UNIVERSITÁ DEGLI STUDI DI ROMA "LA SAPIENZA"

Facoltá di Ingegneria

Corso di Laurea in Ingegneria Informatica (LS)

TESI

Un modello di rilevazione e tracciamento di persone per una piattaforma robotica mobile, basato su metodi multimodali.

(Detecciòn y seguimento multimodal de Personas para Plataformas Robòticas Mòviles)

Candidato: Alessio Vaudi **Relatore: Fiora Pirri**

A.A 2008-2009

UNIVERSITÁ DEGLI STUDI DI ROMA "LA SAPIENZA"

Facoltá di Ingegneria
Corso di Laurea in Ingegneria Informatica (LS)

Tesi Titolata: "Un modello di rilevazione e tracciamento di persone per una piattaforma robotica mobile, basato su metodi multimodali.", che presenta Alessio Vaudi, realizzata sotto la direzione dei Professori D. Modesto Castrillòn Santana , D. Antonio Carlos Domínguez Brito e Fiora Pirri

Roma, A.A 2008-2009

Relatore: la prof.ssa

Tutor: il professor

Tutor: il professor

Fiora Pirri

Modesto Castrillòn Santana

Antonio Carlos Domínguez Brito

Il Laureando

Alessio Vaudi

..Alla mia famiglia..

Ringraziamenti

Ringrazio l'università "La sapienza" per questi anni di studio e per avermi concesso la possibilità di svolgere la tesi all'estero, come vincitore della borsa Erasmus. Ringrazio la professoressa Fiora Pirri per la pazienza, la gentilezza e la disponibilità che mi ha concesso. Ringrazio l'università de Las Palmas de Gran Canaria, ed in particolare i professosi Modesto Castrillòn Santana, Antonio Dominguèz Brito e Daniel Hernandez, per l'accoglienza, la simpatia, la comprensione, l'amicizia e l'aiuto che mi hanno dato in questi 7 mesi. Ringrazio la mia famiglia che mi ha sostenuto ed incitato per tutto il corso dei miei studi.

Grazie mille a tutti.

Contents

List of Figures

List of Tables

List of Algorithms

Prefazione

Questo documento descrive un progetto di intelligenza artificiale e robotica, sviluppato presso il laboratorio del "gruppo di Intelligenza Artificiale e Sistemi (SIANI)" della università de Las Palmas de GranCanaria (Spagna).

Tale progetto consiste nel programmare un robot mobile affinché esso sia in grado di riconoscere e seguire persone presenti nell'ambiente circostante. L'individuazione di una persona avviene con opportuni algoritmi che elaborano dati acquisiti dai sensori (laser e sonar) presenti nella piattaforma mobile.

Principalmente, la piattaforma robotica è costituita da una base mobile, da una camera posta in cima di un payload e da un "pan/tilt".

Per la rilevazione di una persona si utilizzano due distinte componenti che lavorano indipendentemente tra loro: una per la face detection ed una per il riconoscimento delle gambe. L'implementazione di un algoritmo che sincronizza queste due componenti ha permesso la realizzazione di un'applicazione che riduce notevolmente gli errori dovuti a falsi positivi

Per quanto riguarda la rilevazione del volto, viene utilizzato l'algoritmo "ENCARA2" sviluppato dal professor Modesto Castrillòn Santana de la Università de Las Palmas de Gran Canaria, mentre per le gambe viene implementato un algoritmo che si basa su segmentazione, classificazione e raggruppamento dei dati acquisiti tramite sensori laser.

Per la ricezione dei pacchetti di dati provenienti dall'ambiente e l'invio di comandi all'hardware si è utilizzato "CoolBot": un framework di programmazione orientato a componenti, progettato per aiutare gli sviluppatori di sistemi robotici ad ottenere sistemi strutturati e riutilizzabili, senza imporre alcuna specifica architettura.

All'interno di questo quadro sono concepite componenti realizzate con automi a stati finiti e comunicanti tra loro mediante "porte" di input e di ouput.

Coolbot nasconde al programmatore tutti gli aspetti legati alla comunicazione, fornendo meccanismi standard per i diversi modi di scambio di dati tra componenti.

Infine viene utilizzato la componente "Player/Stage". Un simulatore che permette il controllo del robot e l'astrazione dalla piattaforma.

Obiettivi del progetto

L'obiettivo del progetto è decomposto in una serie di punti:

1. Individuazione e tracciamento di persone in un ambiente con diverse modalità, sfruttando la robotica mobile.

2. Integrazione di meccanismi di ancoraggio per la combinazione di diverse modalità: principalmente visione e laser con la possibilità di prendere in considerazione l'audio.

3. Sviluppo di una strategia che sincronizza ed integra l'elaborazione dei dati provenienti da più sensori.

4. Integrazione di riunioni (con relativo ordine del giorno) per discutere delle strategie, dei meccanismi di posizione e dei problemi hardware e software riscontrati nel progetto.

Materiali e mezzi

Le risorse e i mezzi avuti a disposizione sono stati i seguenti:

1. Personal Computer con processore Pentium 2Ghz, 512 MB RAM, 40 GB di HardDisk e portatile "hp pavillon dv 1000".
2. Compilatore C/C++.
3. OpenCV.
4. Camera "pan/tilt" standard.
5. Connessione a internet.
6. Piattaforma Robotica mobile Pioneer P3-DX.
7. Sensori Laser.

Pianificazione temporale

Le tappe effettuate che sono state necessarie per lo sviluppo del progetto sono le seguenti:

a) **Analisi Globale**: Analisi globale dell'intorno dell'applicazione. Apprendimento della lingua spagnola per la comunicazione con i tutori, identificazione delle necessità, degli elementi, dei processi e dei procedimenti.

b) **OpenCV**: Conoscenza della libreria OpenCV (funzioni e modalità d'uso).

c) **ENCARA2**: Conoscenza dei distinti detectors di facce. In particolare il funzionamento (in linea teorica) e l'apprendimento delle modalità d'uso di ENCARA2.

d) **Integrazione di Player/Stage e CoolBot**: Conoscenza ed integrazione di sensori propriocettivi ed esterocettivi. Combinazione di player/stage e CoolBot per integrare dispositivi hardware (del robot mobile) e componenti software.

e) **Sviluppo del prototipo**: Sviluppo di un prototipo che combina le distinte modalità per rilevare e seguire una persona.

f) **Esperimenti**: Su uno stesso scenario utilizzando diverse modalità viene effettuata la verifica del funzionamento del prototipo. Registrazione di una demo video.

g) **Documentazione**: Scrittura della tesi.

Una stima temporale dello sviluppo del progetto, in base ai compiti prevesti per ogni tappa è stata la seguente:

Tabella 1: Pianificazione temporale.

Tappe	Tempo stimato
Tappa a: Analisi Globale	20 ore
Tappa b: OpenCV	20 ore
Tappa c: ENCARA2	20 ore
Tappa d: Player/Stage e CoolBot	40 ore
Tappa e: Svluppo prototipo	75 ore
Tappa f: Esperimenti	75 ore
Tappa g: Documentazione	50ore
Tempo totale stimato	300 ore

Come si vede nella tabella 1, ad eccezione della fase di sviluppo è stata rispettata la stima dei tempi pianificata all'inizio del progetto. Infatti per il prototipo il tempo impiegato è stato molto maggiore di quello stimato. Questo ritardo è dovuto principalmente a problemi legati all'hardware del robot (soprattuto alla camera) e ad alternative da implementare per ovviare ad essi.

Capitolo 1

Introduzione

1.1 Computer Vision

La **computer vision** è l'insieme dei processi che mirano a creare un modello approssimato del mondo reale (3D) partendo da immagini bidimensionali (2D). Lo scopo principale della visione artificiale è quello di riprodurre la vista umana. Vedere è inteso non solo come l'acquisizione di una fotografia bidimensionale di una regione, ma soprattutto come l'interpretazione del contenuto di quella regione. L'informazione è intesa in questo caso come qualcosa che implica una decisione automatica. Lo scopo della computer vision è quindi programmare un calcolatore affinché possa "capire" una scena o le caratteristiche di una immagine.

Un problema classico nella visione artificiale è quello di determinare se l'immagine contiene o no determinati oggetti (Object recognition). Il problema può essere risolto efficacemente e senza difficoltà per oggetti specifici in situazioni specifiche, per esempio il riconoscimento di specifici oggetti geometrici come poliedri, riconoscimento di volti o caratteri scritti a mano. Le cose si complicano nel caso di oggetti arbitrari in situazioni arbitrarie.

Nella letteratura si trovano differenti varietà del problema:

* **Recognition**: uno o più oggetti prespecificati o memorizzati possono essere ricondotti a classi generiche insieme alla loro posizione 2D o 3D nella scena.
* **Identification**: viene individuata un'istanza specifica di una classe. Es. Identificazione di un volto, impronta digitale o veicolo specifico.
* **Detection**: L'immagine è scansionata fino all'individuazione di una condizione specifica. Es. Individuazione di possibili facce.
* **Tracking**: Inseguimento di un oggetto in una sequenza di immagini.

Altro compito tipico è la ricostruzione della scena: date due o più immagini 2D si tenta di ricostruire un modello 3D della scena. Nel caso più semplice si parla di un

set di singoli punti 3D. Casi più complessi tentano di generare superfici 3D complete. Generalmente è importante trovare la matrice fondamentale che rappresenta i punti comuni provenienti da immagini differenti.

Una parte significativa dell'intelligenza artificiale si occupa di gestire sistemi interfacciati a robot o macchine che si muovono nello spazio o compiono movimenti. Questa tipologia di processo implica spesso l'acquisizione di informazioni fornite da un sistema di visione artificiale che occupa il ruolo di sensore visivo. Come nel caso del presente lavoro in cui una camera fornisce informazioni alla piattaforma sulla presenza o meno di persone.

1.2 Rilevamento di persona

La capacità dei robot mobili di monitorare i cambiamenti dell'ambiente, includendo persone, è una delle più importanti funzioni di un robot nel mondo reale, al pari di evitare ostacoli, pianificare percorsi, effettuare servizi, controlli di sicurezza e cosi via..

Il tracciamento ed il rilevamento di più persone in ampi spazi aperti ha al giorno d'oggi numerose applicazioni; tra le quali si trovano ad esempio, il monitoraggio delle attività umane per sistemi di sorveglianza intelligente, misurazione e analisi delle traiettorie, monitoraggio nelle mostre e nelle stazioni ferroviarie, ma anche nello studio dei comportamenti sociali ed umani.

Nel campo della "Computer Vision", il tema del rilevamento e del tracciamento automatico delle persone è stato studiato per anni. I maggiori sforzi hanno prodotto risultati abbastanza buoni per tecniche limitate a semplici situazioni dove gli umani appaiono in regioni isolate o dove le aree di occlusione sono piccole. Altre tecniche sono basate sul riconoscimento visivo di testa e faccia, le quali sono "time-consuming" e non molto affidabili soprattutto in real-time e in scenari reali, dove le condizioni di luce non sono controllabili e dove occlusioni temporanee possono verificarsi in qualsiasi momento. Tali tecniche dipendono dalla configurazione dell'ambiente, dal settaggio della camera, dal punto di vista, dall'orientazione dell'angolo, dai pixels e cosi via...

Più efficiente è il rilevamento ed il tracciamento "laser-based", che prevede rilevazione automatica ed affidabile di umani in scenari variabili, essendo i laser insensibili a condizioni di luce e computazionalmente più efficienti per quanto riguarda l'elaborazione dei dati (infatti consumano meno tempo delle camere). Tali metodi sono preferiti inoltre a metodi "sonar-based", che utilizzano il suono per capire dove è collocata una persona, in quanto nel suono è possibile la presenza del rumore ad ostacolare la rilevazione.

I vantaggi dei laser rispetto alle video camere, sono i seguenti:

- in primo luogo, si tratta di un tipo di misurazione diretta e l'estrazione di oggetti in movimento, nel sistema di coordinate del mondo reale, non è tale da consumare molto tempo come quello di usare una normale video camera;

- in secondo luogo, il sistema di riferimento del laser può essere facilmente con-

vertito in un sistema di coordinate rettangolari su di un piano orizzontale, ed è relativamente semplice coprire vaste zone integrando dati provenienti da sensori distruibuiti nell'ambiente;

- il tracciamento di gruppi di persone può essere raggiunto in real-time grazie al basso costo computazionale;

- infine, sebbene non abbia una grande iterpretabilità del range delle misure rilevate, come può avere un video, evita il problema della privacy, come ad esempio, nei supermercati, nelle sale mostra ecc.

Tuttavia, le limitazioni dei metodi "laser-based" sono ovvie:

- non possono prevedere informazioni sul colore di oggetti ed è difficile distinguere attraverso delle caratteristiche, un oggetto da un altro;

- se una traiettoria è spezzata a causa di un occlusione è difficile connettere frammenti insieme;

- se le persone camminano in gruppi o vicine tra loro le traiettorie dei piedi saranno perse.

In conclusione la computazione visiva è fortemente accoppiata con i metodi "laser-based". Infatti, integrando i due approcci è possibile complementare vantaggi e svantaggi dell'uno e dell'altro in modo tale da rendere affidabile il rilevamento di persone in tempo reale. Vengono quindi combinate informazioni relative alla distanza, provenienti dalle scansioni laser, ed informazioni visive proveniente dalla camera, (con la possiblità eventualmente di considerare la voce).

L'individuazione di facce [Yang02, Hjelmas01a, Castrillòn 07] è un processo necessario per qualsiasi sistema di riconoscimento facciale [Chellapa95], o di analisi di espressione facciale [Ekman73, Russell97].

In situazioni di minor risoluzione o con maggior libertà di movimenti dove il volto non è sempre visibile, si è arrivati al punto di rilevare individui in base alle gambe [Papageorgiou98], al corpo completo [Viola03, Dalal05] o alcuni elementi corporali [Castrillòn05], [Mikolajczyk04].

Nel contesto di piattaforme mobili, con l'esistenza dell'ergomovimento, alcune delle soluzioni perdono di efficacia, dato che è più frequente perdere di vista l'obiettivo. Tuttavia, come già accennato le piattaforme in movimento oltre alle camere, solitamente dispongono di altri sensori con cui è possibile realizzare il compito di individuazione e inseguimento di una persona in forma multimodale.

Un dispositivo integrato di frequente nelle piattaformi mobili è il laser, che è stato precedentemente utilizzato per individuare persone [Pan04, Cui08].

L'utilizzazione di diverse modalità richiede meccanismi che permettono di stabilire la corrispondenza tra esse al fine di raggiungere lo stesso obiettivo, in modo tale che quando una delle modalità perde l'obiettivo, si può seguire lo stesso facendo uso

delle modalità restanti. Questo meccanismo denominato ancoraggio (anchoring) è stato utilizzato in sistemi robotici [Wrede06] basati su piattaforme Pioneer simili a quelle disponibili nel laboratorio del "Grupo de Inteligencia Artificial y Sistemas".

È per questo che si pone in questo progetto la combinazione di distinti strumenti:

1. Analizzare le possibilità che offrono nel contesto della robotica mobile i detectors attuali di persone come "ENCARA2";
2. l'utilizzazione, in detto contesto, di altre modalità come il dispositivo laser,
3. l'utilizzazione di camera e laser congiunti in modo da poter rendere più affidabile il rilevamento di una persona.

1.2.1 Aspetti innovativi

La novità, (o per lo meno una possibile alternativa alle soluzioni presenti in letteratura), introdotta nel progetto è la possibilità di rilevare e seguire una persona utilizzando diverse modalità. Si è cercato di creare una opportuna strategia che mettesse in relazione l'elaborazione dei dati acquisiti contemporaneamente dai sensori del robot e l'integrazione di diverse componenti che trattano algoritmi con differenti proprietà spazio-temporali

L'originalità della strategia adottata consiste nel rilevare dapprima le gambe di una persona e successivamente confermare la presenza della stessa utilizzando la camera, riducendo cosí di molto la probabilità di errore in fase di detection (come si dimostra nella sezione esperimenti nel cap. 6). Il sistema di visione utilizza un "pan/tilt" che direziona la camera, sia orizzontalmente, nella posizione in cui sono rilevate le gambe e sia verticalmente, a seconda della distanza del rilevamento.

L'utilizzo di questa modalità di funzionamento in fase di tracking permette un notevole risparmio di energia, in quanto si riducono al minimo i movimenti del robot a fronte di falsi positivi. Infatti il robot si sposta in una determinata direzione solo quando è sicuro di aver individuato una persona.

Inoltre si e' aggiunto un comportamento di tipo"Master-Slave"che consente tra più persone la possiblità di sceglierne una e di eleggerla come "Master". Di conseguenza il robot si comporterà come "Slave"seguendo questa persona finchè verrà detectata (riferire al cap. 5 per maggiori dettagli).

Nei successivi capitoli verranno descritti i principali sistemi di face detection(cap. 2), il detector ENCARA utilizzato in questo progetto, gli algoritmi implementati (cap. 4), la strategia sopra citata (cap. 5), gli esperimenti effettuati (cap .6), ed infine possibili miglioramenti che possono essere adottati in futuro (cap. 7).

Capitolo 2

Preliminari: metodi di identificazione

La Face detection è una tecnologia per la computer vision che determina le posizioni e le grandezze di facce umane in immagini digitali arbitrarie. Un detector individua caratteristiche facciali ed ignora tutto il resto, come edifici, alberi e corpi.

La Face detection può essere considerata come un caso particolare di object-class detection, dove il compito è trovare le posizioni e le grandezze di tutti gli oggetti che appartengono ad una certa classe in un' immagine. Esempi includono volti, pedoni e automobili.

I primi algoritmi si focalizzavano sul rilevamento di volti frontali, mentre quelli più recenti cercano di risolvere problemi più generali e difficili di multi-view face detection, ovvero l'individuazione dei volti che sono ruotati lungo l'asse dalla faccia all'osservatore (rotazione in piano), ruotati lungo la verticale (out-of-plane rotation) o entrambi. Molti algoritmi implementano il compito di face-detection come un compito di binary pattern-classification. Il contenuto di date parti di immagini sono trasformate in caratteristiche, dopo che un classificatore allenato su esempi di facce decide se la particolare regione dell'immagine è una faccia oppure no.

Spesso è impiegata una tecnica a finestra scorrevole, dove esiste un classificatore usato per classificare le porzioni di una immagine sia come facce che come non facce, a tutte le locazioni e a tutte le scale.

La Face detection trova applicazione in biometria, spesso come una parte di un sistema di riconoscimento facciale, in materia di video sorveglianza (contesto locale), nell'interfaccia uomo-macchina e nella gestione di database di immagini. Alcune recenti fotocamere digitali usano il rilevamento dei volti per messa a fuoco automatica. Inoltre, il rilevamento di volti è utile per selezionare le regioni di interesse in slideshow di foto che utilizzano un effetto Ken Burns.

I differenti metodi di rilevamento di facce possono essere classificati in due differenti famiglie basate sulla conoscenza:

- **Conoscenca implicita**: Cerca somiglianze tra i patterns appresi considerando anche la posizione e le scale di risoluzione utilizzando Feature "Haar-Like" [Viola e Jones].

- **Conoscenza esplicita**: Considera combinazioni di colori, movimenti e geometria del volto.

la conoscenza esplicita fà un estratto di caratteristiche di colori e apparenza (descrittori), mentre la conoscenza implicita combina classificatori complessi in cascata. Nel progetto Encara si integrano i due approcci.

In letteratura si ritrovano vari algoritmi che implementano il riconoscimento facciale, tra i più popolari consideriamo, il lavoro svolto da "Viola e Jones", da "Schneiderman e Kanade" e da "Modesto Castrillòn" che considera l'implementazione di "Encara" basata sulle tesi dei precedenti autori. Nel seguito si darà una descrizione generale dei suddetti lavori.

2.1 Metodo Viola e Jones

Il robusto rilevatore di facce proposto da "Viola e Jones" è composto da tre fasi fondamentali:

a) features selection;

b) face learning;

c) face detection.

I primi due passaggi permettono di far apprendere all'algoritmo la forma e le caratteristiche comuni di un volto umano. Le caratteristiche infatti distinguono parti diverse dell'immagine e permettono di rilevare parti ricorrenti in figure umane, come naso, occhi e bocca. Migliori saranno le features e migliore sarà la detection. L'algoritmo di "training" è basato su **AdaBoost**. AdaBoost è un algoritmo che permette la selezione delle migliori caratteristiche da un set random di classificatori. Una cascata di chiamate ad AdaBoost rende robusta la rilevazione.

Una volta effettuato l'addestramento si utilizza un face detector che scannerizza l'immagine passata in input, utilizzando un meccanismo a finestra scorrevole. La finestra selezionerà ed analizzerà una piccola regione dell'immagine, al fine di rilevare facce. Tale finestra varia in dimensione e posizione in modo tale da analizzare ogni regione dell'immagine e riconoscere facce a diverse scale e posizioni. La detection dipende fortemente dall'addestramento di AdaBoost.

2.1.1 Descrizione metodo

Nella realizzazione del riconoscitore di volti proposto da "Viola e Jones" tre sono le componenti che rendono questo riconoscitore uno dei più efficienti:

1. L'utilizzo di Immagini Integrali.
2. La selezione delle caratteristiche importanti attraverso l'algoritmo "Adaboost".
3. Cascata di classificatori.

Tale riconoscitore utilizza l'insieme di Feature "HaarLike" (un insieme derivato da trasformate di Haar), per analizzare le caratteristiche delle immagini. L'utilizzo di tali feature permette di definire un classificatore come un albero decisionale con almeno due foglie, per determinare se una zona di una immagine di input corrisponda a qualche immagine da lui conosciuta. Inizialmente, il classificatore viene "addestrato" con un numero opportuno (via via crescente) di immagini di un oggetto particolare (nel nostro caso facce) denominate "esempi positivi" (tutti ridotti allo stesso formato 24x24), ed "esempi negativi" (immagini arbitrarie e quindi NON facce, sempre dello stesso formato).

La caratteristica usata in un classificatore è definita dalla sua forma, presentato in seguito, dalla posizione all'interno della regione di interesse e dalla scala.

Features:

Il metodo "Viola e Jones" si basa sulla nozione di feature (un sistema basato su features opera molto più veloce di uno basato su pixels); una feature corrisponde ad un set di due o più rettangoli adiacenti. Come si vede nella figura 2.1 per classificare una immagine vengono utilizzate differenti tipi di features:

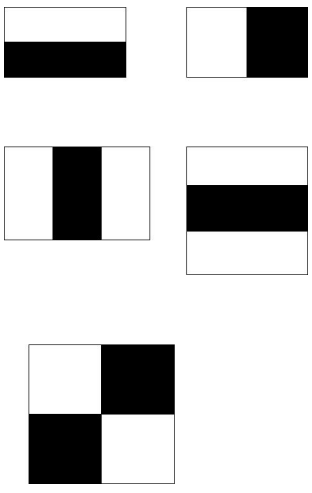

Figura 2.1: La Figura rappresenta i quattro tipi di features Haar Like.

Nella figura 2.1 sono rappresentate rispettivamente le seguenti features:

- **Edge Feature**: feature con 2 rettangoli (direct, 90 gradi),
- **Line Feature**: feature con 3 rettangoli (direct, 90 gradi),
- **four rectangle Feature**: feature con 4 rettangoli

Il valore di una feature è calcolata sottraendo la somma dei pixels nel rettangolo bianco, dalla somma dei pixels dentro quello nero.

Immagini Integrali:

In una immagine, calcolare ripetutamente somme di pixel compresi fra regioni rettangolari potrebbe risultare molto costoso. Per far si che sia possibile calcolarle molto velocemente sono state introdotte le immagini integrali. Queste immagini di supporto sono calcolate molto semplicemente dalle immagini di input attraverso somme cumulative dei pixel: ogni pixel dell'immagine integrale ii(x', y') corrisponde alla somma dei pixel i(x, y) con x minore di x' e y minore di y':

$$ii(x', y') = \sum_{x'<x, y'<y} i(x, y)$$

Attraverso le seguenti formule è possibile calcolare l'immagine integrale in un solo passo computazionale:

$$s(x, y) = s(x, y - 1) + i(x, y)$$
$$ii(x, y) = ii(x - 1, y) + s(x, y)$$

dove s(x, y) è la somma cumulativa della riga. Alla luce di queste formule immediate e semplici, si consideri l'esempio rappresentato in figura 2.2:

La somma dei pixel del punto D può essere calcolata con semplici operazioni sui valori dei punti (1, 2, 3, 4). Il valore del punto 1 corrisponde alla somma cumulativa dei pixel dell'area A; B corrisponde al valore del punto 2 sottratto a quello del punto 1; C è il valore del punto 3 dopo aver sottratto quello del punto 1. Nello specifico i valori sono i seguenti:

- A = 1.
- B = 2 - 1.
- C = 3 - 1.
- D = 4 - A - B - C. $\Rightarrow$ D = 4 - 1 - (2 - 1) - (3 - 1).

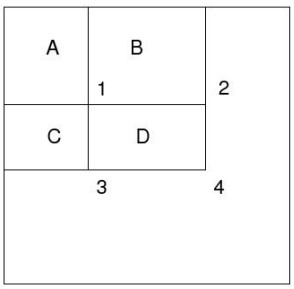

Figura 2.2: La figura rappresenta un esempio di calcolo di una immagine integrale.

AdaBoost

Per costruire un classificatore (indicato con "h") si possono utilizzare diversi approcci. Nel riconoscitore che si sta analizzando, l'algoritmo Adaboost viene utilizzato sia per analizzare l'immagine in input, sia per addestrare i classificatori base. Nella sua forma originale, l'algoritmo in questione viene usato per accelerare le performance di classificazione di un algoritmo di apprendimento. In una immagine 24x24 vi sono più di 160.000 caratteristiche Haar like da poter prendere in considerazione. Nel caso considerassimo un'immagine più grande il numero di caratteristiche andrebbe moltiplicato per tutte le sottofinestre di 24x24 pixel che sono presenti nell'immagine. È evidente il costo proibitivo che questa procedura richiederebbe e quindi per calcolarle tutte in maniera rapida si utilizzano le immagini integrali.

L'algoritmo adaboost si occupa di scegliere la caratteristica più importante fra tutte. Ad ogni ciclo Adaboost analizza tutte le caratteristiche tra le più di 160.000 (presenti nell'immagine 24x24 positive e negative in tonalità di grigio) e seleziona quella che compare più volte, ovvero la caratteristica più frequente, con il miglior "epsilon" e che minimizza la differenza, e quindi l'errore, tra il vettore delle "labels"(apprendimento supervisionato, accennato in seguito) ed il vettore 'e', il cui i-esimo elemento rappresenta la classificazione dell'immagine di input (e=1 se l'immagine x_i e'una faccia, 0 altrimenti).

