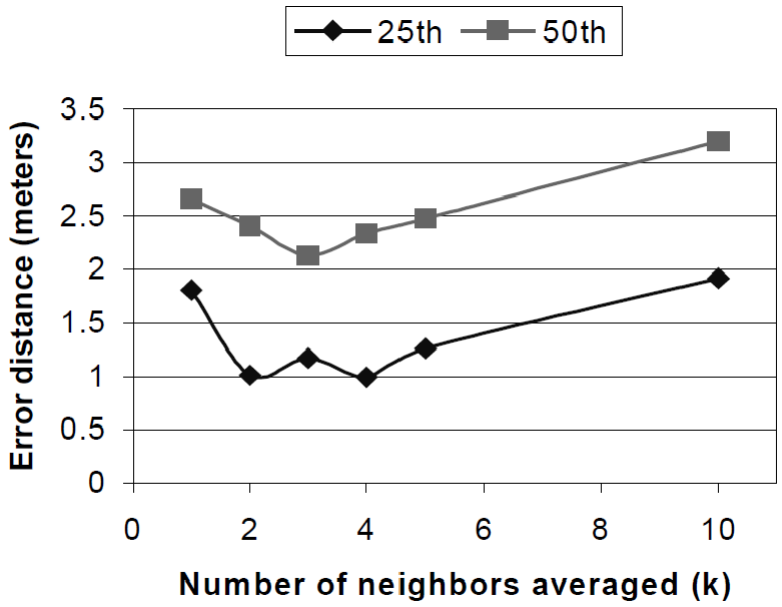


Figura 2.14: Prestazioni del sistema RADAR al variare del numero k di nearest neighbors considerati nei casi di 25° e 50° percentile

Altro sistema di localizzazione basato su tecnologia **Wi-Fi** con un approccio fingerprinting, che rappresenta uno dei principali punti di riferimento, è il sistema Ekahau (si veda [17]). Questo sistema non richiede alcun hardware specifico ed è in grado di localizzare terminali di utente di vario tipo, dai PDA ai TAG specifici Ekahau. il sistema Ekahau propone diverse soluzioni per effettuare ricerche su oggetti o persone, per gestire allarmi, per funzionalità di messaggistica tra utenti (sfruttando la rete wi-fi) o per funzionalità di tracking e mapping in real time, in particolare si parla di RTLS, ovvero Real Time Location System. Vediamo alcune elementi del sistema:

- Tag Wi-Fi: 10 diversi tipi di tag per i diversi casi d'uso, tutti caratterizzati dalla possibilità di comunicare bidirezionalmente e con un elevata durata della batteria.

- Ehahau RTLS Cootroller: è l'unità centrale del sistema: contiene gli algoritmi di elaborazione e gestisce e controlla il sistema

- Ekahau Site Survey: interfaccia per le diverse funzionalità: tracking, mapping, allarmi, comunicazioni.

In condizioni normali è possibile ottenere precisioni da 1 a 3 metri.

Esistono altri sistemi di localizzazione che sfruttano il wi-fi con un approccio fingerprinting; si citano anche i sistemi Horus e Bussola.

L'approccio fingerprinting può essere utilizzato anche con la tecnologia **Bluetooth**, questo è il caso del sistema blueps (si veda Rodriguez, Pece e Escudero [18]). Anche in questo caso esiste una fase on-line e una off-line analogamente al sistema RADAR . In particolare è possibile ottenere un errore sulla localizzazione di 1,2 metri nel 79% dei casi, avendo in vista tre sensori. Il bluetooth è una tecnologia a bassissimo costo nata per connettere le periferiche senza l'utilizzo dei cavi e che presenta delle basse velocità (rispetto alle moderne reti wi-fi), inoltre è una tecnologia a corto raggio (Personal Area Network - PAN). Dalle considerazioni appena fatte sul sistema bluetooth scaturiscono due importanti considerazioni:

1. Per sole finalità di positioning , la velocità di trasferimento non è un parametro essenziale, mentre i costi rappresentano un elemento fondamentale.

2. Il fatto che si abbia corto raggio, se da una parte è un aspetto "negativo" perché non permette l'uso su wide area, dall'altra parte può essere visto come un vantaggio perché si riduce il margine di errore nel caso in cui il dispositivo sia sotto la copertura di un unico nodo di riferimento.

Altro sistema di notevole interesse è quello proposto in Mahtab Hossain et al. [19]: si utilizzano le tecnologie Wi-Fi e Bluetooth con un tecnica fingerprinting. L'aspetto di maggiore interesse nell'approccio proposto è che si sfrutta il fingerprinting basato sulla *Signal Strength Difference* (SSD), ovvero sulle differenze dell'intensità di potenza percepite. Con questo concetto non si utilizzano le potenze "assolute" (RSS) e quindi si rende indipendente il sistema dal livello di potenza, che varia per diversi dispositivi. Infatti, nello scenario considerato e durante la fase off-line, si sono valutate, le potenze ricevute da due access points (con la medesima tecnologia) da parte di un terminale mobile (PDA o laptop). Il parametro preso come riferimento è la differenza di questi due livelli di potenza. Gli esperimenti, in fase off-line, sono stati condotti su 20 diversi punti e i risultati dell'esperimento condotto sono mostrati nelle figure 2.15 e 2.16.

Dalla 2.15 si vede come i livelli di potenza ricevuti da un access point siano diversi, sia nel caso wi-fi che in quello bluetooth, utilizzando, dallo stesso punto, due

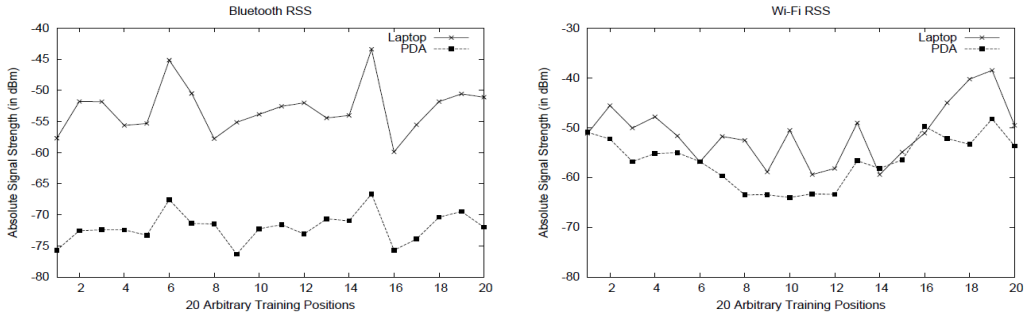


Figura 2.15: RSS percepito su un access point Bluetooth e su un access point Wi-Fi

dispositivi diversi: Laptop e PDA. Per cui, in generale è possibile asserire, che con lo scenario considerato non è possibile, sfruttare le misure RSS per la localizzazione dell'utente.

Dalla figura 2.16 si vede invece come i livelli di potenza SSD, nei casi di Laptop o PDA, si eguaglino.

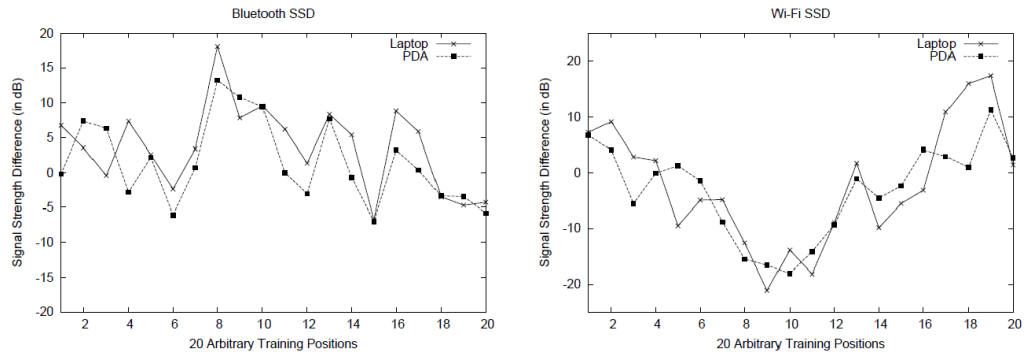


Figura 2.16: SSD percepito tra una coppia di access point Bluetooth e access point Wi-Fi

L'ultimo sistema analizzato è il sistema Landmarc (si veda [20]) basato su tecnologia RFID (Radio Frequency IDentification); questo sistema sfrutta un approccio propagation based, per cui, a differenza dei sistemi descritti in precedenza, non è prevista alcuna fase off-line.

Innanzitutto facciamo una brevissima introduzione alla tecnologia RFID; in un sistema RFID si hanno tre elementi fondamentali:

- Reader: sono gli elementi dell'infrastruttura di rete (apparati di rete); sono

dei ricetrasmettitori in grado di interrogare e ricevere informazioni in risposta dai tag;

- Tag: è l'apparato di utente, ne esistono di due tipi:
  - passivi: ricavano l'energia per il funzionamento dal segnale irradiato dal Reader; i tag passivi reirradiano il segnale del Reader. le distanze a cui possono operare sono dell'ordine di pochi metri al massimo.
  - attivi: sono dotati di ricevitore e trasmettitore e sono alimentati da batteria. Possono operare a distanze anche molto elevate, nell'ordine di decine o centinaia di metri.
- sistema centrale: è il sistema a cui sono connessi i Reader e che è preposto alla gestione del sistema e alle operazioni di localizzazione .

Il sistema Landmarc (LocAtioN iDentification based on dynaMic Active Rfid Calibration), è stato pensato così da ridurre il numero di readers, che rappresentano il costo principale nei sistemi RFID; per questo scopo, si utilizzano come apparati dell'infrastruttura di rete, oltre ai suddetti reader, anche dei tag attivi, che sono collocati in posizione nota all'interno dell'area di interesse, questi tag vengono infatti chiamati "reference tag". Questi tag di riferimento, presentano un raggio di lettura di circa 150 metri, che può essere eventualmente incrementato a 1000 metri con l'aggiunta di antenne specifiche. Oltre al vantaggio economico, l'utilizzo dei reference tags consente di compensare eventuali variazioni ambientali e di migliorare l'accuratezza di stima di localizzazione. Ogni reader è in grado di gestire fino ad un massimo di 500 tag e le aree di copertura dei diversi reader possono essere anche differenti (si veda la figura 2.17). Il tag del terminale di utente è invece chiamato tracking tag. La frequenza di funzionamento del sistema è di 308 Mhz ed è presente anche un interfaccia 802.11b per la comunicazione. Il sistema permette di localizzare in tempo reale la posizione di uno o più tag di tracking.

Si suppone di avere n readers, m reference tags e u tracking tags. Per ogni tracking tag si definisce il vettore dell'intensità del segnale (RSSI), $S = (S_1, S_2, \ldots, S_n)$, in cui gli elementi S_i, con $i \,\epsilon\, (1, n)$, denotano l'intensità del segnale del tracking tag percepito dall'i-esimo reader. Si costruiscono inoltre gli m vettori riferiti ai reference tag $\theta = (\theta_1, \theta_2, \ldots, \theta_n)$ dove θ_i indica sempre l'intensità della potenza. Per ogni tag p, con $p \,\epsilon\, (1, u)$, si definisce la distanza euclidea come segue:

$$E_j = \sqrt{\sum_{i=0}^{n} (\theta_i - S_i)^2} \qquad j \,\epsilon\, (1, m) \tag{2.16}$$

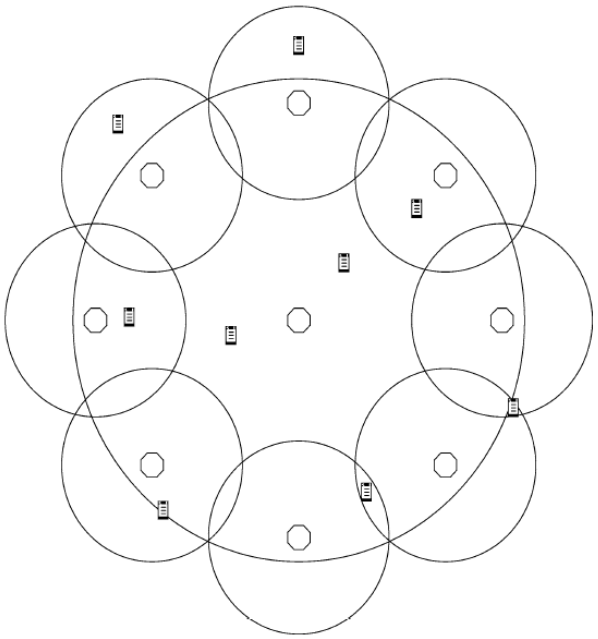


Figura 2.17: Esempio di posizionamento di 9 readers che presentano due differenti regioni di copertura

Avremo quindi m distanze euclidee riferite all'intensità del segnale: $E = (E_1, E_2 \ldots, E_m)$. Si scelgono le k distanze più piccole, e rispetto a queste, si determina la posizione del tag, come espresso nella 2.17:

$$(x, y) = \sum_{i=1}^{k} w_i(x_i, y_i) \tag{2.17}$$

In cui w_i sono i pesi attribuibili a ciascuna coppia di coordinate (x_i, y_i).
(x_i, y_i), sono le coordinate dei k tag di riferimento per i quali i valori della distanza euclidea E_i è il risultato minore rispetto a quello calcolato per gli altri m-k tag di riferimento. La scelta di tali pesi è un altro parametro di progetto. Ovvero ,si vuole che i pesi corrispondenti alle distanze euclidee più piccole, siano maggiori. Si è quindi utilizzata la seguente espressione per il calcolo di questi pesi:

$$w_j = \frac{1/E_i^2}{\sum_{i=1}^{k} 1/E_i^2} \tag{2.18}$$

Gli esperimenti sono stati condotti considerando $n = 4$ readers, $m = 16$ reference tags e $u = 8$ tracking tags, come indicato nella figura 2.18.

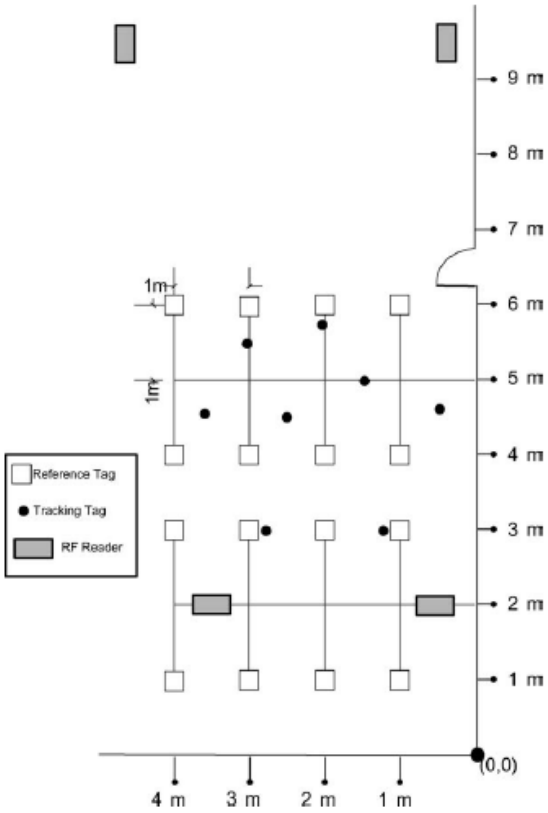


Figura 2.18: Sistema Landmarc: la mappa di riferimento con il posizionamento dei 4 readers, dei 16 reference tags e degli 8 tracking tags

Sono state effettuate una serie di misurazioni nello scenario descritto nell'arco di 48 ore, valutando di volta in volta l'errore commesso nel posizionamento. Indico con (x_0, y_0) le coordinate reali del reference tag e con (x, y) le coordinate stimate, l'errore commesso vale:

$$e = \sqrt{(x - x_0)^2 + (y - y_0)^2} \tag{2.19}$$

le prestazioni sono graficate nelle figure 2.19, 2.20 e 2.21.

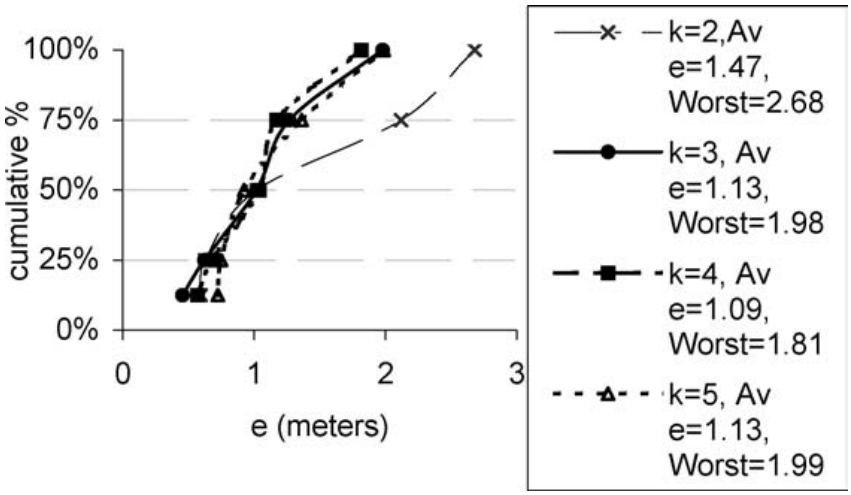


Figura 2.19: Sistema Landmarc: Errore percentile cumulativo sul posizionamento al variare delle k distanze euclidee scelte

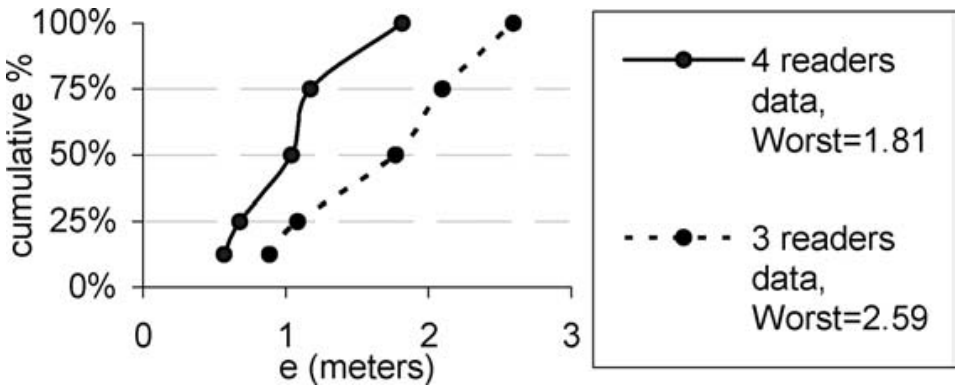


Figura 2.20: Sistema Landmarc: Errore percentile cumulativo sul posizionamento per n=3 e n=4 readers

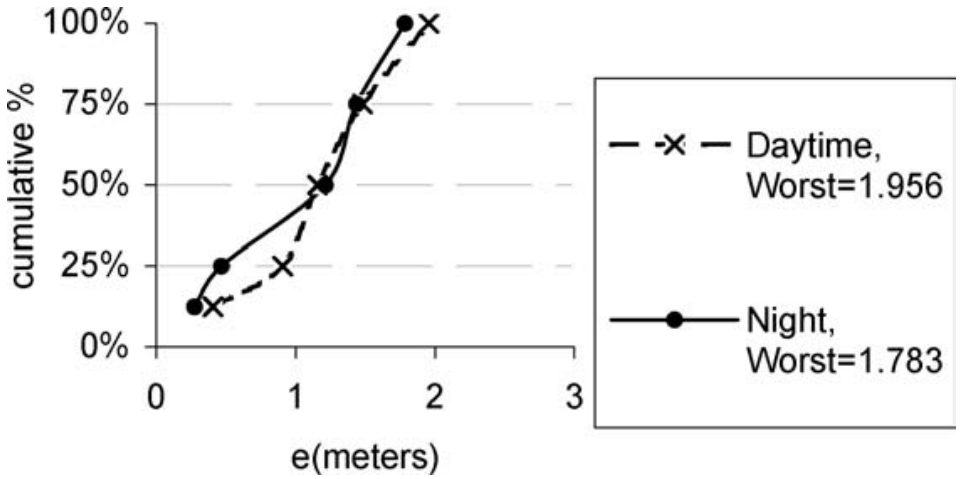


Figura 2.21: Sistema Landmarc: Errore percentile cumulativo sul posizionamento,valutato durante le ore diurne e notturne

Si nota che gli errori minori si commettono per k=3 e k=4, adoperando 4 readers e durante le ore notturne (a causa del minore affollamento)

2.7 Conclusioni

Un risultato evidente che emerge dall'analisi fatta è che le tecniche di localizzazione vanno innanzitutto "catalogate" considerando quelli che sono gli obiettivi finali del processo di localizzazione, in particolare se è l'utente a voler conoscere la propria posizione o è il sistema, che per scopi di sicurezza, ne richiede la localizzazione. In seconda analisi, si può considerare il fattore dei *costi* e dell'*accuratezza*. Considero questi due aspetti assieme, proprio per lo stretto legame, quasi lineare, che presentano nei sistemi indoor. In particolare, si è visto come tecnologie quali UWB ed RFID consentano di ottenere prestazioni molto elevate, sotto il profilo dell'accuratezza e robustezza, e tali prestazioni cresceranno sempre più, al crescere degli apparati di rete che si considerano. In ultima analisi si conclude con il punto di vista del "semplice" utente, il vero elemento centrale del sistema, che per scopi di positioning desidererebbe utilizzare sicuramente un terminale con cui ha "familiarità". Sistemi come Wi-fi e bluetooth, consentirebbero un uso di questo tipo, a fronte però di minori accuratezze e robustezza del sistema.

Capitolo 3

Dead Reckoning

Indice

3.1 Introduzione 43
3.2 Principi del Dead Reckoning 47
3.3 Sensori Inerziali 49
3.3.1 Accelerometro 50
3.3.2 Giroscopio 53
3.3.3 Sensore di campo magnetico 54
3.3.4 Orientamento 57
3.3.5 Errori nelle misure inerziali 61
3.4 Calcolo della distanza 66
3.5 Calcolo della direzione 77
3.6 Stima della nuova posizione 80
3.7 Dead Reckoning Test 82
3.8 Conclusioni 92

3.1 Introduzione

Il progresso tecnologico ha permesso di realizzare dispositivi mobili sempre più complessi, potenti e dotati di un maggior numero di funzionalità. Il classico telefono cellulare, si è evoluto sempre più avvicinandosi, in termini di prestazioni e funzionalità, ai PDA (Personal Digital Assistant). Il risultato di questa evoluzione tecnologica, ha creato una nuova categoria di dispositivi: gli smartphone. In pratica, PDA e telefoni cellulari sono destinati nel giro di pochi anni ad essere rimpiazzati dagli smartphone. La tendenza è quella di integrare all'interno dello smartphone un insieme di funzionalità sempre più ampio: oltre alle caratteristiche di un telefono cellulare e di un PDA, si sono aggiunti altri strumenti, quali lettore MP3, navigatore satellitare, fotocamera... oltre all'incredibile crescita hardware e software e all'inte-

grazione con gli strumenti prima menzionati, si iniziano a inserire anche strumenti alternativi quali accelerometri, giroscopi, bussole, magnetometri, sensori di prossimità... Questi sensori sono destinati, nel breve periodo, ad essere presenti in tutti gli smartphone e con la loro diffusione, come spesso capita, cresceranno le prestazioni. Il campo dei sensori di movimento sta vivendo in questi anni uno sviluppo parallelo incredibile, dovuto all'integrazione di tali sensori nei controller delle consolle di gioco. Vediamo alcuni esempi. Il controller della consolle Nintendo Wii (si veda [21]), chiamato Wii Remote, ha rappresentato al momento del lancio nel 2007, una vera e propria rivoluzione nel campo dei videogiochi; è considerato da molti come il principale elemento del successo della consolle. Il Wii Remote integra un semplice accelerometro prodotto dalla STMicroelectronics, che permette di misurare le accelerazioni prodotte dal movimento del controller da parte dell'utente. Visto l'enorme successo riscontrato, la Nintendo ha continuato a puntare sulla strada dei sensori di movimento, lanciando nel sul mercato, nel 2009, il Wii Motion Plus (in seguito chiamato Wii Remote Plus), un accessorio dotato di due giroscopi, uno per le alte velocità e uno per le basse velocità che si collega direttamente al Wii Remote (mediante la porta di espansione) e consente di ottenere una maggiore precisione nel rilevare i movimenti dell'utente. Altro controller lanciato sul mercato dalla Sony, nel 2010, è il Play Station Move dotato di 3 giroscopi, 3 accelerometri e un magnetometro. in figura 3.1 sono mostrati i due controller menzionati.

Figura 3.1: Play Station Move (sulla sinistra), Wii Remote e WiiMotionPlus

Un dato di notevole interesse, a confermare l'espansione del mercato dei sensori, è quello relativo al bilancio del 2010 della STMicroelectronics (si vedano [22],[23]), che è una delle aziende leder nel settore della produzione di componenti elettronici a microconduttore: I ricavi derivanti dal mercato dei sensori, ha raggiunto una delle

quote principali, ed ha superato gli introiti del settore Wireless.

É utile aggiungere, per completare il quadro, che agli smartphone, si è affiancata una "nuova" categoria, quella dei tablet che spesso presenta caratteristiche e funzionalità analoghe a quelle degli smartphone. Un elemento che ha contribuito in maniera determinante alla diffusione e allo sviluppo degli smartphone e dei tablet è senza dubbio la crescente disponibilità del traffico dati sia in termini di copertura, che in termini di convenienza per l'utente finale.

Nel Capitolo 2, si è fatta una panoramica sulle tecniche di localizzazione, cercando di sottolineare gli aspetti più rilevanti e alcune tecnologie di riferimento appartenenti al vasto mondo del positioning indoor. In figura 2.1 è stata fatta una (possibile) classificazione delle tecniche di localizzazione e nel paragrafo 2.1, si è parlato dell'infrastruttura di rete: tutte le tecniche descritte nel Capitolo 2, necessitano di un infrastruttura di rete, integrata o dedicata, basata su tecnicnologie Wi-Fi, Bluetooth, UWB... o che utilizzi un approccio geometrico, RSSI o di prossimità. Si è visto che tutte le tecniche descritte, sono caratterizzate da diverse accuratezze, raggi di copertura, costi di installazione e manutenzione, tipo di terminale di utente, robustezza, tempistiche di installazione... facendo un passo indietro, ci si può chiedere quale sia l'effettiva necessità di un utente in ambiente indoor e di cosa ha bisogno. La risposta a questa domanda, è dipendente dallo scenario in cui si colloca il sistema di posizionamento e quali sono gli obiettivi del positioning. Si può però fare un importante considerazione concettuale: l'utente o meglio un individuo, conosce ciò che può percepire con i cinque sensi e una "piccola" informazione aggiuntiva alle informazioni già note (percepite tramite i propri sensi), spesso può essere determinante e soddisfacente in tantissimi scenari: centri commerciali, supermercati, maxistore, ospedali, università, uffici pubblici... in queste applicazioni l'utente non ha bisogno di essere posizionato con estrema precisione; le destinazioni sono negozi, sportelli, aule, stanze o al più reparti di un maxistore, per cui quando l'utente si troverà nei pressi della destinazione non avrà bisogno di precisioni spinte. Ricollegandosi a quanto detto nel paragrafo 2.6, in buona parte delle applicazioni la cui finalità è la navigazione, è sufficiente fornire all'utente delle informazioni aggiuntive a quelle già percepite per fornire un servizio soddisfacente. Invece, in tutti quei sistemi di localizzazione il cui obiettivo finale è legato al controllo di varco o al controllo di aree riservate, è necessario che il sistema sia strutturato in una maniera più complessa e accurata. Un altro aspetto di fondamentale importanza è quello dei costi: nelle strutture menzionate non c'è una necessità reale a creare un sistema di localizzazione indoor, e ci si può "accontentare" della cartellonistica presente, perciò non si ha interesse ad investire su in infrastruttura dedicata per la realizzazione di un sistema

di positioning, questo perché il bilancio costi/benefici non è soddisfacente. Questa è una realtà riscontrabile quotidianamente: non esistono, ad eccezione di rarissimi casi, strutture del tipo sopramenzionato, che prevedono sistemi di localizzazione.

In questo Capitolo e nei prossimi verrà descritto **un diverso approccio alla navigazione in ambito indoor** basata sui concetti di **autolocalizzazione** e **dead reckoning**. Parlando di autolocalizzazione, scaturiscono due importanti considerazioni. In primo luogo, l'utente può svolgere un ruolo importante nel fornire al sistema dei *feedback* utili per la propria localizzazione, inoltre il sistema deve supportare degli *strumenti e funzionalità aggiuntive* così da permettere all'utente di trarre dei benefici nel processo di orientamento, ovvero fornendo delle informazioni aggiuntive a quelle già percepite. L'autolocalizzazione in questo lavoro di tesi è effettuata mediante *fotografia* (feedback) scattata ad un codice bidimensionale (per la precisione codice posizionale): l'utente si avvale del proprio smartphone per scattare una fotografia ad un opportuno **codice bidimensionale** situato in punto noto dell'area di interesse; in seguito l'applicazione decodifica la fotografia (in formato .jpg) e decodifica il codice bidimensionale. L'informazione contenuta dal codice bidimensionale è quindi legata alla coordinata e sarà quindi possibile localizzare l'utente, ovvero posizionarlo all'interno della mappa. A seguito dell'autolocalizzazione, potranno essere applicati i principi di dead reckoning, sfruttando i sensori inerziali dello smartphone, per la stima passo-dopo-passo della posizione dell'utente. Infine il sistema potrà essere integrato realizzando gli aspetti legati alla navigazione e a strumenti-funzionalità aggiuntive. L'obiettivo di questo lavoro di tesi, è quello di "gettare le basi" per la realizzazione del sistema di navigazione proposto analizzando le problematiche, i limiti, i punti di forza e tutti gli aspetti di interesse. L'obiettivo finale è quello di proporre questo sistema utilizzando come terminale di utente lo smartphone, e come interfaccia la relativa applicazione. I vantaggi dell'approccio descritto sono i seguenti:

1. Costo dell'infrastruttura e della manutenzione, praticamente nullo, legato alla sola installazione dei cartelloni o adesivi contenenti i codici bidimensionali nei punti di interesse.

2. L'utente non ha bisogno di nessun terminale dedicato, ma può avvalersi semplicemente del proprio smartphone corredato da un applicazione specifica: questo aspetto estende l'usabilità del sistema a tutti gli ambiti.

3. Il sistema è completamente autonomo.

4. Il sistema è passivo, ovvero non radiante, per cui non è soggetto ad emissioni di segnali che possano interferire con altri sistemi.

5. Il sistema non richiede una fase di studio preventiva all'installazione (come avviene ad esempio per il fingerprinting, per l'RFID...) e può essere utilizzato, senza alcuna limitazione o controindicazione in tutti gli ambienti, indipendentemente dalle condizioni operative.

In definitiva il sistema descritto rientra nella categoria dei *sistemi inerziali*. In questi casi si parla di INS (Inertial Navigation System), mentre per le misurazioni dei sensori, si parla di IMU (Inertial Measurement Unit);

In questo Capitolo verrà trattata la parte relativa al dead reckoning e ai sensori inerziali, con particolare attenzione al caso dell'iPhone 4, usato come dispositivo base nel corso di questo lavoro di tesi. In seguito, nel Capitolo 4, si affronteranno i temi relativi al processo di autolocalizzazione con particolare interesse al codice creato per questa applicazione: lo **SPinV Code**, la cui trattazione verrà completata nel Capitolo 5.

3.2 Principi del Dead Reckoning

Il termine "dead reckoning" deriva probabilmente da "deduced reckoning", ovvero stima derivata (si veda Groves [24]). Il dead deckoning è il processo mediante il quale è possibile stimare la posizione attuale a partire da una posizione nota in precedenza sfruttando un insieme di misurazioni effettuate da sensori inerziali. In sostanza, è necessario conoscere, oltre al punto di partenza, la distanza percorsa e la direzione in un certo intervallo temporale per stimare la posizione attuale del terminale di utente. Con un approccio geometrico, analogo a quello descritto in 2.2, risulterà che:
la distanza percorsa determina una circonferenza; conoscendo la direzione e il verso (orientamento), sarà possibile collocarsi nella circonferenza come mostrato in figura 3.2.

Il principale svantaggio delle tecniche basate su Dead Reckoning è che gli errori sulla posizione aumentano con il tempo, sia se il terminale di utente è fermo, sia se è in moto. Ciò è dovuto al fatto che la nuova posizione viene stimata solo sulla base della posizione precedente e quindi l'errore di stima, inevitabilmente crescerà. Per tale motivo se il sistema inerziale è usato per lunghi periodi di tempo, ha bisogno di essere rifasato, e il rifasamento avviene appunto mediante la fotografia al codice bidimensionale (concetto di autolocalizzazione).

In Randell, Djiallis e Muller [25], per valutare l'errore commesso, si considera un Ellisse di Confidenza (vedi figura 3.3 estratta da [25]) che rappresenta un ellisse di errore. Il principio è il seguente: P_0 rappresenta il punto di partenza. In un determinato intervallo temporale, vengono raccolte le misurazioni dei sensori inerziali e, sulla base di questi, si stima la nuova posizione $P1$. La posizione reale, sarà

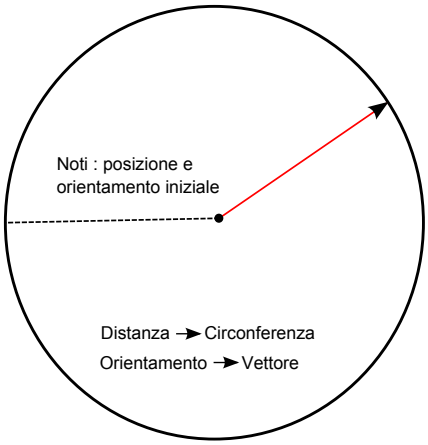


Figura 3.2: Rappresentazione geometrica del dead reckoning

intorno al nuovo punto stimato, in particolare sarà al 95% contenuta nell'ellisse di semiasse maggiore *ab* e semiasse minore *cd*. L'ampiezza del semiasse minore *cd* dipende dall'accuratezza sul calcolo della distanza (ovvero velocità e accelerazione), mentre l'estensione del semiasse maggiore *ab* dipende dall'accuratezza della stima della direzione. Ovviamente, tanto maggiore sarà l'errore sulle misurazioni e tanto più grande sarà tale ellisse di confidenza. $v1$ rappresenta il vettore di movimento stimato tra la posizione di partenza nota P_0 e la posizione stimata inerzialmente P_1. Quella appena descritta rappresenta la valutazione dell'incertezza dopo una stima; l'incertezza sulla posizione ottenuta dopo n passi viene calcolata come somma cumulativa di tutti gli errori commessi fino al passo n, come espresso nella 3.1:

$$P_n = P_0 + \sum_{i=1}^{n}(v_i + v_e) \tag{3.1}$$

In cui v_e rappresenta il vettore di errore ad ogni passo. Assumendo che l'utente si muova lungo un percorso rettileneo, l'ellisse di confidenza dopo n iterazioni è un ellisse i cui semiassi maggiore e minore valgono rispettivamente *nab* e *ncd*; in ogni caso l'ellisse cresce al crescere della distanza percorsa. Da quanto detto si capisce l'importanza sull'accuratezza delle misurazioni e sulla stima iniziale.

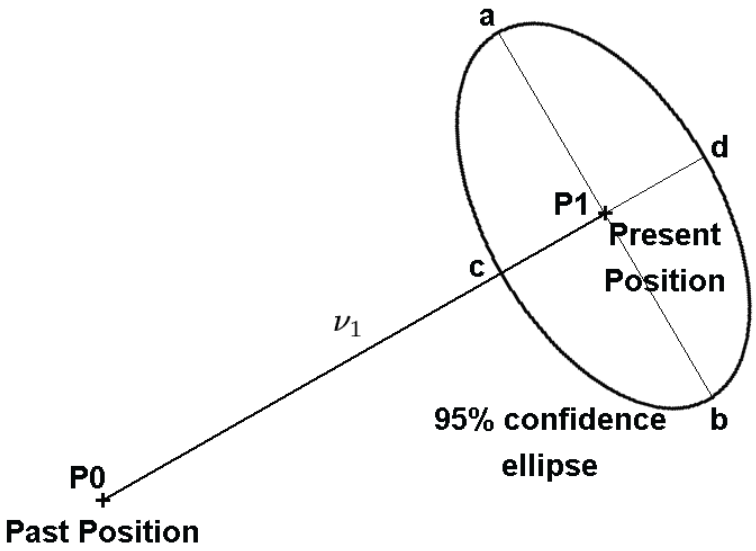


Figura 3.3: Ellisse di confidenza che contiene al 95% l'errore commesso

3.3 Sensori Inerziali

Nel seguito verranno descritti, i sensori presenti nei moderni smartphone, con particolare riferimento ai sensori in iPhone 4:

- accelerometro;
- giroscopio;
- sensore di campo magnetico (magnetometro);

É utile a questo punto, una caratterizzazione dei sistemi di riferimento utilizzati. Il sistema di riferimento su cui si effettua la navigazione è basato su coordinate locali, come detto nel paragrafo 2.1; in questo caso l'origine degli assi è considerata in punto arbitrario della mappa. Indico tale riferimento come sistema di riferimento della navigazione (sdr della navigazione). Altro sistema di riferimento è quello solidale al terminale di utente, ovvero allo smartphone, chiamato sistema di riferimento body (sdr body) o sdr inerziale (si veda la figura 3.4); tale riferimento quindi modifica la sua posizione con il terminale di utente. Il sdr body e il sdr della navigazione sono anche indicati, rispettivamente, con i termini b-frame e n-frame.

In precedenza si è detto, che è necessario conoscere la posizione di partenza, più precisamente, ciò che occorre conoscere è l'assetto del sdr body rispetto al sdr della navigazione. Questi aspetti verranno affrontati nel Capitolo 4. Ritornando al caso dei sensori dei controller, visto nel paragrafo 3.1, si ha che i controller rilevano tramite i propri sensori i movimenti, ma poi per dare una giusta interpretazione a tali movimenti è necessario capire la posizione del controller rispetto al televisore,

quindi vengono usati degli strumenti aggiuntivi. In particolare, nel caso della Wii, si utilizza una barra sensore (Wii Sensor Bar), da porre sotto o sopra il televisore (parallelamente), che comunica con il controller mediante infrarossi. Nel sistema Play Station 3, invece è presente una telecamera (Play Station Eye) in grado di rilevare la luce sferica presente sul PlayStation Move.

3.3.1 Accelerometro

L'accelerometro è un dispositivo in grado di misurare le accelerazioni lineari in m/s^2. In genere, i moderni smartphone sono dotati di accelerometri in grado di valutare l'accelerazione lineare lungo tre fissate direzioni (x, y, z) del sistema di riferimento body, come mostrato in figura 3.4.

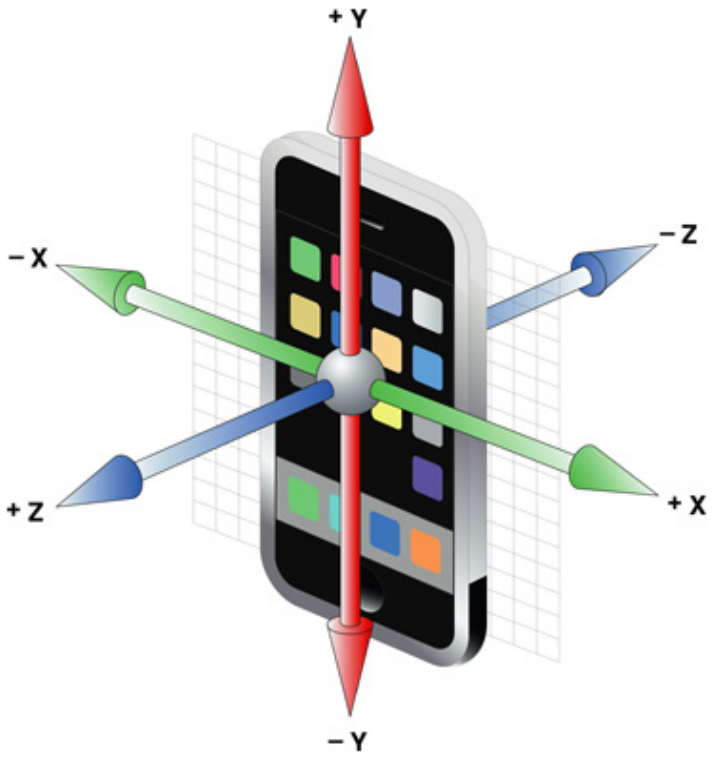


Figura 3.4: Assi di riferimento in iPhone 4 - sdr body per iPhone

É possibile fare una distinzione fra accelerometri meccanici, che sfruttano le accelerazioni che agiscono su una massa di prova, e acceleratori a vibrazione, che valutano l'accelerazione misurando le variazioni della frequenza di oscillazione dei materiali piezoelettrici. Il principio di funzionamento degli oscillatori meccanici è sostanzialmente il seguente (si veda [26]): si basa sulla rilevazione dell'inerzia di una massa quando questa viene sottoposta ad una accelerazione. La *massa* viene sospesa ad un elemento elastico, mentre un qualche tipo di sensore ne rileva lo spostamento

rispetto alla *struttura fissa* del dispositivo. In presenza di un'accelerazione, la massa (che è dotata di una propria inerzia) si sposta dalla propria posizione di riposo in modo proporzionale all'accelerazione rilevata. Il sensore trasforma questo spostamento in un segnale elettrico acquisibile dai sistemi di misura. Nel gergo meccanico, la massa e l'elemento elastico, sono chiamati rispettivamente massa sismica e trave di sospensione e il movimento della massa è chiamata deflessione (si veda [27]). I sensori di accelerazione, sono in genere dei sensori basati su capacità, ovvero *sensori capacitivi*. Il funzionamento generale è questo: la massa sismica e la struttura fissa, rappresentano le due armature della capacità; sotto l'effetto di un accelerazione, la massa di prova si muove provocando una variazione della distanza fra le armature e una conseguente variazione della capacità. Dalla variazione di capacità è calcolabile lo spostamento e quindi l'accelerazione. Un aspetto di notevole interesse degli accelerometri, è che possono essere miniaturizzati. Strumenti di questi tipo vengono chiamati MEMS (Micro Electro Mechanical System). Gli accelerometri, quindi sono fissati al loro sdr, ruotano e si muovono con esso, per cui con la sola informazione delle accelerazioni non sarà possibile capire l'orientamento del sdr body, rispetto al sdr della navigazione.

iPhone 4 monta un accelerometro prodotto dalla STMicroelectronics, il modello è L3G4200D. Uno dei classici utilizzi dell'accelerometro è in alcune applicazioni, in cui l'operazione di scossa del dispositivo, coincide ad eseguire di una certa operazione all'interno dell'applicativo: ad esempio per aggiornare la pagina del browser. Si utilizza l'accelerometro anche in una grande quantità di giochi.

In figura 3.5 vengono mostrati i risultati ottenuti sfruttando l'applicazione **xSensorPro**, che riporta i valori di accelerazione lungo i tre assi nel tempo; L'andamento delle accelerazioni è relativo ad un movimento "lieve" del terminale in un certo intervallo di tempo (circa 5 secondi). Dalla figura 3.5 si nota come i valori dell'accelerazione siano riportati in funzione dell'accelerazione gravitazionale terrestre media pari a $g = 9.80665 m/s^2$. Inoltre, è indicato il campo Total field, questa grandezza (che chiamo a_t) è la somma quadratica delle tre accelerazioni istantanee. Indico con (a_x, a_y, a_z) le accelerazioni lungo le componenti, risulta quindi:

$$a_t = \sqrt{a_x^2 + a_y^2 + a_z^2} \tag{3.2}$$

xSensor Pro è un applicativo realizzato dalla Moog CrossBow, un azienda statunitense, che opera nel mercato dei sensori e del tracking sia in ambito civile che militare. Durante questo capitolo verrà utilizzato questa applicazione, insieme ad altre, per testare il funzionamento dell'apparato sensoristico presente in iPhone 4.

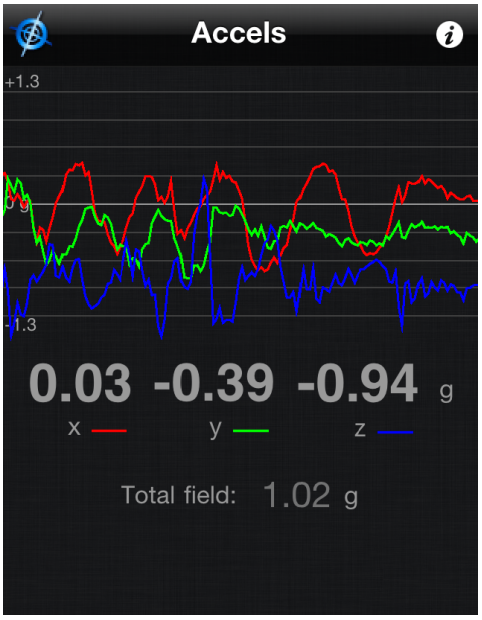


Figura 3.5: Applicazione "XSensor Pro": Accelerazioni misurate lungo i 3 assi durante uno scuotimento "lieve" dell'iPhone

Un importante considerazione che scaturisce dall'analisi dei valori di accelerazione lungo i tre assi, è che sfruttando l'accelerazione gravitazionale "subita" dal terminale, è possibile carpire com'è disposto il terminale in condizioni statiche rispetto al piano terrestre. In riferimento alla figura 3.4 (pag.50), considero che il dispositivo sia poggiato sul piano creato dagli assi (x, y) e che questo sia parallelo al piano tangente alla superficie della terra, e perpendicolare all'asse z, con l'asse z positivo uscente dalla superficie terrestre. In pratica, questa è la situazione del terminale poggiato su un piano "dritto", con il display rivolto verso l'alto. In questo caso, in linea teorica, si dovrebbero leggere dei valori $(a_x, a_y, a_z) = (0g, 0g, -1g)$. Rovesciando il terminale, con il display verso il basso si dovrebbero avere dei valori $(a_x, a_y, a_z) = (0g, 0g, 1g)$. tenendo in verticale l'iPhone si dovrebbe riscontrare la terna $(a_x, a_y, a_z) = (0g, -1g, 0g)$, e così via per le altre tre possibili situazioni. In sostanza, a partire dalla terna di informazioni delle accelerazioni è possibile trarre informazioni riguardanti la disposizione statica del terminale rispetto ad un piano tangente alla sfera terrestre. Se invece il dispositivo, fosse in caduta libera, ad esempio lungo l'asse z, si riscontrerebbe una terna $(a_x, a_y, a_z) = (0g, 0g, 0g)$. Quindi, le informazioni sull'accelerazione sono fornite sempre in relazione all'accelerazione di gravità g. Nella sezione 3.4 si valuterà un possibile utilizzo del sensore di accelerazione.

3.3.2 Giroscopio

Il sensore di accelerazione, anche in un caso puramente teorico di misurazioni non affette da errore, non è in grado da solo, di caratterizzare completamente il movimento di un terminale. Ecco perché, per finalità di positioning, basate su dead reckoning, è necessario un altro dispositivo in grado di percepire le velocità angolari; Tale dispositivo, è il giroscopio. Il giroscopio è un sensore in grado di misurare la velocità angolare rispetto ad un determinato asse. Un giroscopio a tre assi, quindi, valuta le velocità angolari, in rad/s o deg/s, rispetto alla terna di assi del proprio sdr inerziale. lo smartphone Apple iPhone 4 integra un giroscopio a 3 assi, fornito sempre dalla STMicroelectronics, il modello è L3G4200D (data sheet in STMicroelectronics [28]).

Il giroscopio è un dispositivo, inventato dal fisico Jean Bernard Léon Foucault nel 1852 nell'ambito degli studi sulla rotazione terrestre (si veda [29]). In riferimento alla figura 3.6, nel giroscopio meccanico "classico" si ha un rotore libero di ruotare intorno al proprio asse di rotazione (Spin axis). Questo asse è montato su una sospensione cardanica, così che tale asse possa orientarsi in qualsiasi direzione dello spazio. La velocità angolare è valutata tenendo conto del principio di conservazione del momento angolare secondo cui il momento angolare di un sistema è costante nel tempo se è nullo il momento delle forze esterne che agiscono su di esso. É possibile

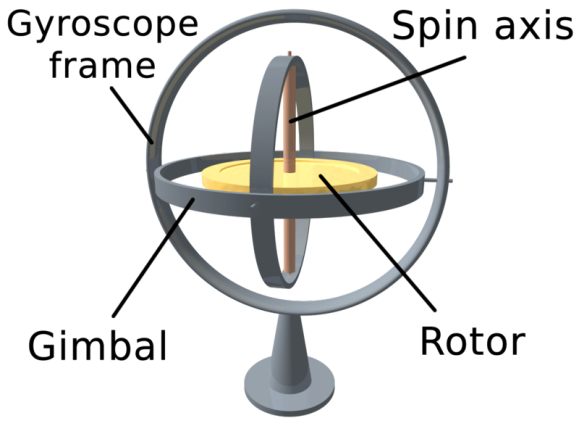


Figura 3.6: Schema di un giroscopio meccanico

identificare due macro-classi di giroscopi: elettromeccanici e ottici. I giroscopi elettromeccanici si basano sul momento angolare di una massa in rotazione, come detto in precedenza, oppure sull'effetto di Coriolis. I giroscopi ottici sono invece basati sul principio di Sagnac.

Dato che il giroscopio valuta le velocità di rotazione in deg/s lungo i tre assi del sdr body: (v_x, v_y, v_z), risulterà che se il terminale è fermo verrà misurata una terna $(v_x, v_y, v_z) = (0, 0, 0)$, indipendentemente dalla posizione del terminale.

3.3.3 Sensore di campo magnetico

Altro sensore di cui sono dotati gli smartphone di ultima generazione, è il sensore di campo magnetico; con questo sensore vengono effettuate delle misurazioni dell'intensità e della direzione del campo magnetico in μT. Le misurazioni effettuate sono sempre espresse rispetto agli assi cartesiani del sdr inerziale. La misura delle componenti di campo magnetico lungo le tre direzioni permette di definire in maniera univoca il vettore campo geomagnetico terrestre e di conseguenza è possibile calcolare l'angolo azimutale tra la direzione del sensore e il polo nord magnetico. É importante sottolineare che le misure sono relative al nord magnetico e non al nord geografico; in particolare il nord magnetico cambia posizione con il tempo ed è spostato verso occidente rispetto al nord geografico. L'angolo compreso tra nord magnetico e nord geografico è indicato come declinazione magnetica, per cui tale valore, oltre a cambiare nel corso nel tempo, varia anche in base al luogo in cui ci troviamo. In ogni caso, è possibile misurare tale declinazione con una buona precisione, ad esempio tramite una semplice applicazione del National Geographysical Data Center (`http://www.ngdc.noaa.gov/geomagmodels/struts/calcDeclination`) inserendo latitudine, longitudine e data. In figura 3.7 è mostrata l'applicazione "Bussola" basata sul sensore magnetico; 48° sono i gradi di differenza rispetto all'asse y crescente; è possibile scegliere se la bussola deve indicare il nord magnetico o quello geografico. La declinazione è valutata utilizzando le informazioni relative alla latitudine e alla longitudine mediante il GPS integrato.

Figura 3.7: Applicazione "Bussola": indicazione del nord magnetico

La differenza fra il magnetrometro e la classica bussola è che quest'ultima funzione solo in orizzontale, mentre il magnetometro fornisce indicazioni sul nord magnetico indipendentemente dalla posizione dell'iPhone.

La presenza di "forti" campi magnetici nell'intorno del terminale genera degli errori di misurazione del campo magnetico terrestre. Questa situazione è descritta in figura 3.8 in cui si è posto l'iPhone su un piano con il display rivolto verso l'alto. Il campo total field indica sempre, la somma quadratica istantanea dei campi magnetici lungo i tre assi. In 3.8.a è indicato il campo magnetico misurato nel tempo (un intervallo di circa 5 secondi), cercando di allontanare, nei limiti del possibile, tutte le fonti elettromagnetiche nell'intorno. In 3.8.b, si è posto un telefono cellulare (cellulare di test) in stand-by, adiacente all'iPhone; si nota una variazione del campo magnetico, lungo le componenti x e z, mentre la componente y rimane sostanzialmente invariata. Le componenti varieranno in base alla disposizione del cellulare di test rispetto all'iPhone. In 3.8.c, è mostrato il caso in cui si allontana e si avvicina il cellulare di test, lungo le diverse direzioni. Inoltre si è anche, spostato l'iPhone, rimettendolo poi nella stessa posizione. Nei test si è verificato che ci sono disposizioni fra cellulare di test e iPhone che generano un Total Field anche superiore ai $800\mu T$.

Il Compass Heading, è lo stesso valore misurato dall'applicazione "Bussola", ovvero la differenza in gradi tra l'asse y positivo del sdr body e il nord magnetico. Dalla figura 3.8.b, si nota che pur variando il campo magnetico, in maniera sostanziale, lungo le due componenti, il Compass Heading non varia: ciò è dovuto al fatto che il valore indicato dalla bussola, non dipende soltanto dai campi magnetici misurati, ma dipende anche dalle misurazioni effettuate dal giroscopio. Infatti, considerando il caso in cui il terminale sia posto in condizioni statiche, e venga misurato un certo valore di Compass Heading; se in seguito il campo magnetico varia bruscamente, e invece il giroscopio non ha registrato nessuna variazione angolare, allora quella variazione viene attribuita ad un campo magnetico esterno e il compass heading non varia. Così facendo, oltre a filtrare le interferenze, è possibile ottenere una maggiore velocità nella risposta della bussola. In figura 3.8.c invece lo spostamento dell'iPhone ha provocato uno sfasamento della misura. A conferma di quanto detto, è stato fatto il seguente test: ho piazzato il cellulare di test sul piano, e in seguito ho posizionato l'iPhone nella stessa posizione dei test precedenti (compass Heading 360°) vicino, al cellulare di test. Questo è stato fatto per provocare una variazione di campo magnetico tale da non essere considerata brusca, ma sufficiente ad ingannare il magnetometro. I risultati, sono stati di errori nell'ordine di 2°,3°.

In alcuni casi, si rende necessaria la calibrazione della bussola a causa di forti

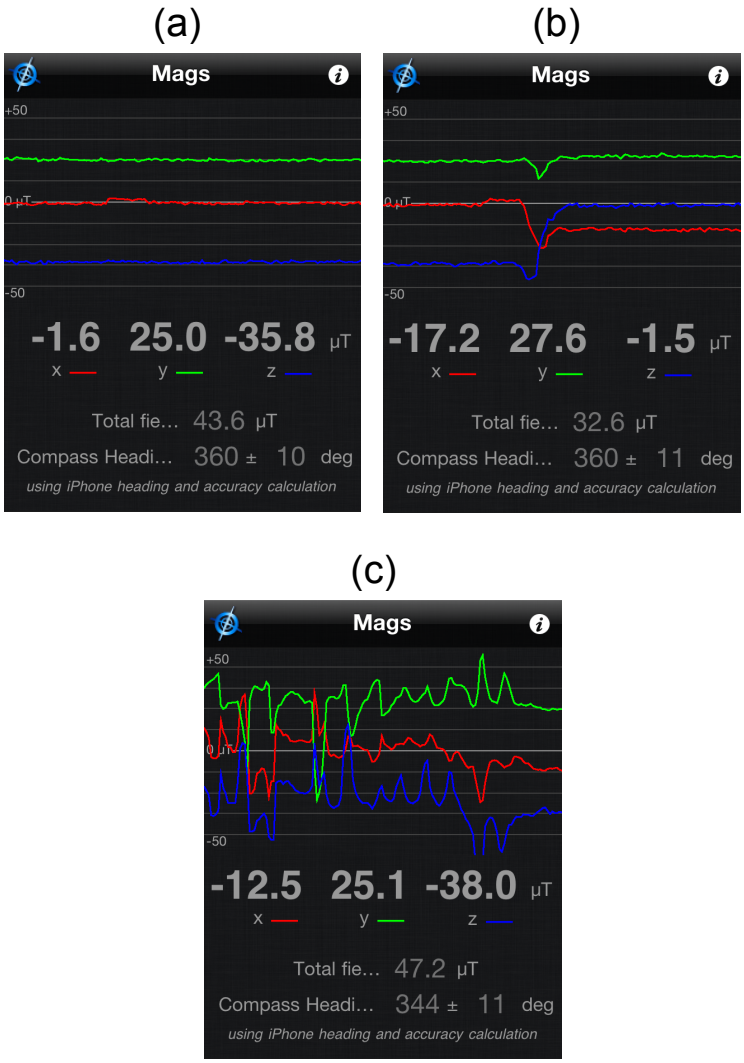


Figura 3.8: Applicazione "XSensor Pro": Misurazioni del campo magnetico al variare delle condizioni

interferenze ambientali o magnetiche (si veda [30]). In figura 3.8 è mostrata la schermata in cui si richiede la calibrazione. La calibrazione deve avvenire manualmente, con il movimento a 8 rappresentato in figura 3.8.

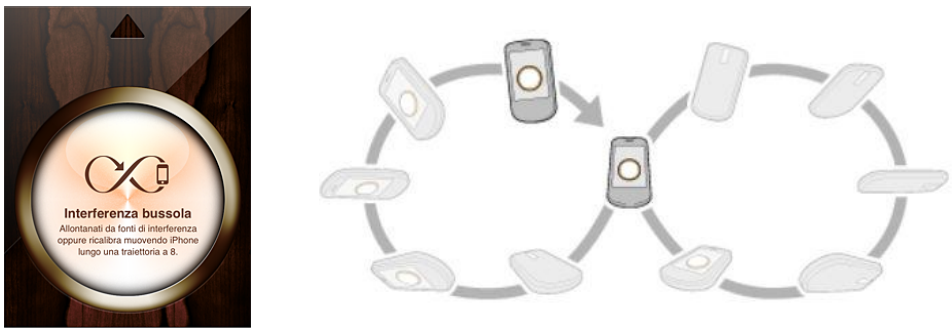


Figura 3.9: Schermata di calibrazione della bussola nei dispositivi Apple e movimento "a 8"

Durante i test, si è potuto verificare che in alcune situazioni, la bussola indicava direzioni completamente errate, con errori dell'ordine di decine di gradi, per lunghi frangenti di tempo, per poi tornare alla giusta direzione. Questa situazione trova riscontro anche nella documentazione Apple:

"Alcune aree sono maggiormente caratterizzate da interferenza magnetica rispetto ad altre. Ad esempio, il cruscotto dell'automobile potrebbe avere un elevato livello di interferenza magnetica. Se desideri utilizzare il dispositivo all'interno di un veicolo, non spostarlo lungo una traiettoria a 8 per eseguire la calibrazione e non tenere in considerazione il messaggio di avviso che viene visualizzato. Il dispositivo si ricalibrerà dopo qualche svolta. Ricorda che potrebbe essere necessario ricalibrare periodicamente la bussola mentre stai guidando, a seconda del livello di interferenza magnetica presente".

3.3.4 Orientamento

Dall'analisi dei tre sensori scaturiscono le seguenti considerazioni:

1. Attraverso il magnetometro è possibile individuare il nord magnetico (misure in μT).

2. Attraverso l'accelerometro è possibile capire la disposizione dell'iPhone rispetto ad un piano tangente alla superficie terrestre (misure rispetto a $g = 9.81\, m/s^2$).

3. Attraverso il giroscopio è possibile valutare gli spostamenti dell'iphone (misure in $gradi/sec$).

In definitiva è possibile capire l'orientamento dell'iPhone sfruttando le informazioni derivanti dai tre diversi sensori. Tutti i dati sul posizionamento possono essere riassunti dalla terna $(Roll, Pitch, Yaw)$. Questi tre valori sono espressi in gradi. In riferimento alla figura 3.10:

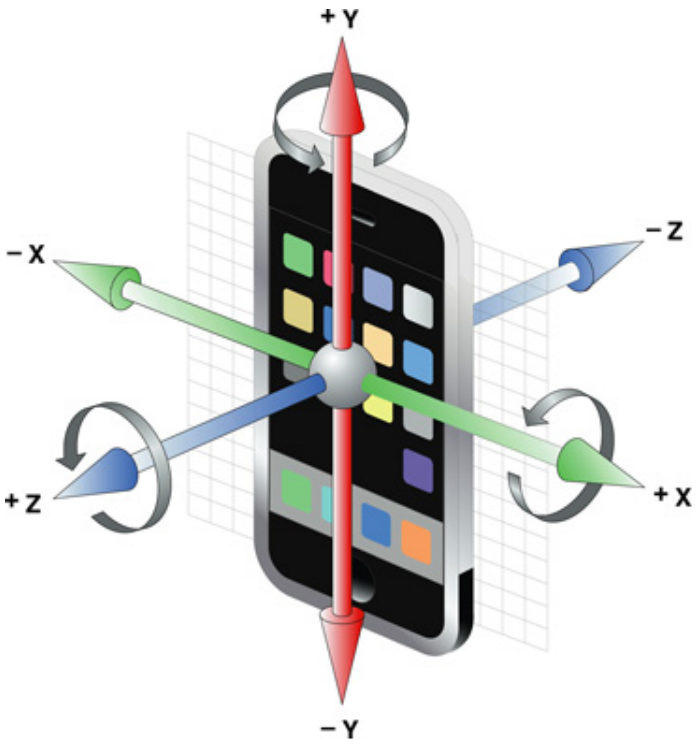


Figura 3.10: Orientamento basato sulla terna $(Roll, Pitch, Yaw)$

- **Roll**: rappresenta la rotazione del terminale, espressa in gradi, intorno all'asse y. Il range di variazione è $-180° \div +180°$. Il caso $roll = 0$, si ha quando il piano individuato dagli assi xy, è parallelo al piano tangente alla superficie terrestre. ruotando il terminale, rispetto all'asse y, in senso orario, si ottengono i valori crescenti di roll tra 0° e +180°. Ruotando in senso antiorario si ottengono i valori decrescenti di roll, tra 0° e −180°.

- **Pitch**: esprime la rotazione, in gradi, intorno all'asse x. Ruotando l'asse x, ruoteranno gli assi y e z. La rotazione è espressa nell'intervallo $-90° \div +90°$. $pitch = 0°$ si ha nei casi in cui il terminale (ovvero il piano x,y) è parallelo al piano tangente alla superficie terrestre. Per cui, questa situazione si riscontra nei casi in cui il terminale ha il display rivolto verso l'alto e display rivolto verso il basso. $pitch = 90°$ si ha quando l'asse y positivo è rivolto verso l'alto

perpendicolarmente al piano tangente terrestre. $pitch = -90°$ si ha nella situazione inversa.

- **Yaw**: indica la rotazione, in gradi, del terminale intorno all'asse z. Tale rotazione è espressa nell'intervallo $-180° \div +180°$. La rotazione dell'asse z determina una rotazione degli assi x e y.
  In alcuni casi, la terna $(Roll, Pitch, Yaw)$ è scelta in maniera tale che l'asse y positivo sia orientato verso il nord magnetico; ovvero per $yaw = 0$, l'asse y è orientato verso il nord magnetico. Per $yaw = +180°$ o $yaw = -180°$, l'asse y positivo è quindi orientato verso sud magnetico. Per $yaw = 90°$ e $yaw = -90°$, l'asse y positivo è orientato rispettivamente verso ovest ed est.
  In iPhone 4, e in particolare con l'applicazione xSensor Pro, il valore di yaw è settato a 0 ad ogni riavvio dell'applicazione. In sostanza, non si sfrutta nell'orientamento il contributo fornito dal magnetometro, per cui all'apertura dell'applicazione, indipendentemente dalla direzione, il valore di yaw è settato a 0.

La terna (Roll,Pitch,Yaw) deriva direttamente dalla terminologia usata in aviazione, la differenza sostanziale è sulla terna originaria (x, y, z). Gli assi x e y risultano invertiti, come mostrato in figura 3.11 dove si evidenziano anche i termini beccheggio, imbardata e rollio.

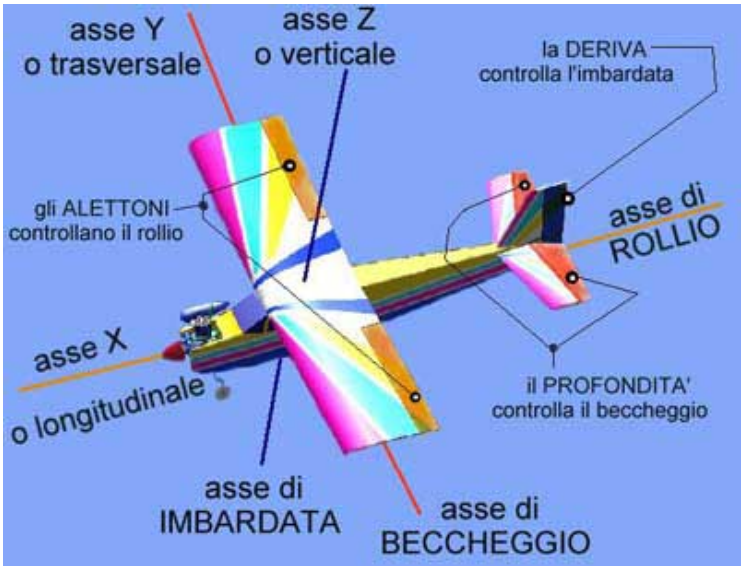


Figura 3.11: Orientamento usato in aviazione

In figura 3.12 sono mostrati i valori (Roll,Pitch,Yaw), insieme ai valori misurati dal giroscopio, mediante l'applicazione "xSensor Pro"; nella stessa figura sono poi

mostrate tutte le misurazioni dei sensori (accelerazione, giroscopio, magnetometro) e in basso i risultati dell'orientamento e le coordinate GPS. L'applicativo in questione è Pocket IMU. In questo caso le velocità angolari sono espresse in rad/s e i valori di orientamento (campo Attitude) sono espressi in rad.

Figura 3.12: xSensor Pro: misurazioni giroscopio e stima orientamento. Pocket IMU: misurazioni sensoriali e orientamento

Il calcolo dell'orientamento, può essere fatto a partire dalle misure dei sensori inerziali lungo i tre assi. In particolare è necessario calcolare i quaternioni e in seguito applicare un apposito algoritmo per ricondursi alla terna roll, pitch, yaw. Un possibile algoritmo è l'algoritmo AHRS Estimation descritto in Madgwick [31].
É importante evidenziare che la Apple mette a disposizione del programmatore un insieme di *API* e *framework* (si veda Novelli [32]), in grado di gestire direttamente i sensori presenti in iPhone. Ad esempio per la bussola, viene fornito uno specifico oggetto: "locationManager", per l'accelerometro sono disponibili la classe "UIAccelleration" e il protocollo "UIAccelerometerDelegate". Anche i risultati relativi all'orientamento, possono essere direttamente estratti utilizzando funzionalità esistenti e messe a disposizione dalla Apple, in particolare si sfrutta la API "CoreMotion", su cui si basa appunto l'applicazione xSensor Pro, come si vede dalla figura 3.12 ("Orientation estimate from CoreMotion API").
Il magnetometro, come detto, è soggetto a problemi di interferenze che ne limitano l'usabilità per scopi di positioning soprattutto in ambienti particolarmente interferenti in cui viene accentuata la non affidabilità di questo strumento. Ecco perché,

con la terna $(roll, pitch, yaw)$ che non fa uso del magnetometro, non è possibile valutare completamente l'orientamento del dispositivo, proprio perché manca l'informazione relativa alla disposizione del sdr body rispetto al sdr della navigazione (o analogamente rispetto al sistema nord-sud-ovest-est). Questo aspetto è stato preso in considerazione, inserendo all'interno del codice posizionale anche un altro parametro (α) che descrive la rotazione rispetto al sdr della navigazione.

3.3.5 Errori nelle misure inerziali

Gli errori sulle misurazioni inerziali possono essere suddivisi in due categorie (si veda Groves [24]):

- Errori sistematici: sono delle deviazioni della misura, rispetto al valore reale, costanti in entità e che mantengono lo stesso segno. Sono errori sistematici il *bias* e il *fattore di scala*, e nel caso di sensori inerziali, si possono considerare anche gli effetti legati alla *non perfetta ortogonalità* delle terne di accelerometri e giroscopi. Gli errori sistematici possono essere eliminati o ridotti mediante delle operazioni di calibrazione.

- Errori casuali: sono dovuti a fattori non controllabili e sono di segno alterno; determinano la variabilità delle misurazioni intorno a un certo valor medio (a parità di condizioni sperimentali). L'origine degli errori casuali può essere interna o esterna; nel primo caso si parla di rumore e nel secondo di disturbo. L'effetto derivante dagli errori casuali può essere ridotto ripetendo più volte la misura, oppure utilizzando degli appositi filtraggi.

Le prestazioni dei sensori inerziali variano notevolmente a seconda del tipo. Noi siamo interessati ai sensori MEMS e per questi tipi di sensori un importante fattore di cui tener conto è quello legato alla componente di *errore run-to-run*, che è un errore di polarizzazione che varia ad ogni nuovo utilizzo del sensore, ma che rimane costante all'interno di ogni esecuzione, ovvero è un errore sistematico sostanzialmente costante per il singolo utilizzo, ma variabile tra diverse misurazioni. Altro tipo di errore sistematico è quello legato alla variazione di temperatura che, analogamente al fattore di scala e al bias, può essere parzialmente compensato in fase di calibrazione. Questo tipo di incertezza è indicata come *in-run bias*. Vediamo più nel dettaglio i vari tipi di errore descritti.

Il *bias* o polarizzazione, è un errore costante, presente in tutti gli accelerometri e giroscopi indipendentemente dalla forza o dalla velocità angolare applicata. Nella maggior parte dei casi, il bias rappresenta la principale causa d'errore nelle misure inerziali. Questa componente di errore può essere espressa come un vetto-

re rispetto alla terna di riferimento (x, y, z), in tal caso si avrà per l'accelerometro (pedice a) e per il giroscopio (pedice b) rispettivamente $b_a = (b_{a,x}, b_{a,y}, b_{a,z})$ e $b_g = (b_{g,x}, b_{g,y}, b_{g,z})$. Tramite l'espressione vettoriale è possibile scindere le componenti di errori su ogni asse. In alternativa, è possibile effettuare una caratterizzazione del bias mediante la somma della componente statica e dinamica: $b_a = b_{as} + b_{as}$ e $b_g = b_{gs} + b_{gs}$. La componente statica include l'incertezza run-to-run e la componente di bias residuo dopo la calibrazione (dato che il bias varia per ogni utilizzo). La componente dinamica è invece una componente che varia anche in intervalli di tempo ridotti, intorno al minuto, ed è legata alle variazioni della temperatura del sensore.

