POLITECNICO DI MILANO
Corso di Laurea in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

RoboWii 2.0.1: sviluppo di un gioco con un robot autonomo dotato di visione

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Relatore: Prof. Andrea Bonarini

Tesi di Laurea di:
Andrea Pontecorvo, matricola 702084

Anno Accademico 2009-2010

Ringraziamenti

Ringrazio i miei genitori e i miei familiari che mi hanno supportato in questi anni permettendomi di raggiungere questo primo traguardo.
Ringrazio inoltre i miei amici e i compagni di università coi quali ho condiviso questi anni di studio.
Ringrazio il professor Andrea Bonarini per avermi permesso di realizzare questo progetto e per avermi seguito durante lo svolgimento del lavoro.
Infine volevo ringraziare tutti i ragazzi dell'AirLab. In particolare Simone Cerinai e Davide Rizzi per il supporto tecnico e i suggerimenti che mi hanno dato.

Indice

Capitolo 1

Introduzione

1.1 Inquadramento generale

Negli ultimi anni sono diventati sempre più comuni piccoli robot autonomi in grado di interagire in modi più o meno vari con l'utente: il fine di questi robot è spesso l'intrattenimento, reagendo agli stimoli tattili, sonori e visivi forniti dall'utente.
Lo scopo di questo lavoro è quello di realizzare un gioco per ambiente domestico, basato sull'interazione tra un robot autonomo, dotato di un sistema di visione, e un giocatore umano, che utilizza il controller Wii Remote della console Nintendo Wii.

1.2 Breve descrizione del lavoro

Il lavoro svloto in questa tesi si colloca a cavallo tra l'area della visione artificiale e quella dell'interazione tra uomo e robot.

Per quanto riguarda la visione è stato realizzato un algoritmo per l'individuazione di oggetti nel caso in cui sia la telecamera sia l'oggetto stesso siano in movimento, bsato sull'identificazione dei punti SURF e su un parziale filtraggio dei colori.

Per quanto riguarda l'aspetto legato all'interazione tra il robot e l'uomo questo lavoro si colloca all'interno del progetto RoboWii, che riguarda la creazione di giochi basati sull'interazione tra robot autonomi e giocatori umani che utilizzano il controller Wiimote come mezzo primario di interfacciamento con il robot.

Inoltre, ci si è occupati di sviluppare i comportamenti del robot utilizzando Mr. BRIAN, un software sviluppato dall'AIRLab all'interno del progetto

MRT (Milano Robocup Team) che permette la creazioni di comportamenti regolati da logica fuzzy e basati su input di diversi sensori.

Il gioco presentato in questa tesi è una versone modificata del classico gioco del nascondino. In questo caso, il robot Spykee deve nascondersi e il giocatore deve riuscire a trovarlo e spargli utilizzando il Wiimote.

I test effettuati hanno dimostrato che il gioco sviluppato risulta essere adatto sopratutto ad un ambiente, come quello domestico, che presenta diversi possibili nascondigli per il robot e che non richeide elevate velocità di spostamento e una capacità visiva a lungo raggio.

Inoltre, il lavoro presenta diversi aspetti che possono essere ulteriormente migliorati. Ad esempio, l'inserimento di un sistema di localizzazione assoluto per il robot e per il giocatore, o ancora il miglioramento della capacità visive del robot.

1.3 Struttura della tesi

La tesi è strutturata nel modo seguente.

Nella sezione due si mostra lo stato dell'arte delle tecnologie utilizzate. In particolare si presenta il robot utilizzato e le modifiche che vi sono state fatte precedentemente, come l'aggiunta di sonar e led.
Quindi si presenta il problema del riconoscimento di oggeti nella visione artificiale e si illustrano alcuni algoritmi e librerie. Infine viene presentato il framework utilizzato come base per lo sviluppo del software di controllo del gioco e dell'intelligenza artificiale.

Nella sezione tre si illustra lo svolgimeneto del progetto. In particolare si analizza l'algoritmo di visione implementato e vengono descritte le differenti componenti software del gioco.
Infine, viene descritto il gioco e i comportamenti che il robot può assumere durante il suo svogimento.

Nel quarto capitolo si presentano i problemi riscotrati durante lo svolgimento della tesi, i risultati ottenuti e i possibili sviluppi futuri.

Nell'appendice A si riporta il manuale del gioco: viene illustrato come installare il software necessario, come configurarlo e come giocare.

Nell'appendice B si riporta la maggior parte del codice sorgente del software.

Nell'appendice C si riportano le variabili fuzzy, i predicati e i comportamenti implementati con Mr. BRIAN.

Capitolo 2

Stato dell'arte

2.1 Spykee

Spykee [2] è un picclo robot umanoide prodotto dalla Meccano, dotato di due braccia non motorizzate, due cingoli, alcune luci decorative, un altoparlante, un microfono e una telecamera sulla testa. Il robot, equipaggiato con una secheda Wi-Fi 802.11 b/g, è in grado di comunicare, direttamente o tramite internet se connesso a un router, con un PC dotato del software di controllo, disponibile solo per Windows e Mac OSX. La telecamera di Spykee invia al computer immagini a colori, compresse in formato jpeg, con una risoluzione di 320x240x24 pixel a circa 10 frame per secondo. E' anche possibile inviare al pc l'audio registrato tramite il microfono di bordo ed invaire dal computer file audio mp3 o wav e farli riprodurre al robot.

Il robot si muove agevolemente solo su terreni uniformi con una velocità massima di circa 30 cm/s ed è in grado di andare avanti, indietro, sterzare e girare sul posto. I cingoli, indipendenti tra loro, non sono putroppo dotati di nessun tipo di odometria, il che consente solo il controllo in velocità.

Spykee è alimentato da un pacco batterie al litio interne che gli consentono una autonomia di circa 2-3 ore di funzionamento. La ricarica avviene per mezzo dell'apposito alimentatore a cui il robot si può agganciare in maniera automatica quando la carica è bassa. Durante la ricarica il robot può comunque essere lasciato accesso in modo da essere direttamente utilizzato tramite computer: al momento della conssione Spykee si sgancia in maniera autonoma dalla base di ricarica.
Internamente il robot dispone un processore ARM ed è equipaggiato con una versione ridotta di un kernel linux. Di recente è stato rilasciato il codice sorgnete del firmware [2].

Figura 2.1: Il robot Spykee

Spykee è stato equipaggiato con un sistema sonar in grado di rilvere le distanze sui quattro lati ad un altezza di circa 20cm da terra, quattro led rossi, quattro gialli e un led verde, normalmente utilizzati come indicatori per il gioco implementato. Sulle spalle sono presenti anche due led infrarossi utilizzati per rendere il robot individuabile dal Wiimote.

2.2 Wii Remote

Il Wii Remote (Wiimote) [7] è il controller principale della console Nintendo Wii. A differenza della magior parte dei controler, usciti nello stesso periodo, è in grado di rilevare movimenti e accelerazioni tramite un accelerometro e un sensore infrarossi ed è progettato per un utilizzo wireless. E' alimentato a batteria e comunica tramie il protocollo Bluetooth.
Il Wiimote è dotato di:

- un accelerometro ADXL330
- un sensore ottico PixArt sensibile all'infrarosso
- 10 tasti digitali, più un tasto sync e un tasto power
- uno speaker
- 4 LED blu
- un motorino interno, collegato a una massa eccentrica, che gli consente di vibrare
- una memoria EEPROM da 16KB
- uno slot di espansione

2.2.1 Bluetooth

Il Wiimote comunica con un altro dispositivo (originariamente la consolle Nintendo Wii) tramite collegamento wireless bluetooth seguendo il protocollo HID (Human Interface Device), non richiede autenticazione e per essere connesso è necessario che si trovi in discoverable mode, stato in cui si porta premendo il tasto sync (nel vano batterie) o contemporaneamente i tasti 1 e 2.
Una volta connesso il Wiimote può essere configurtato per inviare un report sullo stato dei tasti premuti, del accelerometro e del sensore infrarosso con una frequenza massima di 100Hz. Di default invece il Wiimote invia un report solo se:

- viene premuto un tasto
- l'accelerometro o il sensore infrarossi rilevano una variazione
- viene collegato un dispositivo di espansione

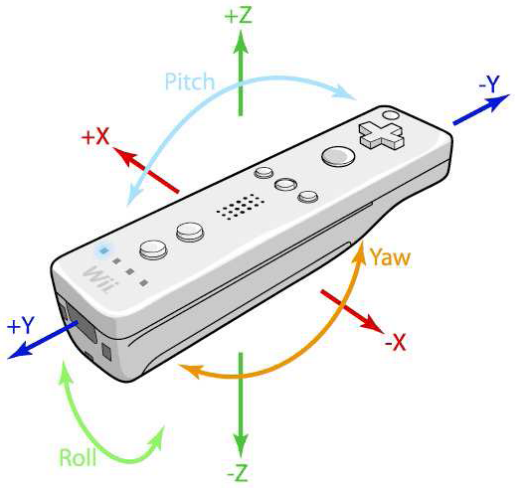

Figura 2.2: Gli angoli rilevati dagli accelerometri

2.2.2 Accelerometro

Il Wiimote è dotato di un accelerometro ADXL330 a 3 assi con un range minimo di +/-3g su ogni asse e una sensibilità del 10%. La forza rilevata su ogni asse viene poi digitalizzata con 8 bit di precisione.
L'accelerometro va abilitato esplicitamente una volta effettuata la connessione al Wiimote.

2.2.3 Infrarossi

Nella parte anteriore del Wiimote è presente un sensore PixArt monocromatico, a cui è applicato un filtro per l'infrarosso, con una risoluzione di 128x96 pixel, un'ampiezza di campo di circa 45° in orizzontale e 35° in verticale e frequenza di scansione di 100Hz.

Il processore integrato è in grado di tracciare il movimento di 4 differenti punti in uno spazio con risoluzione di 1024x768 pixel, ottenuto mediante una 8x subpixel analysis. Per questioni di efficenza al Wiimote sono inviate solo le coordinate di questi quattro punti invece che l'intera immagine che viene analizzata dal processore interno alla telecamera.

L'utilizzo dello spettro infrarsso permette una facile identificazione dei LED in quanto generalmente sono l'unica fonte forte di luce infrarossa, anche se si possono avere delle interferenze da parte della luce solare o da alcuni tipi di lampade.

Quando utilizzato con la Nintendo Wii la fonte di luce infrarossa è una barra, posta sopra o sotto la televisione, con due gruppi di cinque LED posti

Figura 2.3: Wiimote

alle estremità a una distanza prefissata. In questo modo è anche possibile stimare la distanza del Wiimote dalla sorgente infrarossa.
Dato il notevole consumo di energia anche il sensore infrarossi va esplicitamente abilitato dopo aver stabilito la connssione con il Wiimote.

2.3 Visione artificiale

La visione artificiale è l'insieme di tutte quelle tecnologie e algoritmi che consentono a una macchina di vedere e interpretare il mondo che la circonda.
Negli ultimi anni la visione artificiale ha preso piede in molti campi, ad esempio: la videosorveglianza, le console di nuova generazione e sopratutto nella robotica, sia quella industriale che quella dei robot giocattolo, come ad esempio il cane Aibo [1] o lo stesso Spykee. Questa diffusione è stata resa possibile dall'aumento della potenza di calcolo dei computer di uso comune che ha permesso di compiere notevoli passi avanti in questo campo, consentendo anche lo sviluppo di sistemi di visione in tempo reale. Tuttavia, questi sistemi, per quanto avanzati e veloci, non sono ancora in grado di eguagliare gli apparati visivi degli esseri viventi in termini di potenza computazionale e affidabilità.

Un ulteriore problema, tutt'ora non completamente risolto, che riguarda la visione artificiale è l'interpretazione delle informazioni che vengono estratte da video e immagini. Infatti, per quest'ultimo compito esistono algoritmi matematici ben conosciuti e utilizzabili in diversi contesti. L'as-

trazione e l'analisi dei dati richiedono spesso soluzioni ad hoc per il problema in questione e difficilemente utilizzabili in altri contesti.

2.4 Object Detection

Uno dei problemi più interessanti e studiati è il riconscimento di oggetti mediante visione artificiale.
Durante le fasi preliminari di questa tesi sono stati presi in considerazione alcuni tra i molti algoritmi esisteni.

Uno dei primi algoritmi considerati è stato il Template Matching: un modello dell'oggetto da individuare (il template) viene fatto scorrere su tutta l'immagine e per ogni posizione in cui si trova viene calcolato un grado di somiglianza; la migliore corrispondenza sarà dove questo valore è più elevato. A causa del confronto diretto tra immagine e modello, il template matching è in grado di individuare solo oggetti le cui caratteristiche non si discostano molto dal modello. infatti, se le condizioni di illuminazione cambiano significativamente o se la geometria dell'oggetto varia rispetto al template l'algoritmo diventa poco affidabile. Per questo motivo si è deciso di scartarlo.

Un altro algoritmo inizialmente considerato è stato quello proposto da Viola e Jones per il riconoscimento di oggetti utilizzando feature Haar-like [14], cosi chiamate perchè simili alle haar wavelets studiate da Alfred Haar. Per poter utilizzare questo algoritmo è necessario effettuare una fase di training fornendo diverse migliaia di campioni dell'oggetto (o degli oggetti) da riconoscere, chiamati positivi, e campioni negativi (altri oggietti, probabile ambiente circostante). Proprio per l'elevato numero di campioni da fornire in ingresso si è deciso di non utilizzare questo algoritmo.

Alla fine sono stati presi in considerazione due algoritmi: il SURF [6] e parzialmente il blob tracking che si basa sulle caratteristiche cromatiche dell'oggetto da ricercare.

2.5 Speeded Up Robust Features (SURF)

Lo Speeded Up Robust Features (SURF) [6] [11] proposto per la prima volta da Herbert Bay nel 2006 è un algoritmo che permette di rilevare su un'immagine i punti caratteristici invarianti alle rotazioni, alle variazioni di scala e in genere alle trasformazioni affini. A ognuno di questi punti viene associato un descrittore, che lo individua unicamente; indicizzando questi descrittori con un algoritmo di ricerca, come il K-Nearest-Nighboor,

è possibile utilizzare il SURF per l'individuazione di specifici oggetti.

La scelta dell'algoritmo da usare è ricaduta sul SURF prorpio perchè è in grado di rilevare punti invarianti alle trasformazioni affini, caratteristica molto utile dato che nel nostro caso, sia il robot sia il giocatore si spostano costantemente nell'ambiente di gioco.
Le parti principali dell'algoritmo sono descritte nelle seguenti sezioni.

2.5.1 Immagine integrale

Il primo passo dell'algoritmo è il calcolo dell'immagine integrale. Essa permette di velocizzare il calcolo della convoluzione con filtri qudrati utilizzati in seguito.
Nell'immagione integrale il pixel in posizione $p = (x, y)$ è calcolato come la somma di tutti i pixel a partire dall'angolo superiore sinistro fino al punto p.

$$I_{\sum}(p) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(i, j)$$

Una volta calcolata l'immagine integrale per ottenere il valore di intensità di un'area rettangolare di un estensione qualsiasi è sufficente accedere al valore dei quattro vertici che delimitano l'area.

2.5.2 Matrice Hessiana

Il passo successivo è il calcolo della matrice Hessiana per ogni punto $p = (x, y)$ dell'immagine definita come segue

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

dove L è la convoluzione della derivata sconda della Gaussiana $\frac{\partial^2}{\partial x^2} g(\sigma)$ con l'immagine al punto p e con un fattore di scala σ.

L'utilizzo della Gaussiana permette di ottenere risultati che non variano alla rotazione dell'immagine, anche se a causa della sua necessaria discretizzazione il filtro gaussiano assume forma quadrata e si ha una diminuzione della precisione dei risultati per rotazioni multiple di $\frac{\pi}{4}$.

Una volta calcolata l'Hessiana per ogni punto dell'immagine vengono individuate delle strutture di tipo blob-like nei punti in cui il determinante della matrice è massimo. Questi punti vengono poi memorizzati raggrupandoli in base al fattore di scala dell'immagine.

2.5.3 Scale-space

Un metodo molto comune per analizzare la stessa immagine a differenti fattori di scala è la rappresentazione piramidale: essa consiste nel ridurre progressivamente le dimensioni dell'immagine dopo avervi applicato un filtro gaussiano per ridurre i disturbi. Questo metodo presenta però degli svantaggi in quando riducendo l'immagine si ottegono effetti di aliasing e le componenti ad alta frequenza vengono perse.

Per ovviare a questi problemi il SURF, invece di ridurre l'immagine, aumenta le dimensioni del filtro gaussiano applicandolo poi all'immagine integrale in modo da ottere i punti di tipo blob-like a differenti fattori di scala. Aumentando le dimensioni del filtro bisogna fare attenzione a mantenere le proporzioni e a garantire la presenza del pixel centrale.
Lo scale-space è suddiviso in ottave, dove ognuna di esse fa rifermento a una serie di filtri dove l'ultimo ha dimensione doppia rispetto al primo.