L'algoritmo adaBoost è costituto da un ciclo principale in cui l'unico parametro da apprendere è "T" rappresentante il numero di iterazioni e quindi il numero di "weak classifier" che andranno a costituire lo strong classifier. L'algoritmo prende in input le immagini da analizzare ed un vettore di valori binari (labels) che indicano se l'immagine considerata è una faccia oppure no. Questi esempi di immagini positive e negative vengono prelevate da un file e standardizzate nel formato 24 x 24, cosí da poter calcolare l'immagine integrale. Per determinare lo StrongClassifier è necessario definire e normalizzare dei pesi (weights) che si calcolano a partire dal numero di esempi positivi e negativi a disposizione: essi infatti permettono di calcolare il miglior valore di soglia (treshold) che a sua volta ci consente di determinare la feature che porterebbe a minimizzare l'errore.

L'algoritmo 1 rappresenta lo pseudocodice di AdaBoost.

Collect example images

$(x_1, y_1)...(x_n, y_n)$

where $y_1 = 0, 1$ for negative and positive examples rispectively

Initialize weights:

$$w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$$

for $y_i = 0, 1$ respectively,where m is the number of negatives and l is the number of positives examples

for $t = 1$ to T **do**

Normalize the weights:

$$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{N} w_{t,j}}$$

Select the best weak classifier with respect to the weighted error:

$$\epsilon_t = min_{f,p,\theta} \sum_i w_i \left| h(x_i, f, p, \theta) - y_i \right|$$

Define h_t(x):

$$h(x_i, f, p, \theta) = \begin{cases} 1 & \text{if } pf(x) < p\theta \\ 0 & \text{otherwise} \end{cases}$$

Update the weights:

$$w_{t+1,i} = w_{t,i}\, \beta^{1-e_i}$$

where:$e_i = 0, 1$ classified correctly and incorrectly rispectively and $\beta = \dfrac{\epsilon_t}{1 - \epsilon_t}$

Return the strong classifier: $C(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$

where $\alpha_t = log \dfrac{1}{\beta_t}$

end for

Algorithm 1: Algoritmo AdaBoost

Cascata di classificatori:

Ad ogni passo di "boosting" viene creato un classificatore base (weak). Per riconoscere la presenza di un oggetto conosciuto in un'immagine arbitraria bisognerebbe raccogliere tutti questi classificatori base e sommarli per crearne uno che sia valido per il riconoscimento effettivo dell'oggetto, utilizzando questo classificatore generale ("strongClassify") per verificarne la presenza in tutte le possibili sotto-finestre 24x24 dell'immagine da analizzare. Procedendo in questo modo, il costo computazionale diviene visibilmente proibitivo. L'idea di Viola e Jones è quella di utilizzare una struttura a cascata invece del classificatore generale. Come si vede nella figura 2.3, una serie di classificatori vengono applicati a tutte le sotto-finestre presenti nell'immagine.

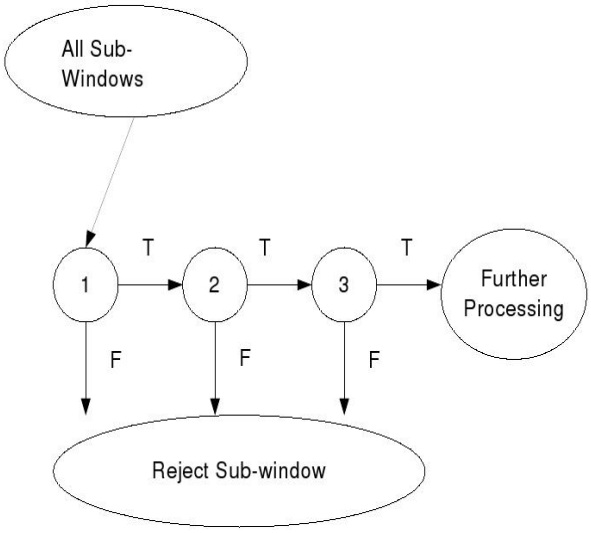

Figura 2.3: La figura rappresenta la cascata di classificatori.

Per ognuna di queste, un risultato positivo dal primo classificatore conduce all'analisi di un secondo classificatore, che in caso di risultato positivo conduce all'analisi di un terzo classificatore e cosí via, eliminando la sotto-finestra corrente alla prima occorrenza di un risultato negativo. In questo modo le risorse verranno consumate solo dalle aree che potrebbero condurre ad una individuazione dell'oggetto cercato, riducendo drasticamente il numero di ricerche nell'immagine.

La cascata è costituita da classificatori sempre più complessi:

- i classificatori semplici e veloci scartano la grande maggioranza delle regioni di immagine; quindi sono usati per scartare immagini "facili " da classificare in modo da escludere più sotto-finestre negative il più presto possibile (nei primi livelli).

- I classificatori più complessi vengono usati solo dove serve abbattere il numero di falsi positivi. Classificatori successivi sono addestrati usando quegli esempi che sono riusciti ad attraversare tutti i livelli precedenti (in quanto facce o falso positivo).

La cascata viene realizzata posizionando nelle prime posizioni i classificatori più semplici, in modo da eliminare rapidamente buona parte delle finestre negative; in un secondo tempo entrano in gioco i classificatori più complessi in grado di scartare i rimanenti falsi positivi. Ogni livello della cascata è costruito addestrando un classificatore attraverso AdaBoost. Per ottenere un detection rate accettabile si modifica adaBoost modificando la soglia del classificatore.

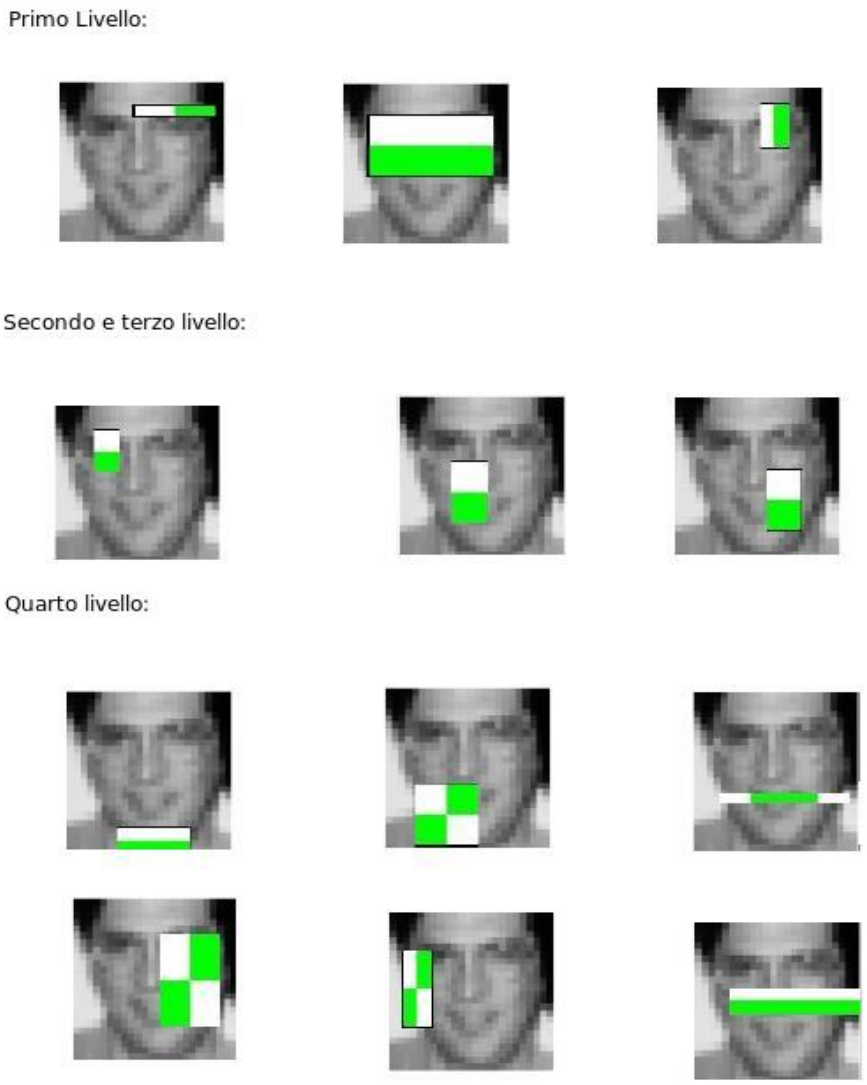

Figura 2.4: La Figura rappresenta le features ottenute dopo 4 livelli della cascata.

La figura 2.4, rappresenta le feature ottenute implementando tale metodo. In particolare queste feature sono state ricavate da una cascata a 4 livelli. Come si vede dalla figura, ogni diversa feature cattura una caratteristica differente del viso.

Detector

Una volta ottenuto il classificatore a cascata, si implementa un detector che scala la grandezza delle features per trovare facce di dimensioni variabili.

2.2 Metodo statistico per il rilevamento di oggetti 3D (Schneiderman e Kanade)

Questo metodo si avvale dell'uso di istogrammi per la rappresentazione statistica di oggetti e di "non oggetti" che appaiono nell'immagine. Ciascun istogramma rappresenta le statistiche in comune di un sottoinsieme di coefficienti di wavelet e la loro posizione in merito all'oggetto. L'approccio si basa sull'uso di un certo numero di istogrammi in modo tale da avere un'ampia varietà di attributi visivi. Usando questo metodo, i due autori, hanno sviluppato un algoritmo in grado di rilevare in modo affidabile i volti umani (considerando rotazioni out-of-plane) e un algoritmo in grado di rilevare le autovetture (considerando una vasta gamma di punti di vista).

I principali problemi da affrontare nel campo della rilevazione di oggetti sono le variazioni degli aspetti visivi. Per esempio, le autovetture variano nella forma, nella dimensione, nella colorazione, e nei piccoli dettagli come fari, griglie e ruote. L'aspetto visivo dipende anche dall'ambiente circostante. Le fonti di luce possono variare l'intensità, il colore e la posizione dell'oggetto, mentre gli oggetti nelle vicinanze possono essere fonti di ombre o riflessi. La comparsa di un oggetto dipende inoltre dalla sua posa (rappresentata dalla sua posizione e dall'orientamento rispetto alla fotocamera): ad esempio, una vista laterale di un volto umano avrà un aspetto molto diverso rispetto una vista frontale.

Per far fronte a tutte queste variazioni si usa una strategia divisa in due parti. Dapprima, per ovviare alle variazioni di posa, si utilizza un approccio "view-based" [K. Sung, T. Poggio] con rilevatori multipli, (ognuno dei quali mantiene una specifica orientazione dell'oggetto); successivamente si considera una modellazione statistica all'interno di ognuno di essi per tenere conto delle variazioni rimanenti.

2.2.1 View-Based Detectors

Si sviluppano rilevatori separati: ad esempio per le facce, uno specializzato nella vista del profilo destro ed uno specializzato nella vista frontale. Tali rilevatori si applicano in parallelo e successivamente si integrano i risultati. Se ci sono multi rilevazioni nella stessa locazione o in locazioni adiacenti viene scelta la detection più forte. Per rilevare profili sinistri si applica il detector dei profili destri con immagini in input invertite. Per le macchine si usano otto rilevatori (come mostrato in figura 2.5).

Ciascuno di questi detectors non solo è specializzato nell'orientazione, ma è anche addestrato per trovare l'oggetto solamente in una dimensione specificata all'interno di una finestra rettangolare dell'immagine. Pertanto, per essere in grado di individuare l'oggetto in una qualsiasi posizione all'interno di un'immagine, si ri-applicano i detectors per tutte le posizioni possibili di questa finestra rettangolare. Mentre, per essere in grado di individuare un oggetto di qualsiasi dimensione, iterativamente si ridimensiona l'immagine in ingresso e si riapplicano i rilevatori allo stesso modo per ogni immagine ridimensionata.

Figura 2.5: La Figura rappresenta esempi di immagini di training per ogni orientazione della macchina.

Per ciascun rilevatore vengono usati modelli statistici che considerano le restanti forme di variazione:

- Il modello statistico per l'oggetto P(image | object);
- il modello statistico per il resto dell'immagine, ossia i "non oggetti" P(image | non-object)

Si utilizza la seguente soglia di verosimiglianza per la classificazione:

$$\frac{P(image \mid object)}{P(image \mid non-object)} > \lambda$$

dove $\lambda = \dfrac{P(non-object)}{P(image)}$

Se tale disuguaglianza è soddisfatta, significa che l'oggetto è presente.

La difficoltà nel modellare le due distribuzioni è che non si conoscono esattamente le caratteristiche statistiche. Non si può sapere se la distribuzione è Gaussiana, Poisson o multimodale, in quanto risulta difficile analizzare caratteristiche comuni di un largo numero di pixels.

Non conoscendo la struttura delle distribuzioni, l'approccio più veloce consisterebbe nello scegliere modelli flessibili. Una classe di modelli flessibili non parametrici e basati sulla memoria, possono essere quelli che utilizzano il metodo di "Parzen" e quelli "nearest neighbor". Lo svantaggio di tali modelli è che per calcolare la probabilità di un dato input, si deve comparare tale input a tutti i dati di "training". Tale computazione risulta essere molto costosa. In alternativa si potrebbero utilizzare modelli parametrici che rappresentano distribuzioni multimodali, come ad esempio, misture di gaussiane. Tutti i metodi però, sono suscettibili a minimi locali, ed è per questo che nel presente lavoro si preferisce utilizzare istogrammi.

La stima dell'istogramma coinvolge il conteggio di quanto spesso un valore di un attributo occorre negli esempi di addestramento. La stima risultante è statisticamente ottima. Essi sono imparziali, coerenti, e soddisfano il limite inferiore di Cramer-Rao.

Lo svantaggio principale di un istogramma è che si può utilizzare solo un piccolo range di valori discreti per descrivere l'aspetto. Per superare questa limitazione, si usano istogrammi multipli, dove ogni istogramma $P_k(pattern_k \mid object)$, rappresenta la probabilità di comparsa su alcuni specificati attributi visivi: "$pattern_k$" è una variabile casuale che descrive alcune caratteristiche visive scelte, come ad esempio il contenuto a bassa frequenza.

Per combinare le probabilità di attributi diversi, si utilizza il seguente prodotto di istogrammi, in cui si approssima ciascuna classe a una funzione di probabilità condizionata:

$$P(image \mid object) \approx \prod_k p_k(pattern_k \mid object)$$

$$P(image \mid non - object) \approx 1 - P(image \mid object)$$

Nel formare queste rappresentazioni per $P(immagine \mid object)$ e $P(Immagine \mid non - object)$, si assume implicitamente che gli attributi ($pattern_k$) siano statisticamente indipendenti (sia per l'oggetto e sia per il non-oggetto).

2.2.2 Decomposizione dell'aspetto in spazio, frequenza ed orientazione

Si decompone l'aspetto dell'oggetto in "Parti", per cui ogni attributo visivo descrive una regione spazialmente localizzata per l'oggetto. Cosí facendo si concentra il potere di modellazione limitato di ogni istogramma su una quantità minore di informazioni visive. Si vorrebbe che queste parti siano adatte alle dimensioni delle caratteristiche di ciascun oggetto. Tuttavia, dal momento che le caratteristiche di facce e macchine occorrono a diverse grandezze, si necessitano attributi multipli su un range di scale diverse. Si definiscono tali attributi effettuando una decomposizione sia nello spazio che nella frequenza, dal momento che basse frequenze esistono su larghe regioni, mentre alte frequenze esistono su piccole regioni. Si definiscono attributi con larga estensione spaziale per descrivere basse frequenze e attributi con piccola estensione spaziale per descrivere alte frequenze.

Gli attributi che coprono piccole estensioni spaziali, catturano regioni ad alta risoluzione come occhi, naso e bocca per le facce; fari e griglie per le auto. Gli attributi che coprono regioni più grandi, catturano caratteristiche a bassa risoluzione come la fronte per le facce ed il cofano per le macchine.

Per quanto riguarda la decomposizione degli attributi per l'orientamento, si considerano attributi che danno maggiore importanza a caratteristiche orizzantali ed altri a caratteristiche verticali.

2.2.3 Rappresentazione di attributi visivi

Per creare attributi visivi, localizzati nello spazio, nella frequenza e nell'orientazione, si necessita di trasformare l'immagine in una rappresentazione comune che lega queste dimensioni. Per fare questo si esegue la trasformata wavelet sull'immagine. La trasformata di wavelet organizza l'immagine in sottobande, che sono localizzate in orientazione e frequenza. Dentro ogni sottobanda ogni coefficiente è spazialmente localizzato. Viene usata una trasformata basata su una decomposizione a 3 livelli usando un *5/3 linear phase filter-bank* producendo 10 sottobande, come mostrato in figura 2.6. Ogni livello della trasformazione rappresenta una ottava più alta di frequenze.

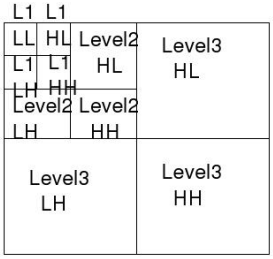

Figura 2.6: Rappresentazione wavelet di una immagine.

Un coefficiente di livello 1 descrive 4 volte la regione di un coefficiente di livello 2, che a sua volta descrive 4 volte la regione di un coefficiente di livello 3. In termini di orientamento, **LH** denota filtraggio passa-basso in direzione orizzontale e filtraggio passa-alto in direzione verticale, cioè rappresenta le caratteristiche orizzontali. **HL** rappresenta caratteristiche verticali.

Questa rappresentazione viene usata come base per specificare attributi visivi. Ad esempio, un attributo potrebbe essere definito per rappresentare una finestra 3x3 di coefficienti nel livello 3. Questo attributo potrebbe catturare patterns orizzontali ad alta frequenza su una piccola regione dell'immagine.

Dal momento che ogni attributo deve assumere solo un numero finito di valori, si dovrebbe calcolare una quantizzazione vettoriale dei suoi coefficienti wavelet pesati. Per lasciare la dimensione degli istogrammi sotto una certa soglia si utilizzano 8 coefficienti per ogni attributo visivo. Nel complesso, si usano 17 attributi che la wavelet trasforma in gruppi di 8 coefficienti in uno dei seguenti modi:

1. **Intra-sottobanda**: Tutti i coefficienti provengono dalla stessa sottobanda. Questi attributi visivi sono i più localizzati in frequenza e in orientamento. Si definiscono 7 di questi attributi per le seguenti sottobande: LL livello 1, LH livello 1, livello 1 HL, livello 2 LH, HL livello 2, livello 3 LH, livello 3 HL.

2. **Inter-frequenza**: Provengono dalla stessa orientazione, ma considerando più bande di frequenza. Questi attributi rappresentano segnali visivi localizzati in un certo range di frequenze. Si definiscono 6 attributi che utilizzano le seguenti

coppie di sottobanda: livello 1 LL - l livello HL, livello 1 LL - l livello di LH, LH livello 1 - livello 2 LH, HL livello 1 - livello 2 HL, livello 2 LH - livello 3 LH, livello 2 HL - livello 3 HL.

3. **Inter-orientamento**: I coefficienti provengono dalla stessa banda di frequenza, considerando però, multiple bande di orientamento. Questi attributi possono rappresentare stimoli che hanno componenti sia orizzontali che verticali, come ad esempio gli angoli. Si definiscono 3 attributi utilizzando le seguenti coppie di sottobanda: livello 1 LH - HL livello 1, livello 2 LH - HL livello 2, livello 3 LH - livello 3 HL.

4. **Inter-frequenza / inter-orientamento**: Questa combinazione è progettata per rappresentare indizi che considerano un arco di frequenze e di orientamenti. Si definisce uno di questi attributi combinando le seguenti sottobande: livello 1 LL, l livello di LH, HL livello 1, livello 2 LH, livello 2 HL.

In termini di decomposizione spazio-temporali, attributi che usano i coefficienti di livello 1, descrivono larghe estensioni spaziali su di un piccolo range di basse frequenze. Gli attributi che usano i coefficenti di livello 2 descrivono una estensione spaziale di media grandezza su un range di frequenze medie. Gli attributi di livello 3 descrivono piccole aree su un largo range di alte frequenze.

2.2.4 Forma finale del detector

La forma finale del detector è data da :

$$\frac{\prod_{x,y \in region} \prod_{k=1}^{17} p_k(pattern_k(x,y), x, y \mid object)}{\prod_{x,y \in region} \prod_{k=1}^{17} p_k(pattern_k(x,y), x, y \mid non-object)} > \lambda$$

dove “region” è la immagine che si sta classificando.

Per il detector di facce, si minimizza l’errore di classificazione sul training set, utilizzando l’algoritmo **AdaBoost** (riferire all’algoritmo 1 ed alla sezione precedente). AdaBoost è un algoritmo di boosting. Il boosting è un meta-algoritmo di ”machine learning“ utilizzato nell’apprendimento supervisionato: ossia una tecnica di deduzione di una funzione a partire da esempi di training. Tali esempi sono rappresentati da coppie di oggetti di input (ad esempio vettori di immagini) e output desiderato (ad esempio un vettore in cui l’i-esimo elemento vale ”1“ se l’input i-esimo rappresenta un volto, ”0“ altrimenti).

Il boosting si basa sull’idea che un set di weak learners (classificatore debole) creano un singolo strong learner (classificatore forte).

Per accelerare il processo di ricerca di oggetti in una immagine si utilizza la seguente euristica. Si valuta dapprima il rapporto della soglia di verosomiglianza per

ogni possibile posizione dell'oggetto usando attributi a bassa risoluzione visiva (per esempio quelli che usano i coefficienti a livello 1) e successivamente si effettua la valutazione ad alta risoluzione per gli oggetti candidati, ossia al di sopra di una soglia minima per la valutazione parziale.

2.3 Cenni al contesto locale

I normali face detector richiedono un gran numero di dettagli facciali per riconoscere un volto. Nei sistemi di video sorveglianza o in sistemi in cui la risoluzione delle immagini può essere bassa è necessario utilizzare face detector robusti e veloci basati sul contesto locale. Si considera quindi, una regione che circoscrive il volto e che comprenda testa, collo e spalle.

Una derivazione formale del contesto locale può essere ottenuta applicando la regola di Bayes e decomponendo la probabilità della presenza dell'oggetto O come segue:

$$P(O \mid v) = \frac{P(O, v)}{P(v)} = \frac{P(vL \mid O, v_C)}{P(v_L \mid v_C)} P(O, v_C)$$

Dove le misurazioni delle immagini v sono in questo caso locali, $v = v_L$, e dov v_C rappresenta la misurazione del contesto dell'oggetto O. Per catturare dipendenze tra un oggetto ed il suo contesto il vettore delle misurazione viene esteso con l'inclusione di features che sono fuori dell'oggetto individuato $(v = v_L, v_C)$. Tutte le probabilità nell'espressione sono condizionate all'informazione contestuale dell'oggetto.

Per implementare il metodo del contesto locale viene addestrato un detector con istanze che contengono gli elementi sopra citati, a differenza dei normali face detector addestrati con immagini contenenti solo volti. Durante la detection la posizione della faccia viene inferita assumendo una posizione fissa dentro la finestra di rilevazione: la dimensione e la posizione della faccia sono direttamente calcolate dalla larghezza e dall'altezza del rettangolo che racchiude il contesto locale detectato.

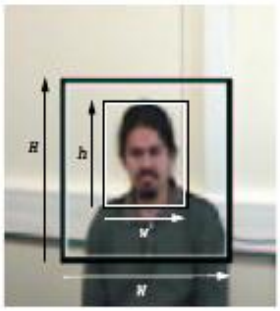

Figura 2.7: Rappresentazione del contesto.

Con riferimento alla figura 2.7, si calcola $w = \frac{W}{2}$, $h = \frac{H}{2}$ e l'angolo in alto a sinistra è settato a $(W/4, H/10)$

Il detector viene sviluppato considerando una versione modificata del metodo "Viola e Jones", descritto nella sezione 2.1. Infatti esso si basa sull'utilizzo di una cascata di classificatori che apprendono mediante differenti tecniche di boosting, e l'utilizzo di un set di feature che estende il set originale considerato nel metodo di cui sopra.

Come si vede nelle figure 2.10, 2.8 e 2.9, sono state aggiunte le seguenti features rispetto al set originale: "rotated features", "center-sorround features" e feature diagonali sussunte da quelle rotate. Il set esteso rende il detector più accurato aumentando di fatto la versatilità e l'espressività.

- **Edge features**:

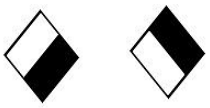

Figura 2.8: La Figura rappresenta il set di features estese: Features Diagonali.

- **Line features**:

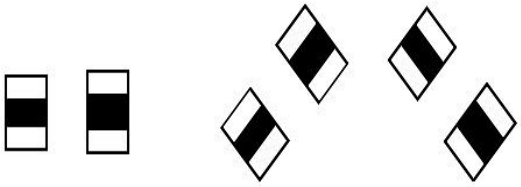

Figura 2.9: La Figura rappresenta il set di features estese: Features Rotated.

- **Center-surround features.**:

Figura 2.10: La Figura rappresenta il set di feature estese: Center-surraund.

Nella figura 2.11 sono mostrate differenti istanze di training consistenti in immagini utilizzate nell'addestramento per la face detection (riga in alto) ed immagini utilizzate nel context local detection (riga in basso). Come si può vedere in tale figura, per quanto riguarda la face detection, le features ottenute dall'apprendimento catturano caratteristiche facciali, mentre features ottenute nel secondo caso catturano i contorni della testa, del corpo e delle spalle.

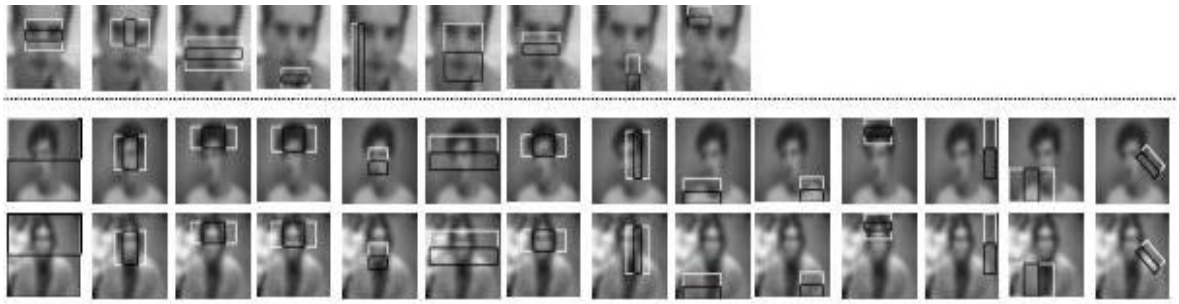

Figura 2.11: Rappresentazione Di feature su istanze object-centered e local context.

Learning

L'implementazione considera un algoritmo di apprendimento a cascata basato sul "decision tree learning", in quanto una cascata di classificatori può essere vista come un albero di decisione. Ad ogni stadio della cascata un classificatore è addestrato ad effettuare il filtraggio di oggetti e non oggetti (scartando quest'ultimi). Il risultante detection rate e false positive rate della cascata sono i seguenti:

- **False positive rate**: $FP = \prod_{i=1}^{K} f_i$.
- **Detection rate**: $D = \prod_{i=1}^{K} d_i$.