Il *fattore di scala*, rappresenta il rapporto che esiste fra l'output e l'input del sensore. Questo rapporto dovrebbe essere pari a 1 per ogni input. Questo concetto è espresso nella figura 3.13 (estratta da [24]), in cui è visualizzata la pendenza input/output. Il caso ideale è quello in cui input=output (curva tratteggiata), invece l'output dello strumento produce una curva non a 45°, per cui al crescere della forza applicata (accelerometro) o della velocità angolare (giroscopio), crescerà l'errore causato dal fattore di scala.

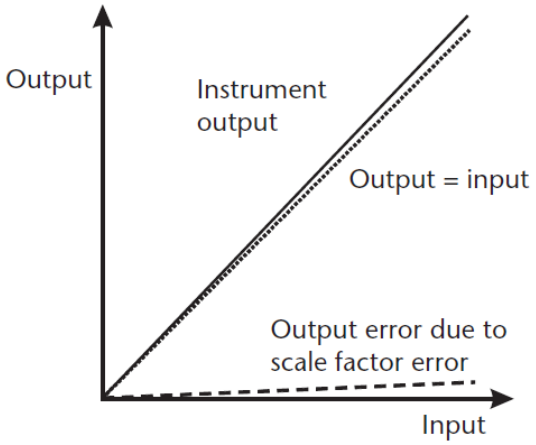


Figura 3.13: Errori legati al fattore di scala

Altro errore sistematico è quello legato alla non ortogonalità degli assi rispetto alla terna del sdr, ovvero si ha un disallineamento fisico degli assi del giroscopio e dell'accelerometro, rispetto alla terna di riferimento (sdr body) a cui sono riferite le misure inerziali. Questo concetto è rappresentato in figura 3.14 (estratta da [24]).

Gli errori di fattore di scala e non ortogonalità vengono di solito indicati in parti per milione (soprattutto nei sensori ad alte prestazioni) o in percentuale. Nei sensori

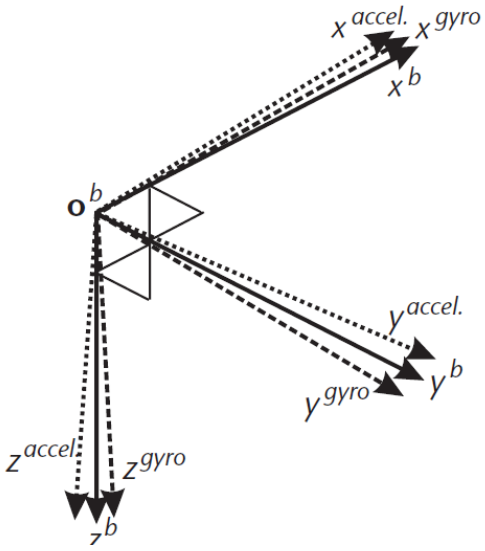


Figura 3.14: Errori legati alla non ortogonalità degli assi reali dei sensori

MEMS gli errori legati al fattore di scala e alla non ortogonalità sono dell'ordine del 1%.

Anche per gli errori casuali può essere effettuata una caratterizzazione vettoriale: Ogni campione misurato dall'accelerometro e dal giroscopio, è affetto da una componente di rumore o disturbo espressa dalle terne $w_a = (w_{a,x}, w_{a,y}, w_{a,z})$ e $w_g = (w_{g,x}, w_{g,y}, w_{g,z})$.

Adesso analizziamo, l'errore di polarizzazione presente, nell'accelerometro e nel giroscopio dell'iPhone 4 in uso, così da caratterizzare la calibrazione iniziale e capire quanto i sensori dell'iPhone 4 siano affetti da bias. Il concetto di calibrazione dei sensori inerziali è già presente in alcune applicazioni Apple. Una delle applicazioni più diffuse ed efficienti è "Carpentiere iHandy" della iHandySoft Inc. in cui sono presenti strumenti come la livella classica, la livella di superficie e il filo a piombo. Per ogni strumento vengono fornite, oltre all'interfaccia visiva, le informazioni sull'inclinazione. É possibile una calibrazione diretta e indipendente degli strumenti. I tre strumenti dell'applicazione sono mostrati in figura 3.15.

Nel test di calibrazione si è disposto l'iPhone 4 in posizione statica su un una struttura piana di test così' che il piano (x, y) del sdr body del terminale fosse parallelo alla struttura piana di test, in sostanza l'iPhone è stato appoggiato sulla struttura con il display rivolto verso l'alto. La struttura di test, è una struttura formata da un piano e quattro piedi di altezza variabile che è stata "messa in bolla" mediante l'uso di una coppia di livelle a bolla. Il test è stato eseguito in un ambiente con temperatura di $19.8°C$. Nelle condizioni descritte, in linea teorica, si dovrebbero misurare delle accelerazioni $(a_x, a_y, a_z) = (0g, 0g, -1g)$ e delle velocità

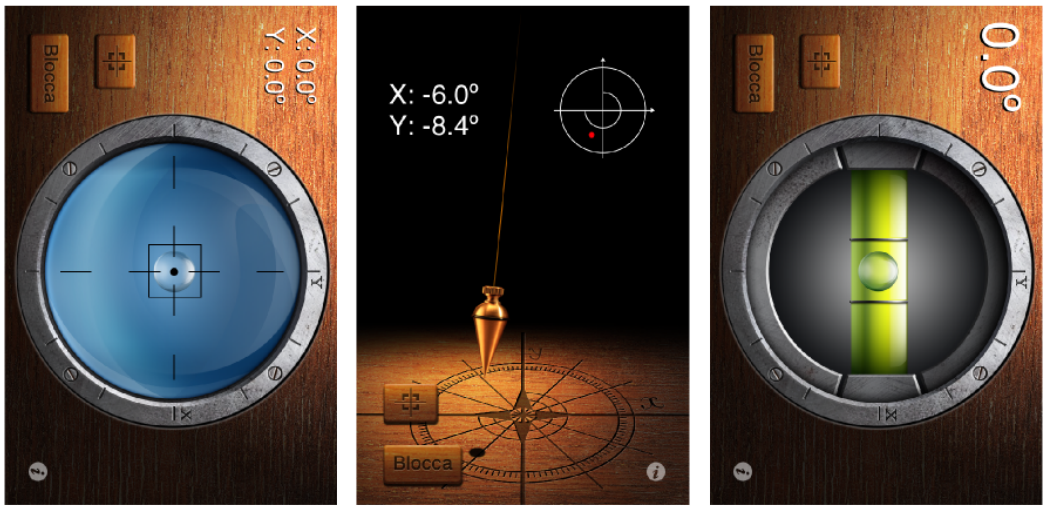


Figura 3.15: Livella di superficie, filo a piombo e livella classica nell'applicazione Carpentiere iHandy

angolari $(v_x, v_y, v_z) = (0, 0, 0)$; mentre i valori stimati di orientamento dovrebbero essere $(roll, pitch, yaw) = (0, 0, 0)$. In realtà, i valori oscillano (rumore) intorno a un valore differente determinato dal bias. Si sono effettuate 5 diverse serie di misurazioni. Ogni serie di misurazioni (a 32 Hz) è effettuata per un intervallo di tempo di 20 secondi (20x32=640 valori). Si calcola quindi il valor medio (analisi polarizzazione) e la varianza (analisi dispersione). I risultati ottenuti nelle cinque prove, approssimati alla quarta cifra decimale, sono mostrati nelle tabelle 3.1, 3.2 e 3.3 in cui sono rappresentate le misurazioni di accelerazione, velocità angolare e i valori di orientamento (ottenuti sfruttando API CoreMotion).

	$\eta_{a,x}(.g)$	$\eta_{a,y}(.g)$	$\eta_{a,z}(.g)$	$\sigma^2_{a,x}(.g)$	$\sigma^2_{a,y}(.g)$	$\sigma^2_{a,z}(.g)$
1	-0.0222	-0.0109	-1.0220	$0.1694.10^{-4}$	$0.0882.10^{-4}$	$0.1603.10^{-4}$
2	-0.0222	-0.0110	-1.0237	$0.1738.10^{-4}$	$0.0914.10^{-4}$	$0.2331.10^{-4}$
3	-0.0226	-0.0108	-1.0250	$0.1932.10^{-4}$	$0.0748.10^{-4}$	$0.2504.10^{-4}$
4	-0.0229	-0.0108	-1.0276	$0.2058.10^{-4}$	$0.0748.10^{-4}$	$0.1822.10^{-4}$
5	-0.0227	-0.0109	-1.0283	$0.1968.10^{-4}$	$0.0818.10^{-4}$	$0.1405.10^{-4}$

Tabella 3.1: Valor medio e varianza dei valori di accelerazione misurati nel test di calibrazione

	$\eta_{g,x}(rad/s)$	$\eta_{g,y}(rad/s)$	$\eta_{g,z}(rad/s)$	$\sigma^2_{g,x}(rad/s)$	$\sigma^2_{g,y}(rad/s)$	$\sigma^2_{g,z}(rad/s)$
1	0.0274	-0.0078	0.0110	$0.1701.10^{-4}$	$0.0647.10^{-4}$	$0.1399.10^{-4}$
2	0.0152	-0.0083	0.0101	$0.2170.10^{-4}$	$0.0967.10^{-4}$	$0.2072.10^{-4}$
3	0.0150	-0.0084	0.0095	$0.2075.10^{-4}$	$0.1071.10^{-4}$	$0.2232.10^{-4}$
4	0.0146	-0.0083	0.0091	$0.2168.10^{-4}$	$0.1065.10^{-4}$	$0.2311.10^{-4}$
5	0.0270	-0.0078	0.0095	$0.1823.10^{-4}$	$0.0536.10^{-4}$	$0.1438.10^{-4}$

Tabella 3.2: Valor medio e varianza dei valori delle velocità angolari misurate nel test di calibrazione

	$\eta_{roll}(rad)$	$\eta_{pitch}(rad)$	$\eta_{yaw}(rad)$	$\sigma^2_{roll}(rad)$	$\sigma^2_{pitch}(rad)$	$\sigma^2_{yaw}(rad)$
1	-0.0229	0.0120	0.0201	$0.1199.10^{-6}$	$0.0642.10^{-6}$	$0.6154.10^{-6}$
2	-0.0230	0.0119	0.0200	$0.0014.10^{-5}$	$0.0059.10^{-5}$	$0.3421.10^{-5}$
3	-0.0230	0.0116	0.0190	$0.0003.10^{-4}$	$0.0024.10^{-4}$	$0.1425.10^{-4}$
4	-0.0231	0.0119	0.0201	$0.0090.10^{-5}$	$0.0121.10^{-5}$	$0.1536.10^{-5}$
5	-0.0231	0.0116	0.0267	$0.0107.10^{-5}$	$0.0298.10^{-5}$	$0.4368.10^{-5}$

Tabella 3.3: Valor medio e varianza dei valori di orientamento stimati nel test di calibrazione (CoreMotion API)

L'errore di polarizzazione medio nelle misure di accelerazione è pari a $(\eta_{a,x}, \eta_{a,y}, \eta_{a,z}) = (-0.0225g, -0.0109g, -0.0253g)$. Vediamo come si ripercuote questo errore sulla stima della velocità e della posizione nei sistemi inerziali. Considero che le misure di accelerazione $\ddot{x}_m$ siano affette da un errore di polarizzazione b_a; considero che non ci siano altre componenti di errore sulla misura. Risulterà $\ddot{x}_m = \ddot{x} + b_a$, dove $\ddot{x}$ è l'accelerazione vera. La velocità misurata vale:

$$\dot{x}_m = \int \ddot{x}_m \, dt = \int \ddot{x} \, dt + \int b_a \, dt = \dot{x} + b_a t \tag{3.3}$$

Mentre la posizione risultante è:

$$x_m = \int \dot{x}_m \, dt = \int \dot{x} \, dt + \int b_a \, t \, dt = x + \frac{1}{2} \, b_a \, t^2 \tag{3.4}$$

Nel caso considerato, si avrà che dopo $t = 20$ secondi, gli errori di polarizzazione medi, determinano un errore in metri, su ogni asse, pari a:

$$\begin{cases} e_x = \frac{1}{2} \, b_{a,x} \, t^2 = -44.2 \, m \\ e_y = \frac{1}{2} \, b_{a,y} \, t^2 = -21.4 \, m \\ e_z = \frac{1}{2} \, b_{a,z} \, t^2 = -49.6 \, m \end{cases} \tag{3.5}$$

Per quanto riguarda le dispersioni medie risulta invece:
$(\sigma^2_{a,x}, \sigma^2_{a,y}, \sigma^2_{a,z}) = (0.1878 \, 10^{-4} \, g, 0.0822 \, 10^{-4} \, g, 0.1933 \, 10^{-4} \, g)$.
In realtà, in questo lavoro di tesi, la posizione è valutata tenendo conto del calcolo del numero di passi (sfruttando i dati dell'accelerometro) e non dei valori istantanei di accelerazione. Per questo motivo, gli errori sull'accelerazione derivanti dal bias, influiscono solo in minima parte sul calcolo della distanza. Questo concetto verrà approfondito nei paragrafi 3.4 e 3.6. In ogni caso, è possibile asserire che non è possibile sfruttare i valori di accelerazione in "modo diretto", in sensori di questo tipo, dato che questi produrrebbero degli errori inaccettabili sulla posizione.

Per quanto riguarda gli errori di polarizzazione medi relativi all'orientamento, espressi in gradi, risulta: $(\eta_{roll}, \eta_{pitch}, \eta_{yaw}) = (-1.3188, 0.6764, 1.2131)$ con varianze praticamente trascurabili. Questa "stabilità" dei valori deriva dal fatto che la terna (roll,pitch,yaw) è valutata utilizzando appositi filtraggi attraverso la CoreMotion API della Apple.

Osservando le varianze relative alle misure di accelerazione e velocità angolare, si vede come le dispersioni assumono dei valori piuttosto ridotti, per questo motivo per una calibrazione dei sensori efficiente, è sufficiente considerare i valori di un unica osservazione. Ed è proprio questo che viene fatto con l'applicativo Carpentiere iHandy, in cui la calibrazione è effettuata istantaneamente, con la semplice pressione di un tasto del touch screen, disponendo il terminale nelle posizioni richieste nella fase di calibrazione

3.4 Calcolo della distanza

Nel paragrafo 3.2, si è detto che per stimare la nuova posizione, è necessario conoscere la distanza percorsa e l'orientamento a partire da una assetto noto. In questo paragrafo ci occuperemo del calcolo della distanza e nel prossimo dell'orientamento.

L'idea che sta alla base del calcolo della distanza è quella di sfruttare le misurazioni effettuate tramite accelerometro, così da capire il numero di passi eseguiti e sulla base di informazioni integrative sul soggetto, valutare la distanza. Per cui il terminale, viene utilizzato come un vero e proprio contapassi (anche chiamato podometro). In particolare, il conteggio dei passi è effettuato sulla base dell'accelerazione totale, espressa nell'equazione 3.2. I podometri commerciali, vengono solitamente fissati alla cintura o al polso, e sono in grado di valutare la distanza percorsa semplicemente moltiplicando il numero di passi, per la lunghezza del passo stesso. Il principio che si vuole applicare è lo stesso. Esistono alcune applicazioni Apple che effettuano il conteggio dei passi, una di queste è Pedometer, che valuta anche il tempo e quindi la velocità. In questo contapassi per dispositivi Apple, come nei comuni podometri, è necessario inserire l'altezza e il peso (per il conteggio delle calorie), inoltre è presente una manopola che controlla la sensibilità dei passi; il livello della sensibilità deve essere tarato direttamente dall'utente. Il grosso problema dei podometri è il calcolo dei passi durante percorsi in salita o in discesa, ad esempio durante una passeggiata in montagna; in questi casi la lunghezza del passo varia notevolmente rispetto al caso in pianura, e ciò determina un errore sul calcolo della distanza che cresce linearmente con il numero di passi. Questo problema non è però riscontrabile in ambito indoor, dove i percorsi sono solitamente in pianura. L'unica variazione di distanza si potrebbe eventualmente avere sulle scale, vedremo in seguito questo caso.

Figura 3.16: Pedometer: Applicazione per il conteggio dei passi

I test sono basati sulle misure di accelerazione estrapolate dell'applicazione xSensor Pro. Infatti con questa applicazione è possibile registrare i dati percepiti da tutti i sensori in un file formato .txt fornendo start e stop alla registrazione; in seguito si può inviare questo file tramite mail. i dati possono essere raccolti con 4 differenti rate temporali: 1 Hz, 4 Hz, 16 Hz, 32 Hz, si è scelta quest'ultima frequenza temporale di osservazione;

Il primo test è stato effettuato nelle seguenti condizioni:

- terminale in mano (destra) con display rivolto verso l'alto, in riferimento al sdr body (figura 3.4 a pag. 50), la camminata viene effettuata (circa) lungo la direzione dell'asse y crescente;
- primo passo dopo 2 secondi dallo start;
- stop dopo 14 passi;
- il terminale è tenuto cercando di "simulare" l'osservazione del display, quindi senza farlo oscillare eccessivamente.
- cammino rettilineo.

I risultati, graficati in Matlab, sono visibili nelle figure 3.17, 3.18, 3.19 e 3.20, in cui si mostrano i valori delle accelerazioni lungo i tre assi e l'accelerazione totale.

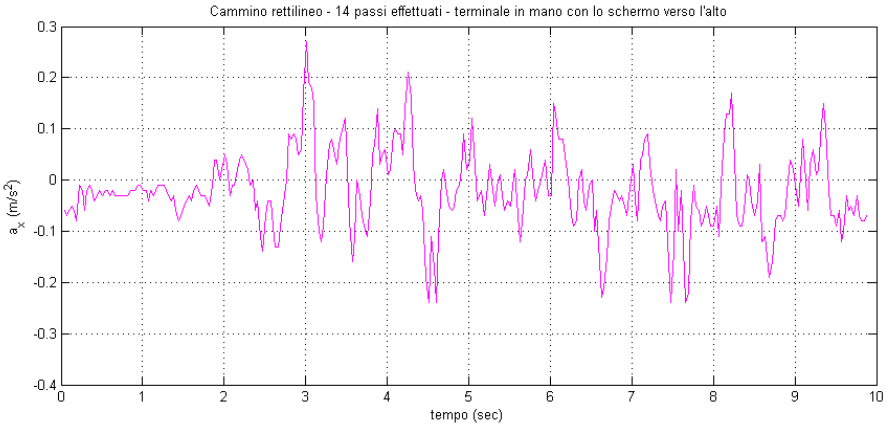


Figura 3.17: Cammino rettilineo: valori di accelerazioni misurati lungo l'asse x

Dalla figura 3.17, si notano delle variazioni di accelerazione piccole , proprio perché durante il cammino non sono state effettuate "svolte", ma si è mantenuta una traiettoria circa dritta. Dai grafici nelle figure 3.18 e 3.19, si notano delle accelerazioni coerenti con il movimento della camminata, in "avanti" e "su e giù". Dall'ultimo grafico, che rappresenta la somma quadratica delle accelerazioni, si notano dei picchi più netti.

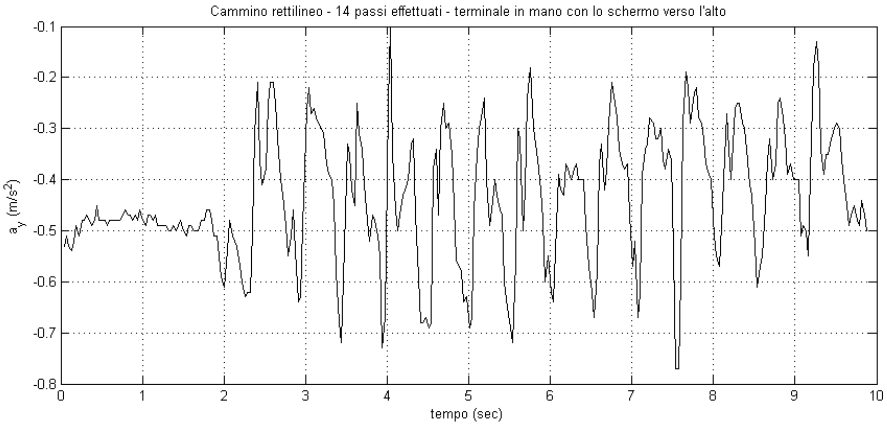


Figura 3.18: Cammino rettilineo: valori di accelerazioni misurati lungo l'asse y

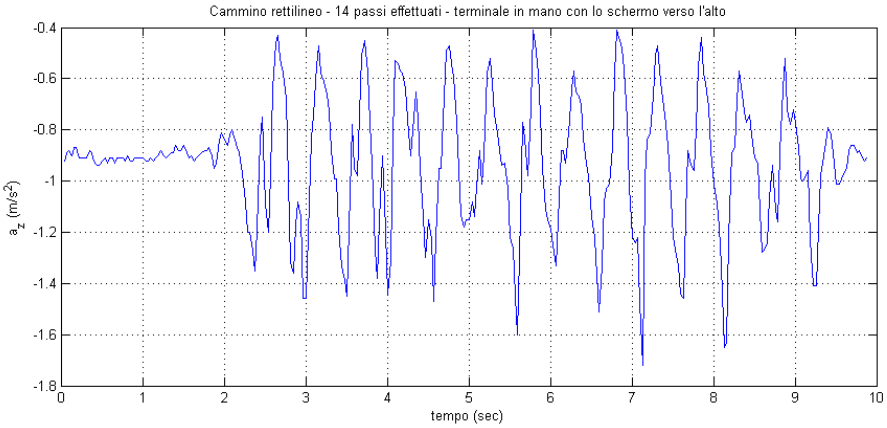


Figura 3.19: Cammino rettilineo: valori di accelerazioni misurati lungo l'asse z

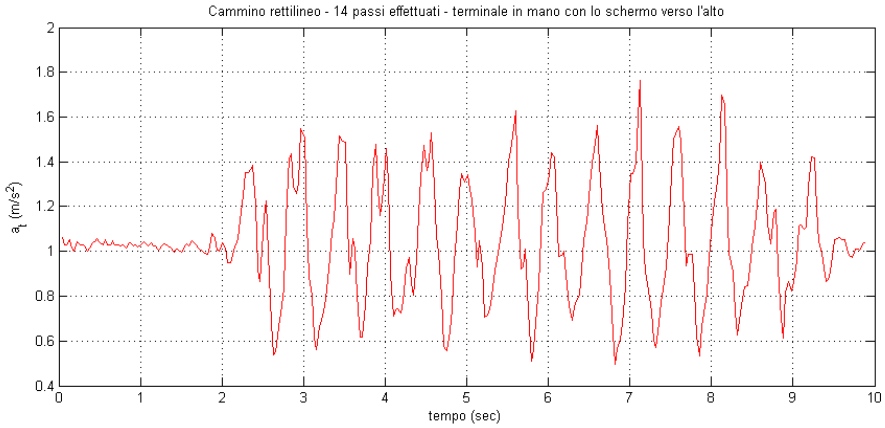


Figura 3.20: Cammino rettilineo: accelerazione totale

Per calcolare il numero di passi sarebbe sufficiente fissare una soglia superiore: il numero di passi effettuato è pari al numero di intervalli in cui la soglia è stata oltrepassata. In realtà è stata considerata una doppia soglia, inferiore e superiore. Si considera il passo effettuato se viene oltrepassata la soglia superiore, quando in precedenza era stata oltrepassata la soglia inferiore, ad eccezione del primo passo, in cui si considera che la soglia inferiore sia stata già oltrepassata. Così facendo si evita il problema del superamento multiplo della soglia. Questo concetto è espresso nel grafico in figura 3.21: i colori rosso e blu indicano rispettivamente gli intervalli sotto soglia e sopra soglia presi in esame per il conteggio del singolo passo.

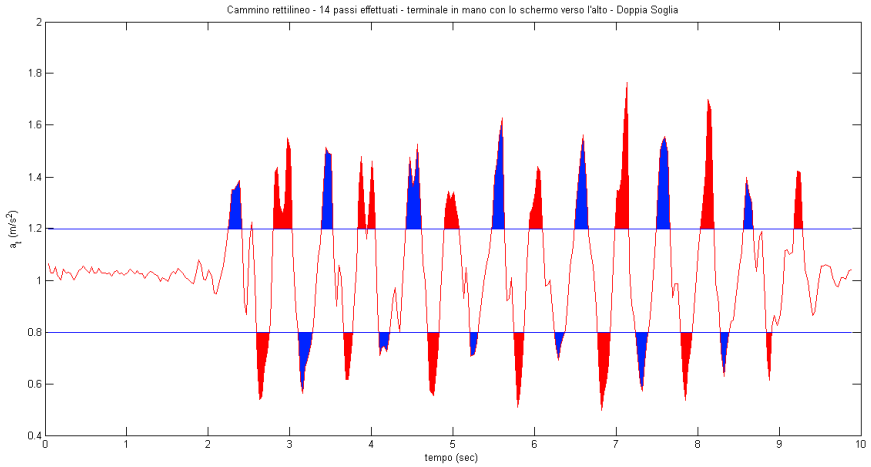


Figura 3.21: Cammino rettilineo: Doppia Soglia per valutare il numero di passi effettuati

Il secondo test, è analogo al precedente, solo che questa volta il percorso prevede una ”curva” a destra e si è cercato di fare oscillare meno il terminale. Le condizioni del test, sono le seguenti:

- terminale in mano (destra) con display rivolto verso l'alto;
- primo passo circa 1 secondo dallo start;
- stop dopo 15 passi;
- percorso con una curva a destra.
- il terminale tenuto cercando di ”simulare” l'osservazione del display: quindi senza farlo oscillare troppo.

I grafici, che mostrano l'andamento delle accelerazioni, sono mostrati nelle figure 3.22 e 3.23. L'algoritmo descritto prima, con soglie fissate a 0.9 g e 1.2 g, risulta efficace e conteggia effettivamente 15 passi.

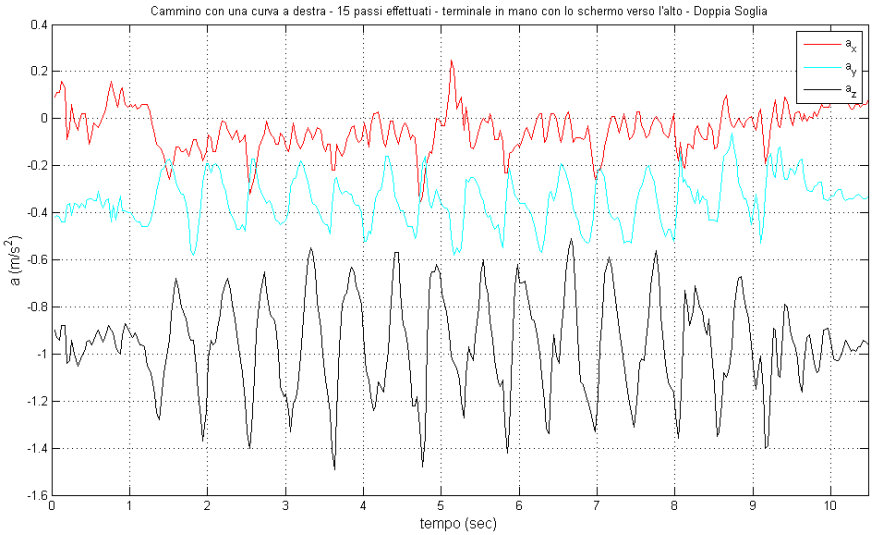


Figura 3.22: Cammino con curva a destra e terminale impugnato con la mano destra: valori di accelerazioni misurati lungo i tre assi

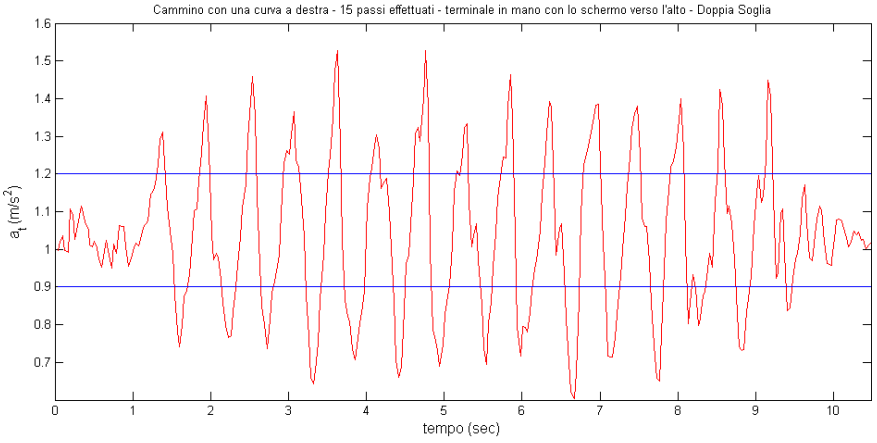


Figura 3.23: Cammino con curva a destra e terminale impugnato con la mano destra: accelerazione totale

Dalla 3.22 relativa ad a_x, si vede come lungo l'asse x i valori oscillano intorno allo 0, proprio perché il terminale è tenuto, circa orizzontalmente. Intorno al quinto secondo, sempre sull'andamento relativo ad a_x si vedono dei picchi, dovuti alla curva a destra. Infatti le accelerazioni laterali, vengono percepite maggiormente su questo asse.
Dall'andamento di a_y si nota che questo è centrato intorno a -0.3, questo sempre a causa del posizionamento del terminale, leggermente inclinato sull'asse y, per poter vedere lo schermo; infatti con il terminale in un piano orizzontale, si ha $a_y = 0$, mentre se si ruota il terminale di 90° lungo l'asse y (ovvero in verticale), risulta $a_y = -1$. Infine dalla curva nera, relativa ad a_z, si registrano le maggiori oscillazioni, dovute all'oscillazione verticale della camminata.

La scelta di utilizzare l'accelerazione totale, svincola il problema dal posizionamento del terminale, infatti anche tenendo l'iPhone in un altra posizione, i risultati sarebbero gli stessi. Questo concetto trova un utile applicazione, nel caso in cui l'iPhone sia impugnato orizzontalmente; questo è il caso tipico di visualizzazione orizzontale della schermata. In questa situazione, si ha che i valori misurati lungo gli assi x e y si invertono, ma il risultato sull'accelerazione totale è il medesimo. Altra possibile situazione è un utente che impugna lo smartphone con la mano sinistra e lo tiene inclinato lungo l'asse x, oltre che lungo l'asse y. Quest'ultima situazione è mostrata nelle figure 3.24 e 3.25, in cui è stato ripetuto il precedente test, variando solo l'impugnatura del terminale. Ovvero le figure 3.24 e 3.25 mostrano l'andamento delle accelerazioni nel caso di percorso con curva a destra e terminale impugnato con la mano sinistra. Si nota, rispetto al caso precedente, come le componenti x e y, presentino un maggior contribuito (in valore assoluto), invece è diminuito il contributo di accelerazione lungo l'asse z. Il contributo totale è invece analogo.

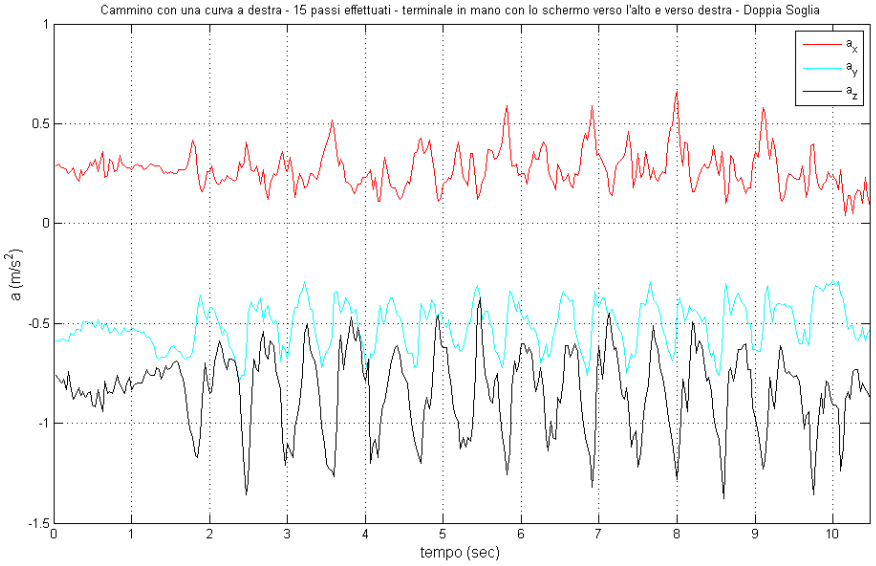


Figura 3.24: Cammino con curva a destra e terminale impugnato con la mano sinistra: valori di accelerazioni misurati lungo i tre assi

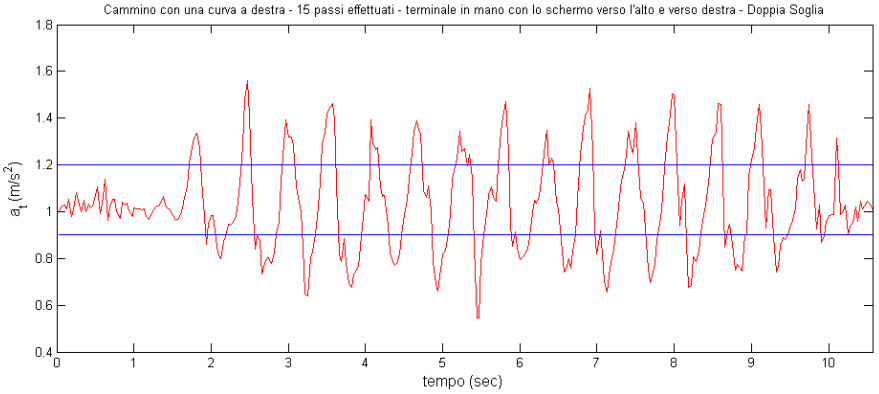


Figura 3.25: Cammino con curva a destra e terminale impugnato con la mano sinistra: accelerazione totale

Nel test successivo, sono stati valutati gli andamenti dell'accelerazione nei casi di 10 scalini percorsi in salita. Il test è stato eseguito nelle seguenti condizioni:

- terminale in mano con il display rivolto verso l'alto;
- primo passo dopo circa 1 secondo dallo start;
- scalinata in salita: un passo per ogni gradino (si potrebbero fare due o più gradini con un unico passo);
- stop dopo 10 gradini;
- il terminale tenuto cercando di "simulare" l'osservazione del display: quindi senza farlo oscillare eccessivamente.

I risultati del caso di scalinata in salita, sono mostrati nelle figure 3.26 e 3.27.

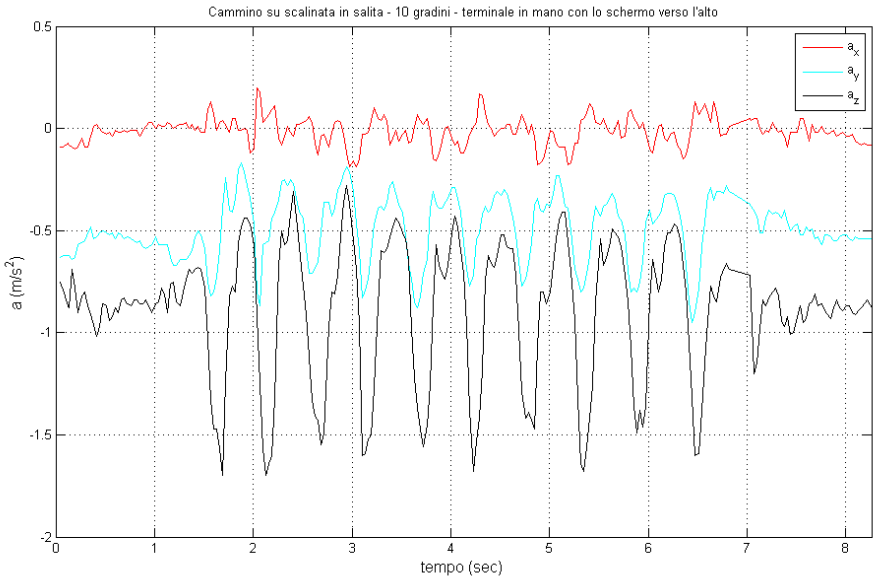


Figura 3.26: Scala in salita: valori di accelerazioni misurati lungo i tre assi

I risultati ottenuti nel caso di scala percorsa in discesa, sono ancora più bruschi, ovvero picchi positivi e negativi ancor più marcati, e "facili" da conteggiare.

Un elemento di fondamentale importanza lo giocano le soglie. Si potrebbe operare una scelta analoga a quella dell'applicativo "Pedometer", ovvero variare la sensibilità, così da spostare le soglie in base alla calibrazione effettuata dall'utente. Un'altra possibilità è quella di avere delle soglie adattative elaborando i dati relativi all'accelerazione. In linea generale però è possibile asserire, che in condizioni operative "normali", il conteggio dei passi risulta essere un buon indice per valutare la distanza percorsa. Ho parlato di condizioni operative normali, considerando che

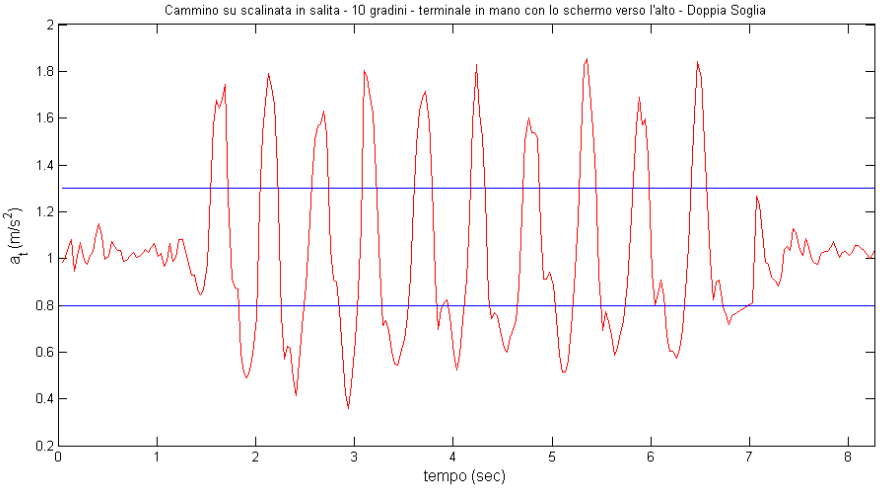


Figura 3.27: Scala in salita: accelerazione totale

l'utente che vuole sfruttare un applicazione di positioning, non ha alcun interesse ad "ingannare" il sistema, ad esempio simulando una camminata che in realtà non è effettuata.

L'altra parte essenziale del metodo descritto è la **lunghezza del passo**. In alcuni podometri, questa lunghezza viene inserita direttamente; di solito i manuali dei contapassi suggeriscono di effettuare 10 passi, calcolare la distanza percorsa e dividere per 10; altra tecnica suggerita è di misurare direttamente la distanza di un passo, tra l'appoggio di un tallone e quello dell'altro tallone. In altri podometri, si inserisce l'altezza e viene adoperato un opportuno fattore di scala moltiplicato per l'altezza dell'individuo. Tale fattore è intorno a 0.4 e può dipendere da diversi parametri, ad esempio dal sesso della persona. Un importante considerazione da fare, è che in alcuni studi (si veda [33]) si considera la lunghezza del passo come la distanza tra due appoggi successivi a terra dello stesso tallone. Mentre la lunghezza del singolo passo è chiamata lunghezza del passo anteriore; praticamente però, come detto, quando viene richiesta la lunghezza del passo, ci si riferisce al caso di singolo passo.
Nei casi descritti si considera quindi, che il passo della camminata sia costante e basato solo sui dati fisici dell'utente. Un altro approccio che è possibile seguire è quello di considerare i passi di lunghezza variabile, in cui la lunghezza del passo è scelta di volta in base alle misurazioni effettuate dai sensori di accelerazione. Un approccio di questo tipo è stato fatto in S. H. Shin [34], in cui si sono utilizzati dei sensori MEMS. Per il calcolo del numero di passi viene utilizzato il metodo zero-crossing, come da figura 3.28.

La lunghezza del passo l_k è calcolata considerando la frequenza della camminata

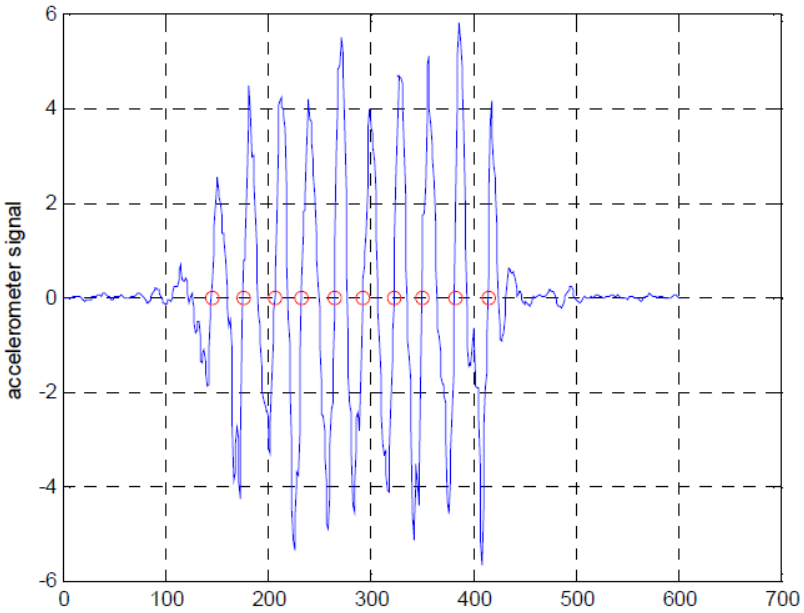


Figura 3.28: Conteggio numero dei passi con metodo Zero-Crossing ([34])

e la varianza dell'accelerazione, come mostrato nell'equazione 3.6.

$$l_k = \alpha f_k + \beta v_k + \gamma \tag{3.6}$$

$$f_k = \frac{1}{(t_k - t_{k-1})} \tag{3.7}$$

$$v_k = \sum_{t=t_k-1}^{t_k} \frac{(a_t - \bar{a_k})^2}{N} \tag{3.8}$$

Dove:

α, β e γ sono dei parametri derivanti dalla fase di calibrazione;
l_k è la lunghezza del k-esimo passo;
t_k è il tempo al passo k-esimo;
f_k è la frequenza della camminata, tra il (k-1)-esimo e il k-esimo passo;
$\bar{a_k}$ è la media dell'accelerazione durante un passo;
a_{t_k} è l'accelerazione al tempo t_k;
N è il numero di output dei sensori durante un passo;
v_k è la varianza dell'accelerometro.

La distanza percorsa D dopo n passi è valutata come:

$$D = \sum_{i=1}^{n} (\alpha f_i + \beta v_i + \gamma) \tag{3.9}$$

I risultati degli esperimenti (S. H. Shin [34]), sono mostrati in figura 3.29 e mostrano la lunghezza del passo al variare della frequenza della camminata e della varianza dell'accelerazione. Secondo lo studio effettuato in S. H. Shin [34], sfruttando l'algoritmo del passo adattativo descritto, per una camminata in pianura e nel peggiore dei casi, si raggiungono accuratezze del 95% sulla stima del passo. Inoltre dallo studio, emerge che le condizioni del fondo su cui si cammina determinano in maniera decisiva l'accuratezza sulla stima della lunghezza del passo.

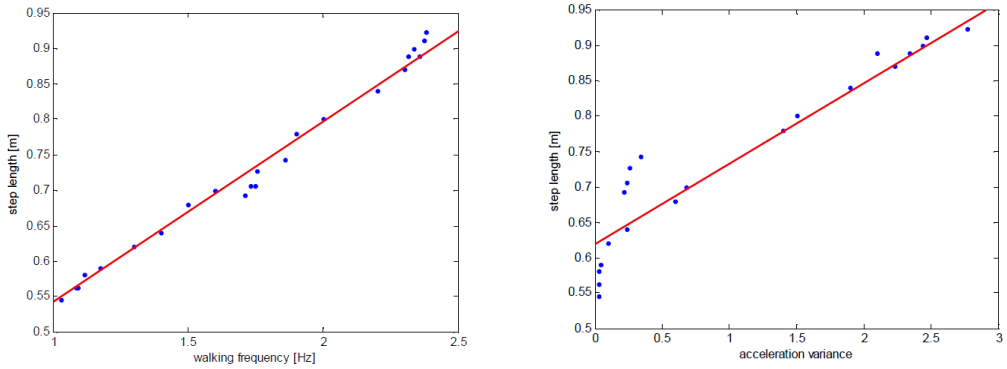


Figura 3.29: Frequenza della camminata versus lunghezza del passo. Varianza dell'accelerazione versus lunghezza del passo

3.5 Calcolo della direzione

Nel paragrafo 3.3.4, si è spiegato il funzionamento della terna $(Roll, Pitch, Yaw)$ determinata sfruttando i tre sensori inerziali: giroscopio, magnetometro e accelerometro. Tuttavia in alcune applicazioni, si preferisce esprimere la terna $(Roll, Pitch, Yaw)$ in un modo leggermente diverso utilizzando solo le misurazioni derivanti da giroscopio e accelerometro. Infatti, come si è detto nel paragrafo 3.3.3, le misurazioni del campo magnetico sono soggette ad interferenze che causano delle variazioni sul campo magnetico misurato con conseguenti errori sull'indicazione del nord magnetico. Ovviamente senza utilizzare le informazioni provenienti dal magnetometro non sarà possibile avere indicazioni sul nord geografico. In questi casi, all'avvio dell'applicazione, viene fissato il sistema di riferimento, come se in quel momento stesse indicando il nord magnetico. In pratica quindi si ha che il valore yaw è pari a 0 all'inizio delle misurazioni. L'applicazione "xSensor Pro" sfrutta questo principio.

Per il calcolo della direzione si sfrutta la variazione dello Yaw. Il primo test è stato effettuato in condizione di cammino rettilineo per una lunghezza di 7 metri, con il terminale in mano, senza farlo oscillare eccessivamente. Il terminale è tenuto così che l'asse y sia frontale (presa verticale), analogamente ai test effettuati in precedenza. In figura 3.30 sono graficati i valori misurati nel tempo di questo primo test. L'applicativo xSensor Pro fornisce i dati in radianti, questi sono stati convertiti in gradi e graficati tramite Matlab.

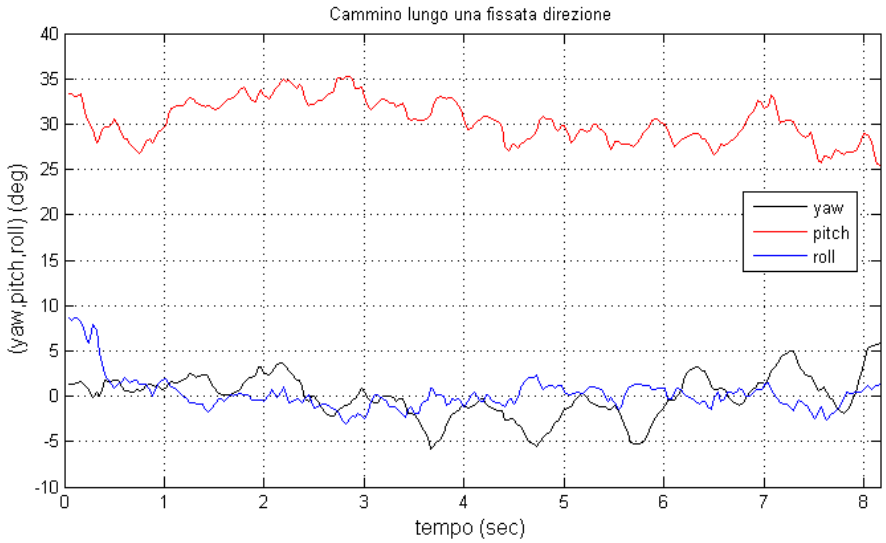


Figura 3.30: Terna (yaw,roll,pitch) durante cammino rettilineo

Dal grafico in figura 3.30 si vede che il valore yaw (curva in nero), che è il valore usato come riferimento per calcolare la variazione di direzione durante la camminata, oscilla intorno allo 0, con scostamenti che non superano i 5 gradi in modulo. Il programma è stato avviato, quando il terminale era in mano, cosi che yaw fosse nullo in partenza. Si sarebbero ottenute le medesime oscillazioni (ovviamente ripetendo il test nelle stesse condizioni) e gli stessi risultati, anche se il valore di partenza di yaw fosse differente, ad esempio se si partiva con un valore yaw pari ad x, si sarebbero osservate le oscillazioni intorno a tale valore. Infatti l'interesse ricade sulle variazioni dello yaw a partire da un assetto noto. Il fatto che il valore pitch (andamento in rosso) sia intorno ai 30° significa che il terminale era inclinato, lungo l'asse x, per osservare il display.

Nella figura 3.31 si fa un focus sull'andamento temporale del solo parametro yaw, inserendo il valor medio e il valor medio istantaneo. É interessante notare come il valor medio (curva blu), sia praticamente nullo.

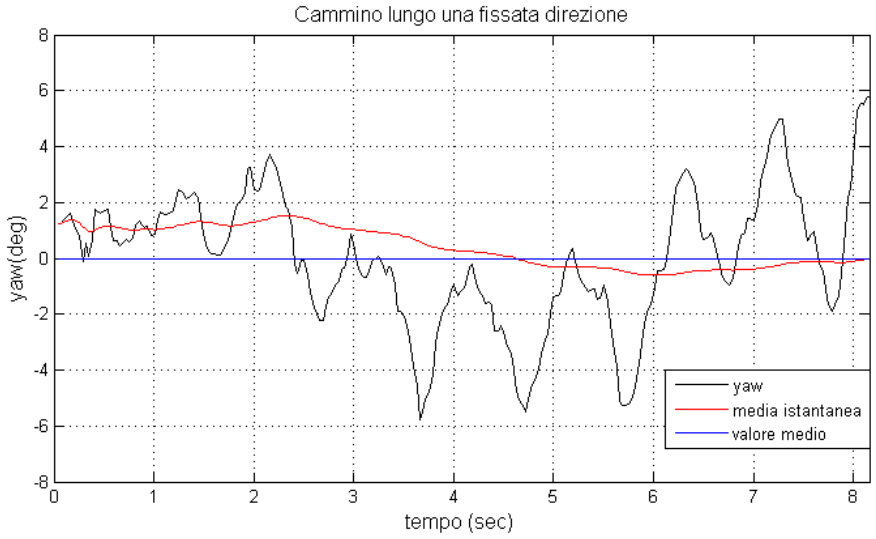


Figura 3.31: yaw, media istantanea e valore medio durante cammino rettilineo

Adesso ripetiamo il test, effettuando una svolta di 90° a destra. Più precisamente, si camminerà lungo la stessa direzione per circa 5 metri e poi si svolterà a destra, percorrendo circa 3 metri. La curva verrà fatta in modo "usuale", ovvero senza brusco cambiamento di direzione, ma girando lentamente. I risultati ottenuti, sono mostrati in figura 3.32.

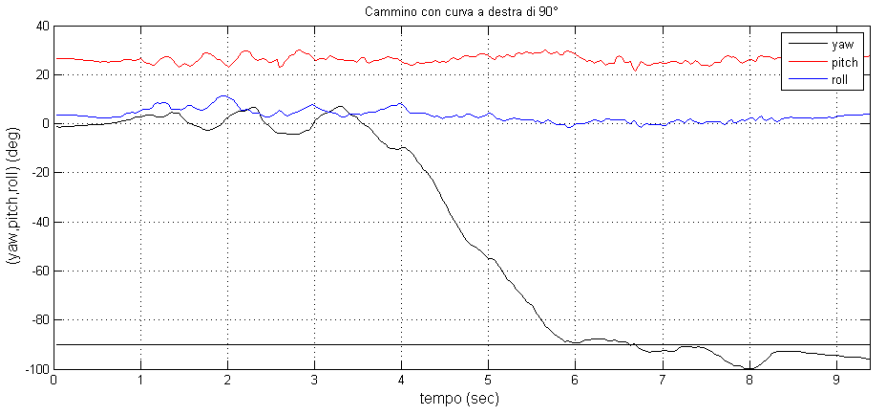


Figura 3.32: Terna (yaw,roll,pitch) durante curva a destra di 90°

In accordo a quanto detto nel paragrafo 3.3.4, si vede quindi che una rotazione in senso orario, intorno all'asse z genera un valore di yaw negativo. In particolare, dal grafico in figura 3.32, si vede come il valore di yaw, si avvicina intorno ai 90°. Bisogna considerare che nei test effettuati si commettono degli errori dovuti al posizionamento vero e proprio del terminale, ovvero nell'ultimo test si è percorsa una curva in maniera tale che tra la disposizione iniziale e quella finale del terminale, lungo l'asse y ci sia un angolo retto, in realtà non è stato utilizzato nessuno strumen-

to per valutare tale angolo, quindi si saranno commessi degli errori di disposizione. Per quanto riguarda gli errori sui calcoli angolari, ovvero sul parametro yaw, vale quanto detto nel paragrafo 3.3.5. Quindi i test effettuati vanno interpretati in linea concettuale.

Ripetiamo l'ultimo test proposto, in condizioni analoghe, però considerando una curva a sinistra di 90°. In questo test, si è mantenuta la direzione finale per un maggiore distanza, ovvero il cambio di direzione è effettuato in modo più repentino, mantenendo nella fase finale una camminata rettilinea. I risultati sono mostrati in figura 3.33.

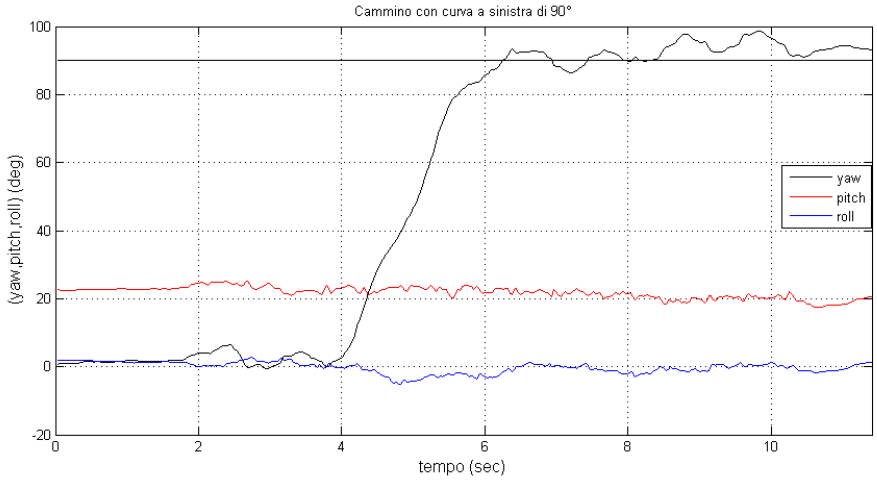


Figura 3.33: Terna (yaw,roll,pitch) durante curva a sinistra di 90°

3.6 Stima della nuova posizione

Per il calcolo della nuova posizione è sufficiente unire i dati risultanti dai calcoli della distanza percorsa e dell'orientamento. Il procedimento è il seguente:
Suppongo che sia *noto l'assetto iniziale*, per cui conosco la posizione del terminale nel sdr della mappa e l'orientamento. L'assetto iniziale è noto grazie al processo di autolocalizzazione effettuato mediante fotografia al codice bidimensionale. Gli aspetti legati al processo di autolocazzazione saranno meglio chiariti nel capitolo 4. Per adesso è sufficiente considerare che ad ogni codice è legata una determinata coordinata (a, b) e un angolo α indispensabile per l'orientamento (angolo rispetto all'asse delle x). Il sdr della mappa è composto da un asse cartesiano, in cui si avrà un certo rapporto pixel/metro sugli assi, ovvero ogni metro sarà rappresentato da un fissato numero di pixel. Per una mappa che rispecchia correttamente le proporzioni lungo i due assi, i rapporti pixel/metro sono i medesimi lungo i due assi. La mappa è bidimensionale e non sono interessato alla terza coordinata (quota). Indico con

(a', b') la posizione che si vuole stimare e con D la distanza percorsa in pixel in un certo intervallo di tempo. La situazione descritta è presente in figura 3.34 in cui si è considerata la mappa del piano terra della facoltà di Ingegneria in San Pietro in Vincoli.

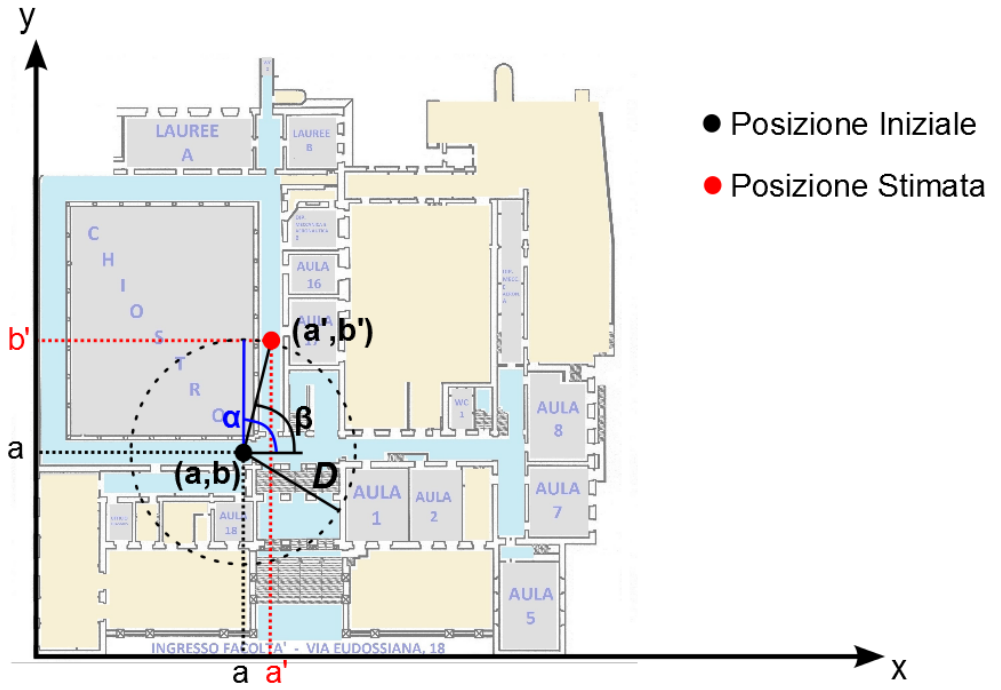


Figura 3.34: Stima della nuova posizione - sullo sfondo la mappa del piano terra della facoltà di Ingegneria in San Pietro in Vincoli

La nuova posizione può essere valutata dalla 3.10, in cui l'operatore round approssima all'intero più vicino.

$$\begin{cases} a' = a + round[D\ \cos(\beta)] \\ b' = b + round[D\ \sin(\beta)] \end{cases} \tag{3.10}$$

Un importante considerazione che va fatta è relativa all'intervallo di tempo che intercorre fra il posizionamento iniziale e la stima successiva o fra due stime; si potrebbe pensare di effettuare la stima ad ogni passo, oppure effettuarla dopo n passi. I concetti introdotti in questo paragrafo, saranno applicati nel prossimo paragrafo, in cui si effettueranno dei test sul dead reckoning.

3.7 Dead Reckoning Test

In questa sezione si riportano i risultati ottenuti in fase di test in accordo a quanto detto nei paragrafi precedenti: sono state effettuate 3 prove per ognuno dei quattro differenti percorsi, come indicato in figura 3.35.

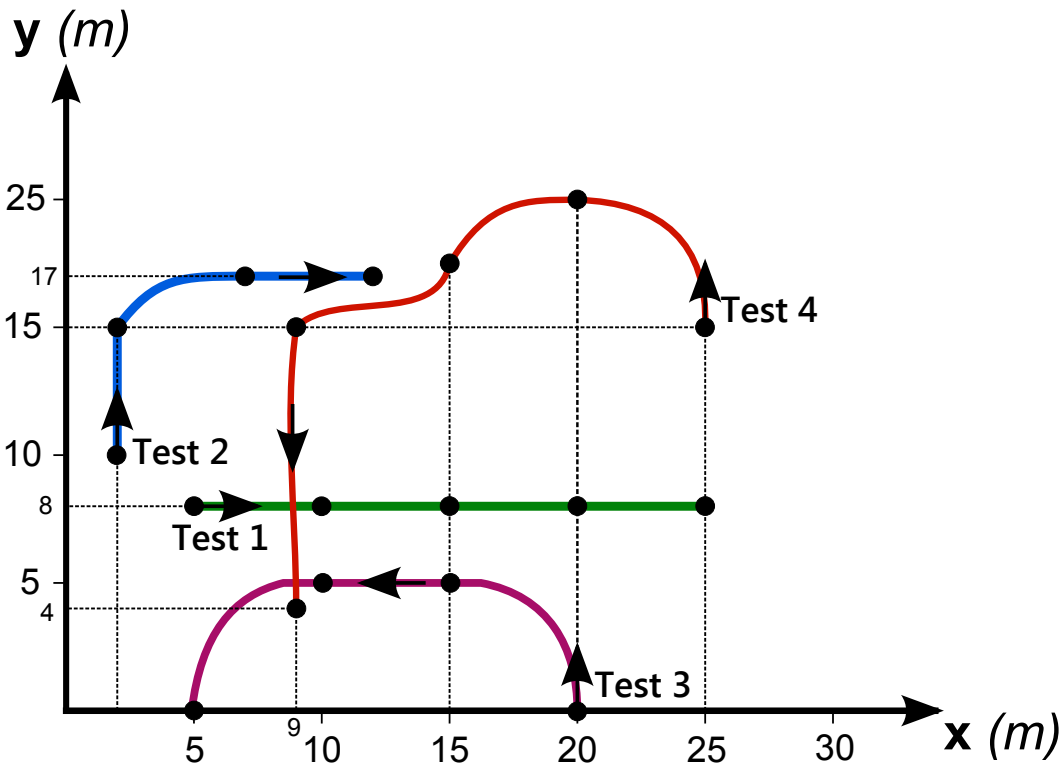


Figura 3.35: I quattro percorsi

Le condizioni operative sono le seguenti:

- Per la valutazione della posizione si sono sfruttate le misurazioni dell'accelerometro e del giroscopio dello smartphone iPhone 4, estrapolate mediante l'applicazione 'xSensor Pro'. Si considera una frequenza temporale delle misurazioni di 32 Hz.

- Il terminale si è impugnato con la mano destra, con l'asse delle y circa parallelo alla direzione della camminata (presa verticale), con la "classica" inclinazione dovuta all'osservazione del display. I risultati che si ottengono impugnando il terminale con l'asse delle x parallelo alla direzione della camminata (presa in orizzontale) sono i medesimi. Infatti, come detto in precedenza, è possibile capire il posizionamento dello smartphone osservando i valori di accelerazione percepiti rispetto all'accelerazione gravitazionale. Questo implica che

l'applicazione per iPhone può funzionare sia in vista orizzontale che in vista verticale.

- Si è considerata una lunghezza del passo costante e pari a $D = 0.67\,m$. Per la valutazione si è misurata la lunghezza di 10 passi, dividendo poi la lunghezza totale per 10. Sono state effettuate 10 diverse prove, scartando i due risultati più bassi e più alti ed è stata poi calcolata la media sulla base dei valori rimanenti.

- I test sono stati eseguiti in un ambiente di circa $600\,m^2$.

- Sono note le coordinate di partenza e la direzione rispetto al sistema di riferimento (x, y) (assetto del sdr body rispetto al sdr della navigazione noto).

- Per ogni test sono presenti dei punti intermedi, oltre ai punti di partenza e di arrivo. Tali punti sono stati presi come riferimento durante la camminata, ovvero il percorso è stato eseguito passando sopra questi punti. Questi punti verranno chiamati nodi del percorso.

- Sono stati memorizzati gli intervalli di tempo tra i passaggi ai diversi nodi del percorso, mediante un cronometro esterno in grado di registrare più intertempi: Le misurazioni sono state effettuate con la seguente procedura:

  - al tempo t=0, start del cronometro e delle misurazioni dei sensori inerziali (start alla registrazione nell'applicazione xSensor Pro);
  - primo passo effettuato dopo circa 1 secondo dallo start;
  - registrazione cronometrica degli intertempi al passaggio ai nodi del percorso;
  - stop del cronometro al passaggio sul nodo di arrivo.
  - in seguito al passaggio sul nodo di arrivo, stop alle misurazioni dei sensori inerziali: si considerano le sole misurazioni dei sensori effettuate nel tempo registrato dal cronometro, ovvero si prendono in esame le misure inerziali entro il tempo di percorrenza del percorso.

- Per la valutazione del numero di passi effettuati si sono considerate le accelerazioni lungo gli assi y e z: $a_{yz} = \sqrt{(a_y)^2 + (a_z)^2}$, a differenza di quanto fatto fatto prima in cui si è considerata l'accelerazione lungo tutti e tre gli assi.

- Per la valutazione della direzione si è considerato il valore yaw. Anche se il terminale si fosse tenuto in presa orizzontale si sarebbero considerate le variazioni dello yaw.

- In Appendice A è presente il codice Matlab utilizzato. In particolare è presente il codice del test 1.B. Negli altri test il codice è il medesimo, si sono modificate solo i dati relativi agli intertempi, coordinate dei nodi del percorso e direzione.
- Le soglie sono costanti in tutti i test e sono pari a:
  - $soglia_{inf} = 0.85m/s^2$
  - $soglia_{sup} = 1.25m/s^2$
- Oltre alla componente di errore derivante dalle misurazioni, bisogna considerare anche altre componenti di errore:
  - camminata eseguita su un percorso diverso dal percorso prestabilito, ovvero il terminale non si muove esattamente lungo il percorso prestabilito;
  - errori nella determinazione delle coordinate dei nodi del percorso;
  - errori nella valutazione (esterna) degli intertempi.
- Non ci sono errori di valutazione dell'assetto iniziale o quantomeno possono essere considerati trascurabili.
- I percorsi sono individuati dai numeri $1, 2, 3, 4$ mentre le tre prove per ogni percorso dalle lettere A, B, C.

Si riportano nel seguito i risultati dei 4 test, in particolare i risultati dei test 1.B, 2.B, 3.B e 4.B. Per ognuno dei 4 test, si riportano tre grafici:

1. Nel primo grafico è riportato, in rosso, l'andamento dell'accelerazione a_{yz} che è la somma quadratica delle accelerazioni misurate lungo gli assi y e z. Con i colori blu e verde sono rappresentate rispettivamente la soglia superiore e inferiore. I segmenti verticali viola, rappresentano gli istanti di passaggio nei nodi del percorso (misurati tramite cronometro esterno). Nella parte bassa del grafico, in nero, sono invece tracciati gli istanti temporali in cui viene "percepito" un nuovo passo dall'algoritmo.
2. Nel secondo grafico sono mostrati gli andamenti dello yaw istantaneo (curva blu) e dello yaw mediato al tempo t (in rosso). Anche in questo caso sono indicati i valori temporali degli intertempi in viola.
3. Nell'ultimo grafico, si riportano le stime delle posizioni effettuate ad ogni passo (cerchi blu) e i punti dei nodi di passaggio (asterischi rossi). La stima della nuova posizione è eseguita utilizzando le equazioni 3.10 ed è basata unicamente sulla posizione precedente. Nel grafico relativo al test rettilineo (Test 1), sono riportate anche le stime basate sulla media istantanea dello yaw (croci in rosso).