2.5.4 Identificazione dei punti di interesse

Per ogni fattore di scala vengono poi identificati i punti di interesse. L'identificazione avviene in tre fasi: prima di tutto viene effettuata una sogliatura utilizzando come soglia un valore inserito dall'utente. Il secondo passo è l'applicazione di una non-maximal suppression in cui ogni pixel è confrontato con gli 8 pixel vicini e con i 9 pixel corrispondenti nella scala precedente e successiva. L'ultimo passo consiste nell'interpolazione dei punti vicini per ottenere una maggiore accuratezza.

2.5.5 Desrittori dei punti di interesse

Una volta identificati i punti di interesse per ognuno di essi vengono calcolati un orientamento e un descrittore a 64 dimensioni. Entrambi vengono calcolati mediante l'utilizzo di filtri basati sulle Haar-wavelets [14] per ottenere i gradienti lungo l'asse verticale e orizzontale.
L'orientamento viene calcolato applicando i filtri Haar ad un'area circolare centrata sul punto di interesse ottenuta facendo ruotare una finestra di ampiezza $\frac{\pi}{3}$. I grandienti vengono poi sommati per ottenere l'orientamento dominante.

Il descrittore vero è proprio è invece un vettore a 64 dimensini calcolato su un'area quadrata centrata sul punto di interesse, la cui dimensione dipende dalla scala e orientata in base all'orientamento calcolato precedentemente. Questa regione viene poi divisa in sottoregioni di $4x4$ quadrati e in ognuno di essi vengono applicati a 25 punti i filtri Haar. La somma dei

gradienti porta ad avere 64 valori che rappresentano il descrittore del punto di interesse.

2.6 Blob detection e colore

Uno dei più semplici metodi di individuazione di oggetti è quello basato sulle loro caratteristiche cromatiche [17]. Nella sua forma più semplice si fornisce in ingresso uno specifico colore a un algoritmo che identifica tutti i pixel di un immagine che corrispondono a quel colore e che vengono generalmente sottoposti a clustering per identificare le zone di maggiore concentrazione. Approcci più moderni e precisi invece prevedono di fornire in ingresso un istogramma colore in modo da poter utilizzare più colori persenti in percentuali diverse, per ottenere una identificazione più precisa.

A causa della sua semplicità questo metdodo di identificazione presenta numerose limitazioni: infatti è necessario che il colore (o i colori) dell'oggetto da identificare siano differenti rispetto a quelli presenti nell'ambiente circostante. Inoltre le variazioni di luce influenzano negativamente questi algoritmi; ad esempio una superfice sotto la luce diretta del sole potrebbe apparire bianca. Per ovviare, almeno parzialmente, a quest'ultimo problema, per l'identificazione del colore si utilizza lo spazio HSV invece del normale RGB (o BGR).

2.6.1 Spazio dei colori HSV

Il metodo più comune di rappresentazione di un colore su computer consiste nello scomporlo in tre colori fondamentali, rosso, verde e blu, il cui valore varia tra 0 e 255 e sono memorizzati in un byte. Nonostante la sua diffusione questo non è il sistema più comodo per l'elaborazione digitale delle immagini. Ad esempio l'identificazione della tonalità o della luminosità di un pixel risulta complessa e inoltre richeide l'accesso a tutte e tre le componenti del colore (lo stesso discorso vale anche per la modifica di un pixel).

Un modo alternativo e più comodo (sopratutto per la computer vision) di memorizzazione delle informazioni sul colore è la sua rappresentazione in HSV (Hue, Saturation, Value). L'Hue indica la tonalità del colore e il suo valore varia tra 0 (rosso) e 360 (di nuovo rosso); Saturtion, o saturazione, del colore ne indica l'intensità e varia tra 0 e 255; invfine Value indica la luminosità del colore e anchesso varia tra 0 e 255.
Tipicamente lo spazio HSV viene rappresentato mediante un cono o un cilindro (Figura 2.4).

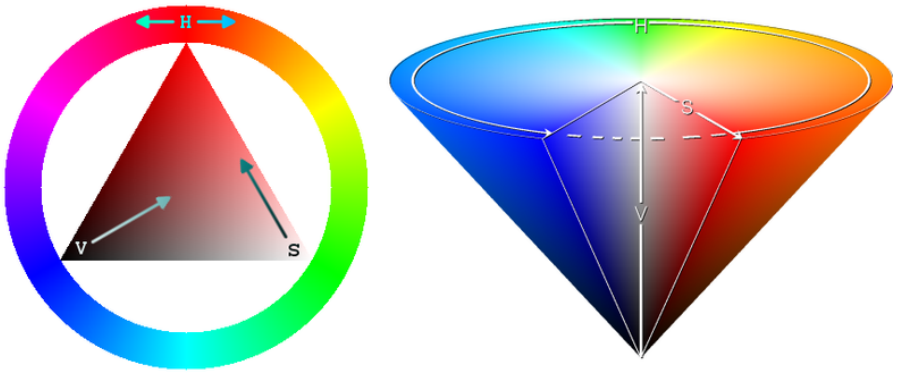

Figura 2.4: Tipica rappresentazione dello spazio colore HSV

2.7 Algoritmi di classificazione

2.7.1 K Nearest Neighbor

Il SURf si occupa solamente di identificare i punti interessanti in una immagine, ma per poterlo utilizzare nel riconoscimento di un oggetto è necessario ricorrere a un algoritmo in grado di classificare i punti trovati e metterli in corrispondenza con quelli del modello dell'oggetto che interessa trovare. Per raggiungere questo scopo si è deciso di utilizzare l'algoritmo K-Nearest-Neighbor (KNN) [16].

Il K-Nearest-Neighbor è un algoritmo utilizzato nel riconoscimento di pattern e per la classificazione di oggetti che si basa esclusivamente sulle caratteristiche degli oggetti vicini a quello considerato; è infatti il più semplice tra gli algoritmi utlizzati nel campo del machine learning.

Nella prima fase, detta di addestramento, si fornisce all'algoritmo un insieme di punti, considerati come vettori in uno spazio multidimensionale, che appartengono alle varie classi in cui vogliamo suddividere gli oggetti e il valore k: un numero intero, in genere piccolo. Nella seconda fase si fornisce in ingresso un vettore contente dei punti che vengono classificati tramite votazione a maggioranza dei suoi k vicini. Per calcolare quali sono i punti vicini a quello considerato viene solitamente utilizzata la distanza euclidea.

Inoltre il K-Nearest-Neighbor è sensibile alla struttura dati in cui sono memorizzati i punti.

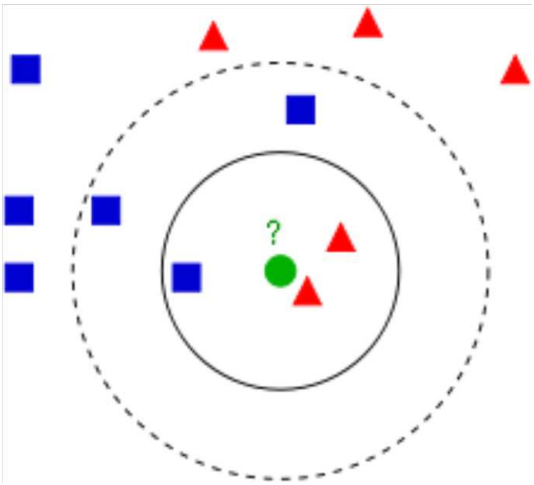

Figura 2.5: Esempio di knn. Se $k = 3$ il punto verde viene assegnato alla classe dei triangoli, se $k = 5$ a quella dei quadrati

2.8 Altri sensori

Solitamente sui robot vengono utilizzati diversi tipi di sensori in modo da permettere al robot di muoversi agevolmente nell'ambiente circostante. Nel nostro caso, alla visione sono stati affiancati quattro sensori sonar che formano una cintura intorno al robot permettendogli di rilevare ostacoli.
I dati rilevati sono inviati al computer tramite una coppia di trasmettitori che utilizzano il prtocollo zigBee [5], [4].

2.8.1 Sonar

I sonar, inizialmente nati per l'utlizzo su navi e sottomarini, hanno trovato un largo uso in robotica grazie al basso costo e a una discreta precisione. Nella loro versione attiva inviano un'onda ultrasonica che viene riflessa dagli oggetti circostanti. Una volta emesso il segnale, il sonar resta in attesa dell'eco e, conoscendo il tempo di propagazione e la velocità del suono nel mezzo (di solito aria), viene calcolata la distanza.

I sonar utlizzati hanno un range operativo che va da circa 20 cm fino a 7 metri, tuttavia vengono utilizzati sopratutto come sensori di prossimità, nel nostro caso fino a circa 2 metri, infatti siccome coprono un un angolo di circa 30° gradi e restituiscono la distanza del primo oggetto che riflette l'onda sonara, c'è il rischio che venga rilevato un oggetto molto fuori asse rispetto al sonar ottenendo in questo modo false letture.

Bisogna inoltre evidenziare che, nonostante la buona precisone, le letture sono spesso soggette a rumore.

2.9 Logica Fuzzy

La logica fuzzy, sfumata, per un'introduzione a essa si veda [8], è un tipo di logica in cui le funzioni, a differenza della logica booleana in cui assumono solo il valore 0 (falso) e 1 (vero), possono assumere un grado di verità compreso tra questi due estremi. Grazie a questa caratteristica la logica fuzzy ha trovato largo impiego nell'intelligenza artificiale e nella robotica, in quanto permette di combinare i risultati di diverse funzioni in base al loro valore di verità. In particolare, in robotica si può ad esempio, considerare ogni funzione un comportamento che dipende da input sensoriali e combinare il valore assunto dalle funzioni in modo da ottenere comportamenti complessi dipendenti da quelli più semplici.
Per raggiungere questo scopo si è deciso di utilizzare Mr. BRIAN.

2.10 MrBrian

Mr. BRIAN (Multilevel Ruling BRIAN) [9] è un software, estensione di BRIAN (BRIAN Reacts by Inferring ActioNs)[10], sviluppato all'interno del progetto MRT (Milano Robocup Team) del Politecnico di Milano. Mr. BRIAN permette di sviluppare i comportamenti del robot definendoli come set di regole fuzzy attivabili in basi a valori forniti in ingresso e producendo in uscita valori che vengono generalmente utilizzati dagli attuatori del robot o da altro software di controllo. In particolare Mr. BRIAN, a differenza di BRIAN, permette di definre i comportamenti strutturati gerarchicamente in base all'importanza che si vuole dare a ciascuno di essi.

L'intero processo che dai dati di input porta ad ottenere i valori in uscita può essere suddiviso in sei fasi illustrate qui di seguito (Figura 2.6).

2.10.1 Fuzzyficazione

Il primo passo da fare è definire quali sono i dati in ingresso e a che tipo di dato fuzzy vanno associati. Per poter definire il tipo di dato BRIAN mette a disposizione alcune forme fuzzy utilizzabili per questo scopo (Figura 2.7);

Per definire il tipo fuzzy si utilizza la seguente sintassi

```
(DISTANCE
(TRA (CLOSE 0 0 30 50))
```

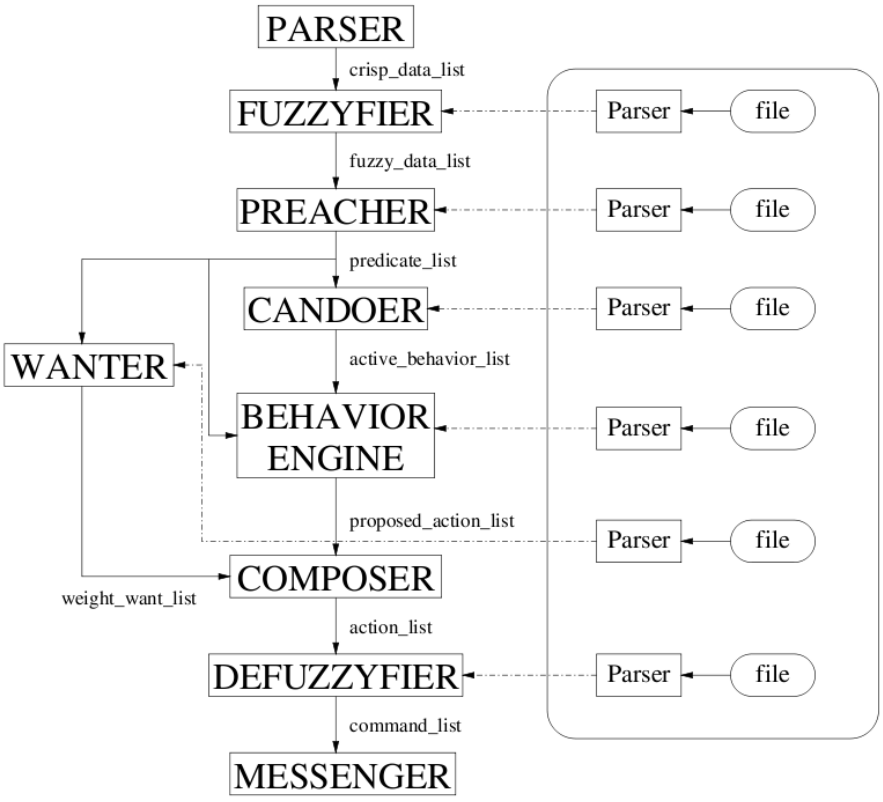

Figura 2.6: Schema di funzionamento di Mr. BRIAN

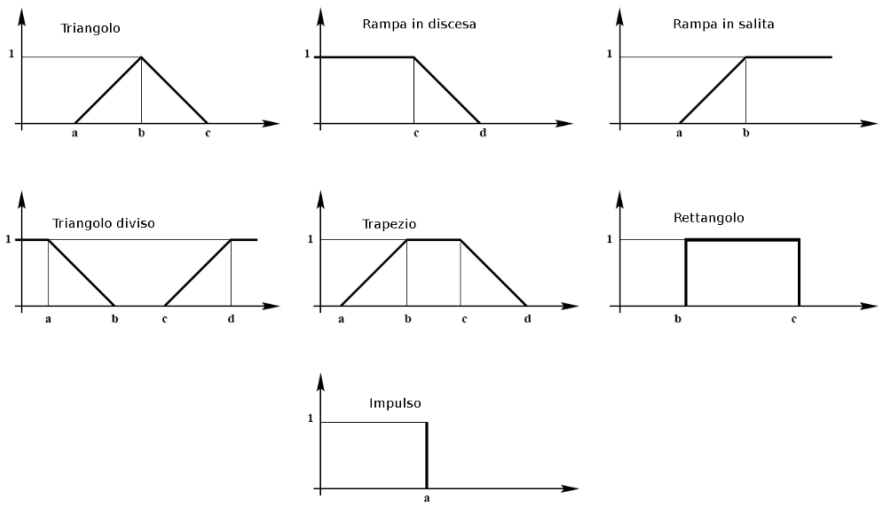

Figura 2.7: Le forme fuzzy utilizzate da Mr. BRIAN

```
(TRA (NEAR 0 30 150 200))
(TOR (FAR 150 200))
)
```

che definisce il tipo di dato DISTANCE (Figura 2.8); CLOSE, NEAR e FAR sono dette etichette.

A questo punto si possono definire le variabili di tipo DISTANCE in questo modo: (SonarFrontDistance DISTANCE).

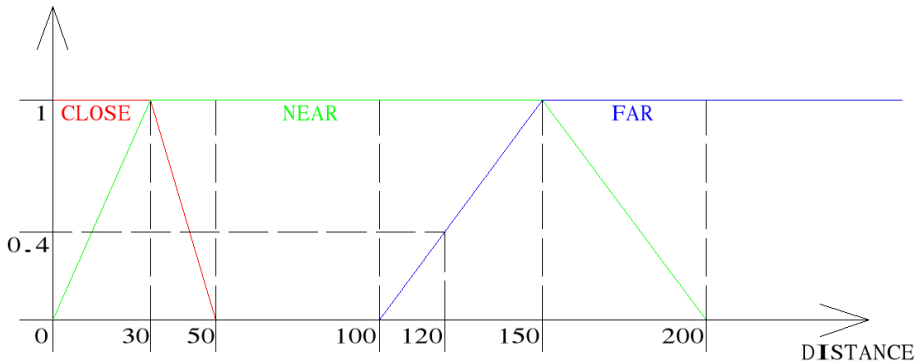

Figura 2.8: Il tipo di dato DISTANCE

2.10.2 Definizione dei predicati

Il modulo Preacher si occupa della creazione dei predicati che sono la composizione di variabili fuzzy o di altri predicati definiti precedentemente e che assumono un valore di di verità compreso tra 0 e 1 che varia in base al grado di apparteneza delle variabili fuzzy al loro insieme e al valore degli eventuali altri predicati associati a quello in considerazione.
Le variabili e i predicati vengono composti utilizzando i classici operatori booleani che in logica fuzzy hanno le seguenti definizioni:

- $NOT(X) = 1 - X$
- $AND(X, Y) = max(X, Y)$
- $OR(X, Y) = min(X, Y)$

Ad esempio:

```
FrontFree = (D SonarFrontDistance FAR);
```

definisce il predicato FrontFree che dipende dal grado di appartenenza della variabile SonarFronDistance al set fuzzy FAR. Se, ad esempio, SonarFrontDistance valesse 120 FrontFree assumerebbe valore 0.4. Allo stesso tempo SonarFrontDistance avrebbe grado di appartenenza 0 all'insieme CLOSE e 1 all'insieme NEAR, Figura 2.8.