Ogni round di boosting, come spiegato nel metodo di sezione 2. 1, seleziona un "weak learner" (corrispondente ad una feature con relativo valore di soglia) che meglio classifica il training set pesato. La combinazione di tali classificatori ottenuti ad ogni livello della cascata determina uno "Strong classifier".

La prima fase di boosting assume pesi uniformi del set di dati di addestramento, mentre le successive fasi assegnano pesi maggiori per non classificare istanze negative come positive. Le differenti varianti di boosting,"Discrete, Real e Gentle" si differenziano su come loro determinano l'aggiornamento dei pesi dei dati di training. Per la trattazione corrente viene adottato Gentle adaboost in quanto più veloce ed accurato rispetto gli altri due. Riferire al punto 3 dell'algoritmo 2 per vedere come vengono aggiornati i pesi.

N weighted training examples: $(x_1, y_1), \ldots, (x_N, y_N)$, where x_i are the images and $y_i \in \{-1, 1\}$ the classified output $i \in \{1, \ldots, N\}$.

Initialize the weights w_i

$w_i = \frac{1}{N}$.

The following three steps are repeated to select simple Features until a given detection rate d is reached:

1. Every simple classifier, i.e., a feature, is fit to the data. Hereby the error e is calculated with respect to the weights w_i.
2. The best feature h_t is chosen for the classification function. The counter t is incremented.
3. The weights are updated with $w_i = w_i \cdot e^{-y_i h_t(x_i)}$ and renormalized.

The final output of the classifier is $\text{sign}(\sum_{t=1}^{T} h_t(x)) > 0$, with $h(x)$ the weighted return value of the feature.
Next, a cascade based on these classifiers is built.

Algorithm 2: Algoritmo Gentle AdaBoost

Capitolo 3

ENCARA

3.1 ENCARA

Il sistema introdotto in questo capitolo, rispetto agli altri, effettua il riconoscimento di volti in video stream. È un approccio solido per il rilevamento di facce in multi-risoluzione ed in tempo reale. Tale implementazione tiene in considerazione relazioni esistenti tra i frames, come per esempio la coerenza temporale e spaziale.

L'approccio quindi, si basa sulla fusione tra le due categorie di conoscenza, descritte nel capitolo precedentemente, in modo da ottenere il meglio dalla conoscenza implicita e da quella esplicita. L'idea anche qui, sta nel combinare incrementalmente più classificatori complessi in cascata, permettendo a regioni antecedenti dell'immagine di essere scartate rapidamente, risparmiando cosi, tempo di computazione. L'attenzione è prorogata alla modellazione in tempo reale di ogni faccia rilevata. Pertanto, la combinazione di informazioni ottenute dalla modellazione e la coerenza temporale accelerano il trattamento del fotogramma successivo. Con questo approccio è possibile ottenere miglior tassi di rilevamento con minor costo di computazione.

La classificazione (ovvero la capacità di determinare se un ipotesi di faccia è vera oppure no) nella face detection, è una caratteristica fondamentale. In merito, esistono approcci basati su classificatori individuali o multipli. L'architettura in esame, solo per quanto riguarda la combinazione di classificatori, segue la struttura di "Viola e Jones", ma non utilizza per niente tale metodo. La differenza è che i classificatori non sono basati solo su features rettangolari, ma sono di diversa natura.

Encara considera una cascata di costituita da diversi moduli, ognuno dei quali rappresenta un classificatore.

Inizialmente il processo seleziona regioni che presentano caratteristiche per cui, è possibile ipotizzare la presenza di una faccia. Un primo modulo classificatore conferma o rifiuta tale ipotesi. Se esso non conferma, l'ipotesi viene immediatamente scartata e la catena viene interrotta, direzionando il sistema verso altre regioni salienti dell'immagine corrente. Se l'ipotesi viene confermata, passa al controllo del successivo modulo nella cascata. Se l'ipotesi consecutivamente viene confermata da tutti i moduli, in output si

ottiene un risultato positivo per una rilevazione di faccia

Per migliorare le performance, tecniche basate sulla conoscenza contestuale della geometria facciale e dell'aspetto vengono combinate e coordinate con l'informazione temporale estratta dallo stream video. Il numero di moduli e la loro complessità deve essere sufficiente a garantire performance di detection simili ad ogni fase, mentre si minimizza la computazione. Quindi dato il tasso di falsi positivi, f_i, e il detection rate, d_i, dell'i−esimo modulo classificatore, si ottengono per la cascata i seguenti valori:

- **False positive rate**: $FP = \prod_{i=1}^{K} f_i$.
- **Detection rate**: $D = \prod_{i=1}^{K} d_i$.

Dove K è il numero dei classificatori.

Queste espressioni mostrano che la combinazione a cascata è in grado di ottenere buoni tassi di classificazione e bassi tassi di falsi positivi, se il tasso di rilevamento di classificatori singoli è buono, vicino a "1", mentre non è buono se vicino a "0".

Le principali caratteristiche di Encara sono le seguenti:

1. Usa solo informazione visiva fornita da una singola camera. Non considera informazione stereo.
2. Non ha bisogno di un dispositivo di acquisizione ad alta qualità. Utilizza camere standard.
3. È progettato per rilevare facce frontali in video streams.
4. Usa coerenza spaziale e temporale.
5. Fà uso di conoscenza implicita ed esplicita in modo da ottenere un sistema real time per ogni generico tipo di hardware. Encara seleziona candidati di facce con la conoscenza esplicita per poi applicare un rapido approccio basato sulla conoscenza implicita.
6. È sviluppato per fornire alte performance in ambienti interattivi dove fallimenti occasionali nel riconoscimento di facce non sono critici.
7. È progettato per essere aperto, in modo tale da poter aggiungere moduli, modifiche e miglioramenti.

ENCARA è brevemente descritto in termini dei seguenti moduli organizzati in cascata, in cui ognuno di essi conferma o rifiuta l'ipotesi:

M0- **Tracking**: Se è presente una rilevazione recente, il prossimo frame sarà analizzato, cercando prima gli elementi facciali detectati nel frame precedente: occhi e bocca. Se le posizioni tracciate sono simili a quelle nel precedente frame ed il test di aspetto è passato, Encara considera che una faccia è stata rilevata.

M1- **Face Candidate Selection**: Vengono selezionate aree rettangolari nell'immagine, dove potrebbero trovarsi facce. L'implementazione attuale utilizza il colore della pelle come approccio per selezionare regioni rettangolari nell'immagine che potrebbero contenere facce.

Il modulo si basa sui seguenti passi:

a) *Input image transformation*: Le immagini in input hanno un formato RGB e vengono normalizzate in uno spazio di colore "rosso−verde" (Wren, 97).

b) *Color blob detection*: Viene utilizzato un *Dilation kernel* 3x3 che viene fatto traslare sull'immagine normalizzata. Si ottiene cosi in output un'immagine in cui vengono evidenziati "blobs" in base al colore della pelle di una presunta faccia. Ovviamente è possibile la presenza di falsi positivi. Per questo vengono analizzati i "blobs" maggiori finché uno viene rilevato come faccia. Questo implica che Encara, permette la detection di una sola faccia per frame.

M2- **Facial Features Detection**: Nella regioni candidate, ottenute dal modulo precedente, il sistema rimuove euristicamente elementi che non sono parti della faccia, ad esempio il collo, per ricercare caratteristiche facciali quali occhi, bocca e naso.

Il modulo M2 consiste nei seguenti passi:

a) *Ellipse approssimation*: Blobs di maggiori dimensione, classificati in base al colore della pelle nell'immagine di input, sono adattati ad un ellisse utilizzando il metodo di Sobottka e Pitas. L'approssimazione del blob a ellisse ritorna l'area, l'orientazione e la lunghezza degli assi in pixel (l'asse maggiore rappresenta l'altezza del viso, mentre quello minore la larghezza).

b) *Ellipse filter*: I candidati vengono scartati in base alla dimensione dell'ellise, in base quindi a filtri geometrici che considerano la lunghezza degli assi ed in base alla coerenza temporale che fornisce informazioni sull'orientamento. Un candidato viene scartato se l'orientazione del frame successivo non è coerente con l'orientazione del frame precedente.

c) *Rotation*: L'approssimazione ad ellisse fornisce informazioni circa l'orientamento, quindi un blob candidato come faccia viene ruotato di quell'angolo affinché gli occhi risultano in linea orizzontale. In questo modo la ricerca di caratteristiche facciali avviene con facce frontali.

M3- **Normalization**: Occorre un processo di normalizzazione per ridurre il problema della dimensionalità.

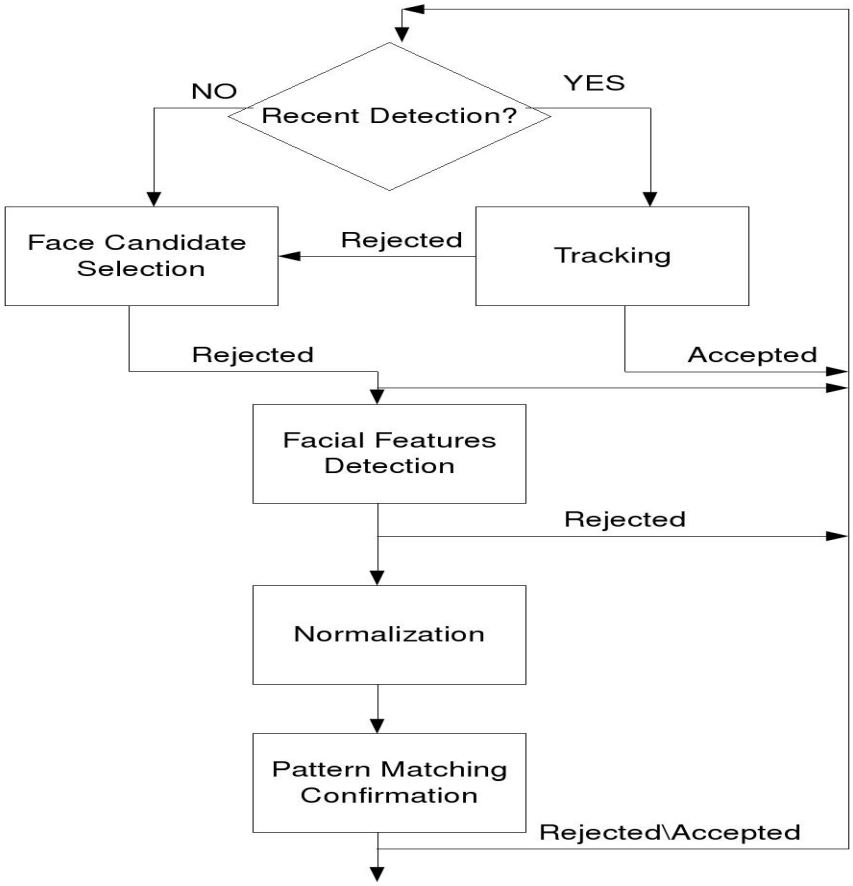

Figura 3.1: La figura rappresenta i moduli di funzionamento di ENCARA.

M4- **Pattern Matching Confirmation**: Come si vede dalla figura 3.1, l'ultimo passo consiste nel classificare l'immagine normalizzata come faccia o non faccia. Riduce il numero di falsi positivi e si basa su tecniche di conoscenza implicita. Per l'aspetto degli occhi, si considera una regione di 11 x 11 pixels.

Pseudocodice Encara:

```
1: M0:-Tracking
2: Last eye and mouth search
3: Test with previous
4: if similarity test is passed then
5:    jumps to normalization M3,returning if something fails
6: end if
7: M1:- Face Candidate Selection
8: Input Image Trasformation
9: Color Blob Detection
10: M2:-Facial Feauteres Detection
11: Ellipse Approssimation
12: Refusing Ellipses
13: Rotation
14: Neck Elimination.New Rotation
15: Integral Projections
16: Eyes Detection.Three pairs are used
17: 1)Gray minima pair searched on areas defined by color
18: 2)Similarity minima pair searches on areas defined by recent detection,and
19: 3)Gray minima pair searched on areas defined by recent detection
20: M3:-Normalization
21: Normalization
22: M4:-Pattern Match Confirmation
23: Eye Appearance Test
24: Face Appearance Test
25: if The Face is Considered frontal then
26:    Mouth detection
27:    Nose detection
28:    Eye patterns saving
29: end if
```

Algorithm 3: Algoritmo ENCARA

3.2 ENCARA2

In questa sezione viene introdotta la versione di Encara che è stata utilizzata nel progetto. Encara2 permette ad un sistema di visione artificiale di riconoscere più volti in video stream, a differenza della precedente versione in cui era possibile la detection di una sola faccia. Encara2 inoltre, combina detectors basati sulla cascata sviluppata nella versione base e detectors basati sul metodo "Viola e Jones" (la versione base non utilizzava tale metodo).

La tecnica di rilevamento di volti qui descritta ha due differenti modi di lavoro in funzione degli ultimi eventi di face detection riportati (vedere figura 3.2 per uno schema riassuntivo):

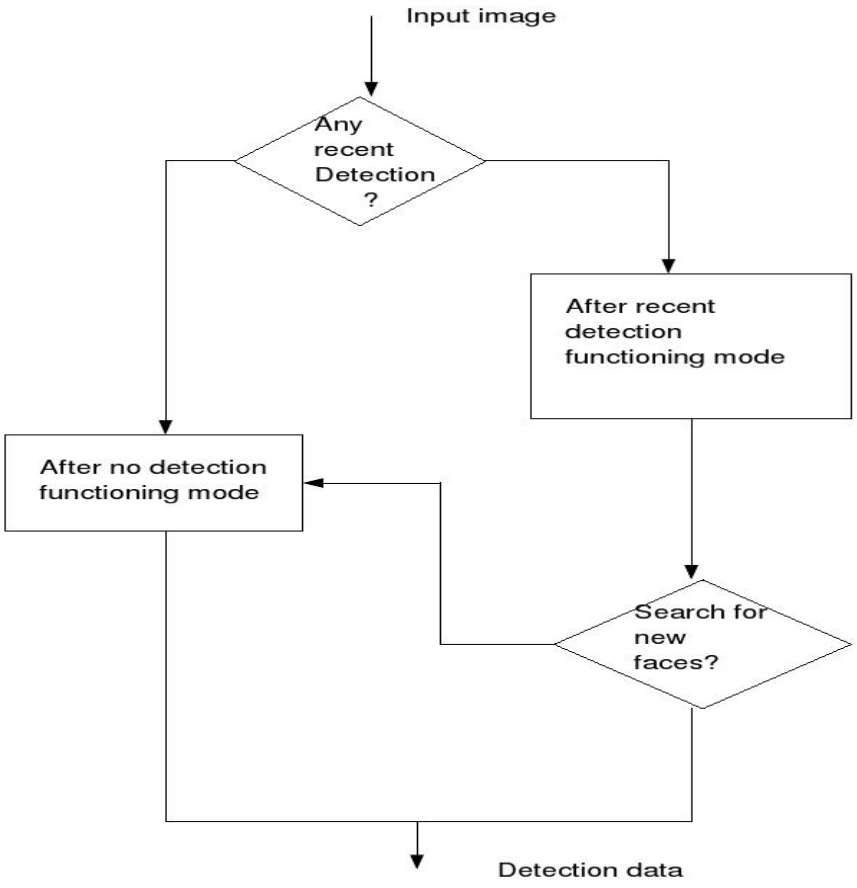

Figura 3.2: La figura rappresenta i due modi di funzionamento di Encara2.

- **Dopo nessuna rilevazione di faccia**:

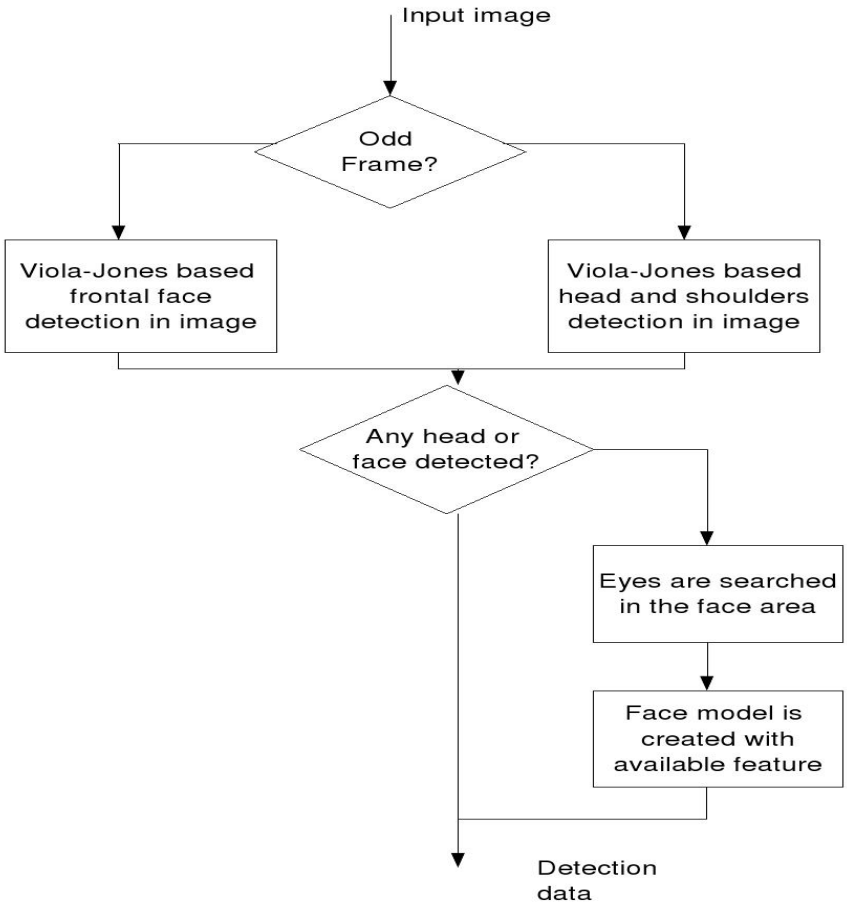

Figura 3.3: La figura rappresenta il modo di funzionamento "Dopo nessuna rilevazione di faccia".

In figura 3.3, è schematizza la modalità di funzionamento che si ha all'inizio di una sessione, nel periodo di tempo in cui tutte le persone sono scomparse dal campo visivo o nessuna di esse venga rilevata in tale sessione.

Questo modo di lavoro utilizza due rilevatori a finestra scorrevole sulla base del rilevamento della detection di oggetti descritto da "Viola e Jones": uno che considera facce frontali, "Viola e Jones", e uno che considera il contesto locale, "Castrillòn". Quest'ultimo ottiene miglior tassi di riconoscimento per immagini a bassa risoluzione in cui testa e spalle sono visibili. La ricerca avviene su una grandezza minima che può essere di "24 x 24" e "20 x 20"pixels. Allo scopo di non perdere tempo di elaborazione, i due detectors si eseguono alternativamente (uno per i frames pari e uno per i frames dispari).

Assumendo che un volto sia rivolto frontalmente, per qualsiasi faccia o testa rilevata, il sistema modella il colore. Da quel modello cerca di individuare le caratteristiche facciali: bocca, naso, ed occhi (l'implementazione attuale ricerca

gli occhi solo per una persona). Inoltre il sistema memorizza non solo posizione e dimensione, ma anche la media del colore, usando uno spazio di colore "rosso-verde" normalizzato, ed il patterns degli occhi (se detectati).

Pertanto, un volto si caratterizza per i seguenti attributi:

- posizione,
- dimensione,
- colore,
- posizione occhio sinistro,
- posizione occhio destro,
- pattern occhio destro,
- pattern occhio sinistro,
- pattern della faccia.

- **Dopo recente rilevazione di faccia**:

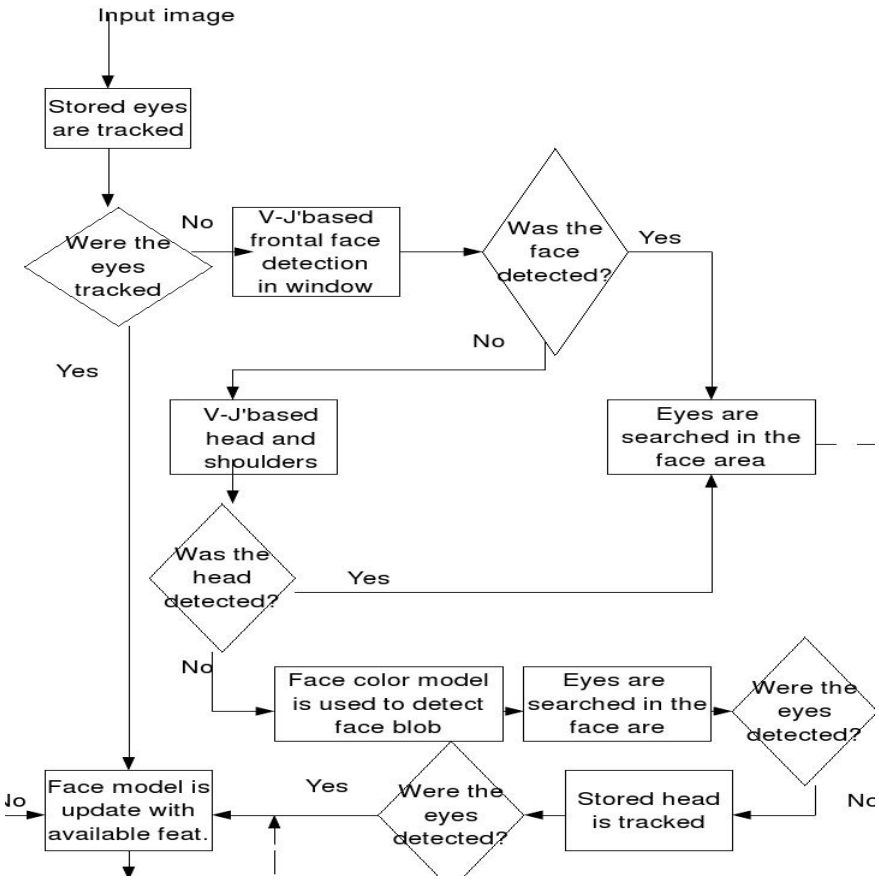

Figura 3.4: La figura rappresenta il modo di funzionamento "Dopo recente rilevazione di faccia".

In questa modalità (consultare la figura 3.4 per un riassunto grafico), per ogni rilevazione si ha un modello con differenti caratterisiche (ogni faccia è stata modellata usando features diverse). Queste caratteristiche (features) andranno applicate su nuove immagini (frame) nella sequenza video per detectare di nuovo una faccia, la zona di investigazione (o regione di ricerca) verrà ampliata per il prossimo frame e le caratteristiche incontrate in precedenza, si convertiranno in una sottofinestra (vedere figura 3.5) per quel frame. Queste tecniche sono utilizzate finchè le caratteristiche ampliate individuano una nuova faccia coerente con il precedente rilevamento, (non vanno applicate necessariamente ad ogni frame). Queste considerazioni accelerano tutto il processo (speed up).

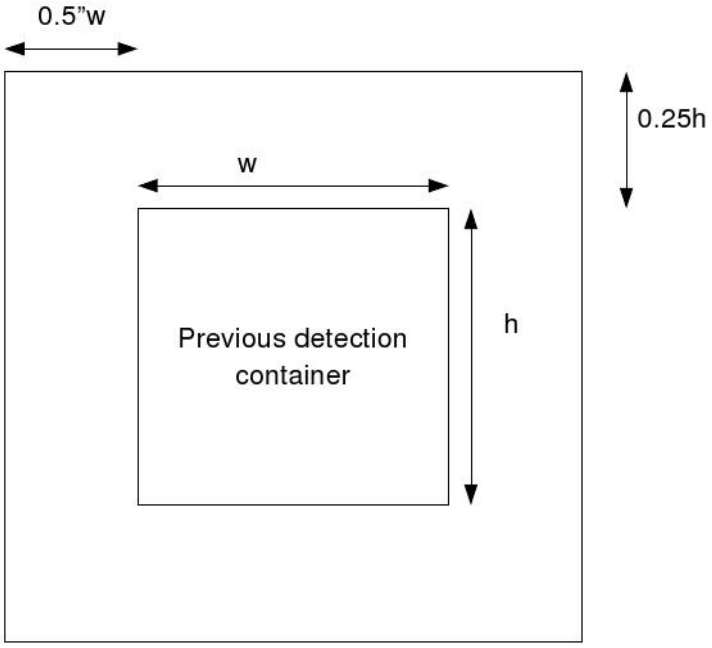

Figura 3.5: La regione di ricerca usata per ogni faccia individuata, nel prossimo frame viene definita come espansione del precedente contenitore della faccia rilevata.

Nello schema di figura 3.4 sono rappresentati i seguenti moduli:

- **Tracciamento degli occhi**: Se gli occhi sono disponibili nel modello di caratteristiche costruito precedentemente, un rapido algoritmo di tracking [C. Guerra] cerca la differenza minima nella zona di ricerca e una soglia dinamica è utilizzata per decidere se gli occhi sono persi o meno.

- **Frontal face detector**: Si cerca di rilevare una faccia utilizzando il metodo di "Viola e Jones", nella finestra di ricerca, solo se il tracker non ha tracciato gli occhi.

- **Detector di faccia in contesto locale**: Se falliscono le precedenti tecniche, viene applicato il detector basato sul contesto locale nella finestra di ricerca.

- **Colore della pelle**: Se falliscono le tecniche anteriori, si utilizza il modello del colore della pelle per localizzare una faccia nella zona di ricerca. Se l'esito è positivo si cercheranno gli occhi utilizzando la tecnica spiegata nel seguito.

- **Face tracking**: Se tutti gli altri falliscono, si effettua il tracking della faccia nella zona di ricerca del face pattern memorizzato precedentemente. Di volta in volta si necessita la conferma della presenza della persona umana, altrimenti sarà considerata persa.

  Quando si individua una faccia, e i suoi occhi non sono tracciati, si utilizza il colore della pelle per la rilevazione di caratteristiche facciali. Inoltre ogni cinque frame, un detector basato su "Viola e Jones" viene applicato a tutta l'immagine per rilevare nuove facce. Si confrontano le nuove facce con quelle già rilevate per la coerenza temporale e si eliminano quelle superflue.

  Se non si rilevano facce per un periodo di tempo, il sistema passa all'altro modo di lavoro (dopo nessuna rilevazione).

Detection degli occhi

Encara2 assume la posizione frontale della faccia rilevata. Si possono avere situazioni in cui Encara2 non fornisce la posizione degli occhi, questo dovuto al fatto che il sistema può fallire il loro rilevamento o perché essi non sono visibili.

Il processo di rilevazione della coppia di occhi è riassunto in figura 3.6

(1) *Skin blob detection*: Il modello della pelle è utilizzato per individuare i confini della faccia eliminando parti superflue, come il collo. Si inserisce un'ellisse per ruotare la faccia verticalmente (come in figura 3.3).