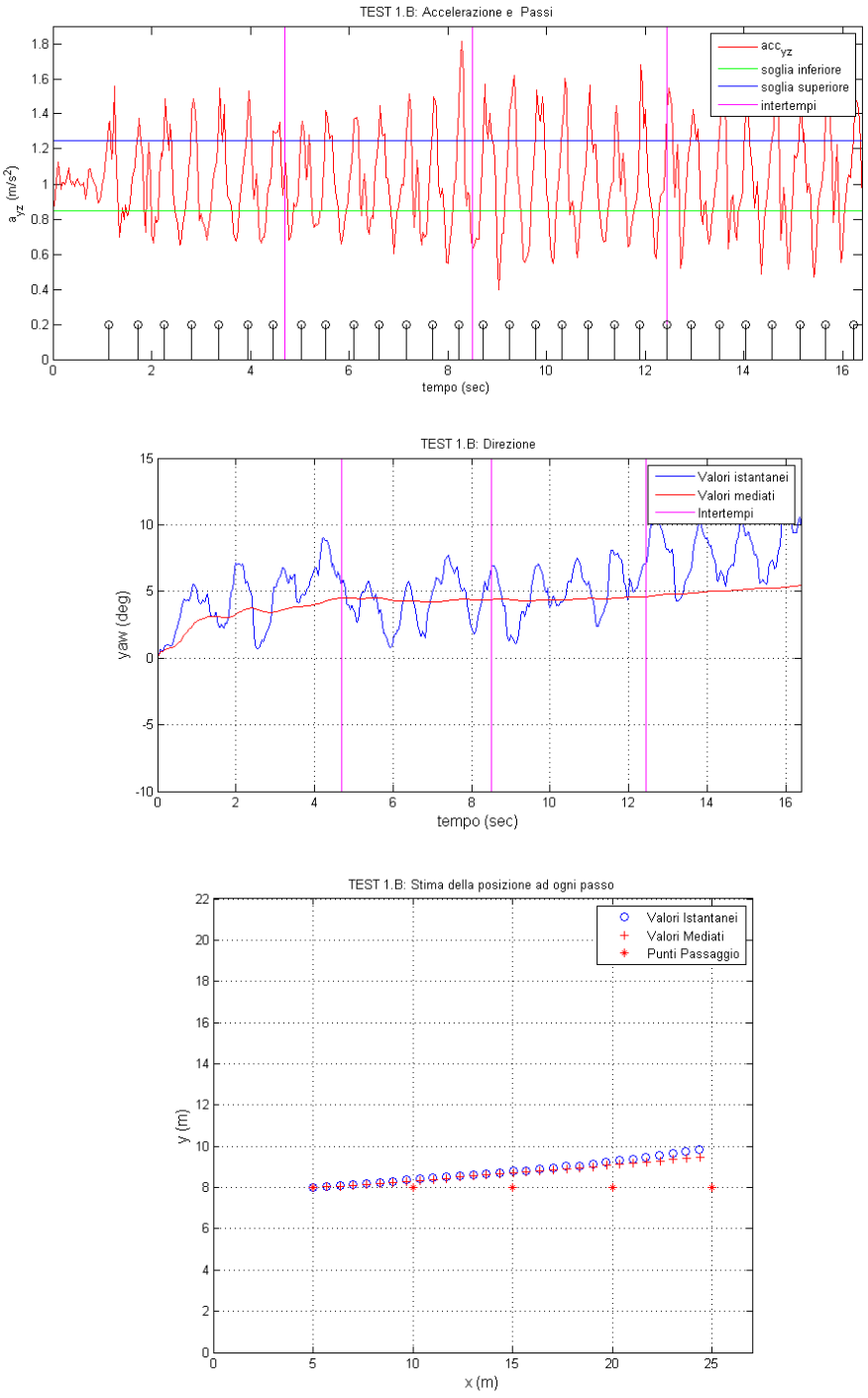


Figura 3.36: Risultati del Test 1.b: percorso rettilineo

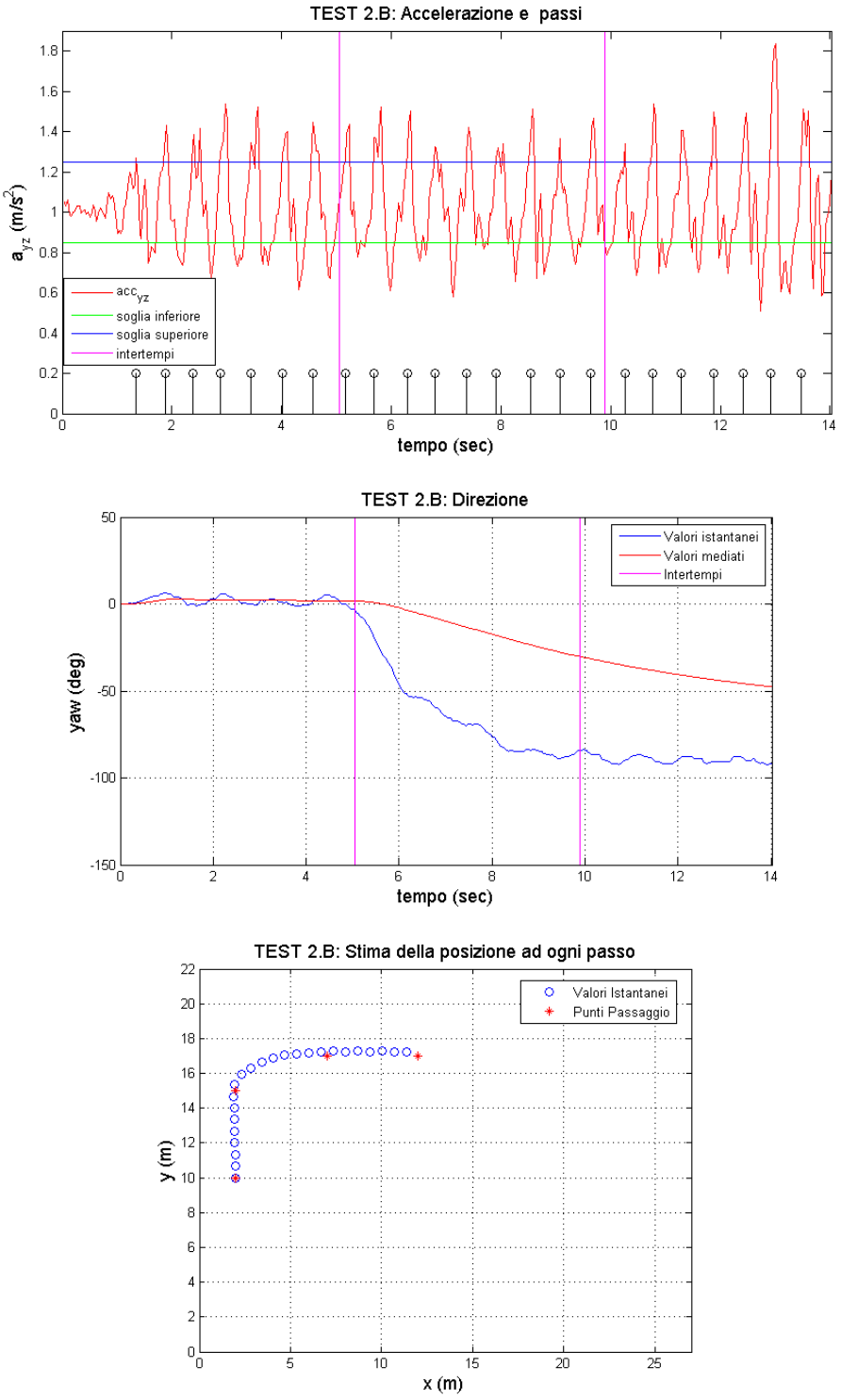


Figura 3.37: Risultati del Test 2.b: percorso con curva a destra

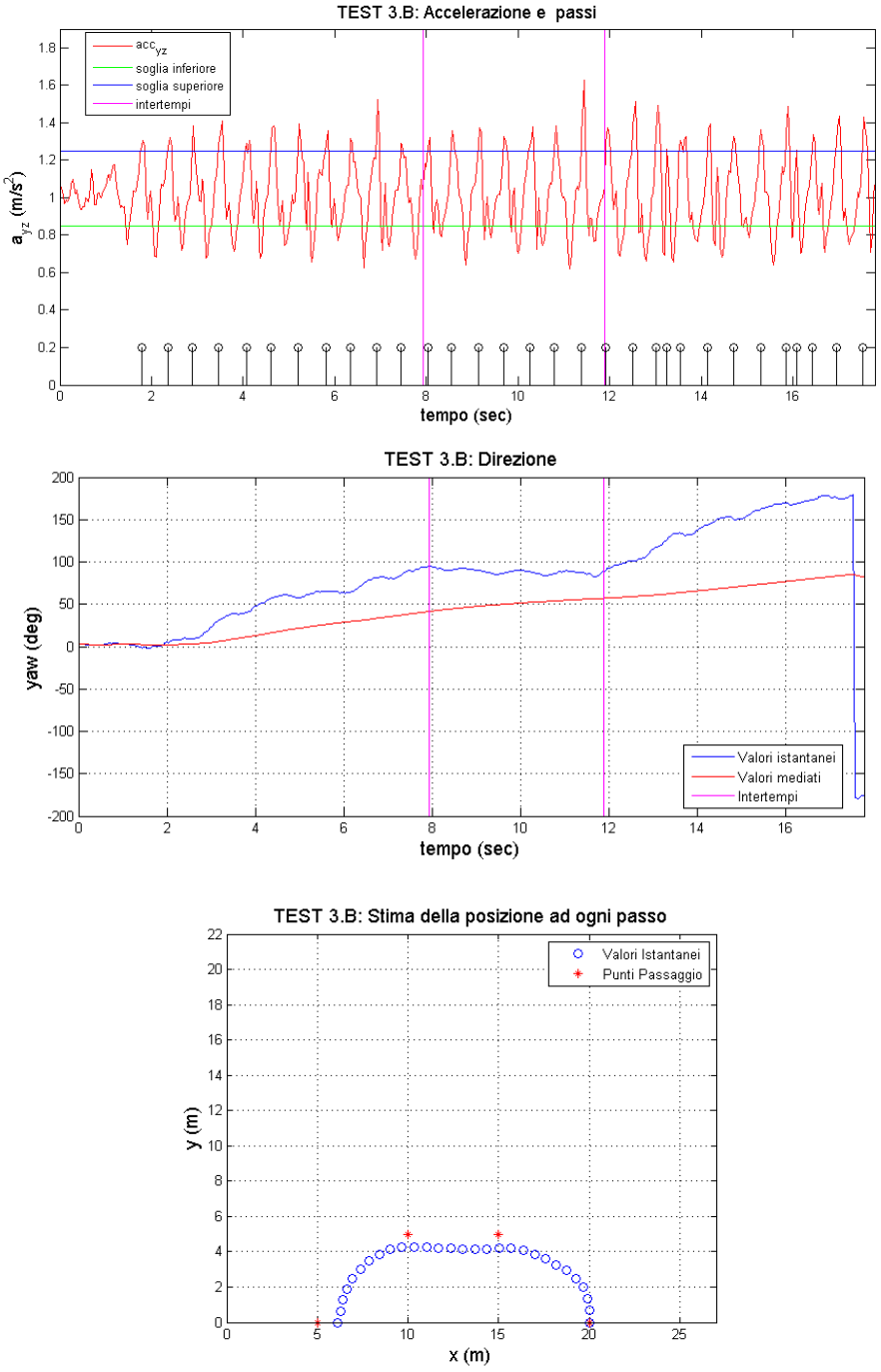


Figura 3.38: Risultati del Test 3.b: percorso con doppia curva a sinistra

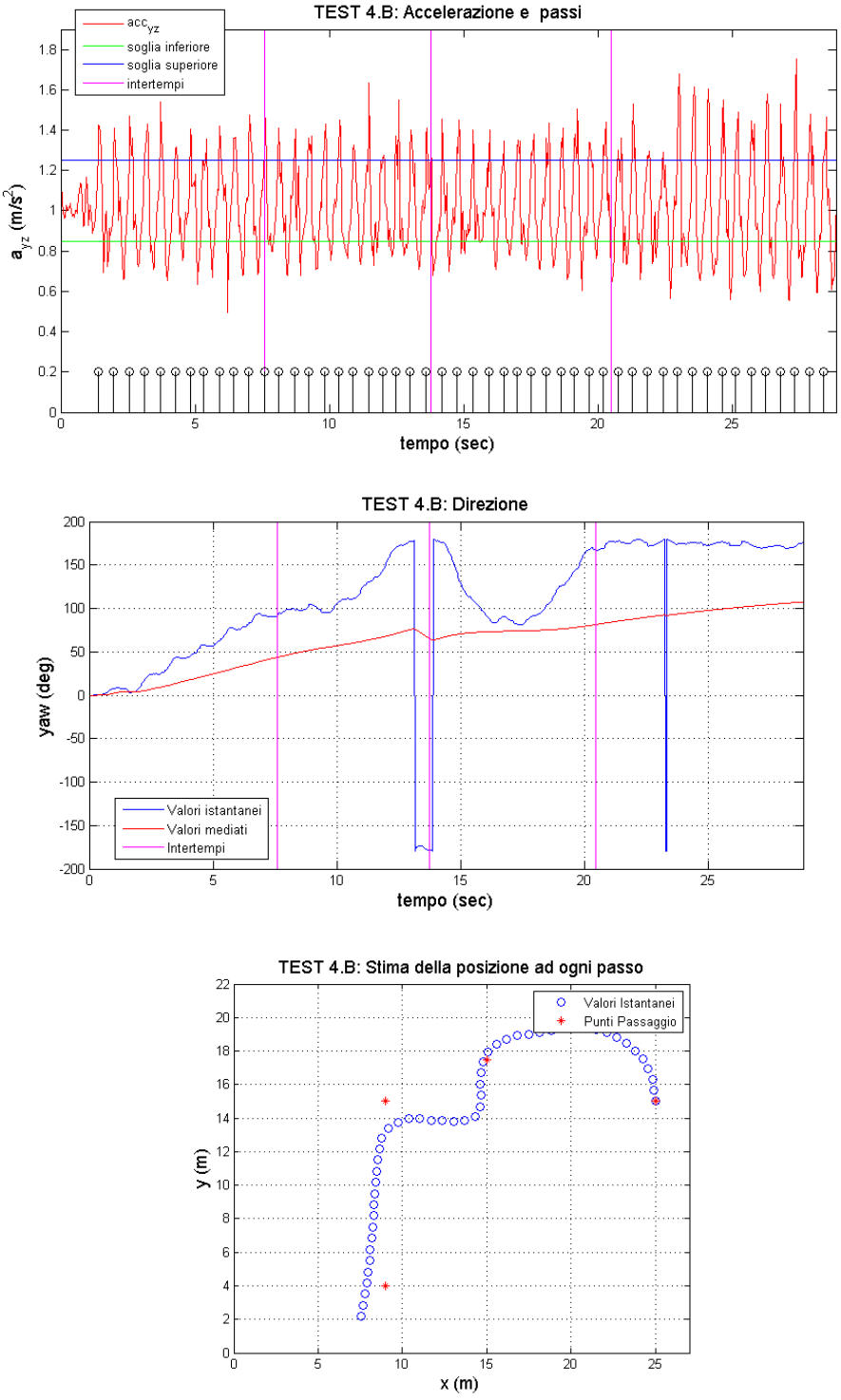


Figura 3.39: Risultati del Test 4.b: percorso misto con 4 curve

Nel seguito si riportano invece le stime delle posizioni ottenute nei tre test di ogni percorso. Si riportano anche il numero totale di passi calcolati durante il percorso $(cont_a, cont_b, cont_c)$ e le coordinate dell'ultima stima di posizione effettuata (x_a, y_a), (x_b, y_b) e (x_c, y_c).

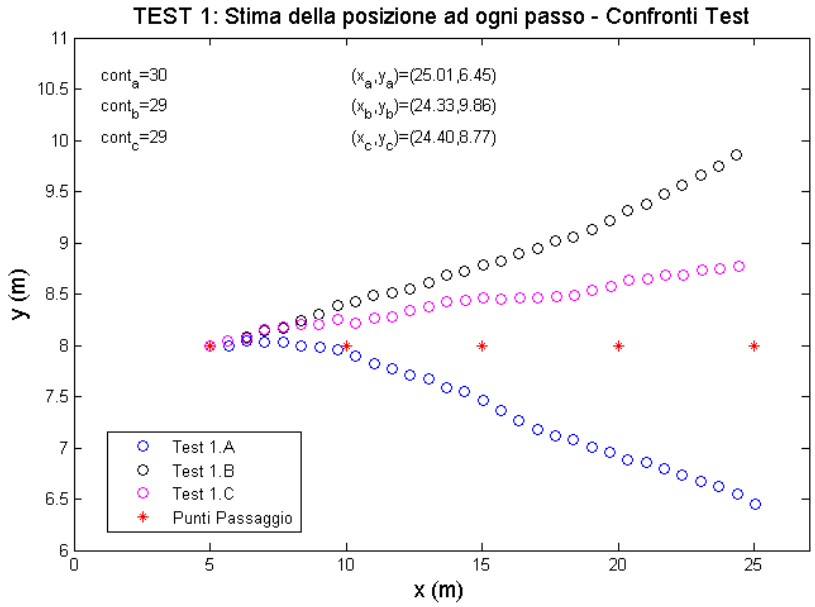


Figura 3.40: Stime di posizione ottenute nelle tre prove del Test 1 - numero di passi contati e coordinate finali

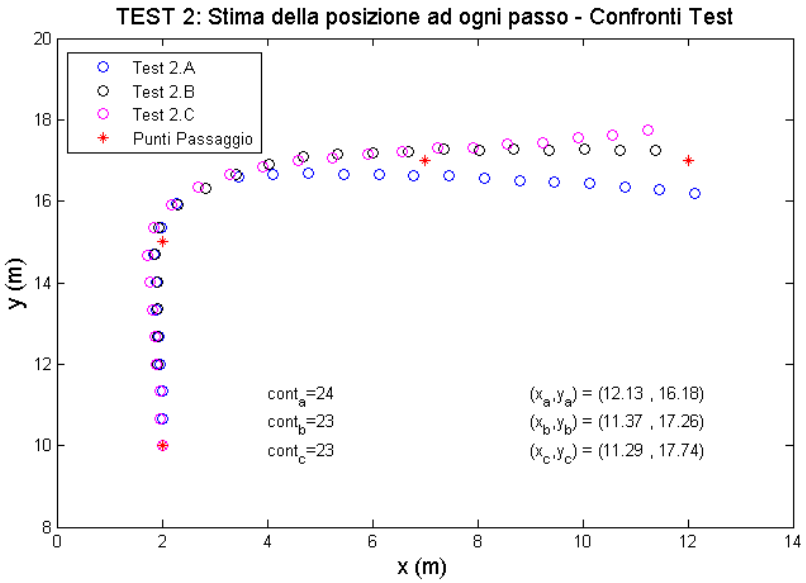


Figura 3.41: Stime di posizione ottenute nelle tre prove del Test 2- numero di passi contati e coordinate finali

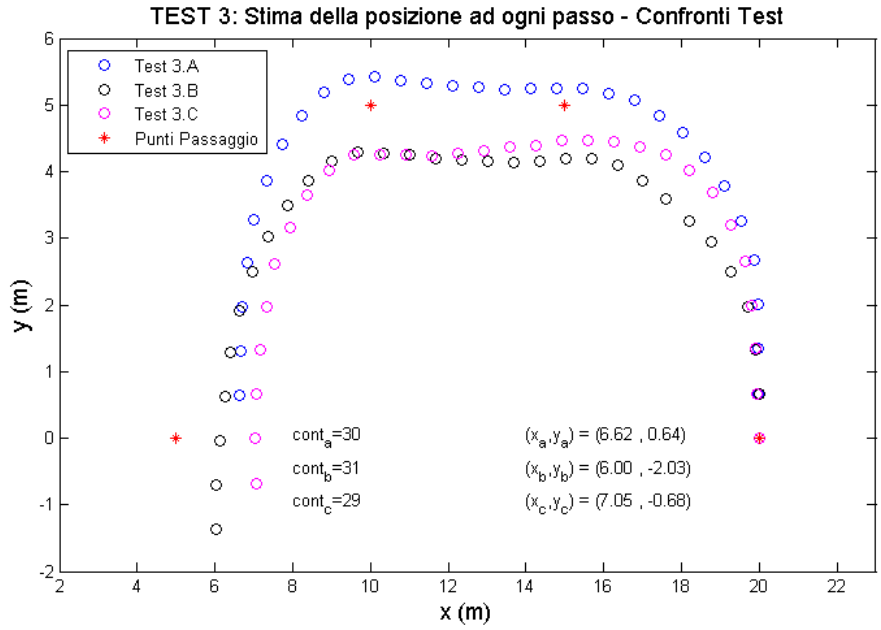


Figura 3.42: Stime di posizione ottenute nelle tre prove del Test 3- numero di passi contati e coordinate finali

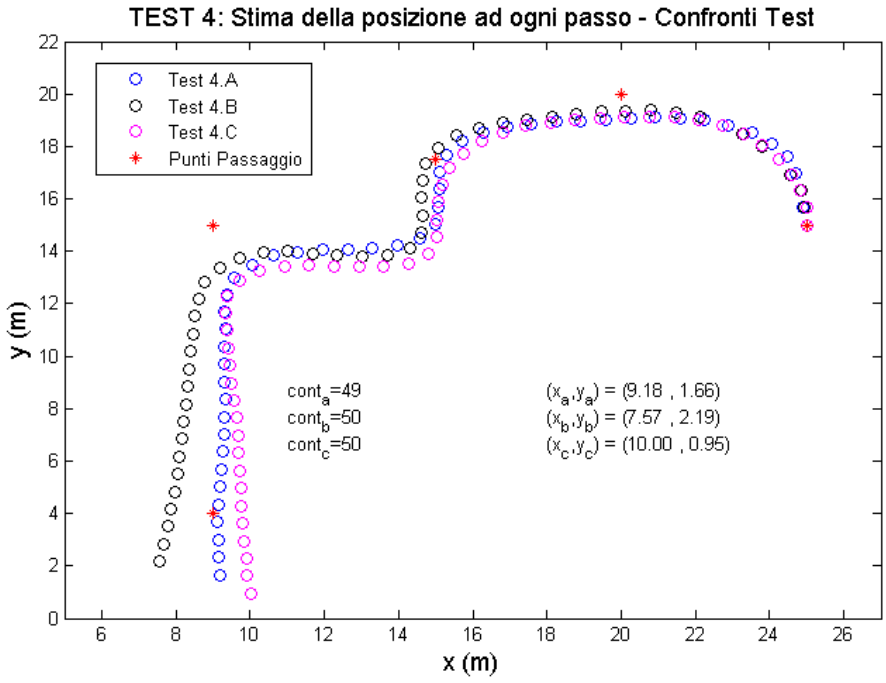


Figura 3.43: Stime di posizione ottenute nelle tre prove del Test 4- numero di passi contati e coordinate finali

Analizzando i grafici e l'approccio considerato è possibile fare le seguenti considerazioni:

- La stima della nuova posizione viene effettuata ad ogni passo percepito e non istantaneamente o dopo n passi. Nel test 2.B, i cui risultati sono mostrati in figura 3.37, sono stati stimati 23 passi lungo il percorso, ovvero 23 passi dalla partenza all'istante di arrivo (misurato dal cronometro). Osservando il grafico relativo alle accelerazioni e passi, si vede come il tempo è stato "troncato" pochi istanti prima che venga percepito il nuovo passo (il 24-esimo). Una situazione analoga, si ha nel passaggio ai nodi intermedi, che avviene in un istante diverso, da quello in cui è stato valutato il passo. Per questi motivi, non è possibile valutare con esattezza l'errore commesso all'istante in cui è avvenuto il passaggio nei nodi del percorso. Nel seguito però si farà un analisi dell'errore, considerando una sorta di interpolazione temporale dei dati.

- L'errore sulla stima della posizione cresce con il passare del tempo, come detto nel paragrafo 3.2. Questo aspetto risulta particolarmente evidente, osservando le figure 3.40 e 3.41.

- In condizioni di camminata "normale", la valutazione della distanza risulta sufficientemente accurata. In particolare dalla 3.40, si vede come la corretta valutazione del numero di passi effettuati, produca un errore sulla distanza percorsa ridotto. Infatti nel test 1.A la distanza totale stimata, risulta essere di $30 * 0,67 = 20,1\,m$ a fronte dei 20 metri percorsi. Mentre nei test 2.A e 3.A la distanza totale stimata è pari a $29 * 0,67 = 19.43\,m$.

- Per quanto riguarda l'orientamento istantaneo, è possibile asserire che gli errori maggiori vengono commessi nelle situazioni in cui avviene una "rapida" variazione della direzione. Infatti nel percorso 2, il cambio della direzione è avvenuto più lentamente rispetto ai percorsi 3 e 4 e ciò ha determinato una migliore accuratezza dello yaw e una conseguente stima della posizione più accurata.

- Dai grafici sulla direzione delle figure 3.38 e 3.39, si nota come ci sia una discontinuità che fa saltare i valori di yaw di 360°. Questa situazione deriva proprio dalla definizione dello yaw (si veda 3.3.4), in cui una variazione di -180° e +180° è equivalente e corrisponde all'inversione della direzione del terminale. Per cui nel test 3.b si ha l'inversione della direzione, nella parte finale del percorso, mentre nel test 4.b ci sono due inversioni, nella parte centrale e in quella finale, ed è proprio in questi casi che si riscontra la discontinuità.

3.8 Conclusioni

In questo Capitolo è stato introdotto il metodo che si è deciso di utilizzare per risolvere il problema del positioning indoor. In particolare, si è parlato dei sensori inerziali e dei concetti legati al dead reckoning. La trattazione verrà completata nei prossimi due Capitoli. La soluzione proposta nasce dalla necessità di un sistema di localizzazione indoor a bassissimo costo e utilizzabile indipendentemente dal tipo di ambiente indoor. Infatti, esistono sul mercato diverse tecniche e tecnologie di localizzazione, ma il principale elemento che né limita l'utilizzo è legato al fattore costi/benefici, dato che spesso non c'è alcuna necessità reale a creare un sistema di localizzazione indoor. Per quanto riguarda il dead reckoning, è stato utilizzato un approccio alternativo da quello "classico" proposto in letteratura. Questo aspetto deriva direttamente dal metodo che considera il calcolo della distanza e della direzione come due processi indipendenti e dai limiti che presenta l'apparato sensoristico a basso costo dell'iPhone. Per la distanza si utilizza la sola informazione fornita dall'accelerometro, in particolare non è possibile utilizzare direttamente le misure di accelerazione per il calcolo della posizione; infatti si è visto come piccoli errori sul valore dell'accelerazione determinino errori sulla posizione inaccettabili. Per questo motivo si è scelto di valutare la distanza andando a rilevare i passi, fissando una coppia di soglie e con la conoscenza della lunghezza del passo. Per valutare la direzione invece, si fa uso dei dati sull'orientamento, per la precisione dello yaw sfruttando le API fornite direttamente dalla Apple. Sulla base dei concetti teorici introdotti, sono stati poi effettuati dei test per la stima della posizione ad ogni passo. Da questi test è emerso che l'errore commesso nella stima della posizione può essere considerato accettabile, considerando che anche dopo intervalli temporali di 15-20 secondi e nel caso peggiore non superano i 4 metri, rispetto alla posizione effettiva. L'approccio utilizzato, è un approccio semplice che tiene conto dei soli parametri essenziali. Sicuramente è possibile apportare al sistema molte migliorie; un esempio è l'uso di una lunghezza del passo variabile; altri esempio sono l'integrazione delle misure del giroscopio per la valutazione della distanza e l'integrazione delle misure di accelerazione per la valutazione della direzione.

Capitolo 4

Autolocalizzazione

Indice

4.1 Introduzione 93
4.2 Codici Bidimensionali 94
4.3 Positional Code 100
4.4 Nozioni di Geometria Proiettiva 103
4.4.1 Coordinate Omogenee 103
4.4.2 Omografia 104
4.5 Modelli di fotocamera 106
4.5.1 Modello semplificato 107
4.5.2 Modello generale 111
4.6 Conclusioni e SPinV Code 118

4.1 Introduzione

In questo capitolo si affronterà il tema dell'autolocalizzazione mediante fotografia al codice, focalizzando l'attenzione sugli aspetti di interesse: codici bidimensionali, aspetti di geometria proiettiva e aspetti legati alla fotocamera. Nel paragrafo 4.2 verrà fatta una panoramica sui codici bidimensionali presenti sul mercato, così da capirne le caratteristiche essenziali e in seguito verrà introdotto il concetto di codice posizionale (paragrafo 4.3). Nei paragrafi 4.4 e 4.5 si descriveranno delle nozioni di geometria proiettiva e dei modelli di fotocamera; l'obiettivo di questa analisi è capire come poter estrarre delle informazioni sulla posizione del terminale, come distanza e angolo di scatto, a partire dalla fotografia del codice. Inoltre, mediante la suddetta analisi geometrica è possibile migliorare la robustezza in fase di decodifica dello SPinV Code. Infine, nel paragrafo 4.6, sulla base delle considerazioni effettuate, verrà proposto un codice (SPinV Code). Alcune delle problematiche verranno solo analizzate da un punto di vista teorico.

4.2 Codici Bidimensionali

Nei codici bidimensionali, l'informazione è contenuta sia orizzontalmente che verticalmente, per cui con un codice 2D è possibile memorizzare la stessa quantità di dati, in uno spazio più ridotto, rispetto al tradizionale codice monodimensionale (codice a barre). I codici bidimensionali sono nati per rispondere alle esigenze di catalogazione dei prodotti, in cui erano richiesti dei codici sempre *più piccoli*, per poterli usare con prodotti di ridotte dimensioni, e di elevata *capacità*, così da caratterizzare in maniera più completa un determinato prodotto. Così ad esempio, un codice bidimensionale può contenere, oltre alle classiche informazioni legate al codice identificativo del prodotto, anche informazioni sulla data di produzione e scadenza di un prodotto alimentare, il numero di lotto di produzione e più in generale tutti i dati di interesse in ambito di catalogazione. I codici monodimensionali classici, possono essere decodificati mediante appositi lettori, sfruttando la tecnologia laser o led. I codici bidimensionali invece sono letti utilizzando dei sensori CCD (Charge Coupled Device), che sono dei sensori che permettono di catturare l'immagine e trasformarla in un segnale elettrico di tipo analogico. I CCD sono i sensori su cui sono basate le fotocamere digitali, per cui il CCD sostituisce la pellicola fotosensibile delle macchine fotografiche analogiche. É proprio questo aspetto che ha consentito un enorme diffusione dei codici bidimensionali in ambiti diversi da quello originario (catalogazione prodotti). In particolare, negli ultimi anni, si sono integrati dei lettori di codici bidimensionali anche negli smartphone sfruttando la fotocamera integrata e appositi applicativi . Attualmente è possibile inserire all'interno dei codici bidimensionali, informazioni di vario tipo: URL, numeri di telefono, indirizzi mail, messaggi informativi, informazioni geografiche.... Così l'utente potrà interagire direttamente con queste informazioni, scattando una foto al codice ed evitando di dover digitare manualmente i dati. Così, ad esempio, tramite appositi applicativi, la scansione di un codice bidimensionale, contenente una Url, porterà direttamente all'apertura del link.

Figura 4.1: Esempio di codice monodimensionale e bidimensionale

I codici bidimensionali possono essere divisi in tre categorie (si vedano [35], [36]) come mostrato in figura 4.2:

- codici a barre multipli;
- codici stackable;
- codici a matrice;

Figura 4.2: Evoluzione dei codici bidimensionali: codici a barre multipli, codici stackable e codici a matrice

I codici a barre multipli presentano il principale svantaggio della scansione multipla, in fase di decodifica, dovuta alla struttura impilata. Nei codici stackable una sola scansione è sufficiente a decodificare il codice, ma presentano il grosso inconveniente che l'apparato usato per la scansione deve essere allineato con il codice. I codici a matrice possono invece essere decodificati con una singola scansione e non è richiesto un allineamento così accurato come nei codici stackable.

Nei codici bidimensionali a matrice l'informazione è associata a una superficie suddivisa in piccoli spazi bianchi e neri: si sfrutta il contrasto fra queste due tinte acromatiche, così da realizzare sistemi di lettura semplici ed affidabili. Sono presenti sul mercato circa una ventina di codici bidimensionali; nella tabella 4.1 si elencano le peculiarità di alcuni tra i più diffusi: MaxiCode, DataMatri, QrCode e si considera anche un codice stackable (PDF417).

Il codice bidimensionale più diffuso è sicuramente il Qr Code (Quick Responce); tale diffusione può essere facilmente compresa, valutando le caratteristiche elencate in tabella 4.1. Le caratteristiche del qr code sono descritte nello standard ISO/IEC 18004 (riferimento [37]), ne vediamo alcune:

- Esistono attualmente 40 versioni di qr code, ognuna caratterizzata da fissate dimensioni e capacità; al crescere della versione cresce la dimensione (di 4

Codice	PDF417	DataMatrix	MaxiCode	QrCode
Sviluppatore (Paese)	Symbol Technologies (USA)	RVSI Acuity CiMatrix (USA)	UPS (USA)	DENSO (Japan)
Tipo di Codice	stackable	matrice	matrice	matrice
Dati Numerici	2710	3166	138	7089
Dati Alfanumerici	1850	2355	93	4298
Dati Binari (Byte)	1018	1556		2953
Caratteri Cinesi	554	778		1817
Principali Caratteristiche	Alta capacità	Ridotta area di stampa	Alta velocità di scansione	Alta capacità. Ridotta Area di stampa. Alta Velocità di scansione

Tabella 4.1: Caratteristiche di alcuni tipi di codici bidimensionali

moduli semplici a versione) e la capacità; Per ogni versione esistono 4 possibili livelli di correzione (ECC: L, M, Q, H). la versione 1, con ECC pari ad L, presenta una dimensione di 21x21 moduli semplici e può contenere fino a un massimo di 41 caratteri numerici, 25 alfanumerici e 17 Byte. La versione 40, con correzione di errore L, ha una dimensione di 177x177 e le sue prestazioni sono descritte nella tabella 4.1.

- I moduli elementari bianchi e neri rappresentano rispettivamente i bit 0 e 1.

- La correzione di errore è attuata mediante codifica di Reed Solomon in quattro possibili livelli. Con i livelli L, M, Q e H in grado di ripristinare rispettivamente il 7%, il 15%, il 25% e il 30% delle parole affette da errore.

- Nel Qr code, come in tutti i codici bidimensionali, è presente un *modello di rivelazione della posizione*, che permette di identificare il codice all'interno dell'immagine acquisita. In particolare, come mostrato in figura 4.3, sono presenti dei pattern in tre dei quattro angoli, che indicano la corretta direzione di lettura. Attraverso opportuni algoritmi, che sfruttano questi pattern, è possibile decodificare il codice, indipendentemente dalla rotazione.

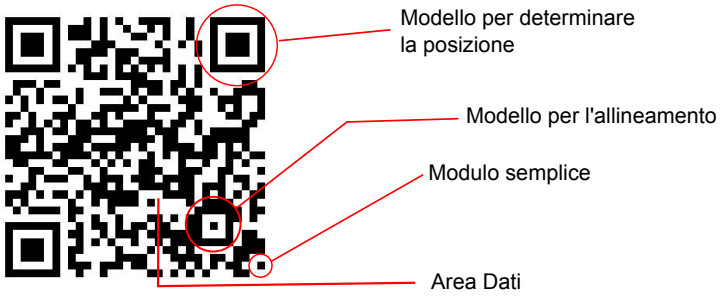


Figura 4.3: Struttura del Qr Code - nell'immagine un QR code versione 4

- Sono presenti inoltre dei pattern per l'allineamento (ad eccezione della versione 1). Il numero di questi pattern cresce con la versione, ad esempio nelle versioni 2, 10, 25, 40 sono presenti rispettivamente 1, 6, 22 e 46 pattern di allineamento. Attraverso un apposito algoritmo ed entro certe condizioni operative, è possibile riallineare il codice, ad esempio quando il codice non viene fotografato frontalmente e si ha una distorsione dell'area del codice.

- Accanto a due dei tre pattern usati per rivelare la posizione, sono presenti le informazioni relative alla versione del codice.

La decodifica dei Qr Code può essere realizzata utilizzando la libreria Zxing. Questa è una libreria open source completamente scritta il linguaggio Java, in grado di decodificare oltre ai Qr Code, una grande varietà di codici monodimensionali e bidimensionali. Zxing include diverse componenti che facilitano l'utilizzo diretto della libreria; ad esempio, nelle piattaforme Android è presente l'applicativo "Bar-Code Scanner", che utilizza la libreria Zxing, in pratica questo strumento può essere utilizzato all'interno di qualsiasi applicazione per decodificare direttamente i codici bidimensionali. Per quanto riguarda il mondo Apple, la programmazione è effettuata in Objective-C e non esistono librerie open-source da poter adoperare direttamente; in ogni caso la Apple non consente l'uso di librerie esterne per la realizzazione delle applicazioni. Infatti ogni applicazione, prima di poter essere pubblicata in Apple Store (per poterla utilizzare sui dispositivi Apple), deve essere controllata e approvata dalla Apple (si veda [32]).

Vediamo adesso alcune caratteristiche del codice bidimensionale DataMatrix:

- I moduli elementari bianchi e neri rappresentano i bit 0 e 1.
- In riferimento alla figura 4.4, in due lati del quadrato, è presente una barra a forma di L, chiamata "Finder Pattern" che funge analogamente agli assi di un sistema cartesiano, infatti lo spazio a ridosso dell'intersezione può essere visto come il punto di origine. Quindi questa barra a L, rappresenta il modello usato per la rivelazione del codice e fornisce al lettore l'informazione sulla direzione di lettura. Negli altri due lati, sono invece presenti altre due barre chiamate Timing Pattern, che presentano dei moduli elementari disposti in modo alterno. Tramite queste due barre è possibile capire il numero di righe e colonne, e la grandezza del modulo elementare.

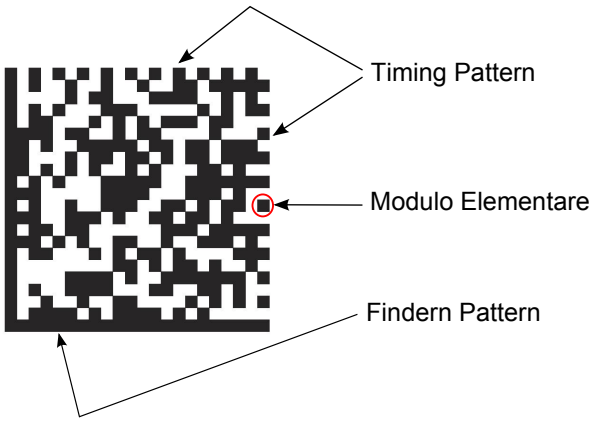


Figura 4.4: Struttura del Data Matrix

- La parte centrale è destinata ai dati.
- Per una corretta decodifica, deve essere presente una fascia perimetrale, di colore bianco, esterna al codice.

Per completare il quadro sull'evoluzione dei codici bidimensionali, è utile parlare di altro tipo di codice con delle caratteristiche diverse rispetto ai codici descritti fin'ora: il Bee Tagg. Questo codice presenta la possibilità di personalizzare il codice con un marchio così da rafforzare il branding. Degli esempi sono in figura 4.5.

Figura 4.5: Esempi di Bee Tagg

Attualmente, la principale sfida, legata ai codici 2D è quella di creare codici con maggiori capacità, mantenendo intatte le caratteristiche di robustezza. Per questi scopi, si stanno sviluppando negli ultimi anni codici che utilizzano moduli elementari di diversi colori. L'esempio più noto di questo tipo di codice è il codice HCCB - High Capacity Color Barcode (si veda [38]). Un esempio di questo codice è mostrato in figura 4.6, in cui i moduli elementari sono rappresentati da triangoli e possono assumere quattro diverse colorazioni: nero, verde, rosso e giallo. La principale problematica che si ha nell'uso di codici colorati sono le condizioni di luminosità dell'ambiente, che determinano una variazione notevole, sul colore percepito nel processo di scansione con conseguenti errori in fase di decodifica. Nel lavoro, Grillo et al. [35], è mostrato un esempio di codice bidimensionale colorato, realizzato a partire da un Qr Code; nei test vengono utilizzati 4 e 16 colori. Il codice è chiamato HCC2D, permette di ottenere delle densità di dati (parametro legato alla quantità di informazione presente in una certa area) simile al codice HCCB con una robustezza analoga a quella dei QR code. Un parametro di interesse, nei codici colorati, è legato ai Bit per modulo (BpM), che nei classici codici a matrice vale 1, mentre nei codici che usano M colori vale $BpM = log_2(M)$.

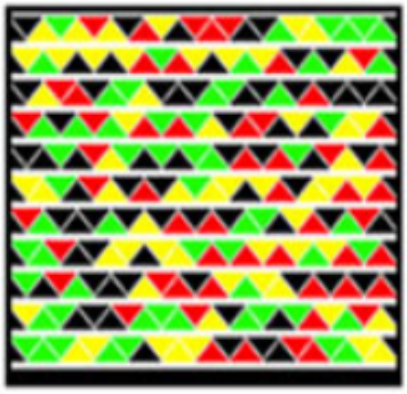


Figura 4.6: High Capacity Color Barcode della Microsoft

4.3 Positional Code

Nel paragrafo precedente si sono analizzati alcuni dei codici bidimensionali presenti sul mercato, così da capirne gli elementi essenziali. In questo paragrafo e nei prossimi, si descriveranno invece dei nuovi aspetti legati al codice bidimensionale che si vuole usare nel progetto SPinV. Questo codice considera un approccio diverso, si parlerà quindi dei diversi elementi che caratterizzano questo nuovo metodo.
I codici bidimensionali sono legati al sistema di localizzazione proposto, proprio perché vengono utilizzati come supporto al processo di autolocalizzazione: l'utente scatta la fotografia al codice, stampato su un cartellone e posizionato in un punto noto all'interno dell'area che si vuole coprire. Il terminale di utente, ovvero lo smartphone, si occupa della decodifica del file .jpg (descritta nel capitolo 5) e della decodifica del codice 2D. Una volta estratta l'informazione dal codice, è possibile legare tale informazione ad una determinata coordinata (x, y) della mappa, sul sdr della navigazione.

Una osservazione di primaria importanza che va fatta, è che i codici bidimensionali prima descritti possono essere definiti dei *codici informativi*: per tali codici l'obiettivo primario, è quello di trasportare la maggior quantità di dati possibile cercando di fornire una buona robustezza in fase di lettura. Per finalità di autolocalizzazione e in merito al codice 2D da adoperare, invece, è possibile fare le seguenti considerazioni:

1. Non è necessario avere un codice che trasporti una gran quantità di dati.

2. Non è necessario un codice di piccole dimensioni.

3. Si vuole un codice leggero, dal punto di vista computazionale, così da garantire una buona velocità in fase di lettura.

4. Si vuole un codice robusto, così da poter essere correttamente decodificato in diverse situazioni operative.

5. Il processo di autolocalizzazione (mediante fotografia al codice 2D) rappresenta l'operazione di rifasamento o calibrazione del sistema inerziale.

6. Il codice e i relativi algoritmi di decodifica dovrebbero fornire delle *informazioni supplementari* così da poter sfruttare il codice bidimensionale per posizionare l'utente, si è detto infatti, nel paragrafo 3.2, che per poter applicare i principi del dead reckoning è necessario capire l'assetto del sdr body rispetto al sdr della navigazione. Questo concetto si traduce, praticamente, in questo modo: la decodifica dei dati contenuti nel codice, porta a identificare le coordinate in cui si trova il cartellone, attaccato alla parete e contenente il codice bidimensionale. Tra i dati del codice deve essere contenuta anche l'informazione relativa all'angolo α, che rappresenta l'angolo in cui si trova il cartellone rispetto all'asse delle x del sdr della mappa (si veda 3.34).

In un *caso teorico*, l'utente scatta la fotografia in modo "ideale", ovvero con l'asse z del sdr body del terminale perpendicolare al cartellone e in maniera tale che la fotografia contenga esattamente il codice (scatto fotografico dalla distanza d_T). In queste condizioni, può essere valutata la posizione del terminale, ovvero può essere valutato l'assetto del sdr body rispetto al sistema di riferimento della mappa.
In un *caso pratico* il terminale, al momento dello scatto fotografico, si trova ad una distanza d dal codice maggiore rispetto al caso teorico descritto prima e in una posizione tale da formare un angolo γ rispetto all'asse perpendicolare al cartellone.
I due casi appena descritti sono mostrati in figura 4.7 in cui:

- α indica l'angolo che si forma fra l'asse perpendicolare al cartellone e l'asse delle x nel sdr della mappa.

- (p_x, p_y) sono le coordinate del cartellone con il codice nel sdr della mappa.

- (a_T, b_T) sono le coordinate identificative del terminale nel caso 'ideale'.

- (a, b) sono le coordinate identificative del terminale in un caso 'non ideale'.

- d_T è la distanza cartellone/terminale nel caso ideale.

- d è la distanza cartellone/terminale in un caso non ideale.

- γ è l'angolo che si forma tra l'asse perpendicolare al cartellone e la "traiettoria" durante lo scatto della fotografia, ovvero rappresenta l'angolo che si forma fra il caso teorico e un caso reale.

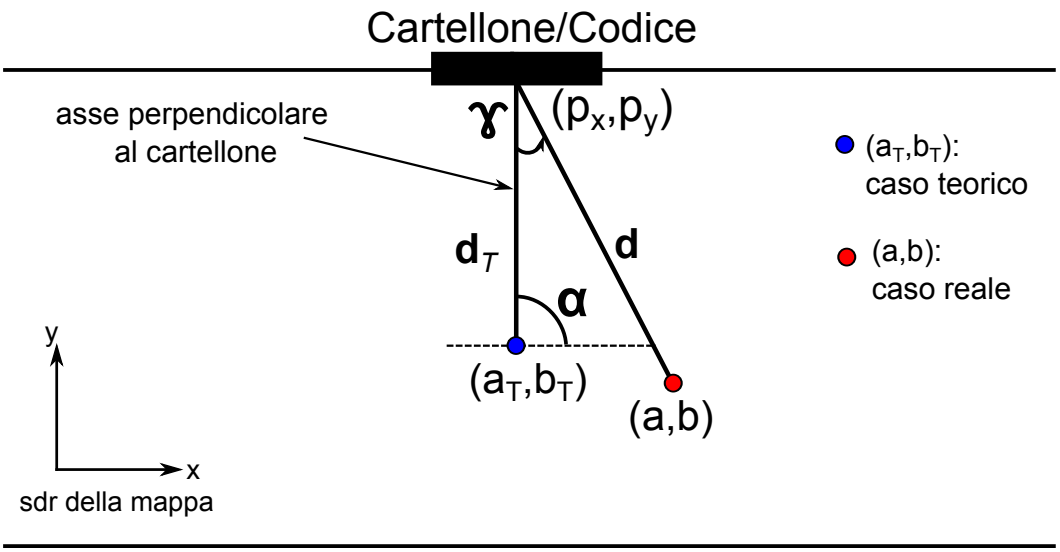


Figura 4.7: Il problema dell'autolocalizzazione

Quindi, al momento in cui viene scattata la foto al codice, gli algoritmi di lettura del codice oltre a fornire l'informazione contenuta dal codice stesso (coordinate cartellone (p_x, p_y) e angolo α), devono essere in grado di estrapolare i valori legati alla distanza (d) e all'angolazione (γ) da cui è stata scattata la fotografia. Nasce quindi il concetto di **Positional Code**. A partire da queste informazioni, sarà possibile ricondursi alle coordinate da cui è avvenuto lo scatto fotografico, mediante la seguente:

$$\begin{cases} a = p_x - d\,cos(\alpha + \gamma) \\ b = p_y - d\,sin(\alpha + \gamma) \end{cases} \tag{4.1}$$

Le procedure e gli algoritmi di lettura/decodifica dei codici bidimensionali presenti sul mercato, non prevedono la presenza di "informazioni posizionali", ecco perché si è pensato di introdurre un nuovo codice pensato e strutturato così da utilizzare dei principi di decodifica e di analisi diversi da quelli standard. Quindi con lo SPinV Code, si propone un **approccio diverso alla realizzazione del codice bidimensionale**. Nei prossimi paragrafi verranno descritti gli elementi che stanno alla base dello SPinV Code.

4.4 Nozioni di Geometria Proiettiva

In questa sezione verranno introdotti alcuni aspetti geometrici legati al mondo della geometria proiettiva. La geometria proiettiva è quella parte della geometria che modellizza i concetti intuitivi di prospettiva e orizzonte. Nella geometria piana proiettiva (due dimensioni) due rette si intersecano sempre, ovvero non sono mai parallele. Il caso "particolare" delle rette parallele viene eliminato aggiungendo al piano i cosiddetti punti all'infinito o punti impropri. Per cui due rette parallele hanno comunque in comune un punto all'infinito. Il riferimento è Fusiello [39].

4.4.1 Coordinate Omogenee

In geometria proiettiva si utilizza il concetto di *punto all'infinito*, che è un punto ideale che si aggiunge all'usuale piano euclideo e che può essere definito utilizzando le coordinate omogenee.
Considero una coppia di assi di riferimento; tutti i punti in $\mathbb{R}^2$ ammettono un unica rappresentazione come coppia di numeri reali (x, y). Considerando due rette distinte di equazioni $ax + by + c = 0$ e $a'x + b'y + c' = 0$, queste si incontrano, se non sono parallele, nel punto (x, y) di $\mathbb{R}^2$. Vale la regola di Cramer, espressa dalla 4.2, per il calcolo del punto di intersezione (x, y):

$$x = -\frac{\det\left(\begin{bmatrix} c & b \\ c' & b' \end{bmatrix}\right)}{\det\left(\begin{bmatrix} a & b \\ a' & b' \end{bmatrix}\right)} \qquad y = -\frac{\det\left(\begin{bmatrix} a & c \\ a' & c' \end{bmatrix}\right)}{\det\left(\begin{bmatrix} a & b \\ a' & b' \end{bmatrix}\right)} \tag{4.2}$$

Dalla regola di Cramer si vede infatti, che nella situazione di parallelismo, il denominatore è nullo e quindi la coppia (x, y) tende all'infinito. Questa osservazione concorda con il fatto che due rette parallele si incontrano all'infinito, o più rigorosamente che si incontrano in un punto improprio o ideale (punto all'infinito). Inoltre, quando le rette sono distinte, delle tre quantità che compaiono nella 4.2 almeno una deve essere diversa da 0. In geometria proiettiva, si utilizzano le tre quantità (a, b, c) per rappresentare dei punti in $\mathbb{P}^2$. In pratica, al posto di rappresentare i punti del piano tramite le coordinate (x, y), si utilizza la terna di valori (a, b, c), chiamata coordinate omogenee, che sono collegate alle coordinate di $\mathbb{R}^2$ mediante la relazioni:

$$(x, y) \to (a, b, c) \qquad x = \frac{a}{c} \qquad y = \frac{b}{c} \tag{4.3}$$

É importante sottolineare che due terne (a, b, c) e (ka, kb, kc) con $k \neq 0$, sono consi-

derate in geometria proiettiva equivalenti, proprio perché si considerano i rapporti tra gli elementi. Sostituendo la 4.3, alle coordinate (a, b, c), si ottiene $(cx, cy, c) = (x, y, 1)$. Inoltre, quando $c \neq 0$, la terna rappresenta un punto effettivo (punto al finito), mentre ogni terna per cui $c = 0$, rappresenta un punto all'infinito. Il grosso vantaggio delle coordinate omogenee è che tutte le trasformazioni diventano moltiplicazioni. Ad esempio una traslazione in $\mathbb{R}^2$: $(x, y) \rightarrow (x + a, y + b)$ può essere scritta in questo modo:

$$\begin{pmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + a \\ y + b \\ 1 \end{pmatrix} \tag{4.4}$$

La possibilità di realizzare tutte le trasformazioni mediante la semplice moltiplicazione per una matrice, cioè mediante un unico genere di meccanismo facilmente implementabile, semplifica notevolmente le attività computazionali e di ingegnerizzazione del software per la computer grafica. Infatti le coordinate omogenee sono ampiamente adoperate nella rappresentazione delle scene 3D e le notazioni matriciali sono impiegate nella maggior parte delle librerie di programmi grafici 3D come OpenGL e Direct3D (si veda [40]).

4.4.2 Omografia

Un omografia o trasformazione proiettiva $f : \mathbb{P}^n \mapsto \mathbb{P}^n$ é un applicazione lineare in coordinate omogenee :

$$f \ : \ x \mapsto Hx \tag{4.5}$$

Dove H è una matrice di dimensione (n+1)x(n+1) non singolare.
In pratica si tratta di una mappatura di punti e linee da $\mathbb{P}^n$ a $\mathbb{P}^n$, ovvero di punti e linee appartenenti ad un piano a punti e linee appartenenti ad un altro piano. Nel caso n=2, che è il caso di interesse, si ha quindi che la matrice H è una matrice $3x3$. Utilizzando le coordinate omogenee per il punto, introdotte nella sezione precedente, risulta:

$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{33} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{4.6}$$

Un esempio di omografia è mostrato in figura 4.8 in cui si ha la mappatura fra punti di due piani.

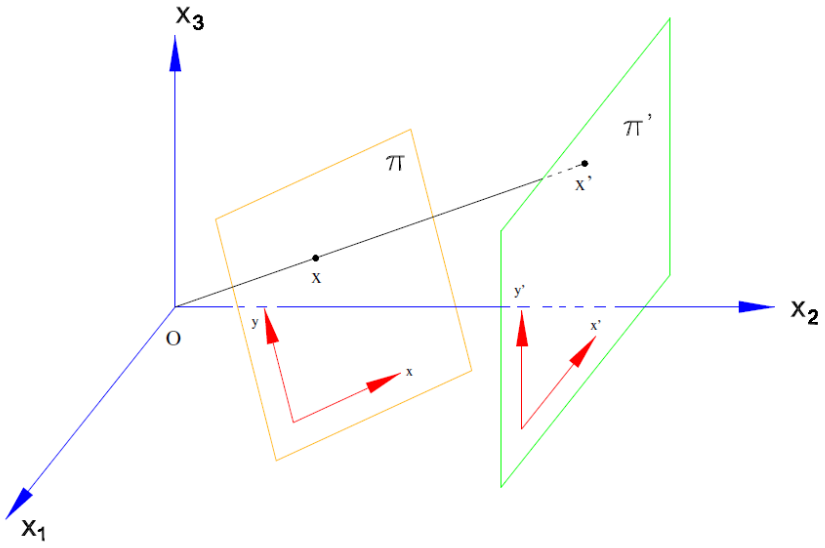


Figura 4.8: La proiezione da un punto dello spazio mappa punti e linee del piano π in punti e linee del piano π'

In pratica, conoscendo la matrice H che sintetizza il legame tra i due piani, sarà possibile mappare ogni punto del piano π in un punto del piano π'.
Esistono diversi tipi di omografia (affinità, similarità, euclidea ...), in cui ogni caso è caratterizzato da opportune matrici H. Noi siamo interessati al caso più generico di omografia, chiamato appunto proiettività, che racchiude tutti i casi particolari, in cui la matrice H è una matrice 3x3 con 8 gradi di libertà. I gradi di libertà sono 8 e non 9 perché si considerano i rapporti indipendenti fra gli elementi della matrice. Infatti, vale un discorso analogo a quello fatto per il punto in cui ci sono due gradi di libertà, anche se ci sono tre valori che lo rappresentano.

Per comprendere meglio questa trasformazione, considero la figura 4.9, in cui sono mostrate due applicazioni pratiche del concetto di omografia.

Proiettando una luce, su una superficie rettangolare, l'ombra prodotta sarà un

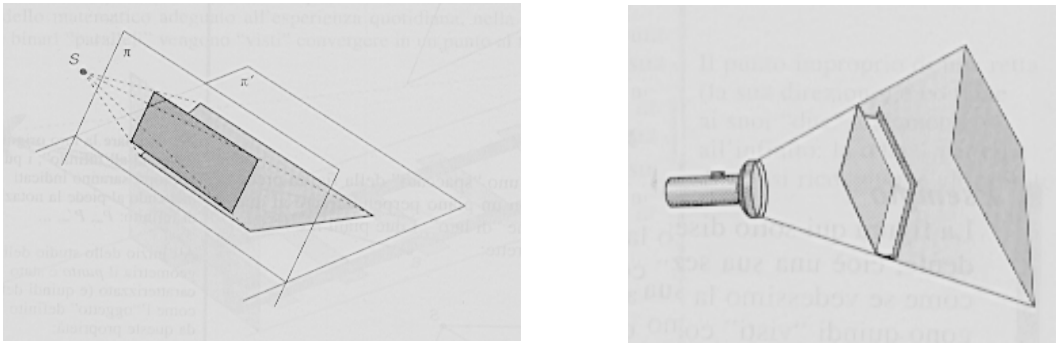


Figura 4.9: Esempi pratici di omografia

quadrilatero, la cui struttura dipenderà dalla posizione e dalla direzione della luce. Si avrà quindi una distorsione del rettangolo originario (si parla spesso di *deformazione prospettica*): gli angoli, non sono più retti e viene perso il parallelismo tra le due coppie di coppie di lati; la grandezza di questo quadrilatero dipenderà invece dalla distanza da cui è stata scattata la fotografia.
In riferimento alla figura 4.9, la torcia (o il punto P), rappresenta la fotocamera dello smartphone, il libro (o il piano rettangolare) rappresenta il codice e l'ombra distorta (o il quadrilatero distorto), rappresenta la fotografia scattata. Nel nostro caso, le informazioni note a priori, sono la fotografia del codice e la conoscenza delle dimensioni reali del codice stesso. L'incognita è rappresentata dalla posizione dell'orientamento della fotocamera.

4.5 Modelli di fotocamera

Nel paragrafo precedente si sono analizzati gli aspetti di geometria proiettiva da un punto di vista puramente teorico. Nel mondo "reale" la fotografia scattata, oltre a dipendere da parametri geometrici, dipenderà anche dalle caratteristiche della fotocamera, quindi, per una caratterizzazione completa del problema bisogna considerare i modelli geometrici della fotocamera. Il riferimento è Fusiello [39].

La fotocamera realizza una mappatura fra lo spazio 3D e un immagine in 2D. Una precisa caratterizzazione di questa mappatura, consente di sfruttare l'informazione visiva acquisita. Collegandosi al concetti di sensori inerziali discussi in questo lavoro di tesi, *la fotocamera può essere vista come un sensore aggiuntivo*: come in tutti i sensori, anche nella fotocamera, occorre definire una certa funzione che lega l'input (scena in 3D), all'output (fotografia in 2D). Si descriveranno due modelli; nel paragrafo 4.5.1, si descriverà un modello semplificato, che tiene conto solo di alcuni fattori, mentre nel 4.5.2, si considererà un caso più completo.

Prima di iniziare la trattazione dei modelli di fotocamera, è utile fare una precisazione. Considero per adesso, che il codice bidimensionale sia costituito da un semplice rettangolo (rettangolo nero su sfondo bianco). Come detto in precedenza, in base al posizionamento dello smartphone rispetto al codice, nel momento in cui viene scattata la fotografia, si avrà nell'immagine fotografica un quadrilatero distorto. Può in realtà succedere che anche se la fotografia fosse scattata in condizioni ideali (fotocamera "in bolla" e con l'asse z del sdr body perpendicolare al cartellone/codice) non si abbia un rettangolo che riproduca fedelmente quello reale. Ciò è dovuto ad un altro tipo di deformazione chiamata *deformazione ottica*. La deformazione ottica deriva dall'aberrazione sferica dell'obiettivo e dal modo e dalla precisione con cui l'obiettivo viene costruito; l'effetto è quello di trasformare delle linee rette in curve e normalmente questo tipo di deformazione è espressa a livello

percentuale (facendo riferimento alla linea reale e alla relativa curva fotografata). Questo tipo di deformazione è spesso indicata come deformazione cuscinetto/barilotto (si veda la figura 4.10). Negli obiettivi a focale variabile (obiettivi con zoom), le deformazioni ottiche sono ancora più evidenti e variano con la focale, in particolare crescono al crescere della focale. Nella trattazione che segue non si tiene conto di questo problematica, dati gli effetti trascurabili che produce per il problema in esame.

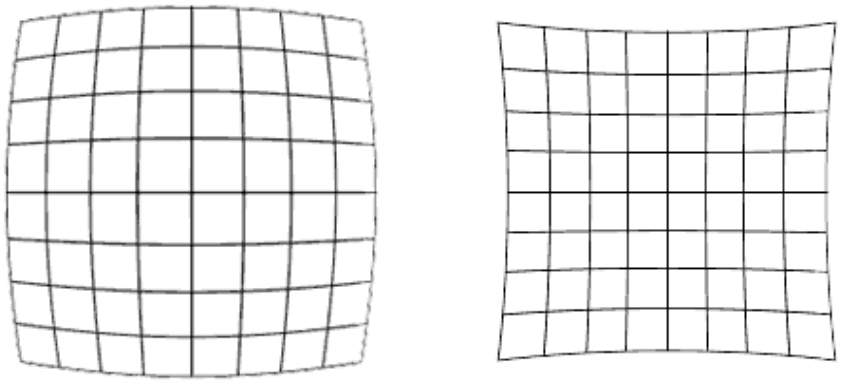


Figura 4.10: Distorsione a barilotto e distorsione a cuscinetto

In definitiva, applicando i principi dell'omografia è possibile eliminare la deformazione prospettica ottenendo una versione "raddrizzata" del rettangolo fotografato (in realtà l'interesse ricade solo su alcuni punti), e attraverso questo processo sarà possibile valutare la distanza d da cui è stata scattata la fotografia. Inoltre mediante l'analisi prospettica mediante omografia, è possibile attuare una corretta lettura/decodifica del codice posizionale, incrementando la robustezza del codice .

4.5.1 Modello semplificato

Prima di iniziare la trattazione è necessario introdurre i sistemi di riferimento usati. Nella terminologia, si parla di *sistema di riferimento mondo*, che è il sistema di riferimento utilizzato per descrivere la posizione di un determinato elemento nello spazio. Tale sistema di riferimento, coincide con il sistema di riferimento della navigazione (sdr della mappa) utilizzato nel capitolo 3, che è un sistema in due dimensioni. Altro riferimento è quello della fotocamera, chiamato *sistema di riferimento della fotocamera*. In questo sdr, l'origine degli assi è fissata sul centro ottico della fotocamera. Questo sistema, non coincide esattamente con il sdr body introdotto nel capitolo 3, ma per i nostri scopi, i due sdr possono considerarsi equi-

valenti: gli assi sono i medesimi (si veda la figura 3.4), ma l'origine degli assi nel sdr della fotocamera, risulta pochi centimetri più in "alto" rispetto all'origine degli assi del sdr body. In pratica si ha una semplice traslazione lungo l'asse y positivo. Inoltre è presente un *sistema di riferimento per la fotografia*, che è quindi un sdr bidimensionale.

Faccio riferimento alla figura 4.11 (estratta da Fusiello [39]). Nel modello semplice, indicato in letteratura anche come modello Basic Pinhole, si ha che l'immagine di un punto M dello spazio, si forma nella relativa fotografia (punto m nel piano $\mathcal{R}$), grazie ad un unico raggio luminoso che attraversa un piccolo foro (C); f è la distanza focale, parametro caratteristico di ogni fotocamera, che rappresenta la distanza tra il centro ottico dell'obiettivo e il piano del sensore CCD della fotocamera. Ovviamente una fotocamera è costruita in modo molto diverso dallo schema di principio descritto, ma questo modello rappresenta in alcuni casi, una buona approssimazione del funzionamento reale. Nel modello semplificato, si considera che il sdr del mondo e il sdr della fotocamera coincidono, per cui un punto nello spazio 3D è espresso rispetto al sdr della fotocamera. In riferimento alla figura 4.11, si ha che l'origine degli assi di tale riferimento è al centro del piano $\mathcal{F}$ ed è espresso dagli assi X, Y e Z. Il sistema di riferimento della fotografia è descritto dagli assi (u, v) e l'origine di tali assi è nel piano $\mathcal{R}$. Un punto M nello spazio è descritto dalla terna cartesiana (x,y,z), mentre il punto m è nel piano della fotografia è rappresentato dalle coordinate (u,v).

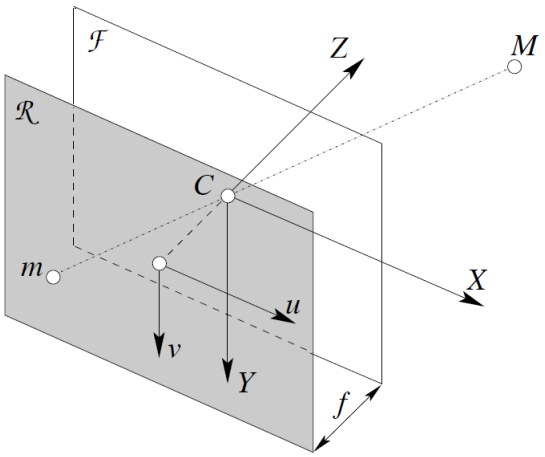


Figura 4.11: Modello geometrico semplice della fotocamera

Con un ragionamento geometrico, osservando il modello frontalmente al piano formato dagli assi X,Z, (si veda figura 4.12 estratta da Fusiello [39]) si ha che i triangoli le cui ipotenuse sono CM e Cm, sono triangoli simili, per cui è possibile scrivere:

$$\frac{f}{z} = \frac{-u}{x} = \frac{-v}{y} \quad \rightarrow \quad \begin{cases} u = \frac{-f}{z}x \\ v = \frac{-f}{z}y \end{cases} \tag{4.7}$$

Bisogna però considerare che gli assi x e y del sdr body dell'iPhone, rappresentati in figura 3.4, presentano delle orientazioni invertite rispetto agli assi X Y del sdr della fotocamera=sdr del mondo. Per cui l'equazione 4.7, può essere riscritta in relazione al sdr body, come segue:

$$\frac{-f}{z} = \frac{-u}{x} = \frac{v}{y} \quad \rightarrow \quad \begin{cases} u = \frac{f}{z}x \\ v = \frac{-f}{z}y \end{cases} \tag{4.8}$$

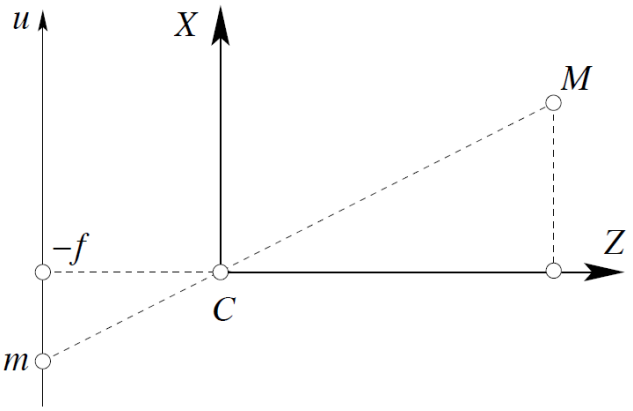


Figura 4.12: Vista lungo il piano X Z del modello geometrico semplice della fotocamera

Le equazioni 4.7 e 4.8, esprimono il meccanismo di proiezione prospettica (equazioni di proiezione prospettica), ovvero le trasformazioni dalle coordinate 3D (punto M) alle coordinate 2D (punto m). A causa della divisione per z, queste relazioni non sono lineari rispetto alle coordinate 3D del punto. Per tale motivo risulta conveniente utilizzare le coordinate omogenee: i punti m ed M in coordinate omogenee valgono:

$$\mathbf{m} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \qquad \mathbf{M} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.9}$$

Ponendo la terza e la quarta coordinata di m e M pari a 1, si è escluso il caso di punti all'infinito. Riferendosi alla 4.7, ovvero al caso di figura 4.11, possiamo scrivere in definitiva:

$$z\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} -fx \\ -fy \\ z \end{bmatrix} = \underbrace{\begin{bmatrix} -f & 0 & 0 & 0 \\ 0 & -f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\mathbf{P}} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.10}$$

In forma matriciale:

$$z\mathbf{m} = P\mathbf{M} \tag{4.11}$$

In cui P è chiamata *matrice di proiezione prospettica* (MPP).
La MPP vale quindi:

$$P = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & -f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.12}$$

La 4.11 può anche essere scritta a meno del fattore di scala z; in tal caso si usa scrivere:

$$\mathbf{m} \backsimeq P\mathbf{M} \tag{4.13}$$

Per cui l'unico parametro che caratterizza il modello di mappatura è la distanza focale. É importante notare che la distanza focale varia al variare dello zoom, ma all'interno di un applicazione è possibile impedire lo zoom, consentendo quindi di utilizzare un unico parametro.

L'aspetto di interesse è che se si conosce la MPP, è possibile risalire agli angoli reali che i raggi proiettivi formano tra di loro: è possibile cioè misurare le direzioni reali degli oggetti visualizzati nell'immagine rispetto alla direzione dell'asse focale, si ha infatti: $d = P^{-1}\,\mathbf{m}$, dove d è un vettore nella direzione del punto M (si veda [41]).

4.5.2 Modello generale

Il modello semplificato considera solo la trasformazione prospettica e non è sufficiente a riprodurre con fedeltà la mappatura operata nelle fotocamere digitali reali; inoltre sono state fatte delle semplificazioni sul sistema di riferimento (unico) e non si è considerato l'effetto della pixelizzazione, che rende il piano R di figura 4.11 un piano a valori discreti e non reali. Nel modello generale, si inseriscono altri parametri che caratterizzano il sensore CCD, il modello in questione è indicato in letteratura come modello CCD Full Perspective. Il problema viene esteso, considerando dei parametri intrinseci, che tengono conto di una rappresentazione più complessa del sensore CCD e dei parametri estrinseci che considerano, che i sdr fotocamera e mondo non coincidono. In figura 4.13 sono rappresentati i tre sistemi di riferimento nel modello generale (figura estratta da Fusiello [39]).

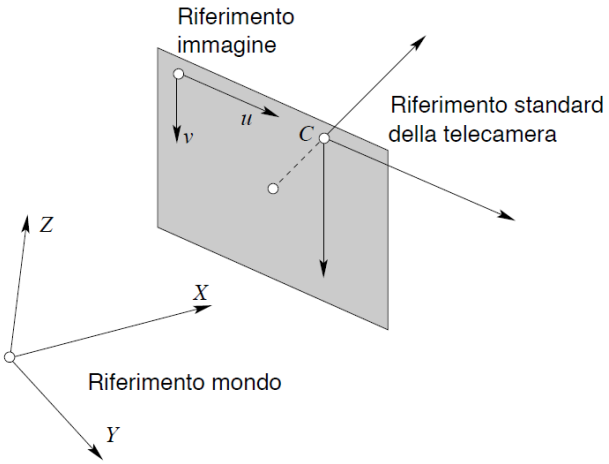


Figura 4.13: Sistemi di riferimento nel modello generale

Per quanto riguarda i parametri intrinseci, oltre a considerare la distanza focale f si considera la *traslazione del centro ottico* di una quantità (u_o, v_0) rispetto al caso precedente. Ovvero, nel caso precedente l'origine del sistema immagine coincideva con il punto principale, mentre in questo caso, l'origine coincide con un altro punto: di solito si considera il punto in alto a sinistra; si parla in questo caso di top-left coordinate system. Altri parametri intrinseci che si considerano sono legati alla scalatura indipendente degli assi (u,v) rispettivamente mediante i fattori k_u e k_v, per cui questi fattori rappresentano i pixel per unità di lunghezza rispetto agli assi u e v . L'unità di misura è infatti pixel/m.