2.10.3 Attivazione dei comportamenti (CANDO)

Il modulo CANDO di Mr. BRIAN è quello che si occupa di stabilire quali comportamenti, sempre attraverso mediante l'utilizzo di regole fuzzy, possano essere attivati in un determinato istante. In questo modo si evita che siano attivati comportamenti che in un determinato contesto non avrebbero senso. La sintassi per la definizione delle regole CANDO è la stessa usata per i predicati, con la differenza che solo questi ultimi possono essere usati per comporre le espressioni. Ad esempio:

```
Escape = (OR (P GameStatusEscape)
(OR (P IRExists)(P PlayerDetected)));
```

dove Escape è il nome di un comportamento.

2.10.4 Behavior engine

Il Behavior engine è la parte principale di Mr. Brian, si occupa infatti di attivare i comportamenti se il valore delle condizioni di CANDO a cui ciascuno è associato è superiore a una certa soglia fissata dall'utente. I vari

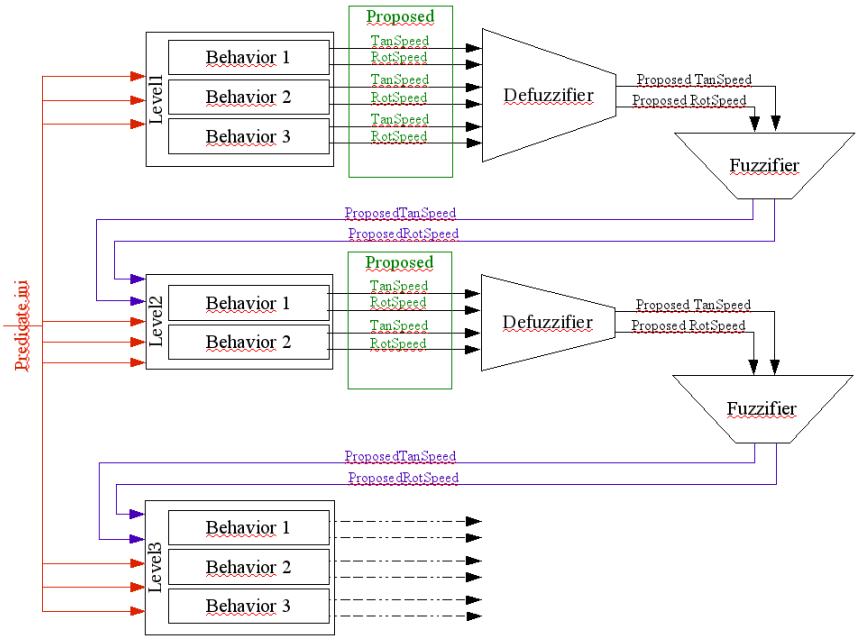

Figura 2.9: Il Behavior engine multilivello di Mr. BRIAN

comportamenti sono specificati nell'apposito file nella forma *(level n NomeComportamento percorso/file.rul)* dove NomeComportamento deve essere lo stesso utilizzato nei file del CANDO e di WANT, mente level n specifica il livello nel quale si vuole inseirire il comportamente.

La disposizione gerarchica dei comportamenti è la caratteristica che distingue Mr. BRIAN da BRIAN. In questo modo è possibile dare maggiore priorità ad alcuni comportamenti evitando che si verifichino conflitti in quanto i comportamenti di livello più alto possono annullare i comandi (variabili fuzzy) in uscita dai comportamenti inferiori e sostituirli con i propri. Inoltre ogni comando in uscita da dai comportamenti di un livello può essere utilizzato dai livelli successivi definendo dei particolari predicati che devono chiamarsi come la variabile in uscita preceduti dal suffiso prop. (Figura 2.9) Nel nostro caso, ad esempio, i comportamenti TraiettoryCorrection e Safe, che si occupano di far evitare a Spykee gli ostacoli, sono al livello più alto e possono cosi annullare ogni altro comportamento attivo.

2.10.4.1 Comportamenti

Ogni comportamento è specificato in un proprio file al cui interno sono elencate una serie di espressioni nella forma:

```
(antecedente) => (var1 valore1)(var2 valore2)...
```

dove *antecedente* indica uno o più predicati (eventualmente collegati tra loro dagli operatori AND, OR e NOT), mentre nella parte destra vi è un elenco di variabili con il rispettivo valore, inoltre a ogni variabile viene associato un grado di importanza pari al valore dell'antecedente. Queste variabili sono anch'esse di tipo fuzzy e sono definite in modo analogo a quanto visto nella sezione 2.10.1 con la differenza che gli insiemi fuzzy possono solo essere composti solo da singleton (Figura 2.7).
Ogni regola può specificare un valore diverso per la stessa variabile d'uscita ognuna con la sua importanza, tutte le variabili con lo stesso nome verrano poi fuse insieme nel passaggio successivo.

2.10.5 WANTER

Il WANTER agisce in modo analogo al modulo CANDO, infatti definizione dei predicati WANT e loro interpretazione è identica.
A differenza del CANDO che stabilisce quali comportamenti possano essere attivati questo modulo stabilisce quali vogliano essere attivati e i valori in uscita dalle equazioni di want vengono utilizzati per pesare le variabili ottenute dai comportamenti attivi valutati dal Behavior engine.

2.10.6 Fusione dei risultati (Composer)

Il Composer si occupa di semplificare le variabili ottenute dal Behavior engine ed agisce in due fasi: nella prima fase prende tutti i valori in uscita da un singolo comportamento e li pesa utilizzando il valore del WANTER relativo a quale comportamento. Nella seconda fase raggruppa tutte le variabili e con la stessa etichetta (valore fuzzy) e ne fa la media, in modo da avere un unico valore in uscita per ogni coppia variabile-etichetta.

2.10.7 Defuzzyficazione

L'ultimo modulo di Mr. BRIAN è quello che si occupa di convertire le variabili in uscita da fuzzy a valori numerici. Le variabili fuzzy sono tutte di tipo singleton (impulsivo) quindi la trasformazione in valori reali consiste nel calcolare la media di tutti i valori associati a una variabile pesati con l'importanza di ogni etichetta.

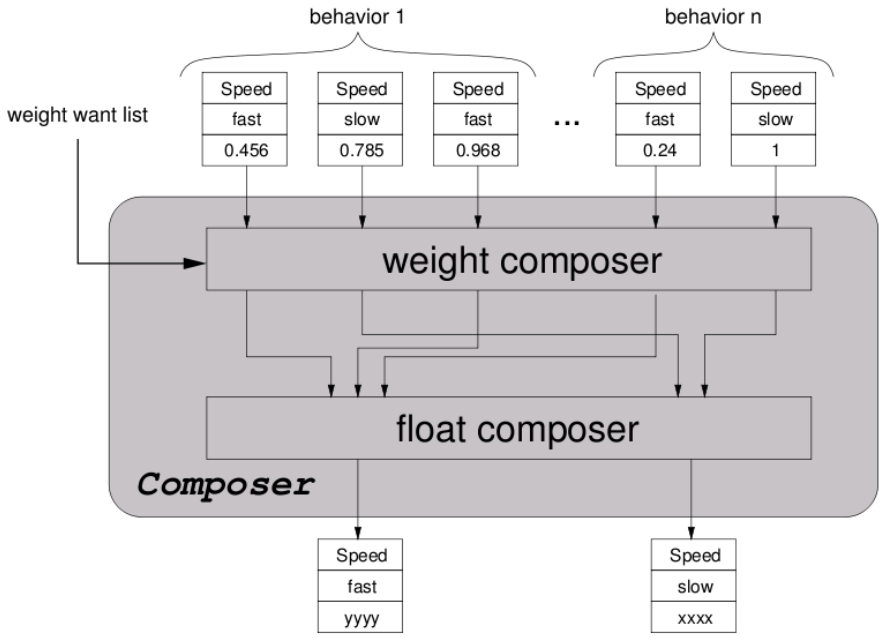

Figura 2.10: Struttura del modulo Composer

2.11 La classe MrBrian

La classe C++ MrBrian incapsula tutti i passaggi descritti precendentemente. Quando un istanza di tale classe viene creata nel costruttore vengono passati i percorsi ai file che contengono le forme fuzzy, le variabili, i predicati, ecc...
Una volta creata la classe bisogna impostare i valori delle variabili in ingresso, attraverso *MrBrian::getFuzzy::get_crisp_data_list* che restituisce una lista a cui si possono aggiungere oggetti del tipo *crisp_data* che contengono il nome della variabile, il suo valore e l'indice di affidabilità. A questo punto richiamando il metodo *MrBian::run* Mr. Brian viene eseguito e calcola i valori in uscita, a cui si accede tramite *MrBrian::getFuzzy::get_command_singleton_list*. Quando si è concluso di utilizzare MrBrian, o se lo si vuole eseguire nuovamente, è necessario richiamare il metodo *flush* per resettarlo.

2.12 DCDT (Device Communities Development Toolkit)

Un robot autonomo è un sistema molto complesso dove diverse componenti software e hardware lavorano in parallelo e interagiscono tra loro per permetterne il corretto funzionamento, nel nostro caso questo compito è svolto dal DCDT: un software, sviluppato dall'Airlab all'interno del progetto MRT.

Il Device Communites Development Toolkit è un software orientato alla gestione degli agenti. Un agente (o expert) può essere considerato come un'entità astratta che svolge un determinato compito.
DCDT permette l'eseguizione parallela di più agenti e consente a questi di comunicare tra loro rendendo gli aspetti legati al multithreading e alla comunicazione tra più thread completamente trasparenti all'utente, permettendogli di concentrarsi sulla scrittura del codice necessario al funzionamento del robot.

Fin ora abbiamo perso in cosiderazione l'eseguzione di una singola istanza di DCDT su una macchina, tuttavia è anche possibile eseguire più istanze (anche con agenti diversi) su più macchine collegate in LAN. Questo permette a DCDT di coordinare e controllare anche diversi robot, sempre mantenendo gli aspetti legati alla comunicazione e alla sincronizzazione trasparenti all'utente.
Di seguito si considererà sempre una singola istanza di DCDT in eseguzione su una sola macchina.

2.12.1 Dispatcher

DCDT è un sistema publish-subscribe e il suo cuore è il dispatcher, o Agorà. Ogni agente è in grado di inviare messaggi, ognuno dei quali ha un determinato tipo. Questi messaggi sono ricevuti dal dispatcher che si preoccupa di distribuirli a tutti gli agenti che si sono sottoscritti alla ricezione di uno o più messaggi. Grazie al dispatcher si evita che lo stesso messaggio venga replicato più volte per inviarlo ad agenti (e quindi a thread) differenti; inoltre ogni agente è dotato di una prorpia coda dei messaggi, in questo modo si assicura che tutti i messaggi inviati arrivino a destnazione, sarà poi compito dell'utente gestire la coda dei messaggi attraverso le funzioni della classe base da cui derivano tutti gli agenti.

Il Dispatcher per funzionare deve, ovviamente, essere inizializzato correttamente: per prima cosa bisogna istanziare un oggetto di tipo *ModuleDispatcher* che come parametro del costruttore accetta un file di configurazione che permette di impostare se DCDT deve lavorare in locale o su rete.

Una volta creato il dispatcher vanno aggunti e attivati i vari agenti, rispettivamente attraverso le funzioni *AddMemeber* e *ActivateMemeber*. Infine per avviare DCDT si richiama la funzione *LetsWork*.
Per una guida completa a DCDT si veda [12], mentre per degli esempi si faccia riferimento sempre a [12] o Appendice B.

2.12.2 Agenti e messaggi

Ogni agente è in realtà una classe C++ ereditata da *StringModuleMember* che a sua volta ha come classe base *DCDT_Memebr*. Ogni agente deve implementare tre funzioni astarette che sono:

- *RunInit*: questa funzione, come dice il nome stesso, viene richiamata all'avvio dell'agente e viene comunemente usata per le inizializzazione. Solitamente è qui che viene richiamata la funzione *ReceiveMessage* per sottoscriversi alla ricezione di un tipo di messagio.
- *RunDuty* è la funzione principale dell'agente, viene eseguita in modo ciclico con intervalli regolari definiti dall'utente in fase di creazione della classe (se l'utente imposta 0 il metodo sarà eseguito in modo continuo).
- *RunClose* viene invece richiamata alla terminazione dell'agente. Qualunque agente può termirare l'eseguzione di DCDT richiamando il metodo *Shutdown*.

2.12.2.1 Messaggi

DCDT non pone vincoli al payload dei messaggi, che sono tratti come dati binari, e non espone funzioni ne per il parsing ne per la creazione degli stessi.

Ogni agente può ricevere i messaggi a cui è registrato utilizzando diverse funzioni della classe base, nel nostro caso è stata usata *ReceiveLastMessage* che preleva il primo messaggio ricevuto in ordine temporale. La funzione restituisce un puntatore all'area di memoria che contiene il mesaggio e può essere utilizzata sia in modalità bloccante che non bloccante.

Proprio a casusa delle difficoltà nella creazione e analisi dei messaggi è stato scritto un middelware che si occupa di questi compiti: la classe StringModuleMemeber. Questa classe permette la creazione di messaggi testuali con una sintassi XML-like:

```
<MESSAGE id:ID_AGENTE sender:INDIRIZZO_IP_AGENTE
timestamp:MICROSECONDI module: NOME_AGENTE>
```

Dove id, sender e module si riferiscono all'agente che ha creato e inviato il messaggio.
Per la creazione si utilizza la funzione *NewMessage*, mentre per insiserire dati nel messaggio:

```
mCurrentMessage << "<C>" << valore << "</C>";
```

Infine per la chiususra e l'invio del messaggio si usano rispettivamente: *CloseMessage* e *SendAllStringMessages*

2.13 Librerie software

2.13.1 OpenCV

OpenCV [13] è una librera opensource sviluppata e mantenuta da Intel (compatibile infatti con la libreria commerciale IPL, Image Processing Library) per la visione artificiale. Le funzioni disponibili sono molto varie e comprendono: eleborazione di immagini, corner e features detection, optical flow, motion tracking, calibrazione e ricostruzione 3D, identificazione di features haar e molte altre. Sono anche presenti diverse funzioni per il calcolo geometrico e algebrico.

Con il recente rilascio della versione 2.0 è stata introdotta l'interfaccia C++ che permette un utlizzo più rapido della libreria, inoltre esistono binding per python e per i linguaggi .NET.

Le OpenCV sono pensate sopratutto per un utilzzo real time, infatti le funzioni sono pesantemente ottimizzate e hanno le migliori prestazioni su hardware Intel sfruttando il multithreading e i set di istruzioni SSE e MMX. Inoltre è possibile rendere ancora più veloce questa libreria utilizzando le Intel IPP (Integrated Performance Primitives).

2.13.2 Wiiuse

WiiUse è una libreria opensoruce sviluppata per permettere l'utilizzo del controller WiiMote indipendentemente dalla sua console. Sono scritte in C, utilizza un solo thread e le sue funzioni non sono bloccanti. Allo stato attuale le Wiiuse sono in grado di gestire completamente il controller Wiimote standard permettendo l'accesso all'accelerometro, infrarossi e a tutti i suoi tasti. Inoltre già supportano alcune estensioni del controller. Questo supporto verrà ulteriormente esteso nelle prossime versioni. Per la documentazione completa si veda [3], mentre per esempi del suo utilizzo si veda l' Appendice B o [7].

Capitolo 3

RoboWii 2.0.1

"HAL: I'm sorry, Frank, I think you missed it. Queen to Bishop 3, Bishop takes Queen, Knight takes Bishop. Mate.
Dr. Frank Poole: Huh. Yeah, it looks like you're right. I resign.
HAL: Thank you for a very enjoyable game.
Dr. Frank Poole: Yeah, thank you."

2001: A space odissey

3.1 Il gioco: Hide and Seek

Il gioco implementato consiste in una versione modificata del classico gioco del nascondino: Spykee deve scappare e nascondersi mentre il giocatore deve trovarlo e colpirlo alle spalle puntandolo per alcuni secondi con il Wiimote e poi premendo il tasto di fuoco. Una volta avviato il gioco Spykee si mette alla ricerca del giocatore ruotando su se stesso ed eventualmente allontanandosi da oggetti troppo vicini in modo da avere un campo visivo maggiore. La seconda fase del gioco, la fuga, inizia se Spykee viene puntato oppure se vede il giocatore: nel primo caso i dati di puntamento sono ottenuti dal sensore infrarosso del Wiimote, mentre nel secondo caso sono ottenuti dall'analisi delle immagini inviate dalla telecamera di Spykee. Durante la fuga Spykee procede principalmente in linea retta, evidando però gli ostacali e girando bruscamente se rileva di essere nuovamente puntato o entra in contatto visivo con l'utente. Questa fase può finire in due modi: se per 10 secondi (o un altro valore impostato prima dell'inizio del gioco) non è nuovamente puntato torna in modalità ricerca, oppure alla rilevazione di un nascondiglio.