(2) *Eyes locations*: Sono usati differenti detectors per stimare la posizione degli occhi, dopo che una faccia è stata rilevata :

   (a) *Dark areas*: Gli occhi sono più scuri rispetto all'ambiente circostante;

   (b) *Detector degli occhi basato su Viola e Jones*: La posizione degli occhi può essere stimata approssimatamente e il detector cerca gli occhi su una finestra di grandezza minima di "16 x 12"pixels;

   (c) *Viola-Jones based eye pair detector*: Se gli altri detectors falliscono, il detector effettua un'altra stima della posizione degli occhi per applicare i passi (a) e (b). Il pattern minimo ricercato è "22 x 5".

(3) *Normalizzazione*: La posizione degli occhi, se detectati, fornisce una misura per normalizzare la faccia candidata a una grandezza standard.

(4) *Pattern matching confirmation*: Una volta che la faccia è stata normalizzata, si effettua il confronto con le immagini del dataset (4000 immagini estratte da internet), in 2 steps, utilizzando l'analisi PCA e considerando i seguenti 2 spazi (Principal Component Analysis):

(a) *Eye appearance test*: Una certa regione ("11 x 11") intorno agli occhi viene proiettata nello spazio PCA e ricostruita. L'errore di ricostruzione può dare errore in fase di detection degli occhi.

(b) *Face appearance test*: L'immagine è prima proiettata in uno spazio PCA e successivamente il suo aspetto è testato utilizzando un classificatore (SVM).

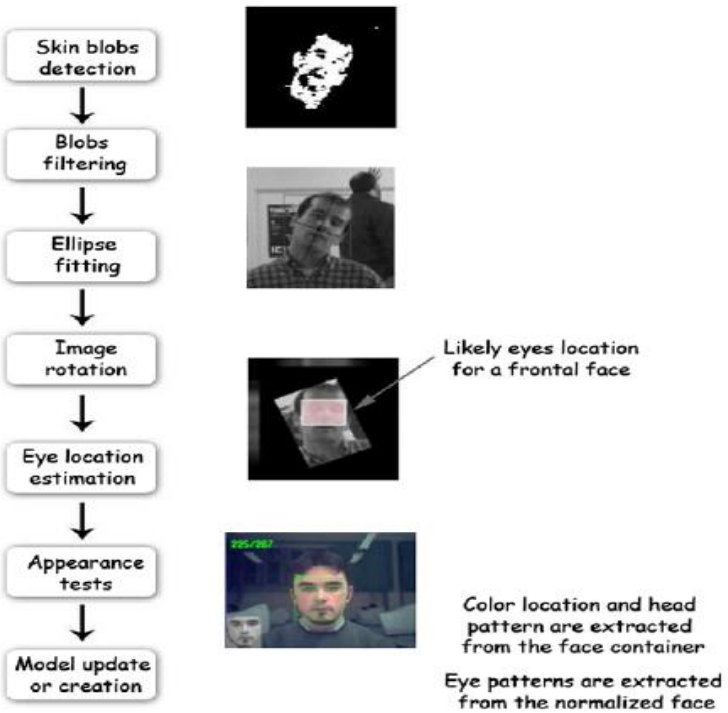

Figura 3.6: Eye detection process.

Come si vede dalla figura 3.6, il processo di detection degli occhi combina moduli della versione base di Encara con detector "Viola e Jones".

3.3 Risultati e limiti

Il componente Encara2 funziona bene quando le condizioni di luce sono stabili (ne luce forte, ne luce debole), quando una persona si sposta lentamente e quando tale persona viene inquadrata frontalmente dalla camera. In tali condizioni la faccia viene sempre riconosciuta. Le caratteristiche facciali (bocca naso ed occhi) a volte non vengono rilevate. Oltre a risultati di successo può succedere che encara inceppi in falsi positivi. Si è provato encara in vari ambienti e con varie condizioni di luce, ponendo mani davanti la faccia, muovendosi sia velocemente che lentamente e a differenti distanze.

Nel seguito si riportano i risultati che fornisce Encara2:

- **Risultati**: Come si vede nella figura 3.7, avendo condizioni di luce ottimali, Encara2 riconosce perfettamente un volto umano con tutte le sue caratteristiche.

Figura 3.7: La figura rappresenta il rilevamento di un volto con tutte le sue caratteristiche facciali.

- **Limiti**: Alcune volte succede che Encara2 riconosce volti senza caratteristiche (figura 3.8), oppure che riconosce volti che non sono volti (falsi positivi, figura 3.9).

Figura 3.8: La figura rappresenta il rilevamento di un volto senza le sue caratteristiche facciali.

Figura 3.9: La figura rappresenta il rilevamento di un falso positivo.

3.4 Utilizzo di Encara2

Nello sviluppo del progetto non è stato necessario implementare l'algoritmo Encara2, in quanto sviluppato dal prof. Castrillòn Santana. Si è considerato Encara2 come una componente già esistente. Quindi, per la detection di facce si sono utilizzate le funzioni che tale componente mette a disposizione. A seguire si descrive come è stata utilizzata.

3.4.1 Uso di ENCARA2.2lib

Per usare la componente, si necessita aggiungere nel codice l' "include" della libreria ENCARA2.2lib.

```
#include "ENCARA2_2lib.h"
```

Si dichiara la variabile:

```
 CENCARA2_2Detector *ENCARAFaceDetector;
```

Si crea il detector nella seguente maniera:

```
//Creation
//ENCARA2.2 data directory (to be modified according to your configuration)
sprintf(ENCARAdataDir,"/home/alessio/cvs-projects/encara-2.09/ENCARA2.2data");

//Defaults dimensions
m_width=320;
m_height=240;

//ENCARA2 creation
ENCARAFaceDetector=new CENCARA2_2Detector(ENCARAdataDir,m_width,m_height);
```

Per distruggere l'oggetto si utilizza la seguente istruzione:

```
//Destruction
delete ENCARAFaceDetector;
```

Il detector viene creato una sola volta all'inizio dell'esecuzione dell'applicazione. Una volta caricato il detector, esso può essere usato come segue:

```
//Using ENCARA2
//Process the image
```

```
//The last parameter indicates that recognition
ENCARAFaceDetector->ProcesaImagenENCARA2MultiCaras(imagen,2);

//Paints results
ENCARAFaceDetector->PaintFacialData(imagen,CV_RGB(0,255,0));
```

- **Face detection**:

  Standard ENCARA2 interface. This routine searches multiple faces on image h.

  ```
  void ApplyENCARA2(IplImage *h, int escala=2, bool boReconoce=false);
  ```

  h Image to be processed

  escala 1 no dimension reduction, 2 scales down the image to half size

  boReconoce Applies recognition (not needed for detection and not used in this version)

  This routine paints the information referred to last detection performed in an image.

  ```
  void PaintFacialData(IplImage *h,CvScalar color, bool boReconoce=false);
  ```

  h Image to be painted

  color color to be used for painting

  boReconoce Applies recognition (not needed for detection and not used in this version)

- **Facial Data**:

  This function allows the user to read data about currently active detection threads.

  ```
  CFacialDataperImage *GetFacialData();
  ```

  Allows the user to read data about already lost detection threads.

  ```
  CFacialDataperImage *GetFacialDataLost();
  ```

  A detection thread contains information related to face detection along time. The "CFacialDataperImage" structure contains for any detection thread the following fields available:

```
//CURRENT FRAME INFORMATION FOR EACH DETECTION THREAD
//Face Location and Geometry
int x1,x2,y1,y2; //Face container
bool hs; //HS container availability
int hsx1,hsy1,hsx2,hsy2; //HS container
bool fb; //Full body container availability
int fbx1,fby1,fbx2,fby2; //Full body container
bool eyepair; //Eye pair container availability
int eyepairx1,eyepairy1,eyepairx2,eyepairy2; //Eye pair container
bool leye,reye; //Eyes (left and right) validity
int e_lx,e_ly,e_rx,e_ry; //Eye pupils
bool nose; //Nose validity
int np_x,np_y; //Nose peak
bool nosecontainer; //Nose container validity
int nosex1,nosey1,nosex2,nosey2; //Nose container
bool mouth; //Mouth validity
int ml_x,ml_y; //Mouth left estimated corner
int mr_x,mr_y; //Mouth right estimated corner
bool mouthcontainer; //Mouth container validity
int mouthx1,mouthy1,mouthx2,mouthy2; //Mouth container
bool boBreastContainer; //Breast container validity
int brx1,brx2,bry1,bry2; //Estimated breast container

//Color features
float rn_min,rn_max,gn_min,gn_max;//Color ranges

           //Set if detection was achieved by means of skin color
bool boColorBasedFound;

           //Set if there is a color model for this detection thread
bool boColorAlreadyInitialized;

           //Consecutive frames of detection thread
long persistence;

        //Number of frames remaining
        //to cancel a not found previous detection

         int framestodie;

        //Eyes have been detected at least once for this face

         bool boEyesDetectedAtLeastOnceforThisFace;
```

```
//Tracking variables
bool tracked  //Set if the head was tracked not detected
int iconttracked; //Consecutive frames tracked
bool eyestracked; //Set if eyes were tracked not detected
bool leyetracked; //Set if left eye was tracked
bool reyetracked; //Set if right eye was tracked
bool mouthtracked; //Set if the mouth was tracked
bool nosetracked; //Set if the nose center was tracked
int escalaCara; //Relates original face image with face pattern size
float escalaCaraf; //Similar but float value

//Trackers

        //Tracker availability
bool boHeadTracker,boLEyeTracker,boREyeTracker,
boLMouthTracker,boRMouthTracker,boNoseTracker;
COpenTracking *HeadTracker;//Tracker instance for head/face
COpenTracking *LEyeTracker;//Tracker instance for left eye
COpenTracking *REyeTracker;//Tracker instance for right eye
COpenTracking *NoseTracker;//Tracker instance for nose
COpenTracking *LMouthTracker,*RMouthTracker;
//Tracker instances for mouth

//Label used if no recognition is applied
char Label[32];

//Normalized images
        //Is memory for normalized images already allocated?
bool boiplFaceImagesNormalized;

        //Have the eyes been detected for this detection thread
bool boStableTrackedFace;
        //Has been saved for this detection thread already an exemplar?
bool boStableTrackedFaceSaved;

        //Indicates if there is a normalized image available
       // for current detection

bool boNormalizedAvailable;

        //Normalized (in size) image containing a little bit more than
IplImage *iplFaceContextNormalized,
 *iplFaceNormalized,//the CARAX*CARAY part
        //CARAX*CARAY including illumination normalization
*iplFaceNormalizedIN;
```

```
       //Exemplars. Information related

        //to the selected normalized images
        //extraced for this detection thread

        //Contains selected exemplars along the whole
        //detection thread history

CExemplars Exemplars;
```

The CExemplar structure contains information related to the images selected along the detection thread history:

```
        // //Exemplars information
int iNImages; //Number of exemplars
int *persistence; //Array of exemplars persistence
long *timestamp; //Indicates the moment when each exemplar was added

//Exemplars normalized images
IplImage **StableImage; //Array of exemplars whole face image
        //Array of exemplars whole face image illumination normalized
IplImage **StableImageIN;
IplImage **StableImageEyes; //Array of exemplars eyes area image
IplImage **StableImageMouth //Array of exemplars mouth area image
THist *Hist;//Array of exemplars histogram
        //Set if the histogram
        //associated to the exemplar has been computed
bool *boHistogramAvailable;

       //Exemplars PCA recosntruction error info

       //Contains the PCA reconstruction error
float *fPCARecError_Quore;

        //Contains the average PCA reconstruction error
float fAvg_PCARecError_Quore;

//Recognition (not in use in detection demo)
char **Id; //Identity assigned (if classified) for each exemplar
        //Label assigned (if classified and verified) for each exemplar
char **Idauth;
        //Gender assigned (if classified and verified) for each exemplar
```

```
char **Gender;
bool *bogenderlabelled;//Set if the exemplar gender has been labelled
bool *boidlabelled; //Set if the exemplar id has been labelled
bool *boauthlabelled; //Set if the exemplar auth has been labelled
int *genderclass; //Class assigned to the exemplar for gender
int *idclass; //Class assigned to the exemplar for id
int *authclass; //Class assigned to the exemplar for auth
```

Capitolo 4

Algoritmi di tracking, leg-detection e sincronizzazione

Come descritto nel capitolo 2, la rilevazione di facce è una componente fondamentale in molte applicazioni di elaborazione di immagini basate sul riconoscimento del volto, sul riconoscimento di persone, su codifiche video e interfacce intelligenti. L'obiettivo in questo passo è rilevare e localizzare in una sequenza di immagini costituenti i frames di un video un numero non prefissato di facce.

La face detection è un problema difficile in quanto i volti umani presentano grande variabilità nelle seguenti caratterizzazioni:

1. *Aspetto*: dimensione viso, incarnato, colore capelli, lineamenti...
2. *Orientazione*: di faccia, profilo, verso il basso, dalla testa..
3. *Illuminazione*: disposizione di luci e ombre sul viso..
4. *Espressione*: corrucciato, sorridente, etc.
5. *Camuffamento*: occhiali, trucco, colore capelli, invecchiamento.

Il tracciamento del volto umano, consiste in tecniche di analisi dell'immagine che permettono di stimare la posizione dello stesso e la deformazione delle sue parti non rigide.

Nel presente capitolo verranno descritti gli algoritmi implementati che permettono al robot la rilevazione ed il tracciamento di una persona.

4.1 Algoritmo di tracking

Utilizzando il metodo Encara2, descritto nel capitolo 3, ed una camera che si orienta a seconda degli spostamenti della faccia individuata, si è sviluppato un algoritmo per il tracciamento di una persona.

Per effettuare il tracking del volto, la camera ruota tramite un supporto pan/tilt: il pan ruota su di un piano orizzontale, il tilt ruota su di un piano verticale. Questo consente di inquadrare il volto rilevato al centro della finestra di visualizzazione delle immagini (ossia dove viene riprodotto ciò che la camera acquisisce). Questa rotazione, insieme a quella della piattaforma, permette di allineare il sistema di riferimento della camera con quello del robot. Il robot, in questo modo, avanza con i due sistemi in fase e simula una persona che, mentre cammina, guarda in avanti.

Durante lo sviluppo si sono incontrati vari problemi inerenti alla camera. Per ovviare a tali incovenienti si sono implementati differenti algoritmi di tracking che verranno accennati nel capitolo 6 (Esperimenti).

Conoscendo la posizione del rettangolo che racchiude il volto, la camera ruota (su piano orizzontale e verticale) in modo da portare tale rettangolo al centro della finestra di visualizzazione. Quindi, come si può vedere nella figura 4.1, si calcola il centro del rettangolo e si comanda la camera affinché si ottenga coincidenza con il centro della finestra.

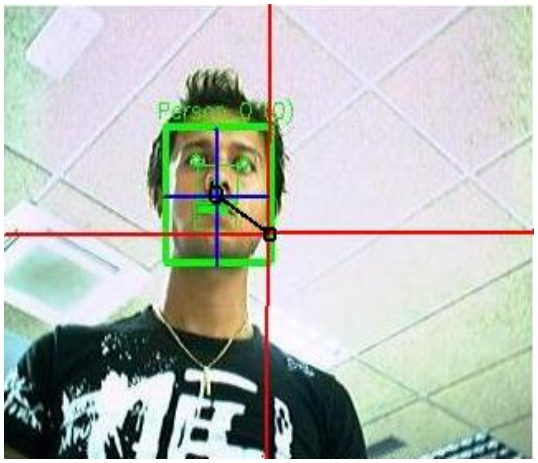

Figura 4.1: La figura rappresenta la finestra di visualizzazione divisa in quadranti. Il segmento in nero rappresenta la differenza tra il centro del rettangolo ed il centro della finestra.

Si considera la finestra divisa in quattro quadranti ed a seconda di dove si trova il rettangolo del volto si ruota il pan a destra o sinistra ed il tilt in alto o in basso.

In particolare se la differenza tra il punto medio dell'asse x del rettangolo rispetto al centro della finestra è maggiore di zero (quadrante a sinistra) si ruota il pan di un angolo che assume, via via, valori decrescenti (ruota a sinistra nel piano orizzontale), viceversa se è minore di zero (quadrante a destra) si ruota di un angolo che assume

valori crescenti (ruota a destra nel piano orizzontale). Considerando l'asse y si usa lo stesso ragionamento per il tilt. Se la differenza è maggiore di zero (quadrante in alto) si ruota il tilt di un angolo che assume valori crescenti (ruota verso l'alto su di un piano verticale); se la differenza è minore di zero (quadrante in basso) si ruota di un angolo che assume valori decrescenti (ruota verso il basso su di un piano verticale). Infine se la differenza, o in piano orizzontale o in piano verticale, è uguale a zero, la camera non ruota.

Per ridurre l'errore di tracking di falsi positivi si è aggiunta una condizione che verifica se la faccia rilevata è effettivamente una faccia. Tale condizione prevede il test sulla presenza degli occhi (riferire all'algoritmo 4). Per ogni fotogramma che la porta della componente riceve, viene applicato Encara2. Se ci sono facce vengono calcolate le posizioni rispetto ai quadranti, sopra menzionati. Per la prima faccia rilevata (ossia il primo elemento del vettore "faces" di "FacialData") viene calcolato di quanto devono spostarsi il pan ed il tilt dalla posizione attuale, affinché la camera inquadri l'immagine in posizione centrale. Tale incremento è dato dal seguente sistema di controllo:

$$panAngle = currentPan + (\frac{difference\_x}{windowHalfPointX}) * (5 * Kp\_pan);$$

$$tiltAngle = currentTilt + (\frac{difference\_y}{windowHalfPointY}) * (5 * Kp\_tilt);$$

dove $difference\_x$ rappresenta la differenza in pixels tra il centro del rettangolo ed il centro della finestra, mentre $windowHalfPointX$ rappresenta la coordinata del punto medio della finestra.

Ad ogni iterazione viene sommato alla posizione attuale, sia del pan che del tilt, un certo incremento dipendente dal quadrante in cui si trova il rettangolo. Viene calcolato quanto esso dista dal centro, sia in piano orizzantale ($difference\_x$) e sia in piano verticale ($difference\_y$): tanto più è distante dal centro, tanto più il pan ed il tilt subiscono un incremento alto e ruotano più rapidamente. Tale incremento è compreso tra 0 e 1. Trattondosi di sistema di controllo, si utilizzano le costanti Kp_pan/tilt per controllare la rotazione della camera.

Infine, per evitare l'accumulo di comandi pendenti nelle code della camera è stata introdotta una formula che calcola il tempo di attesa necessario alla camera per eseguire e smaltire tutti i comandi correttamente. Se tra un comando e l'altro non si mandasse in sleep il programma per tale "delay", la camera si comporterebbe in modo strano, in quanto non farebbe in tempo ad eseguire tutti i comandi che arrivano con velocità maggiore rispetto alla loro esecuzione. Tale ritardo è calcolato con la seguente formula:

- Si considerano dapprima i fattori del pan e del tilt, ottenuti nel seguente modo:

$$panFactor = \frac{|(panAngle - currentPan)|}{(\_PTZ\_MAX\_PAN\_ - (\_PTZ\_MIN\_PAN\_))};$$

$$tiltFactor = \frac{|(tiltAngle - currentTilt)|}{(\_PTZ\_MAX\_TILT\_ - (\_PTZ\_MIN\_TILT\_))}$$

- successivamente si manda in sleep il programma, per un certo ritardo, calcolato nel seguente modo:

  $delay = max(panFactor, tiltFactor) * (\_PTZ\_MAX\_COMMAND\_DELAY\_ - \_PTZ\_MIN\_COMMAND\_DELAY\_) + \_PTZ\_MIN\_COMMAND\_DELAY\_$;

  dove:

  ```
  _PTZ_MIN_COMMAND_DELAY_=290, // milliseconds
  _PTZ_MAX_COMMAND_DELAY_=1600,// milliseconds
  ```

  Più precisamente si calcola il tempo necessario per eseguire un comando e tramite la condizione espressa nell'algoritmo 4 si elaborano solo le immagini correnti.

  Se il tempo attuale è maggiore del tempo per eseguire un comando, sommato al tempo di attesa (delay), l'algoritmo elabora le immagini; altrimenti il programma si mette in attesa che la camera abbia eseguito il comando precedente. In tale periodo la porta di input continua a ricevere immagini che non sono elaborate.

Dettagli implementazione

```
Initialization()
Receive Images from CAMERA_IMAGE port
set CurrentTime
if CurrentTime > ExecuteCommandTime + Delay then
  convert RGB image to BGR image
  Apply Encara
  if One face is detected then

    if Eyes are detected then
      return rectangle's coordinates from FacialData structure
      compute the position of the face
      set Camera with panAngle and tiltAngle
      compute delay
      send Commands to PTZJOINTS port
    end if

  end if
  Wait for execution command, don't process the images
end if
```

Algorithm 4: Tracking algorithm (input: images received from CAMERA_IMAGE port, output: send processed images by ENCARA2 to ENCARATRACKINGIMAGE port)

L'algoritmo 4 rappresenta lo pseudo codice dell'algoritmo di cui sopra.

Dal momento che Encara2 lavora con immagini in formato BGR, si è ricorso all'uso della libreria OpenCv per la conversione delle immagini acquisite che sono in formato RGB.

La parte di codice che effettua questo passaggio è la seguente (riferire al capitolo 5 per dettagli sulle funzioni):

1. La prima istruzione memorizza in una variabile privata (di tipo pImage) l'immagine che riceve dalla porta "CAMERAIMAGE",

```
cvSetImageData(pIplImage,
(unsigned char *) pImage->getImage()->rawCells(),
pIplImage->widthStep);//IMMAGINE RGB
```

2. La seconda effettua la conversione da RGB a BGR (srcBGR) dell'immagine memorizzata precedentemente:

```
cvCvtColor(pIplImage, srcBGR, CV_RGB2BGR);
// RGB to BGR
```

Una volta ottenute immagini in formato BGR, si utilizza la seguente funzione per rilevare la faccia (riferire al cap. 3 per le primitive encara):

```
//Apply eNCARA

ENCARAFaceDetector->ApplyENCARA2(srcBGR);
```

Successivamente si utilizza la sottostante funzione per ottenere attributi che caratterizzano la faccia individuata, come per esempio il numero di facce o la posizione all'interno di in un'immagine:

- x1 Face container upper left x coordinate.
- x2 Face container lower right x coordinate.
- y1 Face container upper left y coordinate.
- y2 Face container lower right y coordinate.
- Detection faces number.
- Coordinate del naso, della bocca degli occhi

```
FacialData=ENCARAFaceDetector->GetFacialData();
numFaces=FacialData->NumFaces;
```

Infine, si pilota la camera inviando il comando nel seguente modo:

```
\\prepara il buffer della porta per inviare comandi

CommandPacket* pPacket=
           static_cast<CommandPacket*>
         (_pOBox_->get(ROBOTCOMMANDS));


pPacket->setPTZPosition(
  RADIANS(panAngle),//Pan
  RADIANS(tiltAngle),// tilt
   -1 // zoom
);
_pOBox_->send(ROBOTCOMMANDS)
```

Si prepara il buffer della porta di uscita, in modo da memorizzare in esso i comandi ed inviarli successivamente.

4.1.1 Inizializzazione

Nello stato di "waitCameraImage" (riferire al capitolo 5 per gli stati della componente che implementa il tracking) prima di poter elaborare le immagini era stato necessario inizializzare la camera. Questo perché la camera al lancio dell'applicazione, restituiva valori del pan e del tilt completamente errati. Risultava impossibile comandarla con i giusti valori. Ad esempio se la camera era posizionata al centro avrebbe dovuto restituire un valore del pan pari a zero, mentre ne restituiva uno diverso.

L'algoritmo di inizializzazione prevede semplicemente, l'invio di comandi che spostano il pan da un estremo all'altro, ossia da -70 a +70, per poi ricadere in 0 (lo stesso per il tilt, da -30 a +25 e successivamente in 0). Dopo l'invio di ogni comando il programma va in sleep per qualche millisecondo, affinché la camera possa eseguire correttamente la richiesta. Si evitano cosí accumuli di comandi pendenti nelle code di attesa. Poteva capitare che la camera alla ricezione di un comando non faceva in tempo ad eseguire il precedente. I comandi si accumulavano ed interferivano tra loro direzionanando il pan/tilt in modo sbagliato.

Si ricorda che 70 e -70 sono i gradi massimi di rotazione del pan, 25 e -30 del tilt.

4.2 Legs-detection

In Robotica mobile 2D i raggi laser sono spesso utilizzati per la localizzazione del robot all'interno dell'ambiente e per evitare gli ostacoli presenti sul cammino dello stesso. Un laser montato ad altezza delle gambe di una persona può essere utilizzato per il riconoscimento di quest'ultima. Nella figura 4.2 un esempio di scenario in cui sono presenti persone ed ostacoli:

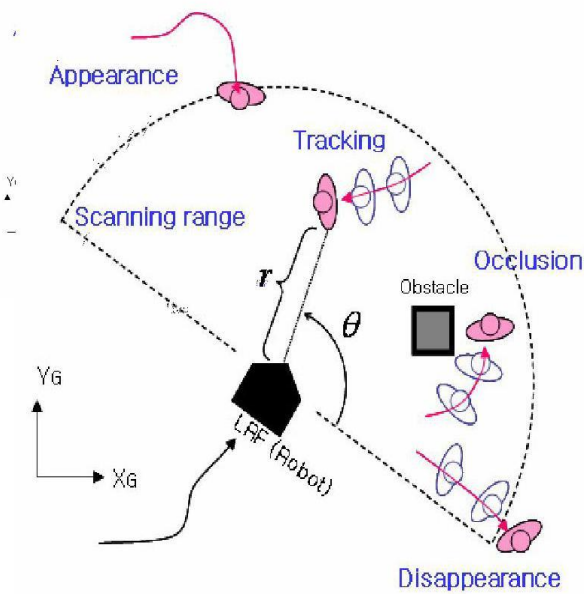

Figura 4.2: People tracking.

La rilevazione delle gambe mediante scanner laser gia era stata considerata per sistemi mobili. Per ogni oggetto individuato dal sensore, sono estratte caratteristiche come il diametro, la forma e la distanza. Per determinare quale degli oggetti sono coppie di gambe possono essere utilizzate le tecniche fuzzy o dei minimi locali.

L'approccio che si considera nell'algoritmo descritto nel seguito, si basa sul fatto che una persona viene rilevata mediante due segmenti posti entro una determinata distanza.

Un segmento in uno scanner laser si compone di una lettura di punti consecutivi con valori simili di distanza, che in generale è il risultato di una superficie liscia di un solo oggetto presente nell'ambiente, sottoposto a rilevazione dei sensori. Grandi differenze di valori di distanza sono dovute ad occlusioni o bordi. Pertanto una singola gamba umana è osservata come unico segmento (ogni gamba è un segmento di punti vicini rispetto a valori di distanza piccoli).