Risulterà quindi che un punto M nello spazio, viene mappato nel riferimento immagine nelle coordinate:

$$\begin{cases} u = k_u \frac{-f}{z} x + u_0 \\ v = k_v \frac{-f}{z} y + v_0 \end{cases} \tag{4.14}$$

La matrice di proiezione prospettica che definisce la mappatura, considerando i soli parametri intrinseci, vale:

$$P = K\,[I|0] = \begin{bmatrix} -fK_u & 0 & u_0 & 0 \\ 0 & -fK_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.15}$$

in cui K vale:

$$K = \begin{bmatrix} -fK_u & 0 & u_0 \\ 0 & -fK_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.16}$$

Per cui possiamo scrivere, ancora:

$$\mathbf{m} \backsimeq P\mathbf{M} = K[I|0]\,\mathbf{M} \tag{4.17}$$

Passiamo ora ai parametri estrinseci, che tengono conto che i sdr mondo e fotocamera non coincidono; infatti nell'analisi fatta fino ad ora si è considerato che il punto M continua ad essere espresso sul riferimento della fotocamera. Per tenere conto di questo è necessario introdurre una trasformazione che lega i due sistemi di riferimento. In particolare, la trasformazione è una roto-traslazione, ovvero una rotazione R seguita da una traslazione t. Indico adesso con M_c le coordinate omogenee di un punto nel sistema di riferimento della fotocamera e con M le coordinate dello stesso punto espresse nel sistema di riferimento mondo. La rototraslazione tra i due sdr può essere espressa come:

$$\mathbf{M_c} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \mathbf{M} = G\mathbf{M} \tag{4.18}$$

La 4.17, può quindi essere riscritta, considerando che adesso il punto M è chiamato M_c, ovvero tenendo conto dei parametri estrinseci.

$$\mathbf{m} \backsimeq P\mathbf{M_c} = K[I|0]\,\mathbf{M_c} \;\rightarrow\; \mathbf{m} \backsimeq K[I|0]\,G\,\mathbf{M} \tag{4.19}$$

In definitiva la matrice di proiezione prospettica che definisce la mappatura vale:

$$P = K\,[I|0]\,G \tag{4.20}$$

K è una matrice 3x3 che definisce i parametri intrinseci caratteristici della fotocamera, $[I|0]$ rappresenta la trasformazione prospettica in coordinate normalizzate e G è una matrice 2x2 che codifica i parametri estrinseci. In letteratura, la matrice di proiezione prospettica è spesso scritta come:

$$P = K\,[R|t] \tag{4.21}$$

La 4.21 è ottenuta operando una fattorizzazione così da esplicitare i parametri legati alla rotazione e alla traslazione:

$$t = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \qquad R = \begin{bmatrix} r_1^T \\ r_2^T \\ r_3^T \end{bmatrix} \tag{4.22}$$

Riassumendo, attraverso la matrice di proiezione prospettica P di dimensione 3x4, è possibile mappare un punto dello spazio di coordinate (x, y, z) in un punto del piano immagine (u, v). Utilizzando le coordinate omogenee, risulta:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = P \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.23}$$

Facciamo adesso un piccolo passo indietro per cercare di mettere in relazione i concetti espressi in questo paragrafo e nei paragrafi precedenti; In precedenza si è detto che la fotocamera realizza una mappatura fra lo spazio 3D e l'immagine fotografica in due dimensioni. In realtà nel nostro caso, siamo interessati ad una mappatura fra due piani, ovvero al mapping fra il piano della fotografia e il piano in cui è presente

il codice (cartellone). Il problema è quindi quello mostrato in figura 4.8 e la relazione rappresentativa del mapping fra i due piani è espressa nell'equazione 4.6. La matrice che mette in relazione i due piani è quindi la matrice H di dimensione 3x3. In sostanza H rappresenta la matrice di proiezione prospettica di interesse, ovvero la matrice P che, nel caso particolare detto, diventa una matrice 3x3.
Continuo a considerare che il mio codice posizionale, sia costituito da un semplice rettangolo nero su sfondo bianco. Sempre in relazione alla figura 4.8, si può considerare che il codice (ovvero il rettangolo) sia nel piano π e la relativa fotografia sia nel piano π'. Suppongo inoltre che gli algoritmi di decodifica/lettura siano in grado di ricavare esattamente i 4 vertici del quadrilatero (distorto) nella fotografia. A partire dalla conoscenza del rettangolo reale e dalla conoscenza del quadrilatero nella fotografia, il problema è di ricavare la matrice di proiezione prospettica, ovvero la matrice dell'omografia. Questa procedura è indicata in letteratura come *calibrazione* e consiste appunto, nel misurare i parametri intrinseci ed estrinseci del modello della fotocamera. L'idea è che, conoscendo le proiezioni di punti 3D di coordinate note, sia possibile ottenere i parametri incogniti risolvendo le equazioni di proiezione prospettica. Esistono in letteratura diversi metodi per la stima della MPP, come ad esempio il metodo DLT o dei metodi non-lineari. In seguito alla stima della MPP è possibile ricavare, utilizzando la fattorizzazione QR, i parametri estrinseci ed intrinseci.
Il problema, analizzato da un punto di vista generale e teorico, è il seguente:
sono note le coordinate dei quattro vertici del rettangolo "reale", che esprimo in coordinate omogenee:

$$(x_i, y_i) \rightarrow (u_i, v_i, w_i) \qquad x_i = \frac{u_i}{w_i} \qquad y_i = \frac{v_i}{w_i} \qquad i = 1, 2, 3, 4 \tag{4.24}$$

Analogamente per i quattro vertici del quadrilatero nella fotografia:

$$(x'_i, y'_i) \rightarrow (u'_i, v'_i, w'_i) \qquad x'_i = \frac{u'_i}{w'_i} \qquad y'_i = \frac{v'_i}{w'_i} \qquad i = 1, 2, 3, 4 \tag{4.25}$$

Risolvendo la forma matriciale espressa nella 4.6, risulta:

$$\begin{cases} u'_i = h_{11}u_i + h_{12}v_i + h_{13}w_i \\ v'_i = h_{21}u_i + h_{22}v_i + h_{23}w_i \\ w'_i = h_{31}u_i + h_{32}v_i + h_{33}w_i \end{cases} \tag{4.26}$$

Per cui, tenendo conto di quanto scritto nella 4.25, si avrà:

$$x'_i = \frac{h_{11}u_i + h_{12}v_i + h_{13}w_i}{h_{31}u_i + h_{32}v_i + h_{33}w_i} \qquad y'_i = \frac{h_{21}u_i + h_{22}v_i + h_{23}w_i}{h_{31}u_i + h_{32}v_i + h_{33}w_i} \qquad i = 1, 2, 3, 4 \quad (4.27)$$

Si avranno in totale otto equazioni e sarà quindi possibile valutare la matrice H (matrice con 8 gradi di libertà). Se si assume $w_i = 1$, escludendo quindi il caso di punti all'infinito, si avrà che le coordinate omogenee relative ai quattro vertici del rettangolo "reale" sono $(x_i, y_i, 1)$ con $i = 1, 2, 3, 4$. Analogamente per i quattro vertici del quadrilatero nella fotografia è possibile scrivere: $(x'_i, y'_i, 1)$ con $i = 1, 2, 3, 4$. La 4.27 può essere riscritta:

$$x'_i = \frac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + h_{33}} \qquad y'_i = \frac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + h_{33}} \qquad i = 1, 2, 3, 4 \quad (4.28)$$

Attraverso la fattorizzazione QR è possibile risalire alla matrice K contenente i parametri intrinseci e le matrici R e t che caratterizzano la roto-traslazione del piano e quindi è possibile capire l'orientamento della fotocamera rispetto al codice posizionale nel momento in cui è stata scattata la fotografia.

Per rendere coerenti le unità di misura tra il "mondo reale" e il piano della fotografia, si può considerare un legame lineare, che associa ad una fissata distanza in pixel tra due punti noti, una misura in metri. Considero ad esempio che il rettangolo rappresentativo del codice abbia il lato maggiore lungo a metri e il lato minore di b metri; fotografando questo rettangolo da una posizione ideale, risulterà che il rettangolo sarà costituito, nel lato maggiore e nel lato minore rispettivamente da c pixel e da d pixel. Se la fotografia è davvero scattata in condizioni ideali, il rapporto fra i due lati in entrambi i casi è lo stesso, ovvero $a/b = c/d$. Inoltre, ogni pixel dell'immagine assume un determinato significato in metri, ovvero ogni pixel rappresenta $a/c = b/d$ metri nel "mondo reale". In presenza di deformazione prospettica, derivante da una fotografia scattata in condizioni non ideali, la considerazione fatta non è più vera.

Altro aspetto di notevole interesse, è la possibilità *eliminare la deformazione prospettica*, ovvero è possibile attuare il "raddrizzamento fotografico" risalendo alla versione fotografica come se lo scatto fosse avvenuto da posizione ottimale (si veda [42]). Questo processo è quello che viene attuato normalmente nel mondo fotografico per correggere le deformazioni prospettiche generate da scatti fotografici non eseguiti perfettamente "in bolla". Sono molti i programmi che consentono di attuare questa

correzione (ad esempio PTGui, PTLens, Adobe Photoshop), ma il funzionamento di questi tool è sostanzialmente lo stesso. Si devono indicare almeno due rette verticali e due orizzontali, ovvero delle rette che nella realtà sono disposte verticalmente e orizzontalmente, ma che nella fotografia non risultano verticali e orizzontali a causa della deformazione prospettica. Solitamente si indicano proprio delle coppie di punti per ognuna delle rette, come mostrato in figura 4.14.

Figura 4.14: Punti di riferimento e relative rette da inserire per attuare la correzione prospettica mediante programma

Inoltre il programma, per attuare la correzione, necessita dei dati relativi alla fotocamera utilizzata e alla fotografia (come la lunghezza focale, valore della distorsione barilotto/cuscinetto, parametri della lente). Solitamente alcuni di questi dati sono incapsulati nel file e possono essere estratti direttamente, come verrà meglio chiarito nel prossimo Capitolo. Altri parametri invece sono caratteristici della fotocamera usata. Con alcuni software (ad esempio PTGui) è possibile regolare singolarmente quali sono i parametri che devono essere ottimizzati; ad esempio se durante lo scatto, la fotocamera è ruotata su tutti gli assi, si può scegliere di ottimizzare i parametri Yaw,Pitch e Roll. Si può anche limitare l'ottimizzazione ad un unico parametro, ad esempio nel caso in cui durante lo scatto fotografico la fotocamera risulta ruotata solo in una delle direzioni. In quest'ultimo caso, il parametro che non viene ottimizzato viene posto a 0. Il risultato della correzione è mostrato in figura 4.15.

Rispetto al caso generale appena descritto, nel caso di interesse sono note anche le dimensioni "reali" del codice posizionale. Si vuole sfruttare questa informazione per il calcolo della distanza. In riferimento alla figura 4.16, indico con a l'altezza "reale" del codice posizionale, con c l'altezza valutata dalla fotografia, con d la distanza e

Figura 4.15: Correzione prospettica dell'immagine di figura 4.14 mediante PTGui

con f la distanza focale. In figura 4.16 è schematizzata la procedura per il calcolo della distanza utilizzando il modello semplice (modello Basic Pinhole).

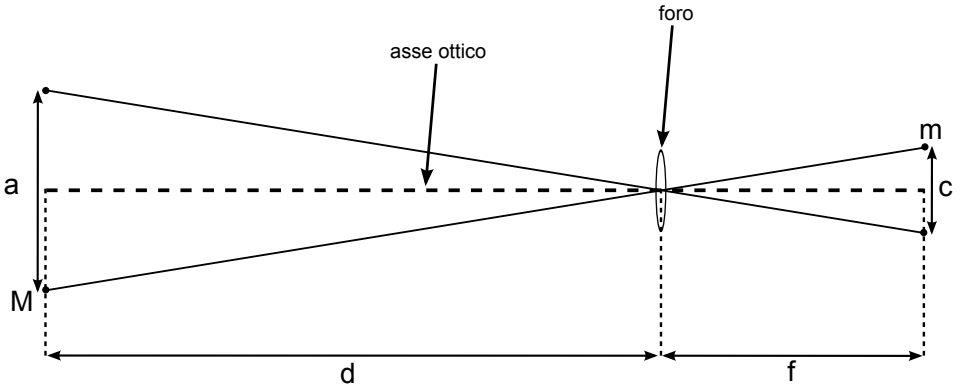


Figura 4.16: Modello semplice: calcolo della distanza

Per completare lo studio è utile aggiungere, che nella trattazione fatta, si è analizzato il problema considerando soltanto la fotografia del codice (rettangolo) e il codice reale. In realtà però si potrebbe integrare l'analisi utilizzando anche le informazioni provenienti dai sensori inerziali per capire quella che è l'inclinazione del terminale nel momento dello scatto fotografico; ad esempio attraverso l'analisi dei dati provenienti dall'accelerometro è possibile valutare l'inclinazione dello smartphone rispetto ad un piano tangente alla superficie terrestre. A tal proposito, è presente nell'Apple Store, l'applicazione EasyMeasure che sfrutta le informazioni provenienti dall'accelerometro e calcola l'inclinazione dell'iPhone con l'obiettivo di valutare le dimensioni degli oggetti inquadrati (distanza, altezza e larghezza). Il funzionamento risulta più chiaro osservando la figura 4.17.

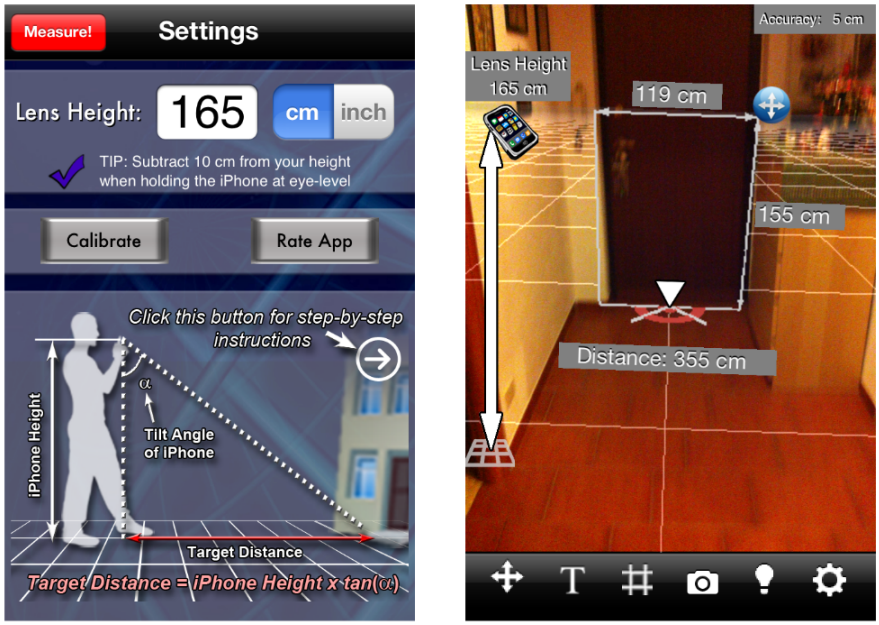


Figura 4.17: Easymeasure: Calcolo della distanza, dell'altezza e della larghezza sfruttando l'angolo α valutato mediante l'accelerometro

L'unico parametro da impostare è l'altezza da terra dell'iPhone, per la precisione dell'obiettivo della fotocamera dell'iPhone (impostato nell'esempio a 165 cm). Il calcolo dell'angolo è abbastanza accurato e ciò permette delle stime abbastanza precise; in particolare nel caso proposta in figura 4.17, si "promette" un accuratezza sulle misure di 5 cm. Le dimensioni del riquadro possono essere variate così da adattare queste dimensioni all'oggetto che si vuole misurare.

4.6 Conclusioni e SPinV Code

Nella prima parte di questo capitolo è stata fatta una panoramica sui codici bidimensionali presenti sul mercato così da capirne il funzionamento sommario. Da questa analisi, e in relazione al codice che si vuole proporre, è possibile fare le seguenti considerazioni:

- I codici bidimensionali utilizzano dei moduli elementari bianchi e neri che rappresentano i bit 0 e 1.

- Al fine di identificare l'area in cui è presente il codice, all'interno dell'immagine acquisita, si utilizzano dei modelli di rivelazione della posizione che sfruttano dei pattern predefiniti. Questi modelli sono anche utilizzati per capire qual'è la direzione di lettura e la dimensione in pixel del modulo elementare.

- Al fine di riallineare il codice, ovvero per il risolvere il problema della distorsione prospettica, si utilizzano degli opportuni modelli per l'allineamento.

- Per una corretta decodifica/lettura del codice bidimensionale, deve essere presente una fascia perimetrale di colore bianco intorno al codice di una certa dimensione.

In seguito, nel paragrafo 4.3, si sono descritte le caratteristiche che dovrebbe avere il codice e una soluzione al problema dell'autolocalizzazione, introducendo il concetto di codice posizionale.
Infine, nei paragrafi 4.4 e 4.5 sono stati descritti e analizzati degli aspetti di geometria proiettiva (omografia) e dei modelli di fotocamera e nel sottoparagrafo 4.5.2 si sono "uniti" questi concetti per ricondursi al problema dell'autolocalizzazione mediante fotografia. La soluzione proposta è parziale, ovvero sono state stabilite solo le "linee guida" senza entrare nel dettaglio o quantomeno senza fornire una soluzione completa. In ogni caso, da questa analisi, è possibile fare le seguenti considerazioni, sempre in relazione al codice che si vuole proporre:

- Il codice posizionale deve avere una struttura tale da permettere l'individuazione di almeno quattro punti, disposti così da individuare rette verticali e orizzontali. Così facendo sarà possibile applicare i principi dell'omografia ed estrarre delle informazioni posizionali dalla fotografia.

- Il codice posizionale deve contenere sia l'informazione legata alla coordinata del codice/cartellone, che l'informazione legata all'angolo per identificare l'angolazione del cartellone rispetto al sistema di riferimento della navigazione.

In base alle considerazioni fatte e agli obiettivi si propone un possibile codice posizionale, chiamato SPinV Code, mostrato in figura 4.18.

La barra rettangolare presente nella parte destra è utilizzata per identificare l'area in cui è presente il codice, per cui questa barra funge da modello di rivelazione della posizione ed è inserita così da sfruttare i principi di omografia. Nella parte sinistra, sono presenti i moduli elementari di colore bianco e nero, che rappresentano rispettivamente i bit 0 e 1. In questa prima versione, è possibile codificare al massimo 10 bit. Tuttavia, sono stati effettuati dei test anche su un'altra versione costituita invece da 40 moduli elementari, ovvero un codice in cui l'area del modulo elementare è di 4 volte più piccola rispetto a quella proposta. In figura 4.18 è visualizzato il contorno dell'immagine così da evidenziare la fascia perimetrale di colore bianco, indispensabile per la corretta lettura del codice. Lo SPinV Code presenta delle proporzioni di 4:3 tra lato maggiore e lato minore, ovvero le medesime proporzioni della fotografia. Il processo di decodifica/lettura dello SPinV Code è legato al processo di

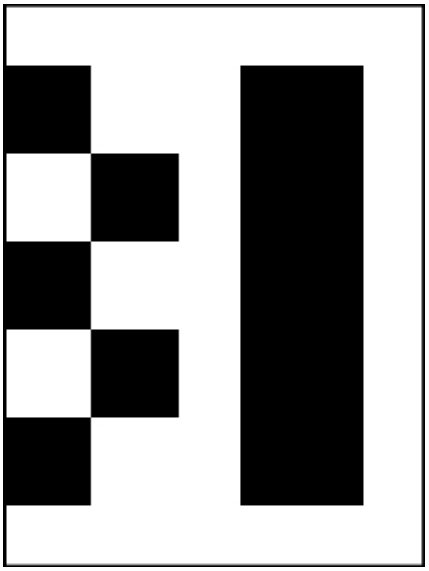


Figura 4.18: SPinV Code che codifica la sequenza 0101010101

decodifica del file jpeg e queste due ”operazioni” sono, in pratica eseguite congiuntamente. Nel paragrafo 5.4 verranno descritte le tecniche utilizzate per la decodifica dello SPinV Code e ulteriori informazioni sono presenti nello script in Appendice B. Per adesso è sufficiente considerare che in fase di decodifica del file jpeg e dello SPinV Code, si sfrutta la miniatura dell'immagine, che sostanzialmente rappresenta una versione a bassa definizione della fotografia ”originale”. I dati della miniatura, sono contenuti rispettando un ordinamento orizzontale, ovvero i dati sono codificati, come se il codice di figura 4.18, fosse ruotato di 90° in senso antiorario. La codifica del file jpg avviene, su questa versione ruotata, da destra verso sinistra e dall'alto verso il basso. In pratica, in fase di decodifica, dapprima verrà individuato il blocco rettangolare e in seguito, sulla base del posizionamento di questo, verrà individuato il posizionamento dei singoli moduli elementari. Verranno letti dapprima i 5 moduli presenti sulla prima riga, e in seguito i 5 moduli sulla seconda riga.

Capitolo 5

Il Processo di Decodifica

Indice

5.1 Introduzione 121
5.2 La piattaforma di sviluppo e le Applicazioni 122
5.3 Lo Standard Jpg 123
5.3.1 Codifica Jpeg 124
5.3.1.1 Nozioni preliminari 125
5.3.1.2 Trasformata Discreta del Coseno (DCT) 129
5.3.1.3 Quantizzazione e preparazione alla codifica . . . 134
5.3.1.4 Codifica Entropica 138
5.3.1.5 Scrittura e Struttura del File 146
5.3.2 Decodifica Jpeg 156
5.4 Decodifica SPinV Code 167

5.1 Introduzione

Nei capitoli precedenti si sono affrontate le tematiche relative alla localizzazione, all'autolocalizzazione e alla navigazione basata sui sensori inerziali. In questo capitolo si entrerà più nel dettaglio, descrivendo il processo di co-decodifica del file nel formato jpg e dello SPinV Code. Si è scelto di descrivere i due processi di decodifica nello stessa sezione, proprio perché nel codice C implementato e riportato in Appendice B, la decodifica del file jpg e la decodifica del codice SPinV sono effettuate congiuntamente, per motivi che saranno spiegati nel seguito. Inizialmente verrà fatta una panoramica sulla piattaforma di sviluppo utilizzata e sul mondo delle applicazioni. In seguito si descriverà il processo di codifica/decodifica dello standard jpg e infine le strategie utilizzate per la decodifica dello SPinV Code.

5.2 La piattaforma di sviluppo e le Applicazioni

Il codice presente in Appendice B è stato scritto interamente in linguaggio C, utilizzando la piattaforma di sviluppo proposta dalla Apple. Questa piattaforma prevede un SDK (Software Development Kit) che è un insieme di strumenti e programmi che consentono di sviluppare applicazioni native per dispositivi Apple. Tramite questo SDK è possibile creare applicativi, non solo per iPhone, ma anche per iPad, iPod Touch e Mac. Alcuni strumenti dell'SDK sono Xcode, Interface Builder, DashCode, Instruments e Quatz Composer. In questo lavoro di tesi si è usato essenzialmente Xcode. Il linguaggio di programmazione previsto è l'Objective C che è un linguaggio di programmazione ad oggetti basato sul C. Objective C è un linguaggio nato nei primi anni Ottanta ed acquisito nel 1995 dalla NeXT, compagnia fondata da Steve Jobs (si veda Novelli [32]). In seguito, la Apple acquista la NeXT, i tool di sviluppo per programmi di terze parti e il sistema operativo NeXTSTEP. Da questo momento l'Objective C diviene il linguaggio di sviluppo ufficiale del mondo Apple. La svolta vera si ha il 6 Marzo 2008, data in cui la Apple presenta un SDK per iPhone, mediante il quale qualsiasi sviluppatore può creare applicazioni per iPhone e iPod Touch; con le versioni successive dell'SDK è possibile sviluppare applicazioni anche per iPad. Sino alla data di lancio dell'SDK, non era possibile scrivere applicazioni native Apple. In pratica, dal lancio dell'Apple iPhone, nel Gennaio del 2007, al Marzo 2008, l'unico modo di estendere le capacità dell'iPhone, era di usare le WebApp. Le WebApp sono applicazioni utilizzabili solo tramite browser e per questo motivo possono essere usate solo se si è connessi ad internet. I linguaggi utilizzati per queste applicazioni sono solo linguaggi web quali JavaScript e HTML e questa caratteristica si "porta dietro" i limiti di questi linguaggi: non è possibile scrivere giochi complessi o in 3D e non è possibile sfruttare appieno le caratteristiche dell'iPhone, quali accelerometro e giroscopio.

Le applicazioni create da qualsiasi sviluppatore possono essere messe in vendita nell'Apple Store, un negozio virtuale, in cui ogni utente può visionare, scegliere e acquistare applicazioni in maniera molto semplice e diretta, senza passare dal sito web dello sviluppatore. Questo innovativo metodo ha avuto un immenso successo, restano comunque moltissime limitazioni imposte dalla Apple:

- ogni applicazione, prima di essere pubblicata in Apple Store, deve essere controllata e approvata dalla Apple;
- non si possono usare librerie esterne;
- Le API fornite non sono complete;

A fronte però di questi problemi si affianca il grossissimo vantaggio della semplicità di installazione e di stabilità di tutte le applicazioni, componenti che hanno influito in

maniera determinante all'enorme successo e sviluppo dell'iPhone e di altri dispositivi Apple in un tempo relativamente breve dal momento del lancio.

Il linguaggio Objective C, come detto in precedenza, deriva dal linguaggio C: in pratica è possibile utilizzare, almeno in parte, il C per la realizzazione di applicazioni Apple. Inoltre, Objective C è utilizzato solo su piattaforme Apple, per cui scrivendo il codice in C, non si preclude la possibilità di portare l'applicazione su altre piattaforme mobili, come ad esempio Windows Phone 7 o Symbian OS (che non supportano invece l'Objective C).

5.3 Lo Standard Jpg

Il processo di autolocalizzazione, come detto nel Capitolo 4, prevede di scattare una fotografia al codice posizionale sfruttando la fotocamera dell'iPhone. Questa fotografia verrà registrata in memoria nel formato jpeg (.jpg, .jpeg, .jpe, .jif, .jfif, .jfi) consentendo così un occupazione più ridotta rispetto all'uso di altri formati non compressi. Il jpeg è lo standard utilizzato, non solo per le fotografie in iPhone, ma nella quasi totalità dei dispositivi fotografici digitali, in questi casi si parla di immagini DSC (Digital Still Camera). jpeg è l'acronimo che sta per joint photographic expert group, che è un comitato ISO/CCITT che ha appunto definito lo standard internazionale jpeg relativo alla compressione di immagini a tono continuo, sia a livelli di grigio, che a colori. É importante evidenziare che lo standard jpeg specifica le tecniche di compressione che permettono di trasformare un immagine in uno stream di byte; però analizzando, a livello di byte, un file jpeg è possibile notare che lo stream memorizzato presenta delle differenze rispetto allo stream "teorico" jpeg. Questa differenza è dovuta al tipo di formato di file utilizzato per la memorizzazione. Un immagine in jpeg può essere memorizzata rispettando diversi formati di file. In questo lavoro di tesi, si è considerato il formato di file JFIF (Jpeg File Interchange Format), che è il formato base. Per cui, quando si parla di file jpeg, ci si può riferire al jfif o ad altri formati di incapsulamento (ad esempio exif). Inoltre, il jpeg è uno standard che ha visto molte "evoluzioni" rispetto alla prima standardizzazione che in pratica è l'unica considerata in questo lavoro di tesi. A parte gli aspetti relativi al formato di incapsulamento, che verranno meglio analizzati nel seguito, il jpeg rappresenta lo standard più utilizzato per la memorizzazione di immagini fotografiche e tale diffusione è dovuta principalmente al fatto che permette di ottenere alti rapporti di compressione su immagini naturali a fronte di una perdita di qualità generalmente contenuta entro certi limiti ([43]). Quindi il primo passo che si è reso necessario, per poter trattare con delle fotografie effettivamente scattate e memorizzate in iPhone, è stata la comprensione completa e specifica del processo di codifica e decodifica delle immagini in jpeg e delle modalità in cui lo stream viene incapsulato in memoria. A

tal proposito si sono presi come riferimento lo standard IEEE ([44]), Wallace [45], Hamilton [46], Scarano [43], Cohen [47] e Colonnese [48]. Per la completa compatibilità di decodifica dei file jpeg, occorre però tener conto anche dei seguenti standard, oltre allo standard ISO/IEC 10918-1:1994 (riferimento [44]).

- ISO/IEC 10918-2:1995: Compliance testing;
- ISO/IEC 10918-3:1997: Extensions;
- ISO/IEC 10918-4:1999: Registration of JPEG profiles, SPIFF profiles, SPIFF tags, SPIFF colour spaces, APPn markers, SPIFF compression types and Registration Authorities (REGAUT);
- ISO/IEC FDIS 10918-5: JPEG File Interchange Format (JFIF);

Lo standard jpeg prevede quattro diverse modalità di funzionamento, che possono essere suddivise considerando la compressione senza perdita di informazione (lossless) e la compressione con perdita di informazione (lossy):

- jpeg lossy:
  - modalità sequenziale;
  - modalità gerarchica;
  - modalità progressiva;
- jpeg lossless;

Nel seguito si prenderà in considerazione la modalità sequenziale, in riferimento a quanto scritto nello standard ISO/IEC 10918-1, in particolare nella sezione relativa alla codifica, si descriveranno i concetti da un punto di vista più concettuale, mentre nella sezione relativa alla decodifica, si sottolineeranno maggiormente gli aspetti pratici e realizzativi.

5.3.1 Codifica Jpeg

Le componenti principali del processo di codifica in modalità sequenziale, sono le seguenti:

1. trasformata discreta del coseno (DCT);
2. quantizzazione;
3. codifica entropica;

Nel seguito si descriveranno ognuno di questi tre passi e verrà inoltre introdotto l'argomento (parte 5.3.1.1) fornendo alcune informazione integrative necessarie alla caratterizzazione del problema. Infine si descriveranno le procedure legate alla scrittura del file.

Lo schema di riferimento del codificatore jpeg è quello rappresentato in figura 5.1:

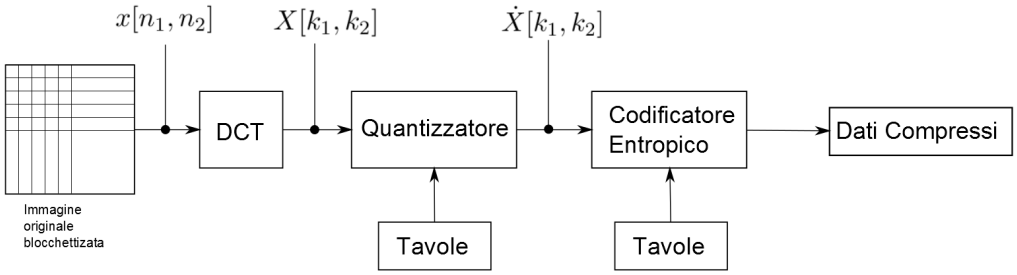


Figura 5.1: Schema a blocchi concettuale del codificatore jpeg

5.3.1.1 Nozioni preliminari

Un immagine digitale monocromatica in scala di grigi (immagine in bianco e nero) può essere rappresentata da una matrice numerica di dimensione M x N, in cui M e N rappresentano rispettivamente il numero di righe e il numero di colonne della matrice, ovvero la dimensione verticale e orizzontale in pixel dell'immagine (si veda la figura 5.7 a pag.137). Il generico elemento (i, j) della matrice, con $i = 1, 2 \dots M$ e $j = 1, 2 \dots N$, è il valore che rappresenta l'intensità del grigio, che varia dal bianco al nero, ovvero il tipo di grigio del pixel nella posizione (i, j) dell'immagine. Questo valore varia in un intervallo dipendente dal numero di bit P utilizzati per rappresentare il singolo elemento. P è spesso indicato come profondità. L'intervallo è quindi $[1; 2^P]$, per cui nelle immagini a 8 bit per pixel è possibile rappresentare il singolo elemento con 256 differenti valori. La rappresentazione numerica appena descritta è relativa alle immagini raster (o immagini bitmap) che si contrappone alle rappresentazione delle immagini di tipo vettoriale in cui non si utilizza una matrice numerica, ma si descrivono gli elementi primitivi, come rette o poligoni.

Se l'immagine raster è invece a colori, per rappresentare ogni pixel è necessaria almeno una terna di numeri, infatti un colore può essere riprodotto miscelando una terna di sorgenti opportunamente scelte, chiamati primari. Con lo spazio RGB,

ogni pixel dell'immagine è rappresentato mediante una terna numerica che descrive l'ammontare di rosso, di verde e di blu. Se la profondità di ogni colore è di 8 bit, sarà possibile rappresentare in totale 256^3 colori, ovvero 16777216 differenti colori, in questo caso si parla di 24 bpp (bit per pixel). Oltre all' $R\,G\,B$, esistono altri spazi di colore, in particolare noi siamo interessati allo spazio $Y\,C_b\,C_r$, in cui la Y rappresenta la componente di luminanza e C_b e C_r rappresentano le componenti di differenza di colore, anche chiamate componenti di crominanza. La terna $Y\,C_b\,C_r$, può essere valutata a partire dalla terna $R\,G\,B$, mediante la seguente trasformazione:

$$\begin{cases} Y = 0.299\,R + 0.587\,G + 0.114\,B \\ C_b = -0.1687\,R - 0.3313\,G + 0.500B \\ C_r = 0.500\,R - 0.4187\,G - 0.0813B \end{cases} \tag{5.1}$$

Lo spazio di colore $Y\,C_b\,C_r$ è definito nello standard internazionale ITU.BT-601, ed è lo standard che definisce la codifica del segnale video analogico interlacciato in forma digitale (si veda Union [49]). Il problema relativo agli spazi di colore e alle trasformazioni, è in realtà più complesso e per una trattazione più rigorosa bisognerebbe esprimere la 5.1 tendendo conto del problema della gamma correction e considerando il processo di quantizzazione (rappresentazione delle immagini a colori in un dominio discreto). Una definizione più rigorosa è presente nello standard ITU.BT-601 (riferimento [49]); in ogni caso, questo aspetto verrà caratterizzato in maniera più completa nel seguito, in particolare nella sezione 5.3.1.5. Per adesso, è sufficiente considerare che nella terna $Y\,C_b\,C_r$, analogamente ad altre terne che sfruttano l'informazione di luminanza e le differenze di colore, la Y è la componente che esprime la luminosità dell'immagine, in particolare la definizione di questa deriva direttamente dalle caratteristiche del sistema visivo, infatti la luminanza trae potenza, approssimativamente per l'11% dalle regioni blu, per il 59% dalle regioni verdi e per il restante 30% dalle regioni rosse. La definizione Y, come si vede dalla 5.1, sfrutta questo concetto per produrre un unico segnale Y in grado di rappresentare la luminosità dell'elemento, senza esprimere informazioni sul colore. La componente C_b si ottiene dalla differenza, opportunamente pesata della componente blu dell'$R\,G\,B$ con la luminanza. La componente C_r è ottenuta dalla differenza, ancora opportunamente pesata, della componente red dell'$R\,G\,B$ con la luminanza. É importante sottolineare, che esiste anche un altro spazio di colore identificato con la terna $Y\,C_b\,C_r$, che è descritto nello standard ITU.BT-709. Ovviamente i due spazi di colore sono differenti.

La rappresentazione $Y\,C_b\,C_r$ è nata a partire dalle peculiarità del nostro sistema visivo, che è maggiormente sensibile alle informazioni di luminanza rispetto alle

informazioni di crominanza, per cui sarà possibile adottare frequenze di campionamento f_c diverse per i due tipi di componenti, ovvero una maggiore f_c per Y rispetto alla f_c di C_b e C_r. In sostanza, a parità di qualità percepita, mediante la rappresentazione $Y\,C_b\,C_r$ si otterrà un occupazione minore, rispetto alla rappresentazione $R\,G\,B$. Per quanto riguarda il sottocampionamento delle componenti di crominanza rispetto alla componente di luminanza, si utilizza in genere la seguente simbologia:

- 4:4:4 → Stessa frequenza di campionamento per tutte e tre le componenti, ovvero a 4 campioni di luminanza, corrispondono 8 campioni di crominanza;
- 4:2:2 → Si intende che i campioni di crominanza sono sottocampionati di un fattore 2 nella sola direzione orizzontale. Ogni 4 campioni di luminanza, ci sono 2 campioni C_b e due campioni C_r, quindi 4 campioni di crominanza;
- 4:2:0 → Sottocampionamento dei campioni di crominanza di un fattore 2 sia in direzione orizzontale, che in direzione verticale. Ogni 4 campioni di luminanza, c'è un campione C_b e un campione C_r, quindi 2 campioni di crominanza.

Esistono anche altre modalità di sottocampionamento, oltre a quelle appena citate e altri formalismi di rappresentazione, tuttavia gli schemi citati sono quelli maggiormente utilizzati.

Nello spazio RGB, se la profondità del colore è a 8 bit ($P = 8$), ogni elemento della terna può assumere un valore che varia tra 0 e 255, mentre se la profondità del colore è di 12 bit ($P = 12$) ogni elemento potrà assumere valori tra 0 e 4095; applicando la trasformazione espressa dalla 5.1, nel caso di $P = 8$, la componente di luminanza Y varia tra 0 e 255 e le componenti di crominanza C_b e C_r varieranno tra -128 e $+127$. Nello standard jpeg, prima dell'applicazione della DCT, è necessario che i valori della terna siano compresi nell'intervallo $[-128; 127]$. Ciò implica un operazione di "*level shift*" così che i valori siano nell'intervallo definito. Nel caso $P = 8$ e spazio RGB, lo shift deve essere eseguito su tutte e tre le componenti; nello spazio YC_bC_r, lo shift deve essere eseguito solo sulla componente Y. In caso di immagini con $P = 12$, lo shift si applica allo stesso modo ma è pari a $2^{12-1} = 2048$:

$$\begin{cases} (R, G, B) \rightarrow SHIFT \rightarrow (R - 2^{P-1}, G - 2^{P-1}, B - 2^{P-1}) \\ (Y, C_b, C_r) \rightarrow SHIFT \rightarrow (Y - 2^{P-1}, C_b, C_r) \end{cases} \tag{5.2}$$

In seguito alla trasformazione $R\,G\,B \rightarrow Y\,C_b\,C_r$ e allo SHIFT, saranno presenti tre matrici di uguali dimensioni (MxN) i cui elementi possono assumere valori nell'intervallo $[-128 \div 127]$. In seguito all'eventuale sottocampionamento, le tre matrici non avranno invece la stessa dimensione. Così ad esempio, se si è operato un sottocampionamento 4:2:0, la matrice di luminanza avrà dimensione MxN, mentre le due matrici di crominanza avranno dimensione (M/2)x(N/2).

Nello standard jpeg, si lavora su blocchetti 8x8 pixel, per cui l'immagine di partenza MxN viene suddivisa in blocchetti 8x8. Nel gergo, questo processo viene chiamato *blocchettizzazione* e il blocchetto 8x8 è chiamato blocchetto elementare. In riferimento alla figura 5.1 di pag.125, nel 'blocco DCT', entreranno i blocchetti 8x8 sottocampionati. L'ordine in cui vengono letti e analizzati i blocchetti delle tre componenti, dipende dalle modalità di sottocampionamento applicata all'immagine. La situazione è descritta in figura 5.2 (immagine estratta da [45]).

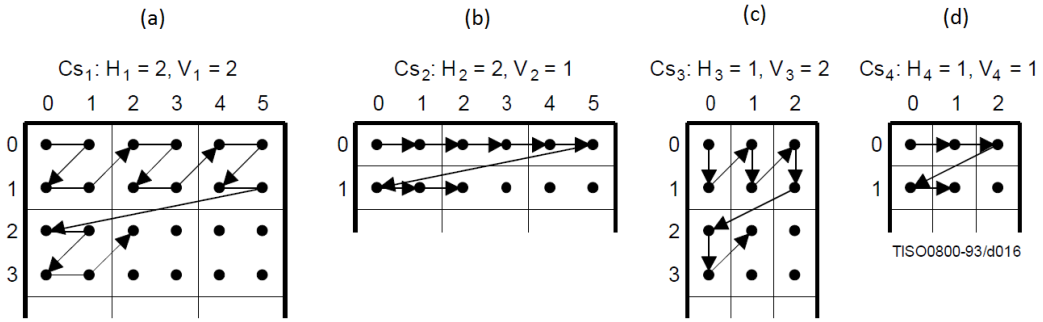


Figura 5.2: Ordine di analisi dei blocchetti 8x8 al variare del sottocampionamento effettuato

In figura 5.2 sono indicati i quattro possibili tipi di sottocampionamento e il relativo ordine di analisi. Nello standard viene usata una nomenclatura leggermente diversa da quella indicata in precedenza, per esprimere il sottocampionamento effettuato. In sostanza (H,V)=(2,2), (H,V)=(2,1) e (H,V)=(1,1) coincidono con i sottocampionamenti 4:4:4, 4:2:2 e 4:2:0. Per esprimere meglio questo concetto considero ad esempio un immagine a colori (3 componenti) di 112x160 pixel. Operando un sottocampionamento 4:2:0, si avranno 14x20 blocchetti di luminanza Y, 7x10 blocchetti di crominanza C_b e 7x10 blocchetti di crominanza C_r. Prima saranno analizzati i 4 blocchetti Y, nell'ordine indicato nella 5.2.a, in seguito un blocchetto C_b ed infine un blocchetto C_r (5.2.d). Nel bitstream, saranno presenti i blocchetti con quest'ordine: $Y, Y, Y, Y, C_b, C_r, Y, Y, Y, Y, C_b, C_r, Y \ldots$. Infatti i 4 blocchetti Y rappresentano una porzione di 64x64 pixel, ma anche il blocchetto C_r e il blocchetto C_b, a valle dell'operazione inversa del sottocampionamento, rappresenta una porzione dell'immagine di 64x64 pixel. Se invece l'immagine è costituita da un unica componente, ovvero è un immagine monocromatica in scala di grigi, l'ordine di analisi dei blocchetti è "left-to-right e top-to-bottom", come indicato in figura 5.3.

In definitiva, in fase di codifica, la prima operazione è la trasformazione $RGB \rightarrow Y C_b C_r$ e in seguito viene effettuata l'operazione di sottocampionamento secondo uno degli schemi proposti.

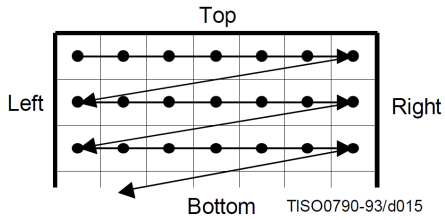


Figura 5.3: Ordine di analisi dei blocchetti in immagini con un unica componente

Finora ho considerato immagini di dimensione multipla del blocchetto elementare 8x8. Vediamo cosa succede per immagini la cui dimensione in pixel non è un multiplo del blocchetto elementare. In questi casi i blocchetti di bordo (inferiore e di destra) andranno completati: si aggiungono tante righe e/o colonne contenenti informazioni uguali all'ultima riga o colonna dell'immagine fino ad ottenere un immagine che presenta una dimensione multipla di 8x8. Così facendo altero l'immagine e la sua dimensione che potrà essere recuperata, in fase di decodifica, fornendo all'interno del file codificato la dimensione originaria. Il processo decritto è rappresentato in figura 5.4.

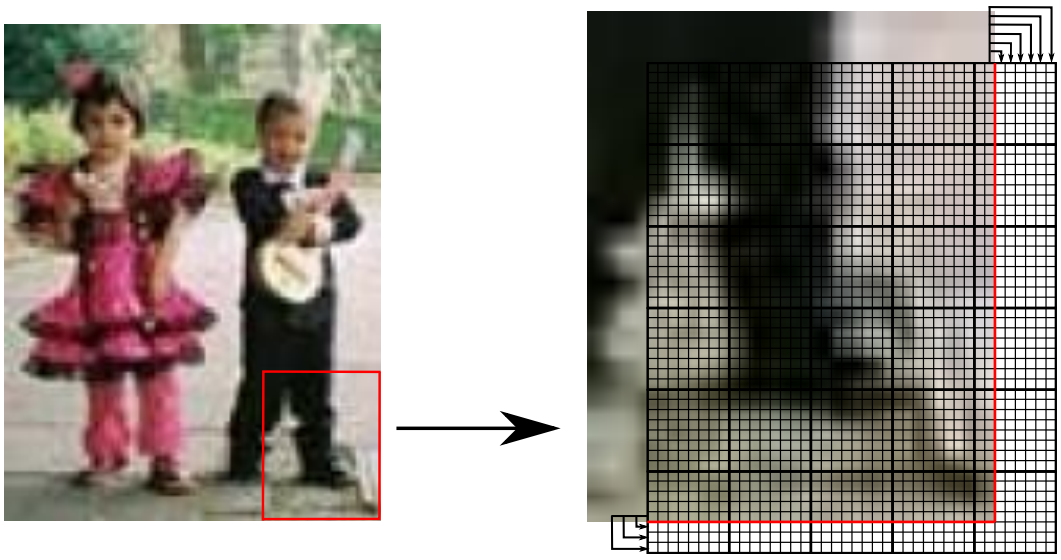


Figura 5.4: Completamento dei blocchetti in immagini con dimensione non multipla del blocchetto elementare

5.3.1.2 Trasformata Discreta del Coseno (DCT)

La Discrete Cosine Transform è un caso particolare della trasformata *Karhounen Loeve* (KLT) per processi di Markov di ordine 1 (si veda Colonnese [48]).
La KLT è una trasformazione che consente di rappresentare in modo approssimato

una sequenza aleatoria di lunghezza finita N, con un numero fissato $M < N$ di coefficienti. Si utilizza una base ortonormale di rappresentazione $\mathbf{b}^{(k)}$ con $k = 0, 1, \ldots N-1$ che è scelta così che venga minimizzato l'errore quadratico medio che viene commesso rappresentando la sequenza $\mathbf{x}$ con una sequenza $\hat{\mathbf{x}}$ composta da un minor numero di coefficienti.

$$\mathbf{x} = \sum_{k=0}^{N} X_k \mathbf{b}^{(k)} \quad \rightarrow \quad \hat{\mathbf{x}} = \sum_{k=0}^{M-1} X_k \mathbf{b}^{(k)} + \sum_{i=M}^{N-1} C_k \mathbf{b}^{(k)} \tag{5.3}$$

Quindi si conservano M coefficienti X_k e si sostituiscono i restanti $(N - M)$ con valori costanti. L'obiettivo di tale trasformazione è minimizzare l'errore quadratico medio:

$$\varepsilon = E\{\|\mathbf{x} - \hat{\mathbf{x}}\|^2\} = E\left\{ \sum_{k=M}^{N-1} |X_k - C_k|^2 \right\} \tag{5.4}$$

Per questo obiettivo e affinché sia valida la 5.3, dovranno essere rispettate le seguenti condizioni sui coefficienti C_k e sugli autovettori della matrice di covarianza $\mathbf{b}^{(k)}$:

$$\begin{cases} C_k = E\{X_k\} = E\{\mathbf{x}\}^H \mathbf{b}^{(k)} & k = 0, 1, \ldots N-1 \\ E\Big\{ (\mathbf{x} - E\{\mathbf{x}\}) \,.\, (\mathbf{x} - E\{\mathbf{x}\})^H \Big\} \mathbf{b}^{(k)} = \lambda_k \mathbf{b}^{(k)} \end{cases} \tag{5.5}$$

dove λ_k, sono gli autovettori della matrice di covarianza. Per dimostrare le condizioni espresse nella 5.5, si parte dalla derivata della 5.4 e si applica il risultato $(C_k = E\{X_k\})$ alla 5.4, considerando la definizione di prodotto scalare in geometria analitica $X_k =< \mathbf{b}^{(k)}, \mathbf{x} >= (\mathbf{b}^{(k)})^H \,.\, \mathbf{x}$, le proprietà dell'operatore Expectation ed infine il metodo dei moltiplicatori di Lagrange.

La KLT rappresenta la trasformata ottima dal punto di vista della compattazione dell'energia (che si traduce in un numero di limitato di coefficienti), inoltre porta ad avere i coefficienti decorrelati fra loro e con differente rilevanza (che si traduce nella possibilità di "risparmiare" bit sui coefficienti meno significativi). Nel caso particolare di processi di Markov di ordine 1, gli autovettori della matrice di covarianza, coincidono con i vettori della base DCT, ovvero la DCT coincide con la KLT. In pratica, una vasta classe di immagini è approssimabile mediante processi di Markov di ordine 1, per cui la DCT applicata ai campioni di un immagine è una buona

approssimazione della trasformata ottima, ecco perché la DCT viene utilizzata nella quasi totalità degli standard di compressione video, in particolare si sfrutta la DCT bidimensionale (DCT-2) di blocchetti quadrati di una certa dimensione.

Considero di voler applicare la DCT-2 ad un blocchetto di pixel dimensione NxN. L'elemento di questo blocchetto è $x[n_1, n_2]$ con $n_1, n_2 = 0, 1, \dots N-1$. Analogamente a quanto fatto prima, è possibile rappresentare il blocco in un opportuna base (bidimensionale) dello spazio vettoriale. Nella DCT-2 tale base vale:

$$b^{k_1,k_2}[n_1, n_2] = \alpha(k_1)\,\alpha(k_2)\,cos\left(\frac{\pi k_1(2\,n_1+1)}{2N}\right) cos\left(\frac{\pi k_2(2\,n_2+1)}{2N}\right) \tag{5.6}$$

con:

$$\begin{cases} n_1, n_2 = 0, 1, \dots N-1 \\ k_1, k_2 = 0, 1, \dots N-1 \\ \alpha(k_1), \alpha(k_2) = \begin{cases} 1/\sqrt{N} & per \quad k_1, k_2 = 0 \\ \sqrt{2/N} & per \quad 1 \le k_1, k_2 \le (N-1) \end{cases} \end{cases}$$

É possibile eseguire la seguente fattorizzazione della base ortonormale:

$$b^{k_1,k_2}[n_1, n_2] = b^{k_1}[n_1]\,.\,b^{k_2}[n_2] \tag{5.7}$$

Questa fattorizzazione può essere interpretata come il prodotto di due componenti, una relativa alle righe e l'altra relative alle colonne del blocchetto in analisi. Queste due basi possono essere considerate come le basi usate nella DCT monodimensionale.

I coefficienti dello sviluppo X_{k_1,k_2} relativi al blocco $x[n_1, n_2]$ si determinano con il prodotto scalare (in geometria analitica) con le relative funzioni della base:

$$X[k_1, k_2] =< x[n_1, n_2]\,,\, b^{k_1,k_2}[n_1, n_2] >= \sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x[n_1, n_2]\, b^{k_1,k_2}[n_1, n_2] \tag{5.8}$$

ovvero, la trasformata DCT-2 (*relazione di analisi*) vale:

$$X[k_1, k_2] = \alpha(k_1)\,\alpha(k_2) \sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x[n_1, n_2]\, cos\left[\frac{\pi k_1(2\,n_1+1)}{2N}\right] cos\left[\frac{\pi k_2(2\,n_2+1)}{2N}\right] \tag{5.9}$$

a partire da questi coefficienti, è possibile sintetizzare il blocco originario (*relazione di sintesi*), ovvero si ottiene la trasformata IDCT:

$$x[n_1, n_2] = \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} \alpha(k_1)\,\alpha(k_2)\; X[k_1, k_2]\, cos\left[\frac{\pi k_1(2\,n_1+1)}{2N}\right] cos\left[\frac{\pi k_2(2\,n_2+1)}{2N}\right] \tag{5.10}$$

Quindi la sequenza bidimensionale rappresentativa dell'immagine è espressa mediante la sovrapposizione, opportunamente pesata, di N^2 sequenze $b^{k_1,k_2}[n_1, n_2]$ che costituiscono la base ortonormale. Per comprendere meglio il significato della base ortonormale, si osservi la figura 5.5 (immagine estratta da [43]), basata su blocchetti 8x8 e la definizione di $b^{k_1,k_2}[n_1, n_2]$ fornita nell'equazione 5.6.

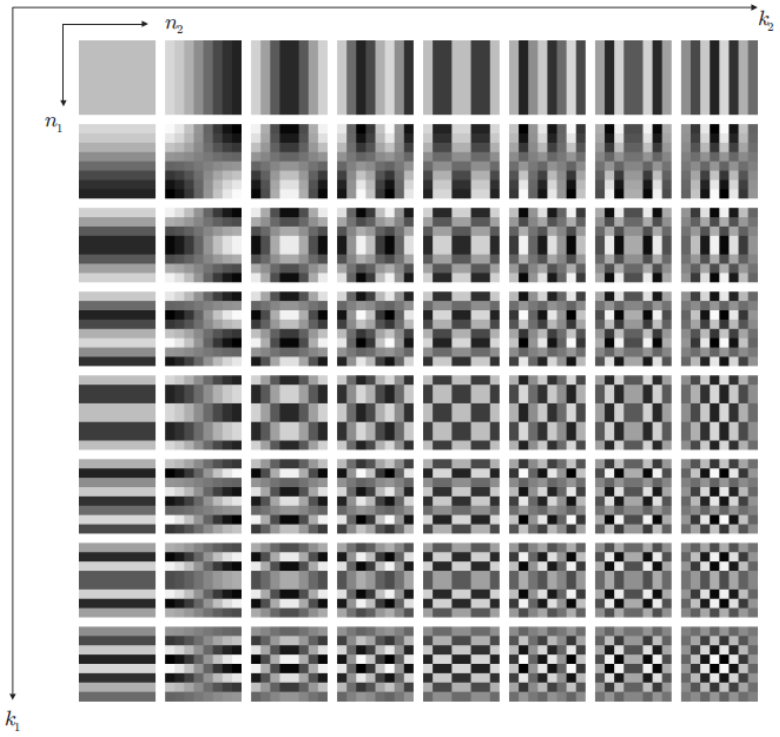


Figura 5.5: Rappresentazione in scala di grigi della base DCT

Al crescere degli indici k_1 e k_2 si ha una rappresentazione via via più fine. Le componenti più significative a livello visivo, sono quelle per bassi valori di k_1 e k_2 e al crescere di questi coefficienti, decresce l'importanza visuale. Così ad esempio per k_1=k_2=0, si ha che tutti gli N^2 valori della base ortonormale (ovvero per ogni valore di n_1 e n_2) assumono ampiezza costante. Nella figura 5.5, questa situazione è rappresentata da un blocco 8x8 di colore uniforme, che indica appunto i 64 coefficienti

della stessa ampiezza. Per $k_1 = 0$ e $k_2 = 1$ si ha lungo le colonne un andamento costante e lungo le righe un andamento cosinusoidale.

Il coefficiente $X[0,0]$ è chiamato *coefficiente DC* ed è la componente che rappresenta il valor medio del blocchetto; i restanti $N^2 - 1$ coefficienti DCT, sono chiamati *coefficienti AC*.

Le considerazioni appena fatte confermano che la trasformata DCT è una trasformata ottima in grado di fornire una diversa rilevanza ai coefficienti più importanti. Il nostro sistema visivo è maggiormente sensibile alle componenti di bassa frequenza e tale sensibilità decresce al crescere della frequenza, per cui i coefficienti $X[k_1, k_2]$ corrispondenti alle basse frequenze necessitano di una "buona" quantizzazione, e al crescere della frequenza si può adottare una quantizzazione via via meno fine.

Il significato "pratico" della DCT-2 è visibile osservando la figura 5.6 (immagine estratta da [43]), in cui è visualizzata, in scala di grigi, l'ampiezza dei coefficienti DCT. In particolare, in bianco sono rappresentati i livelli più alti e in nero quelli più bassi.

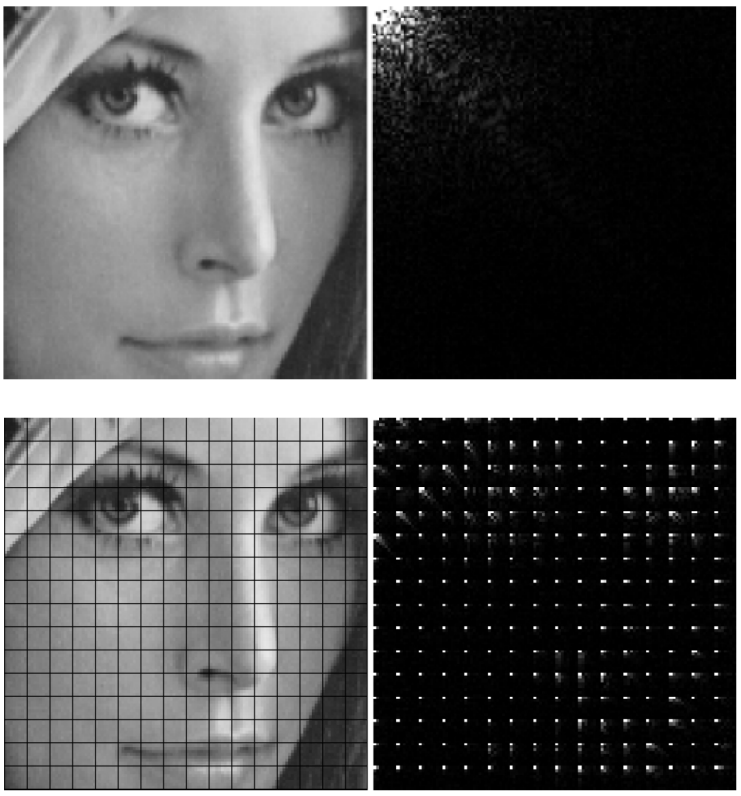


Figura 5.6: Immagine e relativa trasformata DCT-2

Dalla 5.6, si vede come viene compattata l'energia media nelle componenti a bassa frequenza . Questo comportamento può anche essere interpretato osservando l'equazione 5.8, in cui il prodotto scalare tra l'immagine (nel blocchetto NxN)

$x[n_1, n_2]$ e la base $b^{k_1,k_2}[n_1, n_2]$ rappresenta l'energia incrociata. In alto l'immagine originale (non blocchettizzata) e i coefficienti DCT, in basso l'immagine blocchettizzata in blocchetti 8x8 pixel e i coefficienti DCT. Un esempio numerico di trasformata DCT bidimensionale di un blocchetto 8x8 (estratto da un immagine) è mostrato nelle tabelle

11	16	21	25	27	27	27	27
16	23	25	28	31	28	28	28
22	27	32	25	30	28	28	28
31	33	34	32	32	31	31	31
31	33	34	32	32	31	31	31
33	33	33	33	32	29	29	29
34	34	33	35	34	29	29	29
34	34	33	33	35	30	30	30

Tabella 5.1: Valori di luminanza di un blocchetto 8x8

235.620	-2.8073	-9.1411	-4.1851	-0.3750	-3.1184	-1.4903	2.7734
-23.9174	-17.3609	-5.3178	-2.3956	-3.4929	-1.2309	0.8164	0.2551
-11.9060	-7.2085	-4.1668	0.9520	2.5126	-0.1775	-1.6402	-0.6298
-4.3656	-2.5517	-1.0140	0.1197	1.6479	1.9776	-0.5554	-2.2967
1.8750	-1.9223	1.1432	0.6098	-0.1250	0.8327	0.4735	-0.7429
-0.0606	1.3824	0.5347	-0.1134	0.3494	1.0424	0.5913	-0.1803
-3.5922	-0.0371	1.1098	-0.1962	-1.4467	-0.1985	1.6668	1.6578
-2.3359	-0.8359	-0.0301	-0.6404	-1.3320	-0.4814	0.9782	1.1988

Tabella 5.2: DCT bidimensionale del blocchetto in tabella 5.1

Teoricamente la cascata DCT → IDCT non comporta alcuna perdita di informazione, in realtà sono presenti delle funzioni cosinusoidali, che forniscono valori reali, per cui tra i processi di codifica e decodifica, saranno presenti sempre degli errori di approssimazione. Inoltre nello standard non è specificato un preciso algoritmo da seguire, ma viene lasciata libertà allo sviluppatore e per questi motivi potrebbe accadere che due decodificatori forniscano risultati diversi; per risolvere questo problema, lo standard specifica dei test di sull'accuratezza che devono essere rispettati sia dal codificatore, che dal decodificatore.

5.3.1.3 Quantizzazione e preparazione alla codifica

Tra i 64 coefficienti $X[k_1, k_2]$ assumono valore più alto solo quelli a frequenze spaziali più basse, in virtù della compattazione dell'energia derivante dalla DCT-2. Inoltre, l'importanza dal punto di vista visivo dei coefficienti, decresce con il crescere della frequenza spaziale del coefficiente, ovvero al crescere di k_1 e k_2 con $k_1, k_2 =$

$0, 1, \ldots, 7$. Per questi motivi, si utilizzano delle opportune tabelle che specificano il passo di quantizzazione di ognuno dei coefficienti. Tale passo di quantizzazione sarà quindi più piccolo per le componenti in bassa frequenza e più elevato per quelle in alta frequenza.
L'operazione di quantizzazione è così definita:

$$\dot{X}[k_1, k_2] = round\left[\frac{X[k_1, k_2]}{Q[k_1, k_2]}\right] \qquad con \qquad k_1, k_2 = 0, 1, \ldots, 7 \tag{5.11}$$

Lo standard propone due matrici di quantizzazione, una per la componente di luminanza (tabella 5.3) e una per le due componenti di crominanza (tabella 5.4), tuttavia si possono usare anche matrici di quantizzazione diverse. Le matrici di quantizzazione usate devono essere infatti inserite nei dati codificati. Il processo di quantizzazione determina una perdita di informazione.

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Tabella 5.3: Passi di quantizzazione per la luminanza suggeriti nello standard

17	18	24	47	66	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
66	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Tabella 5.4: Passi di quantizzazione per le crominanze suggeriti nello standard

A seguito della traformazione DCT-2 e dell'operazione di quantizzazione, si avranno, in genere, una buona parte di coefficienti nulli. Considero ad esempio la quantizzazione dei coefficienti DCT della tabella 5.2, utilizzando i passi di quantizzazione espressi nella tabella 5.3. Il risultato è mostrato in tabella 5.5.

15	0	-1	0	0	0	0	0
-2	-1	0	0	0	0	0	0
-1	-1	0	0	0	0	0	0
-1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Tabella 5.5: Quantizzazione dei coefficienti DCT di tabella 5.2

Per completezza è utile aggiungere, che per aumentare il livello di compressione dell'immagine, ovvero per ridurre la dimensione del file (e la relativa qualità), con alcuni software (ad esempio Adobe Photoshop o Gimp) è possibile agire direttamente sulle matrici di quantizzazione. In pratica, se si vuole convertire/salvare l'immagine in formato jpeg, al momento del salvataggio del file viene richiesto con che qualità si vuole memorizzare il file. In genere è presente una manopola, con cui è possibile regolare tale qualità. Agendo su questa manopola non si fa altro che scalare le matrici di quantizzazione del software (che in genere coincidono con quelle suggerite nello standard). In particolare l'algoritmo usato è il seguente (si veda [50]):

1. con la manopola si decide il valore di Q, che può variare fra 0 e 100;

2. si calcola il fattore di scalatura S, con la seguente:

$$\begin{cases} Q < 50 & \rightarrow \quad S = 5000/Q \\ Q \geq 50 & \rightarrow \quad S = 200 - 2\,.\,Q \end{cases} \tag{5.12}$$

3. Indico con $Q[k_1, k_2]$ la matrice di quantizzazione di partenza e con $\bar{Q}[k_1, k_2]$ la matrice di quantizzazione modificata in base al fattore di scalatura S. Risulta:

$$\bar{Q}[k_1, k_2] = \left\lfloor \frac{S\,.\,Q[k_1, k_2] + 50}{100} \right\rfloor \tag{5.13}$$

Per $Q = 50$, le matrici di quantizzazione non variano; per $Q < 50$, gli elementi delle matrici di quantizzazione crescono e di conseguenza la quantizzazione diventa più "dura"; inversamente, per $Q > 50$ gli elementi di $\bar{Q}[k_1, k_2]$ sono più piccoli e quindi la quantizzazione è più fine e la qualità più elevata. Inoltre, ogni valore di $\bar{Q}[k_1, k_2]$, che vale 0 (ad esempio se $Q = 100$), viene settato a 1 e in tal caso non verrà effettuata alcuna quantizzazione sull'elemento corrispondente.

Dopo la quantizzazione, dovrà essere effettuato un determinato ordinamento dei dati (preparazione alla codifica) per la fase di codifica entropica. In particolare, a

valle del processo di quantizzazione, si avrà una matrice in cui la maggior parte dei coefficienti sono nulli e i pochi coefficienti non nulli sono quelli che si trovano alle basse frequenze. Inoltre, i coefficienti $\dot{X}[0,0]$ (coefficienti DC quantizzati) di blocchetti adiacenti dello stesso tipo, risultano simili fra loro. Ad esempio, i coefficienti DC quantizzati di due blocchetti 8x8 di luminanza adiacenti, rappresentano i valori medi di luminanza, che saranno quindi analoghi per "pezzetti" vicini dell'immagine . Per i motivi appena spiegati, risulta particolarmente conveniente attuare le seguenti strategie:

1. Si sfrutta una *codifica differenziale* fra il coefficiente DC quantizzato del blocchetto corrente e il coefficiente DC quantizzato del blocchetto precedente. Con questa codifica si otterranno quindi dei valori numerici più piccoli e di conseguenza una riduzione sul numero di bit utilizzati per rappresentare il valore in continua. Questo concetto è espresso nella figura 5.7. Se il blocchetto precedente non esiste, significa che siamo al primo blocchetto, in questo caso DC_{i-1} viene posto a zero e $\Delta DC_i = DC_i$. Questa operazione va effettuata sui coefficienti DC dei blocchetti di tutte le componenti dell'immagine (Y, C_b e C_r).

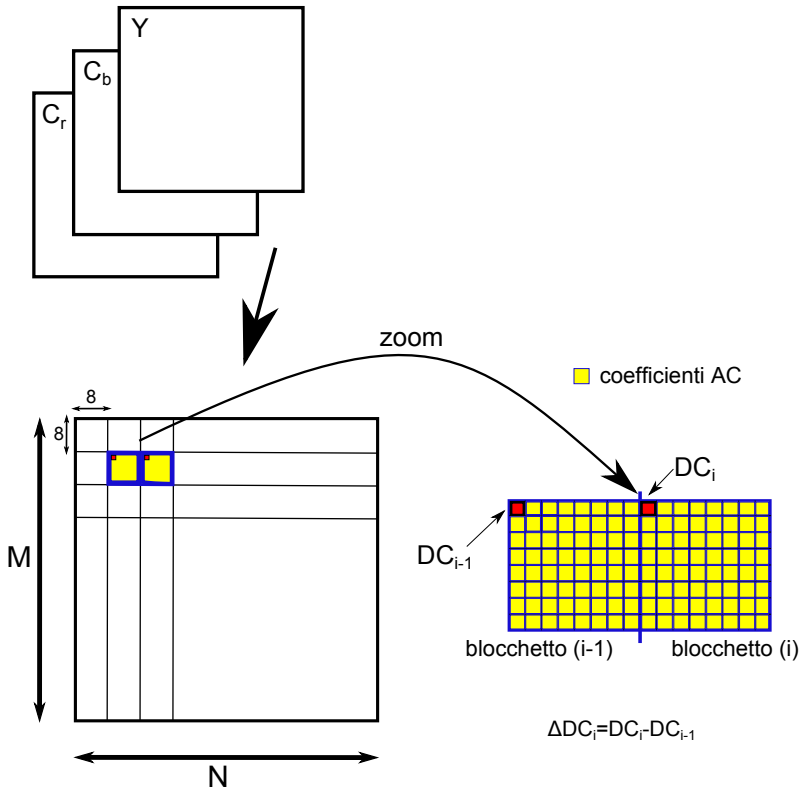
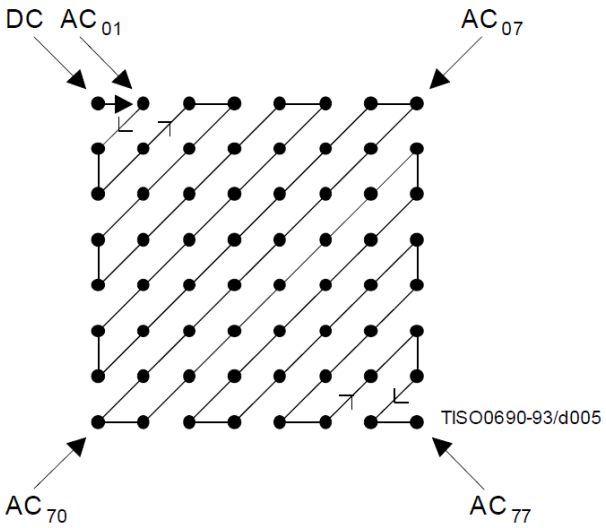


Figura 5.7: Codifica differenziale dei coefficienti DC

2. Per i 63 coefficienti AC, si esegue una scansione a zig zag, come mostrato in figura 5.8. Con questo tipo di scansione si esplorano dapprima le basse frequenze orizzontali e verticali, e successivamente le alte frequenze; si creano quindi delle stringhe di dati caratterizzate dall'avere un alternanza di coefficienti diversi da 0 e una successione di coefficienti nulli (corse di zeri).



0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

Figura 5.8: Scansione a zig-zag

5.3.1.4 Codifica Entropica

Il codificatore entropico (si veda la figura 5.1 di pag.125) riceve in ingresso una stringa di 64 coefficienti quantizzati, il coefficiente DC è codificato in maniera differenziale con il coefficiente DC del blocchetto precedente (si codifica quindi ΔDC) e i 63 coefficienti AC sono disposti seguendo una scansione a zig zag. In riferimento alla tabella 5.5 e supponendo che il coefficiente DC del blocco precedente vale $DC_{i-1} = 12$, a seguito della codifica differenziale e della scansione a zig zag, avremo in ingresso al codificatore entropico la seguente stringa:

$$3, 0, -2, -1, -1, -1, 0, 0, -1, \underbrace{0, 0, 0, \ldots 0}_{55\ zeri} \tag{5.14}$$

In cui il valore 3 rappresenta il coefficiente DC e, tra i restanti 63 coefficienti AC, solo cinque assumono valore non nullo.
Per i due tipi di coefficienti si utilizza la seguente rappresentazione:

- *Coefficiente* ΔDC: Viene rappresentato mediante la coppia di simboli: SIMBOLO 1 $\rightarrow$ *Size*, SIMBOLO 2 $\rightarrow$ *Amplitude*:

  - *Amplitude* coincide con il ΔDC ed è appunto l'ampiezza del coefficiente ΔDC in esame.

  - *Size* è il numero di bit necessari a rappresentare il coefficiente ΔDC (ovvero *Amplitude*); per vedere quanti bit servono per la rappresentazione, si usa la tabella 5.6. *Size* può assumere valori nell'intervallo $[0, 11]$ e di conseguenza *Amplitude* può codificare un valore compreso tra 0 e 2047.

Size	*Amplitude*
0	0
1	-1, 1
2	-3,-2,2 ,3
3	-7,-6,-5,-4, 4,5,6,7
4	-15,. . . ,-8, 8, . . . , 15
5	-31,. . . , -16 , 16, . . . , 31
6	-63, . . . , -32, 32, . . . , 63
7	-127, . . . ,-64 , 64, . . . , 127
8	-255, . . . , -128, 128, . . . , 255
9	-511, . . . -256, 256, . . . , 512
10	-1023, . . . -512, 512, . . . , 1023
11	-2047, . . . -1024, 1024, . . . , 2047

Tabella 5.6: Valori di *Size* in relazione all'ampiezza del coefficiente *Amplitude*

- *Coefficienti AC*: Viene rappresentato mediante la coppia si simboli: SIMBOLO 1 $\rightarrow$ $(Runlegth, Size)$, SIMBOLO 2 $\rightarrow$ *Amplitude*:

  - *Runlength* è il numero di zeri consecutivi e può assumere valori nell'intervallo $[0; 15]$, per cui può rappresentare al massimo corse di 15 zeri consecutivi; per cui serviranno 4 bit per codificarlo.

  - *Size* mantiene lo stesso significato di prima, ovvero rappresenta il numero di bit necessari a rappresentare l'ampiezza del coefficiente AC (ovvero a rappresentare *Amplitude*).

  - *Amplitude* è il primo numero diverso da zero, che viene trovato dopo la sequenza di zeri, oppure il primo numero, se è diverso da 0.

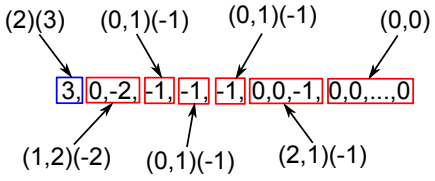
Inoltre per il SIMBOLO 1 del coefficiente AC - $(Runlegth, Size)$ - sono ammessi i seguenti valori speciali:

- $(Runlegth, Size) = (15, 0)$: rappresenta una corsa di 16 zeri. consecutivi; questa sequenza è indicata con la sigla *ZRL*; Si possono avere fino a un massimo di 3 ZRL all'interno dello stesso blocchetto, infatti se ce ne fossero 4 risulterebbe 16x4=64 zeri, il ché è impossibile visto che al massimo possono esserci 63 coefficienti AC e quindi al più 63 zeri.