Per nascondiglio si intende una cavità rilevata dal sonar destro o da

quello sinisto che, essendo il gioco pensato per un ambiente domestico, può essere una porta o la fine di un mobile. Una volta rilevato il nascondiglio Spykee vi si dirige e una volta giunto ricomincia a cercare il giocatore. Il robot si considera nascosto se mentre sta raggiungendo il nascondiglio rileva, a destra o a sinistra, un ostacolo molto vicino. Se dovesse superare la copertura fornita dall'ostacolo, trovandosi potenzialmente in vista, invece di tornare in modalità ricerca Spykee ricomincerà a scappare alla ricerca di un nascondiglio migliore.

Quando il giocatore mette a segno un punto il gioco si interrompe e riprende dopo alcuni secondi con Spykee che torna a cercare il giocatore. Il punteggio è indicato dai quattro led gialli montati sul robot, mentre i led rossi indicano il livello di carica del colpo (valore mostrato anche dai led blu del Wiimote), infine il led verde indica se Spykee è puntato o no. Una volta raggiunti tre punti il gioco si ferma con la vittoria dell'utente ed è necessario farlo ripartire manualmente.

Inoltre bisogna ricordare che per essere individuato da Spykee il giocatore deve necessariamente indossare un marker, possibilmente rigido, sul petto.
Inoltre è possibile controllare direttamente Spykee tramite il Wiimote, registrare video o fare foto.

3.2 Classe Spykee

La classe C++ Spykee, inizialmente sviluppata per [15] è stata modificata per adattarla alle caratteristche di questo lavoro. In particolare è stato totalmente riscritto il codice per l'acquisizione dell'immagine da Spykee rendendolo sincrono con il thread di gestione della visione artificiale (sezione 3.4.1).
Per ottenere un'immagine dal robot bisogna utilizzare la funzione *getImage* che internamente agisce come una macchina a stati: nel primo stato, *WaitForImage*, la funzione analizza i dati ricevuti da Spykee ricercando l'header di un immagine jpeg: se lo trova alloca i buffer per la ricezione dell'immagine e si porta nello stato *Processing*, altrimenti analizza il pacchetto dati successivo. Nello stato *Processing* viene ricevuta e memorizzata il resto dell'immagine fino al raggiungimento del trailer jpeg, infine la funzione restituisce il buffer contente l'immagine.

L'utente non si deve preoccupare di deallocare questo buffer in quanto allocazione e deallocazione sono gestiti internamente dalla classe; inoltre se si vuole conservare i dati ricevuti è necessario copiare il buffer, in quanto una successiva chiamata alla funzione sovrascrivera i dati in esso contenuti.

3.3 Rilevamento marker

La capacità del robot di rilevare la presenza del giocatore è una parte fondamentale del gioco, in questo modo Spykee è in grado di reagire prima che l'avversario cerchi di colpirlo.

Durante la fase preliminare di questo lavoro sono state considerate diverse soluzioni, le principali illustare nella Sezione 2.4. Alla fine si è deciso di usare il SURF come cuore dell'algoritmo di ricerca del marker integrandolo con l'utilizzo di più modelli e di un parziale filtraggio del colore per ovviare ad alcuni limiti della telecamera.

Di seguito vengono discussi i passi principali della classe *SURFModule* che si occupa della rilevazione del marker. Per il codice completo si veda l'Appendice B.

3.3.1 I modelli

La prima fase consiste nell'acquisizione dei modelli del marker. Allo stato attuale i modelli vengono caricati nella funzione *RunInit* della classe *VisionExpert* e sono delle immagini precedentemente salvate su disco acquisite utilizzando Spykee e rappresentati il marker fotografato a diverse angolazioni e sopratutto a diverse distanze; le immagini sono poi ritagliate e adattate utilizzando un software di photo editing (in questo caso The Gimp).

Come descritto nella Sezione 2.5 il SURF è invariante rispetto a rotazioni e alle variazioni di scala, tuttavia durante le prove del sistema di visione si è osservato che utilizzando un solo modello al crescere della distanza tra Spykee e il giocatore SURF non riesce più a riconoscere il marker. Questo è dovuto sopratutto al fatto che la qualità delle immagini ricevute da Spykee è molto bassa a causa della elevata compressione JPEG che queste ultime subiscono per ottimizzare la trasmissione wireless. Utilizzando più modelli del marker a diverse distanze si riesce invece ad ottenere un riconoscimento piu accurato e a distanze maggiori che tuttavia non supera i 2-3 metri.

Per ogni immagine caricata vengono estratti i punti SURF (coordinate, orientazione, dimensione) con i loro descrittori utilizzando l'implementazione presente nelle OpenCV dell'algoritmo (in particolare la funzione è *cvExtractSURF*), inoltre viene calcolato l'istogramma colore sull'intera immagine. Queste informazioni sono memorizzate in una struttura dati, i cui campi principali sono:

- pImage: il puntatore alla struttura dati OpenCV IplImage che contine effettivamente l'immagine e le sue caratteristiche.

- objKeypoints: è un vettore OpenCV che contiene tutti i punti SURF rilevati nell'immagine
- objDescriptors: vettore OpenCv che contiene i descrittori dei punti.
- indices, distances: sono due matrici utilizzate durante la fase ti matching come contenitori per i dati di uscita dell'algorirmo KNN.
- pHist: il puntatore all'istogramma colore dell'immagine.

Ogni struttura viene poi salvata in un vector delle librerie STL per poter essere utilizzata durante la fase di riconoscimento del marker.

3.3.2 Acquisizione e conversione dell'immagine

Una volta completate le operazioni di acquisizione dei modelli la classe *VisionExpert* entra nel loop di gioco vero e proprio. A ogni iterazione la prima cosa che viene fatta è l'acquisizione dell'immagine inviata da Spykee tulizzando la funzione *getImage* della classe Spykee che restituisce un vettore di byte contente l'immagine jpeg ricevuta. Questa viene poi convertita, tramite la funzione *cvDecodeImage*, in una struttura IplImage contenente l'immagine decompressa in formato RGB.

3.3.3 Filtraggio dei colori ed estrazione dei punti SURF

Dopo aver ottenuto l'immagine da Spykee questa viene filtrata utilizzando l'istogramma colore con una tecnica detta di backprojection. L'istogramma viene calcolato sulla base dell'ultimo riscontro positivo con un marker, come spiegato nella sezione 3.3.5, ed è utilizzato per creare il frame di backproject: un immagine a un solo canale in scala di grigi (della stessa dimensione dell'immagine originale) in cui ogni pixel è tanto più bianco quanto più il pixel corrispondente nell'immagine originale appartiene alla gamma di colori dell'istogramma. Al backproject viene poi applicata una sogliatura in modo da rendendolo binario e utilizzare come maschera che viene messa in AND con l'immagine acquisista da Spykee. I pixel dell'immagine risultante assumeranno il valore corrispondente a quelli del fotogramma originale dove la maschera è bianca, mentre gli altri assumeranno colore nero (Figura 3.1) e verranno quiondi ignorati dal SURF. Il valore di soglia è un parametro critico in quanto determina quanto viene tenuta in considerazione l'informazione sul colore. Inoltre bisogna precisare che fino a quando il robot non ha un primo riscontro positivo con uno dei modelli salvati o se è passato troppo tempo dall'ultimo riscontro, non viene utilizzata alcuna maschera.

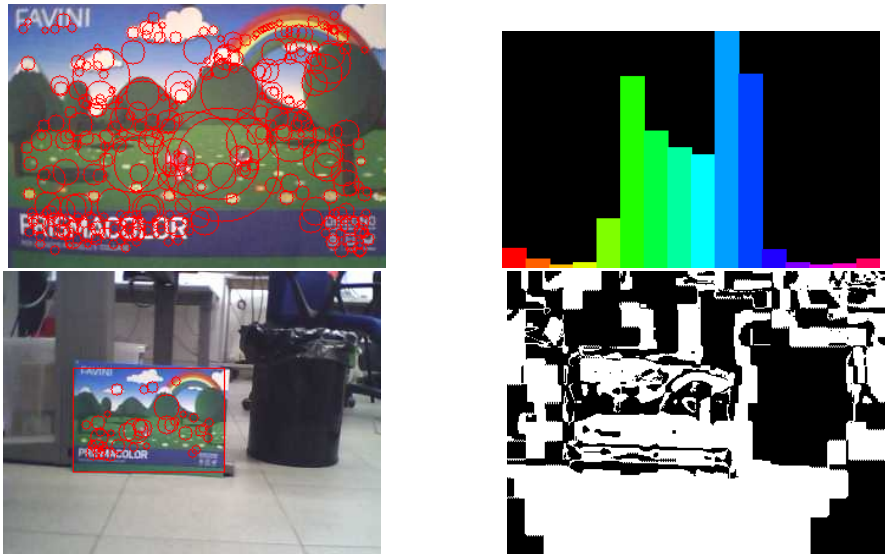

Figura 3.1: Istogramma colore, maschera, modello e immagine di gioco con evidenziati i punti SURF

L'immagine filtrata con la maschera viene poi convertita in scala di grigi, il SURF infatti non utilizza informazioni sul colore, e la si passa alla funzione OpenCV *cvExtractSURF* che restituisce due vettori contenenti i punti caratteristici e i loro descrittori.

3.3.4 K-Nearest-Neighbor

A questo punto inizia la fase di identificazione vera e propria del giocatore: i tentativi di riconoscimento del marker vengono effettuati su un modello alla volta e vi sono diverse fasi.

La prima consiste nell'utilizzare il K-Nearest-Neighbor per classificare i punti SURF ottenuti dal fotogramma corrente. Per fare questo si istanzia la classe *Index* del namespace *flann* delle OpenCV il cui costruttore accetta come parametro una matrice I contente i descrittori dei punti SURF che vengono indicizzati internamente utilizzando strutture dati che rendono la succesiva ricerca molto efficente. Proprio per questo motivo questa è l'operazione computazionalmente più onerosa dell'intero algoritmo. In particolare la matrice I ha dimensione $n.descrittoriX dimensionedescrittori$ (nel SURF i descrittori sono vettori a 6 dimensioni [6]).

Una volta creato l'indice dei punti si può effettuare la ricerca di quelli che corrispondono alle features individuate sul marker utilizzando la funzione *knnSearch*. I paramentri più importanti sono: la matrice Q contente la query di ricerca (in questo caso, i descrittori SURF del marker) che ha di-

mensione verticale qualsiasi, mentre deve avere lo stesso numero di colonne di I; il secondo paramentro è una matrice che contine per ogni i-esima riga gli indici dei k punti più vicini al punto contenuto alla riga i-esima della matrice I; il terzo parametro è una matrice identica alla precedente ma contenente le distanze dei punti, mentre il quarto parametro è il valore di K nell'algoritmo (si veda la sezione 2.7, mentre per la documentazione esatta della libreria flann si veda [13]).

Una volta ottenuti tutti i K vicini per ogni punto della query, viene preso il più vicino e messo in corrispondenza col relativo punto del marker. Ogni coppia di punti viene memorizzata in modo da poter passare alla fase successiva.

3.3.5 Identificazione marker

Una volta trovati i punti che più probabilmente appartengono al marker (il KNN è un algoritmo probabilistico) bisogna verificare che questo sia effettivamente inquadrato ed individuarne la posizione nell'immagine.
La prima e più importante verifica consiste nel trovare l'omografia tra i punti del modello e quelli ottenuti dal passaggio precedente.

Per omografia si intende una trasformazione invertibile tra punti di due spazi tale che ogni punto presente in uno spazio corrisponde solamente a un punto del secondo spazio. In particolare nella computer vision due immagini che inquadrano uno stesso piano da due angolazioni diverse sono legate tra loro da una relazione di omografia. Le OpenCV mettono a disposizione una funzione, *cvFindHomography* che permette di trovare l'omografia tra due insieme di punti e che restituisce la matrice di trasformazione. Utilizzando questa matrice e le dimensioni dell'immagien del modello (in pixel) è possibile calcolare la posizione dei vertici che racchiudono l'area del marker.

A questo punto vengono effettuate altre due verifiche: prima di tutto si verifica che i vertici trovati siano effettivamente quattro e che la forma dell'area del marker coincida con quella di un rettangolo. Queste due ulteriori verifiche si sono rese necessario in quanto la ricerca dell'omografia è approssimata e quindi può restituire valori errati. Una volta superati tutti i test viene effettuato un ulteriore controllo prima di considerare il giocatore come inquadrato: il marker deve rimanere visibile per 3 frame. Questo permette di ridurre ulteriormente i falsi positivi, inoltre permette di variare la reattività di Spykee alla vista del giocatore.

Una volta individuata con precisione l'area dell'immagine che contiene il marker viene ricalcolato l'istogramma colore del marker appena individuato, in modo da rendere l'algoritmo più robusto alle variazioni di luminosità.

L'istogramma viene memorizzato nella struttura dati del modello corrente del marker e verrà utilizzato nel ciclo successivo per filtrare l'immagine. Se invece il marker non viene inquadrato non si effettua nessun aggiornamente dell'istogramma. A questo punto vi sono due possibilità: se il marker è stato individuato si inviano le informazioni di visione al modulo BrianExpert e si esce dal ciclo di controllo dei modelli, altrimenti si passa al modello successivo.
Una volta individuato il giocatore viene calcolato l'angolo con cui viene visto da Spykee, consideranto il centro dell'immagine come la direzione di movimento del robot. Si è misurato che l'ampiezza del campo visivo della telecamera è di circa 50°, quindi effettuando una semplice proporzione tra la dimensione in pixel dell'immagine e la posizione del marker si ottiene l'angolo del giocatore.

3.4 DCDT

Il software di controllo del robot si basa sul farmework DCDT (sezione 2.12) che permette di eseguire i diversi moduli parallelamente; in questo modo diversi parametri, come le distanze misurate dai sonar o le informazioni ottenute tramite visione, possono arrivare senza grossi ritardi al modulo che gestisce Mr. BRIAN in modo che i comportamenti generati da BRIAN corrispondano allo stato attuale del gioco.

I vari moduli, spiegati nelle sezioni seguenti, sono creati e inizializzati nel file *kernel.cpp* che contiene la routine *main*. Qui vengono creati i vari expert e viene inizializzata la classe Spykee, di cui esiste una sola istanza condivisa tra gli expert che la utilizzano.
I principali segnali scambiati dai moduli (Figura 3.2) sono:

- MSG_TO_MOTION: contiene le velocità a cui impostare i motori.
- MSG_FROM_WIIMOTE: contiene le informazioni sul puntamento del robot da parte del giocatore.
- MSG_TO_SONAR: contiene i comandi per accendere o spegnere i led di Spykee
- MSG_FROM_SONAR: contiene le distanze rilevate dai sonar.
- MSG_FROM_VISION: contiene la posizione del giocatore se questo è inquadrato dal robot.
- MSG_TO_LOG: contiene informazioni usate per il debug, ad esempio le coordinate dei led infrarossi.

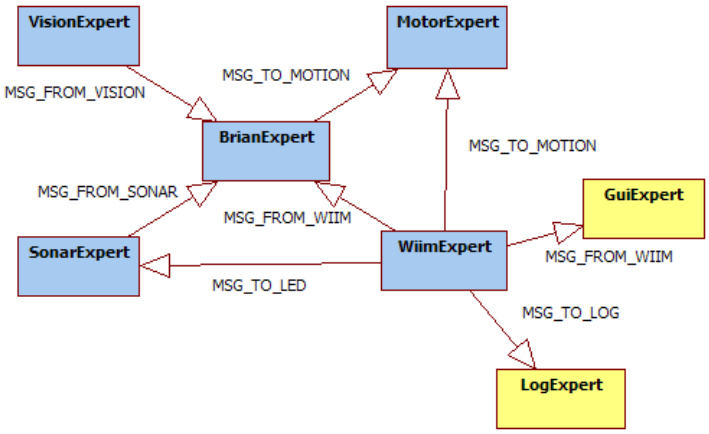

Figura 3.2: Moduli DCDT e messaggi scambiati (in giallo i moduli usati per il debug)

3.4.1 VisionExpert

Il modulo VisionExpert si occupa di gestire tutti gli aspetti legati alla visione artificiale del robot.

Questo modulo invia solo il messaggio *MSG_FROM_VISION* che contiene le eventuali informazioni sulla posizione del giocatore rispetto a Spykee, mentre non è necessario che riceva alcun messaggio. Nel costruttore viene creata la classe *SURFModule* che accetta come parametri la dimensione dell'immagine acquisita dal robot mentre, si memorizza il puntatore alla classe Spykee e si avvia la telecamera, mentre durante l'inizializzazione, funzione *RunInit*, vengono caricati i modelli del marker.