Prima di iniziare la descrizione dell'algoritmo occorre introdurre le seguenti osservazioni. Il sensore laser copre una zona di 180 gradi (con angolo di risoluzione pari a 0.5), possiede un raggio di scansione pari ad 8 metri e restituisce un vettore di tutti

i punti inclusi in quella zona. Tali punti sono rappresentati in coordinate nel sistema di riferimento del laser, il quale è un sistema a due dimensioni (x, y).

La rilevazione delle gambe consiste nell'utilizzare alcune tecniche geometriche e statistiche applicate ai punti ottenuti, in modo da poter caratterizzare quelli vicini.

La detection di coppie di gambe si compone di 3 passi fondamentali: la segmentazione, la classificazione ed il raggruppamento.

- **Segmentazione**: Nel primo passo i punti ottenuti mediante scansione laser vengono divisi in segmenti. Ogni segmento consta di un numero massimo di lettura di punti, dove le differenze dei valori delle distanze tra due punti consecutivi si trovano al di sotto di una certa soglia che risulta essere determinata a 75 mm.
- **Classificazione**: Nel seguente passo ogni segmento viene classificato come "gamba" o "non gamba" sulla base delle seguenti caratteristiche:
  1. numero dei punti di lettura (n),
  2. media delle distanze (μ),
  3. deviazione standard delle distanze (σ),
  4. "width" in coordinate del mondo in direzione perpendicolare al raggio laser (w),
  5. distanze dai segmenti adiacenti (D1 e D2).

  Per caratterizzare un segmento come gamba si sono ottenuti risultati soddisfacenti mediante le seguenti condizioni:

$$(n > 4) \wedge (\mu < 3000mm) \wedge (\sigma < 40mm)$$

$$\wedge(50mm < w < 250mm) \wedge (max(d1, d2) > 250mm)$$

$$\wedge(min(d1, d2) > -50mm).$$

- **Raggruppamento**: Nel passo finale le singole gambe vengono raggruppate in coppie a seconda della distanza. Affinché due segmenti adiacenti possano classificare una persona, la distanza tra essi deve essere inferiore a 500 mm.

4.3 Implementazione dell'algoritmo

In questa sezione verrà trattato l'algoritmo che implementa quello che è stato descritto nella sezione 4.2.

4.3.1 Segmentazione

Per l'implementazione dell'algoritmo si è scelto di usare un approccio basato sulla programmazione orientata agli oggetti, ed è per questo che è stata creata una classe **segment.h**, la quale include le caratteristiche di un segmento. Un oggetto segment presenta i seguenti attributi:

- Tutti i punti di un segmento sono memorizzati in una struttura dati di tipo lista di coordinates2D ($list < Coordinates2D > allPoints$);
- una lista di distanze tra i punti. Quest'ultima è stata necessaria per il calcolo di μ e σ ;
- metodi set e get per leggere gli attributi che caratterizzano la classe, i quali sono:
  - Identification;
  - μ;
  - σ;
  - Width;
  - D1 e D2 che rappresentano le distanze del segmento con quelli adiacenti;
  - max (D1, D2), min (D1, D2);
  - Distanza euclidea tra i punti (per vedere se questi fanno parte di un segmento).

  Tali metodi, insieme ai vari metodi che gestiscono le liste (come ad esempio , costruttori, aggiungere/eliminare un segmento, visitare e cosi via..) si sono implementati nella classe **segment.cpp**.

Per maggiore chiarezza si riporta la dimostrazione geometrica del calcolo del "width". Graficamente ci si trova nella situazione rappresentata nella figura 4.3:

Nella figura 4.3 è rappresentato in rosso un segmento qualsiasi, considerando il sistema di riferimento del laser, ed in verde il width.

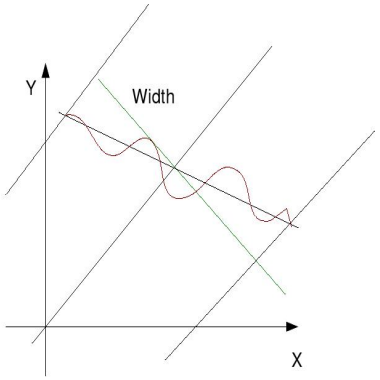

Figura 4.3: La figura rappresenta la larghezza del segmento in direzione perpendicolare al raggio laser.

I passi per calcolare la larghezza (width) sono i seguenti:

1. Si è calcolata la retta passante tra gli estremi del segmento:
   $y = mx + q$
2. Si è calcolato il punto medio della retta ottenendo (xMedio, yMedio):
   $xMedio = \frac{(xA + xB)}{2}$;
   $yMedio = \frac{(yA + yB)}{2}$.
3. Si è calcolata la retta passante tra il punto medio e l'origine del sistema di riferimento del laser:
   $y = mOrigin * x$;
   $mOrigin = \frac{yMedio}{xMedio}$.
   Dove $mOrigin$ rappresenta il coefficiente angolare della retta passante per l'origine.
4. Con il punto precedente si è ottenuto un fascio di rette parallele $y-y1 = m(x-x1)$ e si è tenuto in considerazione quelle che passano negli estremi del segmento. Si considera il termine noto "q":
   $qParallel = -mOrigin * xA + yA$;
5. Successivamente si calcola la retta perpendicolare $y - y1 = \frac{-1}{m(x - x1)}$ al raggio laser, ossia la retta perpendicolare alla retta che passa per l'origine:
   $mPerpendicolar = -\frac{1}{mOrigin}$;
   $qPerpendicolar = -mPerpendicolar * (xMedio + yMedio)$.

6. Infine si è calcolato mediante sistema, i punti di intersezione della retta perpendicolare con le rette parallele agli estremi ottenendo cosi la misura della distanza. Tale distanza rappresenta la larghezza (w):

   $x = \frac{(qPerpendicolar - qParallel)}{(mOrigin - mPerpendicolar)}$;

   $y = mOrigin * x + qParallel.$

   $halfDistance = sqrt((x - xMedio)^2 + (y - halfPointY)^2)$;

   $width = 2 * halfDistance.$

4.3.2 PseudoCodice segmentazione

Come già detto i punti ottenuti dalla scansione laser sono memorizzati in un array chiamato "pScan". L'approccio per ottenere i segmenti è il seguente:

Si esplora tutto l'array calcolando la distanza tra un punto ed il suo adiacente, se tale distanza è inferiore a 75 mm, allora i punti appartengono allo stesso segmento. Si riempe la lista di punti fino a che la differenza tra i punti consecutivi diventa maggiore di 75 mm.

Una volta che il segmento è stato identificato si settano alcuni campi della classe segment.h, come il numero di punti per segmento, l'identificatore e si aggiunge il nuovo oggetto "segmento" in una lista che conterrà tutti i segmenti identificati. Nell'algoritmo 5 viene riportato lo pseudo codice: una semplice scansione con complessità lineare che riempe una lista di segmenti:

```
for all i such that 0 ≤ i ≤ allScannedPoints do
  compute the euclidean distance between adiacentes points
  if distance < 75mm then
    Add the point to the segment
  else
    Add the point to the new segment
  end if
end for
```

Algorithm 5: Segmentation algorithm (Input: pScan, Output: List of Segments)

4.3.3 Classificazione

Una volta ottenuta la lista di tutti i segmenti, si visita tale lista per identificare presunte gambe e scartare le non gambe (si vedrà nella prossima sezione che tali segmenti scartati rappresentano ostacoli). Le possibili gambe detectate verranno salvate in un altra lista, e successivamente si verificherà se rappresentano coppie di gambe.

Una gamba è individuata se è verificata la condizione scritta nella sezione 4.2.

4.3.4 PseudoCodice classificazione

Si scorre la lista dei segmenti e per ognuno di essi si calcolano i parametri caratteristici, verificando se soddisfano la condizione di sezione 4.2.

```
for from first element of the segments list to last element do
    set the distance, width, mean, sigma and distance to adiacentes segments
    if Leg condition is verificated then
        Add leg segment to list of legs
    else
        Compute the distance of the nearest obstacle
    end if
end for
```

Algorithm 6: Classification algorithm (Input: List of Segments, Output: List of legs)

4.3.5 Raggruppamento

Quando si è ottenuta la lista delle presunte gambe, si utilizza un algoritmo avente complessità quadratica per raggruppare le gambe che si presuppongono appartenere ad una sola persona. Si considerano le coordinate del mondo, ed in tali coordinate si crea una coppia, se due gambe sono ad una certa distanza tra loro (ogni coppia classificata rappresenta una persona rilevata).

4.3.6 PseudoCodice Raggruppamento

Riferire alla sezione 4.4 per maggiori dettagli sull'algoritmo che include aspetti legati alla sincronizzazione.

```
for from first element of the legs list to last do
  Compare each leg with each other;
  if Pair condition is verificated then
    (one pair of legs is classified)

    if master is not found  then
      compute the distance of nearest person
      compute teta angle of nearest person
    else
      compute teta angle master
    end if
  end if
end for
```

Algorithm 7: Legs Pairs algorithm (Input: List of legs Output: Pairs leg classified)

4.4 Sincronizzazione e implementazione strategia di tracciamento

Per sincronizzare l'elaborazione dell'immagine con l'elaborazione del laser si sono dovute usare parecchie variabili booleane. Le più significative sono le seguenti:

- *Start Process Image*: controlla l'elaborazione dell'immagine ricevuta dalla camera,
- *Start Encara*: attiva la face detection,
- *Start Base Motion*: controlla il movimento della piattaforma,
- *Pairs Detected*: viene settato a "true" qualora coppie di gambe vengano individuate,
- *Start Tracking Camera*: attiva il tracciamento del volto,
- *Master*: settato a "true" identifica la rilevazione di una persona come master.

Tali variabili quindi, permettono la sincronizzazione delle tre modalità di funzionamento spiegate nel capitolo 5: gli algoritmi 8 e 9 descrivono il modo "Find-Master", mentre l' algoritmo 10 descrive il modo "Tracking-Master" ed il modo "Face-Test".

L'algoritmo 7 restituisce i seguenti due parametri (appartengono ad un master, se il flag *Master* si trova a "true"; altrimenti appartengono ad una persona che deve essere ancora classificata come Master):

1. l'angolo teta;
2. la distanza delle gambe dalla piattaforma.

Tali parametri caratterizzano, rispettivamente, l'orientazione e la distanza della persona più vicina rispetto alla piattaforma mobile. Una volta rilevate gambe, si verifica la presenza della persona spostando il pan della camera nella direzione "teta". Questo avviene in fase iniziale quando tutti i flags sono posti a "false".

L'algoritmo 8 effettua il posizionamento del pan/tilt e sblocca la parte relativa all'elaborazione delle immagini (settando i flags *Start Process Image* e *Start Encara*) per confermare la presenza di un volto. Mentre la camera ruota si è scelto di bloccare l'elaborazione dell'immagine e del tracking della faccia, in modo da evitare accumuli di comandi che la stessa non riesce ad eseguire.

Se la camera conferma la presenza di una persona l'algoritmo la identifica come master ed il robot ruota in direzione della stessa. Mentre ruota viene bloccato Encara2 (settando a false *Start Encara*). Una volta che il robot è allineato frontalmente con la persona viene settato il flag *Master*; si blocca la fase iniziale e si passa alla fase di tracking. A questo punto un master è agganciato e si procede con il tracciamento.

Per quanto riguarda il movimento della camera, come spiegato nella sezione 4.1, è stato introdotto un fattore di ritardo che le permette di eseguire tutti i comandi ricevuti e di evitare problemi di sovrascrittura di comandi pendenti non eseguiti correttamente. La formula del ritardo utilizzata è la stessa usata per la fase di tracking del volto.

```
if master is not found then

  if image processing is blocked then
    compute tilt angle of the nearest person at differents higths
    set panAngle to tetaAngle
    send pan and tilt command
    compute delay, wait for execution command
    set flags: Start process image, start Encara start base motion
  end if
end if
```

Algorithm 8: First step sincronization algorithm

Il modulo 8 permette di ruotare inizialmente la camera in direzione dei segmenti più vicini al robot, classificati come gambe, e di sbloccare, attraverso i flags nel codice, l'elaborazione dell'immagine ed il movimento della base qualora venga confermata la presenza di un volto. Infine la base si muoverà se, oltre al volto saranno rilevati gli occhi.

```
if The base motion is not blocked then

  if Face with eyes are detected then
    block the image process and base motion
    move the platform to the correct direction
    Master = true
  else
    return to the legs detection
  end if
else
  return to the legs detection
end if
```

Algorithm 9: Second step sincronization algorithm

L'algoritmo 9 permette la rotazione della base finchè quest'ultima non si allinea con la persona. Settando "Master=true" il controllo passa dalla modalità "find-master" alla modalità "tracking-master" (algoritmo 10). Contemporaneamente viene calcolata la velocità di rotazione e la velocità di traslazione, in base all'angolo ed alla distanza restituiti dal "matching" (riferire al capitolo 5).

Per effettuare il tracking si verifica se sono presenti gambe (tramite il flag *Pairs Detected*) e se esse appartengono al master. Successivamente si sbloccano i flags relativi

al processamento delle immagini, al tracking effettuato dalla camera (*Start tracking Camera*) e si blocca tutto il resto, in modo tale che l'algoritmo funzioni solamente in modalità "tracking-master".

Dalla modalità "tracking-master" si passa al modo di lavoro "face-test" verificando la presenza di facce. Se "numFaces==0" (o nn ci sono facce o la persona è di spalle) viene attivato il timer (con la funzione clock()): se per circa 10 secondi il volto non viene più rilevato si pone "Master=false" e si torna alla modalità "find-master". In questo modo la camera si posiziona in corrispondenza del nuovo master ed il ciclo si ripete.

if Legs pairs and master legs are detected **then**
  start image processing and start face tracking
  compute master's rotation and tranlation velocity
  start clock for face test module

  if clock expires **then**
    set master=false (master is lost), block face tracking and image processing
    return to the find master module
  else
    continue master tracking
  end if
end if

Algorithm 10: Third step sincronization algorithm

Sistemi di controllo

Per quanto riguarda lo spostamento della piattaforma, si sono utilizzati due sistemi di controllo: uno per il calcolo della velocità di rotazione ed uno per il calcolo della velocità di traslazione. È stato necessario introdurre tali sitemi in quanto la base può essere comandata solo per velocità e non per posizione, e siccome la velocità minima alla quale la base possa muoversi è sempre maggiore della velocità minima di rotazione della camera (che si comanda per posizione e non per velocità) per la quale possa effettuarsi un corretto e fluido tracking della faccia, per far allineare base e camera (che ruotano in contrafase) si è dovuto rallentare il movimento della base con le seguenti formule:

- **Velocità di rotazione**:

$$vRot = \frac{Teta}{(\pi/2) * VROT\_MAX * Kp\_rot};$$

```
if ((fabs(vRot)<RADIANS(VROT_MIN))
          && (fabs(Teta)>RADIANS(DEAD_BAND)) )
 vRot=RADIANS(fabs(vRot)/vRot*VROT_MIN);
```

La velocità di rotazione del robot (calcolata in radianti, in quanto player-robot riceve comandi in radianti) è uguale all'angolo teta (calcolato in funzione della posizione delle gambe rilevate) diviso la velocità massima di rotazione (definita a un certo valore) per una costante di rotazione, che viene regolata affinché non si abbia una velocità di rotazione troppo aggressiva quando l'angolo teta risulta essere troppo grande. Tale velocità dipendende dall'angolo: a seconda del suo valore, la base ruota più o meno rapida (valore grande movimento rapido, valore piccolo movimento lento). Con questo sistema è possibile far allineare contemporaneamente base e camera: "giocando" con il parametro "Kp" è possibile rallentare la rotazione della piattaforma quanto si vuole. Il controllo successivo serve a limitare la velocità di rotazione qualora l'angolo teta risulti troppo grande.

- **Velocità di traslazione**:

$$vTrans = \frac{-(dMax - Distance)}{2} * (VTRANS\_MAX * Kp\_trans)$$

```
if ((fabs(dMax-Distance)>DEAD_BANDTRANS-Distance ))
       vTrans=((-(dMax-Distance))/2.0)
       *(VTRANS_MAX*Kp_trans);
else
    vTrans=0.0;
```

Analogamente, la velocità di traslazione viene limitata quando si ha una distanza troppo grande, in quanto tale velocità dipende dalla distanza della persona dalla piattaforma. Anche qui tramite la costante "Kp" è possibile rendere il robot più rapido o più lento. Da osservare il segno meno che serve a mandare indietro il robot qualora una persona gli vada incontro: Il robot va all'indietro quando una persona gli va incontro, viceversa il robot segue la persona quando questa si allontana da lui. Nella formula viene considerata una distanza massima (dMax) ed una velocità massima (VTRANSMAX). Si è scelto di non far avanzare il robot quando la distanza delle gambe sono all'interno di mezzo metro (vTrans=0.0).

Infine per evitare la presenza degli ostacoli, si considera una non-gamba come ostacolo, successivamente si verifica se tale ostacolo si trova ad una distanza inferiore a due metri e se cosi fosse il robot rallenta avvisando la presenza di pericolo. Per non avere brusche decellerazioni si scala la velocità in maniera "dolce":

```
if(maxDistObs>=2.0)VTRANS_MAX=1.5;
else {  \\maxDistObs rappresenta il segmento piu' vicino al robot
  if(maxDistObs>1.0 && maxDistObs<2.0)
    VTRANS_MAX=1.0;else VTRANS_MAX=0.5;
    }
```

Come si vede dal frammento di codice la velocità diminuisce gradualmente passando per valori intermedi a seconda della distanza dell'ostacolo.

Risultati e Limiti

Le figure 4.4 e 4.5 mostrano esempi di esplorazione dell'ambiente effettuata con laser e considerando una persona in piedi davanti al robot (risultato dell'algoritmo descritto). La presenza di una sedia genera falsi positivi. Oltre alle gambe della persona vengono rilevate le gambe delle sedia.

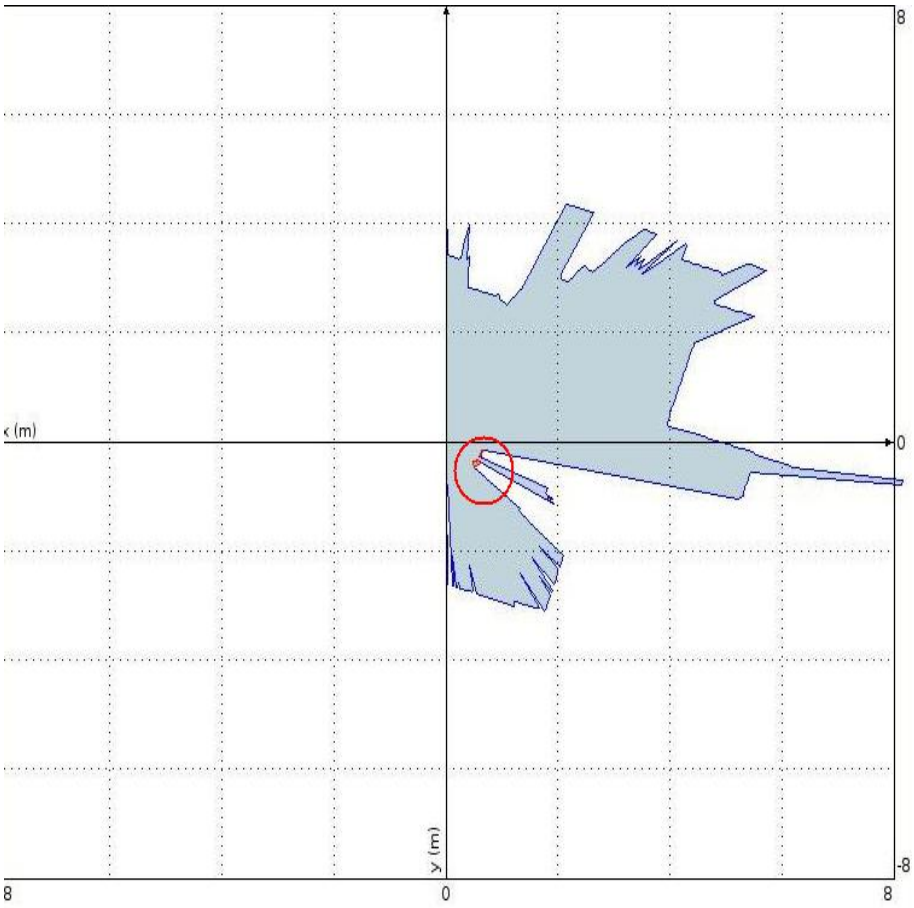

Figura 4.4: La figura rappresenta l'ambiente rilevato dal laser.I segmenti in rosso rappresentano la presenza di una persona rilevata correttamente.

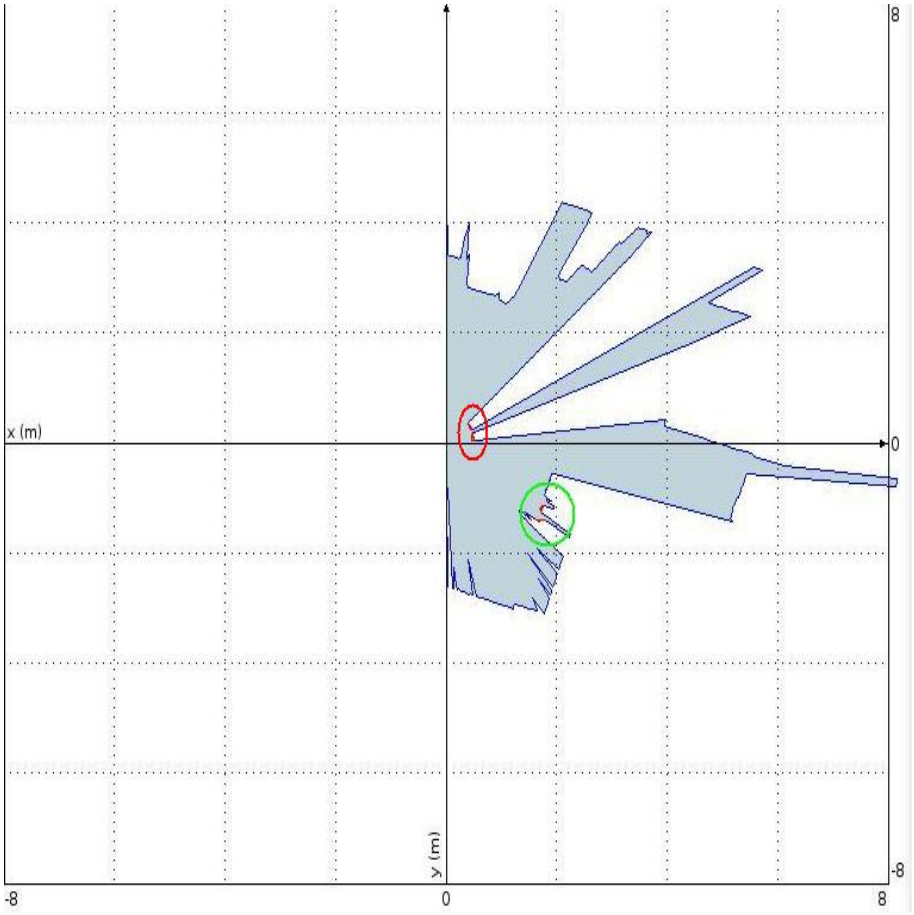

Figura 4.5: La figura rappresenta la rilevazione di una persona e di un falso positivo (cerchiato in verde): un sedia con quattro gambe viene riconosciuta come persona.

Capitolo 5

Aspetti implementativi, architettura hardware e software

5.1 Architettura Hardware: Piattaforma Pioneer (P3-DX)

La PIONEER 3-DX è una piattaforma robotica mobile autonoma, particolarmente versatile e duttile. Costruita sul modello client-server, come tutte le piattaforme robotiche mobili, la P3-DX offre un sistema embedded, che apre la strada all'elaborazione visiva "on-board", basandosi su comunicazione Ethernet, laser, DGPS, e altre funzioni autonome. la P3-DX memorizza fino a 252 watt / ora di batterie hot-swap. Essa ha un anello di 8 sensori sonar in avanti e un anello (opzionale) posteriore (vedere la figura 5.1). La potenza dei motori del 3-DX e le ruote da 19 centimetri permettono ad essa di raggiungere velocità di 1,6 metri al secondo e di trasportare un carico utile fino a 23 kg.

Al fine di mantenere accurato il calcolo dei dati a queste velocità, Pioneer utilizza 500 codificatori ticks. La navigazione avviene utilizzando metodi basati su scanner laser, bumpers di contatto, visione stereo ed altre opzioni che sono in rapida crescita.

5.1.1 Componenti P3-DX

La piattaforma Pioneer 3-DX (figura 5.2) è dotata di un insieme di sensori che le permettono la navigazione in un ambiente reale. Sono incluse batterie, motori di trazione, ruote, codificatori di posizione e velocità. Tali componenti sono controllati per mezzo di un micro controllore incorporato e di un software "client-server", che controlla le azioni del robot in modo da mantenere la velocità e la direzione del movimento per un terreno irregolare o acquisire le letture dai sensori sonar e laser; i quali coprono una zona di 180 gradi permettendo la detection di oggetti davanti ed ai lati. Il raggio laser si trova ad altezza delle gambe, e di conseguenza, gambe umane possono essere rilevate.

Pioneer 3-DX fornisce una piattaforma con le seguenti caratteristiche:

- una porta d'accesso per la presa della batteria;
- una batteria;
- 2 ruote e una ruota di appoggio (caster);
- motori con encoders;
- anello frontale di sonar;
- micro-controller;
- sonar board;
- motor power board;
- ARCOS microcontroller server software;
- user I/O bus integrated into hardware and ARIA software;
- ARIA Robotics API for developers;
- operations manual

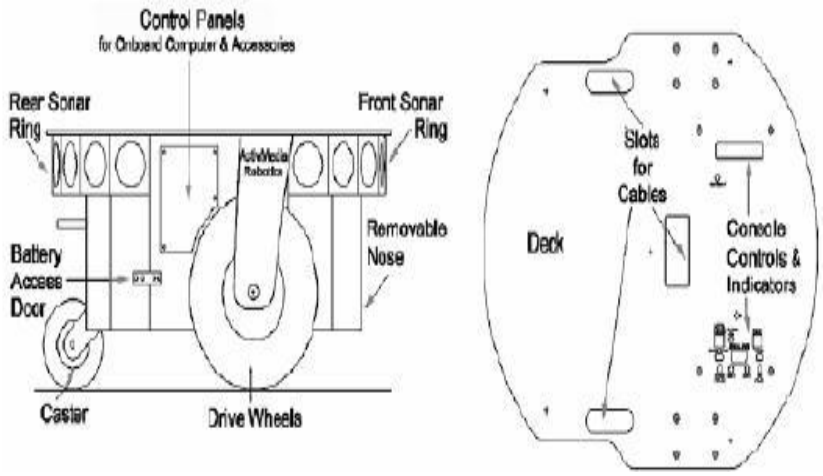

Figura 5.1: La figura rappresenta le componenti della piattaforma mobile.