- $(Runlegth, Size) = (0, 0)$: indica la fine del blocco e viene indicata con l'acronimo EOB (End Of Block); EOB è l'unico valore AC composto dal solo SIMBOLO 1.

Considerando la stringa espressa nella 5.14, e attuando i criteri di codifica entropica appena descritti, risulterà:

	SIMBOLO 1	SIMBOLO 2
Coefficiente DC	(Size)	(Amplitude)
Coefficiente AC	(Runlength, Size)	(Amplitude)



Per cui a partire dai 64 coefficienti originari descritti in tabella 5.1, tramite trasformata DCT bidimensionale, quantizzazione e codifica entropica, si è ottenuta la rappresentazione (con perdita) mediante l'uso di 19 coefficienti numerici (per questo caso particolare) di valore più ridotto rispetto ai valori di partenza. La sequenza ottenuta è una sequenza intermedia che dovrà essere codificata ulteriormente mediante codifica di Huffman, come si vedrà nel seguito. Analizziamo adesso l'occupazione in bit: se i valori dell'immagine sono rappresentati in virgola fissa a 8 bit/pixel, la rappresentazione di ciascun coefficiente DCT richiede 11 bit. Per immagini ad alta qualità a 12 bit/pixel, ciascun coefficiente DCT richiede 15 bit (si veda Scarano [43]). Per cui, considerando il caso di 8 bpp, il campo *Amplitude* può richiedere al massimo 11 bit; in particolare occuperà un numero di bit pari al valore di *Size* corrispondente. Il campo *Runlength*, come detto prima, può assumere valore tra 0 e 15, quindi serviranno 4 bit per rappresentarlo (1 simbolo esadecimale). *Size* invece può assumere valori tra 1 e 11, per cui serviranno 4 bit per rappresentarlo (1 simbolo esadecimale). Si parla di notazione esadecimale, proprio perché nello standard il SIMBOLO 1 è espresso con la notazione esadecimale.
In definitiva, per la rappresentazione del coefficiente AC, si utilizzeranno, per il SIMBOLO 1 8 bit e per il SIMBOLO 2 *Size* bit; per il coefficiente DC invece, si utilizzeranno 4 bit per il SIMBOLO 1 e per il SIMBOLO 2 si utilizzeranno *Size* bit.

Nel paragrafo 5.3, si è effettuata una classificazione delle possibili modalità di funzionamento dello standard jpeg. Per quanto riguarda la modalità sequenziale,

è possibile effettuare un ulteriore classificazione relativa ai processi di codifica che possono essere adoperati. Tale suddivisione deriva direttamente dall'annesso F dello standard usato come riferimento ([49]):

1. approccio baseline con codifica di Huffman;
2. approccio extended con codifica di Huffman, per immagini con profondità di 8 bit;
3. approccio extended con codifica aritmetica, per immagini con profondità di 8 bit;
4. approccio extended con codifica di Huffman, per immagini con profondità di 12 bit;
5. approccio extended con codifica aritmetica, per immagini con profondità di 12 bit;

In questo lavoro di tesi, si è considerato l'approccio baseline con codifica di Huffman.

Il passo successivo consiste nell'applicazione della codifica di Huffman. La *codifica di Huffman* è una codifica che rispetta la regola del prefisso e che permette di ottenere una rappresentazione compatta dei dati; si basa sulla frequenza statistica con cui compare un simbolo in una sequenza: i simboli che compaiono più frequentemente vengono rappresentati con un minor numero di bit rispetto a quelli che compaiono meno frequentemente. Nello standard jpeg si usano delle tabelle che associano al SIMBOLO 1 il relativo codice. Tali tabelle possono essere calcolate in fase di codifica mediante un analisi statistica dei dati. Nello standard sono riportate delle tabelle che sono state sviluppate a partire da un ampio set di immagini con precisione di 8 bit. In particolare sono inserite nello standard quattro diverse tabelle per i due tipi di coefficienti (ΔDC e AC) e per i due tipi di blocchetti (luminanza e crominanza):

- tabella per i coefficienti ΔDC dei blocchetti di luminanza Y;
- tabella per i coefficienti ΔDC dei blocchetti di crominanza C_b e C_r;
- tabella per i coefficienti AC dei blocchetti di luminanza Y;
- tabella per i coefficienti AC dei blocchetti di crominanza C_b e C_r;

In fase di decodifica andranno quindi generate queste tabelle così da associare alle parole di codice il SIMBOLO 1.
Nella tabella 5.7 si riporta la tabella di Huffman presente nello standard per i coefficienti ΔDC dei blocchetti Y.

Category	Code length	Code word
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

Tabella 5.7: Codifica di Huffman proposta nello standard per il SIMBOLO 1 dei coefficienti ΔDC

Come si nota dalla tabella 5.7, sono presenti 12 valori, proprio perché il SIMBOLO 1 ($Size$) del coefficiente ΔDC può assumere 12 possibili valori (si veda la tabella 5.6). Inoltre, le code word di minore lunghezza sono associate ai valori di $Size$ più piccoli e al crescere di $Size$, cresce la Code Length. Questo significa che i simboli con maggiore probabilità, sono i simboli con $Size = 0$ e al crescere del valore $Size$, decresce la frequenza statistica con cui i simboli si presentano e si associa quindi una parola di codice più lunga.

In tabella 5.8 si riportano invece alcuni valori relativi alla codifica di Huffman per i coefficienti AC, sempre considerando i blocchetti di luminanza. Il SIMBOLO 1 dei coefficienti AC è composto dalla coppia $(Runlength, Size)$ in cui $Runlength$ può assumere 16 diversi valori $(0, 1, \ldots, E, F$ in esadecimale) e $Size$ invece 10 diversi valori (non può assumere il valore 0 perché $Size$ rappresenta il numero di bit necessari a rappresentare l'ampiezza del coefficiente AC che è *Amplitude* e che non può essere nullo e non può assumere nemmeno valore 11). In totale avremo quindi una tabella composta da 16x10=160 valori a cui si aggiunge la codifica del simbolo di EOB ($(Runlength, Size) = (0, 0)$), ovvero ci saranno in totale 161 valori. Ecco perché se ne riportano in tabella 5.8 solo alcuni a titolo di esempio. Quindi il SIMBOLO 1 viene rappresentato mediante una parola di codice binaria a lunghezza variabile, ovvero una parola VLC - Variable Length Code - valutata mediante codifica di Huffman; adesso vediamo come si rappresenta il SIMBOLO 2, ovvero il valore di *Amplitude* dei coefficienti ΔDC e AC.
SIMBOLO 2 deve essere convertito in una stringa binaria con un numero di bit pari al valore indicato nel campo $Size$ del SIMBOLO 1 corrispondente. Se il valore di

$(Runlength, Size)$	Code length	Code word
(0,0) (EOB)	4	1010
(0,1)	2	00
(0,A)	16	1111111110000011
(1,2)	5	11011
(1,7)	16	1111111110000101
(1,A)	16	1111111110001000
(2,1)	5	11100
(2,3)	10	1111110111
(3,2)	9	111110111
(3,9)	16	1111111110010100
(4,1)	6	111011
(5,8)	16	1111111110100011
(6,2)	12	111111110110
(7,5)	16	1111111110110000
(8,4)	16	1111111110110111
(9,5)	16	1111111111000001
(B,1)	10	1111111001
(C,8)	16	1111111111011111
(F,0) (ZRL)	11	11111111001
(F,A)	16	1111111111111110

Tabella 5.8: Alcuni valori della codifica di Huffman proposta nello standard per il SIMBOLO 1 dei coefficienti *AC*

SIMBOLO 2 è positivo si utilizza la conversione "classica" da decimale a binario (conversione modulo e segno). Se il valore di SIMBOLO 2 è negativo, si procede nel seguente modo:

a. si sottrae 1 al numero negativo;

b. si considera solo la parte positiva del numero trovato e si trasforma in binario;

c. si applica a questo numero binario il complemento a due (complemento a uno e somma di 1);

d. sul risultato ottenuto, si predono, a partire da destra, un numero di cifre binarie pari a $Size$. Il SIMBOLO 2 è quindi codificato mediante un intero di lunghezza variabile (Variable Length Integer, VLI).

Per comprendere meglio il processo di codifica appena descritto, considero la stringa di simboli ottenuta nell'esempio fatto in precedenza, che equivale alla rappresentazione del blocchetto 8x8 presente in tabella 5.1:

$$\underbrace{(2)(3)}_{1}, \underbrace{(1,2)(-2)}_{2}, \underbrace{(0,1)(-1)}_{3}, \underbrace{(0,1)(-1)}_{3}, \underbrace{(0,1)(-1)}_{3}, \underbrace{(2,1)(-1)}_{4}, \underbrace{(0,0)}_{5} \quad (5.15)$$

Considero che le tabelle di Huffam siano quelle 5.7 (per il coefficiente DC) e 5.8 (per i coefficienti AC). Analizziamo i singoli passi del processo di codifica per la stringa espressa nella 5.15; i numeri che compaiono sotto le coppie di simboli, sono legati ai numeri del seguente elenco numerato:

1. ΔDC: (2)(3)
   SIMBOLO 1=(2) $\xrightarrow[Huffman]{}$ **011**
   SIMBOLO 2=(3) $\xrightarrow[binario]{}$ **11** (rappresentazione con Size=2 bit);

2. AC: (1,2)(-2)
   SIMBOLO 1=(1,2) $\xrightarrow[Huffman]{}$ **11011**
   SIMBOLO 2=(-2) $\underset{a}{\rightarrow}$ -3 $\underset{b}{\rightarrow}$ 11 $\underset{c}{\rightarrow}$ 01 $\underset{d}{\rightarrow}$ **01** (rappresentazione con Size=2 bit);

3. AC: (0,1)(-1)
   SIMBOLO 1=(0,1) $\xrightarrow[Huffman]{}$ **00**
   SIMBOLO 2=(-1) $\underset{a}{\rightarrow}$ -2 $\underset{b}{\rightarrow}$ 10 $\underset{c}{\rightarrow}$ 10 $\underset{d}{\rightarrow}$ **0** (rappresentazione con Size=1 bit);

4. *AC*: (2,1)(-1)
   SIMBOLO 1=(2,1) $\xrightarrow[Huffman]{}$ **11100**
   SIMBOLO 2=(-1) $\underset{a}{\rightarrow}$ -2 $\underset{b}{\rightarrow}$ 10 $\underset{c}{\rightarrow}$ 10 $\underset{d}{\rightarrow}$ **0** (rappresentazione con Size=1 bit);

5. *AC*: (0,0)
   SIMBOLO 1=(0,0) $\xrightarrow[Huffman]{}$ **1010**

In definitiva la stringa binaria rappresentativa del blocchetto 8x8 è la seguente:

$$011\ 11 \quad 11011\ 01 \quad 00\ 0 \quad 00\ 0 \quad 00\ 0 \quad 11100\ 0 \quad 1010 \tag{5.16}$$

Considerando la suddivisione in byte:

$$\underbrace{01111110}_{Byte\,1} \quad \underbrace{11010000}_{Byte\,2} \quad \underbrace{00000111}_{Byte\,3} \quad \underbrace{0001010*}_{Byte\,4} \tag{5.17}$$

In conclusione, con 31 bit si sono rappresentati i 64 coefficienti caratteristici di un blocchetto 8x8 di luminanza di un immagine. Si è ottenuta questa altissima compressione anche in virtù del fatto che nel blocchetto originario, i valori risultano molto correlati fra loro (si veda la tabella 5.1). L'operazione appena descritta va eseguita per tutti i blocchetti dell'immagine di ognuna delle componenti presenti, secondo l'ordinamento descritto nella sezione 5.3.1.1. Infine, avremo una sequenza di byte a cui bisogna applicare ancora due operazioni:

1. *Byte Stuffing*: Si scandisce la sequenza byte a byte e ogni volta che si trova un byte che rappresenta il numero esadecimale $0xFF$, ovvero il byte composto da tutti 1, si inserisce un byte subito dopo composto da tutti 0 ($0x00$ in esadecimale). Infatti, come si vedrà nella sezione 5.3.1.5, il byte $0xFF$ è un byte che riveste un importanza fondamentale nel processo di scrittura del file.

2. Se alla fine della stringa, il byte è incompleto, questo verrà completato inserendo dei bit '1' così da completare l'ultimo byte.

Per quanto detto, risulta evidente, che per valutare esattamente l'occupazione in byte di un file jpeg, occorre necessariamente codificare l'immagine per intero.

5.3.1.5 Scrittura e Struttura del File

Il file jpeg, è costituito da una serie di segmenti che iniziano con uno specifico *marker*. Ogni marker è composto da una coppia di byte $0xFF\,0x**$, in cui il byte successivo a $0xFF$, ovvero il secondo byte, non può essere $0x00$. Infatti nella sezione precedente (5.3.1.4), si è detto che la coppia di byte $0xFF\,0x00$, è rappresentativa dei dati. In sostanza, in fase di decodifica, se si incontra un byte $0xFF$, si verifica qual'è il successivo, se è diverso da $0x00$, siamo in presenza di un marker, se invece è pari a $0x00$ si elimina il byte (procedura byte stuffing inversa). Esistono 64 diversi marker, che identificano altrettanti segmenti, e 190 marker che identificano segmenti riservati. Nel seguito verranno descritti solo i segmenti di interesse (JFIF in modalità Baseline descritti in [44]) che sono riassunti nella tabella 5.9.

Marker (Hex $0x$)	Simbolo	Nome del Marker
$FF\,D8$	SOI	Start Of Image
$FF\,E0$	APP_0	Application Segment 0
$FF\,DB$	DQT	Define Quantization Table
$FF\,C0$	SOF_0	Start Of Frame 0
$FF\,C4$	DHT	Define Huffman Table
$FF\,DA$	SOS	Start Of Scan
$FF\,D9$	EOI	End Of Image

Tabella 5.9: Marker di interesse in JFIF nella modalità Baseline

I marker SOI e EOI delimitano il file. In ogni file jpeg, con qualsiasi modalità di codifica adottata, i primi due byte sono $0xFF\,0xD8$ e gli ultimi due sono $0xFF\,0xD9$.

Osservando la tabella 5.9, si nota che i segmenti "Application" e "Start Of Frame", sono identificati da uno '0' a pedice. In sostanza, APP_0 indica che il file rispetta il formato Jfif e SOF_0 indica che la codifica è effettuata nella modalità Baseline.
Per quanto riguarda il segmento **Start of Frame**, questo include alcuni dati supplementari che sono necessari nel processo di decodifica e tali dati sono dipendenti dalle modalità con cui è stato codificato il file. Come detto in precedenza, in questo lavoro di tesi si è affrontata la modalità Baseline e in tal caso i dati di interesse sono incapsulati nel segmento caratterizzato da marker SOF_0. La modalità Baseline è quella maggiormente utilizzata, impiegata anche in Iphone e nella quasi totalità delle fotocamere digitali. In tabella 5.10 si elencano i marker e i simboli relativi alle possibili modalità di codifica.

Per quanto riguarda invece l'**Application Segment** ne esistono di 16 tipi (marker $0xFF\,0xE0 \div 0xFF\,0xEF$). Il formato **Jfif** è il formato "base" ed è appunto

Marker (Hex)	Simbolo	Modalità, Tipo di codifica
$FFC0$	SOF_0	Baseline DCT (sequential)
$FFC1$	SOF_1	Extended sequential DCT, Huffman
$FFC2$	SOF_2	Progressive DCT, Huffman
$FFC3$	SOF_3	Lossless (sequential), Huffman
$FFC5$	SOF_5	Differential sequential DCT, Huffman
$FFC6$	SOF_6	Differential progressive DCT, Huffman
$FFC7$	SOF_7	Differential lossless, Huffman
$FFC9$	SOF_9	Extended Sequential DCT, Aritmetic
$FFCA$	SOF_{10}	Progressive DCT, Aritmetic
$FFCB$	SOF_{11}	Lossless (sequential), Aritmetic
$FFCD$	SOF_{13}	Differential sequential DCT, Aritmetic
$FFCE$	SOF_{14}	Differential progressive DCT, Aritmetic
$FFCF$	SOF_{15}	Differential lossless (sequential), Aritmetic

Tabella 5.10: Marker e relativi segmenti di interesse nel file JFIF nella modalità Baseline

caratterizzato dal segmento Application con marker $0xFF\,0xE0$, inserito subito dopo il SOI. I file nel formato Jfif possono essere correttamente decodificati rispettando le direttive fornite nello standard [44] e in Hamilton [46]. Quando si parla di file jpeg incapsulato secondo il formato Jfif, ci si riferisce ad un insieme di dati che forniscono delle informazioni supplementari, rispetto a quelle fornite nel processo di codifica "standard" jpeg, che consentono di ottenere una completa compatibilità fra le varie piattaforme e applicazioni. Inoltre, con il Jfif sono fornite delle direttive, non incapsulate direttamente nel bitstream, ma che consentono la corretta decodifica del file. Infatti, in Jfif sono specificati un insieme di informazioni che non sono fornite direttamente nello standard jpeg. Per fare un esempio pratico, nello standard jpeg non si parla di spazio di colore o di trasformazioni, queste informazioni sono invece descritte in Jfif. Infatti, la trasformazione espressa nell'equazione 5.1, è stata estratta dall'articolo di riferimento sul Jfif (Hamilton [46]). Inoltre in Jpeg, non sono definite le modalità con cui si effettua il sottocampionamento, che sono invece specificate in Jfif. Il formato Jfif è utilizzato in gran parte dei programmi per PC, ovvero quando viene creata o rielaborata un immagine e poi salvata in jpeg, generalmente si fa uso del formato di file Jfif.

Altro formato di file, molto diffuso è **Exif** - Exchangeable image file format. Le direttive di questo formato sono descritte nello standard ISO-12234. Lo standard Exif, oltre a specificare il formato per alcuni tipi di immagini (Jpeg, Tiff rev.6.0), stabilisce il formato anche per alcuni formati audio; si veda a tal proposito Camera e Association [51]. Attualmente la quasi totalità dei DSC, compresi gli smartphone, utilizzano Exif. Anche iPhone utilizza il formato Exif. Valgono delle considerazioni

analoghe a quelle fatte in precedenza per il Jfif; vediamo ora nel dettaglio alcune caratteristiche di di Exif:

- Dati supplementari sono inseriti nel segmento APP_1.
- É stato creato dalla JEIDA - Japan Electronic Development Association. L'ultima versione è la 2.3 datata Aprile 2010.
- Alcuni dei dati supplementari che possono essere inseriti nel segmento APP_1 sono: data e ora, marca e modello del dispositivo, lunghezza focale, scala e profilo del colore, presenza del canale alpha, velocità dell'otturatore, velocità ISO, tempo di esposizione, posizione all'istante dello scatto (estratta dai dati del GPS), informazioni sul copyright, commenti. Esistono comunque molti altri campi.
- Esistono dei software in grado di estrarre i dati contenuti nel segmento APP_1. Ad esempio il software *ExifReader* per Windows, mentre su Mac OS parte dei dati Exif possono essere visualizzati direttamente (click con il tasto destro e "Ottieni Informazioni" sul file jpeg).
- Vediamo come si comportano i programmi di elaborazione delle immagini salvando delle immagini che erano state acquisite e memorizzate mediante fotocamera digitale (jpeg/Exif), i casi sono due:
  - Il programma preserva i dati Exif: in questo caso l'immagine è memorizzata in jfif, ma i dati supplem sono reincapsulati nel file.
  - Il programma non conserva i dati supplementari, che verranno quindi persi. Questo è il caso dei programmi più obsoleti.

Passiamo all'analisi dettagliata dei segmenti di interesse, riassunti in tabella 5.9.

SEGMENTO APPLICATION 0 - $0xFF\,0xE0$

Occupazione	**Parametri**	**Descrizione**
2 Byte	APP_0	Application Marker ($0xFF\,0xE0$)
2 Byte	$length$	Lunghezza Application Segment (marker escluso)
5 Byte	$identifier$	Stringa Jfif: $identifier = 0x4A\,, 0x46\,, 0x49\,, 0x46\,, 0x00$
2 Byte	$version$	Numero identificativo della versione: se la versione è la 1.02 $\rightarrow$ $version = 0x01\,0x02$;
1 Byte	$units$	Unità per le densità X e Y: $units = 0x00 \rightarrow$ nessuna unità, X e Y specificano il rapporto d'aspetto; $units = 0x01 \rightarrow$ valori X e Y relativi a punti per inch; $units = 0x02 \rightarrow$ valori X e Y relativi a punti per cm;
2 Byte	$Xdensity$	densità orizzontale in pixel; $Xdensity = 0x00\,, 0x01 \rightarrow$ non specificato;
2 Byte	$Ydensity$	densità verticale in pixel;
1 Byte	$Xthumbnail$	Quantità di pixel orizzontali nella miniatura;
1 Byte	$Ythumbnail$	Quantità di pixel verticali nella miniatura;
$3\,n$ Byte	$(RGB)_n$	Valori 24 bit RGB per la miniatura (8 *bpp*); $n = Xthumbnail * Ythumbnail$

Tabella 5.11: Struttura dell' Application Segment 0

Il segmento Application è situato all'inizio del file, subito dopo lo Start Of Frame. Le informazioni relative al segmento Application non sono contenute nello standard, bensì nelle specifiche del formato di file, nel caso in esame, la tabella 5.11 è relativa al jfif (riferimento Hamilton [46]). Tramite la lettura di questo segmento è possibile avere, in maniera diretta, delle informazioni sull'immagine senza doverla necessariamente decodificare per intero. Se $units = 0x00$, la coppia $(Xdensity, Ydensity)$ rappresenta il rapporto d'aspetto, altrimenti rappresenta la densità di pixel per cm o per inch. Aspetto di fondamentale importanza è la possibilità di inserire all'interno dell'application segment la **miniatura** (thumbnail) dell'immagine. La miniatura è una versione a bassa risoluzione dell'immagine originale che è usata nei dispositivi fotografici e nei Pc cosicché, visualizzando tante foto contemporaneamente, è possibile decodificare solo una piccola parte del file, riducendo quindi il costo computazionale. In figura 5.9 è mostrata la visualizzazione multipla , mediante thumbnail, delle foto

in iPhone 4 sfruttando le miniature.

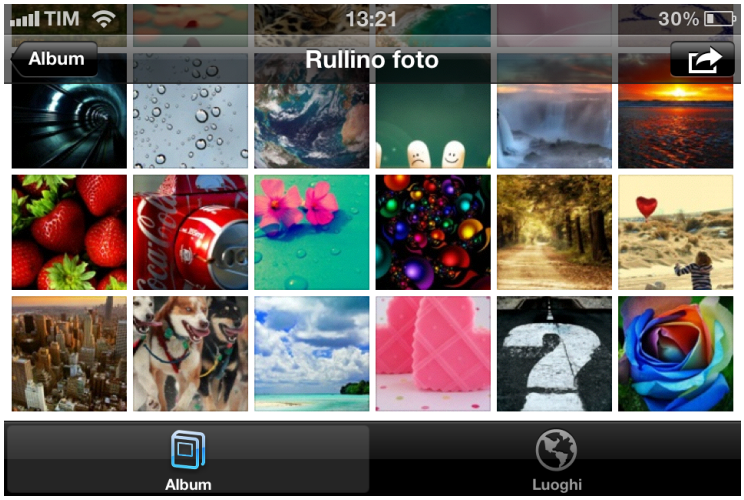


Figura 5.9: Visualizzazione multipla delle immagini in iPhone 4 utilizzando le miniature

Le variabili $Xthumbnail$ e $Ythumbnail$ rappresentano le dimensioni in pixel della miniatura. In $(RGB)_n$ si inseriscono i valori RGB di ogni pixel della miniatura; per un immagine a colori ogni pixel è descritto da una terna di byte, proprio perché si considera una profondità di 8 bpp (standard della modalità Baseline). Ad esempio, per una miniatura con $(Xthumbnail, Ythumbnail) = (160, 120)$, si ha un occupazione totale di $160 * 120 * 3Byte = 57,6\,KByte$, che è un occupazione abbastanza elevata (in relazione alla dimensione del file), proprio perché non viene effettuata nessuna compressione dei dati. Ecco perché è stata definita un estensione dell'Application Segment APP_0 che prevede la presenza di campi supplementari, oltre a quelli elencati in tabella 5.11. Questi sono i campi *extension code* (1 byte) e *extension data* (dimensione variabile). Nel campo *extension data* si inseriscono i valori dei pixel RGB della miniatura codificati jpeg. In pratica, sarà presente una vera e propria immagine jpeg, di dimensione ridotta, costituita da tutti i vari segmenti caratteristici di un immagine jpeg e presenti nella tabella 5.9. *Anche nel formato Exif è previsto l'uso della miniatura; in questo lavoro di tesi si è utilizzata la miniatura presente all'inizio del file incapsulata rispettando le direttive Jfif.* La struttura appena descritta, per un file Exif, è rappresentata in figura 5.10.

Altra considerazione va fatta per il parametro $identifier$, che è una stringa di 5 byte, che identifica il Jfif, ovvero è la stringa Jfif terminata con 0. Nel caso di formato di file Exif tale stringa vale invece $0x45\,, 0x78\,, 0x69\,, 0x66\,, 0x00$.

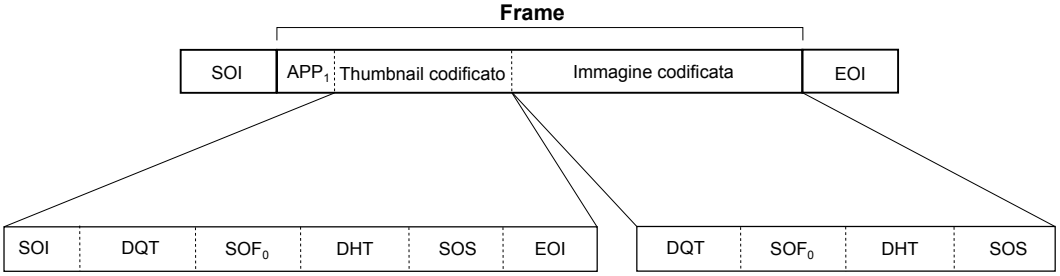


Figura 5.10: Struttura in un file Exif con miniatura

SEGMENTO MATRICE DI QUANTIZZAZIONE - $0xFF\,DB$

Occupazione	Parametri	Descrizione
2 Byte	DQT	Define Quantization Table ($0xFF\,,0xDB$)
2 Byte	L_q	Lunghezza del segmento in Byte (marker escluso). Nella modalità Baseline vale 67 Byte, ovvero $L_q = 0x00\,,0x84$
4 bit	P_q	Precisione degli elementi della matrice di quantizzazione. $P_q = 0000 \rightarrow$ precisione a 8 bit; $P_q = 0001 \rightarrow$ precisione a 16 bit; Nella modalità Baseline, $P_q = 0000$.
4 bit	T_q	Parametro identificativo della matrice di quantizzazione; varia tra 0 e 3 per cui: $T_q = 0000$ or $T_q = 0001$ or $T_q = 0010$ or $T_q = 0011$.
64 Byte	Q_k	Sono i 64 elementi della matrice di quantizzazione ($k = 1, 2 \ldots 64$), 1 Byte per ogni elemento. La precisione di ogni elemento è sempre a 8 bit: i valori sono compresi tra 1 e 255. Il valore 0 per il singolo elemento non è ammesso.

Tabella 5.12: Struttura del segmento Matrice di Quantizzazione

Il segmento matrice di quantizzazione specifica le matrici 8x8 che devono essere usate nel processo di quantizzazione. Ogni matrice, specificata nel bitstream codificato ha un segmento proprio e le viene assegnato un identificatore (variabile T_q). Si possono avere al massimo quattro matrici di quantizzazione distinte nello stesso file, anche se in genere ne vengono utilizzate due, una per la quantizzazione della componente di luminanza ($T_q = 0$) e una per le componenti di crominanza ($T_q = 1$). Quindi, per un immagine a colori, ci saranno in genere due segmenti DQT adiacenti che sono posti dopo il segmento Start Of Image. Gli elementi della matrice di

quantizzazione non possono assumere valore nullo, per quanto detto nel paragrafo 5.3.1.3. I dati di queste matrici sono poi ordinati a zig-zag, analogamente ai blocchi DCT (si veda la figura 5.8).

SEGMENTO FRAME HEADER - $0xFFC0$

Occupazione	Parametri	Descrizione
2 Byte	SOF_0	Start Of Frame $(0xFF, 0xC0)$ - Baseline DCT
2 Byte	L_f	Lunghezza del segmento in Byte (marker escluso). $L_f = 8 + 3 * N_f$
1 Byte	P	Precisione in bit; nella modalità Baseline è pari a 8 bit $\rightarrow P = 0x08$.
2 Byte	Y	Numero di linee dell'immagine. Il valore massimo consentito è quindi $2^{16} = 65536$.
1 Byte	X	Numero di campioni per linea. Il valore massimo consentito è quindi $2^{16} = 65536$.
1 Byte	N_f	Numero di componenti dell'immagine; caso $(Y, C_b, C_r) \rightarrow N_f = 0x03$; i campi sottostanti (C_i, H_i, V_i) si ripetono N_f volte.
1 Byte	C_i	Identificatore della componente $(C_i = 0 \div 255)$; di solito risulta: componente $Y \rightarrow C_i = 0x01$; componente $C_b \rightarrow C_i = 0x02$; componente $C_r \rightarrow C_i = 0x03$.
4 bit	H_i	Fattore orizzontale di campionamento.
4 bit	V_i	Fattore verticale di campionamento.
1 Byte	Tq_i	Identificatore della matrice di quantizzazione; $Tq_i = (0 \div 3)$.

Tabella 5.13: Struttura del segmento Frame Header

Nel segmento Frame Header, sono incapsulate delle informazioni relative all'immagine e al campionamento effettuato.
Nel caso tipico di immagine a colori, si avrà $N_f = 3$, per cui la sequenza dei parametri del segmento Frame Header, sarà la seguente:

$$SOF_0\,, L_f\,, P\,, Y\,, X\,, N_f\,, C_i\,, H_i\,, V_i\,, Tq_i\,, H_i\,, V_i\,, Tq_i\,, H_i\,, V_i\,, Tq_i \tag{5.18}$$

Il parametro Tq_i è direttamente collegato al parametro T_q del segmento matrice

di quantizzazione, per cui sulla componente C_i, va "applicata" la matrice di quantizzazione $Tq_i = T_q$. Nel segmento frame header vengono fornite le dimensioni in pixel dell'immagine; attraverso queste dimensioni è possibile operare, in fase di decodifica, l'operazione inversa del completamento dei blocchetto effettuata in fase di codifica (si veda figura 5.4).

SEGMENTO CODICI DI HUFFMAN - $0xFFC4$

Occupazione	Parametri	Descrizione
2 Byte	DHT	Define Huffman Table ($0xFF, 0xC4$).
2 Byte	L_h	Lunghezza del segmento in Byte (marker escluso); calcolabile come: $2 + \sum_{t=1}^{n}(17 + m_t)$ con $m_t = \sum_{i=0}^{16} L_i$.
4 bit	T_c	Identificatore della tabella; $T_c = 0000 \rightarrow$ tabella DC; $T_c = 0001 \rightarrow$ tabella AC
4 bit	T_h	Identificatore della tabella di destinazione di Huffman; $T_h = 0000 \rightarrow$ componente di luminanza; $T_h = 0001 \rightarrow$ componenti di crominanza.
16 Byte	L_i	Numero di codici di Huffman di lunghezza i.
(L_h- 19) Byte	V_{ij}	Valori associati ai codici di Huffman.

Tabella 5.14: Struttura del segmento Codici di Huffman

Il segmento Codici di Huffman, viene prodotto in fase di codifica in base ad un analisi statistica dei dati da codificare; in seguito a tale analisi, si assegna una code word costituita da un ridotto numero di bit per i simboli che compaiono più frequentemente, e via via la lunghezza delle code word cresce con il decrescere della frequenza statistica con cui compaiono i simboli nella sequenza da codificare. In questo lavoro di tesi, siamo interessati alla fase di decodifica e per questo motivo, verranno descritte le modalità che permettono di ottenere le tabelle di Huffman e in seguito risalire al simbolo. Nel seguito quindi si descriveranno, a livello concettuale, le procedure da eseguire in fase di decodifica, per poi completare la trattazione nel paragrafo 5.3.2, in cui si introdurrà un esempio pratico.

Il parametro L_i è una stringa di 16 byte, il cui i-esimo byte, rappresenta il numero di codici di lunghezza i presenti. L_i è costituito da 16 byte proprio perché la lunghezza delle code word può variare tra 1 e 16 bit. Così ad esempio, se i primi tre byte dei sedici totali, sono $0x01$, $0x08$, $0x2A$, significa che è necessario produrre, in

fase di decodifica, *1* parola di lunghezza uno, *8* parole di lunghezza due , *42* parole di lunghezza tre. Con il parametro L_i, viene quindi fornita l'informazione relativa alla frequenza statistica e alla lunghezza delle codeword di Huffman da generare. A queste codeword, saranno associati i relativi valori forniti dal parametro V_{ij}. Questo parametro ha dimensione pari a 1 byte per ogni code word. Vediamo il significato pratico di questo parametro:

- Nel caso di coefficiente ΔDC, V_{ij} rappresenta il numero di bit successivi alla codeword di Huffman da prendere (ovvero $Size$) tra i dati. Presi questi $Size$ bit, si decodificano, ottenendo il valore di ampiezza del coefficiente ΔDC (*Amplitude*). Si vedano a tal proposito le tabelle 5.6 e 5.7. La decodifica deve avvenire con un processo inverso, rispetto a quello descritto nella sezione 5.3.1.4.

- Nel caso di coefficiente AC, i primi 4 bit del byte V_{ij}, rappresentano il numero di bit da prendere dei dati successivi alla codeword di Huffman. Presi questi bit, si decodificano, ottenendo il valore di *Runlength* (lunghezza della corsa di zeri). Gli ultimi 4 bit del byte V_{ij} rappresentano il numero di bit da prendere, in seguito ai bit relativi a *Runlength*. La decodifica di questi bit è l'*Amplitude* del coefficiente AC dopo la corsa di zeri.

Quanto appena detto troverà completezza nel paragrafo 5.3.2, in cui verrà fatto un esempio pratico di utilizzo dei segmenti dei codici di Huffman.

L'unione dei due nibble T_c e T_h, può produrre quattro differenti valori:

1. T_c *unione* $Th = 0x00 \rightarrow$ tabelle di Huffman per coefficienti ΔDC di luminanza;

2. T_c *unione* $Th = 0x01 \rightarrow$ tabelle di Huffman per coefficienti ΔDC di crominanza;

3. T_c *unione* $Th = 0x10 \rightarrow$ tabelle di Huffman per coefficienti AC di luminanza;

4. T_c *unione* $Th = 0x11 \rightarrow$ tabelle di Huffman per coefficienti AC di crominanza.

Infatti, per le immagini con tre componenti, saranno presenti 4 segmenti dei codici di Huffman. Per immagini monocromatiche invece, i segmenti dei codici di Huffman saranno due (non ci saranno tabelle di Huffman per le componenti cromatiche).

SEGMENTO SCAN HEADER - $0xFF\,DA$

Occupazione	Parametri	Descrizione
2 Byte	SOS	Start Of Scan ($0xFF\,,0xDA$).
2 Byte	L_s	Lunghezza del segmento in Byte (marker escluso); calcolabile come: $6 + 2\,N_s$
1 Byte	N_s	Numero di componenti dell'immagine nello scan; I campi Cs_j,Td_j e Ta_j sono ripetuti N_s volte.
1 Byte	Cs_j	Identificatore della componente dello scan ($Cs_j = 0 \div 255$).
4 bit	Td_j	Codici di Huffman relativi alle componenti DC. ($Td_j = 0, 1$)
4 bit	Ta_j	Codici di Huffman relativi alle componenti AC. ($Td_j = 0, 1$)
1 Byte	Ss	Inizio selezione spettrale. Nella modalità Baseline Ss=0.
1 Byte	Se	Fine selezione spettrale. Nella modalità Baseline Se=63 (0x3F).
4 bit	Ah	Approssimazioni successive sulla posizione del "bit higth". Nella modalità Baseline Ah=0.
4 bit	Al	Approssimazioni successive sulla posizione del "bit low". Nella modalità Baseline Al=0.

Tabella 5.15: Struttura del segmento Scan Header

I dati contenuti nel segmento Scan Header forniscono le informazioni aggiuntive necessarie all'applicazione dei codici di Huffman alle diverse componenti. Nel caso tipico, in cui $N_s = 3$, la disposizione dei diversi parametri nel segmento è la seguente:

$$SOS, L_s, N_s, Cs_j, Td_j, Ta_j, Cs_j, Td_j, Ta_j, Cs_j, Td_j, Ta_j, Ss, Se, Ah, Al$$

Il parametro Cs_j è legato al parametro C_i del segmento Frame Header (si veda 5.13) così da "legare" alla componente identificata dal parametro $C_i = Cs_j$ i relativi parametri dei segmenti Frame Header (campionamento) e Codici di Huffman (tabelle).
I parametri Td_j e Ta_j sono legati ai parametri T_c e T_h del segmento Codici di Huffman (si veda 5.14). Per cui, alla componente dell'immagine con parametri Cs_j e Td_j, si applicano le tabelle di Huffman identificate con la coppia di parametri T_c e T_h.

Dopo il segmento Scan Header, vengono scritti i dati veri e propri dell'immagine codificati secondo le procedure descritte in precedenza.

5.3.2 Decodifica Jpeg

Nei paragrafi precedenti, sono state descritte le modalità di codifica del file jpeg sottolineando gli aspetti di interesse da un punto di vista teorico e fornendo degli esempi numerici. In questa sezione, verrà invece descritto in maniera sommaria, il processo di decodifica del file jpeg da un punto di vista realizzativo, a partire dall'effettivo file prodotto dal dispositivo di interesse, ovvero dall'iPhone 4. Maggiori informazioni sulla realizzazione del codice sono presenti nei commenti dello script in Appendice B in cui è inserito, in un unico script, il codice relativo alla decodifica del file jpeg e alla decodifica dello SPinV Code.

Il processo di decodifica jpeg è sempre relativo al formato di file Jfif e alla modalità Baseline di jpeg, infatti come detto in precedenza, in questo lavoro di tesi si è sfruttata la miniatura presente all'inizio del bitstream (si veda 5.10) che rispetta appunto le direttive Jfif in modalità Baseline.
Il processo di decodifica ripercorre inversamente le operazioni elencate in figura 5.1. Lo schema a blocchi è mostrato in figura 5.11:

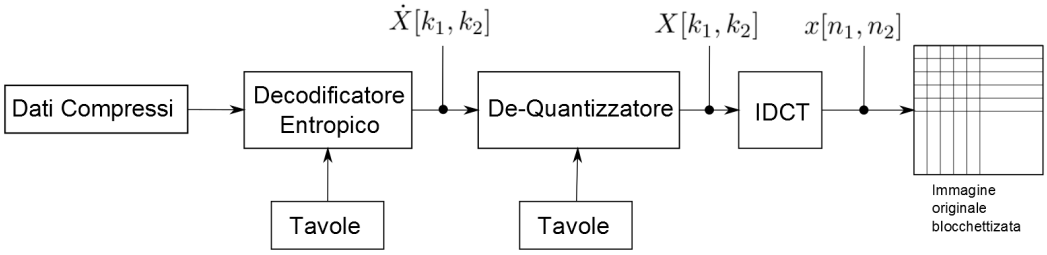


Figura 5.11: Schema a blocchi concettuale del decodificatore jpeg

I test di decodifica sono stati effettuati con *Xcode versione 3.2.2* su sistema operativo *Mac OS X versione 10.6.8*. Per la visualizzazione dei risultati, si è utilizzata la *Debugger Consolle*. Nel seguito, per illustrare la decodifica jpeg, si prenderà in considerazione una semplice immagine jpeg di 16x16 pixel (SedicixSedici.jpeg); in figura 5.12 è presente il contenuto in byte di questo file, letto e visualizzato mediante un editor esadecimale. In particolare, si è usato l'editor Hex Fiend. In figura 5.12, si sono evidenziati i diversi segmenti, e cerchiati con un rettangolo rosso i relativi marker. Dalla figura si nota come nel segmento Application (marker $0xFF$,$0xEO$) sia presente la stringa $identifier = 0x4A\,, 0x46\,, 0x49\,, 0x46\,, 0x00$ caratteristica del

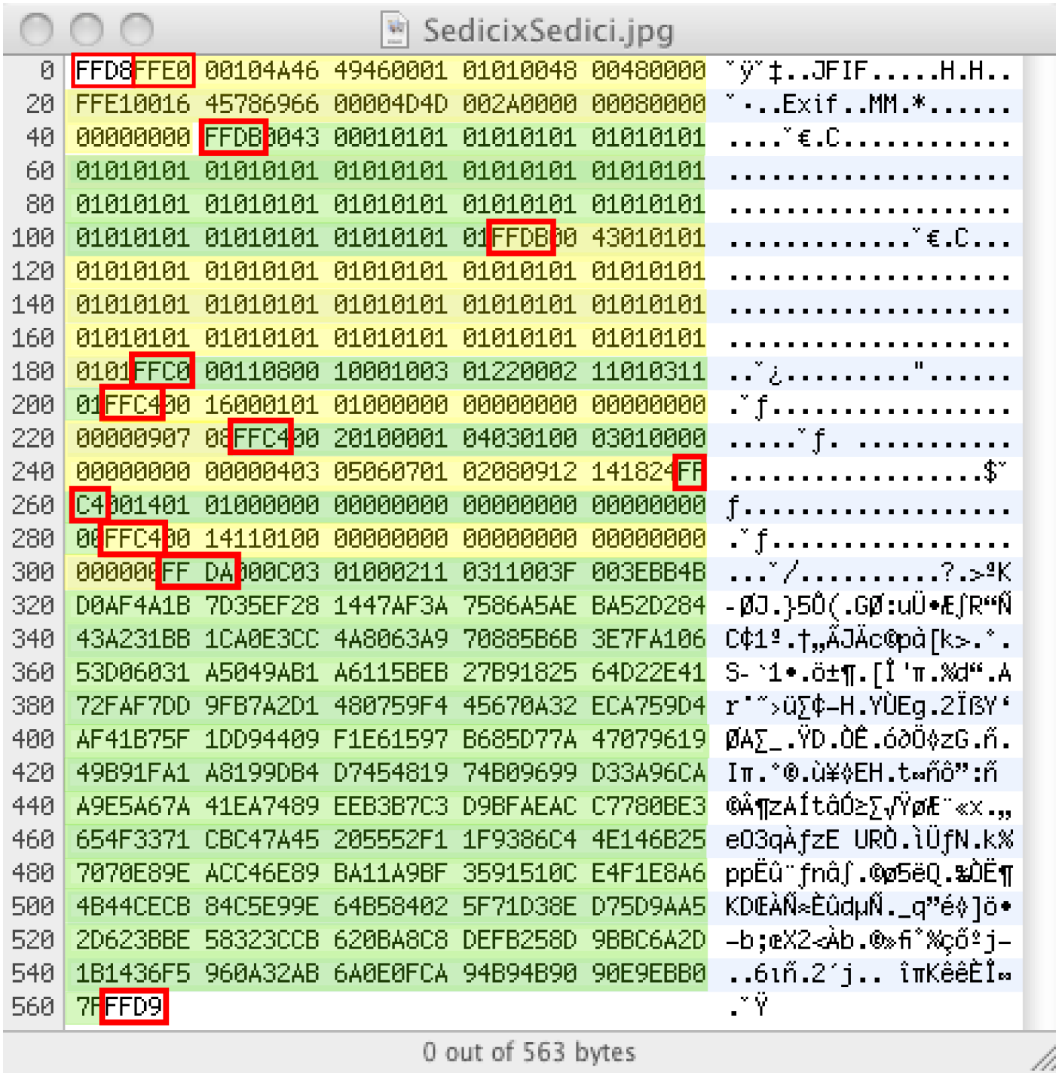


Figura 5.12: Struttura di un file jpeg - visualizzazione mediante hex-fiend

formato Jfif, inoltre è presente anche il marker $0xFF\,0xE1$, che contiene altri dati relativi al formato di file Exif. I vari segmenti sono disposti rispettando l'ordinamento descritto nella sezione 5.3.1.5. É importante sottolineare, che l'ordinamento dei diversi segmenti può anche essere diverso. Ad esempio in alcuni file jpeg, pur essendo presenti più matrici di quantizzazione, è presente un solo marker $0xFF\,0xDB$, ovvero le diverse matrici di quantizzazione non sono "delimitate" dal relativo marker; altro esempio è che il segmento 'Codici di Huffman', può trovarsi prima del segmento 'Frame Header'. Lo script prevede queste eventualità.

Il Primo passo è caricare il file jpeg nello script. Il codice, sulla base della lettura del SOI e dell'EOI, è in grado di determinare se il file caricato è un file jpeg. Inoltre, è possibile risalire alla dimensione in byte del file e all'eventuale presenza della miniatura. In figura 5.13 è inserito il risultato visualizzato nella debugger consolle. Il codice presente in Appendice B, è privo, almeno nella parte legata alla decodifica jpeg, delle istruzioni *printf* per la stampa su schermo, per cui i risultati che verranno mostrati nel seguito, sono relativi ad un altro script che esegue la sola decodifica del file jpeg e che comprende le *printf*. Infatti, come detto all'inizio di questa sezione, il codice in Appendice B effettua sia la decodifica jpeg che la decodifica dello SPinV Code.

```
File Caricato Correttamente

Dimensione del file Caricato:563 Byte
Il primo Byte del file Caricato è:ff        Il secondo Byte del file Caricato è:d8
Il penultimo Byte del file Caricato è:ff    L'ultimo Byte del file Caricato è:d9
--> Il file Caricato è un .jpg

*** Nel file Non è presente la miniatura ---> immagine=1(no miniatura) ***
```

Figura 5.13: Debugger Consolle: dati iniziali

Dopo l'operazione iniziale descritta, verranno letti ed analizzati i segmenti relativi alla matrice di quantizzazione, caratterizzati da marker $0xFF, 0xDB$. Nel caso in esame (si veda figura 5.14), sono presenti due matrici di quantizzazione, una per la componente di luminanza e una per entrambe le componenti di crominanza. La presenza di una terza matrice di quantizzazione, implica che le due componenti di crominanza utilizzino matrici distinte. La quarta matrice è invece relativa alla quantizzazione dell'eventuale canale alpha che è un canale aggiuntivo che descrive il grado di trasparenza/opacità di ogni pixel nell'immagine. Dalla figura 5.14 si vede come non sia stata effettuata nessuna quantizzazione dato che tutti gli elementi di entrambe le matrici di quantizzazione presentano valore unitario.

Il passo successivo è la decodifica del segmento 'Frame Header', caratterizzato da marker $0xFF, 0xC0$, che contiene alcune informazioni relative al campionamento effettuato. I risultati della lettura e decodifica di questo segmento, nel file SedicixSedici.jpeg, sono mostrati in figura 5.15.

```
*** SEGMENTO MATRICE DI QUANTIZZAZIONE 0--> Marker FF DB ***

Inizio al byte numero 44 del file caricato
Lunghezza Segmento Matrice Quantizzazione Lq=67 byte
Pq=0 -->La precisione degli elementi della matrice di quantizzazione è a 8 bit
Tq=0 --> L'identificatore della matrice di quantizzazione è 0
Vettore Matrice di Quantizzazione di Luminanza Y ordinato (Non a Zig-Zag)
       1       1       1       1       1       1       1       1
       1       1       1       1       1       1       1       1
       1       1       1       1       1       1       1       1
       1       1       1       1       1       1       1       1
       1       1       1       1       1       1       1       1
       1       1       1       1       1       1       1       1
       1       1       1       1       1       1       1       1
       1       1       1       1       1       1       1       1



*** SEGMENTO MATRICE DI QUANTIZZAZIONE 1--> Marker FF DB ***

Inizio al byte numero 113 del file caricato
Lunghezza Segmento Matrice Quantizzazione Lq=67 byte
Pq=0 -->La precisione degli elementi della matrice di quantizzazione è a 8 bit
Tq=1 --> L'identificatore della matrice di quantizzazione è 1
Vettore Matrice di Quantizzazione di Crominanza Cr ordinato (Non a Zig-Zag)
       1       1       1       1       1       1       1       1
       1       1       1       1       1       1       1       1
       1       1       1       1       1       1       1       1
       1       1       1       1       1       1       1       1
       1       1       1       1       1       1       1       1
       1       1       1       1       1       1       1       1
       1       1       1       1       1       1       1       1
       1       1       1       1       1       1       1       1



Nel file Caricato sono specificate le seguenti matrici di Quantizzazione:
Matrice di quantizzazione della Luminanza Y
Matrice (Unica) di quantizzazione delle componenti di Crominanza Cb e Cr
```

Figura 5.14: Debugger Consolle: matrici di quantizzazione

```
*** SEGMENTO FRAME HEADER --> Marker FF C0 ***

Inizio alla posizione:182 -> ff c0

Lunghezza Segmento Frame Header Lq=17 byte
L'immagine ha una risoluzione:        16x16
Le componenti dell'immagine sono:     3
Identificatore delle componenti:              1       2       3
Fattore Orizzontale di compressione:          2       1       1
Fattore Verticale di Compressione:            2       1       1
Identificatore della matrice di quant.        0       1       1
8+3*Nf==Lf --> Verifica Segmento Scan Header Riuscita
```

Figura 5.15: Debugger Consolle: segmento Frame Header

Dopo il segmento Frame Header, sono presenti i quattro segmenti relativi ai codici di Huffman. Sulla base della coppia di parametri (T_c, T_h), si capisce se i parametri L_i e V_{ij} sono relativi ai coefficienti ΔDC o ai coefficienti AC e inoltre se tali coefficienti sono legati alla componente di luminanza o di crominanza. In figura 5.16 si riportano i risultati ottenuti per gli elementi AC e per la componente Y. Nella debugger consolle ovviamente saranno presenti quattro tabelle, ma se ne riporta solo una. L_i è un vettore di 16 elementi, il cui i-esimo elemento rappresenta il numero di codici di Huffman di lunghezza i. In seguito, deve essere costruito il vettore $huffSize$, che replica la lunghezza del codice sulla base del vettore L_i. A partire da queste informazioni è possibile generare i codici di Huffman espressi in decimale e in binario rispettivamente attraverso i parametri *huffCode* e *huffCodeBin*. A questi vettori va poi associato *HuffVal* che rappresenta il parametro V_{ij} il cui significato è stato descritto nella sezione 5.3.1.5 e verrà chiarito nel seguito con un esempio pratico di decodifica. Per la costruzione di $HuffSize$ e dei codici di Huffman, si sono utilizzate le direttive fornite nello standard. Sono visualizzati anche i vettori *indiceIni* e *indiceFine*, che sono stati creati per ridurre il costo computazionale in fase di decodifica.

L'ultimo segmento da decodificare è lo Scan Header (marker $0xFF, 0xDA$). Nella parte iniziale di questo segmento, sono contenute le informazioni relative all'identificazione delle componenti dell'immagine e delle tabelle di Huffman. Al termine di questa parte iniziale sono presenti i dati veri propri dell'immagine codificati secondo i principi di codifica entropica descritti nel paragrafo 5.3.1.4. Prima di effettuare la decodifica dei dati, viene valutato l'ordinamento dei blocchetti 8x8 in relazione al sottocampionamento effettuato; in seguito, in base alla dimensione dell'immagine, si stabilisce se in fase di codifica è stato effettuato il completamento dei blocchetti (procedura mostrata nella figura 5.4). In figura 5.17 sono presenti i risultati di questi due passi. Si nota come il sottocampionamento nel caso in esame sia del tipo 4:2:0, ovvero i campioni di crominanza sono sottocampionati di un fattore 2 sia in direzione orizzontale che in direzione verticale, per cui a quattro campioni di luminanza corrisponde un campione di crominanza C_b e un campione di crominanza C_r. Infatti la sequenza di blocchetti è: Y, Y, Y, Y, C_r, C_b. In fase di decodifica, per risalire alla terna RGB o YC_rC_b di ogni pixel, è necessario replicare le varie componenti di C_r e C_b.

```
*** SEGMENTO CODICI DI HUFFMAN --> Marker FF C4***
*** PARTE 2 ***

Inizio alla posizione:225 -> ff c4
Lh=32 ---> La lunghezza del segmento codici di Huffman  in esame è di 32 byte
Tc=1 ---> Codici di Huffman relativi agli elementi AC
Th=0 ---> Codici di Huffman per le componenti di luminanza Y

Numero codici di Huffman:  13
Li[2]=BITS[2]=      0     1     4     3     1     1     1     1     1     0     0     0     0     0     0     0
indiceIni[2]=      -1     0     1     5     8     9    12    -1    -1    -1    -1    -1    -1    -1    -1    -1
indiceFine[2]=     -1     0     4     7     8    11    12    -1    -1    -1    -1    -1    -1    -1    -1    -1

huffSize[2]       huffCode[2]       HuffCodeBin[2]       HuffVal[2]
         2                 0                  00                4
         3                 2                 010                3
         3                 3                 011                5
         3                 4                 100                6
         3                 5                 101                7
         4                12                1100
         4                13                1101                1
         4                14                1110                2
         5                30               11110                8
         7               124              111110                9
         7               125             1111101               12
         7               126             1111110               14
         8               254            11111110               18
         0                 0            11111111               24
                     Nessuno                                     0

VERIFICA: sommaLi=lastk ---> RIUSCITA
Ultimo byte del segmento Codici di Huffman in esame alla posizione numero 258
```

Figura 5.16: Debugger Consolle: segmento Codici di Huffman

```
*** SEGMENTO SCAN HEADER --> Marker FF DA ***

Inizio alla posizione:303 -> ff da
Lunghezza dello Scan Header: 12 byte
Numero di componenti dell'immagine nello scan: 3

Id. delle componenti dello scan:          1       2       3
codici di Huffman relativi a DC:          0       1       1
codici di Huffman relativi ad AC:         0       1       1
Inizio Selezione Spettrale: 63
Fine Selezione Spettrale: 63
VERIFICA segmento Scan Header ---> RIUSCITA




*** INIZIO SCANSIONE DEI DATI EFFETTIVI ***

Inizio dati al byte numero: 317

L'immagine ha una risoluzione di 16x16

Sottocampionamento orizzontale e verticale delle componenti (Segmento Frame Header):
       Y       Cr        Cb
       2       1       1
       2       1       1

Macroblocco della componenete di Luminanza:
componente 0 -->(1,1) di dimensione 2x2 blocchetti

Numero blocchetti 8x8 delle 3 componenti:
componente 0 -->(2,2) --> 4 blocchetti 8x8 pixel totali
componente 1 -->(1,1) --> 1 blocchetti 8x8 pixel totali
componente 2 -->(1,1) --> 1 blocchetti 8x8 pixel totali

Dimensione in pixel delle 3 componenti sottocampionate:
componente 0 -->(16,16)
componente 1 -->(8,8)
componente 2 -->(8,8)

pixel della componente 0 multipli della dimensione 8x8 del blocchetto
--> eliminazione righe/colonne NON necessaria

pixel della componente 1 multipli della dimensione 8x8 del blocchetto
--> eliminazione righe/colonne NON necessaria

pixel della componente 2 multipli della dimensione 8x8 del blocchetto
--> eliminazione righe/colonne NON necessaria
```

Figura 5.17: Debugger Consolle: segmento Scan Header e analisi preliminare sull'ordinamento dei blocchetti

I dati veri e propri iniziano al byte numero 317 (si veda la figura 5.17). In figura 5.18 si riportano i primi 5 byte di dati e la relativa decodifica entropica, che consente di ottenere tre valori.

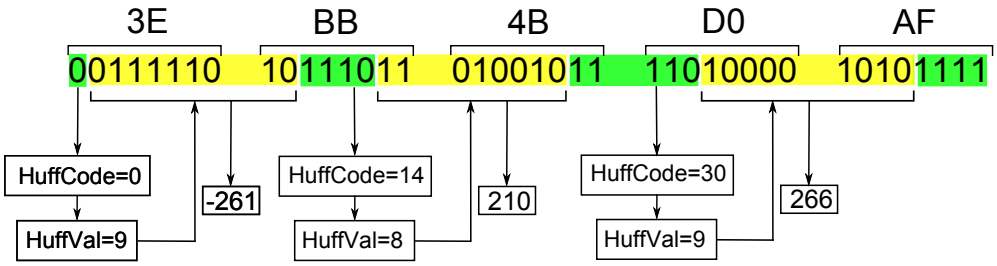


Figura 5.18: Esempio di decodifica entropica dei primi coefficienti nel file SedicixSedici.jpeg

In verde si evidenziano i codici di Huffman selezionati ($HuffCodeBin$) e in giallo la relativa stringa determinata dal valore di $HuffVal$. Per il primo valore si deve far riferimento alla tabella di Huffman degli elementi DC ($T_c = 0$) e alla componente di luminanza ($T_h = 0$). In questo caso, è presente tra i valori di $HuffCodeBin$ della tabella in questione, la parola di codice di lunghezza uno che vale 0 a cui corrisponde $HuffVal = 9$, per cui vanno presi i 9 bit successivi (evidenziati in giallo). Questi 9 bit andranno quindi decodificati. Il bit più significativo dei 9 selezionati, è uno 0, per cui il valore corrispondente sarà negativo. La procedura, in caso di numero negativo, è la seguente:

a. si effettua il complemento a 1 della stringa binaria;

b. si trasforma in decimale;

c. il valore ottenuto si considera negativo;

Al termine di questa procedura, si otterrà quindi il valore -261.

Per i restanti coefficienti, bisogna considerare la tabella di Huffman per i coefficienti AC relativa alla componente di luminanza. Questa è la tabella riportata in figura 5.16. Deve essere ricercato il codice di Huffman da questa tabella, che è presente nella stringa in esame. Si ottiene la stringa binaria $HuffCodeBin = 1110$ a cui corrisponde $HuffVal = 8$, per cui si prenderanno gli 8 bit successivi. Il bit più significativo di questa stringa di 8 bit è un 1, per cui il numero è positivo; sarà sufficiente effettuare direttamente la trasformazione in decimale. Si otterrà il valore 210.

Stesso discorso anche per il valore successivo: si trova $HuffCodeBin = 11110$ a cui corrisponde $HuffVal = 9$. Questa stringa di nove bit corrisponde al valore 266 in decimale.

Se il valore di $HuffVal$, nelle tabelle di Huffman dei coefficienti AC ($T_c = 1$), è minore o uguale a 9, si ha che *Runlength* è pari a 0, ovvero non ci saranno corse di zeri e la decodifica deve essere effettuata sugli $HuffVal$ bit. Questo è il caso per i due coefficienti AC nell'esempio rappresentato in figura 5.18. Se invece il valore è maggiore di 9, ovvero è un valore in "doppia cifra", la cifra decimale rappresenta il numero di bit da prendere relativi alla lunghezza della corsa di zeri e la cifra delle unità rappresenta il numero di bit da prendere relativi all'*Amplitude*. Un esempio pratico della situazione appena descritta è mostrato in figura 5.19, in cui $HuffVal = 24$. In questo caso i 2 bit successivi sono destinati alla corsa di zeri, e i 4 bit seguenti determinano il valore dopo la corsa di zeri.

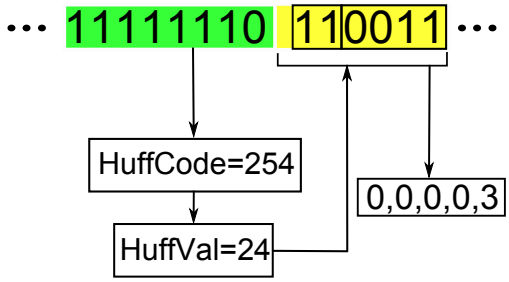


Figura 5.19: Esempio di decodifica entropica nel file SedicixSedici.jpeg - presenza della corsa di zeri

Con la procedura descritta verrà calcolato un vettore di 64 coefficienti relativi al blocchetto Y. Nel file SedicixSedici.jpeg, ci saranno poi altri tre blocchetti di luminanza e i due blocchetti delle crominanze. Il vettore di 64 coefficienti ottenuto è disposto secondo un ordine a zig zag, per cui andrà "dezigzagato", con un operazione inversa rispetto a quella mostrata in figura 5.8, ottenendo quindi una matrice 8x8. Inoltre, bisogna tener conto della codifica differenziale operata sui coefficienti in continua, per cui occorrerà tener memoria del coefficiente nel blocchetto precedente. In realtà, nel caso in esame, stiamo analizzando il primo blocchetto per cui non va attuata alcuna correzione differenziale (solo per questo blocchetto). La matrice così ottenuta contiene quindi i valori DCT non ancora dequantizzati; tale quantità è identificata dal simbolo $\dot{X}[k_1, k_2]$ con $k_1, k_2 = 0, 1, \ldots 7$.

In seguito questa matrice andrà dequantizzata utilizzando le matrici di quantizzazione estratte in precedenza ($Q[k_1, k_2]$).

$$X[k_1, k_2] = Q[k_1, k_2] * \dot{X}[k_1, k_2] \qquad con \qquad k_1, k_2 = 0, 1, \ldots, 7 \tag{5.19}$$

In questo particolare caso, tutti gli elementi della matrice di quantizzazione valgono 1, per cui tutti i valori ottenuti in uscita al decodificatore entropico, saranno

uguali a quelli in uscita al dequantizzatore (si veda figura —— Schema a blocchi concettuale del decodificatore jpeg).

Blocchetto Temporaneo 8*8 Luminanza DeZigZagato e Dequantizzato:

-261	210	309	242	148	82	12	21
266	-18	-61	-69	-55	-15	11	-31
296	-46	-90	-49	5	-55	-7	12
188	-30	-46	-29	-52	15	-22	0
157	-36	-33	-18	-26	-7	6	-9
71	-12	-24	-7	-5	-6	-4	6
10	1	8	-21	-1	3	9	-4
0	0	-1	-3	8	-1	-5	11

Blocchetto 8*8 Luminanza a cui ho applicato la DCT inversa (blocchetto Finale):

125	126	118	116	113	117	111	107
118	4	-57	-45	-37	-31	-35	-26
117	-62	-128	-121	-117	-111	-101	-74
117	-53	-122	-110	-111	-68	-63	-77
116	-57	-125	-114	-115	-82	-43	-86
114	-58	-126	-114	-113	-97	-52	-87
117	-56	-128	-104	-43	-91	-55	-73
120	-52	-117	-96	-35	-73	-70	-34

Figura 5.20: Valori finali del primo blocchetto di luminanza del file SedicixSedici.jpeg

In figura 5.20 sono mostrati i valori ottenuti per il primo blocchetto di luminanza dei quattro totali, prima e dopo l'applicazione della DCT inversa. Ci saranno poi i blocchetti relativi alle due componenti di crominanza.
L'ultimo passo è quello dell'unione dei valori numerici dei blocchetti di diverso tipo, così da stabilire una terna di valori per ogni pixel dell'immagine; per cui è necessario replicare i valori delle componenti di crominanza. La terna risultante appartiene allo spazio $Y\,C_b\,C_r$, e può eventualmente essere trasformata in RGB, tenendo conto dello SHIFT operato in fase di codifica. La trasformazione che consente di ottenere la terna $R\,G\,B$ è espressa nella 5.20.

$$\begin{cases} R = (Y + 128) + 1.402\,C_r \\ G = (Y + 128) - 0.34414\,C_b - 0.71414\,C_r \\ B = (Y + 128) + 1.772\,C_b \end{cases} \tag{5.20}$$

5.4 Decodifica SPinV Code

In questa sezione verranno descritte le procedure che consentono la lettura/decodifica dello SPinV Code. Ulteriori informazioni sono presenti in Appendice B. Come detto nel paragrafo 4.6 (pag.118), lo SPinV Code è costituito da una barra rettangolare di colore nero con intorno una fascia perimetrale bianca, e da una zona in cui sono presenti i moduli elementari che codificano i dati nello SPinV Code e che rappresentano i bit 0 e 1 (si veda la figura 4.18 a pagina 120). La barra rettangolare ha diverse "funzioni":

- Presenza dello SPinV Code: se in fase di lettura/decodifica, si individua la barra rettangolare, vuol dire che all'interno della fotografia è contenuto lo SPinV Code.
- Modello di rivelazione della posizione: una volta individuata la barra rettangolare, ovvero il posizionamento dei 4 vertici del rettangolo, è possibile risalire al posizionamento dei moduli elementari.
- Deformazione prospettica: si vuole sfruttare la deformazione della barra rettangolare per ottenere informazioni aggiuntive sul posizionamento del terminale, in riferimento a quanto detto nel Capitolo 4.

Il riconoscimento dello SPinV Code è basato essenzialmente su 3 algoritmi:

- algoritmo di analisi a pezzi;
- algoritmo a croce;
- algoritmo di riduzione dei punti;

Con l'*algoritmo di analisi a pezzi* si decodifica e si analizza un "pezzetto" dell'immagine alla volta, per cui il processo di decodifica jpeg e quello di ricerca della barra rettangolare dello SPinV Code, vengono effettuati congiuntamente. Infatti, decodificare l'immagine completamente e in seguito leggere il codice, significa memorizzare una terna di valori per ogni pixel dell'immagine, e ciò comporterebbe un occupazione di memoria troppo elevata (e inutile), soprattutto considerando che i dispositivi mobili e le relative applicazioni, richiedono un occupazione di memoria piuttosto limitata. Inoltre, con questo approccio si riduce il tempo di elaborazione. La fase di ricerca della barra rettangolare consiste nella ricerca dei quattro vertici caratteristici del blocco rettangolare. Per cui si decodifica un "pezzetto" dell'immagine e su questo si ricerca un punto di vertice che è l'unico parametro che viene (eventualmente) memorizzato al termine della decodifica di questo "pezzetto".

l'*algoritmo a croce* si basa sulle differenze di contrasto acromatico tra bianco e nero, con l'obiettivo di stabilire se un determinato pixel rappresenta un pixel di

vertice (vertice in alto a sinistra, alto a destra, basso a sinistra, basso a destra). Inoltre, l'algoritmo a croce determina le coordinate x,y dei pixel di angolo relative al sdr della fotografia. Questo algoritmo prende in esame i soli pixel neri e ne analizza un intorno: l'analisi avviene nelle quattro direzioni (a croce) in alto, in basso, a destra e a sinistra. L'ampiezza dell'intorno di analisi può essere variata in base ad un parametro ($marg$). E' necessario analizzare un intorno e non solo i punti adiacenti a causa dei bordi non netti (contrasto che decresce non bruscamente), causati sia dal fatto che uno o più pixel possono trovarsi nel bordo (creando un grigio intermedio tra bianco e nero), e sia da processi attuati in fase di codifica che tendono a sfumare i bordi (l'uso del jpeg implica una perdita di informazione). Questo algoritmo, data la sua "struttura", fornisce nell'intorno del punto di vertice effettivo, una serie di coordinate che, se da una parte fanno decrescere il falso allarme e aumentano la probabilità di rivelare correttamente il vertice, dall'altra potrebbero generare un errore sulla vera posizione dei vertici del rettangolo. Ecco perché si usa l'algoritmo di riduzione dei punti.

L'*algoritmo di riduzione dei punti* è in grado di analizzare le diverse coordinate estrapolate dall'algoritmo a croce e carpire se tali coordinate sono riconducibili ad un punto di vertice. L'algoritmo prevede anche il conteggio dei punti dell'intorno che sono stati rivelati (parametro utile per il falso allarme). In definitiva saranno presenti (in caso di funzionamento ottimale) 4 coppie (x,y) che rappresentano le coordinate sul piano dei 4 vertici del rettangolo di posizionamento e sulla base di queste coordinate e con la conoscenza a priori della struttura del codice, è possibile valutare gli intervalli di analisi degli slot dei moduli elementari in cui è presente il bianco o il nero. Con questo approccio, una volta capito dove si trovano i dieci slot, posso saltare alcune fasi del processo di decodifica, agendo solo sulle zone di interesse e aumentando di fatto la velocità di elaborazione. È interessante notare che per ottenere un corretto riconoscimento dello SPinV Code, è sufficiente avere un immagine a bassa risoluzione, questa considerazione, aggiunta al fatto che la fase di riconoscimento ha un costo computazionale abbastanza ridotto (principalmente confronti e somme), implica che tale codice può essere rivelato sfruttando la miniatura dell'immagine e potrebbe eventualmente essere rivelato in real time durante un video. Quest'ultimo caso non è stato però analizzato.

Per la fase di test sono state scattate una serie di fotografie con iPhone 4 in diverse condizioni di illuminazione, con e senza flash, variando la posizione di scatto fotografico e su diversi SPinV Code (ovvero su SPinV Code che codificano diverse sequenze di bit). Le foto sono state poi usate da input allo script. Lo script, riportato in Appendice B, non prevede in questa prima versione, l'analisi e la compensazione delle distorsioni e in particolare delle rotazioni. Per spiegare meglio questo concetto, faccio riferimento alla figura 5.21; sulla parte sinistra della figura è mostrato il

codice ”originale” che codifica la sequenza di bit 0101010101 e sulla parte destra si riporta uno scatto fotografico effettuato da posizione non ideale (non in bolla) e in particolare che risulta ruotato rispetto all'originale. In entrambi i casi sono tracciati dei segmenti, che ”spiegano” il funzionamento delle procedure di decodifica: in rosso sono presenti i segmenti che identificano il posizionamento della barra rettangolare e con i quadratini blu, si identificano le aree, in cui lo script andrà a ricercare i moduli elementari. La posizione di questi quadratini blu, dipende dalla posizione utilizzata della barra rettangolare (in rosso). I pallini di colore verde identificano i punti di vertice del quadrilatero individuati dall'algoritmo a croce. Come si vede dalla figura, le aree che si prendono in considerazione per ”capire” se si sta analizzando un modulo bianco o un modulo nero, sono più piccole delle aree dei moduli così da migliorare il riconoscimento. In particolare sono dei quadratini i cui lati hanno una dimensione di 2/5 rispetto ai lati del quadrato relativi al modulo elementare.