La funzione *RunDuty* in questo expert viene eseguita in loop, senza intervallo di tempo, si occupa di acquisire le immagini tramite il metodo *Spykee::getImage* che restituisce l'immagine in formato jpeg come vettore di byte che viene poi decompresso e passato alla classe *SURFModule* per essere eleborato dall'algoritmo di visione.
Subito dopo viene inviato il messagio *MSG_FROM_VISION* che contiene due parametri:

- VisionPlayerDetected: se vale 1 il giocatore è inquadrato, 0 altrimenti
- PlayerAngle: se il giocatore è inquadrato contiene l'angolo a cui si trova, altrimenti questo parametro viene omesso.

Il resto del codice serve solamente a mostrare a schermo l'immagine ripresa da Spykee con sovrapposte alcune informazioni come il marker e i frame al secondo. E' inoltre possibile catturare i fotogrammi e restrirare dei filmati.

3.4.2 SonarExpert

Questo expert si occupa di occupa di gestire i dati provenineti dai sonar e di controllare i led e viene eseguito in loop.

Durante l'inizializzazione viene creata la classe parser dei messaggi del sonar e viene inviato il comando di avvio degli stessi. Invece durante il ciclo principale vengono ricevuti i messaggi, in formato stringa, dai sonar e convertiti in valori numerici. Inoltre le distanze rilevate sono memorizzate in un buffer circolare in modo da poter accedere a tutte le distanze rlevate fino a circa un secondo prima. I quattro valori dei sonar sono poi inviati tramite un messaggio del tipo *MSG_FORM_SONAR* che, per una volta ogni secondo, invia anche i dati letti un secondo prima.

Sempre nel ciclo principale viene anche verificato se ci sono messaggi del tipo *MSG_TO_SONAR* da leggere che contengono i comandi per l'accensione e lo spegnimento dei led aggiuntivi montati su Spykee.

3.4.3 MotorExpert

Questo modulo si occupa di ricevere i messaggi del tipo *MSG_TO_MOTION* e inviare il valore contenuto in essi ai motori del robot tramite la classe Spykee. Se nessun messaggio di questo tipo è presente i set point dei motori vengono mantenuti all'ultimo valore ricevuto.

Un parametro importante per questo expert è il l'intervallo di tempo con cui il ciclo principale deve essere eseguito: valori troppo bassi portano infatti a una saturazione del buffer della wirless facendo perdere la maggior parte dei comandi ricevuti o eseguendoli comunque in ritardo, valori troppo alti invece faranno muovere Spykee a scatti.

3.4.4 BrianExpert

BrianExpert è il modulo che controlla la logica di gioco e definisce quali comportamenti attiviare in base ai messaggi che gli arrivano dagli altri expert (Figura 3.2).

Durante la fase di inizializzazione la classe si registra per la ricezione dei messaggi *MSG_FROM_WIIMOTE*, *MSG_FROM_SONAR* e *MSG_FROM_VISION*; viene inoltre inizializzata un istanza di Mr. BRIAN passando come parametri del costruttore i suoi file di configurazione (sezione 2.10).

Nel ciclo principale la prima cosa che viene effettuata è l'eventuale ricezione dei messaggi inviati dagli altri expert. Una volta ricevuti questi vengono parsificati e le variabili e i valori letti vengono memorizzati in una std::map utilizzando come chiave il nome della variabile e come valore una struttra

contente due valori in virgola mobile: l'effettivo valore della variabile e il suo grado di affidabilità (di solito sempre impostato a 1). L'utilizzo della mappa si è reso necessario per memorizzare quei dati che vengono inviati, da altri expert, con un intervallo di tempo superiore al tempo di ciclo di BrianExpert (ad esempio i dati dei sonar) e che quindi non sarebbero sempre disponibili a Mr. Brian per decidere i comportamenti. La mappa rende invece questi dati sempre disponibili anche se alcuni non risultano sempre aggiornati, tuttavia, considerando la maccanica di gioco e la velocità di movimento del robot, questo produce errori del tutto trascurabili.

Il compito principale di questo expert è di regolare alcuni aspetti relativi alla meccanica di gioco che non sono stati implementabili attraverso le regole fuzzy e Mr. Brian.
Il primo di questi è la gestione della fuga di Spykee una volta puntato: se infatti il robot non trova un nascondiglio o non è più puntato per un certo lasso di tempo torna in modalità di ricerca. Questo intervallo di tempo e l'impostazione degli stati vengono fatti tramite codice C++ utilizzando come timer il segnale di clock generato da WiimExpert.

Un altro importante compito svolto da BrianExpert è quello di permettere a Spykee di girare di 90° a sinistra o a destra: l'assenza di odometria sul robot rende infatti impossibile sapere di quanto il robot si sta effettivamente muovendo e di conseguenza di farlo ruotare su se stesso con precisione. Per aggirare, in parte, questo problema si è deciso di utilizzare due regole di Mr. Brian, HideLeft e HideRight, che interagiscono direttamente col codice per far compiere al robot le diverse azioni necessarie per nascondersi, ovvero: una volta individuata una cavità laterale il robot si ferma, in seguito ruota su se stesso di circa 90° e poi prosegue dritto fino a quando non rileva un ostacolo ai lati. A questo punto il robot torna in modalità di ricerca. Se una volta nascosto Spykee rileva che a destra e a sinistra non vi sono ostacoli e quandi ci sono forti probabilità che sia in vista, il robot riprende la fuga. Questa situazione può verificarsi, ad esempio, se i sonar rilevano la gamba di un tavolo o di una sedia.

L'ultimo compito svolto dalla classe è la retroazione di alcune variabili in uscita da Mr. Brian, ad esempio lo stato del gioco o la direzione di ricerca e la cancellazione di altre; questo è necessario in quanto Mr. Brian non è dotato di memoria e alla fine di ogni eseguzione è necessario reimpostare tutte le variabili in ingresso per mantere il gioco in uno stato coerente.

3.4.5 WiimExpert

Questo modulo, eseguito in loop, controlla la comunicazione con il WiiMote e contiene il codice che gestisce il gioco e l'interazione con l'utente. WiimExpert riceve messaggi del tipo *MSG_FROM_BRIAN* che vengono utilizzati per stampare a terminale lo stato del gioco e altre informazioni di debug su Mr BRIAN.
Quando il sistema è impostato in modalià di controllo manuale (sezione 3.1 l'unico compito di questo expert è inviare direttamente ai motori i comandi di moto tramite il messaggio *MSG_TO_MOTION*. Inoltre quando viene avviata questa modalità Mr. Brian viene messo in pausa inviato il messaggio *MSG_FROM_WIIMOTE* (in questo expert tutte le volte che vegono inviati dati in uscita si usa questo tipo di messaggio, se non diversamente specificato) contente la variabile la variabile *GameStatus*.

In modalità automatica, o di gioco, il WiimExpert svolge invece molti compiti. Il più importante è sicuramente la gestione dei dati del sensore infrarosso del Wiimote. Da esso riceve due parametri: le coordinate sul piano immagine del punto effettivamente puntanto dal Wiimote, calcolate utilizzando i due LED infrarossi di Spykee, e un flag che ne indica la validità e che assume valore negativo nel caso in cui non siano puntati almeno due LED o nel caso in cui le wiiuse non riescano a calcolare le coordinate, ad esempio a causa di altre sorgenti di luce infrarossa.

Quando le coordinate sono vicine a (512,384) Spykee viene considerato puntato; se rimane tale per circa 4 secondi il colpo viene caricato e se il giocatore preme il tasto di fuoco totalizza un punto, indicato da una vibrazione del Wiimote. Il livello di carica viene indicato accendendo in sequenza i LED blu sul WiiMote e inviando il messaggio *MSG_TO_SONAR* per accendere i led rossi, che indicano il livello di carica del colpo, e il led verde, che invece indica se il robot è puntato. Le coordinate e il flag di validità vengono anche inviate a Mr. Brian in modo da attivare i comportamenti appropriati a questa situazione.

Altri importanti compiti del WiimExpert sono: la gestione di alcuni timer di gioco, ad esempio quello di avvio partita e quello di pausa dopo che viene totalizzato un punto, la generazione di valori casuali inviati a Mr. Brian e di un segnale di clock (1 Hz) per la sincronizzazione e la temporizzazione di alcuni comportamenti (sezione 3.5).

3.5 MrBrian

Per il corretto svolgimento del gioco sono state implementate 7 diversi comportamenti che di seguito vengono elencati e spiegati da quelli appartenti al livello più basso fino a quello più alto.
In questa sezione vengono anche illustrati i principali predicati utilizzati come condizioni di CANDO e di WANT e come ingressi per i singoli comportamenti.

3.5.1 Principapli predicati

Dal modulo WiimExpert vegnono ricevuti informazioni sul puntamento degli infrarossi. Queste informazioni permettono di calcolare i seguenti predicati:

- IRDirCloseR, IRDirNearR, IRDirFarR: indicano rispettivamente se il Wiimote è puntato vicino, a media distanza e lontano alla destra del robot.
- IRDirCloseL, IRDirNearL, IRDirFarL: indicano rispettivamente se il Wiimote è puntato vicino, a media distanza e lontano alla sinistra del robot.
- IRExists: indica che gli infrarossi sono stati rilevati da Spykee.
- IRCenterP: indica che il robot è puntato.
- Random*: servono per trasferire a Mr. Brian dei numeri casuali utilizzate per generare i movimenti.
- CkHigh, CkLow: identificano il segnale di clock generato dalla classe WiimExpert. Rispettivamente indicano il valore 1 e 0 del clock.
- IsGameStarted: indica se il gioco è partito o se ci si trova in modalità di controllo manuale.

Le informazioni del modulo VisionExpert vengono invece utilizzate per genrare i seguenti predicati:

- PlayerDetected: il giocatore viene individuato da Spykee.
- PlayerOnLeft, PlayerOnRight, PlayerOnCenter: indicano in quale zona dell'immagine si trova il giocatore.
- NoPlayer: indica che il giocatore non è inquadrato e che non sta puntando a Spykee (informazione ottenuta da WiimExpert).

Con informazioni ricevute dal modulo SonarExpert vengono generati predicati che indicano la distanza rilevata dai quattro sonar di Spykee. Per questi predicati, di facile comprensione, si veda l'Appendice A.

Il modulo BrianExpert invia e riceve da Mr. Brian diverse informazioni sotto forma di varibili fuzzy, molte delle quali sono utilizzate per consentire al robot di nascondersi correttamente.

- MotorTurningL, MotorTurningR: indicano che il robot sta ruotando di 90° rispettivamente a destra e a sinistra per nascondersi.
- GoForwardP: indica che il robot ha terminato la rotazione e procede dritto per raggiungere il nascondiglio.
- MotorIsAuto: indica che i motori sono sotto il controllo diretto di BrianExpert per permettere la rotazione temporizzata.
- : MotorIsManual: indica che i motori del robot sono controllati dalle variabili fuzzy in uscita da Mr. Brian. Questo avviene in ogni fase di gioco ad eccezione di quando il robot si nascondo.

3.5.2 Variabili in uscita

Dalle varie regole definite in Mr. Brian possono uscire le seguenti variabili di controllo:

- TanSpeed: indica la velocità tangenziale del robot.
- RotSpeed: indica la velocità di rotazione del robot.
- MotorCommand: indica la direzione in cui Spykee deve ruotare per nascondersi.
- GameStatus: indica lo stato in cui si trova il gioco: ricerca del giocatore, fuga o quando Spykee si sta nascondendo.
- RotSearch: indica la direzione (destra o sinistra) in cui cercare il giocatore; viene solo retroazionata al ciclo successivo di Mr. Brian.

3.5.3 Comportamenti

Qui di seguito le regole vengono spiegate nel loro comportamento generale, per i dettagli e per le condizioni di attivazione (CANDO e WANT) si veda l'Appendice A).

3.5.3.1 SearchDir

Predicati in ingresso: LeftSearch, RightSearch, LeftBusy, RightBusy.
Variabili in uscita: RotSearch.

Questa semplice regola di livello 1 viene utilizzata solo in fase di ricerca del giocatore per stabile in che direzione il robot deve ruotare e, nel caso in quella direzione vi sia un ostacolo, invertirla. Il valore in uscita RotSearch viene utilizzato solo dalla regola Searching di livello 2.

3.5.3.2 Searching

Predicati in ingresso: CkHigth, propRotSearchRight, propRotSearchLeft, FrontFree, RightBusy, LeftBusy, FrontVeryNear, BackFree
Variabili in uscita: TanSpeed, RotSpeed, GameStatus.

Questa regola di livello 2 utilizza i valori in uscita da SearchDir per decidere come ruotare su se stesso Spykee durante la fase di ricerca del giocatore. In questa situazione il robot gira su se stesso ogni qual volta il clock interno (generato da WiimExpert) è a 1, mentre quando è a 0 resta fermo. Nel caso in cui Spykee, ruotando, rilevi un ostacolo la direzione di moto viene invertita. In questo modo la telecamera non viene puntata verso oggetti troppo vicini e Spykee osserva solo le aree libere del campo di gioco, dove probabilmente si trova il giocatore. Inoltre se Spykee rileva delle ostruzioni sia a destra che a sinistra si sposta in avanti o indietro cercando di liberarsi sui lati.
L'ultimo compito svolto dalla regola è il mantenimento dello stato di gioco in modalità ricerca.

3.5.3.3 Escape

Predicati in ingresso: GameStatusEscape, propTanSpeedForward, propRotSpeedRight, propRotSpeedLeft, PlayerOnLeft, PlayerOnRight, PlayerOnCenter, RightFree, LeftFree, FrontBusy, IRExists, IRAnyL, IRAnyR.
Variabili in uscita: RotSpeed, TanSpeed, GameStatus.

Questa regola, di livello 3, viene attivata solo se si verificano due condizioni: Spykee vede il giocatore o quest'ultimo punta il robot. Queste condizioni stabiliscono anche, tramite le singole azioni della regola, come far muovere il robot attraverso le varibili in uscita RotSpeed e TanSpeed.

Quando Spykee vede il giocatore indietreggia e sterza a destra o a sin-

istra a seconda che il giocatore sia inquadrato rispettivamente nella parte destra o sinistra dell'immagine, se invece il robot rileva di essere puntato con il WiiMote si muove in avanti a massima velocita e compie delle rapide sterzate in base alla direzione da cui viene puntato (stimata utilizzando le coordinate dei punti infrarossi rilevati dal Wiimote).

Quando invece le due condizioni non sonon più vere il robot procede in linea retta, per avere una lettura chiara dai sonar laterali e trovare un nascondiglio, fino allo scadere del timeout.
L'ultimo compido di questa regola è il mantenimento dello stato del gioco in modalità di fuga.

3.5.3.4 HideRight e HideLeft

Predicati in ingresso: GameStatusHide, propTanSpeedForward, propTanSpeedBackward, GoForwardP, FrontFree, FrontVeryNear. *Variabili in uscita*: TanSpeed, RotSpeed, GameStatus, MotorCommand.

Queste due regole, entrambe di livello 4, vengono attivate quando viene rilevata una cavità a destra o a sinistra del robot. Il conceetto di cavità viene gestito tramite l'utilizzo di predicati che considerano la distanza rilevata attualmente dal sonar e quella rilevata circa un secondo prima.

In regola vengono impostate alcune variabili in uscita, come ad esempio *MotorCommand*, che vengono utilizzate unicamente dal codice di BrianExpert per gestire correttamente la temporizzazione della rotazione di Spykee. Invece variabili come RotSpeed e TanSpeed vengono normalmente utilizzate per controllare la velocità dei motori. Infine questa regola si occupa anche di far ritornare il robot in modalità di ricerca una volta che si è nascosto.

3.5.3.5 Safe

Predicati in ingresso: GameStatusSearch, FrontVeryNear, RightVeryNear, LeftVeryNear, propTanSpeedForward.
Variabili in uscita: RotSpeed, TanSpeed.

Questa semplice regola di livello 5 si occupa di evitrare che Spykee si schianti contro ostacoli che si trovano davanti o dietro il robot a seconda che stia procedendo rispettivamente in avanti o indietro facendolo girare bruscamente a destra o a sinistra, a seconda delle distanze rilevate dai sonar. Questa regola è posizionata ad un livello alto in quanto ha la priorità su qualsiasi altra azione definita dalle regole precenti.

3.5.3.6 TrajectoryCorrection

Predicati in ingresso: propTanSpeedForward, propTanSpeedBackward, RightBusy, LeftBusy *Variabili in uscita*: RotSpeed. Qeusto comportamento, di livello 6, si occupa della correzione della traiettoria di Spykee facendolo allontare dagli ostacoli. Questa regola è al livello più alto in quando spesso dopo aver sterzato, ad esempio per eveitare un muro che si trovava di fronte, spesso il robot è costretto ad aggiustare la traiettoria in quanto, a casusa dell'assenza di odometria, non è possibile sapere di quanto il robot abbia voltato.