P3-DX è 44 x 38 x 22 centimetri con corpo in alluminio e 16,5 centimetri di diametro ruote motrici. I due motori usano gear rations e contengono tick encoder. Questo permette alla piattaforma di ruotare entrambe le ruote in movimento od oscillare intorno ad una ruota stazionaria in un cerchio di 32 centimetri di raggio. Il caster posteriore permette il bilanciamento del robot.

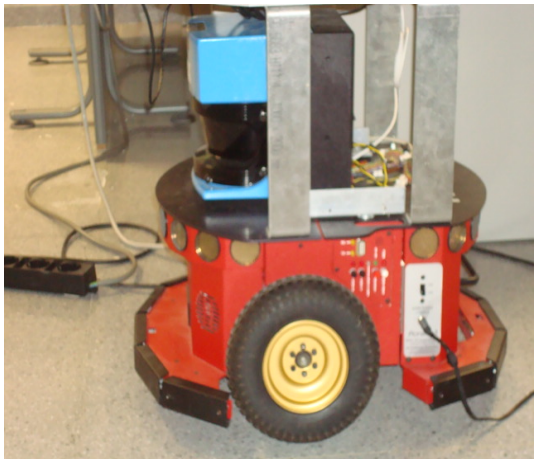

Figura 5.2: La figura rappresenta la vista laterale della piattaforma mobile.

L'architettura software si completa con due client che si eseguono in due computers (uno a bordo e uno nel portatile) connessi alla piattaforma per una connessione seriale (pc on board) e una w-lan (portatile), permettendo un controllo intelligente ad alto livello come l'individuazione degli ostacoli, la pianificazione di traiettorie o la detection ed il tracking. (I sistemi operativi si basano su versioni di linux: nel portatile è installato la versione 2.6.24-23-generic di kubuntu.)

Nel server della piattaforma si esegue un'istanza di player-robot che permette di acquisire dati dai sensori, visualizzati nel portatile dalle interfacce grafiche create con gtk. Inoltre alla base sono connessi un monitor, un mouse ed una tastiera per inserire comandi da shall.

Nel portatile si esegue un'altra istanza di payer-robot che permette di acquisire immagini provenienti dalla camera e lanciare l'eseguibile che pilota il robot e che effettua la detection e l'inseguimento della persona.

Specifiche Video-Camera

L'architettura include una video camera "*Logitech QuickCam sphere*" di 1.3 MegaPixel (figura 5.3) e che possiede le seguenti caratteristiche:

- VGA CCD sensor,
- Mechanical pan e tilt: dove il pan ruota su piano orizzontale da -70 gradi (verso sinistra) fino a +70 gradi (verso destra), il tilt ruota su piano verticale (alzandosi e abbassandosi) da -30 gradi fino a +25 gradi.
- Acquisizione di immagini: 1280 x 960 pixels (software interpolated),
- USB 2.0/USB 1.1,
- Acquisizione di video: 640 x 480 pixels,

- Frame Rate: 30 frames per second
- Supporto di 22,9cm.

Figura 5.3: La figura presenta la camera ed il pan/tilt.

In conclusione come si vede nelle figure 5.4 e 5.5, l'architettura hardware prevede una piattaforma con una camera ed un pan/tilt in cima di un payload:

Vista frontale:

Figura 5.4: La figura rappresenta la vista frontale dell'architettura Hardware.

Vista Laterale:

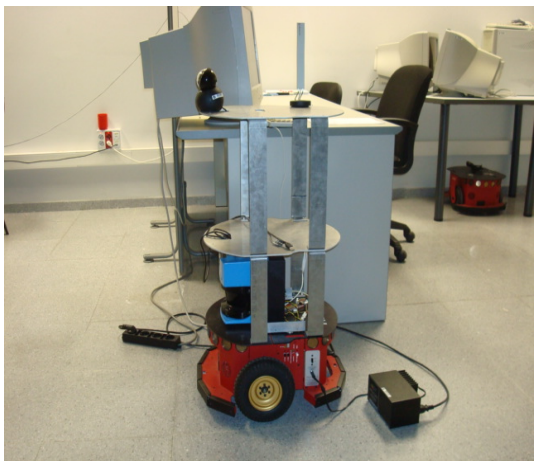

Figura 5.5: La figura rappresenta la vista laterale dell'architettura Hardware.

5.2 Architettura software

In questa sezione si descrive la struttura dell'applicazione, le classi e tutto ciò che è stato necessario per lo sviluppo.

5.2.1 Componenti software

Libreria OpenCV

La libreria OpenCV è una collezione di funzioni che permette di lavorare ad alto rendimento sopra le immagini. OpenCV implementa una ampia varietà di strumenti per l'interpretazioni delle immagini.

OpenCV è principalmente una libreria ad alto livello che implementa algoritmi e tecniche di calibrazione, rilevamento di caratteristiche, tracking, analisi di forme, analisi di movimento, ricostruzioni 3D, segmentazione e riconoscimento di oggetti.

Nello sviluppo del progetto oltre alle funzioni utili per il disegno di figure (ad es: rettangoli, cerchi e cosi via) sono state usate le seguenti funzioni:

The function SetImageData sets the pointer to data and step parameters to given values.

- **void cvSetImageData(IplImage* image, void* data, int step)**;

  image Image header.

  data User data.

step Distance between the raster lines in bytes.

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

src The source 8-bit (8u), 16-bit (16u)

or single-precision floating-point (32f) image.

dst The destination image of the same data type as the source one.

The number of channels may be different.

codeColor conversion operation

that can be specifed using CV srcColorSpace2dstColorSpace constants.

The function cvCvtColor converts input image from one color space to another. The function ignores colorModel and channelSeq fields of IplImage header, so the source image color space should be specified correctly (including order of the channels in case of RGB space, e.g. BGR means 24-bit format with B0 G0 R0 B1 G1 R1 ... layout, whereas RGB means 24-bit format with R0 G0 B0 R1 G1 B1 ... layout).

The conventional range for R,G,B channel values is:

* 0..255 for 8-bit images
* 0..65535 for 16-bit images and
* 0..1 for floating-point images.

Of course, in case of linear transformations the range can be arbitrary, but in order to get correct results in case of non-linear transformations, the input image should be scaled if necessary.

Dal momento che ENCARA2 lavora con immagini di tipo "BGR" e le immagini acquisite dalla camera sono in formato "RGB", è stato necessario usare queste due funzioni per la conversione del formato, come spiegato nel capitolo 4.

Gtk: una GUI per il C++

GTK è un toolkit per la creazione di interfacce grafiche, che vanta compatibilità cross-platform e una facile API da usare. GTK è scritto in C e costruito per essere usato anche da molti altri linguaggi di programmazione come il c++, python ecc.. Esso è autorizzato sotto la licenza GNU LGPL 2.1. Tale GUI è stata utilizzata per la creazione di viste in cui è possibile vedere i dati rilevati dai sensori e le immagini acquisite dalla camera.

Player-Robot: un'interfaccia per i dispositivi robotici

Player è un server di rete per il controllo di robot. In esecuzione sul robot, Player offre un'interfaccia semplice e pulita dei sensori e degli attuatori del robot su rete IP. Il programma client comunica con Player attraverso socket TCP, leggendo dati dai sensori, scrivendo comandi agli attuatori e configurando dispositivi "on the fly".

Player supporta una vasta gamma di robot. La piattaforma hardware originale è la **ActivMedia Pioneer 2 family**, ma molti altri robot e molti comuni sensori sono supportati.

L'architettura modulare di Player rende facile aggiungere il supporto per il nuovo hardware, mentre un'attiva comunità di users/developers contribuisce a sviluppare nuovi drivers.

Player permette l'astrazione dell'hardware (Hardware Abstraction Layer: HAL) per dispositivi robotici. Come il sistema operativo (Linux, Mac OS X, ecc.) nasconde i dettagli dell'hardware definendo concetti come "mouse" e "stampante" (ciascuno con la propria interfaccia), così player nasconde i dettagli della piattaforma robotica. Per questo lo si può cosi considerare un vero e proprio sistema operativo per i robot (Robot OS). Esso definisce un'insieme di interfacce standard, ognuna delle quali rappresenta uno dei modi con cui è possibile interagire con alcuni tipi di dispositivi. Ad esempio, l'interfaccia "position2d" consente ai robot mobili di accettare comandi che permettono loro lo spostamento (sia in velocità che per posizione) restituendo lo stato attuale (la velocità e la posizione corrente). Il lavoro del driver è quello di rendere il robot capace di comprendere un'interfaccia standard in modo tale che il codice di controllo scritto per un robot possa funzionare (limitatamente) in un altro robot.

Player, mette a disposizione degli utenti dei meccanismi di trasporto che permettono ai dati di essere intercambiati tra i drivers e i programmi di controllo che sono in esecuzione in macchine distinte. Il metodo di trasporto più comune al giorno d'oggi è basato su socket TCP in un modello client/server.

Anche se la maggior parte dei drivers di player controlla direttamente l'hardware, esistono dei drivers astratti. Ad esempio Un driver astratto usa altri drivers come fonti di dati e come luogo dove inviare comandi. Lo scopo di tali drivers è incapsulare algoritmi utili che potranno riutilizzarsi facilmente.

Player funziona su Linux (e PC embedded), Solaris e * BSD.

Caratteristiche di Player-Robot

Player è stato progettato per essere indipendente dalla piattaforma e dal linguaggio. Il programma client può essere eseguito su qualsiasi macchina che ha una connessione di rete al robot e può essere scritto in qualsiasi linguaggio che supporta socket TCP.

Attualmente lato client, si hanno utility disponibili in C + +, Tcl, Java e Python. Inoltre, Player non fa alcuna ipotesi su come si potrebbe desiderare di strutturare il programma di controllo del robot. In questo modo, è molto più "ristretto" rispetto ad

altre interfacce robot. Permette al programma client la concorrenza, il multi-threading, il controllo interattivo del robot e cosi via. Player consente a più dispositivi di presentare la stessa interfaccia. Ad esempio, il Pioneer 2 e i drivers RWI usano entrambi l'interfaccia di posizione di Player per consentire il controllo del movimento del robot. Cosí lo stesso codice di controllo potrebbe guidare entrambi i tipi di robot. Questa funzione è molto utile quando è combinata con il simulatore **Stage**.

Player è anche progettato per supportare virtualmente un numero qualsiasi di clienti. Ogni cliente può collegarsi e leggere i dati dal sensore (e anche scrivere i comandi a motore) in qualsiasi istanza di Player sul robot. Oltre a sensori distribuiti per il controllo, è inoltre possibile utilizzare Player per il monitoraggio degli esperimenti. Ad esempio, mentre il client C + + controlla un robot, è possibile eseguire uno strumento di visualizzazione grafica che mostra i dati attuali dei sensori e un logger che li salva per una successiva analisi. On-the-fly consente ai clienti di accedere a differenti sensori e attuatori, se necessario.

Player is free software, released under the GNU Public License.

Stage

Stage è un simulatore per le applicazioni robotiche. Fornisce un mondo virtuale per i robot mobili, sensori e vari oggetti che il robot è in grado di rilevare e manipolare. Simula quindi, una popolazione di robot mobili in una bitmap a due dimensioni. Questo programma è normalmente utilizzato insieme con il server Player in modo da avere un mondo simulato per i programmi di controllo: Stage è usato con Player-Robot per fornire popolazioni di dispositivi virtuali. Gli utenti scrivono programmi client che permettono di controllare il robot ed i suoi sensori. Può anche essere usato come libreria di simulazione di automi.

È stato progettato per sostenere la ricerca di sistemi autonomi multi-agente in modo da fornire semplici modelli computazionali a basso costo piuttosto che cercare di emulare dispositivi con grande fedeltà.

Nella figura 5.6 un esempio di ambiente simulato: in rosso il robot che fa lo scanner dell'ambiente per rilevare ostacoli e l'eventuale presenza di altri automi.

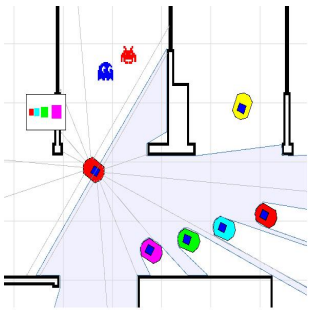

Figura 5.6: La figura rappresenta un mondo simulato.

CoolBOT: una macro di programmazione orientata a componenti

Nel seguito si darà una breve descrizione delle caratteristiche principali di Coolbot.

Generalità

I sistemi robotici sono complessi perché condividono parecchie fonti di problemi che complicano la loro programmazione. Al momento non esistono chiari paradigmi per programmare sistemi robotici, ma ci sono molti approcci basati su soluzioni provenienti dai vari gruppi di ricerca. Questo dimostra come il campo della robotica sia ancora un dominio troppo giovane e recente, dove è costoso e difficile riprodurre i risultati prodotti da tali gruppi.

In generale riscontriamo frequentemente alcuni problemi comuni quando si programmano sistemi robotici. Ad esempio: multithreading e multiprocessing, calcolo distribuito, astrazione dell'hardware, integrazione hardware e software, livelli multipli di astrazione e controllo, sviluppo di strumenti di programmazione e cosi via.

Per lo sviluppo di questo progetto è stato utilizzato CoolBot, il quale è un framework orientato alla programmazione delle componenti, sviluppato in C++, implementa primitive e meccanismi volti a supportare la risoluzione di alcuni dei problemi accennati precedentemente. Questo framework permette di costruire sistemi robotici in termini di composizioni ed integrazioni delle componenti software e modelli basati su automi, favorendo controllabilità ed osservabilità del funzionamento della componente.

Ogni componente software è un'unità di esecuzione indipendente che fornisce una data funzionalità, nascosta dietro un'interfaccia esterna, specificando in modo chiaro quale siano i dati in input e in output. Le componenti una volta definite e costruite potrebbero essere instanziate, integrate ed usate quante più volte necessario.

Port Automata Model for Components

In coolBot le componenti sono modellate come automi. Questo concetto stabilisce una chiara distinzione tra funzionalità interne di una entità attiva, un automa, e le sue interfacce esterne (porte di input ed output).

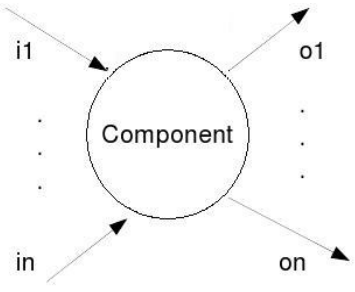

Figura 5.7: La figura rappresenta la vista esterna della componente.

In figura 5.7 è rappresentata la vista esterna di una componente: la componente è rappresentata da un cerchio, le porte di input sono rappresentate da frecce orientate verso il cerchio, mentre le porte di output sono rappresentate da frecce orientate verso l'esterno, ovvero, in uscita dal cerchio.

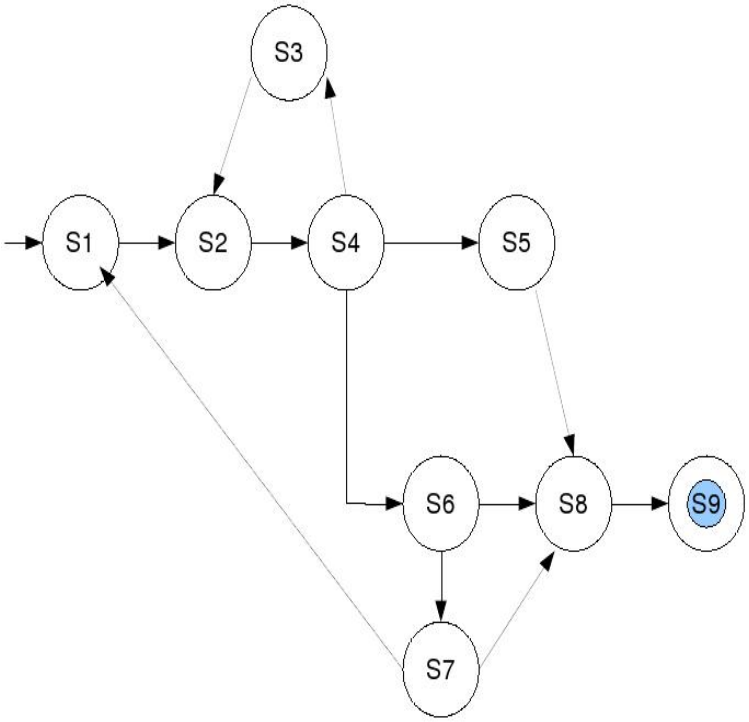

Figura 5.8: La figura rappresenta una possibile vista interna della componente.

In figura 5.8 è rappresentata una possibile vista interna della componente, l'automa. I cerchi rappresentano gli stati dell'automa e le frecce rappresentano le transizioni tra gli stati.

Le transizioni sono attivate dagli eventi (ei), causati dalle informazioni che ricevono le porte di input della componente, dal verificarsi di condizioni interne o da una combinazione di entrambi. Doppi cerchi rappresentano stati finale dell'automa.

Le componenti CoolBot comunicano ed interagiscono tra loro mediante i collegamenti stabiliti tra le porte di ingresso e di uscita. I dati sono trasmessi attraverso le connessioni tra le porte, in unità discrete chiamate pacchetti, i quali sono anche classificati a seconda del tipo; usualmente ogni porta di input e di output può solo accettare specifici tipi di pacchetto.

Osservabilità e Controllabilità

Le componenti dovrebbero essere osservabili per sapere se si sta lavorando correttamente o meno, ed in tal caso dovrebbero essere controllate per fare qualche aggiustamento nel loro interno, disciplinando il loro comportamento e la loro regolare operazione. Per questo CoolBot introduce due tipi di variabili al fine di monitorare e controllare la regolare esecuzione delle componenti:

- **Variabili Osservabili**: Rappresentano caratteristiche interne delle componenti che si interessa osservare e controllare dall'esterno.
- **Variabili Controllabili**: Rappresentano aspetti della componente che possono essere controllati dall'esterno, ad esempio la modifica o l'aggiornamento di valori. Attraverso queste variabili può essere cambiato il comportamento interno di una componente.

Inoltre prevede due porte speciali per garantire l'osservabilità ed il controllo esterno (come si vede nelle figure 5.9 e 5.10):

- **Monitoring Port**: È una porta di output pubblica dove sono publicizzate le variabili osservabili, attraverso questa porta un supervisore può osservare e monitorare le componenti.
- **Control Port** : È una porta di input pubblica attraverso la quale è possibile modificare ed aggiornare le variabili controllabili.

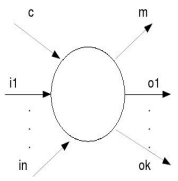

Figura 5.9: La figura rappresenta la vista esterna della componente. La porta di controllo e di monitoraggio sono rappresentate rispettivamente da c ed m.

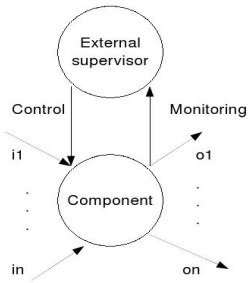

Figura 5.10: La figura rappresenta un tipico loop di controllo della componente.

Automa di default

Tutte le componenti sono modellate usando lo stesso automa di default riportato in figura 5.11, che contiene tutti i possibili percorsi di controllo che una componente potrebbe seguire.

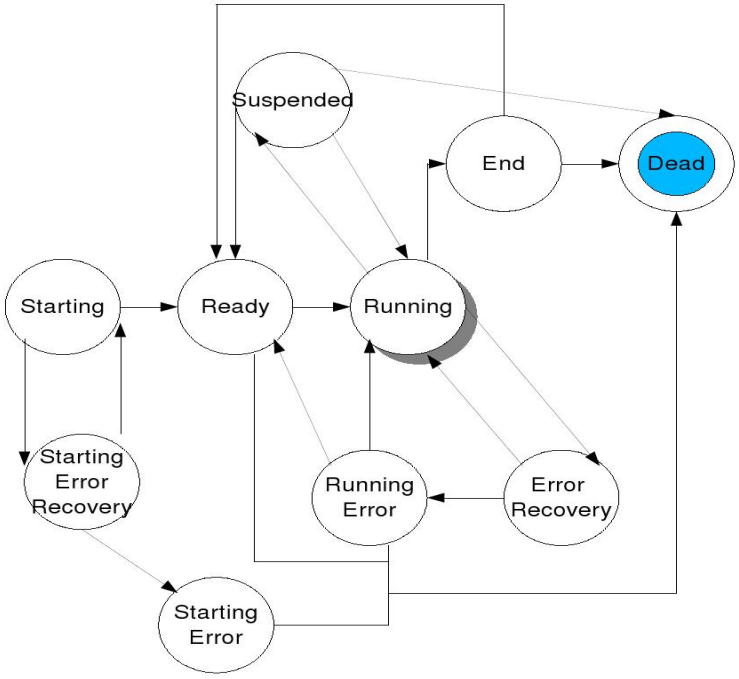

Figura 5.11: La figura rappresenta approssimatamente gli stati dell'automa.

L'automa di default organizza la vita di ogni componente in diverse fasi, alle quali corrispondono differenti stati:

- **Starting**: Alloca risorse per una corretta esecuzione del task;
- **Ready**: In questo stato la componente è pronta per essere eseguita aspettando un comando esterno per partire;
- **Running**: La componente è in esecuzione;
- **Suspended**: La componente sospende l'esecuzione e rimane in idle finché non arriva un comando per transitare ad un altro stato;
- **End**: In questo stato la componente ha terminato l'esecuzione e pubblica il risultato attraverso la porta di monitoring.

Multithreading

Le componenti coolBot sono mappate come threads quando loro sono in esecuzione (Win32 Threads in windows e Posix in gnu/linux che sono gli unici sistemi operativi in cui coolbot è supportato). Le componenti sono eseguite in parallelo o in concomitanza per raggiungere indipendentemente i proprio obbiettivi.

Durante il ciclo di esecuzione ogni componente processa le porte, in attesa di un qualche pacchetto, ed a seconda del pacchetto che riceve si attiva un'opportuna funzione che genera la transizione da uno stato ad un altro. Come in figura 5.12:

```
PortPacket* packet;
Initialization();
while true do
  packet = waitForSomething(input ports);

  if !processPortPacket(packet) then
    break;
  end if
end while
Finalization();
```

Algorithm 11: l'algoritmo rappresenta una componente in esecuzione: un loop continuo che elabora pacchetti nella porta di input.

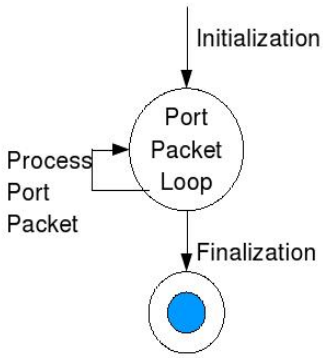

Figura 5.12: La figura rappresenta graficamente l'algoritmo 11.

In generale ogni componente ha bisogno di un main thread.

Inter Component Communications(ICC)

Coolbot prevede un ICC (inter component communicate) che permette alle componenti di comunicare tra loro. Di seguito vengono riportati i vari tipi di porte e le varie combinazioni previste per avere comunicazioni corrette tra esse.

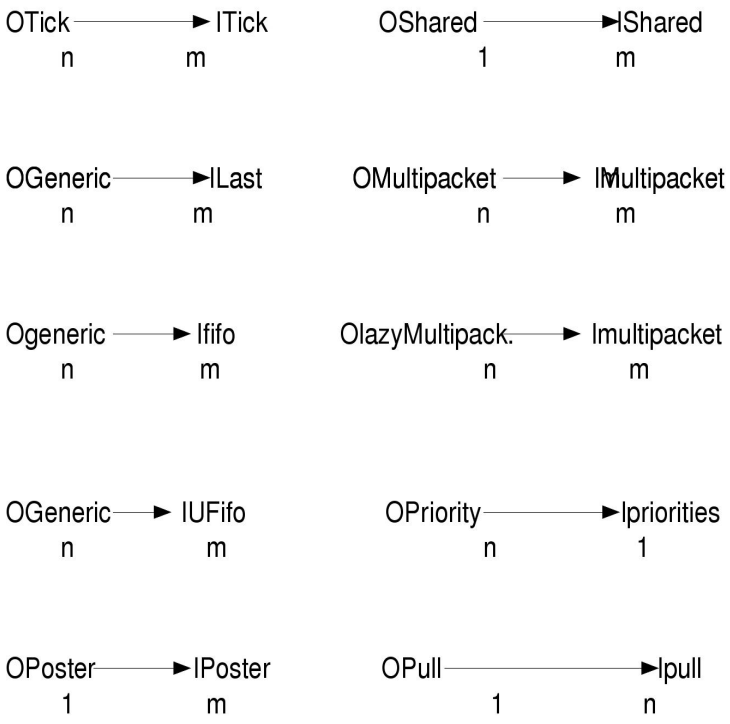

Figura 5.13: Port connections ($\forall n, m \in N; n, m \geq 1$).

Nella tabella 5.1 sono descritte le connessioni rappresentate nella figura 5.13.

Tabella 5.1: Connessioni di porte

Porta di Output	Porta di input	Breve Descrizione
OTick	ITick	*Tick Connection*: Implements a protocol to signal events between components
OGeneric	ILast, IFifo, IUFifo	*Last, Fifo and Unbounded Fifo Connections*: There is a queue of port packets in the input port
OPoster	IPoster	*Poster Connections*: There is a "master copy" of port packets in the output port, input ports keep local copies
OShared	IShared	*Shared Connections*: Components share a "shared memory" residing in the output port. Implements a protocol of shared memory
OMultiPacket, OLazyMultiPacket	IMultiPacket	*Multi Packet Connections*: Accepts multiple types of port packets through the same port connection
OPrioriy	IPriorities	*Priority Connections*: Implements a protocol of sending with priority
OPull	IPull	*Pull Connections*: Implements a protocol of request/answer between components

Developing Components

Il processo di sviluppo delle componenti CoolBot è riassunto in questi passi:

1. **Definizione e sviluppo**: La componente è definita e progettata. In questo passo si conoscono porte di input di output, variabili osservabili e controllabili, transizioni, timer ecc.

2. **Generazione dello Skeleton**: Scheletro delle classi *.cpp *.h (**coolBot -c "nome-componente.coolbot"**).

3. **Code FulFilling**: Si completano le classi generate nel passo precedente.

4. **Generazione libreria**: Compilazione della componente e creazione della libreria.

5. **Integrazione di Sistema**: La componente generata può essere sola o integrata con altre componenti.

6. **Generazione di Sistema**: Generazione di un eseguibile per eseguire test.