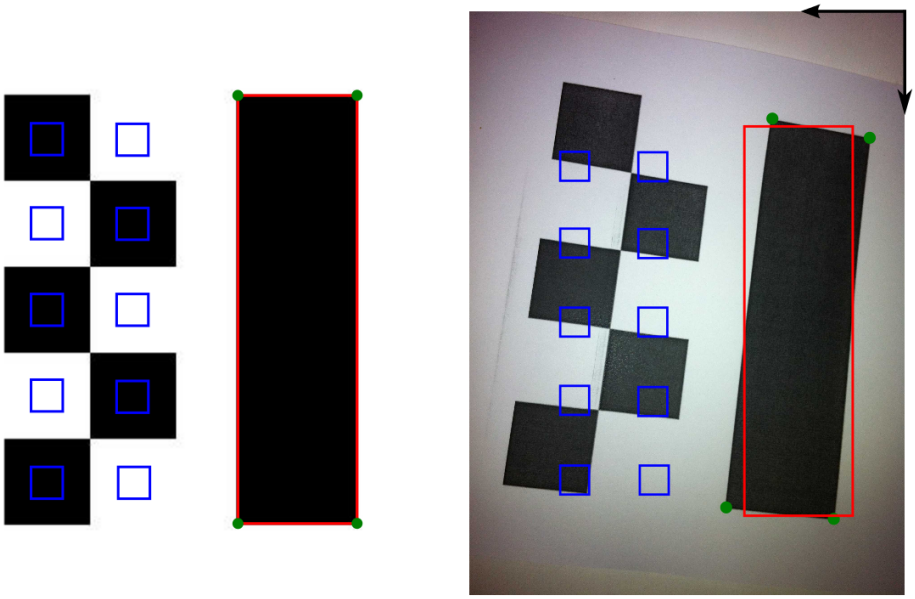


Figura 5.21: Aree di analisi dello slot e problema di rotazione dello SPinV Code

Lo script è in grado di applicare i propri algoritmi solo rispetto alla direzione verticale e orizzontale (in riferimento al sdr della fotografia). Per cui, nelle fotografie che presentano delle forti rotazioni, anche se l'algoritmo a croce fosse in grado di ricavare correttamente gli angoli del quadrilatero, verrà utilizzato un rettangolo (in rosso) i cui lati sono paralleli agli assi del sdr della fotografia e che quindi non rispecchiano la barra rettangolare (ruotata) presente in fotografia. Infatti nell'immagine ”originale” le coordinate dei vertici coincidono con il rettangolo realmente utilizzato. Nel caso della fotografia ruotata, il rettangolo non coincide con i punti di vertice verdi. Nell'esempio di figura 5.21, la prima riga di moduli è interpretata correttamente

come 01010, mentre la seconda riga è letta in modo errato: 00101 al posto di 10101, sbagliando di fatto la decodifica del secondo bit. Inoltre, c'è una forte incertezza sulla decodifica di quasi tutti i bit della seconda linea, in cui l'area di analisi si trova sempre tra due moduli elementari del codice. Per valutare se all'interno dell'area di analisi (quadratino blu) è presente bianco o nero, si effettua una media tra i valori del colore assunti in questa zona rispetto a una certa soglia presa come riferimento. Tale soglia è una soglia di bianco ed è in particolare una *soglia adattativa del bianco* stabilita analizzando la fascia perimetrale bianca che si trova tra la barra rettangolare e i moduli elementari. Così facendo in una foto particolarmente scura, la soglia adattativa si abbassa consentendo di ottenere una lettura più efficace del codice. Il problema della rotazione dello SPinV Code potrebbe risolversi semplicemente attuando un analisi, non necessariamente lungo le direzioni orizzontali e verticali ottenendo quindi una barra rettangolare ruotata come mostrato in figura e un conseguente corretto posizionamento delle aree di analisi dei moduli. Questa correzione è mostrata in figura 5.22.

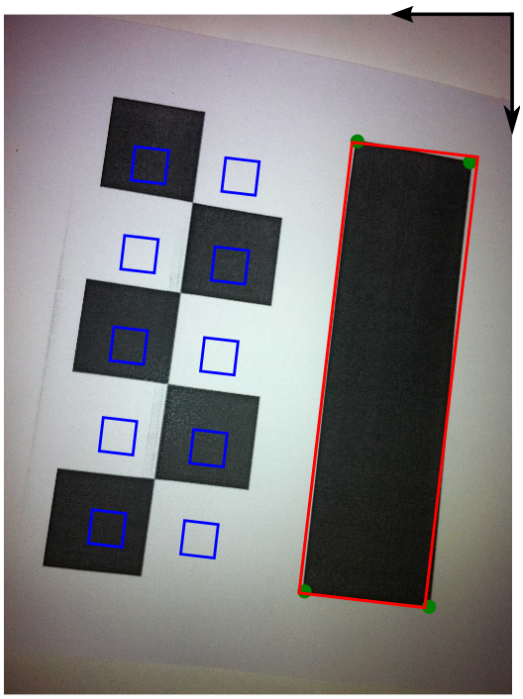


Figura 5.22: Rotazione della barra rettangolare

Altro problema è quello della deformazione prospettica il cui effetto "pratico" è quello di modificare le proporzioni-distanze tra i vari elementi dello SPinV Code. Nel capitolo 4 sono state definite le "linee guida" per la risoluzione del problema, utilizzando eventualmente una versione corretta dell'immagine. In riferimento alla figura 5.22, la deformazione prospettica fa si che le aree di analisi (quadratini

blu) non si trovino esattamente al centro del modulo elementare dello SPinV Code. Inoltre, variano le proporzioni tra i lati dei vari elementi dello SPinV Code, per cui il rapporto fra i lati della barra rettangolare variano, così come variano i rapporti fra i lati dei moduli elementari. Ogni modulo elementare subirà una deformazione diversa, per cui le aree di analisi relative ai moduli dovrebbero variare.

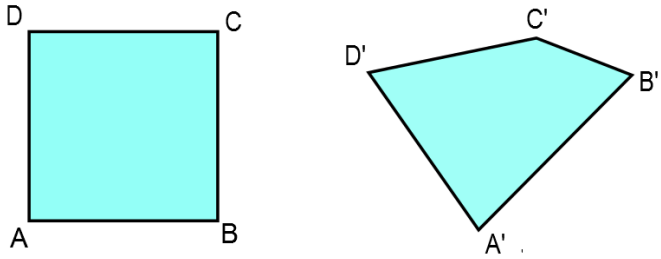
In definitiva, i problemi che possono verificarsi sono sostanzialmente due: deformazione prospettica e rotazione del codice. L'utilizzo di un maggior numero di moduli elementari "peggiora la situazione", determinando in pratica molti errori sul riconoscimento del bit (ovvero del colore bianco/nero) proprio perché verranno analizzate molte zone intermedie fra i moduli. In effetti, si sono considerati anche dei codici con 24 e 40 moduli, ma i risultati hanno suggerito di trovare una soluzione ai problemi di prospettiva e rotazione (robustezza) ed in seguito, eventualmente, incrementare il numero di moduli. D'altronde una volta stabilito il metodo per correggere le rotazioni ed analizzare delle corrette zone sui moduli elementari, il problema può essere esteso inserendo un numero di moduli via via crescente.

Nel seguito si propone un analisi geometrica/grafica del problema della deformazione prospettica, il cui obiettivo è quello di trovare le zone corrette che devono essere analizzate. L'analisi che segue può essere considerate parallela ed integrativa a quella svolta nel capitolo 4. Considero le seguenti affermazioni:

Definizione. Una trasformazione proiettiva (o proiettività) di α in β è una trasformazione biunivoca, continua, che conserva l'allineamento.;

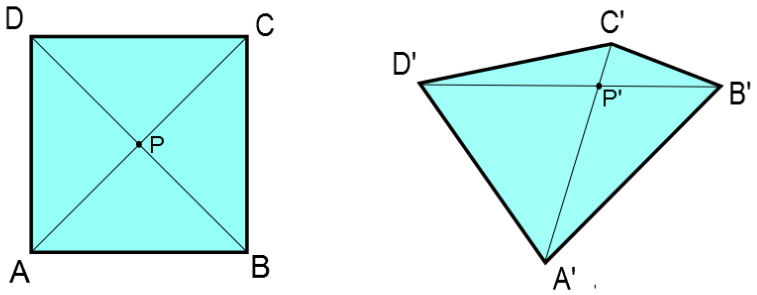
Teorema. Dati due piani proiettivi α e β, una trasformazione proiettiva F di α in β è univocamente determinata conoscendo l'immagine di quattro punti di α a tre a tre non allineati.

Considero un quadrangolo ABCD del piano α e siano A',B',C',D' i punti relativi ai quattro vertici: Un punto qualsiasi è costruibile, a partire dal quadrangolo iniziale

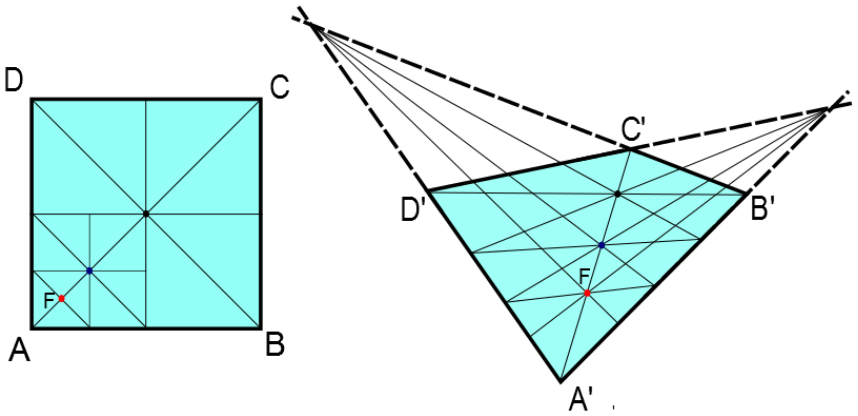


ABCD, se si ottiene come intersezione di rette passanti per punti già costruiti.

Il punto di intersezione delle diagonali (punto P) nel quadrangolo ABCD, è un punto costruibile sul quadrangolo A'B'C'D' sempre mediante intersezione delle diagonali: Per cui il punto P avrà come immagine il punto P' ottenuto intersecando A'C' con

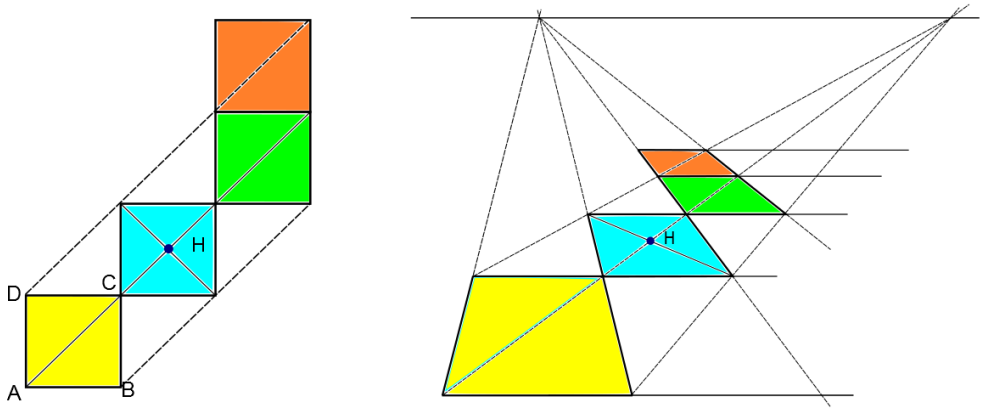


B'D', il quale è quindi determinato univocamente a partire dai punti A',B',C',D'.
Il problema quindi si riconduce alla costruzione di una maglia di punti, costruibile a partire dai punti noti, che invada il piano e che sia sempre più fitta. Si può procedere nel seguente modo: dato il quadrangolo ABCD, lo dividiamo in 4 parti sfruttando l'intersezione derivante dal prolungamento dei lati. Ognuna di queste parti può essere ancora divisa con lo stesso processo. In questo modo, dato un qualunque punto F, interno al quadrangolo, è possibile costruire una successione di quadrangoli sempre più piccoli, uno incluso nell'altro, che contengono il punto F. Il punto F' dovrà a sua volta essere individuato dai quadrangoli corrispondenti: L'aspetto di interesse, è che è possibile trovare l'immagine di un punto F, anche



se questo si trova all'esterno del quadrangolo. Infatti, è possibile aggiungere un quadrangolo adiacente a quello dato lungo uno qualunque dei lati e poi aggiungerne uno e un altro ancora e così di seguito fino a costruire una rete che arrivi a qualunque

punto al finito o all'infinito nel piano. Questo processo è mostrato nella seguente figura:



Quindi, a partire da quattro punti iniziali,è possibile costruire infiniti altri punti, intersecando le rette che passano per punti già costruiti.

Per quanto riguarda i moduli elementari, in alcuni test, si sono utilizzati dei moduli colorati, modificando opportunamente le soglie. Per la precisione si sono utilizzati, oltre al bianco e al nero, anche giallo, blu, verde, magenta, ciano e rosso o solo alcuni di questi colori. La corretta decodifica dipende però fortemente dalle condizioni di illuminazione, e comunque in alcuni casi genera troppa ambiguità. L'errata decodifica dipende anche dal fatto che gli spazi di colore sono dipendenti dal dispositivo (spazi di colore relativi), per cui uno stesso colore creato con un programma grafico, varia una volta che viene stampato (a seconda della stampante), e varia ancora fotografandolo (anche in relazione alle condizioni di illuminazione). Per cui, l'utilizzo di moduli colorati richiede un analisi più approfondita delle problematiche relative alla variazione del colore e agli spazi di colore relativi.

Fino ad ora sono stati descritti i limiti dello script, nella pagina successiva si riportano invece alcune fotografie che i processi di decodifica sono in grado di decodificare correttamente.

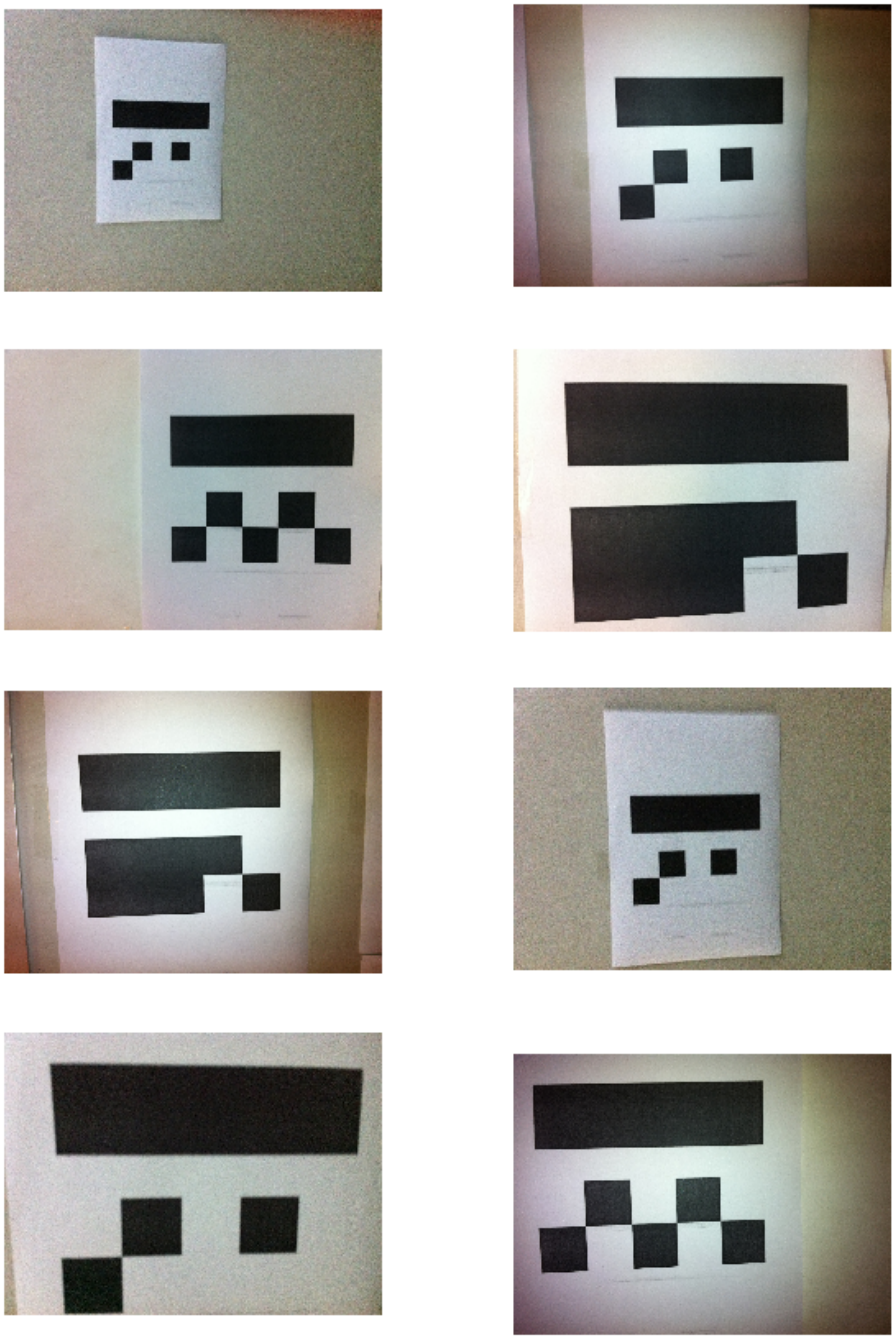


Figura 5.23: Alcuni scatti fotografici dello SPinV Code correttamente decodificati

Capitolo 6

Conclusioni e Sviluppi Futuri

Per affrontare tutti gli aspetti relativi al positioning indoor, con il metodo proposto, si è considerato un approccio semplice e generico. Sono stati trattati molti degli aspetti di interesse, per cui questo lavoro di tesi può considerarsi, a mio avviso, come una sorta di punto di partenza per sviluppi futuri.
La semplicità dell'approccio proposto è "emersa" in tutti gli aspetti analizzati. Nello studio sul dead reckoning, il calcolo della distanza e della direzione sono stati considerati come due processi indipendenti. La distanza è stata valutata tenendo conto delle misure di accelerazione lungo due assi, considerando delle soglie fisse e un lunghezza del passo costante con l'obiettivo del conteggio del numero di passi effettuati. Per la valutazione della direzione si è tenuto conto delle variazioni di un unico parametro (yaw). Il terminale usato per i test è lo smartphone Apple iPhone 4.
Le problematiche relative all'autolocalizzazione sono state trattate da un punto di vista generale fornendo delle soluzioni teoriche e parziali. Il codice posizionale introdotto (SPinV Code) è un codice strutturalmente semplice, che integra al suo interno i soli elementi essenziali, codificando una quantità di dati ridotta (10 bit).
Lo script presente in Appendice B effettua il processo di decodifica descritto nel Capitolo 5; con il termine 'decodifica' si intende la decodifica congiunta sia del file in formato jpeg sia dello SPinV Code. Per i test si è utilizzata la miniatura del file jpeg.
Sono molte le possibili evoluzioni del sistema sotto tutti i punti di vista:

1. Sviluppo delle tecniche sul dead reckoning:
    - lunghezza del passo variabile;
    - soglie variabili e/o analisi dell'andamento;
    - integrazione delle misure inerziali;

2. Sviluppo delle tecniche di autolocalizzazione:
    - aspetti teorici di geometria proiettiva;
    - sviluppo del codice posizionale;
3. Sviluppo del processo di decodifica e creazione dell'interfaccia:
    - funzionamento in real time;
    - fotocamera come sensore aggiuntivo;
    - creazione dell'applicazione;
    - principi di navigazione e servizi LSB;

Per quanto riguarda lo sviluppo delle tecniche sul dead reckonig, è possibile ottenere dei miglioramenti sulla stima della distanza percorsa. Si può procedere su due strade: lunghezza del passo variabile (come mostrato in S. H. Shin [34]) e analisi dei dati acquisiti, ovvero analizzare l'andamento precedente all'istante di analisi, per migliorare la stima del numero di passi effettuati; nel nostro caso si è considerato il conteggio dei passi basato su una coppia di soglie, senza analizzare gli istanti precedenti; queste soglie potrebbero essere rese variabili e inoltre potrebbe considerarsi l'analisi di alcuni parametri, ad esempio la frequenza dei passi, per evitare errori nel conteggio. Un analisi di questo tipo, necessita l'approfondimento delle tematiche relative alla biomeccanica del passo in relazione alle misurazioni fornite dai sensori inerziali. Altro aspetto da approfondire, è sicuramente legato all'utilizzo delle diverse misure inerziali per migliorare le stime. Ad esempio per valutare il numero dei passi, non si sono considerate le misure del giroscopio che potrebbero invece fornire dei dati aggiuntivi per migliorare il processo di individuazione del passo.
Per quanto riguarda l'autolocalizzazione, sono ancora molti gli aspetti di geometria proiettiva da caratterizzare. Questi concetti sono direttamente legati alla struttura del codice posizionale così da permettere di inserire all'interno del codice un maggior numero di moduli semplici (e quindi di bit) ed eventualmente dei diversi modelli per la valutazione dell'area del codice. Nella versione proposta di SPinV Code è possibile codificare al massimo 10 bit. Per cui, in un scenario di questo tipo, sarebbe possibile usare fino a un massimo di $2^{10} = 1024$ cartelloni e associare poi ad ogni sequenza di bit la coppia di coordinate e l'angolo α. In una versione successiva del codice posizionale, è pensabile che al suo interno siano codificate direttamente le coordinate di interesse e l'angolo, senza effettuare alcuna associazione.
L'approccio usato in questo lavoro di tesi, analizza il file jpeg relativo alla fotografia scattata allo SPinV Code. Per questo motivo, si è reso necessario uno studio approfondito dei processi di co-decodifica dello standard jpeg e delle modalità di incapsulamento del bitstream. L'analisi del file non è strettamente necessaria, dato

che l'applicazione potrebbe funzionare in real-time, mediante video, così da evitare l'inutile processo di compressione (codifica) e decompressione (decodifica) del file jpeg. Un eventualità di questo tipo necessita un interazione diretta con il flusso video e per questo servono delle conoscenze avanzate della piattaforma di sviluppo. Nel caso di applicazioni per dispositivi Apple è indispensabile la conoscenza del linguaggio di programmazione Objective-C, inoltre sono necessari tool come XCode, Instruments, Interface Builder, Quartz Composer ... e la conoscenza approfondita degli specifici framework per la gestione dei flussi multimediali e la gestione dell'apparato sensoristico. D'altronde gli smartphone di ultima generazione sono dotati di hardware molto performante in grado di gestire applicativi anche molto complessi e in tempo reale; basti pensare che l'iPhone 4S è dotato di un processore a doppio core con frequenza di 1 Ghz e RAM di 512 MB. Indubbiamente, sia le prestazioni a livello grafico, sia le prestazioni a livello di sensori sono destinate a migliorare sempre di più nel tempo, con i nuovi modelli. Esistono già sul mercato applicazioni in grado di decodificare in real time codici bidimensionali (ad esempio qr code), sfruttando la videocamera. Un esempio di applicativo è i-nigma della 3GVision Inc. In uno scenario di questo tipo la videocamera del dispositivo può quindi essere considerata come un sensore aggiuntivo. Per completare il quadro, è utile sottolineare un altro aspetto, non trattato nel corso di questo lavoro di tesi. Infatti si è parlato degli aspetti di posizionamento, senza accennare invece a tutti quei servizi aggiuntivi basati sulla posizione, ovvero i Location Based Service, temi che sono stati affrontati maggiormente nei precedenti lavori di tesi sul progetto SPinV.

Appendice A

Il codice Matlab: Dead Reckoning

Si riporta nel seguito il codice Matlab utilizzato nel test sul dead reckoning, per il conteggio dei passi e per la stima della posizione. I parametri settati sono relativi al percorso 1.

```
clear;
clc

%carico i dati acquisiti
load('dati_iniziali.mat')

%SETTARE :
%INTERTEMPI - COORDINATE INIZIALI - LUNGHEZZA PASSO -  COORDINATE
%INTERTEMPI - DIREZIONE

%intertempi
intertempi= [4.69 3.81 3.95 3.94];
num_intertempi=length(intertempi);
%coordinate del punto iniziale
x_ini=5;
y_ini=8;
%lunghezza passo
D=0.67;
%coordinate partenza e intertempi:
x_intertempi=[5   10   15   20    25];
y_intertempi=[8   8     8    8     8];
%direzione (lungo le x -> 0     lungo le y --> pi/2)
direzione=0.

%CALCOLO TEMPO TOTALE E GLI INTERTEMPI CUMULATIVI
tempi_totali=zeros(num_intertempi,1);
tempi_totali(1)=intertempi(1);
for h=2:length(intertempi)
    tempi_totali(h)=tempi_totali(h-1)+intertempi(h);
end


%VALORI EFFETTIVI CHE SI DEVONO CONSIDERARE (SULLA BASE DEL TEMPO AL NODO DI
      ARRIVO)
```

```
val_effettivi=ceil(tempi_totali(num_intertempi)*32);

val=length(tempo); %LUNGHEZZA  VETTORE TOTALE
acc_tot=ones(val,1); %CREO VETT ACC.TOTALE


%ACCELERAZIONE TOTALE (LUNGO Y E Z)
for i=1:val
   acc_tot(i)=sqrt((acc_x_y_z(i,2)).*(acc_x_y_z(i,2))+  (acc_x_y_z(i,3)).*(
        acc_x_y_z(i,3)));
end

%SOGLIE
soglia_inf=0.85;
soglia_sup=1.25;

%ANALISI DISTANZA
%ALGORITMO PER IL CALCOLO DEL NUMERO DI PASSI E DELL'ISTANTE IN CUI AVVIENE
max_passi=1000;
ist_passi=zeros(max_passi,1);    %ISTANTI IN CUI AVVIENE IL NUOVO PASSO
elemento_passi=zeros(max_passi,1); %ELEMENTO DEL VETTORE COINCIDENTE AL
      MOMENTO IN CUI VIENE FATTO IL NUOVO PASSO
k=1;
sotto=1;
sopra=0;
cont=0;
for i=1:val_effettivi
    if (acc_tot(i)>soglia_sup)
        sopra=1;
    end

    if (acc_tot(i)<=soglia_sup)
        sopra=0;
    end

    if (acc_tot(i)<soglia_inf)
        sotto=1;
    end

    if (sotto==1 && sopra==1)
        cont=cont+1;
        sotto=0;
        sopra=0;
        ist_passi(k)=tempo(i);
        elemento_passi(k)=i;
        k=k+1;
    end
end
cont %numero di passi effettuati


passi_effettuati=0.2*ones(1,cont);
for t=1:cont
    ist_passi_effettuati(t)=ist_passi(t);
end
```

```
%ANALISI DIREZIONE
yaw_deg=(yaw*360)/(2* pi); %converto in DEG
%calcolo della media istantanea
tot_deg=0;
tot_rad=0;
for i=1:val
    tot_deg=tot_deg+yaw_deg(i);
    tot_rad=tot_rad+yaw(i);
    media_temp_deg(i)=tot_deg/i; %vettore usato per il grafico(3)
    media_temp_rad(i)=tot_rad/i; %vettore usato per il calcolo della
        posizione
end

%gradi di yaw medi percepiti negli istanti di calcolo del nuovo passo
for r=1:cont
    media_yaw_rad(r)=media_temp_rad(elemento_passi(r)); %vettore che contiene
        il valore medio dello yaw negli istanti del passo
    yaw_ist_rad(r)=yaw(elemento_passi(r)); %vettore che contiene il valore
        istantaneo dello yaw negli istanti del passo
end


%CALCOLO DELLE POSIZIONI AD OGNI PASSO PERCEPITO
x(1)=x_ini;
y(1)=y_ini;

x_mis(1)=x_ini;
y_mis(1)=y_ini;

for t=2:(cont+1)
    %calcolo posizione nuova x,y basata sulla media istantanea dei valor di
        yaw
    x(t)=x(t-1)+ D*(cos(media_yaw_rad(t-1)));
    y(t)=y(t-1)+ D*(sin(media_yaw_rad(t-1)));
    %calcolo posizione nuova x,y basata sul valor misurati di yaw
    x_mis(t)=x_mis(t-1)+ D*(cos(yaw_ist_rad(t-1) + direzione));
    y_mis(t)=y_mis(t-1)+ D*(sin(yaw_ist_rad(t-1) + direzione));
end


%GRAFICI   GRAFICI   GRAFICI

%GRAFICO ACCELERAZIONE TOTALE E  PASSI
figure(1);
plot(tempo,acc_tot,'r');
title('TEST 1.B: Accelerazione e  Passi');
xlim([0 tempi_totali(num_intertempi)])
ylim([0 1.9])
xlabel('tempo (sec)');
ylabel('a_{yz} (m/s^2)');
%soglie (orizzontali)
hold on
```

```
plot([0 tempi_totali(num_intertempi)], [soglia_inf soglia_inf],'g','Linewidth
    ', 1.2)
hold on
plot([0 tempi_totali(num_intertempi)], [soglia_sup soglia_sup],'b','Linewidth
    ', 1.2)
%asintoti verticali (intertempi)
for p=1:num_intertempi
hold on
plot([tempi_totali(p)  tempi_totali(p)],[-2 2],'m')
end
%grafico i passi
hold on
stem(ist_passi_effettuati,passi_effettuati,'k')
legend('acc_y_z','soglia inferiore','soglia superiore','intertempi')


%GRAFICO DIREZIONE
figure(2);
plot(tempo,yaw_deg,'b')
hold on
plot(tempo,media_temp_deg,'r');
%asintoti verticali (intertempi)
for p=1:num_intertempi
    hold on
    plot([tempi_totali(p)  tempi_totali(p)],[-10 15],'m')
end
xlim([0 tempi_totali(num_intertempi)])
title('TEST 1.B: Direzione')
xlabel('tempo (sec)','FontSize',12)
ylabel('yaw (deg)','FontSize',12)
legend('Valori istantanei','Valori mediati','Intertempi')
grid on


%GRAFICO NUOVA POSIZIONE
figure(3)
plot(x_mis,y_mis,'bo')
hold on
plot(x,y,'r+')
hold on
plot(x_intertempi,y_intertempi,'r*')
title('TEST 1.B: Stima della posizione ad ogni passo')
xlabel('x (m)','FontSize',12)
ylabel('y (m)','FontSize',12)
legend('Valori Istantanei','Valori Mediati','Punti Passaggio')
xlim([0 27])
ylim([0 22])
grid on
```