Capitolo 4

Direzioni future di ricerca e conclusioni

4.1 Problemi riscontrati

Durante la prima fase di realizzazione del sistema di visione si sono riscontrate alcune difficoltà nel realizzare l'algoritmo per il rilevamento del giocatore.

Un primo approccio prevedeva l'utilizzo di un algoritmo, costruito intorno al CamShift, e basato unicamente sull'identificazione dell'istogramma colore dell'oggetto. Tuttavia i primi test hanno dimostrato che non è adatto al nostro scopo in quanto non è in grado di rilevare correttamente oggetti che escono e rientrano nel campo visivo, come succede durante il gioco, a meno che le condizioni di luce non siano costati e il marker sia sempre di una tonalità facilmente distinguibile dal contesto. Ovviamente condizioni del genere sono impensabili in un ambiente domestico, per il quale è pensato il gioco, di conseguenza si è deciso di utilizzare l'algoritmo SURF.

In questo caso si sono riscontrare alcune difficoltà iniziali nell'adattare il SURF alle immagini inviate da Spykee. Queste ,infatti, per essere inviate più velocemente al computer, vengono compresse in jpeg. Questa compressione va ad intaccare proprio quelle caratteristiche dell'immagine (corner, edge) che vengono identificate dal SURF come punti caratteristici. Il problema è stato risolto utilizzando più modelli dello stesso marker e filtrando parzialmente l'immagine con l'istogramma colore di quest'ultimo.

In definitiva il robot è in grado di identificare correttamente il giocatore tranne nel caso in cui quest'utiltimo o Spykee si stiano muovendo troppo velocemente l'uno rispetto all'altro.

Si sono anche riscontrate alcune difficoltà nella definizione del concetto di nascondigilo per il robot. La difinizione più appropriata sarebbe un luogo in cui il robot non può essere visto dal giocatore, tuttavia l'assenza di un sistema di localizzazione non permette di poter soddisfare la condizione imposta da questa definizione. Si è deciso quindi di far nascondere Spykee in qualunque cavità che venga rilevata ai suoi lati. In questo modo si ha una buona probabilità di uscire dal campo visivo del giocatore che, per essere visto o per colpire Spykee, deve necessariamente trovarsi dietro o davanti ad esso.

4.2 Valutazioni conclusive

L'obiettivo princiaple di questa tesi è stato quello di realizzare un gioco in cui un umano e un robot autonomo dotato di visione possano interagire tra loro, utilizzando anche il controller Wiimote.

Durante le prime prove si è notato che il gioco è adatto ad un ambiente domestico. In questo ambiente, infatti, la presenza di diversi ostacoli (mobili, muri) costringono il robot a continui cambi di direzione, per nascondersi o evitare gli oggetti, rendono il robot difficile da puntare, anche se la scarsa velocità lo rende un bersaglio facile quando procede il linea retta. Tuttavia aver dotato Spykee di un sistema di visione lo rende un bersaglio indubbiamente più difficile da colpire.

4.3 Sviluppi futuri

Il gioco può essere ulteriormente sviluppato e migliorato sotto vari aspetti.

Volendo mantenere come piattaforma il robot Spykee si potrebbe decidere di utilizzare una telecamera migliore oppure di modificare il firmware, di recente diventato opensource, in modo da ridurre o eliminare la compressione jpeg delle immagini. In questo modo gli algoritmi di visione implementati diveterebbero più affidabili.

Una migliore qualità dell'immagine permetterebbe anche l'implementazione di sistemi di localizzazione dinamici: ad esempio MonoSLAM. Questo permetterebbe al robot di creare una mappa dell'ambiente circostante durante il gioco e di non utilizzare altri marker oltre a quello indossato dal giocatore.

Un altro approccio potrebbe essere quello di cambiare il robot. In questo caso se ne potrebbe usare uno dotato anche di movimenti laterali, al quale risulterebbe più facile schiavare i colpi e nascondersi. Inoltre, si potrebbe

decidere di dotarlo di un doppio sistema di visione. Uno per localizzare il giocatore, come gia avviene, e l'altro utilizzato come sistema di localizzazione, ad esempio utilizzando dei marker a soffitto per la creazione di una mappa.

Bibliografia

[1] Aibo. http://support.sony-europe.com/aibo.

[2] Spykee, the spy robot.

[3] Wiiuse, a c++ library for wiimote. http://wiiuse.net.

[4] xbee. http://www.digi.com.

[5] Zigbee alliance. http://www.zigbee.org.

[6] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *CVIU*, 110(3):346–359, 2008.

[7] Antonio Bianchi and Ben Chen. Sviluppo di un gioco tramite l'interazione fra un robot ed il controller wii remote. Master's thesis, Politecnico di Milano, 2007-2008.

[8] Sergio Bittanti. *Identificazione dei Modelli e Sistemi Adattivi (sesta edizione)*, chapter 5. Pitagora Editrice Bologna, sesta edizione edition, 2004.

[9] A. Bonarini, M. Matteucci, and M. Restelli. A novel model to rule behavior interaction. In *8th Conference on Intelligent Autonomous System (IAS-8)*, pages 199–206, Amsterdam, 2004. IOS Press.

[10] Andrea Bonarini, Giovanni Invernizzi, Thomas Halva Labella, and Matteo Matteucci. An architecture to coordiante fuzzy behaviors to control an autonomous robot. *Fuzzy Sets and Systems*, 134, 2003.

[11] Alessandro Martini Claudio Fantacci. P-surf: A surf library for processing. Master's thesis, Università degli studi di Firenze, 2008-2009.

[12] Cristian Giussani. Dcdt - guida per l'utente, 2005.

[13] Intel. Opencv. http://opencv.willowgarage.com/wiki.

[14] M. J. Jones and P. Viola. Rapid object detection using a boosted cascade of simple feature. *IEEE CVPR*, 2001.

[15] Antonio Micali. Robowii 2.0: un gioco interattivo con un robot autonomo. Master's thesis, Politecnico di Milano, 2008-2009.

[16] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration.

[17] Matteo Merlin Simone Mattia Antonio Laurenziano. Un sistema per il tracking visivo di persone con telecamera in movimento. Master's thesis, Politecnico di Milano, 2006-2007.

Appendice A

Regole fuzzy e comportamenti

A.1 File di configurazione di Mr. Brian

Qui di seguito sono elencati i file di configurazione di Mr. Brian

A.1.1 Forme fuzzy

```
(CAVITYSIZE
(TRA (SMALL 1 1 490 490))
(TOR (BIG 500 500))
)

(ROTSEARCH
(SNG (LEFT 0))
(SNG (RIGHT 1))
)

(SINGLETON
(SNG (EXIST 1))
)

(HIDEDIR
(SNG (STOP 0))
(SNG (LEFT 1))
(SNG (RIGHT 2))
)

(IRDIR
(TRA (CENTER -100 -50 50 100))
(TRA (CLOSEL -100 -50 0 0))
(TRA (CLOSER 0 0 50 100))
(TRA (NEARL -300 -50 0 0))
(TRA (NEARR 0 0 50 300))
(TOL (FARL -640 -200))
(TOR (FARR 200 640))
(TRA (EXIST -640 -640 640 640))
)

(IRDIST
(TOL (CLOSE 50 80))
(TRA (NEAR 60 110 150 200))
(TOR (FAR 170 200))
)

(HIT
(SNG (V 1))
(SNG (F 0))
(SNG (N -1))
)

(CLOCK
(SNG (V 1))
(SNG (F 0))
```

```
)

(BOOLEAN
(SNG (TRUE 1))
(SNG (FALSE 0))
)

(NUMBER
(SNG (ZERO 0))
(SNG (ONE 1))
(SNG (TWO 2))
(SNG (THREE 3))
(SNG (FOUR 4))
(SNG (FIVE 5))
)

(TAN2SPEED
(TOR (FAST_FORWARD 50 75))
(TRA (FORWARD 1 1 50 75))
(TRA (STOP 0 0 0 0))
(TRA (BACKWARD -75 -50 -1 -1))
(TOL (FAST_BACKWARD -75 -50))
)

(ROT2SPEED
(TOR (RIGHT 1 1))
(TOL (LEFT -1 -1))
)

(OBJECTDISTANCE
(TOL (VERYNEAR 250 300))
(TOL (NEAR 300 500))
(TRA (MEDIUM 450 530 1000 1100))
(TOR (FAR 1000 1250))
)

(PLAYERPOSITION
(TRA (LEFT -20 -20 159 159))
(TRA (CENTER 0 120 200 320))
(TRA (RIGHT 161 161 340 340))
)

(GAMESTATUS
(SNG (SEARCHING 0))
(SNG (ESCAPING    1))
(SNG (HIDING   2))
)

(MOTORSTATUS
(SNG (MANUAL 0))
(SNG (AUTO 1))
)

(MOTORDIRECTION
(SNG (FORWARD 0))
(SNG (BACKWARD 1))
(SNG (RIGHT 2))
(SNG (LEFT 3))
)
```

A.1.2 Variabili fuzzy

```
(Hit HIT)
(Clock CLOCK)

(Random NUMBER)
(Always BOOLEAN)
(GameStart BOOLEAN)

(SonarLeft OBJECTDISTANCE)
(SonarRight OBJECTDISTANCE)
(SonarBack OBJECTDISTANCE)
(SonarFront OBJECTDISTANCE)

(SonarLeftOld OBJECTDISTANCE)
(SonarRightOld OBJECTDISTANCE)
(SonarLeftDiff CAVITYSIZE)
(SonarRightDiff CAVITYSIZE)

(GameStatus GAMESTATUS)

(RotSearch ROTSEARCH)
(ProposedRotSearch ROTSEARCH)


(IRCenter IRDIR)
(IRDist IRDIST)
(VisionPlayerDetected BOOLEAN)

(AutoTurn ROTSEARCH)
(GoForward SINGLETON)
```

```
(MotorStatus MOTORSTATUS)
(EndTurn BOOLEAN)                (PlayerAngle PLAYERPOSITION)
                                 (IRAngle PLAYERPOSITION)
(ProposedTanSpeed TAN2SPEED)     (MotorDirection MOTORDIRECTION)
(ProposedRotSpeed ROT2SPEED)
```

A.1.3 Predicati

```
### IR ###
IRDirCloseR = (D IRCenter CLOSER);
IRDirCloseL = (D IRCenter CLOSEL);
IRDirNearR = (D IRCenter NEARR);
IRDirNearL = (D IRCenter NEARL);
IRDirFarR = (D IRCenter FARR);
IRDirFarL = (D IRCenter FARL);

IRAnyR = (OR (OR (P IRDirCloseR)(P IRDirNearR))(P IRDirFarR));
IRAnyL = (OR (OR (P IRDirCloseL)(P IRDirNearL))(P IRDirFarL));
IRCenterP = (D IRCenter CENTER);

IRDirClose = (OR (P IRDirCloseR) (P IRDirCloseL));
IRDirNear = (OR (P IRDirNearR) (P IRDirNearL));

IRClose = (D IRDist CLOSE);
IRNear = (D IRDist NEAR);
IRFar = (D IRDist FAR);

IRExists = (D IRCenter EXIST);

### Utility
HumanPoint = (D Hit V);
RobotPoint = (D Hit N);

RandomZero = (D Random ZERO);
RandomOne = (D Random ONE);
RandomTwo = (D Random TWO);
RandomThree = (D Random THREE);
RandomFour = (D Random FOUR);
RandomFive = (D Random FIVE);

TurnRandomLeft = (D Random ZERO);
TurnRandomRight = (D Random ONE);

### Clock ###
CkHigth = (D Clock V);
CkLow = (D Clock F);

IsGameStarted = (D GameStart TRUE);
```

```
### Vision ###

PlayerDetected = (D VisionPlayerDetected TRUE);

PlayerOnLeft = (AND (P PlayerDetected)(D PlayerAngle LEFT));
PlayerOnRight = (AND (P PlayerDetected)(D PlayerAngle RIGHT));
PlayerOnCenter = (AND (P PlayerDetected)(D PlayerAngle CENTER))
    ;

NoPlayer = (AND (NOT (P IRExists))(NOT (P PlayerDetected)));

### Game status ###
GameStatusSearch = (D GameStatus SEARCHING);
GameStatusEscape = (D GameStatus ESCAPING);
GameStatusHide = (D GameStatus HIDING);

### Searching  ###
LeftSearch = (D RotSearch LEFT);
RightSearch = (D RotSearch RIGHT);

### Utility ###
TheTruth = (D Always TRUE);
InGame = (OR (D Clock V)(D Clock F));

### Sonar ###
LeftVeryNear = (D SonarLeft VERYNEAR);
LeftBusy = (D SonarLeft NEAR);
LeftSemiBusy = (D SonarLeft MEDIUM);
LeftFree = (OR (D SonarLeft MEDIUM)(D SonarLeft FAR));

RightVeryNear = (D SonarRight VERYNEAR);
RightBusy = (D SonarRight NEAR);
RightSemiBusy = (D SonarRight MEDIUM);
RightFree = (OR (D SonarRight MEDIUM)(D SonarRight FAR));

BackVeryNear = (D SonarBack VERYNEAR);
BackBusy = (D SonarBack NEAR);
BackSemiBusy = (D SonarBack MEDIUM);
BackFree = (OR (D SonarBack MEDIUM)(D SonarBack FAR));

FrontVeryNear = (D SonarFront VERYNEAR);
FrontBusy = (D SonarFront NEAR);
FrontSemiBusy = (D SonarFront MEDIUM);
FrontFree = (D SonarFront FAR);

### Old sonar values ###
LeftFreeOld = (OR (D SonarLeftOld MEDIUM)(D SonarLeftOld FAR));
LeftBusyOld = (D SonarLeftOld NEAR);
```

```
RightFreeOld = (OR (D SonarRightOld MEDIUM)(D SonarRightOld FAR
    ));
RightBusyOld = (D SonarRightOld NEAR);

### Cavity ###
CavityOnLeft = (AND (P LeftBusyOld)(P LeftFree));
CavityOnRight = (AND (P RightBusyOld)(P RightFree));

### HideRight, HideLeft stuff ###
MotorTurningL = (D AutoTurn LEFT);
MotorTurningR = (D AutoTurn RIGHT);

GoForwardP = (D GoForward EXIST);
MotorIsAuto = (D MotorStatus AUTO);
MotorIsManual = (D MotorStatus MANUAL);
MotorEndTurn = (D EndTurn TRUE);
```

A.1.4 Predicate actions

```
propTanSpeedFF = (D ProposedTanSpeed FAST_FORWARD);
propTanSpeedF = (D ProposedTanSpeed FORWARD);
propTanSpeedFB = (D ProposedTanSpeed FAST_BACKWARD);
propTanSpeedB = (D ProposedTanSpeed BACKWARD);
propTanSpeedBackward = (OR (D ProposedTanSpeed BACKWARD)(D
    ProposedTanSpeed FAST_BACKWARD));
propTanSpeedForward = (OR (D ProposedTanSpeed FORWARD)(D
    ProposedTanSpeed FAST_FORWARD));
propRotSpeedRight = (D ProposedRotSpeed RIGHT);
propRotSpeedLeft = (D ProposedRotSpeed LEFT);

### Searching
propRotSearchLeft = (D ProposedRotSearch LEFT);
propRotSearchRight = (D ProposedRotSearch RIGHT);
#propRotSearchStop = (D ProposedRotSearch STOP);
```

A.1.5 CANDO

```
SearchDir = (P GameStatusSearch);
Searching = (P GameStatusSearch);

Escape = (OR
         (P GameStatusEscape)
         (OR
            (P IRExists)
            (P PlayerDetected)
         )
      );

HideRight = (AND
```

```
            (NOT (P GameStatusSearch))
            (OR
               (P CavityOnRight)
               (AND
                  (P MotorTurningR)
                  (NOT (P MotorTurningL))
               )
            )
         );

HideLeft = (AND
            (NOT (P GameStatusSearch))
            (OR
               (P CavityOnLeft)
               (AND
                  (P MotorTurningL)
                  (NOT (P MotorTurningR))
               )
            )
         );

TraiettoryCorrection = (NOT(P TheTruth));
Safe = (P TheTruth);
```

A.1.6 WANT

```
SearchDir = (AND
            (NOT (P PlayerDetected))
            (NOT (P IRExists))
         );


Searching = (AND
            (NOT (P PlayerDetected))
            (NOT (P IRExists))
         );


Escape = (OR
         (P GameStatusEscape)
         (AND
            (P GameStatusSearch)
            (OR
               (P IRExists)
               (P PlayerDetected)
            )
         )
      );
```

```
HideRight = (AND
             (NOT (P GameStatusSearch))
             (OR
                (P CavityOnRight)
                (AND
                   (P MotorTurningR)
                   (NOT (P MotorTurningL))
                )
             )
          );

HideLeft = (AND
             (NOT (P GameStatusSearch))
             (OR
                (P CavityOnLeft)
                (AND
                   (P MotorTurningL)
                   (NOT (P MotorTurningR))
                )
             )
          );

TraiettoryCorrection = (NOT (P TheTruth));
Safe = (P TheTruth);
```

A.1.7 Behaviour

```
( level 1 SearchDir ./config/brian/spykee/SearchDir.rul )
( level 2 Searching ./config/brian/spykee/Searching.rul )

( level 3 Escape ./config/brian/spykee/Escape.rul )

( level 4 HideRight ./config/brian/spykee/HideRight.rul )
( level 4 HideLeft ./config/brian/spykee/HideLeft.rul )

( level 5 Safe ./config/brian/spykee/Safe.rul )

( level 6 TraiettoryCorrection ./config/brian/spykee/
    TraiettoryCorrection.rul )
```

A.1.8 Forme fuzzy in uscita

```
( TanSpeed TANSPEED )            ( GameStatus GAMESTATUS )
( RotSpeed ROTSPEED )            ( RotSearch ROTSEARCH )
( MotorCommand MOTORCOMMANDS )
```

A.1.9 Varabili in uscita

```
(TANSPEED                          (SNG (FAST_LEFT -75))
(SNG (VERY_FAST_FORWARD 100))      (SNG (VERY_FAST_LEFT -100))
(SNG (FAST_FORWARD 80))            )
(SNG (FORWARD 50))
(SNG (SLOW_FORWARD 20))            (MOTORCOMMANDS
(SNG (VERY_SLOW_FORWARD 10))       (SNG (RX90 0))
(SNG (STOP 0))                     (SNG (LX90 1))
(SNG (VERY_SLOW_BACKWARD -10))     (SNG (RX180 2))
(SNG (SLOW_BACKWARD -20))          (SNG (LX180 3))
(SNG (BACKWARD -50))               (SNG (RX360 4))
(SNG (FAST_BACKWARD -80))          (SNG (LX360 5))
(SNG (VERY_FAST_BACKWARD -100))    (SNG (STOP 6))
)                                  )

(ROTSPEED                          (ROTSEARCH
(SNG (VERY_FAST_RIGHT 100))        (SNG (LEFT 0))
(SNG (FAST_RIGHT 75))              (SNG (RIGHT 1))
(SNG (RIGHT 50))                   )
(SNG (SLOW_RIGHT 15))
(SNG (VERY_SLOW_RIGHT 5))          (GAMESTATUS
(SNG (STOP 0))                     (SNG (SEARCHING 0))
(SNG (VERY_SLOW_LEFT -5))          (SNG (ESCAPING    1))
(SNG (SLOW_LEFT -15))              (SNG (HIDING   2))
(SNG (LEFT -50))                   )
```

A.2 Comportamenti

Di seguito sono invece elencati i comportamenti del robot.