In figura 5.14 è rappresentato uno schema per maggior chiarezza:

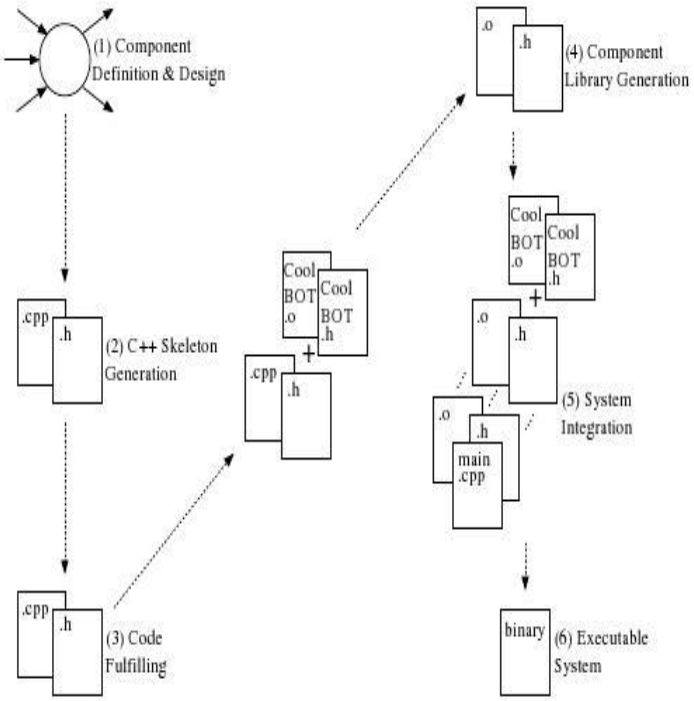

Figura 5.14: La figura rappresenta lo scheletro generato dal compilatore coolbot.

5.2.2 Installazione Componenti

CoolBot come il resto delle componenti, è presente nel server **cvs mozart.dis.ulpgc.es**. Nel seguito si mostrerà come si realizza l'installazione di coolbot e player-robot. Si è creato un direttorio chiamato **cvs-projects** dove è stato installato tutto il software necessario.

5.2.2.1 Coolbot

- **Download**:

  ```
  cvs -d :ext: ‘‘user‘‘@mozart.dis.ulpgc.es:/home/cvsadmin/repository
  ```

  In repository cercare coolbot (checkout coolbot).

- **Variabile di intorno**: Includere nel file **.bashrc** la variabile di stato **COOLBOT_PATH** indicando dove si trova Coolbot e dove si troverà la libreria generata dalla compilazione.

  Introdurre la segeunte stringa:

  ```
  COOLBOT_PATH=$HOME/cvs-projects/coolbot
  LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$COOLBOT_PATH/lib
  export COOLBOT_PATH LD_LIBRARY_PATH
  ```

  Successivamente inizializzare la shell, per aggiornare le variabili di stato.

- **Compilazione**: Nel direttorio dove si è scaricato coolBot, si trovano vari "make file" con estensione .mak necessari per la compilazione.

  Il primo file si denomina makeMakeObjects.mak e si utilizza per realizzare la generazione delle regole di compilazione di coolBot. Si utilizza quando si compila per la prima volta o quando si aggiungono nuovi files. Per eseguirlo si utilizza il seguente ordine di linea:

  make extraFlags="-ggdb -O0" -f makeMakeObjects.mak (extraFlags per effettuare il debug).

  La seguente istruzione invece serve a generare la libreria dinamica di coolbot:

  make -f coolbot-lib.mak

5.2.2.2 Player-Robot

Analagamente player-robot si scaricherà nel direttorio cvs-projects dal server cvs, si compilerà allo stesso modo di coolbot e si genererà la libreria dinamica con la seguente istruzione:

make -f player-robot-lib.mak

Le altre componenti installate ed utili all'utilizzo del simulatore sono:

- **Grid-Map**: una componente che realizza una mappa dell'ambiente;
- **Short-Term-Planner**: una componente che rappresenta un pianificatore di traiettorie;
- **Nd-navigation**: una componente che implementa l'"obstacles avoidance", permette di evitare ostacoli;
- **Encara2**: la componente encara che realizza la face-detection;
- **Encara-Tracking**: la componente che realizza il rilevamento ed il tracciamento di persona.

Il procedimento di installazione è analago.

5.2.3 Rete delle componenti

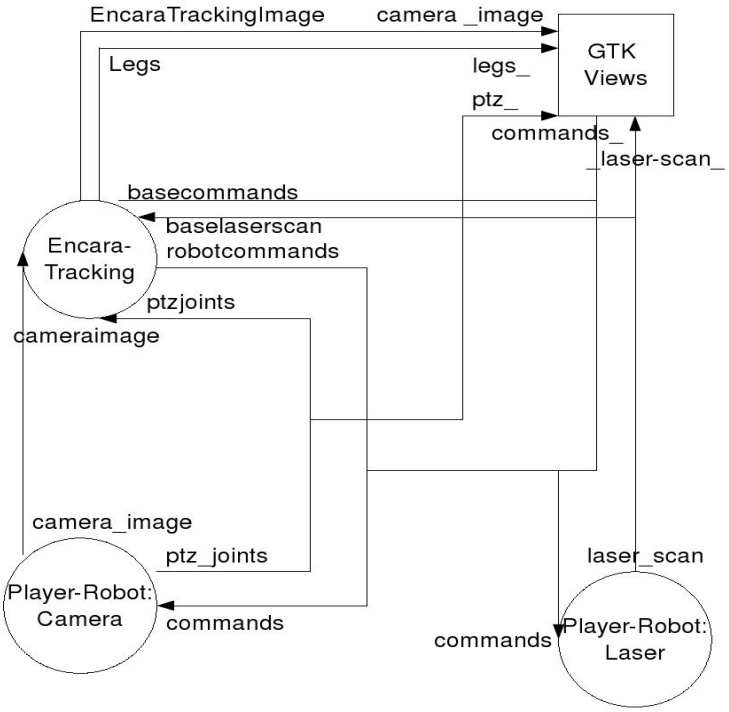

Figura 5.15: La figura rappresenta la rete delle componenti realizzate per il progetto: si definiscono i collegamenti tra le componenti utilizzate.

Come si può vedere dalla figura 5.15, l'applicazione prevede quattro componenti:

1. **Encara-Tracking**: Rappresenta la componente principale relativa alla logica di funzionamento del robot.

2. **GTK views**: Rappresenta la componente per la supervisione con cui è possibile visualizzare i dati acquisiti dai sensori e verificare visivamente il corretto funzionamento degli algoritmi di tracking e di legs detection.

3. Due instanze di player-robot che lavorano indipendentemente e contemporaneamente:

   - **Camera-instance**: necessaria per il controllo della camera e per l'acquisizione di immagini;
   - **Laser-instance**: necessaria per l'acquisizione dei dati rilevati dal sensore e per il controllo della base mobile.

Osserviamo successivamente che la componente "Encara-tracking" presenta le seguenti porte d'entrata e d'uscita:

Porte di input:

- **PTZJOINTS**: È una porta di tipo *last*, necessaria principalmente a ricevere informazioni sulla posizione attuale del tilt e del pan. Tali informazioni vengono inviate dalla componente **Camera-instance**, la quale invia informazioni sulla sua porta d'uscita *PTZ_JOINTS*.
- **CAMERAIMAGE**: È una porta di tipo *poster*, necessaria alla ricezione delle immagini inviate dalla componente **Camera-instance** per mezzo della sua porta d'uscita *CAMERA_IMAGE*.
- **BASELASERSCAN**: È una porta di tipo *last*, necessaria alla ricezione dei dati acquisiti mediante laser. Tali informazioni vengono inviate dalla componente **Laser-instance** attraverso la sua porta d'uscita *LASER_SCAN*.

  Altre porte come **BASELASERODOMETRY**, **BASELASERCONFIG** e **LASERGEOMETRY** sono state omesse per semplicità del disegno, in quanto esse danno informazioni solo sulla posizione del robot nell'ambiente, sulla configurazione dello stesso ed informazioni riguardanti la carica residua della batteria.

Porte di output:

- **ROBOTCOMMANDS**: È una porta di tipo *generic*, necessaria per l'invio dei comandi della camera (sposta di un certo angolo il pan ed il tilt). Tali comandi verranno ricevuti dalla componente **Camera-instance** attraverso la sua porta *COMMANDS*.
- **BASECOMMANDS**: È una porta di tipo *generic*, necessaria per l'invio dei comandi di movimento della base mobile (comanda la base in modo che possa traslare e ruotare con una determinata velocità). Tali comandi sono ricevuti dalla componente **Laser-instance** attraverso la porta *COMMANDS*.
- **ENCARATRACKINGIMAGE**: È una porta di tipo *poster*, necessaria all'invio delle immagini processate dalla componente **Encara2**, al supervisore GTK in modo da visualizzare l'eventuale faccia riconosciuta.
- **LEGS**: È una porta di tipo *generic*, necessaria per l'invio delle immagini segmentate che illustrano le regioni delle eventuali gambe riconosciute, in modo tale da verificare visivamente la correttezza del risultato.

La componente GTK views non fa altro che ricevere dati, direttamente dai sensori e dalla componente encara-tracking (a seguito di elaborazione), e visualizzarli nelle rispettive interfacce grafiche.

La scelta del tipo delle porte dipende dal tipo già determinato in player-robot. Per non incorrere in errori di "miss-match", dovuti ad incoerenze di tipo, si sono rispettate le connessioni definite nella tabella 5.1

5.2.3.1 Componente Encara-Tracking

In questo passo, utilizzando coolbot , si è creato la componente introdotta precedentemente. Una volta definito il modello grafico sono state definite tutte le porte d'entrata, d'uscita e le transizioni che dipendono da condizioni esterne o dai dati che arrivano alla porta.

Automa a stati finiti:

In particolare, la componente include due stati (figura 5.16):

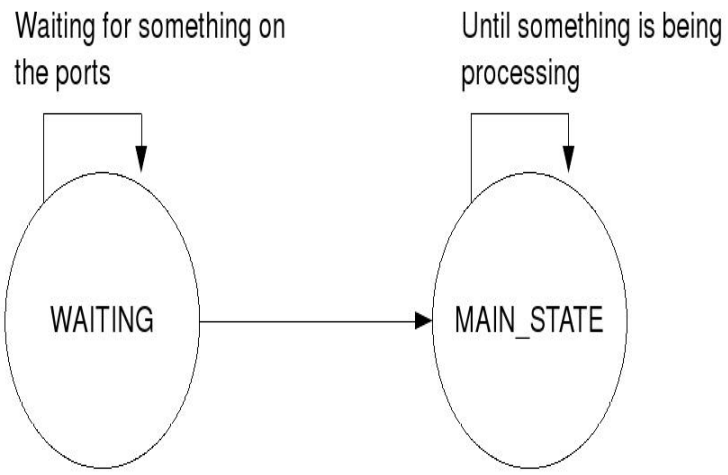

Figura 5.16: La figura rappresenta l'automa della componente Encara-Tracking.

- **Waiting**: In questo stato la componente si pone in attesa di ricevere dati alle sue porte.
- **Main State**: Dallo stato di wait, quando le porte ricevono dati da elaborare, scatta la transizione allo stato "main", che esegue tutta l'applicazione.

  Le transizioni si producono attraverso variabili private di tipo *boolean*; quando lo stato di wait le pone a "true" si ottiene una transizione allo stato di main ed i dati possono essere elaborati. Come si vede nella sezione di codice sottostante, se qualcosa arriva ad una delle porte il relativo flag cambia di stato permettendo il lancio della funzione "main" (in quanto la condizione diventa vera). Per ogni porta è previsto uno stato di wait (ad esempio "waitCameraImage" per la porta di ricezione delle immagini).

```
bool _somethingOnPortPTZJoints_;
bool _somethingOnPortCameraImage_;
bool _somethingOnPortBaseLaserGeometry_;
```

```
bool _somethingOnPortBaseLaserScan_;
bool _somethingOnPortBaseBaseConfig_;
...
if(_somethingOnPortCameraImage_ &&
   _somethingOnPortPTZJoints_ &&
   _somethingOnPortBaseLaserGeometry_ &&
   _somethingOnPortBaseLaserScan_&&
   _somethingOnPortBaseBaseConfig_)
       return MAINSTATE;
```

Definizione componente

Questa parte di codice rappresenta il disegno della componente (dove sono definite porte e transizioni):

```
component EncaraTracking
{

 input port CameraImage type poster
 port packet PlayerRobot::CameraImagePacket;

 input port PTZJoints type last
 port packet PlayerRobot::PacketPTZJoints;

 input port BaseLaserGeometry type last
 port packet PacketFrame3D;

 input port BaseLaserScan type last
 port packet PlayerRobot::LaserPacket;

 input port BaseConfig type last
 port packet PlayerRobot::ConfigPacket;

 input port BaseOdometry type last
 port packet PlayerRobot::OdometryPacket;

 input port BaseOdometryReset type last
 port packet PacketTime;

 output port RobotCommands type generic
 port packet PlayerRobot::CommandPacket;

 output port EncaraTrackingImage type poster
 port packet PlayerRobot::CameraImagePacket;
```

```
 output port Legs type generic
 port packet LegsPacket;

 output port BaseCommands type generic
 port packet PlayerRobot::CommandPacket;

entry state Waiting
{
  transition on CameraImage;
  transition on PTZJoints;
  transition on BaseLaserGeometry;
  transition on BaseLaserScan;
  transition on BaseConfig;
  transition on BaseOdometry;
  transition on BaseOdometryReset;

        };

state MainState
{
  transition on CameraImage;
  transition on PTZJoints;
  transition on BaseLaserGeometry;
  transition on BaseLaserScan;
  transition on BaseConfig;
  transition on BaseOdometry;
  transition on BaseOdometryReset;
};

};
```

Struttura Componente

Una volta che la componente è modellata graficamente, può essere generata per mezzo della direttiva di compilazione del compilatore Coolbot:

coolbot -c encara-tracking.

Tale direttiva genererà lo scheletro spiegato nella sezione 5.2.1, definendo i files che rappresentano le classi ".h" e ".cpp" (ad esempio encara-tracking.h e encara-tracking.cpp). Queste classi verranno successivamente dettagliate con le funzioni necessarie a far si che la componente possa fare quello per cui è stata ideata.

Come già detto, tutte le componenti sono state definite in una cartella chiamata "cvs-projects", in maniera tale da rendere più ordinato lo sviluppo e sfruttare la politica "cvs". Così facendo, la versione aggiornata del software può essere scaricata dal server in un qualsiasi pc, tramite la primitiva **cvs update**.

La cartella include le seguenti componenti (definite nella sezione 5.2):

- **coolbot**
- **coolbot-compiler**
- **encara-2.09**
- **encara-tracking**
- **grid-map**
- **nd-navigation**
- **short-term-planner**
- **player-robot**

La componente coolbot genera le seguenti cartelle ed i seguenti files:

Cartelle:

- **bin**: Questa cartella include l'eseguibile **"encara-tracking-gtk-test"**, che permette di lanciare l'applicazione. Come parametro viene passato l'indirizzo IP del robot (ad esempio: 10.209.0.123), in modo da connettersi ad esso.
- **examples**: Questa cartella include i files necessari a stabilire le connessioni tra le porte delle componenti e per gestire le viste. Rappresenta, quindi, il supervisore gtk a cui sono connesse tutte le componenti. Tali componenti inviano i risultati dei dati elaborati ad esso in modo che possa effettuare il controllo ed il monitoraggio del corretto funzionamento dell'automa.

  Tale cartella include i seguenti files:

  - **sphere-gtk.h**: Questo file è necessario per la vista delle immagini elaborate.
  - **sphere-gtk.cpp**: Crea la finestra di visualizzazione.
  - **player-robot-gtk.h/.cpp**: Questi files sono necessari per vedere i dati che arrivano dai sensori laser, sonar e camera.
  - **detection-legs-gtk.h**: Questo file è necessario per vedere le regioni che identificano le gambe riconosciute dall'algoritmo implementato.
  - **detection-legs-gtk.cpp**: Questo file implementa i metodi del relativo "header".
  - **encara-tracking-gtk-test.cpp**: Questo file crea l'eseguibile dell'applicazione e definisce tutte le connessioni tra le porte. Un esempio di connesione tra componenti è il seguente:

```
\\Si crea l'istanza di player-robot con l'indirizzo ip

PlayerRobot robot(100,"localhost");

\\Primitiva di connessione

robot.connect(
            PlayerRobot::CAMERA_IMAGE,
            //porta d'uscita

            encaraTracking.getConnection
            (EncaraTracking::CAMERAIMAGE)
            //porta d'entrata
        );
```

dove "robot" è una istanza della componente player robot, mentre "connect()" connette la porta di uscita della componente player-robot con la porta d'entrata della componente encara-tracking.

Per ogni componente instanziata si crea un supervisore di instanza chiamato **probe** (ad esempio ComponentProbe encaraProbe); e si esegue lanciando la primitiva run() (ad esempio la componente player-robot della camera sarà avviata con la seguente istruzione: robot.run()).

- **lib**
- **obj**

  Queste ultime due cartelle contengono le librerie e gli oggetti generati (inclusi nei make files) necessari per una corretta compilazione.

  Files:

- **encara-tracking.h**: In questa classe (definita dal compilatore) sono state aggiunte solo le inclusioni di librerie e le variabili private necessarie per la classe "encara-tracking.cpp".
- **encara-tracking.cpp**: Questa classe include tutto il necessario per il funzionamento del robot e gli algoritmi spiegati nel capitolo 4. In particolare prevede funzioni per gli stati di "wait" (una funzione di wait per ogni porta) ed una funzione di "main state"(una funzione di main state per ogni porta).

  Nello stato di *waitCameraImage*, ad esempio, è stato implementato l'algoritmo di inizializzazione della camera, nello stato di *mainCameraImage* è stato implementato il tracking della faccia, mentre nello stato *mainBaseLaserScan* è stato implementato la detection di gambe.

Come già accennato coolbot permette il multi-threading, pertanto queste funzioni ricevono ed elaborano contemporaneamente dati provenienti dai sensori. Successivamente inviano i risultati dell'elaborazione al supervisore gtk.

- **encara-tracking-packets.h**: In questa classe si sono definiti tutti i pacchetti che si inviano tra le porte delle componenti. In particolare è stato definito il pacchetto **LegPacket** che include una lista di oggetti di tipo **CoordinateList**, necessario ad inviare la lista delle coppie di gambe dalla porta "Legs" alla porta "leg_" della componente GTK.
- **coordinateList.h**: Questa classe è stata aggiunta per definire un nuovo tipo di pacchetto che non era standard in coolbot, necessaria ad inviare al visualizzatore la lista dei segmenti riconosciuti come gambe.
- **coordinateList.cpp**: Questa classe implementa i metodi della rispettiva classe.h.
- **segment.h**: Questa classe definisce i metodi e gli attributi che caratterizzano un oggetto segmento.
- **segment.cpp**: Implementazione del relativo "header".

Make file:

- **makeMakeObjects.mak**
- **encara-tracking-lib.mak**
- **encara-tracking-gtk-test.mak**

  Make Files necessari per la compilazione.

5.3 Aspetti implementativi

Per l'implementazione del progetto e per la creazione del software si è scelto di seguire un modello di sviluppo basato su un ciclo di vita evolutivo. Si è optato per tale modello in quanto permette scalabilità e facilità di integrazione di nuove funzioni. Come linguaggio di programmazione si è utilizzato il C++ (utilizzando kdevelop come ambiente di sviluppo); linguaggio nel quale si distribuiscono le librerie di software utilizzate. Questo linguaggio permette la programmazione orientata agli oggetti, capacità sfruttata al fine di seguire un'implementazione evolutiva. Questo approccio ha permesso di aggiungere nuove funzionalità, sviluppando nuovi oggetti con la possibilità di eseguire il riuso di quelli esistenti.

5.3.1 Strategia di tracking

La strategia, come accennato nel capitolo 1, utilizza una politica di tipo "Master-Slave": una volta definito un "master", scelto tra le varie persone individuate, il robot si comporta come "slave" seguendo solo quella persona.

La strategia prevede tre modalità di funzionamento:

1. **Find-Master**:

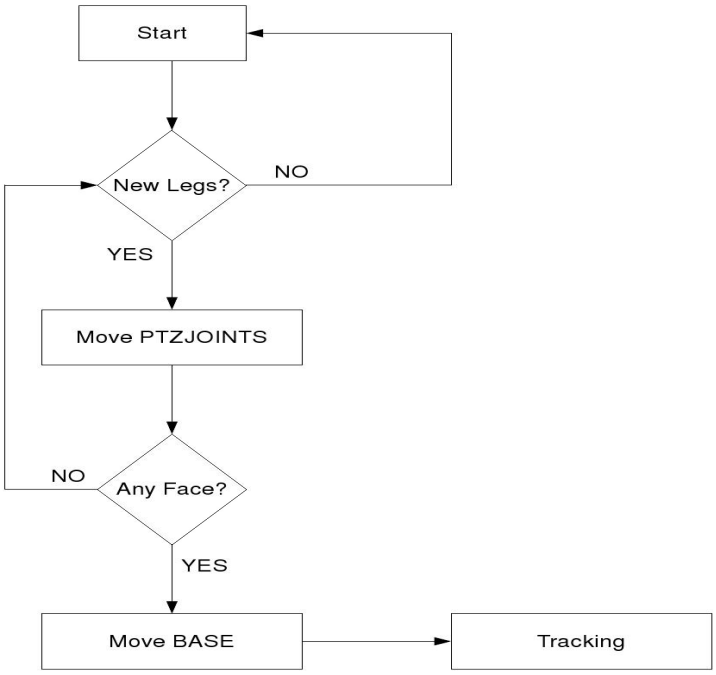

Figura 5.17: La figura rappresenta la modalità di funzionamento "find-master".

Come si vede dal diagramma di flusso riportato in figura 5.17, in questa prima

modalità, rappresentante la fase iniziale, ci si trova in uno scenario in cui il robot è fermo e si chiede se ci sono persone da rilevare, in particolare con la condizione "new legs". In base alla distanza il robot considera le gambe più vicine ad esso. Prima di muoversi, direzionerà la camera in posizione delle gambe rilevate (move PTZJOINTS): se si avrà un riscontro positivo del volto (Any faces?), allora il robot sarà sicuro di aver individuato una persona e la aggancerà come "master". Questa fase si concluderà con l'allineamento della base, e automaticamente della camera, con la persona rilevata. Il robot gira "corpo" e "testa" in modo da trovarsi di fronte alla persona. Successivamente si passa alla modalità di inseguimento. Ovviamente se non c'è riscontro del volto, in corrispondenza delle gambe rilevate, il robot tornerà alla modalità "find - master", in quanto si è in presenza di un falso positivo. Questa strategia in uno scenario reale perde in robustezza, in quanto è possibile che mentre si identificano gambe, in corrispondenza ad esse, il robot potrebbe rilevare un volto appartenente ad un'altra persona, ad esempio affacciata ad una finestra o più alta del master, nel caso in cui quest'ultima si trovi dietro di esso. Trattandosi di un prototipo sviluppato in laboratorio e considerando un raggio di azione limitato del robot, questa strategia risulta essere abbastanza robusta per gli obiettivi di questo lavoro di tesi. Tale strategia può essere modificata a seconda degli ambienti in cui si vuole far agire il robot.

Una volta rilevate le gambe, o presunte tali, si è cercato di comandare pan e tilt affinché la camera potesse inquadrare correttamente il volto di una persona, qualora fosse presente. Come mostrato nella figura 5.18, per quanto riguarda il pan si è calcolato dapprima la retta passante per l'origine ed il punto medio del segmento che congiunge le due gambe, e successivamente si è calcolato l'angolo teta che tale retta forma con l'asse y del sistema di riferimento del robot. Quest'angolo rappresenta il valore da dare al pan della camera affinché esso si direzioni nella posizione delle gambe.

Per quanto riguarda il tilt, come si vede nella figura 5.19, esso viene direzionato a seconda della distanza della persona dal robot e dall'altezza della stessa. Viene considerata un'altezza media di 175 cm quando una persona è in piedi ed un'altezza di 140 cm quando essa è seduta. Se la camera non vede il volto ad una determinata altezza, dopo circa 5 secondi cambia e prova a vedere se la persona è seduta.

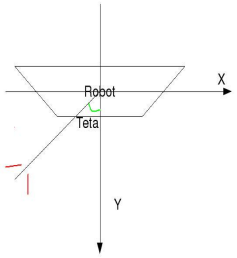

Figura 5.18: La figura rappresenta l'angolo tra l'origine del sistema di riferimento della piattaforma mobile ed il punto medio della retta che congiunge i segmenti classificati come gambe.

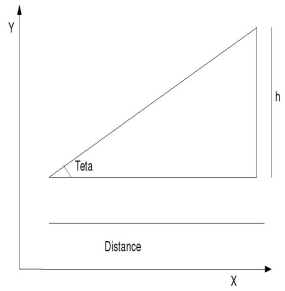

Figura 5.19: La figura rappresenta l'angolo da dare al tilt considerando altezza e distanza della persona.

2. **Tracking-Master**:

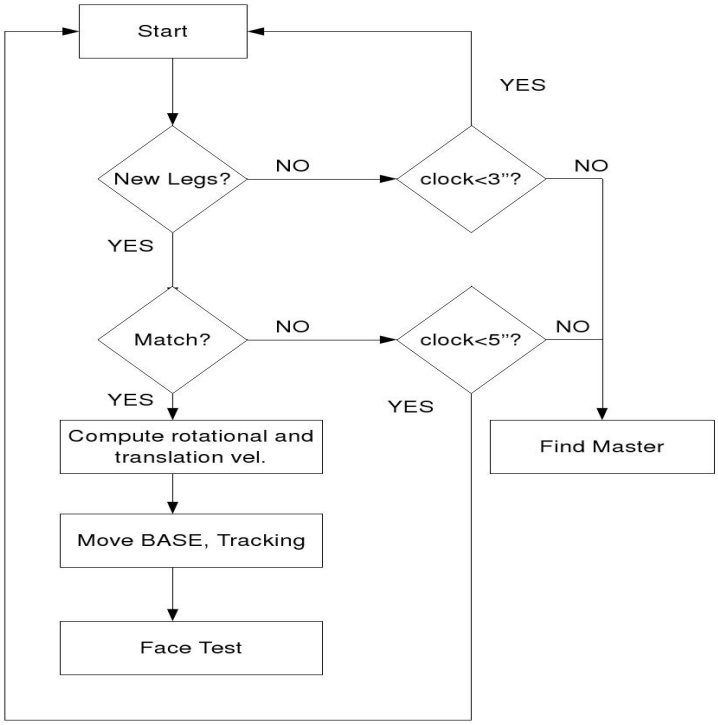

Figura 5.20: La figura rappresenta la modalità di funzionamento "tracking".

Premessa: un master è tale finché i suoi movimenti sono limitati entro un angolo teta compreso tra +/- 10 gradi ed una distanza compresa tra 0.5 e 1.5 metri.

Arrivati in questa modalità il robot già ha agganciato un master e si appresta a seguirne i movimenti. Finché il master non esce dal perimetro specificato nella premessa, il robot segue perfettamente i suoi spostamenti calcolando, di volta in volta, la velocità di rotazione e di traslazione mediante il sistema di controllo definito nel capitolo 4.