Appendice B

Il codice C: Il Processo di Decodifica

Nel seguito è inserito il codice sorgente in linguaggio C utilizzato per il processo di decodifica; per la fase di test sono state effettuate delle variazioni al codice implementando cicli e variabili diverse, in base alle esigenze di studio. La differenza tra il codice usato e quello allegato nel seguito, è che la combinazione di caratteri doppio backslash (per i commenti), è stata sostituita con %.

```
%  main.m
%  Riconoscimento

%  Created by Roberto Bosco on 8/31/11.
%  Copyright Roberto Bosco 2011. All rights reserved.

#include <stdio.h>
#include <math.h>
#define SIZE 8

%PROTOTIPI DI FUNZIONE
int ByteElemento(FILE *,unsigned int);
int ricercaDoppia(FILE *,int,unsigned int,int,int);
int unioneHex(int,int);
int separaHexPrimo(int);
int separaHexSecondo(int);
void stampaVettoreInt(const int *,int);
void stampaBits(int,int);
unsigned int funzioneDecimale(unsigned short,unsigned short,const short *);
unsigned int proceduraNumNegativo(unsigned short,unsigned short,const short
    *);
void IDCTBlocchetto (const int *,int *);



int main (int argc, const char * argv[]){

  FILE *PuntFile;
  PuntFile=fopen("Image_Test/IMG_0498_gimp_edit.jpg","rb");
  %file da caricare
```

```
if (PuntFile==NULL){
  printf("\nErrore nel caricamento del File\n");
  exit(1); %esce dal main
}
else{
  printf("\nFile Caricato Correttamente\n");
}


fseek(PuntFile,0,SEEK_END);
%con l'istruzione SEEK_END si posiziona il puntatore di posizione
%all'ultimo byte del file (puntato da PuntFile) più in generale
%con l'istruzione fseek(Punt,posizione,SEEK_SET) è possibile
%settare la posizione del puntatore al file nella posizione
%desiderata.

int dimFile=ftello(PuntFile);
rewind(PuntFile);
%porto il puntatore di posizione all'inizio (posizione 0)



int primoElemento=ByteElemento(PuntFile,0);
int secondoElemento=ByteElemento(PuntFile,1);
int penultimoElemento=ByteElemento(PuntFile,dimFile-2);
int ultimoElemento=ByteElemento(PuntFile,dimFile-1);


%int fread (void *vet, int size, int n, FILE *fp);
%legge (al piu') n oggetti dal file puntato da fp, collocandoli
%nel vettore vet, ciascuno di dimensione size. Restituisce un
%intero che rappresenta il numero di oggetti effettivamente
%letti (k). ovvero se int n=10, vuol dire che nel vettore dati
%saranno presenti 10 elementi, ognuno di dimensione pari a
%sizeof(int)=4 byte


if(primoElemento!=0xFF || secondoElemento!=0xD8 || penultimoElemento!=0xFF
      || ultimoElemento!=0xD9){
  printf("Il file Caricato Non è un file jpg");
  exit(1);
}

%fseek(PuntFile,scansione,SEEK_SET);
%posiziona il puntatore di posizione alla posizione scansione

%fread(&elementoFile,1,1,PuntFile);
%legge il file alla posizione in cui si trova il puntatore


unsigned int scansione;


%Verifico se nel file in questione è presente la preview:
```

```
if ( ricercaDoppia(PuntFile, dimFile,3,0xFF,0xD9)==dimFile-2)
  printf("\n\n*** Nel file Non è presente la preview ---> immagine=1(no
       preview) ***");
else {
  printf("\n\n*** Nel file è presente la preview ---> immagine=1(preview)
       || immagine=2(completa) ***");
}

%MINIATURA O IMMAGINE?
unsigned short immagine=1;
%immagine=1 ---> considera la preview dell'immagine
%immagine=2 ---> considera l'immagine vera e propria

if (immagine==1)
  scansione=2;
  %scansione rappresenta la posizione del byte di lettura
  %parte dal byte 2--> scansione=2
  %per l'analisi sull'immagine completa scansione deve essere
  %settata oltre il secondo FFDB, ovvero dopo l'inzio del
  %secondo segmento matrice di quantizzazione

else if(immagine==2)
  scansione=ricercaDoppia(PuntFile, dimFile,3,0xFF, 0xD9);

unsigned int scansioneRif; %questo parametro serve per
%"risolvere il problema",che in alcuni file il segmento
%codici di Huffman  si trova prima del segmento Frame Header,
%per cui la ricerca dell'Header di questi segmenti parte in ogni caso
%dal byte dato da scansioneRif

unsigned int elementoQuantizzazione;
int Lq;
int Pq;
int Tq;


int matriceQuantizzazioneY_ZZ[SIZE*SIZE];
%int matriceQuantizzazioneY[SIZE*SIZE];
%utile solo per la visualizzazione,
%ovvero non viene impiegato nell'algoritmo

int matriceQuantizzazioneCr_ZZ[SIZE*SIZE];
%int matriceQuantizzazioneCr[SIZE*SIZE];
%utile solo per la visualizzazione,
%ovvero non viene impiegato nell'algoritmo

int matriceQuantizzazioneCb_ZZ[SIZE*SIZE];
%int matriceQuantizzazioneCb[SIZE*SIZE];
%utile solo per la visualizzazione,
%ovvero non viene impiegato nell'algoritmo

%int matriceQuantizzazioneAlpha_ZZ[SIZE*SIZE];
%int matriceQuantizzazioneAlpha[SIZE*SIZE];
%utile solo per la visualizzazione,
```

```
%ovvero non viene impiegato nell'algoritmo

140

unsigned short contrY=0;
unsigned short contrCr=0;
unsigned short contrCb=0;

unsigned short controlloDQT=0;

for(unsigned short i=0;i<4;i++){ %il ciclo è eseguito
%4 volte perchè è possibile che in alcuni file siano presenti fino
%a 4 matrici di quantizzazione: una per Y, una per Cr, una per Cb
150 %e una per il cananale alpha (trasparenza).
%in genere saranno presenti solo 2 matrici: una per Y e l'altra
%per le componeneti di crominanza.

  if(controlloDQT==0){
    elementoQuantizzazione=ricercaDoppia(PuntFile,dimFile,scansione,0xFF,0
        xDB);
    %numero del byte del segmento di quantizzazione, trova il byte
    %in cui è contenuto la coppia di byte ricercati (4° e 5°
    %argomento della funzione) e lo restituisce. La funzione termina
    %una volta ritrovato il primo elemento
160

    scansione=elementoQuantizzazione+2;

    Lq=unioneHex(ByteElemento(PuntFile,scansione),ByteElemento(PuntFile,
        scansione+1) );
    scansione=scansione+2;
  }


  Pq=separaHexPrimo(ByteElemento(PuntFile, scansione));
  % Precisione degli elementi della matrice di quantizzazione.
170 %0=8bit 1=16 bit
  Tq=separaHexSecondo(ByteElemento(PuntFile, scansione));
  %Identifica la matrice di quantizzazione 0,1,2,3

  if (Pq>1 || Tq>3 || Pq<0 || Tq<0)
    printf("\n*****************ERRORE NELLA LETTURA DEL VALORE Pq e/o Tq
        *****************");

  scansione=scansione+1;

  if (Tq==0){
180 %Matrice di quantizzazione Luminanza Y
    for (int i=0;i<(SIZE*SIZE);i++){
      matriceQuantizzazioneY_ZZ[i]=ByteElemento(PuntFile,scansione+i);
      %è il vettore matrice di Quantizzazione di Luminanza a zig zag
      %ancora non ordinato
    }
    contrY=1;
  }
```

```
190     else if (Tq==1){
          for (int i=0;i<(SIZE*SIZE);i++){
            matriceQuantizzazioneCr_ZZ[i]=ByteElemento(PuntFile,scansione+i); %è
                  il vettore matrice di Quantizzazione di Luminanza a zig zag -
                  ancora non ordinato
          }
          contrCr=1;
        }


        else if (Tq==2){
          for (int i=0;i<(SIZE*SIZE);i++){
200         matriceQuantizzazioneCb_ZZ[i]=ByteElemento(PuntFile,scansione+i); %è
                  il vettore matrice di Quantizzazione di Luminanza a zig zag -
                  ancora non ordinato
          }
          contrCb=1;
        }

        scansione=scansione+64;
        %in alcuni file jpg le matrici di quantizzazione presentano
        %due diverse strutture:
        % 1-  FF DB Lq(1 Byte) Pq(4 bit) Tq(4 bit) Qk (64 Byte)
        %FF DB Lq(1 Byte) Pq(4 bit) Tq(4 bit) Qk (64 Byte)....
210
        % 2-  FF DB Lq(1 Byte) Pq(4 bit) Tq(4 bit) Qk (64 Byte)
        %Lq(1 Byte) Pq(4 bit) Tq(4 bit) Qk (64 Byte).... FFC4%FFC0

        %ovvero il segmento matrice di quantizzazione può
        %non essere preceduto dal Marker FF DB
        %Per cui il for viene eseguito 2 volte tranne se si
        %incontra FFC4 o FFCo (in tal caso si esce)
        %Se si incontra in ogni caso il byte FF, vuol dire che
        %non deve essere effettuata nuovamente la ricerca del
220     %byte di inziio segmento

        if(ByteElemento(PuntFile,scansione)!=0xFF)
          controlloDQT=1; %e quindi non viene effettuata la ricercaDoppia (sono
                sicuramente nel caso 2)

        if( (ByteElemento(PuntFile,scansione)==0xFF && ByteElemento(PuntFile,
              scansione+1)==0xC4) || (ByteElemento(PuntFile,scansione)==0xFF &&
              ByteElemento(PuntFile,scansione+1)==0xC0))
          break;
      }

      scansioneRif=scansione;
230
      unsigned int elementoHeader;
      unsigned int Lf;
      unsigned int pixelY;
      unsigned int pixelX;
      unsigned short Nf;
      int Ci[Nf]; %identifica la componente (Y=1; Cr=2...)
```

```
int Hi[Nf]; %identifica il fattore di compressione orizzontale della
      componente in questione
int Vi[Nf]; %identifica il fattore di compressione verticale della
      componente in questione
int Tqi[Nf]; %identifica la matrice di quantizzazione usata (ovvero il Tq
      del segmento della matrice di quantizzazione)

elementoHeader=ricercaDoppia(PuntFile,dimFile,scansione,0xFF,0xC0);

Lf=unioneHex(ByteElemento(PuntFile,elementoHeader+2),ByteElemento(PuntFile,
      elementoHeader+3) );

pixelY=unioneHex(ByteElemento(PuntFile,elementoHeader+5), ByteElemento(
      PuntFile,elementoHeader+6));
pixelX=unioneHex(ByteElemento(PuntFile,elementoHeader+7), ByteElemento(
      PuntFile,elementoHeader+8));

Nf=ByteElemento(PuntFile,elementoHeader+9);


scansione=elementoHeader+10;

for(unsigned short c=0;c<Nf;c++){
  Ci[c]=ByteElemento(PuntFile,scansione);
  Hi[c]=separaHexPrimo(ByteElemento(PuntFile,scansione+1));
  Vi[c]=separaHexSecondo(ByteElemento(PuntFile,scansione+1));
  Tqi[c]=ByteElemento(PuntFile,scansione+2);
  scansione=scansione+3;
}

if (8+3*Nf!=Lf)
  printf("\n\n\n\n\n ERRORE NEL SEGMENTO FRAME HEADER - Verifica Segmento
       Scan Header NON Riuscita \n\n\n\n\n\n\n\n\n\n");


unsigned short huffComponenti;
unsigned int elementoHuffman;
int Lh;
int Tc,Th;
unsigned short idHuff;

unsigned int Li[4][16];
%int huffSizeRigheNew;

unsigned short kk;
int jj;
unsigned short ll;


int huffSizeRighe=163;
int huffCode[4][huffSizeRighe];
int huffSize[4][huffSizeRighe];
int huffVal[4][huffSizeRighe];
```

```
int lastk;

int kappa;
int code;
int Sl;

int inizio;
int fine;
int indiceIni
      [5][17]={{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1

int indiceFine
      [5][17]={{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1


unsigned short controlloDHT=0;
%Tale parametro ha lo stesso scopo del parametro controlloDQT,
%ovvero serve per decodificare il segmento tabelle di Huffman,
%anche quando non è presente il matrker all'inizio di ognuna
%delle sue 4 parti.
unsigned short sommaLi;

scansione=scansioneRif;
for(huffComponenti=1;huffComponenti<=4;huffComponenti++){
%faccio il for sulle quattro componenti che rappresentano i
%4 vettori bidimensionali legati alle quattro
%tabelle di huffman (Y_DC, Y_AC,Cr_DC,CR_AC)


  if(controlloDHT==0){
    elementoHuffman=ricercaDoppia(PuntFile,dimFile,scansione,0xFF,0xC4);

    %leggo il segmento frame header:
    Lh= unioneHex(ByteElemento(PuntFile,elementoHuffman+2), ByteElemento(
          PuntFile,elementoHuffman+3));

    scansione=elementoHuffman+4;
  }


  else{
    elementoHuffman=scansione+1;
    scansione=elementoHuffman;
  }


  Tc=separaHexPrimo(ByteElemento(PuntFile,scansione));
  Th=separaHexSecondo(ByteElemento(PuntFile,scansione));


  switch (ByteElemento(PuntFile,scansione)) {
    case 0x00:
      idHuff=1;
      break;
```

```
        case 0x10:
          idHuff=2;
          break;

        case 0x01:
          idHuff=3;
          break;

        case 0x11:
          idHuff=4;
          break;

        default:
          printf("ERRORE NELLA LETTURA DEL SEGMENTO HUFFMAN TABLE: Tc e/o Th
                hanno assunto valori non consentiti\n");
          break;
      }
      scansione=scansione+1;



      %Acquisisco Li=BITS --> Numero codici di Huffman di lunghezza i
      sommaLi=0;
      for (unsigned short i=0;i<=15;i++){
        Li[idHuff][i]=ByteElemento(PuntFile,scansione+i);
        sommaLi=sommaLi+Li[idHuff][i];
      %che deve coincidere al lastk generato mediante il diagramma
      %di flusso di pag.51 (standard)

      }
      scansione=scansione+16;



      %GENERAZIONE DELLA TABELLA DELLA DIMENSIONE DEI CODICI
      %diagramma di flusso pag.51 (modificato)- HUFFSIZE
      kk=0;
      ll=0;
      jj=1;
      do{
        while(jj<=Li[idHuff][ll]){
          huffSize[idHuff][kk]=ll+1;
          kk=kk+1;
          jj=jj+1;
        }
        ll=ll+1;
        jj=1;
      }
      while(ll<16);

      huffSize[idHuff][kk]=0;
      lastk=kk;


      %GENERAZIONE DEI CODICI - diagramma di flusso
```

```
        %di pag.52 (standard)- HUFFCODE
390     kappa=0;
        code=0;
        Sl=huffSize[idHuff][0];
        %huffCode[idHuff][lastk];

        do{

          do{
            huffCode[idHuff][kappa]=code;
            code=code+1;
400         kappa=kappa+1;
          }
          while(huffSize[idHuff][kappa]==Sl);


          if(huffSize[idHuff][kappa]==0){
            break;
          }


410       do{
            code=code<<1;
            Sl=Sl+1;
          }
          while (huffSize[idHuff][kappa]!=Sl);



        }
        while (huffSize[idHuff][kappa]==Sl);
420
        huffCode[idHuff][kappa]=0;
        %assegno all'ultimo valore di huffcode, il valore nullo



        %ACQUISISCO I VALORI VIJ - OVVERO HUFFVAL
        for (unsigned short c=0;c<=Lh-20;c++){
          huffVal[idHuff][c]=ByteElemento(PuntFile,elementoHuffman+21+c);
        }
430     huffVal[idHuff][Lh-19]=0; %ultimo valore di HuffVal

        scansione=scansione+sommaLi-1;



      %vettore inizio e fine simboli di lunghezza i (che va da 1 a 16)
      %Indice Ini: l'elemento i-esimo rappresenta l'indice di inizio di
      %vettori di lunghezza i
      %IndiceFine: l'elemento i-esimo rappresenta l'indice finale di
440   %vettori di lunghezza i
      %Così se devo cercare un codice di lunghezza x, posso fare
      %la ricerca nel vettore HuffCode dall'elemento
      %IndiceIni[lunghezza codice ] all'elemento
```

```
%IndiceFine[lunghezza codice]
%Gli elementi di lunghezza i non presenti sono idicati con -1
%L'elemento di indice 0 dei vettori indiceIni e indiceFine
%sono sempre identificati con -1, dato che non esistono
%codici di lunghezza nulla
    for (int c=0;c<=sommaLi;c++){
      fine=huffSize[idHuff][c];
      indiceFine[idHuff][fine]=c;
    }
    indiceFine[idHuff][0]=-1;

    for(int c=sommaLi;c>=0;c--){
      inizio=huffSize[idHuff][c];
      indiceIni[idHuff][inizio]=c;
    }
    indiceIni[idHuff][0]=-1;


   %Verifica sul lastk che deve coincidere alla variabile sommaLi
    if(sommaLi!=lastk)
      printf("\nERRORE NEL SEGMENTO DI HUFFMAN: verifica sommaLi==lastk non
            riuscita\n");


    if(ByteElemento(PuntFile,scansione+1)!=0xFF)
      controlloDHT=1;

}%Chiude il ciclo for (da 1 a 4) del segmento codici di Huffman



int elementoScanHeader=ricercaDoppia(PuntFile,dimFile,scansione,0xFF,0xDA);
int Ls=unioneHex(ByteElemento(PuntFile,elementoScanHeader+2),ByteElemento(
      PuntFile,elementoScanHeader+3));
int Ns=ByteElemento(PuntFile,elementoScanHeader+4);
int Csj[Ns];
%identifica la componente dello scan
int Tdj[Ns];
%codici di Huffman relativi a DC:  Tc=0/Th=0(idHuff=0)
%oppure  Tc=0/Th=1 (idHuff=3)
int Taj[Ns];
%codici di Huffman relativi ad AC: Tc=1/Th=0(idHuff=1)
%oppure  Tc=1/Th=1 (idHuff=2)

scansione=elementoScanHeader+5;

for(unsigned short c=0;c<Ns;c++){
  Csj[c]=ByteElemento(PuntFile,scansione);
  Tdj[c]=separaHexPrimo(ByteElemento(PuntFile,scansione+1));
  Taj[c]=separaHexSecondo(ByteElemento(PuntFile,scansione+1));
  scansione=scansione+2;
}
```

```
    scansione=scansione+3; %PRIMO ELEMENTO DEI DATI

    if(elementoScanHeader+2+Ls!=scansione)
500   printf("ERRORE: VERIFICA segmento Scan Header ---> NON RIUSCITA\n");




    % CALCOLO NUMERO BLOCCHETTI - DEFINIZIONE VARIABILI - INIZIALIZZAZIONE
          MATRICI

    %calcolo Hmax e Vmax - che sono i valori massimi del
    %sottocampionamento orizzontale e verticale
510 %Hmax
    unsigned short indicemax_x=0;
    for(unsigned short i=1; i<Nf; i++){
      if(Hi[i]>Hi[i-1])
        indicemax_x=i;
    }
    unsigned short Hmax=Hi[indicemax_x];

    %Vmax
    unsigned short indicemax_y=0;
520 for(unsigned short i=1; i<Nf; i++){
      if(Hi[i]>Hi[i-1])
        indicemax_y=i;
    }
    unsigned short Vmax=Vi[indicemax_y];



    %calcolo il numero di macroblocchi della componente di luminanza

530 unsigned int macroBlocchiNum_X=ceil((double)pixelX/(Hi[0]*SIZE));
    unsigned int macroBlocchiNum_Y=ceil((double)pixelY/(Vi[0]*SIZE));



    %calcolo il numero di blocchetti per ogni componente:

    unsigned int blocchettiNum_X[Nf];
    unsigned int blocchettiNum_Y[Nf];

540 for (unsigned short i=0;i<Nf;i++){

      blocchettiNum_X[i]=macroBlocchiNum_X*Hi[i];
      blocchettiNum_Y[i]=macroBlocchiNum_Y*Vi[i];
    }


    short precY,precCr,precCb;
    short attualeY,attualeCr,attualeCb;

550
```

```
    unsigned short bytePresi;
    %è la variabile che tiene memoria dei byte prelevati
    %dal file (voglio che siano al massimo 5)

    unsigned short contByteEliminati; %contatore - da azzerare
    unsigned short contInterno; %contatore - da azzerare
    unsigned short dimCoeffDC=5;
    %è la dimensione del vettore vettoreBitTempDC - nel caso peggiore
560 %servono 26 bit (4 byte completi) ---->ceil(26/8)+1;

    unsigned short dimCoeffAC=206 ;
    %è la dimensione del vettore vettoreBitTempDC
    %nel caso peggiore servono ceil(26*63/8)+1=206;

    short indiciByteEliminatiDC[dimCoeffDC];
    %contiene gli indici dei byte eliminati, ovvero dei byte 00
    %che non vengono considerati perchè si trovavano dopo il byte ff;
    short indiciByteEliminatiAC[dimCoeffAC];
570
    short vettoreBitTempDC[SIZE*dimCoeffDC];
    %memorizza i 5 byte che verranno usati poi per la fase di
    %decodifica vera e propria, contiene 40 bit
    short vettoreBitTempAC[SIZE*dimCoeffAC];
    %memorizza i 205 byte che verranno usati poi
    %per la fase di decodifica vera e propria, contiene 1640 bit
    int numerohex; %contiene il valore esadecimale prelevato dal file


580 int blocchettoTemp[SIZE*SIZE];
    %%è il vettore di 64 elementi che contiene i 64 valori
    %temporanei dei coefficienti decodificati
    %(ancora quantizzati e disposti a zig zag)

    int matriceTemp[SIZE][SIZE];
    %è vettore bidimensionale di 64 elementi che contiene i
    %64 valori dei coefficienti dequantizzati e ordinati
    int matriceTempIDCT[SIZE][SIZE];
    int tab;
590 %rappresenta l'indice della tabella di huffman che vado
    %a leggere (tab=0-->Y DC; tab=1---> Y AC; tab=2---> Cr DC;)

    short bitsuccessivi; %è la variabile che memorizza i
    %bitsuccessivi da prendere, ovvero l'HuffVal corrispondente
    %al codice trovato

    unsigned short LunghCodice;
    unsigned int codice; %è il codice preso dal vettoreBitTempDC

600 unsigned short scansionebin=0; %è la variabile che tiene conto
    %del bit di inzio scansione all'interno dei byte presi
    %temporaneamente

    unsigned short scansionebinTemp=0;
```

```
    unsigned short bytesuccessivo; %memorizza i byte che sono
    %utilizzati all'interno di ogni scansione

    unsigned short totbit;
    unsigned short indiceCont;
    unsigned short incrementoscansione;

    int codiceHuffVal;

    int ww;
    int run,size;

    unsigned short controlloIniziale;
    short pres;


    unsigned short marg=4;
    %marg è un valore che va da 1 a un massimo di 7,
    %il valore di marg determina un intorno di analisi
    short soglia_R=100;
    short soglia_G=100;
    short soglia_B=100;
    short differenza=85;

    short vettore_Y[2*SIZE][macroBlocchiNum_X*Hi[0]*SIZE];
    short vettore_Cr[2*SIZE][macroBlocchiNum_X*Hi[0]*SIZE];
    short vettore_Cb[2*SIZE][macroBlocchiNum_X*Hi[0]*SIZE];

    short R[2*SIZE][macroBlocchiNum_X*Hi[0]*SIZE];
    short G[2*SIZE][macroBlocchiNum_X*Hi[0]*SIZE];
    short B[2*SIZE][macroBlocchiNum_X*Hi[0]*SIZE];

    short R_temp[2*SIZE][macroBlocchiNum_X*Hi[0]*SIZE];
    short G_temp[2*SIZE][macroBlocchiNum_X*Hi[0]*SIZE];
    short B_temp[2*SIZE][macroBlocchiNum_X*Hi[0]*SIZE];

    short vettore_Y_plus[4*marg][macroBlocchiNum_X*Hi[0]*SIZE];
    short vettore_Cr_plus[4*marg][macroBlocchiNum_X*Hi[0]*SIZE];
    short vettore_Cb_plus[4*marg][macroBlocchiNum_X*Hi[0]*SIZE];

    short R_plus[4*marg][macroBlocchiNum_X*Hi[0]*SIZE];
    short G_plus[4*marg][macroBlocchiNum_X*Hi[0]*SIZE];
    short B_plus[4*marg][macroBlocchiNum_X*Hi[0]*SIZE];

    short contrDispari=0;
    short indice_moltiplicazione;


    short vettore_Cr_Temp[2*SIZE][2*SIZE];
    short vettore_Cb_Temp[2*SIZE][2*SIZE];


    unsigned int indice_a;
    unsigned int indice_b;
```

```
    unsigned short aa;
    unsigned short bb;

    unsigned short dimConfini=marg*marg*20;

    unsigned short confine_A_S[2][dimConfini];
    unsigned short confine_A_D[2][dimConfini];
    unsigned short confine_B_S[2][dimConfini];
    unsigned short confine_B_D[2][dimConfini];
670

    unsigned short eliminati_A_S[dimConfini];
    unsigned short eliminati_A_D[dimConfini];
    unsigned short eliminati_B_S[dimConfini];;
    unsigned short eliminati_B_D[dimConfini];


    %azzero l vettori di confine:
    for (unsigned short i=0; i<dimConfini;i++){
680   confine_A_S[0][i]=confine_A_D[0][i]=confine_B_S[0][i]=confine_B_D[0][i
          ]=0;
      confine_A_S[1][i]=confine_A_D[1][i]=confine_B_S[1][i]=confine_B_D[1][i
          ]=0;
      eliminati_A_S[i]=eliminati_A_D[i]=eliminati_B_S[i]=eliminati_B_D[i]=0;
    }
    unsigned short vettore_contrasto[4];
    %tale vettore vale 1 quando c'è contrasto
    %in uno dei punti delle direzioni di analisi
    %(ALTO, BASSO, DESTRA, SINISTRA)
    vettore_contrasto[0]=vettore_contrasto[1]=vettore_contrasto[2]=
        vettore_contrasto[3]=0;

690


    unsigned short ind_conf_A_S=0;
    unsigned short ind_conf_A_D=0;
    unsigned short ind_conf_B_S=0;
    unsigned short ind_conf_B_D=0;

    unsigned int indice_riga;
    short blocchi_rettangolari=-1;

700 unsigned short memo;

    unsigned short localizzazione=0;
    unsigned short ingresso=0;



    if (Hi[0]==2) %caso di sottocampionamento 2 1 1
      indice_moltiplicazione=1;
    else{ %in caso sottocampionamento 1 1 1
710   indice_moltiplicazione=2;
    }
```

```
    unsigned short alpha;
    unsigned short beta;
    unsigned short gamma;
    unsigned short delta;
    float A;
    float BI;
720 float ratio;
    short ingressoUnico=0;
    unsigned short margine_coordinate=6;
    unsigned short righe_analisi[4];
    unsigned short righe_soglia[2];
    unsigned short colonne_analisi[10];
    unsigned short blocchi_rettangolari_analisi[4];
    unsigned short blocchi_rettangolari_soglia[2];
    unsigned int somma_R[10]={0,0,0,0,0,0,0,0,0,0};
    unsigned int somma_G[10]={0,0,0,0,0,0,0,0,0,0};
730 unsigned int somma_B[10]={0,0,0,0,0,0,0,0,0,0};
    unsigned int somma_R_bianco[5]={0,0,0,0,0};
    unsigned int somma_G_bianco[5]={0,0,0,0,0};
    unsigned int somma_B_bianco[5]={0,0,0,0,0};

    short R_medio[10];
    short G_medio[10];
    short B_medio[10];

    short R_bianco_soglia[5];
740 short G_bianco_soglia[5];
    short B_bianco_soglia[5];

    unsigned short finalCode[10];


    unsigned int x;
    for(x=0; x<(macroBlocchiNum_X*macroBlocchiNum_Y) ;x++){


750   %printf("\n\n\n\n\n\n\n\n ITERAZIONE NUMERO %d/%d ",x+1,(
          macroBlocchiNum_X*macroBlocchiNum_Y));


      %********** 1 - LUMINANZA Y - COEFFICIENTE DC  **********
      %********** 1 - LUMINANZA Y - COEFFICIENTE DC  **********
      %********** 1 - LUMINANZA Y - COEFFICIENTE DC  **********



      for (unsigned short iterY=1; iterY<=(Hi[0]*Vi[0]);iterY++){
760

        %Nel caso di coefficienti DC lavoro su 5 byte alla volta:
        %ovvero trasformo in binario 5 byte alla volta
        %così che il vettore binario sia costituito da 1 bit
        %per ogni elemento ---> 40 bit
```

```
%variabili da inizializzare
bytePresi=0;
contByteEliminati=0;
contInterno=0;

%inizializzo la variabili indiciByteEliminatiDC pongo a -1
%i valori del vettore indiciByteEliminatiDC
for (unsigned short i=0;i<dimCoeffDC;i++){
%Al massimo posso eliminare 5 byte 00 nel caso
%00 FF 00 FF 00 FF 00 FF 00 FF (caso peggiore)
  indiciByteEliminatiDC[i]=-1;
}



%Pongo a -1 tutti gli elementi del vettore vettoreBitTemp[40]
for(unsigned short i=0;i<SIZE*dimCoeffDC;i++){
  vettoreBitTempDC[i]=-1;
}


while(bytePresi<dimCoeffDC && ( (scansione+contInterno)<(dimFile-1)) ){
%devo prendere almeno 5 byte consecutivi perchè la lunghezza
%massima di bit totali presi (codice+size)può essere al
%massimo di 26 bit (4 byte completi)
%se prendessi solo 4 byte, e devo iniziare a scansionare
%il primo byte dall'ultimo bit allora avrei
%25 bit possibili, un bit in meno del caso peggiore
%il while va avanti fin tanto chè non si prendono 5 byte,
%oppure si esce dal while se non è possibile prendere i
%5 byte successivi perchè il file è terminato.In questo
%caso i byte*8 rimanenti, non presi
%(dei 40 elementi totali) rimangono inizializzati a -1;
  numerohex=ByteElemento(PuntFile,scansione+contInterno);

  %if ---> se ho uno 00 dopo FF
  if(numerohex==0x00 && ByteElemento(PuntFile,scansione+contInterno-1)
        ==0xFF){
    indiciByteEliminatiDC[contByteEliminati]=contInterno;
    contByteEliminati=contByteEliminati+1;
  }

  %else ---> se il byte non è uno 00 dopo un FF,
  %posso mettere gli 8 bit del byte in esame nel
  %vettoreBitTempDC[40]
  else{

    for (unsigned short b=0;b<SIZE;b++) {
      vettoreBitTempDC[7-b+SIZE*bytePresi]=numerohex%2;
      numerohex=floor(numerohex/2);
    }

    bytePresi=bytePresi+1;
  }
```

```
  contInterno=contInterno+1;
}

%printf("\n\n1 - LUMINANZA - COEFFICIENTE DC\n");
%for(int y=1;y<=SIZE*dimCoeffDC;y++){
%printf("%d",vettoreBitTempDC[y-1]);
% if(y%8==0)
%   printf("\t");
%}

%printf("\nbyteeliminati[i]= ");
%for(int y=0;y<dimCoeffDC;y++){
% printf("%d\t",indiciByteEliminatiDC[y]);
%}

% Vado ad effettuare la decodifica vera e propria, ovvero
%ricerco il codice dalla tabella di huffman del coeff.DC di
%luminanza e prendo i bitsuccessivi dati dal valore huffval
%corrispondete(ovvero size). Solo dopo aver capito quanti bit
%totali ho usato (bittotali=codice+huffval) è possibile
%aggiornare il valore della scansione;

%azzero i valori del blocchetto Temporaneo - è in tale
%blocchetto che andrò a registrare volta per volta i 64
%coefficienti (1 DC e 63 AC) delle 3 componenti
%Una volta che il blocchetto sarà completo,
%è possibile copiare i valori nel file opportuno

for (unsigned short a=0;a<SIZE*SIZE;a++){
  blocchettoTemp[a]=0;
}



tab=1;
%tab=1 PERCHE' è IL CASO DI LUMINANZA E COEFFICIENTE DC
LunghCodice=0;
bitsuccessivi=-1; %Numero di bit successivi da prendere
%per decodificare il coeff DC di luminanza,ovvero size
while (bitsuccessivi==-1){
  LunghCodice=LunghCodice+1;
  codice= funzioneDecimale(scansionebin,LunghCodice,&vettoreBitTempDC
      [0]);
  %nella variabile codice è presente il codice di lunghezza
  %Lunghcodice preso dal vettoreBitTempDC a partire
  %dall'elemento scansionebin

  if(indiceIni[tab][LunghCodice]!=-1 && indiceFine[tab][LunghCodice
      ]!=-1){
    for (int g=indiceIni[tab][LunghCodice];g<=indiceFine[tab][
        LunghCodice]; g++){
      if(huffCode[tab][g]==codice){
      %se si entra in questo if, il valore di bitsuccessivi
```

```
            %viene posto pari ad huffVal, che è sicuramente un
            %valore diverso da -1, per cui si uacirà dal ciclo while
               bitsuccessivi=huffVal[tab][g];
               scansionebinTemp=scansionebin+LunghCodice;
               break;
            }
          }
880     }
      }

  %all'uscita dal ciclo sopra:
  %codice rappresenta il codice che è stato preso dal vettore e che
  %coincide ad un codice della tabella dei codici di huffman di
  %luminanza e DC (Huffcode-HuffCodeBin);
  %bitsuccessivi  rappresenta il numero di bit successivi da prendere
  %(a partire dalla fine del codice). i biti presi rappresnetano il
  %valore del coeff. dc(che andrà poi decodificato e dequantizzato)
890 %scansionebin rappresenta l'indice del vettore vettoreBitTempDC
  %che contiene il primo bit successivo al codice preso
  %(ovvero il primo degli huffval bit)
  %printf("\nscansione di partenza=%d",scansione);
  %printf("\t\tscansionebin di partenza=%d",scansionebin);
  %printf("\t\tcodice preso=%d",codice);
  %printf("\t\tLunghezzacodice=%d",LunghCodice);
  %printf("\t\tbitsuccessivi=%d",bitsuccessivi);


900

  %devo aggiungere al valore di DCdiff il valore del DCprec,
  %ovvero DC_corrente=DC_diff+DC_prec
  %(infatti in fase di codifica si ha:
  % DC_diff = DC_corrente-DC_prec )
  %se sono nel primo blocchetto della riga il valore DC_prec
  %è nullo per cui il DCcorrente coincide con il DC_diff
  %altrimenti: DC_corrente=DC_diff+DC_prec


910     %decodifico il coefficiente DC - ovvero inserisco il valore
      %decodificato (ma non ancora dequantizzato) nel primo
      %elemento del blocchettoTemp
      %se il primo bit dei bitsuccessivi presi dopo il codice è 0,
      %faccio il complemento a 1 e la conversione del valore assoluto

      if(x==0 && iterY==1){ %inzia con bit=0 ---> numero negativo
        precY=0;
      }


920


      if(vettoreBitTempDC[scansionebinTemp]==0){
      %inzia con bit=0 ---> numero negativo
        attualeY=proceduraNumNegativo(scansionebinTemp, bitsuccessivi,&
              vettoreBitTempDC[0]);
      }
      else{
```

```
    attualeY=funzioneDecimale(scansionebinTemp,bitsuccessivi,&
          vettoreBitTempDC[0]);

  }
%printf("\nattualeY=%d\n",attualeY);
  blocchettoTemp[0]=attualeY+precY;
%printf("\nblocchettoTemp[0]=%d",blocchettoTemp[0]);
  precY=blocchettoTemp[0];
%printf("\nprecY=%d",precY);


  %aggiorno il valore di scansionebin
  %(bit di partenza dal vettoreBitTempDC che verrà preso in seguito)

  %aggiorno il valore di scansione per prendere
  %il byte seguente dal file ---> definizione di indicebytesucc

  totbit=LunghCodice+bitsuccessivi+scansionebin;
  bytesuccessivo=floor(totbit/SIZE);
  %variabile che tiene conto del numero di byte utilizzati
  %(considerando Lunghcodice, bitsuccessivi e lo scanbin
  %della precedente iterazione)

  scansionebin= totbit-(bytesuccessivo)*SIZE;
%printf("\ntotbit=%d\t\tbytesuccessivo=%d",totbit,bytesuccessivo);

  %Valuto se è stato scartato qualche byte 00 (dopo un byte FF),
  %che si trovava prima dell'elemento scansione
  %a cui sono arrivato

  incrementoscansione=0;
  indiceCont=0;
  while(indiceCont<=bytesuccessivo){
    if(incrementoscansione!=indiciByteEliminatiDC[0] &&
          incrementoscansione!=indiciByteEliminatiDC[1] &&
          incrementoscansione!=indiciByteEliminatiDC[2] &&
          incrementoscansione!=indiciByteEliminatiDC[3] &&
          incrementoscansione!=indiciByteEliminatiDC[4]){
      indiceCont=indiceCont+1;
    }
    incrementoscansione=incrementoscansione+1;
  }
  incrementoscansione=incrementoscansione-1;


  %Per cui, il prossimo vettoreBitTemp (vettoreBitTempAC) partirà
  %dall'elemento scansione incrementato di incrementoscansione
  %dal bit numero scansionebin

  scansione=scansione+incrementoscansione;
  %printf("\t\tincrementoscansione=%d\t\tscansione successiva=%d\t\
        tscansionebin successiva=%d",incrementoscansione, scansione,
        scansionebin);
```

```
%********** 2 - LUMINANAZA Y - COEFFICIENTI AC  **********
%********** 2 - LUMINANAZA Y - COEFFICIENTI AC  **********
%********** 2 - LUMINANAZA Y - COEFFICIENTI AC  **********


%Nel caso di coefficienti AC lavoro su 206 byte alla volta:
%ovvero trasformo in binario 206 byte alla volta
%così che il vettore binario sia costituito da 1 bit
%per ogni elemento ---> 1648 bit

%variabili da inizializzare
bytePresi=0;
contByteEliminati=0;
contInterno=0;
%inizializzo la variabili indiciByteEliminatiAC
%pongo a -1 i valori del vettore indiciByteEliminatiAC
for (unsigned short i=0;i<dimCoeffAC;i++){
%Al massimo posso eliminare 206 byte 00 nel caso
%00 FF 00 FF 00 FF 00 FF 00 FF 00 ff.... (caso peggiore)
  indiciByteEliminatiAC[i]=-1;
}

%Pongo a -1 tutti gli elementi del vettore vettoreBitTemp[1648]
for(unsigned short i=0;i<SIZE*dimCoeffAC;i++){
  vettoreBitTempAC[i]=-1;
}

while(bytePresi<dimCoeffAC && ( (scansione+contInterno)<(dimFile-1)) ){
%devo prendere almeno 206 byte consecutivi perchè la lungh
%massima di bit totali presi (codice+size)può essere al massimo
%di 26 bit (4 byte completi)
%per ogni codice ---> dato che dovrò trattare 63 elementi:
%63*26=1638 bit ovvero 204.75 byte,
%ceil-->205--->+1 viene 206
%il while va avanti fin tanto chè non si prendono 206 byte,
%oppure si esce dal while se non è possibile prendere i 206
%byte successivi perchè il file è terminato. In questo caso
%i byte*8 rimanenti, non presi (dei 1648 elementi totali)
%rimangono inizializzati a -1;

  numerohex=ByteElemento(PuntFile,scansione+contInterno);
  %if ---> se ho uno 00 dopo FF
  if(numerohex==0x00 && ByteElemento(PuntFile,scansione+contInterno-1)
       ==0xFF){
    indiciByteEliminatiAC[contByteEliminati]=contInterno;
    contByteEliminati=contByteEliminati+1;
  }

  %else ---> se il byte non è uno 00 dopo un FF, posso mettere gli 8 bit
         del byte in esame nel vettoreBitTempDC[40]
  else{
```

```
        for (unsigned short b=0;b<SIZE;b++) {
          vettoreBitTempAC[7-b+SIZE*bytePresi]=numerohex%2;
          numerohex=floor(numerohex/2);
        }

        bytePresi=bytePresi+1;
      }

      contInterno=contInterno+1;
    }


    tab=2;
    ww=1;
    LunghCodice=0;

    controlloIniziale=0;


    do{
      codiceHuffVal=-1;

      if(controlloIniziale==1){
        ww=ww+1;
      }
      else if(controlloIniziale==2){
        ww=ww+16;
      }



      %Ricerco il codice nel filebinario che è presente nella tabella,
      %ed estraggo il corrispondente valore di huffval-->(size/run)
      while (codiceHuffVal==-1){
        LunghCodice=LunghCodice+1;
        codice= funzioneDecimale(scansionebin,LunghCodice,&vettoreBitTempAC
              [0]);
        if(indiceIni[tab][LunghCodice]!=-1 && indiceFine[tab][LunghCodice
              ]!=-1){
          for (int g=indiceIni[tab][LunghCodice];g<=indiceFine[tab][
                LunghCodice]; g++){
            if(huffCode[tab][g]==codice){
              codiceHuffVal=huffVal[tab][g];
              scansionebin=scansionebin+LunghCodice;
              %aggiornamento valore scanbin -parte 1/2
              run=separaHexPrimo(codiceHuffVal);
              size=separaHexSecondo(codiceHuffVal);
              break;
            }
          }
        }
      }


      %Controllo errore:
```

```
1080        if(size>10 || size<0 || run>15 || run<0)
              printf("\n\n  ERRORE NELLA LETTURA/DECODIFICA DI RUN SIZE - RUN E/O
                    SIZE HA ASSUNTO UN VALORE NON CONSENTITO \n\n");


            %printf("codice preso=%d\t\tLunghezza codice=%d",codice,LunghCodice);
            %printf("\t\t\t(run/size)=(%d/%d)",run,size);
            codiceHuffVal=0;
            LunghCodice=0;


1090



            %decodifico il coefficiente AC[ww]--->bloccoTem[ww] in esame;

            %caso 1
            if(size!=0){
              controlloIniziale=1;
              ww=ww+run;

              if(vettoreBitTempAC[scansionebin]==0){
1100          %complementoauno e conversione del valore assoluto
                blocchettoTemp[ww]=proceduraNumNegativo(scansionebin, size, &
                      vettoreBitTempAC[0]);
              }
              else {
                blocchettoTemp[ww]=funzioneDecimale(scansionebin,size, &
                      vettoreBitTempAC[0]);
              }
              %printf("\t\tblocchettoTemp[%d]=%d\n",ww,blocchettoTemp[ww]);
              scansionebin=scansionebin+size;
              %aggiornamento valore scanbin - parte 2/2

1110        }


            %caso 2 - size=0 e run=15 --> corsa di 16 zeri- ZRL
            else if(size==0 && run==15){
              controlloIniziale=2;

            }


1120        %caso 3 - size=0 e run=0 ---> fine blocchetto - EOB
            else if (size==0 && run==0){
              break; %esco dal do...while se incontro EOB
            }


          }while(ww<(SIZE*SIZE-1));

          %printf("\nblocchettoTemp=\n");
          %stampaVettoreInt(&blocchettoTemp[0],SIZE*SIZE);
1130
          %ho usato un totale di bit pari a scansionebin:
```

```
totbit=scansionebin;

bytesuccessivo=floor(totbit/SIZE); %byte usati nel vettore
      vettoreBitTemp (che non include i byte eliminati)
scansionebin= totbit-(bytesuccessivo)*SIZE; %bit di inizio scansione
      del vettoreBitTemp successivo

%printf("\n\ntotbit=%d\t\tbytesuccessivo=%d\t\t",totbit,bytesuccessivo)
      ;

%valuto il byte effettivo successivo (quello del file), ovvero
%valuto se è stato scartato qualche byte 00 (dopo un byte FF),
%che si trovava prima dell'elemento scansione a cui sono arrivato
incrementoscansione=0;
indiceCont=0;
pres=0;
while(indiceCont<=bytesuccessivo){
  for (unsigned int i=0 ; i<dimCoeffAC; i++){
    if (incrementoscansione==indiciByteEliminatiAC[i]){
      pres=1;
      break;
    }
  }
  if(pres==0){
    indiceCont=indiceCont+1;
  }

  incrementoscansione=incrementoscansione+1;
  pres=0;
}
incrementoscansione=incrementoscansione-1;

%Per cui, il prossimo vettoreBitTemp (vettroeBitTempDC)
%partirà dall'elemento scansione incrementato di
%incrementoscansione e partirà dal bit numero scansionebin

scansione=scansione+incrementoscansione;
%printf("\t\tincrementoscansione=%d\t\tscansione successiva=%d\t\
      tscansionebin successiva=%d",incrementoscansione, scansione,
      scansionebin);



%Applico il processo di quantizzazione inverso utilizzando
%la matrice matriceQuantizzazioneY_ZZ che il vettore
%di 64 elementi disposti a zig-zag, analogamente a
%blocchettoTemp. Memorizzo i valori in matriceTemp che è un
%vettore bidimensionale che rappresenta gli elementi
%dequantizzati e ordinati

%COEFFICIENTE DC
matriceTemp[0][0]=blocchettoTemp[0]*matriceQuantizzazioneY_ZZ[0];
```

```
%COEFFICIENTI AC
%Riga1
matriceTemp[0][1]=blocchettoTemp[1]*matriceQuantizzazioneY_ZZ[1];
matriceTemp[0][2]=blocchettoTemp[5]*matriceQuantizzazioneY_ZZ[5];
matriceTemp[0][3]=blocchettoTemp[6]*matriceQuantizzazioneY_ZZ[6];
matriceTemp[0][4]=blocchettoTemp[14]*matriceQuantizzazioneY_ZZ[14];
matriceTemp[0][5]=blocchettoTemp[15]*matriceQuantizzazioneY_ZZ[15];
matriceTemp[0][6]=blocchettoTemp[27]*matriceQuantizzazioneY_ZZ[27];
matriceTemp[0][7]=blocchettoTemp[28]*matriceQuantizzazioneY_ZZ[28];
%Riga2
matriceTemp[1][0]=blocchettoTemp[2]*matriceQuantizzazioneY_ZZ[2];
matriceTemp[1][1]=blocchettoTemp[4]*matriceQuantizzazioneY_ZZ[4];
matriceTemp[1][2]=blocchettoTemp[7]*matriceQuantizzazioneY_ZZ[7];
matriceTemp[1][3]=blocchettoTemp[13]*matriceQuantizzazioneY_ZZ[13];
matriceTemp[1][4]=blocchettoTemp[16]*matriceQuantizzazioneY_ZZ[16];
matriceTemp[1][5]=blocchettoTemp[26]*matriceQuantizzazioneY_ZZ[26];
matriceTemp[1][6]=blocchettoTemp[29]*matriceQuantizzazioneY_ZZ[29];
matriceTemp[1][7]=blocchettoTemp[42]*matriceQuantizzazioneY_ZZ[42];
%Riga3
matriceTemp[2][0]=blocchettoTemp[3]*matriceQuantizzazioneY_ZZ[3];
matriceTemp[2][1]=blocchettoTemp[8]*matriceQuantizzazioneY_ZZ[8];
matriceTemp[2][2]=blocchettoTemp[12]*matriceQuantizzazioneY_ZZ[12];
matriceTemp[2][3]=blocchettoTemp[17]*matriceQuantizzazioneY_ZZ[17];
matriceTemp[2][4]=blocchettoTemp[25]*matriceQuantizzazioneY_ZZ[25];
matriceTemp[2][5]=blocchettoTemp[30]*matriceQuantizzazioneY_ZZ[30];
matriceTemp[2][6]=blocchettoTemp[41]*matriceQuantizzazioneY_ZZ[41];
matriceTemp[2][7]=blocchettoTemp[43]*matriceQuantizzazioneY_ZZ[43];
%Riga4
matriceTemp[3][0]=blocchettoTemp[9]*matriceQuantizzazioneY_ZZ[9];
matriceTemp[3][1]=blocchettoTemp[11]*matriceQuantizzazioneY_ZZ[11];
matriceTemp[3][2]=blocchettoTemp[18]*matriceQuantizzazioneY_ZZ[18];
matriceTemp[3][3]=blocchettoTemp[24]*matriceQuantizzazioneY_ZZ[24];
matriceTemp[3][4]=blocchettoTemp[31]*matriceQuantizzazioneY_ZZ[31];
matriceTemp[3][5]=blocchettoTemp[40]*matriceQuantizzazioneY_ZZ[40];
matriceTemp[3][6]=blocchettoTemp[44]*matriceQuantizzazioneY_ZZ[44];
matriceTemp[3][7]=blocchettoTemp[53]*matriceQuantizzazioneY_ZZ[53];
%Riga5
matriceTemp[4][0]=blocchettoTemp[10]*matriceQuantizzazioneY_ZZ[10];
matriceTemp[4][1]=blocchettoTemp[19]*matriceQuantizzazioneY_ZZ[19];
matriceTemp[4][2]=blocchettoTemp[23]*matriceQuantizzazioneY_ZZ[23];
matriceTemp[4][3]=blocchettoTemp[32]*matriceQuantizzazioneY_ZZ[32];
matriceTemp[4][4]=blocchettoTemp[39]*matriceQuantizzazioneY_ZZ[39];
matriceTemp[4][5]=blocchettoTemp[45]*matriceQuantizzazioneY_ZZ[45];
matriceTemp[4][6]=blocchettoTemp[52]*matriceQuantizzazioneY_ZZ[52];
matriceTemp[4][7]=blocchettoTemp[54]*matriceQuantizzazioneY_ZZ[54];
%Riga
matriceTemp[5][0]=blocchettoTemp[20]*matriceQuantizzazioneY_ZZ[20];
matriceTemp[5][1]=blocchettoTemp[22]*matriceQuantizzazioneY_ZZ[22];
matriceTemp[5][2]=blocchettoTemp[33]*matriceQuantizzazioneY_ZZ[33];
matriceTemp[5][3]=blocchettoTemp[38]*matriceQuantizzazioneY_ZZ[38];
matriceTemp[5][4]=blocchettoTemp[46]*matriceQuantizzazioneY_ZZ[46];
matriceTemp[5][5]=blocchettoTemp[51]*matriceQuantizzazioneY_ZZ[51];
matriceTemp[5][6]=blocchettoTemp[55]*matriceQuantizzazioneY_ZZ[55];
matriceTemp[5][7]=blocchettoTemp[60]*matriceQuantizzazioneY_ZZ[60];
%Riga7
```

```
matriceTemp[6][0]=blocchettoTemp[21]*matriceQuantizzazioneY_ZZ[21];
matriceTemp[6][1]=blocchettoTemp[34]*matriceQuantizzazioneY_ZZ[34];
matriceTemp[6][2]=blocchettoTemp[37]*matriceQuantizzazioneY_ZZ[37];
matriceTemp[6][3]=blocchettoTemp[47]*matriceQuantizzazioneY_ZZ[47];
matriceTemp[6][4]=blocchettoTemp[50]*matriceQuantizzazioneY_ZZ[50];
matriceTemp[6][5]=blocchettoTemp[56]*matriceQuantizzazioneY_ZZ[56];
matriceTemp[6][6]=blocchettoTemp[59]*matriceQuantizzazioneY_ZZ[59];
matriceTemp[6][7]=blocchettoTemp[61]*matriceQuantizzazioneY_ZZ[61];
%Riga8
matriceTemp[7][0]=blocchettoTemp[35]*matriceQuantizzazioneY_ZZ[35];
matriceTemp[7][1]=blocchettoTemp[36]*matriceQuantizzazioneY_ZZ[36];
matriceTemp[7][2]=blocchettoTemp[48]*matriceQuantizzazioneY_ZZ[48];
matriceTemp[7][3]=blocchettoTemp[49]*matriceQuantizzazioneY_ZZ[49];
matriceTemp[7][4]=blocchettoTemp[57]*matriceQuantizzazioneY_ZZ[57];
matriceTemp[7][5]=blocchettoTemp[58]*matriceQuantizzazioneY_ZZ[58];
matriceTemp[7][6]=blocchettoTemp[62]*matriceQuantizzazioneY_ZZ[62];
matriceTemp[7][7]=blocchettoTemp[63]*matriceQuantizzazioneY_ZZ[63];



%Applico la IDCT
%printf("\n\nBlocchetto 8*8 Luminanza a cui ho applicato la DCT inversa
      (blocchetto Finale):\n\n\n");
IDCTBlocchetto(&matriceTemp[0][0],&matriceTempIDCT[0][0]);
%stampaVettoreInt(&matriceTempIDCT[0][0],SIZE*SIZE);


if(Hi[0]*Vi[0]==4){
%caso di sottocampionamento 2 1 1
  if(iterY==1){
    for (aa=0;aa<SIZE;aa++){
      for (bb=0;bb<SIZE;bb++){
        indice_b=((x%(blocchettiNum_X[1]))*Hmax)*SIZE;
        %(il % non è un commento ma è l'operatore modulo)
        vettore_Y[aa][bb+indice_b]=matriceTempIDCT[aa][bb];

      }

    }
  }


  else if(iterY==2){

    for (aa=0;aa<SIZE;aa++){
      for (bb=0;bb<SIZE;bb++){
        indice_b=((x%(blocchettiNum_X[1]))*Hmax+1)*SIZE;
        %(il % non è un commento ma è l'operatore modulo)
        vettore_Y[aa][bb+indice_b]=matriceTempIDCT[aa][bb];

      }
    }
  }

  else if(iterY==3){
```

```
            for (aa=0;aa<SIZE;aa++){
              for (bb=0;bb<SIZE;bb++){
                indice_a=SIZE;
                indice_b=((x%(blocchettiNum_X[1]))*Hmax)*SIZE;
                %(il % non è un commento ma è l'operatore modulo)
                vettore_Y[aa+indice_a][bb+indice_b]=matriceTempIDCT[aa][bb];
              }
            }
          }

          else if(iterY==4){

            for (aa=0;aa<SIZE;aa++){
              for (bb=0;bb<SIZE;bb++){
                indice_a=SIZE;
                indice_b=((x%(blocchettiNum_X[1]))*Hmax+1)*SIZE;
                %(il % non è un commento ma è l'operatore modulo)
                vettore_Y[aa+indice_a][bb+indice_b]=matriceTempIDCT[aa][bb];
              }
            }
          }

        }%chiude l'if (Hi[0]*Vi[0]==4)
        %caso di sottocampionamento 2,1,1

        else if(Hi[0]*Vi[0]==1){
        %caso di sottocampionamento 1 1 1

          %con le seguenti istruzioni if/else, utilizzo la variabile
          %contrDispari per capire se sono nella prima o
          %nella seconda riga


          if( x%(macroBlocchiNum_X)==0 && x!=0){
          %(il % non è un commento ma è l'operatore modulo)
            if(contrDispari==0)
              contrDispari=1;
            else{
              contrDispari=0;
            }


          }

          %printf("\n\ncontrDispari=%d\n\n\n",contrDispari);


          if(contrDispari==0){
            for (aa=0;aa<SIZE;aa++){
              for (bb=0;bb<SIZE;bb++){
                indice_b=((x%(blocchettiNum_X[1]))*Hmax)*SIZE;
                %(il % non è un commento ma è l'operatore modulo)
                vettore_Y[aa][bb+indice_b]=matriceTempIDCT[aa][bb];

              }
```

```
            }

          }
          else{ %contrDispari=1...
            for (aa=0;aa<SIZE;aa++){
              for (bb=0;bb<SIZE;bb++){
                indice_a=SIZE;
                indice_b=((x%(blocchettiNum_X[1]))*Hmax)*SIZE;
                %(il % non è un commento ma è l'operatore modulo)
                vettore_Y[aa+indice_a][bb+indice_b]=matriceTempIDCT[aa][bb];
              }

            }

          }
        }

      }%chiude il for interno dei Hi[0]*Vi[0] blocchetti di luminanza



      %********** 3 - CROMINANZA CR - COEFFICIENTE DC  **********
      %********** 3 - CROMINANZA CR - COEFFICIENTE DC  **********
      %********** 3 - CROMINANZA CR - COEFFICIENTE DC  **********

      for (unsigned short iterCr=1; iterCr<=(Hi[1]*Vi[1]);iterCr++){


        %Nel caso di coefficienti DC lavoro su 5 byte alla volta:
        %ovvero trasformo in binario 5 byte alla volta
        %così che il vettore binario sia costituito da 1 bit
        %per ogni elemento ----> 40 bit


        %variabili da inizializzare
        bytePresi=0;
        contByteEliminati=0;
        contInterno=0;

        %inizializzo la variabili indiciByteEliminatiDC pongo a -1 i valori del
              vettore indiciByteEliminatiDC
        for (unsigned short i=0;i<dimCoeffDC;i++){ %Al massimo posso eliminare
              5 byte 00 nel caso 00 FF 00 FF 00 FF 00 FF 00 FF (caso peggiore)
          indiciByteEliminatiDC[i]=-1;
        }


        %Pongo a -1 tutti gli elementi del vettore vettoreBitTemp[40]
        for(unsigned short i=0;i<SIZE*dimCoeffDC;i++){
          vettoreBitTempDC[i]=-1;
        }


        while(bytePresi<dimCoeffDC && ( (scansione+contInterno)<(dimFile-1)) ){
```

```
        %devo prendere almeno 5 byte consecutivi perchè la lunghezza
1400    %massima di bit totali presi (codice+size)può essere al massimo
        %di 26 bit (4 byte completi) se prendessi solo 4 byte, e devo
        %iniziare a scansionare il primo byte dall'ultimo bit allora
        %avrei 25 bit possibili, un bit in meno del caso peggiore
        %il while va avanti fin tanto chè non si prendono 5 byte,
        %oppure si esce dal while se non è possibile prendere i 5 byte
        %successivi perchè il file è terminato.In questo caso
        %i byte*8 rimanenti, non presi (dei 40 elementi totali)
        %rimangono inizializzati a -1;

1410      numerohex=ByteElemento(PuntFile,scansione+contInterno);

          %if ---> se ho uno 00 dopo FF
          if(numerohex==0x00 && ByteElemento(PuntFile,scansione+contInterno-1)
                ==0xFF){
            indiciByteEliminatiDC[contByteEliminati]=contInterno;
            contByteEliminati=contByteEliminati+1;
          }

          %else ---> se il byte non è uno 00 dopo un FF,
          %posso mettere gli 8 bit del byte in esame nel vettoreBitTempDC[40]
1420      else{

            for (unsigned short b=0;b<SIZE;b++) {
              vettoreBitTempDC[7-b+SIZE*bytePresi]=numerohex%2;
              %(il % non è un commento ma è l'operatore modulo)
              numerohex=floor(numerohex/2);
            }

            bytePresi=bytePresi+1;
          }
1430
          contInterno=contInterno+1;
        }

        %printf("\n\n3 - CROMINANZA CR - COEFFICIENTE DC\n");
        %for(int y=1;y<=SIZE*dimCoeffDC;y++){
        %printf("%d",vettoreBitTempDC[y-1]);
        % if(y%8==0)
        %   printf("\t");
        %}
1440
        %printf("\nbyteeliminati[i]= ");
        %for(int y=0;y<dimCoeffDC;y++){
        % printf("%d\t",indiciByteEliminatiDC[y]);
        %}



        %Vado ad effettuare la decodifica vera e propria, ovvero
        %ricerco il codice dalla tabella di huffman del coeff.DC di
1450    %luminanza e prendo i bitsuccessivi dati dal valore huffval
        %corrispondete(ovvero size). Solo dopo aver capito quanti
        %bit totali ho usato: (bittotali=codice+huffval)
```

```
%è possibile aggiornare il valore della scansione;

%azzero i valori del blocchetto Temporaneo
% è in tale blocchetto che andrò a registrare volta per volta
%i 64 coefficienti (1 DC e 63 AC) delle 3 componenti
%Una volta che il blocchetto sarà completo, è possibile
%copiare i valori nel file opportuno

for (unsigned short a=0;a<SIZE*SIZE;a++){
  blocchettoTemp[a]=0;
}


tab=3;
%tab=3 PERCHE' è IL CASO DI CROMINANZA E COEFFICIENTE DC
LunghCodice=0;
bitsuccessivi=-1;
%Numero di bit successivi da prendere per decodificare
%il coeff DC di luminanza,ovvero size
while (bitsuccessivi==-1){
  LunghCodice=LunghCodice+1;
  codice= funzioneDecimale(scansionebin,LunghCodice,&vettoreBitTempDC
      [0]);
  %nella variabile codice è presente il codice di lunghezza
  %Lunghcodice preso dal vettoreBitTempDC a partire
  %dall'elemento scansionebin

  if(indiceIni[tab][LunghCodice]!=-1 && indiceFine[tab][LunghCodice
      ]!=-1){
    for (int g=indiceIni[tab][LunghCodice];g<=indiceFine[tab][
        LunghCodice]; g++){
      if(huffCode[tab][g]==codice){
      %se si entra in questo if, il valore di bitsuccessivi
      %viene posto pari ad huffVal, che è sicuramente un
      %valore diverso da -1, per cui si uacirà dal ciclo while
        bitsuccessivi=huffVal[tab][g];
        scansionebinTemp=scansionebin+LunghCodice;
        break;
      }
    }
  }
}
%all'uscita dal ciclo sopra:

%codice rappresenta il codice che è stato preso dal vettore e
%che coincide ad un codice della tabella dei codici di huffman
%di luminanza e DC (Huffcode-HuffCodeBin);

%bitsuccessivi  rappresenta il numero di bit successivi da
%prendere (a partire dalla fine del codice). i biti presi
%rappresnetano il valore del coefficiente dc
%(che andrà poi decodificato e dequantizzato)

%scansionebin rappresenta l'indice del vettore
%vettoreBitTempDC che contiene il primo bit successivo
```

```
        %al codice preso (ovvero il primo degli huffval bit)
        %printf("\nscansione di partenza=%d",scansione);
        %printf("\t\tscansionebin di partenza=%d",scansionebin);
        %printf("\t\tcodice preso=%d",codice);
        %printf("\t\tLunghezzacodice=%d",LunghCodice);
1510    %printf("\t\tbitsuccessivi=%d",bitsuccessivi);



        %devo aggiungere al valore di DCdiff il valore del DCprec,
        %ovvero DC_corrente=DC_diff+DC_prec
        % (infatti in fase di codifica si ha:
        %DC_diff = DC_corrente-DC_prec )

        % se sono nel primo blocchetto della riga il valore DC_prec
1520    %è nullo per cui il DCcorrente coincide con il DC_diff
        %altrimenti: DC_corrente=DC_diff+DC_prec

        if(x==0 && iterCr==1){ %inzia con bit=0 ---> numero negativo
          precCr=0;
        }

        %decodifico il coefficiente DC - ovvero inserisco il valore
        %decodificato (ma non ancora dequantizzato) nel primo elemento
        %del blocchettoTemp; se il primo bit dei bitsuccessivi presi
1530    %dopo il codice è 0 faccio il complemento a 1 e la conversione
        %del valore assoluto

        if(vettoreBitTempDC[scansionebinTemp]==0){
        %inzia con bit=0 ---> numero negativo
          attualeCr=proceduraNumNegativo(scansionebinTemp, bitsuccessivi,&
                vettoreBitTempDC[0]);
        }
        else{
          attualeCr=funzioneDecimale(scansionebinTemp,bitsuccessivi,&
                vettoreBitTempDC[0]);

1540    }

        blocchettoTemp[0]=attualeCr+precCr;

        precCr=blocchettoTemp[0];


        %printf("\t\tblocchettoTemp[0]=%d",blocchettoTemp[0]);

        %aggiorno il valore di scansionebin
1550    %(bit di partenza dal vettoreBitTempDC che verrà
        %preso in seguito)

        %aggiorno il valore di scansione per prendere il byte
        %seguenete dal file ---> definizione di indicebytesucc
```

```
totbit=LunghCodice+bitsuccessivi+scansionebin;

bytesuccessivo=floor(totbit/SIZE);
%variabile che tiene conto del numero di byte utilizzati
%(considerando Lunghcodice, bitsuccessivi e lo scanbin
%della precedente iterazione)
scansionebin= totbit-(bytesuccessivo)*SIZE;
%printf("\ntotbit=%d\t\tbytesuccessivo=%d",totbit,bytesuccessivo);

%Valuto se è stato scartato qualche byte 00 (dopo un byte FF),
%che si trovava prima dell'elemento scansione a cui
%sono arrivato

incrementoscansione=0;
indiceCont=0;
while(indiceCont<=bytesuccessivo){
  if(incrementoscansione!=indiciByteEliminatiDC[0] &&
        incrementoscansione!=indiciByteEliminatiDC[1] &&
        incrementoscansione!=indiciByteEliminatiDC[2] &&
        incrementoscansione!=indiciByteEliminatiDC[3] &&
        incrementoscansione!=indiciByteEliminatiDC[4]){
    indiceCont=indiceCont+1;
  }
  incrementoscansione=incrementoscansione+1;
}
incrementoscansione=incrementoscansione-1;


%Per cui, il prossimo vettoreBitTemp (vettoreBitTempAC) partirà
%dall'elemento scansione incrementato di incrementoscansione
%dal bit numero scansionebin

scansione=scansione+incrementoscansione;
%printf("\t\tincrementoscansione=%d\t\tscansione successiva=%d\t\
      tscansionebin successiva=%d",incrementoscansione, scansione,
      scansionebin);


%********** 4 - CROMINANZA CR - COEFFICIENTI AC  **********
%********** 4 - CROMINANZA CR - COEFFICIENTI AC  **********
%********** 4 - CROMINANZA CR - COEFFICIENTI AC  **********


%Nel caso di coefficienti AC lavoro su 206 byte alla volta:
%ovvero trasformo in binario 206 byte alla volta
%così che il vettore binario sia costituito da 1 bit
%per ogni elemento ----> 1648 bit

%variabili da inizializzare
bytePresi=0;
contByteEliminati=0;
contInterno=0;
```

```
%inizializzo la variabili indiciByteEliminatiAC
%pongo a -1 i valori del vettore indiciByteEliminatiAC
for (unsigned short i=0;i<dimCoeffAC;i++){
%Al massimo posso eliminare 206 byte 00 nel caso
%00 FF 00 FF 00 FF 00 FF 00 FF 00 ff.... (caso peggiore)
  indiciByteEliminatiAC[i]=-1;
}

%Pongo a -1 tutti gli elementi del vettore vettoreBitTemp[1648]
for(unsigned short i=0;i<SIZE*dimCoeffAC;i++){
  vettoreBitTempAC[i]=-1;
}


while(bytePresi<dimCoeffAC && ( (scansione+contInterno)<(dimFile-1)) ){
  %devo prendere almeno 206 byte consecutivi perchè la lungh
  %massima di bit totali presi (codice+size)può essere al
  %massimo di 26 bit (4 byte completi)
  %per ogni codice ---> dato che dovrò trattare 63 elementi:
  %63*26=1638 bit ovvero 204.75 byte, ceil-->205->+1 viene 206
  %il while va avanti fin tanto chè non si prendono 206 byte,
  % oppure si esce dal while se non è possibile prendere i
  %206 byte successivi perchè il file è terminato.
  %In questo caso i byte*8 rimanenti, non presi
  %(dei 1648 elementi totali) rimangono inizializzati a -1;

  numerohex=ByteElemento(PuntFile,scansione+contInterno);
  %if ---> se ho uno 00 dopo FF
  if(numerohex==0x00 && ByteElemento(PuntFile,scansione+contInterno-1)
       ==0xFF){
    indiciByteEliminatiAC[contByteEliminati]=contInterno;
    contByteEliminati=contByteEliminati+1;
  }

  %else ---> se il byte non è uno 00 dopo un FF, posso
  %mettere gli 8 bit del byte in esame nel vettoreBitTempDC[40]
  else{
    for (unsigned short b=0;b<SIZE;b++) {
      vettoreBitTempAC[7-b+SIZE*bytePresi]=numerohex%2;
      numerohex=floor(numerohex/2);
    }
    bytePresi=bytePresi+1;
  }
  contInterno=contInterno+1;
}

%printf("\n\n4 - CROMINANZA CR - COEFFICIENTI AC\n");
%for(int y=1;y<=SIZE*dimCoeffAC;y++){
%printf("%d",vettoreBitTempAC[y-1]);
%if(y%8==0)
% printf("\t");
%}
```

```
%printf("\nbyteeliminati[i]= ");
%for(int y=0;y<10;y++){ %for(int y=0;y<dimCoeffAC;y++){
% printf("%d\t",indiciByteEliminatiAC[y]);
%}

%printf("\nscansione di partenza=%d\t\tscansionebin di partenza=%d\n\n
      ",scansione,scansionebin);


tab=4;
ww=1;
LunghCodice=0;

controlloIniziale=0;
int ww=1;

do{
  codiceHuffVal=-1;

  if(controlloIniziale==1){
    ww=ww+1;
  }
  else if(controlloIniziale==2){
    ww=ww+16;
  }


  %Ricerco il codice nel filebinario che è presente nella
  %tabella, ed estraggo il corrispondente valore di
  %huffval-->(size/run)
  while (codiceHuffVal==-1){
    LunghCodice=LunghCodice+1;
    codice= funzioneDecimale(scansionebin,LunghCodice,&vettoreBitTempAC
          [0]);
    if(indiceIni[tab][LunghCodice]!=-1 && indiceFine[tab][LunghCodice
          ]!=-1){
      for (int g=indiceIni[tab][LunghCodice];g<=indiceFine[tab][
            LunghCodice]; g++){
        if(huffCode[tab][g]==codice){
          codiceHuffVal=huffVal[tab][g];
          scansionebin=scansionebin+LunghCodice;
           %aggiornamento valore scanbin - parte 1/2
          run=separaHexPrimo(codiceHuffVal);
          size=separaHexSecondo(codiceHuffVal);
          break;
        }
      }
    }
  }


  %Controllo errore:
  if(size>10 || size<0 || run>15 || run<0)
    printf("\n\n********** ERRORE NELLA LETTURA/DECODIFICA DI RUN
          SIZE - RUN E/O SIZE HA ASSUNTO UN VALORE NON CONSENTITO
```

```
                **********\n\n");


            %printf("codice preso=%d\t\tLunghezza codice=%d",codice,LunghCodice);
            %printf("\t\t\t(run/size)=(%d/%d)",run,size);
            codiceHuffVal=0;
            LunghCodice=0;


1720
            %decodifico il coefficiente AC[ww]---> bloccoTem[ww] in esame;

            %caso 1 tipico
            if(size!=0){
              controlloIniziale=1;
              ww=ww+run;

              if(vettoreBitTempAC[scansionebin]==0){
              %complementoauno e conversione del valore assoluto
1730            blocchettoTemp[ww]=proceduraNumNegativo(scansionebin, size,&
                    vettoreBitTempAC[0]);
              }
              else {
                blocchettoTemp[ww]=funzioneDecimale(scansionebin,size,&
                    vettoreBitTempAC[0]);
              }
              %printf("\t\tblocchettoTemp[%d]=%d\n",ww,blocchettoTemp[ww]);
              scansionebin=scansionebin+size;
              %aggiornamento valore scanbin - parte 2/2

            }
1740

            %caso 2 - size=0 e run=15 --> corsa di 16 zeri- ZRL
            else if(size==0 && run==15){
              controlloIniziale=2;

            }

            %caso 3 - size=0 e run=0 ---> fine blocchetto - EOB
            else if (size==0 && run==0){
1750          break; %esco dal do...while se incontro EOB
            }


          }while(ww<(SIZE*SIZE-1));

          %printf("\nblocchettoTemp=\n");
          %stampaVettoreInt(&blocchettoTemp[0],SIZE*SIZE);

          %ho usato un totale di bit pari a scansionebin:
1760
          totbit=scansionebin;
```

```
bytesuccessivo=floor(totbit/SIZE);
%byte usati nel vettore vettoreBitTemp (che non include i byte
      eliminati)
scansionebin= totbit-(bytesuccessivo)*SIZE;
%bit di inizio scansione del vettoreBitTemp successivo

%printf("\n\ntotbit=%d\t\tbytesuccessivo=%d\t\t",totbit,bytesuccessivo)
      ;

%valuto il byte effettivo successivo (quello del file), ovvero
%valuto se è stato scartato qualche byte 00 (dopo un byte FF)
%,che si trovava prima dell'elemento scansione
%a cui sono arrivato
incrementoscansione=0;
indiceCont=0;
pres=0;
while(indiceCont<=bytesuccessivo){
  for (unsigned int i=0 ; i<dimCoeffAC; i++){
    if (incrementoscansione==indiciByteEliminatiAC[i]){
      pres=1;
      break;
    }
  }
  if(pres==0){
    indiceCont=indiceCont+1;
  }

  incrementoscansione=incrementoscansione+1;
  pres=0;
}
incrementoscansione=incrementoscansione-1;

%Per cui, il prossimo vettoreBitTemp(vettroeBitTempDC)
%partirà dall'elemento scansione incrementato di
%incrementoscansione  e partirà dal bit
%numero scansionebin

scansione=scansione+incrementoscansione;
%printf("\t\tincrementoscansione=%d\t\tscansione successiva=%d\t\
      tscansionebin successiva=%d",incrementoscansione, scansione,
      scansionebin);


%Applico il processo di quantizzazione inverso utilizzando la
%matr matriceQuantizzazioneY_ZZ che il vettore di 64 elementi
%disposti a zig-zag, analogamente a blocchettoTemp.
%Memorizzo i valori in matriceTemp che è una vettore
%bidimensionale che rappresenta gli elementi
%dequantizzati e ordinati

%COEFFICIENTE DC
matriceTemp[0][0]=blocchettoTemp[0]*matriceQuantizzazioneCr_ZZ[0];
```

```
%COEFFICIENTI AC
%Riga1
matriceTemp[0][1]=blocchettoTemp[1]*matriceQuantizzazioneCr_ZZ[1];
matriceTemp[0][2]=blocchettoTemp[5]*matriceQuantizzazioneCr_ZZ[5];
matriceTemp[0][3]=blocchettoTemp[6]*matriceQuantizzazioneCr_ZZ[6];
matriceTemp[0][4]=blocchettoTemp[14]*matriceQuantizzazioneCr_ZZ[14];
matriceTemp[0][5]=blocchettoTemp[15]*matriceQuantizzazioneCr_ZZ[15];
matriceTemp[0][6]=blocchettoTemp[27]*matriceQuantizzazioneCr_ZZ[27];
matriceTemp[0][7]=blocchettoTemp[28]*matriceQuantizzazioneCr_ZZ[28];
%Riga2
matriceTemp[1][0]=blocchettoTemp[2]*matriceQuantizzazioneCr_ZZ[2];
matriceTemp[1][1]=blocchettoTemp[4]*matriceQuantizzazioneCr_ZZ[4];
matriceTemp[1][2]=blocchettoTemp[7]*matriceQuantizzazioneCr_ZZ[7];
matriceTemp[1][3]=blocchettoTemp[13]*matriceQuantizzazioneCr_ZZ[13];
matriceTemp[1][4]=blocchettoTemp[16]*matriceQuantizzazioneCr_ZZ[16];
matriceTemp[1][5]=blocchettoTemp[26]*matriceQuantizzazioneCr_ZZ[26];
matriceTemp[1][6]=blocchettoTemp[29]*matriceQuantizzazioneCr_ZZ[29];
matriceTemp[1][7]=blocchettoTemp[42]*matriceQuantizzazioneCr_ZZ[42];
%Riga3
matriceTemp[2][0]=blocchettoTemp[3]*matriceQuantizzazioneCr_ZZ[3];
matriceTemp[2][1]=blocchettoTemp[8]*matriceQuantizzazioneCr_ZZ[8];
matriceTemp[2][2]=blocchettoTemp[12]*matriceQuantizzazioneCr_ZZ[12];
matriceTemp[2][3]=blocchettoTemp[17]*matriceQuantizzazioneCr_ZZ[17];
matriceTemp[2][4]=blocchettoTemp[25]*matriceQuantizzazioneCr_ZZ[25];
matriceTemp[2][5]=blocchettoTemp[30]*matriceQuantizzazioneCr_ZZ[30];
matriceTemp[2][6]=blocchettoTemp[41]*matriceQuantizzazioneCr_ZZ[41];
matriceTemp[2][7]=blocchettoTemp[43]*matriceQuantizzazioneCr_ZZ[43];
%Riga4
matriceTemp[3][0]=blocchettoTemp[9]*matriceQuantizzazioneCr_ZZ[9];
matriceTemp[3][1]=blocchettoTemp[11]*matriceQuantizzazioneCr_ZZ[11];
matriceTemp[3][2]=blocchettoTemp[18]*matriceQuantizzazioneCr_ZZ[18];
matriceTemp[3][3]=blocchettoTemp[24]*matriceQuantizzazioneCr_ZZ[24];
matriceTemp[3][4]=blocchettoTemp[31]*matriceQuantizzazioneCr_ZZ[31];
matriceTemp[3][5]=blocchettoTemp[40]*matriceQuantizzazioneCr_ZZ[40];
matriceTemp[3][6]=blocchettoTemp[44]*matriceQuantizzazioneCr_ZZ[44];
matriceTemp[3][7]=blocchettoTemp[53]*matriceQuantizzazioneCr_ZZ[53];
%Riga5
matriceTemp[4][0]=blocchettoTemp[10]*matriceQuantizzazioneCr_ZZ[10];
matriceTemp[4][1]=blocchettoTemp[19]*matriceQuantizzazioneCr_ZZ[19];
matriceTemp[4][2]=blocchettoTemp[23]*matriceQuantizzazioneCr_ZZ[23];
matriceTemp[4][3]=blocchettoTemp[32]*matriceQuantizzazioneCr_ZZ[32];
matriceTemp[4][4]=blocchettoTemp[39]*matriceQuantizzazioneCr_ZZ[39];
matriceTemp[4][5]=blocchettoTemp[45]*matriceQuantizzazioneCr_ZZ[45];
matriceTemp[4][6]=blocchettoTemp[52]*matriceQuantizzazioneCr_ZZ[52];
matriceTemp[4][7]=blocchettoTemp[54]*matriceQuantizzazioneCr_ZZ[54];
%Riga
matriceTemp[5][0]=blocchettoTemp[20]*matriceQuantizzazioneCr_ZZ[20];
matriceTemp[5][1]=blocchettoTemp[22]*matriceQuantizzazioneCr_ZZ[22];
matriceTemp[5][2]=blocchettoTemp[33]*matriceQuantizzazioneCr_ZZ[33];
matriceTemp[5][3]=blocchettoTemp[38]*matriceQuantizzazioneCr_ZZ[38];
matriceTemp[5][4]=blocchettoTemp[46]*matriceQuantizzazioneCr_ZZ[46];
matriceTemp[5][5]=blocchettoTemp[51]*matriceQuantizzazioneCr_ZZ[51];
matriceTemp[5][6]=blocchettoTemp[55]*matriceQuantizzazioneCr_ZZ[55];
matriceTemp[5][7]=blocchettoTemp[60]*matriceQuantizzazioneCr_ZZ[60];
%Riga7
```

```
matriceTemp[6][0]=blocchettoTemp[21]*matriceQuantizzazioneCr_ZZ[21];
matriceTemp[6][1]=blocchettoTemp[34]*matriceQuantizzazioneCr_ZZ[34];
matriceTemp[6][2]=blocchettoTemp[37]*matriceQuantizzazioneCr_ZZ[37];
matriceTemp[6][3]=blocchettoTemp[47]*matriceQuantizzazioneCr_ZZ[47];
matriceTemp[6][4]=blocchettoTemp[50]*matriceQuantizzazioneCr_ZZ[50];
matriceTemp[6][5]=blocchettoTemp[56]*matriceQuantizzazioneCr_ZZ[56];
matriceTemp[6][6]=blocchettoTemp[59]*matriceQuantizzazioneCr_ZZ[59];
matriceTemp[6][7]=blocchettoTemp[61]*matriceQuantizzazioneCr_ZZ[61];
%Riga8
matriceTemp[7][0]=blocchettoTemp[35]*matriceQuantizzazioneCr_ZZ[35];
matriceTemp[7][1]=blocchettoTemp[36]*matriceQuantizzazioneCr_ZZ[36];
matriceTemp[7][2]=blocchettoTemp[48]*matriceQuantizzazioneCr_ZZ[48];
matriceTemp[7][3]=blocchettoTemp[49]*matriceQuantizzazioneCr_ZZ[49];
matriceTemp[7][4]=blocchettoTemp[57]*matriceQuantizzazioneCr_ZZ[57];
matriceTemp[7][5]=blocchettoTemp[58]*matriceQuantizzazioneCr_ZZ[58];
matriceTemp[7][6]=blocchettoTemp[62]*matriceQuantizzazioneCr_ZZ[62];
matriceTemp[7][7]=blocchettoTemp[63]*matriceQuantizzazioneCr_ZZ[63];

%printf("\n\nBlocchetto Temporaneo 8*8 Crominanza Cr DeZigZagato e
      Dequantizzato:\n");
%stampaVettoreInt(&matriceTemp[0][0],SIZE*SIZE);


%Applico la IDCT
%printf("\n\nBlocchetto Temporaneo 8*8 Crominanza Cr a cui ho applicato
       la DCT inversa (blocchetto Finale):\n\n\n");
IDCTBlocchetto(&matriceTemp[0][0],&matriceTempIDCT[0][0]);
%stampaVettoreInt(&matriceTempIDCT[0][0],SIZE*SIZE);




%memorizzo i "numeri trovati"(in matriceTempIDCT) che
%rappresentano i colori veri e propri, nei vettori bidimensionali
%vettore_Y[pixelX][pixelY]
%vettore_Cr[pixelX][pixelY] vettore_Cb[pixelX][pixelY]
%tali vettori hanno la dimensione del vettore di luminanza,
%che è la medesima della risoluzione dell'immagine

%Caso di sottocampionamento 2 1 1   2 1 1 -->
%devo replicare i risultati ottenuti a causa del

%sottocampionamento orizzontale e verticale -->
%creo il vettoreCr_Temp
if(Hi[0]*Vi[0]==4){

  %vettoreTemp16*16 contiene i
  for (aa=0;aa<SIZE;aa++){
    for (bb=0;bb<SIZE;bb++){
      vettore_Cr_Temp[aa*Hmax+1][bb*Vmax+1]=vettore_Cr_Temp[aa*Hmax+1][
          bb*Vmax]=vettore_Cr_Temp[aa*Hmax][bb*Vmax+1]=
          vettore_Cr_Temp[aa*Hmax][bb*Vmax]=matriceTempIDCT[aa][bb];

    }
```

```
1920          }

              for (aa=0;aa<Hmax*SIZE;aa++){
                for (bb=0;bb<Vmax*SIZE;bb++){
                  indice_b=((x%(blocchettiNum_X[1]))*Hmax)*SIZE;
                  %(il % non è un commento ma è l'operatore modulo)
                  vettore_Cr[aa][bb+indice_b]=vettore_Cr_Temp[aa][bb];

                }
              }
1930          }%chiude if(Hi[0]*Vi[0]==4)

             %Caso di sottocampionamento 1 1 1
             %1 1 1 ---> In tal caso non è necessario replicare gli elementi

              else if(Hi[0]*Vi[0]==1){
             %caso di sottocampionamento 1 1 1

              if(contrDispari==0){
                for (aa=0;aa<SIZE;aa++){
1940              for (bb=0;bb<SIZE;bb++){
                    indice_b=((x%(blocchettiNum_X[1]))*Hmax)*SIZE;
                    %(il % non è un commento ma è l'operatore modulo)

                    vettore_Cr[aa][bb+indice_b]=matriceTempIDCT[aa][bb];

                  }
                }
              }
              else{ %contrDispari==1
1950            for (aa=0;aa<SIZE;aa++){
                  for (bb=0;bb<SIZE;bb++){
                    indice_a=SIZE;
                    indice_b=((x%(blocchettiNum_X[1]))*Hmax)*SIZE;
                    %(il % non è un commento ma è l'operatore modulo)
                    vettore_Cr[aa+indice_a][bb+indice_b]=matriceTempIDCT[aa][bb];
                  }

                }

1960          }
            }

           }%chiude il for interno dei Hi[1]*Vi[1] blocchetti di crominanza Cr




1970
```

```
%********** 5 - CROMINANZA CB - COEFFICIENTI DC  **********
%********** 5 - CROMINANZA CB - COEFFICIENTI DC  **********
%********** 5 - CROMINANZA CB - COEFFICIENTI DC  **********


1980


for (unsigned short iterCb=1; iterCb<=(Hi[2]*Vi[2]);iterCb++){


  %Nel caso di coefficienti DC lavoro su 5 byte alla volta:
  %ovvero trasformo in binario 5 byte alla volta
  %così che il vettore binario sia costituito
  %da 1 bit per ogni elemento ---> 40 bit
1990


  %variabili da inizializzare
  bytePresi=0;
  contByteEliminati=0;
  contInterno=0;

  %inizializzo la variabili indiciByteEliminatiDC pongo a -1
  %i valori del vettore indiciByteEliminatiDC
2000  for (unsigned short i=0;i<dimCoeffDC;i++){
  %Al massimo posso eliminare 5 byte 00 nel caso
  %00 FF 00 FF 00 FF 00 FF 00 FF (caso peggiore)
    indiciByteEliminatiDC[i]=-1;
  }


  %Pongo a -1 tutti gli elementi del vettore vettoreBitTemp[40]
  for(unsigned short i=0;i<SIZE*dimCoeffDC;i++){
    vettoreBitTempDC[i]=-1;
2010  }


  while(bytePresi<dimCoeffDC && ( (scansione+contInterno)<(dimFile-1)) ){
  %devo prendere almeno 5 byte consecutivi perchè la lungh
  %massima di bit totali presi (codice+size)può essere al
  %massimo di 26 bit (4 byte completi); se prendessi solo
  %4 byte, e devo iniziare a scansionare il primo byte
  %dall'ultimo bit allora avrei 25 bit possibili, un bit in
  %meno del caso peggiore
2020  %il while va avanti fin tanto chè non si prendono 5 byte,
  %oppure si esce dal while se non è possibile prendere i 5
  %byte successivi perchè il file è terminato.In questo caso
  %i byte*8 rimanenti, non presi (dei 40 elementi totali)
  %rimangono inizializzati a -1;

    numerohex=ByteElemento(PuntFile,scansione+contInterno);

    %if ---> se ho uno 00 dopo FF
```

```
      if(numerohex==0x00 && ByteElemento(PuntFile,scansione+contInterno-1)
          ==0xFF){
2030    indiciByteEliminatiDC[contByteEliminati]=contInterno;
        contByteEliminati=contByteEliminati+1;
      }

      %else--> se il byte non è uno 00 dopo un FF, posso mettere gli 8 bit
             del byte in esame nel vettoreBitTempDC[40]
      else{

        for (unsigned short b=0;b<SIZE;b++) {
          vettoreBitTempDC[7-b+SIZE*bytePresi]=numerohex%2;
          numerohex=floor(numerohex/2);
2040    }

        bytePresi=bytePresi+1;
      }

      contInterno=contInterno+1;
    }

    %printf("\n\n5 - CROMINANZA CR - COEFFICIENTE DC\n");
    %for(int y=1;y<=SIZE*dimCoeffDC;y++){
2050 %printf("%d",vettoreBitTempDC[y-1]);
    % if(y%8==0)
    %   printf("\t");
    %}


    %printf("\nbyteeliminati[i]= ");
    %for(int y=0;y<dimCoeffDC;y++){
    % printf("%d\t",indiciByteEliminatiDC[y]);
    %}
2060




    % Vado ad effettuare la decodifica vera e propria, ovvero
    %ricerco il codice dalla tabella di huffman del coeff. DC
    %di luminanza e prendo i bitsuccessivi dati dal valore huffval
    %corrispondete(ovvero size). Solo dopo aver capito quanti
    %bit totali ho usato(bittotali=codice+huffval) è possibile
    %aggiornare il valore della scansione;
2070
    %azzero i valori del blocchetto Temporaneo - è in tale
    %blocchetto  che andrò a registrare volta per volta i 64
    %coefficienti (1 DC e 63 AC) delle 3 componenti
    %Una volta che il blocchetto sarà completo,
    %è possibile copiare i valori nel file opportuno
    for (unsigned short a=0;a<SIZE*SIZE;a++){
      blocchettoTemp[a]=0;
    }


2080
    tab=3;
```

```
%tab=1 PERCHE' è IL CASO DI CROMINANZA E COEFFICIENTE DC
LunghCodice=0;
bitsuccessivi=-1;
%Numero di bit successivi da prendere per decodificare
%il coeff DC di luminanza,ovvero size
while (bitsuccessivi==-1){
  LunghCodice=LunghCodice+1;
  codice= funzioneDecimale(scansionebin,LunghCodice,&vettoreBitTempDC
      [0]);
  %nella variabile codice è presente il codice di lunghezza
  %Lunghcodice preso dal vettoreBitTempDC a partire
  %dall'elemento scansionebin

  if(indiceIni[tab][LunghCodice]!=-1 && indiceFine[tab][LunghCodice
      ]!=-1){
    for (int g=indiceIni[tab][LunghCodice];g<=indiceFine[tab][
        LunghCodice]; g++){
      if(huffCode[tab][g]==codice){
      %se si entra in questo if, il valore di bitsuccessivi
      %viene posto pari ad huffVal, che è sicuramente un
      %valore diverso da -1, per cui si uacirà dal ciclo while
        bitsuccessivi=huffVal[tab][g];
        scansionebinTemp=scansionebin+LunghCodice;
        break;
      }
    }
  }
}


%all'uscita dal ciclo sopra:
%codice rappresenta il codice che è stato preso dal vettore e
%che coincide ad un codice della tabella dei codici di huffman
%di luminanza e DC (Huffcode-HuffCodeBin);

%bitsuccessivi  rappresenta il numero di bit successivi da
%prendere (a partire dalla fine del codice). i bit presi
%rappresentano il valore del coefficiente dc
%(che andrà poi decodificato e dequantizzato)

%scansionebin rappresenta l'indice del vettore vettoreBitTempDC
%che contiene il primo bit successivo al codice preso
%(ovvero il primo degli huffval bit)

%printf("\nscansione di partenza=%d",scansione);
%printf("\t\tscansionebin di partenza=%d",scansionebin);
%printf("\t\tcodice preso=%d",codice);
%printf("\t\tLunghezzacodice=%d",LunghCodice);
%printf("\t\tbitsuccessivi=%d",bitsuccessivi);



%devo aggiungere al valore di DCdiff il valore del DCprec,
%ovvero DC_corrente=DC_diff+DC_prec
% (infatti in fase di codifica si ha:
```

```
%DC_diff = DC_corrente-DC_prec )

% se sono nel primo blocchetto della riga il valore
%DC_prec è nullo per cui il DCcorrente coincide con il DC_diff
%altrimenti: DC_corrente=DC_diff+DC_prec

if(x==0 && iterCb==1){
%inzia con bit=0 ---> numero negativo
  precCb=0;
}
%decodifico il coefficiente DC - ovvero inserisco il valore
%decodificato (ma non ancora dequantizzato) nel primo elemento
%del blocchettoTemp; se il primo bit dei bitsuccessivi presi
%dopo il codice è 0 faccio il complemento a 1 e
%la conversione del valore assoluto

if(vettoreBitTempDC[scansionebinTemp]==0){ %inzia con bit=0 ---> numero
      negativo
  attualeCb=proceduraNumNegativo(scansionebinTemp, bitsuccessivi,&
       vettoreBitTempDC[0]);
}
else{
  attualeCb=funzioneDecimale(scansionebinTemp,bitsuccessivi,&
       vettoreBitTempDC[0]);

}

blocchettoTemp[0]=attualeCb+precCb;

precCb=blocchettoTemp[0];


%printf("\t\tblocchettoTemp[0]=%d",blocchettoTemp[0]);

%aggiorno il valore di scansionebin
%(bit di partenza dal vettoreBitTempDC che verrà preso in seguito)

%aggiorno il valore di scansione per prendere il byte
%seguenete dal file ---> definizione di indicebytesucc

totbit=LunghCodice+bitsuccessivi+scansionebin;
bytesuccessivo=floor(totbit/SIZE);
%variabile che tiene conto del numero di byte utilizzati
%(considerando Lunghcodice, bitsuccessivi e lo scanbin
%della precedente iterazione)
scansionebin= totbit-(bytesuccessivo)*SIZE;
%printf("\ntotbit=%d\t\tbytesuccessivo=%d",totbit,bytesuccessivo);

%Valuto se è stato scartato qualche byte 00 (dopo un byte FF),
%che si trovava prima dell'elemento scansione
%a cui sono arrivato

incrementoscansione=0;
indiceCont=0;
while(indiceCont<=bytesuccessivo){
```

```
    if(incrementoscansione!=indiciByteEliminatiDC[0] &&
          incrementoscansione!=indiciByteEliminatiDC[1] &&
          incrementoscansione!=indiciByteEliminatiDC[2] &&
          incrementoscansione!=indiciByteEliminatiDC[3] &&
          incrementoscansione!=indiciByteEliminatiDC[4]){
      indiceCont=indiceCont+1;
    }
    incrementoscansione=incrementoscansione+1;
  }
  incrementoscansione=incrementoscansione-1;


  %Per cui, il prossimo vettoreBitTemp(vettoreBitTempAC) partirà
  %dall'elemento scansione incrementato di incrementoscansione
  %dal bit numero scansionebin

  scansione=scansione+incrementoscansione;
  %printf("\t\tincrementoscansione=%d\t\tscansione successiva=%d\t\
        tscansionebin successiva=%d",incrementoscansione, scansione,
        scansionebin);


  %********** 6 - CROMINANZA CB - COEFFICIENTI AC  **********
  %********** 6 - CROMINANZA CB - COEFFICIENTI AC  **********
  %********** 6 - CROMINANZA CB - COEFFICIENTI AC  **********

  %Nel caso di coefficienti AC lavoro su 206 byte alla volta:
  %ovvero trasformo in binario 206 byte alla volta
  %così che il vettore binario sia costituito
  %da 1 bit per ogni elemento ---> 1648 bit

  %variabili da inizializzare
  bytePresi=0;
  contByteEliminati=0;
  contInterno=0;
  %inizializzo la variabili indiciByteEliminatiAC pongo a -1
  %i valori del vettore indiciByteEliminatiAC
  for (unsigned short i=0;i<dimCoeffAC;i++){
  %Al massimo posso eliminare 206 byte 00 nel caso
  %00 FF 00 FF 00 FF 00 FF 00 FF 00 ff.... (caso peggiore)
    indiciByteEliminatiAC[i]=-1;
  }

  %Pongo a -1 tutti gli elementi del vettore vettoreBitTemp[1648]
  for(unsigned short i=0;i<SIZE*dimCoeffAC;i++){
    vettoreBitTempAC[i]=-1;
  }

  while(bytePresi<dimCoeffAC && ( (scansione+contInterno)<(dimFile-1)) ){
  %devo prendere almeno 206 byte consecutivi perchè la lungh
  %massima di bit totali presi (codice+size)può essere al
  %massimo di 26 bit (4 byte completi)
  %per ogni codice ---> dato che dovrò trattare 63 elementi:
  %63*26=1638 bit ovvero 204.75 byte, ceil-->205 -->+1 viene 206
```

```
%il while va avanti fin tanto chè non si prendono 206 byte,
%oppure si esce dal while se non è possibile prendere i 206
%byte successivi perchè il file è terminato. In questo caso i
%byte*8 rimanenti, non presi (dei 1648 elementi totali)
%rimangono inizializzati a -1;
  numerohex=ByteElemento(PuntFile,scansione+contInterno);

  %if ---> se ho uno 00 dopo FF
  if(numerohex==0x00 && ByteElemento(PuntFile,scansione+contInterno-1)
       ==0xFF){
    indiciByteEliminatiAC[contByteEliminati]=contInterno;
    contByteEliminati=contByteEliminati+1;
  }
  %else ---> se il byte non è uno 00 dopo un FF, posso mettere gli 8 bit
        del byte in esame nel vettoreBitTempDC[40]
  else{

    for (unsigned short b=0;b<SIZE;b++) {
      vettoreBitTempAC[7-b+SIZE*bytePresi]=numerohex%2;
      numerohex=floor(numerohex/2);

    }
    bytePresi=bytePresi+1;
  }
  contInterno=contInterno+1;
}

%printf("\n\n6 - CROMINANZA CB - COEFFICIENTI AC\n");
%for(int y=1;y<=SIZE*dimCoeffAC;y++){
%printf("%d",vettoreBitTempAC[y-1]);
%if(y%8==0)
% printf("\t");
%}


%printf("\nbyteeliminati[i]= ");
%for(int y=0;y<10;y++){ %for(int y=0;y<dimCoeffAC;y++){
% printf("%d\t",indiciByteEliminatiAC[y]);
%}

%printf("\nscansione di partenza=%d\t\tscansionebin di partenza=%d\n\n
      ",scansione,scansionebin);



tab=4;
ww=1;
LunghCodice=0;

controlloIniziale=0;


do{
  codiceHuffVal=-1;
```

```
if(controlloIniziale==1){
  ww=ww+1;
}
else if(controlloIniziale==2){
  ww=ww+16;
}


%Ricerco il codice nel filebinario che è presente nella
%tabella, ed estraggo il corrispondente valore di
%huffval-->(size/run)
while (codiceHuffVal==-1){
  LunghCodice=LunghCodice+1;
  codice= funzioneDecimale(scansionebin,LunghCodice,&vettoreBitTempAC
      [0]);
  if(indiceIni[tab][LunghCodice]!=-1 && indiceFine[tab][LunghCodice
      ]!=-1){
    for (int g=indiceIni[tab][LunghCodice];g<=indiceFine[tab][
        LunghCodice]; g++){
      if(huffCode[tab][g]==codice){
        codiceHuffVal=huffVal[tab][g];
        scansionebin=scansionebin+LunghCodice; %aggiornamento valore
            scanbin -parte 1/2
        run=separaHexPrimo(codiceHuffVal);
        size=separaHexSecondo(codiceHuffVal);
        break;
      }
    }
  }
}


%Controllo errore:
if(size>10 || size<0 || run>15 || run<0)
  printf("\n\n********** ERRORE NELLA LETTURA/DECODIFICA DI RUN
      SIZE - RUN E/O SIZE HA ASSUNTO UN VALORE NON CONSENTITO
      **********\n\n");



%printf("codice preso=%d\t\tLunghezza codice=%d",codice,LunghCodice);
%printf("\t\t\t(run/size)=(%d/%d)",run,size);
codiceHuffVal=0;
LunghCodice=0;

%decodifico il coefficiente AC[ww]--->bloccoTem[ww] in esame;

%caso 1
if(size!=0){
  controlloIniziale=1;
  ww=ww+run;

  if(vettoreBitTempAC[scansionebin]==0){
  %complementoauno e conversione del valore assoluto
```

```
            blocchettoTemp[ww]=proceduraNumNegativo(scansionebin, size,&
                vettoreBitTempAC[0]);
          }
          else {
            blocchettoTemp[ww]=funzioneDecimale(scansionebin,size,&
                vettoreBitTempAC[0]);
          }
          %printf("\t\tblocchettoTemp[%d]=%d\n",ww,blocchettoTemp[ww]);
          scansionebin=scansionebin+size;
          %aggiornamento valore scanbin -parte 2/2

        }


        %caso 2 - size=0 e run=15 --> corsa di 16 zeri- ZRL
        else if(size==0 && run==15){
          controlloIniziale=2;

        }

        %caso 3 - size=0 e run=0 ---> fine blocchetto - EOB
        else if (size==0 && run==0){
          break; %esco dal do...while se incontro EOB
        }

      }while(ww<(SIZE*SIZE-1));

      %printf("\nblocchettoTemp=\n");
      %stampaVettoreInt(&blocchettoTemp[0],SIZE*SIZE);


      %ho usato un totale di bit pari a scansionebin:

      totbit=scansionebin;

      bytesuccessivo=floor(totbit/SIZE);
      %byte usati nel vettore vettoreBitTemp (che non include i byte
          eliminati)
      scansionebin= totbit-(bytesuccessivo)*SIZE;
      %bit di inizio scansione del vettoreBitTemp successivo

      %printf("\n\ntotbit=%d\t\tbytesuccessivo=%d\t\t",totbit,bytesuccessivo)
          ;

      %valuto il byte effettivo successivo (quello del file),
      %ovvero valuto se è stato scartato qualche byte 00
      %(dopo un byte FF),che si trovava prima dell'elemento
      %scansione a cui sono arrivato
      incrementoscansione=0;
      indiceCont=0;
      pres=0;
      while(indiceCont<=bytesuccessivo){
        for (unsigned int i=0 ; i<dimCoeffAC; i++){
          if (incrementoscansione==indiciByteEliminatiAC[i]){
            pres=1;
```

```
            break;
          }
        }
        if(pres==0){
          indiceCont=indiceCont+1;
        }

        incrementoscansione=incrementoscansione+1;
        pres=0;
      }
      incrementoscansione=incrementoscansione-1;

      %Per cui, il prossimo vettoreBitTemp (vettroeBitTempDC)
      %partirà dall'elemento scansione incrementato di
      %incrementoscansione e partirà dal bit numero scansionebin

      scansione=scansione+incrementoscansione;
      %printf("\t\tincrementoscansione=%d\t\tscansione successiva=%d\t\
            tscansionebin successiva=%d",incrementoscansione, scansione,
            scansionebin);



      %Applico il processo di quantizzazione inverso utilizzando la
      %matrice matriceQuantizzazioneY_ZZ che il vettore di 64
      %elementi disposti a zig-zag, analogamente a blocchettoTemp.
      %Memorizzo i valori in matriceTemp che è una vettore
      %bidimensionale che rappresenta gli elementi
      %dequantizzati e ordinati

      %COEFFICIENTE DC
      matriceTemp[0][0]=blocchettoTemp[0]*matriceQuantizzazioneCr_ZZ[0];


      %COEFFICIENTI AC
      %Riga1
      matriceTemp[0][1]=blocchettoTemp[1]*matriceQuantizzazioneCr_ZZ[1];
      matriceTemp[0][2]=blocchettoTemp[5]*matriceQuantizzazioneCr_ZZ[5];
      matriceTemp[0][3]=blocchettoTemp[6]*matriceQuantizzazioneCr_ZZ[6];
      matriceTemp[0][4]=blocchettoTemp[14]*matriceQuantizzazioneCr_ZZ[14];
      matriceTemp[0][5]=blocchettoTemp[15]*matriceQuantizzazioneCr_ZZ[15];
      matriceTemp[0][6]=blocchettoTemp[27]*matriceQuantizzazioneCr_ZZ[27];
      matriceTemp[0][7]=blocchettoTemp[28]*matriceQuantizzazioneCr_ZZ[28];
      %Riga2
      matriceTemp[1][0]=blocchettoTemp[2]*matriceQuantizzazioneCr_ZZ[2];
      matriceTemp[1][1]=blocchettoTemp[4]*matriceQuantizzazioneCr_ZZ[4];
      matriceTemp[1][2]=blocchettoTemp[7]*matriceQuantizzazioneCr_ZZ[7];
      matriceTemp[1][3]=blocchettoTemp[13]*matriceQuantizzazioneCr_ZZ[13];
      matriceTemp[1][4]=blocchettoTemp[16]*matriceQuantizzazioneCr_ZZ[16];
      matriceTemp[1][5]=blocchettoTemp[26]*matriceQuantizzazioneCr_ZZ[26];
      matriceTemp[1][6]=blocchettoTemp[29]*matriceQuantizzazioneCr_ZZ[29];
      matriceTemp[1][7]=blocchettoTemp[42]*matriceQuantizzazioneCr_ZZ[42];
      %Riga3
      matriceTemp[2][0]=blocchettoTemp[3]*matriceQuantizzazioneCr_ZZ[3];
      matriceTemp[2][1]=blocchettoTemp[8]*matriceQuantizzazioneCr_ZZ[8];
```

```
        matriceTemp[2][2]=blocchettoTemp[12]*matriceQuantizzazioneCr_ZZ[12];
2440    matriceTemp[2][3]=blocchettoTemp[17]*matriceQuantizzazioneCr_ZZ[17];
        matriceTemp[2][4]=blocchettoTemp[25]*matriceQuantizzazioneCr_ZZ[25];
        matriceTemp[2][5]=blocchettoTemp[30]*matriceQuantizzazioneCr_ZZ[30];
        matriceTemp[2][6]=blocchettoTemp[41]*matriceQuantizzazioneCr_ZZ[41];
        matriceTemp[2][7]=blocchettoTemp[43]*matriceQuantizzazioneCr_ZZ[43];
        %Riga4
        matriceTemp[3][0]=blocchettoTemp[9]*matriceQuantizzazioneCr_ZZ[9];
        matriceTemp[3][1]=blocchettoTemp[11]*matriceQuantizzazioneCr_ZZ[11];
        matriceTemp[3][2]=blocchettoTemp[18]*matriceQuantizzazioneCr_ZZ[18];
        matriceTemp[3][3]=blocchettoTemp[24]*matriceQuantizzazioneCr_ZZ[24];
2450    matriceTemp[3][4]=blocchettoTemp[31]*matriceQuantizzazioneCr_ZZ[31];
        matriceTemp[3][5]=blocchettoTemp[40]*matriceQuantizzazioneCr_ZZ[40];
        matriceTemp[3][6]=blocchettoTemp[44]*matriceQuantizzazioneCr_ZZ[44];
        matriceTemp[3][7]=blocchettoTemp[53]*matriceQuantizzazioneCr_ZZ[53];
        %Riga5
        matriceTemp[4][0]=blocchettoTemp[10]*matriceQuantizzazioneCr_ZZ[10];
        matriceTemp[4][1]=blocchettoTemp[19]*matriceQuantizzazioneCr_ZZ[19];
        matriceTemp[4][2]=blocchettoTemp[23]*matriceQuantizzazioneCr_ZZ[23];
        matriceTemp[4][3]=blocchettoTemp[32]*matriceQuantizzazioneCr_ZZ[32];
        matriceTemp[4][4]=blocchettoTemp[39]*matriceQuantizzazioneCr_ZZ[39];
2460    matriceTemp[4][5]=blocchettoTemp[45]*matriceQuantizzazioneCr_ZZ[45];
        matriceTemp[4][6]=blocchettoTemp[52]*matriceQuantizzazioneCr_ZZ[52];
        matriceTemp[4][7]=blocchettoTemp[54]*matriceQuantizzazioneCr_ZZ[54];
        %Riga
        matriceTemp[5][0]=blocchettoTemp[20]*matriceQuantizzazioneCr_ZZ[20];
        matriceTemp[5][1]=blocchettoTemp[22]*matriceQuantizzazioneCr_ZZ[22];
        matriceTemp[5][2]=blocchettoTemp[33]*matriceQuantizzazioneCr_ZZ[33];
        matriceTemp[5][3]=blocchettoTemp[38]*matriceQuantizzazioneCr_ZZ[38];
        matriceTemp[5][4]=blocchettoTemp[46]*matriceQuantizzazioneCr_ZZ[46];
        matriceTemp[5][5]=blocchettoTemp[51]*matriceQuantizzazioneCr_ZZ[51];
2470    matriceTemp[5][6]=blocchettoTemp[55]*matriceQuantizzazioneCr_ZZ[55];
        matriceTemp[5][7]=blocchettoTemp[60]*matriceQuantizzazioneCr_ZZ[60];
        %Riga7
        matriceTemp[6][0]=blocchettoTemp[21]*matriceQuantizzazioneCr_ZZ[21];
        matriceTemp[6][1]=blocchettoTemp[34]*matriceQuantizzazioneCr_ZZ[34];
        matriceTemp[6][2]=blocchettoTemp[37]*matriceQuantizzazioneCr_ZZ[37];
        matriceTemp[6][3]=blocchettoTemp[47]*matriceQuantizzazioneCr_ZZ[47];
        matriceTemp[6][4]=blocchettoTemp[50]*matriceQuantizzazioneCr_ZZ[50];
        matriceTemp[6][5]=blocchettoTemp[56]*matriceQuantizzazioneCr_ZZ[56];
        matriceTemp[6][6]=blocchettoTemp[59]*matriceQuantizzazioneCr_ZZ[59];
2480    matriceTemp[6][7]=blocchettoTemp[61]*matriceQuantizzazioneCr_ZZ[61];
        %Riga8
        matriceTemp[7][0]=blocchettoTemp[35]*matriceQuantizzazioneCr_ZZ[35];
        matriceTemp[7][1]=blocchettoTemp[36]*matriceQuantizzazioneCr_ZZ[36];
        matriceTemp[7][2]=blocchettoTemp[48]*matriceQuantizzazioneCr_ZZ[48];
        matriceTemp[7][3]=blocchettoTemp[49]*matriceQuantizzazioneCr_ZZ[49];
        matriceTemp[7][4]=blocchettoTemp[57]*matriceQuantizzazioneCr_ZZ[57];
        matriceTemp[7][5]=blocchettoTemp[58]*matriceQuantizzazioneCr_ZZ[58];
        matriceTemp[7][6]=blocchettoTemp[62]*matriceQuantizzazioneCr_ZZ[62];
        matriceTemp[7][7]=blocchettoTemp[63]*matriceQuantizzazioneCr_ZZ[63];
2490
        %printf("\n\nBlocchetto Temporaneo 8*8 Crominanza Cb DeZigZagato e
              Dequantizzato:\n");
        %stampaVettoreInt(&matriceTemp[0][0],SIZE*SIZE);
```

```
%Applico la IDCT
%printf("\n\nBlocchetto Temporaneo 8*8 Crominanza Cb a cui ho applicato
       la DCT inversa(Blocchetto Finale):\n\n\n");
IDCTBlocchetto(&matriceTemp[0][0],&matriceTempIDCT[0][0]);
%stampaVettoreInt(&matriceTempIDCT[0][0],SIZE*SIZE);


%memorizzo i "numeri trovati"(in matriceTempIDCT) che
%rappresetano i colori veri e propri, nei vettori bidimensionali
% vettore_Y[pixelX][pixelY]
%vettore_Cr[pixelX][pixelY]
%vettore_Cb[pixelX][pixelY]
%tali vettori hanno la dimensione del vettore di luminanza,
%che è la medesima della risoluzione dell'immagine


%Caso di sottocampionamento 2 1 1   2 1 1
if(Hi[0]*Vi[0]==4){

  %vettoreTemp16*16 contiene i
  for (aa=0;aa<SIZE;aa++){
    for (bb=0;bb<SIZE;bb++){
      vettore_Cb_Temp[aa*Hmax+1][bb*Vmax+1]=vettore_Cb_Temp[aa*Hmax+1][
          bb*Vmax]=vettore_Cb_Temp[aa*Hmax][bb*Vmax+1]=
          vettore_Cb_Temp[aa*Hmax][bb*Vmax]=matriceTempIDCT[aa][bb];
    }
  }

  for (aa=0;aa<Hmax*SIZE;aa++){
    for (bb=0;bb<Vmax*SIZE;bb++){
      indice_b=((x%(blocchettiNum_X[1]))*Hmax)*SIZE;
      vettore_Cb[aa][bb+indice_b]=vettore_Cb_Temp[aa][bb];

    }
  }
}


%Caso di sottocampionamento 1 1 1     1 1 1
else if(Hi[0]*Vi[0]==1){ %caso di sottocampionamento 1 1 1


  if(contrDispari==0){
    for (aa=0;aa<SIZE;aa++){
      for (bb=0;bb<SIZE;bb++){
        indice_b=((x%(blocchettiNum_X[1]))*Hmax)*SIZE;
        vettore_Cb[aa][bb+indice_b]=matriceTempIDCT[aa][bb];

      }
    }
```

```
          }
          else{ %contrDispari==1
            for (aa=0;aa<SIZE;aa++){
              for (bb=0;bb<SIZE;bb++){
                indice_a=SIZE;
2550            indice_b=((x%(blocchettiNum_X[1]))*Hmax)*SIZE;
                vettore_Cb[aa+indice_a][bb+indice_b]=matriceTempIDCT[aa][bb];
              }
            }
          }
        }


      }
      %chiude il for interno dei Hi[2]*Vi[2]
2560  %blocchetti di crominanza Cb



      %arrivati a questo punto dello script, è stato creato un bloccone
      %8x8 (in caso di sottocampionamento 1 1 1 ),ovvero ho
      %decodificato gli 8x8 coefficienti di Y, gli 8x8 coeff. di Cr,
      %gli 8x8 coeff di Cb. Nel caso di sottocampionamento 2 1 1 è
      %stato creato un bloccone 16x16, ovvero ho decodificato i 16x16
      %coefficienti di Y, i 16x16 coeff. di Cr (creati a partire dagli
2570  %8x8 coeff.), i 16x16 coeff di Cb
      %(creati a partire dagli 8x8 coeff.)



    %ANALISI DEI PUNTI DI BORDO
    %ANALISI DEI PUNTI DI BORDO


2580 if((x+1)%(indice_moltiplicazione*macroBlocchiNum_X)==0){
    %si entra in questo ciclo quando vengono riempiti i tre vettori di
    %16 righe..., così ad esempio se si entra in questo ciclo, vuol dire
    %che i vettori: vettore_Y[][]; vettore_Cr[][];vettore_Cb[][] sono
    %stati riempiti
      blocchi_rettangolari=blocchi_rettangolari+1;
      indice_riga=blocchi_rettangolari*2*SIZE;

      %converto in RGB (YcrCb --> RGB)
        for (unsigned short l=0;l<2*SIZE;l++){
2590      for (unsigned short m=0;m<pixelX;m++){
            R[l][m]=round((vettore_Y[l][m]+128)+(double)1.402*vettore_Cb[l][m]);
            G[l][m]=round((vettore_Y[l][m]+128)-(double)0.34414*vettore_Cr[l][m
                ]-(double)0.71414*vettore_Cb[l][m]);
            B[l][m]=round((vettore_Y[l][m]+128)+(double)1.772*vettore_Cr[l][m]);
          }
        }

        if(localizzazione==0){
```

```
%printf("\nPrimo If --> blocco=%d\n\n",blocchi_rettangolari);
%printf("vettore pezzetto:\n");
%for (unsigned short r=0;r<2*SIZE;r++){
  %printf("\n\n");
  %for (unsigned short c=0; c<pixelX;c++){
  %printf("[%d][%d]=(%d,%d,%d)\t",r,c,vettore_Y[r][c],vettore_Cr[r][c],
        vettore_Cb[r][c]);
  %}
%}


%PARTE A - VERIFICA DELLA PRESENZA DI PUNTI DI BORDO NELL'ULTIMO
%PEZZETTO IN ESAME CREATO - APPLICO L'ALGORITMO A CROCE E
%L'ALGORITMO DI RIDUZIONE DEI PUNTI nel rettangolo 16xPixelX

%L'algoritmo a croce e l'algoritmo di riduzione dei punti si
%applicano nel processo di ricerca dei punti di angolo, ovvero
%per il modello di rivelazione della posizione; per cui si
%effettua fin tanto chè non è stato rivelato il rettangolo
%di posizionamento, ovvero fino a chè la variabile
%localizzazione non vale 1;

for (unsigned int l=marg; l<2*SIZE-marg;l++){
  for (unsigned int m=marg;m<(pixelX-marg);m++){
    if ( (R[l][m]<soglia_R) && (G[l][m]<soglia_G) && (B[l][m]<soglia_B))
        {
    %l'analisi si effettua solo se il pixel in esame è nero...
    %se queste condizioni sono verificate il pixel in
    %esame è "quasi nero"
      vettore_contrasto[0]=vettore_contrasto[1]=vettore_contrasto[2]=
          vettore_contrasto[3]=0;

      %printf("\n\n\n1 Sono entrato nell'analisi del pixel nero alla
          cordinata (%d,%d) --> (%d,%d,%d)",indice_riga+l,m,R[l][m],G[l
          ][m],B[l][m]);


      %analisi verso l'alto
      %printf("\n -Analisi verso l'alto [%d][%d]:",indice_riga+l,m);
      for (unsigned short i=1; i<=marg;i++){
        %printf("\t%d ->%d\t",i,vettore_Y[l][m]-vettore_Y[l-i][m]);
        if((abs(R[l][m]-R[l-i][m])>differenza) && (abs(G[l][m]-G[l-i][m
            ])>differenza) && (abs(B[l][m]-B[l-i][m])>differenza))
          vettore_contrasto[0]=1;
          %se c'è UN SOLO punto di contrasto
          %nelle direzioni di analisi, vale 1

      }
      %printf("\t\tvettore_contrasto[0]=%d;",vettore_contrasto[0]);

      %analisi "verso il basso"
      %printf("\n -Analisi verso il basso[%d][%d]:",indice_riga+l,m);
      for (unsigned short i=1; i<=marg;i++){
        %printf("\t%d ->%d\t",i,vettore_Y[l][m]-vettore_Y[l+i][m]);
```

```
    if((abs(R[l][m]-R[l+i][m])>differenza) && (abs(G[l][m]-G[l+i][m
        ])>differenza) && (abs(B[l][m]-B[l+i][m])>differenza))
      vettore_contrasto[1]=1;
  }
  %printf("\t\tvettore_contrasto[1]=%d;",vettore_contrasto[1]);


  %analisi "verso destra"
  %printf("\n -Analisi verso destra[%d][%d]:",indice_riga+l,m);
  for (unsigned short i=1; i<=marg;i++){
    %printf("\t%d ->%d\t",i,vettore_Y[l][m]-vettore_Y[l][m+1]);
    if((abs(R[l][m]-R[l][m+i])>differenza) && (abs(G[l][m]-G[l][m+i
        ])>differenza) && (abs(B[l][m]-B[l][m+i])>differenza))
      vettore_contrasto[2]=1;
  }
  %printf("\t\tvettore_contrasto[2]=%d;",vettore_contrasto[2]);


  %analisi "verso sinistra"
  %printf("\n -Analisi verso sinistra[%d][%d]:",indice_riga+l,m);
  for (unsigned short i=1; i<=marg;i++){
    %printf("\t%d ->%d\t",i,vettore_Y[l][m]-vettore_Y[l][m-i]);
    if((abs(R[l][m]-R[l][m-i])>differenza) && (abs(G[l][m]-G[l][m-i
        ])>differenza) && (abs(B[l][m]-B[l][m-i])>differenza))
      vettore_contrasto[3]=1;
  }
  %printf("\t\tvettore_contrasto[3]=%d;",vettore_contrasto[3]);



  %4 confronti del vettore_contrasto per determinare l'eventuale tipo
       di angolo in esame


  %ALGORITMO DI RIDUZIONE DEI PUNTI
  %ALGORITMO DI RIDUZIONE DEI PUNTI
  %ALGORITMO DI RIDUZIONE DEI PUNTI
  %1    %1    %1    %1    %1    %1    %1    %1
  if ((vettore_contrasto[0]==1) && (vettore_contrasto[1]==0) && (
      vettore_contrasto[2]==0) && (vettore_contrasto[3]==1)){
    %printf("A_S->(%d,%d)\n",indice_riga+l,m);
    if(ind_conf_A_S==0){
    %se l'elemento trovato è il primo allora viene
    %memorizzato direttamente - sto supponendo quindi
    %che il primo elemento trovato sia sempre buono...
    %ovvero suppongo che non ci siano "macchie" che
    %determinino un falso allarme
      confine_A_S[0][0]=indice_riga+l;
      confine_A_S[1][0]=m;
      ind_conf_A_S=1;
    }
    else{
      memo=1;
      for (unsigned short ind=0; ind<ind_conf_A_S;ind++){
      %confronto l'indice attuale (indice_riga+l,m) con gli
```

```
        %indici memorizzati in precedenza nel vettore
        %confine_A_S[0][ind]

2700            if( (abs((indice_riga+l-confine_A_S[0][ind]))<marg) && (abs((
                      m-confine_A_S[1][ind]))<marg)){
              %se il confronto è positivo, non memorizzo la
              %variabile e conteggio il numero di valori eliminati
                memo=0;
                eliminati_A_S[ind]=eliminati_A_S[ind]+1;
                if((indice_riga+l)<=confine_A_S[0][ind] && m<=confine_A_S
                      [1][ind]){ %sovrascivo il nuovo valore
                  confine_A_S[0][ind]=indice_riga+l;
                  confine_A_S[1][ind]=m;
                }
              }
2710          }
            if(memo==1){%memorizzo i valori attuali nel vettore
              confine_A_S[0][ind_conf_A_S]=indice_riga+l;
              confine_A_S[1][ind_conf_A_S]=m;
              ind_conf_A_S=ind_conf_A_S+1;
            }
          }

        }

2720

        %ALGORITMO DI RIDUZIONE DEI PUNTI
        %2    %2    %2    %2    %2    %2    %2    %2
        else if ((vettore_contrasto[0]==1) && (vettore_contrasto[1]==0) &&
              (vettore_contrasto[2]==1) && (vettore_contrasto[3]==0)){
          %printf("A_D->(%d,%d)\n",indice_riga+l,m);
          if(ind_conf_A_D==0){
          %se l'elemento trovato è il primo allora viene
          %memorizzato direttamente - sto supponendo quindi che
          %il primo elemento trovato sia sempre buono...ovvero
          %suppongo che non ci siano "macchie" che detrminino un
2730      %falso allarme
            confine_A_D[0][0]=indice_riga+l;
            confine_A_D[1][0]=m;
            ind_conf_A_D=1;
          }
          else{
            memo=1;
            for (unsigned short ind=0; ind<ind_conf_A_D;ind++){
              if( (abs((indice_riga+l-confine_A_D[0][ind]))<marg) && (abs((
                      m-confine_A_D[1][ind]))<marg)){
                memo=0;
2740            eliminati_A_D[ind]=eliminati_A_D[ind]+1;
                if((indice_riga+l)<=confine_A_D[0][ind] && m>=confine_A_D
                      [1][ind]){
                  confine_A_D[0][ind]=indice_riga+l;
                  confine_A_D[1][ind]=m;
                }
              }
            }
```

```
        if(memo==1){%memorizzo i valori attuali nel vettore
          confine_A_D[0][ind_conf_A_D]=indice_riga+l;
          confine_A_D[1][ind_conf_A_D]=m;
          ind_conf_A_D=ind_conf_A_D+1;
        }
      }
    }


    %ALGORITMO DI RIDUZIONE DEI PUNTI
    %3    %3    %3    %3    %3    %3    %3    %3
    else if ((vettore_contrasto[0]==0) && (vettore_contrasto[1]==1) &&
          (vettore_contrasto[2]==0) && (vettore_contrasto[3]==1)){
      %printf("B_S->(%d,%d)\n",indice_riga+l,m);
      if(ind_conf_B_S==0){
      %se l'elemento trovato è il primo allora viene
      %memorizzato direttamente - sto supponendo
      %quindi che il primo elemento trovato sia sempre
      %buono...ovvero suppongo che non ci siano "macchie"
      %che detrminino un falso allarme
        confine_B_S[0][0]=indice_riga+l;
        confine_B_S[1][0]=m;
        ind_conf_B_S=1;
      }
      else{
        memo=1;
        for (unsigned short ind=0; ind<ind_conf_B_S;ind++){
          if( (abs((indice_riga+l-confine_B_S[0][ind]))<marg) && (abs((
                m-confine_B_S[1][ind]))<marg)){
            memo=0;
            eliminati_B_S[ind]=eliminati_B_S[ind]+1;
            if((indice_riga+l)>=confine_B_S[0][ind] && m<=confine_B_S
                  [1][ind]){
              confine_B_S[0][ind]=indice_riga+l;
              confine_B_S[1][ind]=m;
            }
          }
        }
        if(memo==1){
        %memorizzo i valori attuali nel vettore
          confine_B_S[0][ind_conf_B_S]=indice_riga+l;
          confine_B_S[1][ind_conf_B_S]=m;
          ind_conf_B_S=ind_conf_B_S+1;
        }
      }
    }

    %ALGORITMO DI RIDUZIONE DEI PUNTI
    %4    %4    %4    %4    %4    %4    %4    %4

    else if ((vettore_contrasto[0]==0) && (vettore_contrasto[1]==1) &&
          (vettore_contrasto[2]==1) && (vettore_contrasto[3]==0)){
      %printf("B_D->(%d,%d)\n",indice_riga+l,m);
      if(ind_conf_B_D==0){
      %se l'elemento trovato è il primo allora viene
```

```
            %memorizzato direttamente - sto supponendo quindi che
            %il primo elemento trovato sia sempre buono...ovvero
            %suppongo che non ci siano "macchie" che detrminino
            %un falso allarme
              confine_B_D[0][0]=indice_riga+l;
              confine_B_D[1][0]=m;
              ind_conf_B_D=1;
            }
            else{
              memo=1;
              for (unsigned short ind=0; ind<ind_conf_B_D;ind++){
                if( (abs((indice_riga+l-confine_B_D[0][ind]))<marg) && (abs((
                      m-confine_B_D[1][ind]))<marg)){
                  memo=0;
                  eliminati_B_D[ind]=eliminati_B_D[ind]+1;
                  if((indice_riga+l)>=confine_B_D[0][ind] && m>=confine_B_D
                        [1][ind]){
                    confine_B_D[0][ind]=indice_riga+l;
                    confine_B_D[1][ind]=m;
                  }
                }
              }
              if(memo==1){%memorizzo i valori attuali nel vettore
                confine_B_D[0][ind_conf_B_D]=indice_riga+l;
                confine_B_D[1][ind_conf_B_D]=m;
                ind_conf_B_D=ind_conf_B_D+1;
              }
            }
          }


        }%chiude l'if che analizza i soli pixel neri
      } %chiude for(unsigned int m=marg;m<=(pixelX-marg);m++)
      %(for sulle colonne del pezzetto in analisi)

    }% chiude for (unsigned int l=marg; l<=2*SIZE-marg;l++)
    %(for sulle righe del pezzetto in analisi)









    %PARTE B - DAPPRIMA COPIO LE PRIME 2*marg RIGHE DEL PEZZETTO IN
    %ANALISI NELLE ULTIME DUE DEL VETTORE_PLUS
    %E IN SEGUITO ANALIZZO (PEZZETTO VETTORE_TEMP COMPLETO) - QUESTA
    %PARTE DELL'ALGORITMO NON VA EFFETTUATA  NEL CASO DI PRIMO PEZZETTO
    %(ovvero non va effettuata quando blocchi_rettangolari==0)
    %inoltre non mi interessa costruire il vettore_Y_plus o R_Plus...
    %nemmeno quando ho trovato il rettangolo del modello di
    %posizionamento, ovvero quando localizzazione=1!
    %(ovvero l'analisi va effettuata solo se localizzazione vale 0)
```

```
if(blocchi_rettangolari!=0){
for (unsigned int l=0; l<2*marg;l++){
  for (unsigned int m=0;m<pixelX;m++){
    vettore_Y_plus[2*marg+l][m]=vettore_Y[l][m];
    vettore_Cr_plus[2*marg+l][m]=vettore_Cr[l][m];
    vettore_Cb_plus[2*marg+l][m]=vettore_Cb[l][m];

  }
}

for (unsigned short l=0;l<4*marg;l++){
  for (unsigned short m=0;m<pixelX;m++){
    R_plus[l][m]=round((vettore_Y_plus[l][m]+128)+(double)1.402*
        vettore_Cb_plus[l][m]);
    G_plus[l][m]=round((vettore_Y_plus[l][m]+128)-(double)0.34414*
        vettore_Cr_plus[l][m]-(double)0.71414*vettore_Cb_plus[l][m]);
    B_plus[l][m]=round((vettore_Y_plus[l][m]+128)+(double)1.772*
        vettore_Cr_plus[l][m]);
  }
}


%printf("\n\n\nvettore pezzetto_plus:\n");
%for (unsigned short r=0;r<4*marg;r++){
  %printf("\n\n");
  %for (unsigned short c=0; c<pixelX;c++){
    %printf("[%d][%d]=(%d,%d,%d)\t",r,c,R_plus[r][c],G_plus[r][c],B_plus[
        r][c]);
  %}
%}

%analizzo il vettore_plus
%ALGORITMO A CROCE  E ALGORITMO DI RIDUZIONE DEI PUNTI

for (unsigned int l=marg; l<3*marg;l++){
  for (unsigned int m=marg;m<(pixelX-marg);m++){


    if ( (R_plus[l][m]<soglia_R) && (G_plus[l][m]<soglia_G) && (B_plus[l
        ][m]<soglia_B)){
      %l'analisi si effettua solo sel pixel in esame è nero...
      %se queste condizioni sono verificate il pixel in esame è
      %"quasi nero"

      %printf("\n\n\n2 Sono entrato nell'analisi del pixel nero alla
          cordinata (%d,%d) --> (%d,%d,%d)",indice_riga+l,m,R[l][m],G[l][
          m],B[l][m]);

        vettore_contrasto[0]=vettore_contrasto[1]=vettore_contrasto[2]=
            vettore_contrasto[3]=0;
        indice_riga=blocchi_rettangolari*2*SIZE-2*marg;
        %printf("\n\n\nSono entrato nell'analisi del pixel nero alla
            cordinata (%d,%d) --> (%d,%d,%d)",indice_riga+l,m,
```

```
        vettore_Y_plus[l][m],vettore_Cr_plus[l][m],vettore_Cb_plus[l
        ][m]);


      %analisi verso l'alto
2900  %printf("\n -Analisi verso l'alto [%d][%d]:",indice_riga+l,m);
      for (unsigned short i=1; i<=marg;i++){
        %printf("\t%d ->%d\t",i,vettore_Y_plus[l][m]-vettore_Y_plus[l-i][
            m]);
        if((abs(R_plus[l][m]-R_plus[l-i][m])>differenza) && (abs(G_plus[
            l][m]-G_plus[l-i][m])>differenza) && (abs(B_plus[l][m]-
            B_plus[l-i][m])>differenza))
          vettore_contrasto[0]=1;
      }
      %printf("\t\tvettore_contrasto[0]=%d;",vettore_contrasto[0]);

      %analisi "verso il basso"
      %printf("\n -Analisi verso il basso[%d][%d]:",indice_riga+l,m);
2910  for (unsigned short i=1; i<=marg;i++){
        %printf("\t%d ->%d\t",i,vettore_Y_plus[l][m]-vettore_Y_plus[l+i][
            m]);
        if((abs(R_plus[l][m]-R_plus[l+i][m])>differenza) && (abs(G_plus[
            l][m]-G_plus[l+i][m])>differenza) && (abs(B_plus[l][m]-
            B_plus[l+i][m])>differenza))
          vettore_contrasto[1]=1;
      }
      %printf("\t\tvettore_contrasto[1]=%d;",vettore_contrasto[1]);


      %analisi "verso destra"
      %printf("\n -Analisi verso destra[%d][%d]:",indice_riga+l,m);
2920  for (unsigned short i=1; i<=marg;i++){
        %printf("\t%d ->%d\t",i,vettore_Y_plus[l][m]-vettore_Y_plus[l][m
            +1]);
        if((abs(R_plus[l][m]-R_plus[l][m+i])>differenza) && (abs(G_plus[
            l][m]-G_plus[l][m+i])>differenza) && (abs(B_plus[l][m]-
            B_plus[l][m+i])>differenza))
          vettore_contrasto[2]=1;
      }
      %printf("\t\tvettore_contrasto[2]=%d;",vettore_contrasto[2]);


      %analisi "verso sinistra"
      %printf("\n -Analisi verso sinistra[%d][%d]:",indice_riga+l,m);
2930  for (unsigned short i=1; i<=marg;i++){
        %printf("\t%d ->%d\t",i,vettore_Y_plus[l][m]-vettore_Y_plus[l][m-
            i]);
        if((abs(R_plus[l][m]-R_plus[l][m-i])>differenza) && (abs(G_plus[
            l][m]-G_plus[l][m-i])>differenza) && (abs(B_plus[l][m]-
            B_plus[l][m-i])>differenza))

          vettore_contrasto[3]=1;
      }
      %printf("\t\tvettore_contrasto[3]=%d;",vettore_contrasto[3]);
```

```
2940

            %4 confronti del vettore_contrasto per determinare
            %l'eventuale tipo di angolo in esame
            %ALGORITMO DI RIDUZIONE DEI PUNTI
            %1    %1    %1    %1    %1    %1    %1    %1
            if ((vettore_contrasto[0]==1) && (vettore_contrasto[1]==0) && (
                  vettore_contrasto[2]==0) && (vettore_contrasto[3]==1)){
              %printf("A_S->(%d,%d)\n",indice_riga+l,m);
               if(ind_conf_A_S==0){
              %se l'elemento trovato è il primo allora viene
2950          %memorizzato direttamente - sto supponendo quindi
              %che il primo elemento trovato sia sempre buono...
              %ovvero suppongo che non ci siano "macchie" che
              %determinino un falso allarme
                 confine_A_S[0][0]=indice_riga+l;
                 confine_A_S[1][0]=m;
                 ind_conf_A_S=1;
               }
               else{
                 memo=1;
2960             for (unsigned short ind=0; ind<ind_conf_A_S;ind++){
                   if( (abs((indice_riga+l-confine_A_S[0][ind]))<marg) && (abs((
                         m-confine_A_S[1][ind]))<marg)){
                     memo=0;
                     eliminati_A_S[ind]=eliminati_A_S[ind]+1;
                     if((indice_riga+l)<=confine_A_S[0][ind] && m<=confine_A_S
                           [1][ind]){
                       confine_A_S[0][ind]=indice_riga+l;
                       confine_A_S[1][ind]=m;
                     }
                   }
                 }
2970             if(memo==1){%memorizzo i valori attuali nel vettore
                   confine_A_S[0][ind_conf_A_S]=indice_riga+l;
                   confine_A_S[1][ind_conf_A_S]=m;
                   ind_conf_A_S=ind_conf_A_S+1;
                 }
               }

            }


2980

            %ALGORITMO DI RIDUZIONE DEI PUNTI
            %2    %2    %2    %2    %2    %2    %2    %2
            else if ((vettore_contrasto[0]==1) && (vettore_contrasto[1]==0) &&
                  (vettore_contrasto[2]==1) && (vettore_contrasto[3]==0)){
              %printf("A_D->(%d,%d)\n",indice_riga+l,m);
               if(ind_conf_A_D==0){
              %se l'elemento trovato è il primo allora viene
              %memorizzato direttamente - sto supponendo quindi
```

```
      %che il primo elemento trovato sia sempre buono...
2990  %ovvero suppongo che non ci siano "macchie" che
      %determinino un falso allarme
        confine_A_D[0][0]=indice_riga+l;
        confine_A_D[1][0]=m;
        ind_conf_A_D=1;
      }
      else{
        memo=1;
        for (unsigned short ind=0; ind<ind_conf_A_D;ind++){
          if( (abs((indice_riga+l-confine_A_D[0][ind]))<marg) && (abs((
              m-confine_A_D[1][ind]))<marg)){
3000        memo=0;
            eliminati_A_D[ind]=eliminati_A_D[ind]+1;
            if((indice_riga+l)<=confine_A_D[0][ind] && m>=confine_A_D
                [1][ind]){
              confine_A_D[0][ind]=indice_riga+l;
              confine_A_D[1][ind]=m;
            }
          }
        }
        if(memo==1){
        %memorizzo i valori attuali nel vettore
3010      confine_A_D[0][ind_conf_A_D]=indice_riga+l;
          confine_A_D[1][ind_conf_A_D]=m;
          ind_conf_A_D=ind_conf_A_D+1;
        }
      }
    }


    %ALGORITMO DI RIDUZIONE DEI PUNTI
    %3    %3    %3    %3    %3    %3    %3    %3
3020  else if ((vettore_contrasto[0]==0) && (vettore_contrasto[1]==1) &&
        (vettore_contrasto[2]==0) && (vettore_contrasto[3]==1)){
      %printf("B_S->(%d,%d)\n",indice_riga+l,m);
      if(ind_conf_B_S==0){ %se l'elemento trovato è il primo
      %allora viene  memorizzato direttamente - sto supponendo
      %quindi che il primo elemento trovato sia sempre buono...
      %ovvero suppongo che non ci siano "macchie" che
      %determinino un falso allarme
        confine_B_S[0][0]=indice_riga+l;
        confine_B_S[1][0]=m;
        ind_conf_B_S=1;
3030  }
      else{
        memo=1;
        for (unsigned short ind=0; ind<ind_conf_B_S;ind++){
          if( (abs((indice_riga+l-confine_B_S[0][ind]))<marg) && (abs((
              m-confine_B_S[1][ind]))<marg)){
            memo=0;
            eliminati_B_S[ind]=eliminati_B_S[ind]+1;
            if((indice_riga+l)>=confine_B_S[0][ind] && m<=confine_B_S
                [1][ind]){
              confine_B_S[0][ind]=indice_riga+l;
```

```
                    confine_B_S[1][ind]=m;
                  }

                }
              }
              if(memo==1){
              %memorizzo i valori attuali nel vettore
                confine_B_S[0][ind_conf_B_S]=indice_riga+l;
                confine_B_S[1][ind_conf_B_S]=m;
                ind_conf_B_S=ind_conf_B_S+1;
              }
            }
          }
          %ALGORITMO DI RIDUZIONE DEI PUNTI
          %4    %4    %4    %4    %4    %4
          else if ((vettore_contrasto[0]==0) && (vettore_contrasto[1]==1) &&
                (vettore_contrasto[2]==1) && (vettore_contrasto[3]==0)){
            %printf("B_D->(%d,%d)\n",indice_riga+l,m);
            if(ind_conf_B_D==0){
            %se l'elemento trovato è il primo allora viene
            %memorizzato direttamente - sto supponendo quindi
            %che il primo elemento trovato sia sempre buono...
            %ovvero suppongo che non ci siano "macchie"
            %che detrminino un falso allarme
              confine_B_D[0][0]=indice_riga+l;
              confine_B_D[1][0]=m;
              ind_conf_B_D=1;
            }
            else{
              memo=1;
              for (unsigned short ind=0; ind<ind_conf_B_D;ind++){
                if( (abs(indice_riga+l-confine_B_D[0][ind])<marg) && (abs(m-
                      confine_B_D[1][ind])<marg)){ %coefficienti vicini
                  memo=0;
                  eliminati_B_D[ind]=eliminati_B_D[ind]+1;
                  if((indice_riga+l)>=confine_B_D[0][ind] && m>=confine_B_D
                        [1][ind]){
                    confine_B_D[0][ind]=indice_riga+l;
                    confine_B_D[1][ind]=m;
                  }
                }
              }
              if(memo==1){
              %memorizzo i valori attuali nel vettore
                confine_B_D[0][ind_conf_B_D]=indice_riga+l;
                confine_B_D[1][ind_conf_B_D]=m;
                ind_conf_B_D=ind_conf_B_D+1;
              }
            }
          }



        }
      } %chiude for (unsigned int m=0;m<pixelX;m++)
```

```
      %(for sulle colonne del pezzetto plus in analisi)

    }% chiude for (unsigned int l=marg; l<=3*marg;l++)
    %(for sulle righe del pezzetto plus in analisi)

  } % chiude if(blocchi_rettangolari!=0)
  %(analisi sui blocchetti_rettangolari)


3100



  %PARTE C - COPIO LE ULTIME  2*MARG RIGHE DEL PEZZETTO IN ANALISI
  %NELLE PRIME 2*MARG RIGHE DEL PEZZETTO_PLUS

  % tale parte viene eseguita solo se costruisco il vettore plus,
  %ovvero solo se mi interessa cercare il rettangolo di
3110 %posizionamento per cui non mi interessa seguire questa
  %strada se localizzazione=0;

  %if( localizzazione==0 || ingresso==1){

    for (unsigned int l=0; l<2*marg;l++){
      for (unsigned int m=0;m<pixelX;m++){
        vettore_Y_plus[l][m]=vettore_Y[2*SIZE-2*marg+l][m];
        vettore_Cr_plus[l][m]=vettore_Cr[2*SIZE-2*marg+l][m];
        vettore_Cb_plus[l][m]=vettore_Cb[2*SIZE-2*marg+l][m];
3120    }
    }


  }% chiude if del controllo di localizzazione, ovvero quando
  %localizzazione diviene 1, non si effettuerà più la ricerca del
  %rettangolo di posizionamento
  %(algoritmo a croce e algoritmo di riduzione dei punti )


3130

  %ALGORITMO DELLA FINE RICERCA MODELLO DI POSIZIONAMENTO
  %&& INIZIO RICERCA INTERVALLI DI ANALISI CODICE

  %se sono a fine blocchetto_rettangolare e ho iniziato a trovare i
  %primi punti del margine inferiore, mi aspetto che nel blocco
  %rettangolare successivo ci continueranno ad essere punti di angolo
  %che miglioreranno la mia stima di posizione degli angoli.
  %Per cui se mi trovo in questa situazione,devo effettuare l'analisi
  %per la ricerca del blocco rettangolare anche nei 16 pixel
3140 %successivi e poi iniziare la ricerca dei punti alpha,beta,gamma.
  %sullo stessso blocco invece se gli ultimi punti del margine
  %inferiore li trovo all'inizio del blocchetto_rettangolare,
  %la ricerca di alpha,beta,gamma...deve essere fatta a
  %partire dal blocco attuale;
```

```
    if((ind_conf_B_D>=1)&&(localizzazione==0) ){
      %printf("\nSecondo if --> blocco=%d\n",blocchi_rettangolari);
      ingresso=ingresso+1;
      %MEMORIZZO IL PEZZETTO RETTANGOLARE DI 16 RIGHE TEMPORANEAMENTE
      %IL PEZZETTO CHE SI MEMORIZZA È QUELLO PRECEDENTE
      %ALL'ANALISI DI alpha,beta....

      if(ingressoUnico==0){
        for (unsigned short l=0;l<2*SIZE;l++){
          for (unsigned short m=0;m<pixelX;m++){
            R_temp[l][m]=R[l][m];
            G_temp[l][m]=G[l][m];
            B_temp[l][m]=B[l][m];
          }
        }
        ingressoUnico=1;
      }
    }




    %ANALISI DEL RETTANGOLO DI POSIZIONAMENTO
    %RICERCA DI alpha,beta,gamma,delta,A,BI,ratio



    if(ingresso==2 && localizzazione==0){
    %se si entra in questo ciclo vuol dire che la ricerca del
    %quadrato di posizionamento è finita,
    %ovvero suppongo di trovare sempre l'ultimo angolo...

      %printf("\nblocchi_rett=%d\n",blocchi_rettangolari);

      printf("\ndimConfini=%d\t\tmarg=%d\t\tSoglie Nero=(%d,%d,%d)\t\
            tdifferenzaContrasto=%d",dimConfini,marg,soglia_R,soglia_G,
            soglia_B,differenza);

      printf("\n\nCoordinate punti di margine in alto a sinistra:\n");
      for (unsigned short g=0; g<ind_conf_A_S;g++){
        printf("(%d,%d)-(%d)\t",confine_A_S[0][g],confine_A_S[1][g],
              eliminati_A_S[g]);
      }


      printf("\n\nCoordinate punti di margine in alto a destra:\n");
      for (unsigned short h=0; h<ind_conf_A_D;h++){
        printf("(%d,%d)-(%d)\t",confine_A_D[0][h],confine_A_D[1][h],
              eliminati_A_D[h]);
      }

      printf("\n\nCoordinate punti di margine in basso a sinistra:\n");
```

```
for (unsigned short i=0; i<ind_conf_B_S;i++){
  printf("(%d,%d)-(%d)\t",confine_B_S[0][i],confine_B_S[1][i],
       eliminati_B_S[i]);
}

printf("\n\nCoordinate punti di margine in basso a destra:\n");
for (unsigned short i=0; i<ind_conf_B_D;i++){
  printf("(%d,%d)-(%d)\t",confine_B_D[0][i],confine_B_D[1][i],
       eliminati_B_D[i]);
}



%if(ind_conf_A_S>=1){
  %unsigned short e=0;
  %while ((e<ind_conf_A_S) && (eliminati_A_S[e]<marg)){
    %e=e+1;
  %}
%}

%margineriga[0]=
%printf("--------------\n");
%printf("Coordinate di margine selezionate:")


printf("\n\n\n");

%alpha - riga superiore
if(ind_conf_A_S>=1 && ind_conf_A_D>=1){
  if((abs(confine_A_S[0][0]-confine_A_D[0][0]))<margine_coordinate){
    alpha=round((double)(confine_A_S[0][0]+confine_A_D[0][0])/2);
    printf("\nverifica su alpha correttamente eseguita");
  }
  else{
    printf("\ncodice non riconosciuto --> modello di rivelazione
         posizione fallito: verifica su alpha non riuscita");
    exit(1);
  }
}

else if (ind_conf_A_S>=1 && ind_conf_A_D==0){
  alpha=confine_A_S[0][0];
  printf("\nverifica su alpha non eseguibile --> assente l'angolo A_D")
       ;
}

else if(ind_conf_A_S==0 && ind_conf_A_D>=1 ){
  alpha=confine_A_D[0][0];
  printf("\nverifica su alpha non eseguibile --> assente l'angolo A_S")
       ;
}
else if (ind_conf_A_S==0 && ind_conf_A_D==0){
  printf("\ncodice non riconosciuto --> modello di rivelazione
       posizione fallito: angoli in alto non trovati");
  exit(1);
```

```
}


%Beta - riga inferiore
if(ind_conf_B_S>=1 && ind_conf_B_D>=1){
  if((abs(confine_B_S[0][0]-confine_B_D[0][0]))<margine_coordinate){
  beta=round((double)(confine_B_S[0][0]+confine_B_D[0][0])/2);
  printf("\nverifica su beta correttamente eseguita");
  }
  else{
    printf("\ncodice non riconosciuto --> modello di rivelazione
          posizione fallito:verifica su beta non riuscita");
    exit(1);
  }
}

else if (ind_conf_B_S>=1 && ind_conf_B_D==0){
  beta=confine_B_S[0][0];
  printf("\nverifica su beta non eseguibile --> assente l'angolo B_D");
}

else if(ind_conf_B_S==0 && ind_conf_B_D>=1 ){
  beta=confine_B_D[0][0];
  printf("\nverifica su beta non eseguibile --> assente l'angolo B_S");
}
else if (ind_conf_B_S==0 && ind_conf_B_D==0){
  printf("\ncodice non riconosciuto --> modello di rivelazione
        posizione fallito: angoli in alto non trovati");
  exit(1);
}



%gamma - colonna di sinistra
if(ind_conf_A_S>=1 && ind_conf_B_S>=1){
  if((abs(confine_A_S[1][0]-confine_B_S[1][0]))<margine_coordinate){
    gamma=round((double)(confine_A_S[1][0]+confine_B_S[1][0])/2);
    printf("\nverifica su gamma correttamente eseguita");
  }
  else{
    printf("\ncodice non riconosciuto --> modello di rivelazione
          posizione fallito:verifica su gamma non riuscita");
    exit(1);
  }

}

else if (ind_conf_A_S>=1 && ind_conf_B_S==0){
  gamma=confine_A_S[1][0];
  printf("\nverifica su gamma non eseguibile --> assente l'angolo B_S")
        ;
}

else if(ind_conf_A_S==0 && ind_conf_B_S>=1 ){
```

```
    gamma=confine_B_S[1][0];
    printf("\nverifica su gamma non eseguibile --> assente l'angolo A_S")
        ;
  }
  else if (ind_conf_A_S==0 && ind_conf_B_S==0){
    printf("\ncodice non riconosciuto --> modello di rivelazione
        posizione fallito: angoli in alto non trovati");
    exit(1);
  }


  %delta - colonna di destra
  if(ind_conf_A_D>=1 && ind_conf_B_D>=1){
    if((abs(confine_A_D[1][0]-confine_B_D[1][0]))<margine_coordinate){
      delta=round((double)(confine_A_D[1][0]+confine_B_D[1][0])/2);
      printf("\nverifica su delta correttamente eseguita");
    }
    else{
      printf("\ncodice non riconosciuto --> modello di rivelazione
          posizione fallito:verifica su delta non riuscita");
      exit(1);
    }

  }

  else if (ind_conf_A_D>=1 && ind_conf_B_D==0){
    delta=confine_A_D[1][0];
    printf("\nverifica su delta non eseguibile --> assente l'angolo B_D")
        ;
  }

  else if(ind_conf_A_D==0 && ind_conf_B_D>=1 ){
    delta=confine_B_D[1][0];
    printf("\nverifica su delta non eseguibile --> assente l'angolo A_D")
        ;
  }
  else if (ind_conf_A_D==0 && ind_conf_B_D==0){
    printf("\ncodice non riconosciuto --> modello di rivelazione
        posizione fallito: angoli in alto non trovati");
    exit(1);
  }



  %analisi dei risultati;
  printf("\n\nalpha=%d\tbeta=%d\tgamma=%d\tdelta=%d\t
      blocchi_rettangolari=%d",alpha,beta,gamma,delta,
      blocchi_rettangolari);
  %(il % non è un commento ma è l'operatore del printf)
  A=(float)(delta-gamma+1);
  BI=(float)(beta-alpha+1);
  ratio=A/BI;
  printf("\nA=%.2f\tBI=%.2f\tratio=%.2f",A,BI,ratio);
```

```
%(il % non è un commento ma è l'operatore del printf)
if(alpha>beta || gamma>delta ||((beta-alpha)>(delta-gamma)) || ratio
      <2.6 || ratio >5 || A>pixelX || BI>pixelY/2){
  printf("\ncodice non riconosciuto --> modello di rivelazione
        posizione fallito: alpha,beta,gamma,delta,A B non validi o
        ratio non conforme");
  exit(1);
}

%calcolo punti di analisi: ci saranno
%2 intervalli di righe e 5 intervalli di colonne

%Righe:
%[(beta+5B/7) : (beta+B)]
%[(beta+8B/7) : (beta+10B/7)]


%Colonne:
%[(gamma+3A/50):(gamma+7A/50)]
%[(gamma+13A/50):(gamma+17A/50)]
%[(gamma+23A/50):(gamma+27A/50)]
%[(gamma+33A/50):(gamma+37A/50)]
%[(gamma+43A/50):(gamma+47A/50)]

double fattorePrimaRiga=ratio/3.57;
double fattoreSecondaRiga=ratio/3.57;

righe_analisi[0]=round((double)((beta+5*BI/7)*fattorePrimaRiga));
righe_analisi[1]=round((double)(beta+BI)*fattorePrimaRiga);

righe_analisi[2]=round((double)((beta+10*BI/7)*fattoreSecondaRiga));
righe_analisi[3]=round((double)((beta+12*BI/7)*fattoreSecondaRiga));



colonne_analisi[0]=round((double)(gamma+3*A/50));
colonne_analisi[1]=round((double)(gamma+7*A/50));

colonne_analisi[2]=round((double)(gamma+13*A/50));
colonne_analisi[3]=round((double)(gamma+17*A/50));

colonne_analisi[4]=round((double)(gamma+23*A/50));
colonne_analisi[5]=round((double)(gamma+27*A/50));

colonne_analisi[6]=round((double)(gamma+33*A/50));
colonne_analisi[7]=round((double)(gamma+37*A/50));

colonne_analisi[8]=round((double)(gamma+43*A/50));
colonne_analisi[9]=round((double)(gamma+47*A/50));



blocchi_rettangolari_analisi[0]=floor((double)(righe_analisi[0]/(2*SIZE
      )));
```

```
    blocchi_rettangolari_analisi[1]=floor((double)(righe_analisi[1]/(2*SIZE
        )));

    blocchi_rettangolari_analisi[2]=floor((double)(righe_analisi[2]/(2*SIZE
        )));
    blocchi_rettangolari_analisi[3]=floor((double)(righe_analisi[3]/(2*SIZE
        )));

3400

    %RIGHE DI ANALISI DEL BIANCO PER LA SOGLIA ADATTATIVA
    %LE COLONNE SONO LE STESSE DI PRIMA

    righe_soglia[0]=round((double)((beta+3*BI/35)));
    righe_soglia[1]=round((double)((beta+4*BI/35)));
    blocchi_rettangolari_soglia[0]=floor((double)((righe_soglia[0])/(2*SIZE
        )));
    blocchi_rettangolari_soglia[1]=floor((double)((righe_soglia[1])/(2*SIZE
        )));


3410    %printf("\nrighe_analisi -> (%1.f:%1.f) (%1.f:%1.f)\t\t\
            tcolonne_analisi --> (%d:%d) (%d:%d) (%d:%d) (%d:%d) (%d:%d)",
            round((double)((beta+5*BI/7))),round((double)(beta+BI)),round((
            double)((beta+10*BI/7))),round((double ((beta+12*BI/7))),
            colonne_analisi[0], colonne_analisi[1], colonne_analisi[2],
            colonne_analisi[3], colonne_analisi[4], colonne_analisi[5],
            colonne_analisi[6], colonne_analisi[7], colonne_analisi[8],
            colonne_analisi[9]);
    %printf("\nrighe_analisi spostate-> (%d:%d) (%d:%d)\t",righe_analisi
            [0],righe_analisi[1],righe_analisi[2],righe_analisi[3]);
    %printf("blocchi rettangolari --> (%d,%d,%d %d)" ,
            blocchi_rettangolari_analisi[0] ,blocchi_rettangolari_analisi[1],
            blocchi_rettangolari_analisi[2], blocchi_rettangolari_analisi[3])
            ;
    %printf("\nrighe_soglia --> (%d,%d)\t",righe_soglia[0],righe_soglia[1])
            ;
    %printf("blocchi rettangolari soglia --> (%d,%d)\n",
            blocchi_rettangolari_soglia[0],blocchi_rettangolari_soglia[1]);
    localizzazione=1;
  }%chiude l'if sulla ricerca del rettangolo di posizionamento


3420


  %printf("\n\nblocchi_rett=%d loc=%d\n",blocchi_rettangolari,
        localizzazione);
  %printf("(%d,%d,%d,%d)\n\n",blocchi_rettangolari_analisi[0],
        blocchi_rettangolari_analisi[1],blocchi_rettangolari_analisi[2],
        blocchi_rettangolari_analisi[3]);

  %RICERCO E ANALIZZO I PUNTI DI INTERESSE NEL CODICE
  %in questo ciclo si entra solo nei casi di interesse
  %(ovvero per i blocchi_rettangolari di interesse)
  if((localizzazione>=1)){
```

```
        indice_riga=blocchi_rettangolari*2*SIZE;
3430
        %CALCOLO SUL BLOCCO PRECEDENTE MEMORIZZATO IN R_temp,G_temp,B_temp
        while (localizzazione==1){
          unsigned short indice_riga_prec=(blocchi_rettangolari-1)*2*SIZE;

          if(((blocchi_rettangolari-1)>=blocchi_rettangolari_soglia[0] && (
                blocchi_rettangolari-1)<=blocchi_rettangolari_soglia[1])){
            for (unsigned short r=0; r<2*SIZE; r++){
              if((r+indice_riga_prec)>=righe_soglia[0] && (r+indice_riga_prec)
                    <=righe_soglia[1]){
                for (unsigned short i=0;i<5;i++){
                  %printf("\n");
3440              for (unsigned short c=colonne_analisi[2*i]; c<=
                        colonne_analisi[2*i+1];c++){
                    somma_R_bianco[i]=somma_R_bianco[i]+R_temp[r][c];
                    somma_G_bianco[i]=somma_G_bianco[i]+G_temp[r][c];
                    somma_B_bianco[i]=somma_B_bianco[i]+B_temp[r][c];
                    %printf("\n0a) (r+indice_riga_prec,c)=(%d,%d)\t\tRGB_bianco
                          =(%d,%d,%d)",indice_riga_prec+r,c,R_temp[r][c],G_temp
                          [r][c],B_temp[r][c]);
                  }
                }
              }
            }
          }
3450

          if(((blocchi_rettangolari-1)>=blocchi_rettangolari_analisi[0] && (
                blocchi_rettangolari-1)<=blocchi_rettangolari_analisi[1])){
            for (unsigned short r=0; r<2*SIZE; r++){
              if((r+indice_riga_prec)>=righe_analisi[0] && (r+indice_riga_prec
                    )<=righe_analisi[1]){
                for (unsigned short i=0;i<5;i++){
                  %printf("\n");
                  for (unsigned short c=colonne_analisi[2*i]; c<=
                        colonne_analisi[2*i+1];c++){
                    somma_R[i]=somma_R[i]+R_temp[r][c];
                    somma_G[i]=somma_G[i]+G_temp[r][c];
3460                somma_B[i]=somma_B[i]+B_temp[r][c];
                    %printf("\n1a) (r+indice_riga_prec,c)=(%d,%d)\t\tRGB=(%d,%d
                          ,%d)",indice_riga_prec+r,c,R_temp[r][c],G_temp[r][c],
                          B_temp[r][c]);
                  }
                }
              }
            }
          }

          %printf("\n\n\n");
          if(((blocchi_rettangolari-1)>=blocchi_rettangolari_analisi[2] && (
                blocchi_rettangolari-1)<=blocchi_rettangolari_analisi[3])){
3470        for (unsigned short r=0; r<2*SIZE; r++){
              if((r+indice_riga_prec)>=righe_analisi[2] && (r+indice_riga_prec
                    )<=righe_analisi[3]){
```

```
            for (unsigned short i=0;i<5;i++){
              %printf("\n");
              for (unsigned short c=colonne_analisi[2*i]; c<=
                    colonne_analisi[2*i+1];c++){
                somma_R[i+5]=somma_R[i+5]+R_temp[r][c];
                somma_G[i+5]=somma_G[i+5]+G_temp[r][c];
                somma_B[i+5]=somma_B[i+5]+B_temp[r][c];
                %printf("\n2a) (r+indice_riga_prec,c)=(%d,%d)\t\tRGB=(%d,%d
                      ,%d)",indice_riga_prec+r,c,R_temp[r][c],G_temp[r][c],
                      B_temp[r][c]);
              }
            }
          }
        }

        localizzazione=2;
      }



      if((blocchi_rettangolari>=blocchi_rettangolari_soglia[0] &&
          blocchi_rettangolari<=blocchi_rettangolari_soglia[1])){
        for (unsigned short r=0; r<2*SIZE; r++){
          if((r+indice_riga)>=righe_soglia[0] && (r+indice_riga)<=
              righe_soglia[1]){
            for (unsigned short i=0;i<5;i++){
              %printf("\n");
              for (unsigned short c=colonne_analisi[2*i]; c<=colonne_analisi
                    [2*i+1];c++){
                somma_R_bianco[i]=somma_R_bianco[i]+R[r][c];
                somma_G_bianco[i]=somma_G_bianco[i]+G[r][c];
                somma_B_bianco[i]=somma_B_bianco[i]+B[r][c];
                %printf("\n0b) (r+indice_riga,c)=(%d,%d)\t\tRGB_bianco=(%d,%d
                      ,%d)",indice_riga+r,c,R[r][c],G[r][c],B[r][c]);
              }
            }
          }
        }
      }


      if((blocchi_rettangolari>=blocchi_rettangolari_analisi[0] &&
          blocchi_rettangolari<=blocchi_rettangolari_analisi[1])){
        for (unsigned short r=0; r<2*SIZE; r++){
          if((r+indice_riga)>=righe_analisi[0] && (r+indice_riga)<=
              righe_analisi[1]){
            for (unsigned short i=0;i<5;i++){
              %printf("\n");
              for (unsigned short c=colonne_analisi[2*i]; c<=colonne_analisi
                    [2*i+1];c++){
                somma_R[i]=somma_R[i]+R[r][c];
                somma_G[i]=somma_G[i]+G[r][c];
                somma_B[i]=somma_B[i]+B[r][c];
```

```
                %printf("\n1b) (r+indice_riga,c)=(%d,%d)\t\tRGB=(%d,%d,%d)",
                        indice_riga+r,c,R[r][c],G[r][c],B[r][c]);
              }
            }
          }
        }
      }

      %printf("\n\n\n");
      if((blocchi_rettangolari>=blocchi_rettangolari_analisi[2] &&
          blocchi_rettangolari<=blocchi_rettangolari_analisi[3])){
        for (unsigned short r=0; r<2*SIZE; r++){
          if((r+indice_riga)>=righe_analisi[2] && (r+indice_riga)<=
              righe_analisi[3]){
            for (unsigned short i=0;i<5;i++){
              %printf("\n");
              for (unsigned short c=colonne_analisi[2*i]; c<=colonne_analisi
                  [2*i+1];c++){
                somma_R[i+5]=somma_R[i+5]+R[r][c];
                somma_G[i+5]=somma_G[i+5]+G[r][c];
                somma_B[i+5]=somma_B[i+5]+B[r][c];
                %printf("\n2b) (r+indice_riga,c)=(%d,%d)\t\tRGB=(%d,%d,%d)",
                        indice_riga+r,c,R[r][c],G[r][c],B[r][c]);
              }
            }
          }
        }
      }
  }


}%chiude l'if((x+1)%(indice_moltiplicazione*macroBlocchiNum_X)==0)


  %if(blocchi_rettangolari>blocchi_rettangolari_analisi[3]){
    %printf("decodifica dello SPinV Code corretta");
    %exit(1);
  %}

}%chiude il for globale su x (x iterazioni totali)


  printf("\n\nSoglia Bianco/Nero:\t");
  for (unsigned short i=0;i<5;i++){
    R_bianco_soglia[i]=somma_R_bianco[i]/(2*((righe_soglia[1]-righe_soglia
        [0]+1)*(colonne_analisi[2*i+1]-colonne_analisi[2*i]+1)));
    G_bianco_soglia[i]=somma_G_bianco[i]/(2*((righe_soglia[1]-righe_soglia
        [0]+1)*(colonne_analisi[2*i+1]-colonne_analisi[2*i]+1)));
    B_bianco_soglia[i]=somma_B_bianco[i]/(2*((righe_soglia[1]-righe_soglia
        [0]+1)*(colonne_analisi[2*i+1]-colonne_analisi[2*i]+1)));
    %printf("(%d,%d,%d)\t",R_bianco_soglia[i],G_bianco_soglia[i],
        B_bianco_soglia[i]);

  }
```

```
    printf("\nSotto Soglia --> Nero --> 1\t\t\t\tSopra Soglia --> Bianco -->
        0");


    unsigned short f;
    unsigned short g;
    unsigned short h;
    printf("\n\n\n");
    for (unsigned short i=0; i<10;i++){
3570
      %indice relativi alle righe
      if(i<5)
        f=0;
      else {
        f=2;
      }

      %indici relativi alle colonne
      switch (i){
3580    case 0:case 5:
          g=0;

        case 1:case 6:
          g=2;

        case 2:case 7:
          g=4;

        case 3:case 8:
3590      g=6;

        case 4:case 9:
          g=8;
      }

      %indici relativi alla valuzione bianco/nero
      switch (i){
        case 0:case 5:
          h=0;
3600
        case 1:case 6:
          h=1;

        case 2:case 7:
          h=2;

        case 3:case 8:
          h=3;

3610    case 4:case 9:
          h=4;
      }
```

```
      R_medio[i]=somma_R[i]/( (righe_analisi[f+1]-righe_analisi[f]+1) * (
          colonne_analisi[g+1]-colonne_analisi[g]+1));
      G_medio[i]=somma_G[i]/( (righe_analisi[f+1]-righe_analisi[f]+1) * (
          colonne_analisi[g+1]-colonne_analisi[g]+1));
      B_medio[i]=somma_B[i]/( (righe_analisi[f+1]-righe_analisi[f]+1) * (
          colonne_analisi[g+1]-colonne_analisi[g]+1));

      if(R_medio[i]>=R_bianco_soglia[h] && G_medio[i]>=G_bianco_soglia[h] &&
          B_medio[i]>=B_bianco_soglia[h])
3620    finalCode[i]=0;
      else if(R_medio[i]<R_bianco_soglia[h] && G_medio[i]<G_bianco_soglia[h] &&
           B_medio[i]<B_bianco_soglia[h])
        finalCode[i]=1;
      else
        finalCode[i]=2;

      %printf("\nsomma_R[%d]=%d\tsomma_G[%d]=%d\tsomma_B[%d]=%d",i,somma_R[i],i
          ,somma_G[i],i,somma_B[i]);
      %printf("(%d,%d,%d)\t",R_medio[i],G_medio[i],B_medio[i]);
    }

3630

    if((finalCode[0]==0 || finalCode[0]==1) && (finalCode[1]==0 || finalCode
        [1]==1) && (finalCode[2]==0 || finalCode[2]==1) && (finalCode[3]==0
         || finalCode[3]==1) && (finalCode[4]==0 || finalCode[4]==1) && (
        finalCode[5]==0 || finalCode[5]==1) && (finalCode[6]==0 || finalCode
        [6]==1) && (finalCode[7]==0 || finalCode[7]==1) && (finalCode[8]==0
         || finalCode[8]==1) && (finalCode[9]==0 || finalCode[9]==1) ){
      unsigned short finalDec=0;
      for (unsigned short c=0; c<10;c++){
        finalDec=finalDec+(finalCode[c])*pow(2,10-c-1);
      }


      %STAMPA DEI RISULTATI

3640  printf("\n\n");
      printf("\t\t\t-----------------------------------------------------------\n");
      printf("\t\t\t");
      for (unsigned short z=0;z<10;z++){
        printf("%d    ",finalCode[z]);
        if(z==4)
          printf("\t\t");
      }
      printf("--> %d",finalDec);
      printf("\n\t\t\t-----------------------------------------------------------");
3650



    }
```

```
    %verifica finale
3660 if((ByteElemento(PuntFile,scansione)==0xFF && ByteElemento(PuntFile,
        scansione+1)==0xD9)  || (ByteElemento(PuntFile,scansione+1)==0xFF &&
        ByteElemento(PuntFile,scansione+2)==0xD9))
      printf("\n\n\nVERIFICA BYTE STUFFING TERMINATA CON SUCCESSO");

    fclose(PuntFile);

    printf("\n\nSONO ARRIVATO ALLA FINE DEL MAIN\n\n\n\n\n\n");
    return 0;



3670

  }%chiude ciclo main %chiude ciclo main   %chiude ciclo main   %chiude ciclo
       main   %chiude ciclo main


  %%%%%%%%%%%%%%%%%   FUNZIONI %%%%%%%%%%%%%%%%%


  %Restituisce un intero che rapresenta il byte
  %alla posizione elemento
  int ByteElemento(FILE *Puntatore,unsigned int elemento){
3680  unsigned int pokerhex;
    fseek(Puntatore, elemento, SEEK_SET);
    fread(&pokerhex,4,1, Puntatore);

    int tre=floor(pokerhex/(0x1000000));

    int num1=pokerhex-tre*(0x1000000);

    int due=floor((num1)/(0x10000));

3690  int num2=num1-due*(0x10000);
    int uno=floor((num2)/(0x100));


    int zero=floor(num2-uno*(0x100));

    return zero;

  }

3700

  %FUNZIONE RICERCADOPPIA: ritorna l'elemento del puntatore di
  %poszione in cui è presente la coppia di chiavi ricercate
  int ricercaDoppia(FILE *Puntatore,int dim, unsigned int puntoanalisi,int
      chiave1,int chiave2){
    unsigned int n;
    int a;
    int b;

    for (n=puntoanalisi; n<dim; n++){
```

```
3710      a=ByteElemento(Puntatore, n);
          %printf("a=%x\t",a);

          b=ByteElemento(Puntatore, n+1);
          %printf("b=%x\n",b);

          if(a==chiave1 && b==chiave2){
            return n;
            break;
          }
3720    }
        return -1;
      }


      %FUNZIONE unioneHex, prende in input i due elementi esadecimali e
      %ritorna un unico vlore che ne rappresenta l'unione
      int unioneHex(int primo,int secondo){
        return primo*(0x100)+secondo;
      }
3730


      %FUNZIONE separaHexPrimo,prende in input un elemento esadecimale
      %(2 simboli) e ritorna il primo elemento
      int separaHexPrimo(int coppia){
        int primo=floor(coppia/(0x10));
        return primo;
      }


3740  %FUNZIONE separaHexSecondo,prende in input un elemento esadecimale
      %(2 simboli) e ritorna il secondo elemento
      int separaHexSecondo (int coppia){
        int primo=floor(coppia/(0x10));
        int secondo=coppia-primo*(0x10);
        return secondo;
      }



3750

      %FUNZIONE stampaVettoreINT prendi in input il vettore
      %da stampare e la dimensione
      void stampaVettoreInt(const int *inizioVettore,int grandezza){
        for (int ii=1;ii<=grandezza;ii++){
          printf("%8d",*(inizioVettore+ii-1));
          if (ii%8==0)
            printf("\n");
        }
      }
3760


      %FUNZIONE CHE STAMPA I BIT DELL'INTERO int PASSATO COME ARGOMENTO
```

```
%della lunghezza dell'int passato come argomento
void stampaBits(int codice, int numbit){
  int c, displayMask=1<<(numbit-1);

  for (c=(32-numbit+1); c<=32;c++){
    putchar(codice & displayMask ? '1' : '0');
    codice<<=1;

    if (c %32 ==0){
      %putchar(' ');
      printf("");
    }

  }

}



%Funzione DECIMALE - PRENDE IN INGRESSO  L'INDICE DI ANALISI DEL
%FILE BINARIO, LA LUNGHEZZA DEL CODICE DA PRELEVARE E IL VETTORE
%DEI DATI. RITORNA IL CODICE IN DECIMALE

unsigned int funzioneDecimale(unsigned short scabin,unsigned short lunghezza,
      const short *dati){
  int decimale=0;
  for (int c=0; c<lunghezza;c++){
    decimale=decimale+(*(dati+scabin+c))*pow(2,lunghezza-c-1);
  }
  return decimale;

}




%FUNZIONE PROCEDURANUMNEGATIVO - PRENDE IN INGRESSO L'INDICE DI
%SCANSIONE DEL VETTORE DATI BINARIO,  IL NUMERO DI BITI CHE DEVO
%PRENDERE (OVVERO HUFFVAL) E IL VETTORE DEI DATI
%RITORNA IL VALORE IN DECIMALE DECODIFICATO
unsigned int proceduraNumNegativo(unsigned short scabin,unsigned short
      bitsucc,const short *dati){

  int complDueDecimale=0;
  int singolobit;

  for (short c=0;c<bitsucc;c++){
    singolobit=*(dati+scabin+c);
    if(singolobit==0) %complemento a 1 (inversione dei bit)
      singolobit=1;
    else{
      singolobit=0;
```

```
      }
      complDueDecimale=complDueDecimale+singolobit*pow(2,bitsucc-c-1);
3820  }


    return -complDueDecimale;
  }



  % unsigned int proceduraNumNegativo(unsigned short scabin,unsigned short
      bitsucc,short dati[]){
3830
   %int complDueDecimale=0;
   %int singolobit;

   %for (int c=0;c<bitsucc;c++){
   %singolobit=dati[scabin+c];
   %if(singolobit==0) %complemento a 1 (inversione dei bit)
   %singolobit=1;
   %else{
   %singolobit=0;
3840 %}
   %complDueDecimale=complDueDecimale+singolobit*pow(2,bitsucc-c-1);
   %}


   %return -complDueDecimale;
   %}



3850


  %funzione IDCT - prende come argomento la matrice su cui effettuare
  %la dct e la matrice su cui verrà memorizzato il risultato.
  void IDCTBlocchetto (const int *matrice,int *matricerisultato){

    unsigned short x,y,u,v;
    double Cu;
    double Cv;
    double sommatoria=0;
3860
    for(x=0; x<SIZE; x++){
      for (y=0; y<SIZE; y++){

        for (u=0; u<SIZE; u++) {
          for (v=0; v<SIZE; v++){

            if (u==0)
              Cu=1/(sqrt(2));
            else {
3870          Cu=1;
            }
```

```
                if (v==0)
                  Cv=1/(sqrt(2));
                else {
                  Cv=1;
                }

                sommatoria =(*(matrice+u*8+v))*Cu*Cv*cos((2*x+1)*u*M_PI/16)* cos
                      ((2*y+1)*v*M_PI/16) + sommatoria;
3880          }
            }


            *(matricerisultato+x*8+y)=round(0.25*sommatoria);
            sommatoria=0;
          }

        }
      }
```