A.2.1 SearchDir

```
(AND
   (LeftSearch)
   (LeftBusy)
) => (RotSearch RIGHT);

(AND
   (LeftSearch)
   (NOT (LeftBusy))
) => (RotSearch LEFT);

(AND
   (RightSearch)
   (RightBusy)
) => (RotSearch LEFT);

(AND
```

```
   (RightSearch)
   (NOT (RightBusy))
) => (RotSearch RIGHT);

(AND
   (NOT (RightSearch))
   (NOT (LeftSearch))
) => (RotSearch LEFT);
```

A.2.2 Searching

```
(AND
   (CkHigth)
   (propRotSearchRight)
) => (RotSpeed VERY_SLOW_RIGHT); #VERY_SLOW_RIGHT

(AND
   (CkHigth)
   (propRotSearchLeft)
) => (RotSpeed VERY_SLOW_LEFT);  #VERY_SLOW_LEFT

(CkLow) => (TanSpeed STOP)(RotSpeed STOP);

(AND
   (FrontFree)
   (AND
      (RightBusy)
      (LeftBusy)
   )
) => (TanSpeed SLOW_FORWARD); #SLOW_FORWARD

(AND
   (AND
      (FrontVeryNear)
      (BackFree)
   )
   (AND
      (RightBusy)
      (LeftBusy)
   )
) => (TanSpeed SLOW_BACKWARD);

(TheTruth) => (GameStatus SEARCHING);
```

A.2.3 Escape

```
(AND
   (GameStatusEscape)
   (propTanSpeedForward)
) => (&DEL.TanSpeed ANY);
```

```
(AND
   (GameStatusEscape)
   (OR
      (propRotSpeedRight)
      (propRotSpeedLeft)
   )
) => (&DEL.RotSpeed ANY);

#imposta lo stato del robot
(TheTruth) => (GameStatus ESCAPING);

#vede il giocatore a sinistra
(AND (PlayerOnLeft)(RightFree)) => (TanSpeed STOP)(RotSpeed
    SLOW_RIGHT);

#vede il giocatore a destra
(AND (PlayerOnRight)(LeftFree)) => (TanSpeed STOP)(RotSpeed
    SLOW_LEFT);

#Se vede il giocatore va indietro
(AND (PlayerOnCenter)(BackFree)) => (TanSpeed BACKWARD);

#rilevo ir
(AND (IRExists)(NOT(FrontBusy))) => (TanSpeed FAST_FORWARD);

#Se davanti cè' un ostacolo gira a destra
(AND
   (AND(IRExists)(FrontBusy))
   (AND
      (LeftFree)
      (NOT (RightFree))
   )
) => (RotSpeed RIGHT);

#Se davanti cè' un ostacolo gira a sinistra
(AND
   (AND(IRExists)(FrontBusy))
   (AND
      (RightFree)
      (NOT (LeftFree))
   )
) => (RotSpeed LEFT);

(AND
   (AND(IRExists)(FrontBusy))
   (AND
      (RightFree)
      (LeftFree)
```

```
   )
) => (RotSpeed LEFT);


(AND
   (IRAnyL)
   (RightFree)
) => (RotSpeed RIGHT);

(AND
   (IRAnyR)
   (LeftFree)
) => (RotSpeed LEFT);


#se non rilevo IR allora continuo a scappare fino a quando non
    scade il timeout (settato da codice)
(AND
   (GameStatusEscape)
   (NoPlayer)
) => (&DEL.TanSpeed ANY)(&DEL.RotSpeed ANY)(TanSpeed FORWARD);
```

A.2.4 HideLeft

```
(TheTruth) => (&DEL.GameStatus ANY);

(AND
   (NOT (GameStatusHide))
   (OR (propTanSpeedForward)(propTanSpeedBackward))
) => (&DEL.TanSpeed ANY)(&DEL.RotSpeed ANY);

(AND
   (TheTruth)
   (NOT (GoForwardP))
) => (RotSpeed FAST_LEFT)(MotorCommand LX90)(GameStatus HIDING)
    ;


(AND
   (GoForwardP)
   (FrontFree)
) => (&DEL.RotSpeed ANY)(TanSpeed SLOW_FORWARD)(GameStatus
    HIDING);


(AND
   (GoForwardP)
   (FrontVeryNear)
```

```
) => (&DEL.TanSpeed ANY)(&DEL.RotSpeed ANY)(TanSpeed STOP)(
    RotSpeed STOP)(MotorCommand STOP)(GameStatus SEARCHING);

#(AND (MotorEndAuto) (OR (NOT (IRExist))(NOT (PlayerDetected)))
    ) => (TanSpeed STOP)(RotSpeed STOP);
```

A.2.5 HideRight

```
(TheTruth) => (&DEL.GameStatus ANY);

#Annullo i movimenti
(AND
   (NOT (GameStatusHide))
   (OR (propTanSpeedForward)(propTanSpeedBackward))
) => (&DEL.TanSpeed ANY)(&DEL.RotSpeed ANY);

#Inizializzazione rotazione a destra
#Imposto velocita e flags
(AND
   (TheTruth)
   (NOT (GoForwardP))
) => (RotSpeed FAST_RIGHT)(MotorCommand RX90)(GameStatus HIDING
    );


#Fine rotazione, muovo il robot avanti
(AND
   (GoForwardP)
   (FrontFree)
) => (&DEL.RotSpeed ANY)(&DEL.TanSpeed ANY)(TanSpeed
    SLOW_FORWARD)(GameStatus HIDING);

#Ho raggiunto il nascondiglio, fermo il robot
#e torno in àmodalit di ricerca
(AND
   (GoForwardP)
   ( OR
      (FrontSemiBusy)
      (FrontBusy)
   )
) => (&DEL.TanSpeed ANY)(&DEL.RotSpeed ANY)(TanSpeed STOP)(
    RotSpeed STOP)(MotorCommand STOP)(GameStatus SEARCHING);
```

A.2.6 Safe

```
(AND
   (NOT(GameStatusSearch))
   (FrontVeryNear)
)=> (&DEL.TanSpeed ANY)(&DEL.RotSpeed ANY)
   (RotSpeed FAST_LEFT);
```

```
(AND
   (propTanSpeedForward)
   (RightVeryNear)
) => (&DEL.RotSpeed ANY)
    (RotSpeed SLOW_LEFT);

(AND
   (propTanSpeedForward)
   (LeftVeryNear)
) => (&DEL.RotSpeed ANY)
    (RotSpeed SLOW_RIGHT);
```

A.2.7 TraiettoryCorrection

```
(AND (propTanSpeedForward)(RightBusy)) => (&DEL.RotSpeed ANY)(
    RotSpeed FAST_LEFT);

(AND (propTanSpeedForward)(LeftBusy)) => (&DEL.RotSpeed ANY)(
    RotSpeed FAST_RIGHT);

(AND (propTanSpeedBackward)(RightBusy)) => (&DEL.RotSpeed ANY)(
    RotSpeed FAST_RIGHT);

(AND (propTanSpeedBackward)(LeftBusy)) => (&DEL.RotSpeed ANY)(
    RotSpeed FAST_LEFT);
```

Appendice B

Listato

B.1 kernel

Listing B.1: src/kernel.cpp

```
1 #include <const.h>
2 #include <string>
3 #include <type.h>
4 #include <cmath>
5
6 #include "Spykee.h"
7
8 #define SPYKEE
9
10
11 #define BRIAN
12 #define WIIM
13 #define LOG
14 #define MOTOR
15
16 #define MOTOR_DEBUG
17 #define MOTOR_DETAIL
18 #define MOTOR_FORCE_DEBUG
19
20 #ifdef MOTOR_FORCE_DEBUG
21 #define MOTOR_DEBUG
22 #endif
23
24 #ifdef MOTOR
25   #ifdef MOTOR_DETAIL
26     #define MOTOR_DEBUG
27   #endif
28 #endif
29
```

```
30 #define SONAR
31
32 #ifdef SPYKEE
33   #define VISION
34 #endif
35
36 //#define PC_GUI
37
38 #ifdef BRIAN
39 #include "BrianExpert.h"
40 int brian_period = 1; //in microsecondi
41 #endif
42
43 #ifdef SONAR
44 #include "SonarExpert.h"
45 int sonar_period=0;
46 #endif
47
48 #ifdef WIIM
49 #include "WiimExpert.h"
50 int wiim_period = 100;
51 #endif
52
53 #ifdef MOTOR
54 #include "MotorExpert.h"
55 int motor_period=150000;
56 #endif
57
58 #ifdef LOG
59 #include "LogExpert.h"
60 int log_period=1;
61 #endif
62
63 #ifdef VISION
64 #include "VisionExpert.h"
65 int Vision_period = 0;
66 #endif
67
68 #include "ControlExpert.h"
69 #include "SpykeeGui.h"
70 int gui_period = 0;
71
72
73 #include "Configurator.h"
74
75 // DCDT Configuration files
76 string dcdt_config_file("./config/dcdt.conf");
77 string default_hostname("hostname");
78 string spykee_config_file("./config/spykee.cfg");
```

```
79
80 void OnSignal(int s);
81 ModuleDispatch *pDispatch = NULL;
82
83 int main(int argc, char** argv){
84   Spykee *s = NULL;
85
86   if (!Configurator::init(spykee_config_file, argc, argv))
87     cout << "Unable to load configuration file!" << endl;
88
89 #ifdef SPYKEE
90   if (!Configurator::simulation){
91     s = new Spykee(Configurator::spykee_user, Configurator::
         spykee_pass);
92     s->unplug();
93     s->startCamera();
94     s->turnOffCameraLight();
95   } else {
96     s = NULL;
97   }
98 #endif
99
100  // Create the Dispatcher
101  ModuleDispatch* Dispatch = new ModuleDispatch ("", basename(
       argv[0]));
102  pDispatch = Dispatch;
103
104  // Create the Agent list
105  std::vector< StringModuleMember*> agents;
106
107  // ----------- BRIAN ----------- //
108 #ifdef BRIAN
109  BrianExpert* pBrian = new BrianExpert (Dispatch,brian_period)
       ;
110  agents.push_back(pBrian);
111  cout << endl << "--- BrianExpert Created --- "<<endl;
112 #endif
113
114  // ----------- MOTOR ----------- //
115 #ifdef MOTOR
116  MotorExpert* pMotor = new MotorExpert (Dispatch, motor_period
       , s);
117  agents.push_back(pMotor);
118  cout << endl << "--- MotorExpert Created --- "<<endl;
119 #endif
120
121 #ifdef LOG
122  LogExpert* pLog = new LogExpert (Dispatch, log_period);
123  agents.push_back(pLog);
```

```
124   cout << endl << "--- LogExpert Created --- "<<endl;
125 #endif
126
127 #ifdef PC_GUI
128   DCDT_Mutex mMutex;
129   SpykeeGui *pSpykeeGui = new SpykeeGui(&mMutex, &argc, &argv);
130   ControlExpert *pControl = new ControlExpert(Dispatch, 0,
          pSpykeeGui);
131   agents.push_back(pControl);
132   cout << endl << "--- ControlExpert Created --- " << endl;
133 #else
134   SpykeeGui *pSpykeeGui = NULL;
135 #endif
136
137
138   // -------------- VISION ----------------- //
139 #ifdef VISION
140   if (!Configurator::simulation){
141     #ifdef PC_GUI
142       VisionExpert* pVision = new VisionExpert (Dispatch,
              Vision_period, s, pSpykeeGui, &mMutex);
143     #else
144       VisionExpert* pVision = new VisionExpert (Dispatch,
              Vision_period, s, NULL, NULL);
145     #endif
146
147     agents.push_back(pVision);
148     cout << endl << "--- Vision Created --- "<<endl;
149   }
150 #endif
151
152   // ----------- WIIM ----------- //
153 #ifdef WIIM  MemFlag
154   WiimExpert* pWiim = new WiimExpert (Dispatch, wiim_period, s)
          ;
155   agents.push_back(pWiim);
156   cout << endl << "--- WiimExpert Created --- "<<endl;
157 #endif
158
159   // ----------- SONAR ----------- //
160 #ifdef SONAR
161   SonarExpert* pSonar = new SonarExpert (Dispatch, "SonarExpert
          ", sonar_period);
162   agents.push_back(pSonar);
163   cout << endl << "--- SonarExpert Created --- "<<endl;
164 #endif
165
166   for (std::vector < StringModuleMember * >::iterator i =
          agents.begin (); i != agents.end (); i++)
```

```
167     Dispatch->AddMember (*i);
168
169   cout << endl << "--- Community Created --- "<<endl;
170
171   for (std::vector < StringModuleMember * >::iterator i =
          agents.begin (); i != agents.end (); i++)
172     Dispatch->ActivateMember (*i);
173
174   signal(SIGINT, OnSignal);
175   signal(SIGTERM, OnSignal);
176
177   cout << endl << "--- ''let's work, be proud...'' --- "<<endl;
178   Dispatch->LetsWork ();
179
180   signal(SIGINT, 0);
181   signal(SIGTERM, 0);
182
183   cout << endl << "--- Community Session Terminated --- "<<endl
          ;
184
185   return 0;
186 }
187
188 void OnSignal(int s){
189   switch(s){
190     case SIGINT:
191     case SIGTERM:
192       cout << "\nCtrl-c detected -> Shutting down..." << endl;
193       pDispatch->SetStatus(DCDT_TERMINATING);
194     break;
195   }
196
197   signal(s,OnSignal);
198 }
```