Considerando il diagramma di flusso riportato in figura 5.20, si può osservare che fino a quando il robot rileva delle gambe (è possibile la presenza di altre per-

sone oltre il master) e verifica che tali gambe appartengono al master (Match), si calcoleranno le velocità sopra citate e verrà comandata la base a seconda della distanza e dell'angolo in cui si trova il master. Contemporaneamente viene eseguito il tracking della faccia e successivamente il controllo passa al modulo "face-test".

Se nessuna coppia di gambe viene individuata per circa 3 secondi, il robot considera il master scomparso dalla scena e torna in modalità "Find-Master". Se invece, rileva coppie di gambe che non verificano la condizione espressa nella premessa, il robot aspetta 5 secondi prima di ritornare in modalità "Find-Master". Questo perché può accadere che il master sparisca dal campo visivo del robot: può succedere che esso si muova più rapidamente di quest'ultimo. Di conseguenza aspettando qualche secondo è possibile che il master torni nella visuale del robot riavviando la modalità "tracking". Se rileva altre persone il robot non le segue.

Il matching, semplicemente, verifica che le gambe rilevate appartengono al master. Si effettua testando la premessa.

In conclusione in questa fase viene effettuato contemporaneamente il tracking del volto mediante camera ed il tracking delle gambe mediante laser. Le due componenti sono indipendenti. Mediante l'utilizzo dei sistemi di controllo il robot avanza con i sistemi di riferimento allineati.

3. **Face-test**:

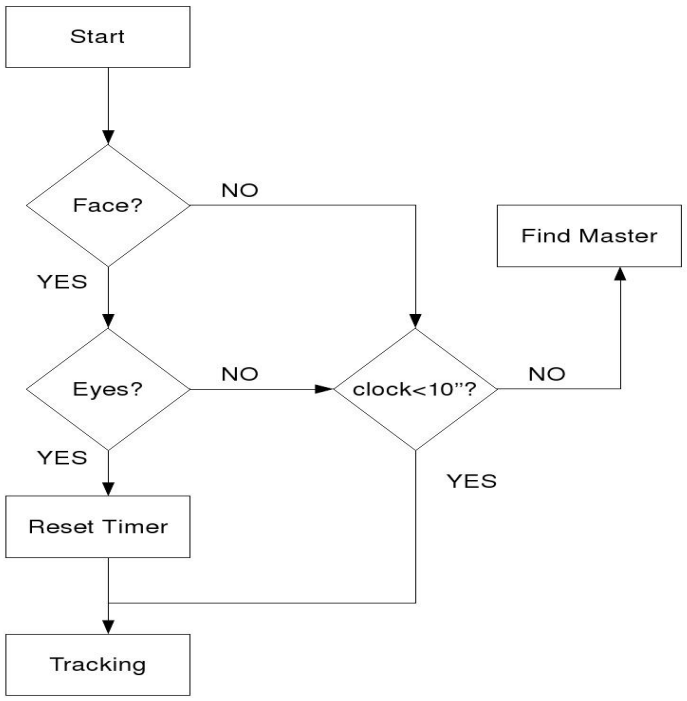

Figura 5.21: La figura rappresenta la modalità di funzionamento "face test".

Il tracking può essere effettuato anche se il robot non rileva il volto o se esso va fuori dal campo visivo della camera. Una persona può camminare di spalle ed il robot continuerà a seguirla. Questo modulo fa si che, dopo un certo numero di secondi (stimato a 10) il robot verifica, mediante camera, che effettivamente sta seguendo una persona. Come illustrato nel diagramma di figura 5.21, il robot effettua test sulla presenza del volto e degli occhi (per avere un sistema più affidabile). Se per 10 secondi il robot non riscontra un volto umano emetterà un "beep sonoro" che avvisa la persona che cammina di spalle di girarsi, in modo che il robot possa essere sicuro di seguire effettivamente una persona. Si resetta il timer e si ritorna alla modalità "tracking". Se il volto non viene riscontrato significherà che il robot avrà perso il master e ritornerà in modalità "Find-Master".

5.3.2 Presenza di ostacoli

La presenza di ostacoli non faceva parte degli obiettivi del progetto, anche perché già esiste la componente "nd-navigation" che se inclusa permette al robot di evitare ostacoli. Quello che è stato implementato invece, è una simulazione della presenza degli ostacoli. Insieme al gruppo di lavoro si è scelto di modificare la velocità di traslazione in corrispondenza di ostacoli vicini. Quando il robot passa vicino ad ostacoli rallenta la velocità per non urtarli. Ad esempio quando passa attraverso una porta rallenta e prosegue con cautela. Contrariamente aumenta la velocità quando non avvisa presenza di ostacoli nelle sue vicinanze.

Capitolo 6

Esperimenti

Come si vede nelle figure 6.1, 6.2 e 6.3, lo scenario in cui il robot rileva persone è il laboratorio "SIANI". Il robot una volta che riconosce una persona la segue per tutto il laboratorio fino ad uscire fuori. Successivamente prosegue lungo il corridorio dell'edificio.

L' obiettivo quindi, è di farsi seguire dal robot fuori del laboratorio e vedere come si comporta all'esterno, dove è possibile la presenza di altre persone.

All'interno del laboratorio sono presenti numerose fonti di oggetti che il robot può rilevare come possibili coppie di gambe, ad esempio: sedie, scatoloni rettangolari vicini tra loro, tubi o rotoli di cartone posti in piedi.

Figura 6.1: La figura rappresenta il laboratorio "SIANI".

Figura 6.2: La figura rappresenta l'interno del laboratorio "SIANI", dove si muove il robot. Sono presenti ostacoli.

Figura 6.3: La figura rappresenta il corridoio del laboratorio "SIANI". È possibile il passaggio di altre persone mentre il robot segue il master.

In uno stesso scenario vengono provate tre diverse modalità di individuazione e tracciamento, ottenute :

- **solo utilizzando la visione**:

  Tale modo di funzionamento, consiste nell'effettuare detection e tracking solo mediante l'utilizzo della camera. L' implentazione dell'algoritmo di tracking non prevede l'utilizzo del sistema di controllo per l'incremento del pan e del tilt, ma consiste di un incremento costante pari ad 1 pixel per ogni iterazione. Tale algoritmo prevede la suddivisione della finestra di visualizzazione in ranghi, ovvero delle zone in cui viene a trovarsi il rettangolo della faccia. Questo è necessario affinché base e camera riescano a sincronizzarsi per raggiungere un obiettivo comune (centrare il volto al centro della finestra). Vengono quindi considerate due differenti "zone morte": una per la camera ed una per la base. Se il rettangolo si trova in tali zone non vengono inviati comandi o alla base o alla camera. La zona

morta della camera è inclusa in quella della base. Ad esempio come si vede nel seguente frammento di codice, se la differenza tra il punto medio del rettangolo ed il centro della finestra è maggiore di zero, si settano i parametri della camera solo se il rettangolo è fuori dalla zona morta (rango pari a 15 pixels). Stessa cosa per quanto riguarda la base, si setta una velocità costante di rotazione pari a 7 Rad/sec e si invia il comando alla piattaforma se e solo se il rettangolo si trova fuori della zona morta per la base (rango pari a 25). Quando ci si trova nella zona morta la piattaforma avanza (sempre a velocità costante).

```
if(difference_x>0  ){

 if(difference_x>15 ){

   if(difference_y>0){

    if(difference_y>15){

     pPacket->setPTZPosition(
        RADIANS(DEGREES(pJoints->get().pan) + 1),
        RADIANS(DEGREES(pJoints->get().tilt) + 1),
         -1 // zoom
        );

        if(difference_x>25 )

            pPacketBase->setSpeed(
                   7
                   0
            );
         _pOBox_->send(BASECOMMANDS);
         _pOBox_->send(ROBOTSCOMMANDS);

        ...
```

Con questa tecnica si ottengono ottimi risultati a livello del solo tracking della faccia (in quanto la segue perfettamente), ma non si riduce la presenza di errori dovuti a falsi positivi, occlusioni e condizioni di luce. È possibile che Encara2 riconosce una "non faccia" come una faccia producendo instabilità del comportamento del robot: non avendo riscontro con le gambe segue qualsiasi cosa riconosciuta come faccia. Inoltre, la presenza di più persone porta la camera (e la base) ad effettuare movimenti bruschi da un volto ad un altro. Questa soluzione non viene presa in considerazione, anche perché capitano situazioni in cui il robot avanza con i sistemi di riferimento non allineati, succede che il robot prosegue con la camera che mira ai lati, elaborando altre cose.

- **solo utilizzando i laser**:

  Essendo i laser immuni da variazioni luminose e dai fattori descritti nel capitolo 1, è possibile riconoscere una persona in maniera più affidabile, per mezzo della rilevazione delle gambe (una persona, anche se di spalle viene rilevata). Utilizzando solo l'algoritmo leg detection, il robot segue la persona per il percorso finchè non rileva la presenza di altre gambe o presunte tali. Quindi il comportamento che si ottiene è simile al comportamento precedente, ovvero utilizzando solo la visione. Il robot è instabile in presenza di altre gambe od oggetti simili. Migliori prestazioni fornisce la soluzione che sincronizza le due componenti.

- **utilizzando insieme laser e visione**.

  In questa modalità, descritta nei capitoli precedenti, il robot parte dal laboratorio e segue perfettamente la persona, elaborando informazione proveniente sia dai laser e sia dalla camera. Considerando, come già detto, lo stesso scenario per tutte e tre le modalità, si è in una situazione iniziale in cui sono presenti davanti ad esso 5 persone poste a distanze pressochè uguali. Il robot aggancia la persona più vicina ad esso e la traccia. Nel percorso sono presenti ostacoli ed altre persone. Questo permette di studiare e modificare il comportamento del robot in tali circostanze. Quello che si osserva è che il robot ignora la presenza di altre persone e rallenta in presenza di ostacoli vicini, procedendo con "cautela" attraverso la porta che lo conduce all'esterno del laboratorio. Una volta fuori, il robot aumenta la sua velocità procedendo con andamento più rapido in corrispondenza di ampi spazi liberi (privi di ostacoli). La limitazione stà nel fatto che fuori dal laboratorio si hanno grandi variazioni di luce (dovute al tempo, alle lampade ecc..) che fanno lavorare Encara2 con molta più fatica. Di conseguenza, è possibile che il robot pur rilevando le gambe si blocca, in quanto non ha un riscontro visivo del volto, ed in particolare degli occhi della persona.

  Le prove effettuate si basano su variazioni dei valori delle variabili dei sistemi di controllo (capitolo 4) e sul tempo di attesa.

  Si osserva che un tempo troppo corto non è adeguato. Ad esempio, una persona camminando di spalle al robot può seguire un passo più rapido rispetto ad un andamento frontale al robot, e quindi uscire più di frequente dal perimetro in cui il robot riconosce una persona come master (vedere la premessa del capitolo precedente). Se il tempo è troppo corto la persona non fà in tempo a rientrare nel perimetro ed il robot, torna in modalità "find-master" più facilmente. Si è visto che 5 secondi , rappresentano un tempo sufficiente affinché il master torni ad essere rilevato. Sono stati effettuati test anche sul tempo di attesa in cui il robot segue la persona pur non rilevando la faccia, ossia mentre cammina di spalle (o se la faccia sparisce dalla camera). Si è visto che un tempo sufficiente affinché il master si giri e si faccia riconoscere è di 10 secondi. Minor tempo provoca il ritorno alla modalità "find-master".

  Infine, per rendere meno bruschi ed aggressivi i movimenti della piattaforma si "gioca" con i parametri delle costanti dei sistemi di controllo. Si parte con valori bassi fino ad arrivare ad una giusta configurazione che rende il robot più "sciolto" nei movimenti.

Prima di raggiungere questa modalità che da risultati abbastanza buoni, si è provato ad utilizzare la seguente strategia consistente di sei passi, descritti in figura 6.4:

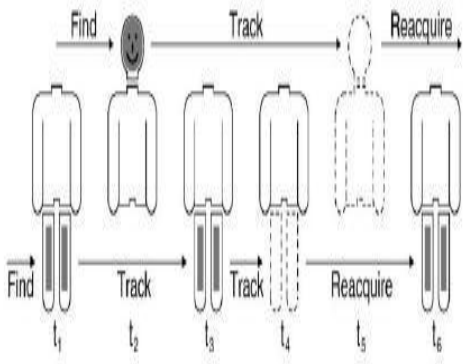

Figura 6.4: La figura rappresenta la strategia di anchoring

t1 : In questo passo, il processo di riconoscimento di persona è iniziato e tutti i componenti realizzano i propri processi di detection ("find"). La rilevazione delle gambe genera una percezione e la coppia di gambe viene agganciata per la prima volta.

Il processo cambia di modo, ed effettua lo switch dal modo "find" al modo "track". Successivamente viene utilizzata la posizione delle gambe per far puntare la camera nella giusta direzione, attivando la modalità "find" della camera (come la strategia descritta nel capitolo 5).

t2 : La detection di faccia genera una percezione ed il processo della camera cambia dalla modalità "find" a "track".

t3 : Di nuovo, la detection di gambe genera una nuova percezione. Viene aggiornata la posizione delle gambe.

t4 : In questo passo, nuovi dati vengono elaborati, ma non vengono rilevate gambe. Si passa alla modalità "reacquire".

t5 : Una nuova immagine viene elaborata, ma non viene riscontrata faccia con la predizione della posizione della persona. Si passa alla modalità "reacquire".

t6 : Nella nuova scansione laser una percezione delle gambe coincide con la predizione della posizione della persona: le gambe sono detectate un' altra volta.

Riassumendo, la strategia illustrata sopra, fornisce la rilevazione delle coppie di gambe, identificando cosi la posizione della presunta persona, in modo da poter direzionare la camera in quella posizione. Successivamente l'applicazione esegue il tracking della camera e delle gambe indipendentemente senza sincronizzazione dei componenti.

Questa strategia però, non può essere implementata in quanto, si sono riscontrano problemi con la camera. Questo modo di lavorare prevede che piattaforma e camera,

per effettuare tracking indipendente devono muoversi contemporaneamente in contro fase tra loro. Il problema è che la base e la camera possono solo essere comandate, rispettivamente, per velocità e per posizione, quindi per far in modo che la camera si allinei con la rotazione della base, si deve modificare l'incremento del pan a 3 pixels (minimo) per iterazione. Questo esegue l'allineamento, ma di contro rende il tracking instabile, ovvero la camera si muove in maniera brusca ed incontrollata quando tenta di inseguire un volto. (Come già detto, la strategia prevede di effettuare il tracking delle gambe e della faccia in maniera indipendente quando si è sicuro di rilevare una persona). Questo problema può essere risolto abbassando la velocità rotazionale della piattaforma, ma ciò non è possibile in quanto la velocità angolare minima di rotazione, affinché la base possa muoversi, è di 7 RAD/SEC. Tale velocità è sempre maggiore della velocità minima di rotazione della camera. Per ovviare a questo si utilizza la strategia descritta nel documento, inserendo quindi i sistemi di controllo ed utilizzando il tracking solo dopo che un master viene agganciato.

Infine si prova il robot, con persone sedute su sedie mobili. Quello che si riscontra è che il robot riconosce e segue la persona. Il tilt della camera, se non si trova faccia ad un altezza media in corrispondenza delle gambe, si abbassa per verificare se tale persona è seduta. Ovviamente se non è nemmeno seduta il robot cade in un falso positivo, ma non si muove perché non trova riscontro visivo.

I limiti riscontrati sono dovuti semplicemente alle diverse condizioni di luce. Per rendere il sistema affidabile si considera la presenza degli occhi, questa condizione permette di evitare falsi positivi, ma di contro blocca il robot se la luce non permette ad Encara2 di rilevarne la presenza. Considerare solo il rettangolo del volto, ovvia al caso precedente, ma è sensibile alla presenza di falsi positivi.

6.1 Test

I grafici riportati nelle figure 6.5, 6.6 e 6.7, rappresentano i risultati dei test effettuati nel laboratorio. In particolare nel grafico di figura 6.5, si riporta il tasso di rilevamento delle gambe: sull'asse y si rappresenta il Detection Rate (D.R) e sull'asse x il numero di falsi positivi (F.P); in figura 6.6 si considera il tasso di rilevamento della faccia: sull'asse y si riporta il D.R. e sull'asse x il numero di sequenza dello stream video; ed infine, in figura 6.7 i tempi di tracciamento: sull'asse y si riportano i tempi di tracking e sull'asse x il numero del test.

I risultati sono stati graficati considerando un certo numero di step: ogni step rappresenta una tratta del laboratorio.

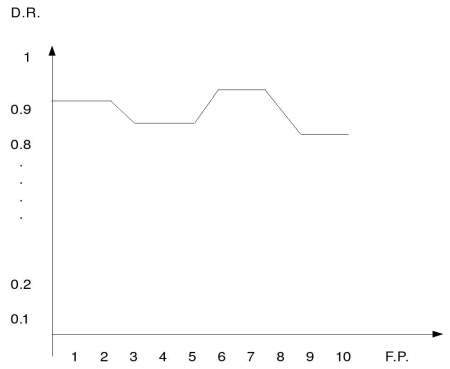

Figura 6.5: La figura rappresenta il tasso di rilevamento delle gambe.

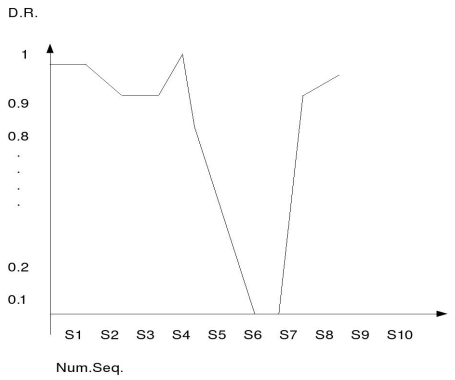

Figura 6.6: La figura rappresenta il tasso di rilevamento della faccia.

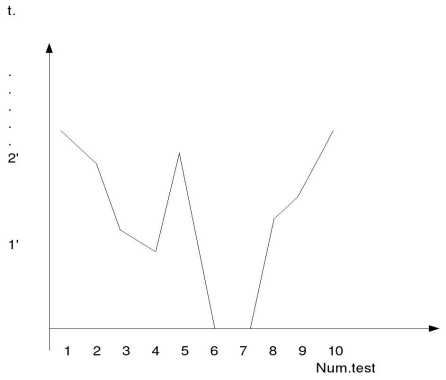

Figura 6.7: La figura rappresenta i tempi di tracciamento.

Si considerano 10 test in cui il robot da fermo rileva un master e lo segue fin fuori del laboratorio. Tale scenario prevede 10 passi. Nel grafico in figura 6.7 si può osservare nel primo step, al momento della partenza, che in condizioni stabili di luce il robot rileva un master e lo segue per circa due minuti. Tale osservazione trova riscontro nei grafici di figura 6.5 e 6.6, dove si ha un D.R alto sia per la faccia che per le gambe. Per quanto riguarda il detection rate della faccia si considerano, ad ogni step, sequenze di 100 frames in cui si calcolano falsi positivi e falsi negativi. Per quanto riguarda quello delle gambe, ad ogni step vengono aggiunti incrementalmente falsi positivi (es: step 4, 4 falsi positivi). Si considerano falsi positivi: sedie, oggetti rettangolari o persone. Man mano che il robot avanza nel laboratorio le condizioni di luce variano. Quindi, come si osserva nei passi 6 e 7 (che rappresentano il centro del laboratorio) del grafico in figura 6.7, il robot non segue il master (tempo di tracciamento zero). Questo trova riscontro nel grafico di figura 6.6, in cui si ha un D.R. pari a zero: sia faccia che occhi non vengono rilevati. Cambiando orientazione al robot e cambiando tratta, Il robot torna a seguire il master e a rilevare facce; infatti dalla sequenza numero 8 il D.R. della faccia aumenta. Nonostante la luce in queste tratte, l'applicazione continua a rilevare gambe, infatti dal grafico di figura 6.5 il D.R. risulta essere sempre alto, come detto nei capitoli precedenti il laser non è sensibile alla luce.

In conclusione i casi migliori si ottengono nelle tratte iniziali (1 e 2), dove non ci sono grandi "accecamenti", e fuori del laboratorio (step 9 e 10), nei giorni in cui la luce solare non è ne troppo forte e ne troppo debole. I casi peggiori sono registrati al centro del laboratorio, dove è concentrata l'illuminazione (numero test 6 e 7).

Capitolo 7

Conclusioni e sviluppi futuri

Lo sviluppo di questo progetto è stato possibile grazie alla vincita della borsa erasmus ed il consenso della professoressa **Fiora Pirri**, la quale mi ha aiutato con la scelta della destinazione e del tutore con cui effettuarlo.

L' esperienza erasmus, se provata, è un capitolo della vita che si porta avanti nel tempo senza mai dimenticare. Questo perché è un' esperienza che segna chi ne usufruisce. Segna nel senso, che fa maturare, crescere, imparare a comportarsi ed a districarsi tra le difficoltà quotidiane.

Oltre a questo, l'erasmus proietta lo studente verso un mondo a parte, dove è possibile conoscere persone più o meno interessanti, incontrare nuovi amici, confrontarsi con altre culture immergendosi negli "usi e costumi" del paese ospitante. Ad esempio per quanto mi riguarda nella città in cui risiedevo mi ritrovavo a festeggiare ogni settimana una festività differente, in ambienti accoglienti e generosi. Ovviamente non è tutto oro ciò che luccica.. ma è comunque una bella esperienza da consigliare a chiunque abbia voglia di conoscere il mondo.

Personalmente sono soddisfatto della mia esperienza, in quanto ho trovato un ambiente di amicizia e di disponibilità, soprattutto nel periodo iniziale, in cui erano evidenti i problemi di comunicazione. Fortunatamente la lingua spagnola è simile a quella italiana e quindi in poco tempo è stato possibile apprenderla. Per quanto riguarda i tutori ed il gruppo di ricerca, devo dire di essermi trovato molto bene, nel senso che ho incontrato dei professori veramente "alla mano": gentili, pazienti, simpatici e disponibili ad aiutarmi e risolvere ogni mio dubbio, come del resto lo è stata l'università di Las Palmas.

Sono soddisfatto anche del fatto di aver raggiunto, nel tempo avuto a disposizione (7 mesi), tutti gli obiettivi prefissati all'inizio. Il robot rileva e segue una persona, a meno di eccezioni come descritto nel capitolo 6.

In questo lavoro di tesi si è voluto vedere come i metodi di face detection potessero essere utilizzati in applicazioni real time di intelligenza artificiale. In particolare tra i metodi descritti nel capitolo 2, che sono tra i migliori, si è utilizzato ENCARA2 che fornisce prestazioni migliori in tempo reale quando si considerano stream

video. Principalmente, tali metodi sono implementazioni che si basano su detectors di tipo "Viola e Jones". Per quanto mi riguarda ho verificato che Encara2 fornisce performance migliori (in termini di tassi di riconoscimento e falsi positivi) del suddetto metodo, in quanto dopo aver implementato il metodo di "Viola e Jones", per l'esame di "Visione e Percezione", ho visto che esso, applicato ad immagini in cui erano presenti facce frontali in formato 24x24 aveva un più alto tasso di falsi positivi. Probabilmente questo era dovuto al fatto di non aver eseguito un apprendimento sufficiente e di non aver implementato un detector multi-scala e multi-risoluzione. Encara2 risulta essere più affidabile perchè la sua implementazione prevede il legame di robusti face detectors: infatti esso considera l'utilizzo di face detectors basati su "Viola e Jones" che riconoscono immagini frontali e contesto locale. Infine Encara2 risulta essere migliore della sua precedente versione, la quale non considera per niente il metodo "Viola e Jones", per il motivo che quest' ultimo riconosce una sola faccia in una immagine, mentre la versione 2 effettua il rilevamento di più facce.

Per quanto riguarda il futuro, sicuramente si dovrà migliorare l'algoritmo "ENCARA2"; il suo comportamento dipende molto dalle condizioni di luce: sia con luce debole che forte è difficile che vengano riconosciuti gli occhi. Quest' ultima è una caratteristica fondamentale ai fini del corretto funzionamento del robot. Per il futuro è previsto anche l'utilizzo di primitive openCv che lavorano sul colore delle immagini, al fine di riconoscere una persona dal colore della pelle, dal colore dei capelli o dal colore della maglia. Si potrebbe considerare non solo il riconoscimento frontale ma anche il riconoscimento laterale, come ad esempio il metodo "Schneiderman e Kanade" descritto nel capitolo 2. Inoltre si potrebbe aggiungere l'uso di sonar per il riconoscimento vocale e si potrebbe rendere capace il robot di seguire più persone contemporaneamente (invece che un solo master) evitando ostacoli.

Bibliografia

[1] M.Castrillòn, O. Déniz, C. Guerra, M. Hernàndez, *ENCARA2: Real-time Detection of Multiple Faces at Different Resolutions in Video Streams*, Journal of Visual Communication and Image Representation, ISSN 1047-3203 , vol 18, issue 2, pp. 130-140, April 2007.

[2] Cui, J., Zha, H., Zhao, H., and Shibasaki, *Multi-modal tracking of people using laser scanners and video camera*, Image Vision Comput. pp 240-252 , February 2008, `http://dx.doi.org/10.1016/j.imavis.2007.05.005`.

[3] Jae Hoon Lee; Takashi Tsubouchi; Kenjiro Yamamoto; Saku Egawa, *People Tracking Using a Robot in Motion with Laser Range Finder*, Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on , vol., no., pp.2936-2942, October 2006.

[4] Anand Panangadan, Maja J. Mataric, and Gaurav S. Sukhatme, *Detecting Anomalous Human Interactions using Laser Range-finders*, In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 2136-2141, September 2004, IEEE Press.

[5] P. Viola, M.J. Jones, *Rapid Object Detection using a Boosted Cascade of Simple Features*.

[6] M. Castrillon Santana, F. Hernandez Tejera, J. Cabrera Gamez, *Encara: real-time detection of frontal faces*, in: International Conference on Image Processing, Barcelona, Spain, 2003.

[7] Schneiderman e Kanade, *A Statistical Method for 3D Object Detection Applied to Faces and Cars*.

[8] H. Kruppa, M. Castrillon Santana, B. Schiele,*Fast and Robust Face Finding via Local Context*, In Joint IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance (VS-PETS), 2003, pp.157-164.

[9] A.Domínguez-Brito, D. Hernàndez-Sosa, J.p Isern-Gonzàlez, Gàmez, *CoolBOT: a Component Based Software Infrastructure for Robotics.*

[10] M.Castrillòn, *Sobre la Detecciòn en Tiempo Real de Caras en Secuencias de Video. Una Aproximaciòn Oportunista.*

[11] Fritsch, M. Kleinehagenbrock, S. Lang, T. Plotz, G.A. Fink, G. Sagerer, *Multi-modal anchoring for human robot interaction.*

[12] *Player/Stage*. `http://playerstage.sourceforge.net/index.php?src=player`.