Bibliografia

[1] The White House Office of the Press Secretary, cur. *Statement by the president regarding the United States decision to stop degrading Global Positioning System accuracy*. 2000 (cit. a p. 1).

[2] Bill Clinton. *Improving the Civilian Global Positioning System (GPS)*. 2000 (cit. a p. 1).

[3] Ahmed El Rabbany. *Introduction to GPS - The Global Positioning System*. Artech House Mobile Communications Series, 2002 (cit. a p. 1).

[4] Guoqiang Mao e Bar Fidan. *Localization Algorithms and Strategies for Wireless Sensor Networks*. Information Science Reference, 2009 (cit. alle pp. 4, 7, 14, 17).

[5] Krzysztof W. Kolodziej e Johan Hjelm. *Local Positioning Systems - LBS Applications and Services*. Taylor e Francis Group, 2006 (cit. alle pp. 8, 11, 13).

[6] Nel Samama. *Global Positioning System - Technologies and Performance*. John Wiley e Sons, 2008 (cit. alle pp. 8, 9, 11, 13).

[7] Martin Eladio et al. "Positioning Technologies in Location-Based Services". In: *Location-Based Services Handbook - Applications, Technologies, and Security*. Taylor e Francis Groups, 2011 (cit. alle pp. 8, 11, 13, 15, 17, 23).

[8] Patrick Loschmidt, Georg Gaderer e Thilo Sauter. "Clock Synchronization for Wireless Positioning of COTS Mobile Nodes". In: *Precision Clock Synchronization for Measurement, Control and Communication. IEEE International Symposium* (2007), pp. 64 –69 (cit. alle pp. 12, 28).

[9] Liu Yunhao e Yang Zheng. *Location, Localization, and Localizability - Location-awareness Technology for Wireless Networks*. Springer New York Dordrecht Heidelberg London, 2011 (cit. alle pp. 15, 17).

[10] Cristiano Monti et al. "Indoor Localization System based on Wireless Sensor Networks". In: *The Fifth IEEE International Conference on Mobile Adhoc and Sensor Systems* 7133 (2009) (cit. a p. 17).

[11] Eximia Srl, cur. *Tecnologia RTL Ubisense*. 2010. URL: www.eximia.it (cit. a p. 25).

[12] André Gunther e Christian Hoene. "Measuring Round Trip Times to Determine the Distance between WLAN Nodes". In: 3462 (2005), pp. 768 –779 (cit. a p. 27).

[13] AT & T Laboratories Cambridge. *The Bat System*. URL: http://www.cl.cam.ac.uk/research/dtg/research/wiki/BatSystem (cit. a p. 28).

[14] Paramvir Bahl e Venkata N. Padmanabhan. "RADAR: An In-Building RF-based User Location and Tracking System". In: *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE* 2 (2000), pp. 775 –784 (cit. a p. 29).

[15] Microsoft Research. *Radar*. URL: http://research.microsoft.com/en-us/projects/radar/ (cit. a p. 29).

[16] A. Roxin et al. "Survey of Wireless Geolocation Techniques". In: *Globecom Workshops, IEEE* (2007), pp. 1 –9 (cit. a p. 32).

[17] Ekahau. *Ekahau Real Time Location System (RTLS) Overview*. URL: http://www.ekahau.com/ (cit. a p. 34).

[18] Miguel Rodriguez, Juan P. Pece e Carlos J. Escudero. "In-building location using Bluetooth". In: (2005) (cit. a p. 35).

[19] A.K.M. Mahtab Hossain et al. "Indoor Localization using Multiple Wireless Technologies". In: *Mobile Adhoc and Sensor Systems, IEEE Internatonal Conference* (2007), pp. 1 –8 (cit. a p. 35).

[20] M.Ni Lionel et al. "LANDMARC: Indoor Location Sensing Using Active RFID". In: *Pervasive Computing and Communications, IEEE International Conference* (2003), pp. 407 –415 (cit. a p. 36).

[21] Wikipedia l'enciclopedia libera. *Wii*. Versione del 29 Ottobre 2011. 2011. URL: http://it.wikipedia.org/wiki/Wii (cit. a p. 44).

[22] STMicroelectronics NV (STM). *Annual and transition report of foreign private issuers pursuant to sections 13 or 15(d)*. Filed on 03/07/2011 - Filed Period 12/31/2010. 2010 (cit. a p. 44).

[23] Wikipedia l'enciclopedia libera. *STMicroelectronics*. Versione del 22 Ottobre 2011. 2011. URL: http://it.wikipedia.org/wiki/STMicroelectronics (cit. a p. 44).

[24] Paul D. Groves. *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems*. Artech House, 2008 (cit. alle pp. 47, 61, 62).

[25] Cliff Randell, Chris Djiallis e Henk Muller. "Personal Position Measurement Using Dead Reckoning". In: *Wearable Computers, 2003. Proceedings. Seventh IEEE International Symposium* (2005), pp. 166 –173 (cit. a p. 47).

[26] Wikipedia l'enciclopedia libera. *Accelerometro*. Versione del 28 Settembre 2011. 2011. URL: `http://it.wikipedia.org/wiki/Accelerometro` (cit. a p. 50).

[27] Gabriele Rescio. "Progettazione di un circuito integrato di interfaccia a 16 bit per sensori capacitivi a basso consumo di potenza e rumore". Tesi di Laurea Specialistica. Università degli studi di Pavia - Facoltà di Ingegneria - Corso di Laurea in Ingegneria Elettronica, 2007 (cit. a p. 51).

[28] STMicroelectronics. *L3G4200D - MEMS motion sensor: ultra-stable three-axis digital output gyroscope*. Dic. 2010 (cit. a p. 53).

[29] Wikipedia l'enciclopedia libera. *Giroscopio*. Versione del 4 Settembre 2011. 2011. URL: `http://it.wikipedia.org/wiki/Giroscopio` (cit. a p. 53).

[30] Apple. *iPhone e iPad: calibrazione della bussola*. Articolo TS2767. URL: `http://support.apple.com/kb/TS2767?viewlocale=it_IT&locale=it_IT` (cit. a p. 57).

[31] Sebastian O.H. Madgwick. *An efficient orientation filter for inertial and inertial/magnetic sensor arrays*. 2010 (cit. a p. 60).

[32] Francesco Novelli. *Programmare applicazioni per iPhone e iPad*. Edizioni Fag Milano, 2010 (cit. alle pp. 60, 98, 122).

[33] Andrea Corvi. *Biomeccanica del passo*. Laboratorio di Bioingegneria Industriale - Dipartimento di Meccanica e Tecnologie Industriali - Università degli Studi di Firenze. URL: `http://www.masteringegneriaclinica.it/Bioingegneria/lucidi/Bioingegneria_med/Passo.PDF` (cit. a p. 75).

[34] J. W. Kim H. S. Hong J. M. Lee S. H. Shin C. G. Park. "Adaptive Step Length Estimation Algorithm Using Low-CostMEMS Inertial Sensors". In: *IEEE Sensors Applications Symposium* (2007), pp. 1 –5 (cit. alle pp. 75–77, 176).

[35] Antonio Grillo et al. "High Capacity Colored Two Dimensional Codes". In: *Computer Science and Information Technology (IMCSIT) - Proceedings of the 2010 International Multiconference* (2010), pp. 709 –716 (cit. alle pp. 95, 99).

[36] Secur Edge. *Raffronto Codici Bidimensionali*. 2006 (cit. a p. 95).

[37] ISO/IEC 18004:2000(E). *Information technology - Automatic identification and data capture techniques - Bar code symbology - QR Code*. International Standard. First Edition. ISO/IEC, 2000 (cit. a p. 95).

[38] Microsoft Research. *High Capacity Color Barcodes (HCCB)*. 2010. URL: http://research.microsoft.com/en-us/projects/hccb/ (cit. a p. 99).

[39] Andrea Fusiello. *Visione Computazionale*. 2008 (cit. alle pp. 103, 106, 108, 111).

[40] Wikipedia l'enciclopedia libera. *Coordinate omogenee*. Versione del 1 Settembre 2011. 2011. URL: http://it.wikipedia.org/wiki/Coordinate_omogenee (cit. a p. 104).

[41] Benedetto Alotta, Duccio Fioravanti e Monica Malvezzi. *Note alle lezioni di Modellistica e Controllo di Sistemi Meccanici*. 2007 (cit. a p. 110).

[42] Enrico Cinalli. *Tecniche avanzate di Photo stitching Control points t1/t2 e opzioni Layers di PTgui*. 2006. URL: http://www.photoactivity.com/Pagine/Articoli/017%20Stitch%20approfondimento/Tecniche%20di%20Stitching%20-appr.asp (cit. a p. 115).

[43] Gaetano Scarano. *Lezioni di Elaborazione delle Immagini*. 2006 (cit. alle pp. 123, 124, 132, 133, 140).

[44] ITU International Telecommunication Union, cur. *Information technology - Digital compression and coding of continuous-tone still images: Requirements and guidelines*. ITU-T Rec. T.81 (09/92). ISO/IEC 10918-1:1994. 1992 (cit. alle pp. 124, 146, 147).

[45] Gregory K. Wallace. "The Jpeg Still Picture Compression Standard". In: *IEEE Transaction on Consumer Electronics*. 1ª ser. 38 (1992) (cit. alle pp. 124, 128).

[46] Eric Hamilton. *JPEG File Interchange Format*. version 1.02. 1992 (cit. alle pp. 124, 147, 149).

[47] Kevin Cohen. "Digital Still Camera Forensics". In: *Small Scale Digital Device Forensics Journal* 1 (2007). NO.1 (cit. a p. 124).

[48] Stefania Colonnese. *Dispense di Sistemi Multimediali*. 2011 (cit. alle pp. 124, 129).

[49] ITU International Telecommunication Union, cur. *Studio encoding parameters of digital television for standard 4:3 and wide screen 16:9 aspect ratios*. Recommendation ITU-R BT.601-7. 2011 (cit. alle pp. 126, 141).

[50] Jesse D. Kornblum. "Using JPEG quantization tables to identify imagery processed by software". In: *ScienceDirect* (2008). Digital Investigation (cit. a p. 136).

[51] Camera e Imaging Products Association, cur. *DC-008-2010 - Exchangeable image file format for digital still cameras: Exif Version 2.3*. Prepared by: Standardization Commiteee. Standard of the Camera e Imaging Products Association. 2010 (cit. a p. 147).

[52] Massachusetts Institute of Technology Computer Science e Artificial Intelligence Laboratory. *The Cricket Indoor Location System*. URL: `http://www.cl.cam.ac.uk/research/dtg/research/wiki/BatSystem`.