B.2 VisionExpert

Listing B.2: src/VisionExpert.h

```
1  #ifndef VisionExpert_h
2  #define VisionExpert_h
3
4  #include "StringModuleMember.h"
5  #include "SurfModule.h"
6
7  #include <sys/time.h>
8  #include <stdlib.h>
9  #include <stdio.h>
10 #include <const.h>
```

```
11 #include <msg.h>
12
13 #include <sys/time.h>
14
15 #include "Spykee.h"
16 #include "SpykeeGui.h"
17 #include "FileLogger.h"
18
19 #include <opencv/cv.h>
20 #include <opencv/highgui.h>
21 #include <opencv/cxcore.h>
22
23 using namespace std;
24 using namespace cv;
25
26 #define IMAGE_WIDTH   320
27 #define IMAGE_HEIGHT  240
28
29 #define SURF_THR 225
30
31 #define RED 0
32 #define GREEN 1
33 #define BLUE 2
34 #define ORANGE 3
35 #define VIOLET 4
36 #define YELLOW 5
37 #define BLACK 6
38 #define WHITE 7
39
40 static const CvScalar colors[] = { CV_RGB(255,0,0), CV_RGB
       (0,255,0), CV_RGB(0,0,255),
41                    CV_RGB(255,255,0), CV_RGB(255,0,255), CV_RGB
                          (0,255,255),
42                    CV_RGB(0,0,0), CV_RGB(255,255,255) };
43
44
45 class VisionExpert : public StringModuleMember {
46   class ModelData {
47     public:
48     ModelData();
49     ModelData(string _file, IplImage *image, CvSeq *objKey, Mat
           *objDesc, Mat *ind, Mat *dist, CvSize s, double a);
50     ~ModelData();
51
52     vector<CvPoint2D32f> pt1;
53     vector<CvPoint2D32f> pt2;
54
55     string file;
56     IplImage *pImage;
```

```
57      CvSeq *objKeypoints;
58      Mat *objDescriptors;
59      Mat *indices;
60      Mat *distances;
61      CvSize imgSize;
62      double area;
63    };
64
65    typedef vector<VisionExpert::ModelData*> ModelsVector;
66
67    public:
68    VisionExpert(ModuleDispatch* kernel, int period, Spykee *s,
          SpykeeGui *g, DCDT_Mutex *mutex);
69    ~VisionExpert();
70
71    void RunInit();
72    void RunDuty();
73    void RunClose();
74
75    protected:
76    void AddBrianData(char* name,float value,float rel);
77
78    Spykee *pSpykee;
79
80    bool m_bRecordVideo;
81    CvVideoWriter *m_videoWriter;
82    CvCapture *pCap;
83    unsigned int mFramesCount;
84
85    IplImage *videoImage;
86
87    private:
88    SpykeeGui *gui;
89    DCDT_Mutex *pMutex;
90    FileLogger m_logger;
91    SURFModule *pSurf;
92 };
93
94 #endif
```

Listing B.3: src/VisionExpert.cpp

```
1  #include "VisionExpert.h"
2
3  VisionExpert::VisionExpert(ModuleDispatch* kernel, int period,
       Spykee *s, SpykeeGui *g, DCDT_Mutex *mutex)
4    : StringModuleMember(kernel, period, "VisionExpert"),
         m_logger("/log/VisionLog.txt", true){
5    pSpykee = s;
6    gui = g;
```

```
7   pMutex = mutex;
8
9   this->pSurf = new SURFModule(320,240);
10
11  if (pSurf == NULL){
12    cout << "Unable to initialize vision module! Memory?" <<
          endl;
13    this->Shutdown(1);
14  }
15
16  m_bRecordVideo = false;
17
18  if (pSpykee)
19    pSpykee->startCamera();
20
21  printf("VisionExpert()\n");
22 }
23
24 VisionExpert::~VisionExpert(){
25  if (pSurf)
26    delete pSurf;
27
28  printf("\nVisionExpert::~VisionExpert()\n");
29 }
30
31 void VisionExpert::RunInit(){
32  printf("VisionExpert::RunInit()\n");
33
34  pSurf->AddModel("./models/big.bmp");
35  pSurf->AddModel("./models/big2.bmp");
36  pSurf->AddModel("./models/small.bmp");
37  pSurf->AddModel("./models/medium.bmp");
38
39  m_bRecordVideo = false;
40  mFramesCount = 0;
41 }
42
43 void VisionExpert::RunDuty(){
44  int imageLen = 0;
45  char imgName[50] = {0};
46
47  if (!pSpykee){
48    cout << "pSpykee == NULL" << endl;
49    return;
50  }
51
52  if (pMutex)
53    pMutex->lock();
54
```

```
55  unsigned char *jpegBuffer = pSpykee->getImage(imageLen);
56
57  if (jpegBuffer == NULL || imageLen <= 0){
58    printf("\nImage null!\n");
59    return;
60  }
61
62  CvMat jpgMat = cvMat(1, imageLen, CV_8UC1, jpegBuffer);
63  IplImage *pFrame =  cvDecodeImage(&jpgMat, 1);
64  IplImage *pResFrame = NULL;
65
66  if (true){
67    this->pSurf->process(pFrame);
68    pResFrame = pSurf->getFrame();
69
70    NewMessage(MSG_FROM_VISION, VISION_DATA);
71
72    if (pSurf->isDetected()){
73      CvPoint center = pSurf->getCenter();
74
75      AddBrianData("VisionPlayerDetected", 1.0, 1.0);
76      AddBrianData("PlayerAngle", center.x, 1);
77      //AddBrianData("PlayerAngle", ((center.x * 50.0) / 320.0)
            - 25, 1);
78      //cout << "Position: " << center.x << endl;
79
80      cvLine(pResFrame, cvPoint(center.x,0), cvPoint(center.x
            ,240), CV_RGB(255,255,255), 2);
81      cvLine(pResFrame, cvPoint(0,center.y), cvPoint(320,center
            .y), CV_RGB(255,255,255), 2);
82    } else {
83      AddBrianData("VisionPlayerDetected", 0.0, 1.0);
84    }
85
86    CloseCurrentMessage();
87    SendAllStringMessages();
88  }
89
90  if (pMutex)
91    pMutex->unlock();
92
93  if (m_bRecordVideo)
94    drawInfo(pResFrame, "Rec", 0);
95  else
96    drawInfo(pResFrame, "", 0);
97
98  if (gui)
99    gui->showImage(pResFrame);
100 else
```

```
101     cvShowImage("img", pResFrame);
102
103   char c = cvWaitKey(10) & 0xFF;
104
105   switch(c){
106     case 27:   //ESC
107       this->Shutdown(1);
108       break;
109     case 's':
110       sprintf(imgName, "./pics/pic %d.bmp\0", mFramesCount++);
111       cvSaveImage(imgName, pFrame);
112       cout << "Screenshot taken: " << imgName << endl;
113       break;
114     case 'r':
115       if (!m_bRecordVideo){
116         m_bRecordVideo = true;
117         m_videoWriter = cvCreateVideoWriter("./pics/video.avi",
                CV_FOURCC('t', 'h', 'e', 'o'),  10, cvSize
               (320,240), 1);
118         cout << "Recording video..." << endl;
119       } else {
120         cvReleaseVideoWriter(&m_videoWriter);
121         m_bRecordVideo = false;
122         cout << "Stop recording." << endl;
123       }
124       break;
125   }
126
127   if (m_bRecordVideo)
128     cvWriteFrame(m_videoWriter, pResFrame);
129
130   cvReleaseImage(&pFrame);
131 }
132
133 void VisionExpert::RunClose(){
134   printf("\nVisionExpert::RunClose()\n");
135
136   if (m_bRecordVideo)
137     cvReleaseVideoWriter(&m_videoWriter);
138 }
139
140 void VisionExpert::AddBrianData(char* name,float value,float
      rel){
141   if(fabs(value) < 0.001) //evito che vengano mandati valori in
         notazione scientifica
142     value = 0;
143
144   if(fabs(rel) < 0.001)
145     rel = 0;
```

```
146
147   mCurrentMessage << "<D>" << name << " " << value << " " <<
          rel << " </D>\n";
148 }
```

B.3 SurfModule

Listing B.4: src/SurfModule.h

```
1  #ifndef SURF_MODULE_H
2  #define SURF_MODULE_H
3
4  #include <opencv/cv.h>
5  #include <opencv/highgui.h>
6
7  #include <ctime>
8  #include <stdlib.h>
9  #include <stdio.h>
10 #include <string>
11
12 #include <log4cpp/Category.hh>
13 #include <log4cpp/BasicLayout.hh>
14 #include <log4cpp/SimpleLayout.hh>
15 #include <log4cpp/FileAppender.hh>
16 #include <log4cpp/OstreamAppender.hh>
17
18 using namespace std;
19
20
21 #ifdef _WIN32
22 #ifdef _DEBUG
23 #pragma comment(lib,"cv200d.lib")
24 #pragma comment(lib,"cxcore200d.lib")
25 #pragma comment(lib,"highgui200d.lib")
26 #else
27 #pragma comment(lib,"cv200.lib")
28 #pragma comment(lib,"cxcore200.lib")
29 #pragma comment(lib,"highgui200.lib")
30 #endif
31 #endif
32
33 #include <string>
34 #include <iostream>
35
36 using namespace std;
37 using namespace cv;
38
39 #define IMAGE_WIDTH   320
40 #define IMAGE_HEIGHT  240
```

```
41
42 #define SURF_THR 250
43
44 #define RED 0
45 #define GREEN 1
46 #define BLUE 2
47 #define ORANGE 3
48 #define VIOLET 4
49 #define YELLOW 5
50 #define BLACK 6
51 #define WHITE 7
52
53 class ModelData {
54 public:
55   ModelData();
56   ModelData(string _file, IplImage *image, CvSeq *objKey, Mat *
       objDesc, Mat *ind, Mat *dist, CvSize s, int a);
57   ~ModelData();
58
59   vector<CvPoint2D32f> pt1;
60   vector<CvPoint2D32f> pt2;
61
62   string file;         // file
63   IplImage *pImage;      // immagine
64   CvSeq *objKeypoints;     // features surf
65   Mat *objDescriptors;     // descrittori features surf
66   Mat *indices;
67   Mat *distances;
68   CvSize imgSize;        // dimensioni dell'immagine
69   int area;
70   CvHistogram *pHist;      // istogramma del colore
71 };
72
73 typedef vector<ModelData*> ModelsVector;
74
75 class SURFModule {
76 public:
77   SURFModule(int w, int h);
78   ~SURFModule();
79
80   void process(IplImage *pFrame);
81   void AddModel(string file);
82
83   static const CvScalar colors[];
84
85   IplImage *getFrame() const { return curFrame; }
86
87   ModelsVector mModels;
88
```

```
89    static void drawHist(CvHistogram *hist, IplImage *histimg);
90    bool isDetected() const { return mNumMatched >= 1; }
91    CvPoint getCenter() const { return mCenter; }
92
93  protected:
94    IplImage *LoadModel(string file, CvSeq **objKeypoints, CvSeq
          **objDescriptors, CvSize &size);
95    void createImages();
96    CvRect points2Rect(CvPoint pts[4]);
97
98    void deallocateHelper(CvSeq *camDesc, CvSeq *camKeypts,
          CvMemStorage *pLoopStorage);
99
100   CvMemStorage *mStorage;
101   CvMemStorage *mLoopStorage;
102   int mNModels;
103
104   int mNumMatched;
105   CvPoint mCenter;
106
107   CvSURFParams SURFParams;
108   cv::flann::KDTreeIndexParams knnTreeParams;
109   cv::flann::SearchParams searchParams;
110
111   double hMatrix[9];
112   CvMat _hMatrix;
113
114   CvPoint src_corners[4];
115     CvPoint dst_corners[4];
116
117   CvPoint squareModelPoints[4];
118   CvContour *mSquareModel;
119   CvContour *mApproxedSquare;
120   Ptr<CvHistogram> mHist;
121
122   IplImage *curFrame;
123   Ptr<IplImage> curGray;
124   Ptr<IplImage> curHSV;
125   IplImage *curHUE;
126   IplImage *curValue;
127   Ptr<IplImage> curBackProject;
128   Ptr<IplImage> histImage;
129
130   bool mInit;
131   int mHeight, mWidth;
132
133   //logger stuff
134   log4cpp::FileAppender *pFileApp;
135   log4cpp::OstreamAppender *pCoutApp;
```

```
136   log4cpp::Category &logger;
137 };
138
139
140 inline CvScalar hsv2rgb(float hue){
141     int rgb[3], p, sector;
142     static const int sector_data[][3]=
143         {{0,2,1}, {1,2,0}, {1,0,2}, {2,0,1}, {2,1,0}, {0,1,2}};
144     hue *= 0.033333333333333333333333333333333f;
145     sector = cvFloor(hue);
146     p = cvRound(255*(hue - sector));
147     p ^= sector & 1 ? 255 : 0;
148
149     rgb[sector_data[sector][0]] = 255;
150     rgb[sector_data[sector][1]] = 0;
151     rgb[sector_data[sector][2]] = p;
152
153     return cvScalar(rgb[2], rgb[1], rgb[0],0);
154 }
155
156 inline void drawInfo(IplImage *img, const char *s, int pos){
157   static int counter = 0;
158   static clock_t t;
159   static float fps;
160   char fps_text[50] = {0};
161   CvFont font;
162
163   cvInitFont(&font,CV_FONT_HERSHEY_COMPLEX_SMALL, 1,1,0);
164
165   // Add fps figure (every 10 frames)
166   if (counter > 10) {
167     fps = (10.0f/(clock()-t) * CLOCKS_PER_SEC);
168     t=clock();
169     counter = 0;
170   }
171
172   counter++;
173
174   // Get the figure as a string
175   sprintf(fps_text,"FPS: %.2f %s", fps, s);
176
177   // Draw the string on the image
178   cvPutText (img,fps_text,cvPoint(10,25), &font, cvScalar
          (255,255,0));
179 }
180 #endif
```

Listing B.5: src/SurfModule.cpp

```
1 #include "SurfModule.h"
```

```
2
3 const CvScalar SURFModule::colors[] = { CV_RGB(255,0,0), CV_RGB
    (0,255,0), CV_RGB(0,0,255),
4                           CV_RGB(255,255,0), CV_RGB(255,0,255),
                                CV_RGB(0,255,255),
5                   CV_RGB(0,0,0), CV_RGB(255,255,255) };
6
7 int hdims = 16;
8 float hranges_arr[] = {0,180};
9 float* hranges = hranges_arr;
10
11 SURFModule::SURFModule(int w, int h) : mHeight(h), mWidth(w),
    logger(log4cpp::Category::getInstance("SURF")) {
12   mNModels = 0;
13   mNumMatched = 0;
14   mInit = false;
15
16   pFileApp = new log4cpp::FileAppender("vision.surf.file", "log
       /SurfLog.txt", false);
17   pCoutApp = new log4cpp::OstreamAppender("vision.surf.cout", &
       cout);
18
19   pFileApp->setLayout(new log4cpp::BasicLayout());
20   pCoutApp->setLayout(new log4cpp::SimpleLayout());
21   logger.setAppender(pFileApp);
22   logger.setAppender(pCoutApp);
23
24   //pFileApp->setThreshold(log4cpp::Priority::NOTSET);
25   pCoutApp->setThreshold(log4cpp::Priority::INFO);
26
27   mStorage = cvCreateMemStorage(0);
28   mLoopStorage = cvCreateMemStorage(0);
29
30   curFrame = cvCreateImage(cvSize(w,h), 8, 3);
31   curGray = cvCreateImage(cvSize(w,h), 8, 1);
32
33   curHSV = cvCreateImage(cvSize(w,h), 8, 3);
34   curHUE = cvCreateImage(cvSize(w,h), 8, 1);
35   curValue = cvCreateImage(cvSize(w,h), 8, 1);
36   curBackProject = cvCreateImage(cvSize(w,h), 8, 1);
37
38   histImage = cvCreateImage(cvSize(320,200),8,3);
39
40   cvZero(curBackProject);
41   cvNot(curBackProject, curBackProject);
42
43   //Homography matrix
44   _hMatrix = cvMat(3, 3, CV_64F, hMatrix);
45
```

```
46   //Approssimazione a rettangolo
47   mApproxedSquare = (CvContour*)cvCreateSeq(CV_SEQ_POLYGON,
         sizeof(CvContour), sizeof(CvPoint), mStorage);
48   mSquareModel = (CvContour*)cvCreateSeq(CV_SEQ_POLYGON, sizeof
         (CvContour), sizeof(CvPoint), mStorage);
49
50   mHist = cvCreateHist(1, &hdims, CV_HIST_ARRAY, &hranges, 1);
51
52   // Rettangolo...
53   squareModelPoints[0] = cvPoint(0,0);
54   squareModelPoints[1] = cvPoint(1,0);
55   squareModelPoints[2] = cvPoint(1,1);
56   squareModelPoints[3] = cvPoint(0,1);
57
58   cvSeqPush((CvSeq*)mSquareModel, &squareModelPoints[0]);
59   cvSeqPush((CvSeq*)mSquareModel, &squareModelPoints[1]);
60   cvSeqPush((CvSeq*)mSquareModel, &squareModelPoints[2]);
61   cvSeqPush((CvSeq*)mSquareModel, &squareModelPoints[3]);
62
63   cvNamedWindow("hist");
64   cvNamedWindow("back");
65 }
66
67 SURFModule::~SURFModule(){
68   cvReleaseImage(&curFrame);
69   cvReleaseImage(&curHUE);
70   cvReleaseImage(&curValue);
71
72   for (ModelsVector::iterator it = mModels.begin(); it !=
         mModels.end(); it++){
73     logger.info("Model %s deleted\n", (*it)->file.c_str());
74     delete *it;
75   }
76
77   cvDestroyAllWindows();
78
79   cvReleaseMemStorage(&mStorage);
80   cvReleaseMemStorage(&mLoopStorage);
81 }
82
83
84 void SURFModule::process(IplImage *pFrame){
85   cvCopy(pFrame, curFrame, NULL);
86   int mIndex = 0;
87
88   createImages();
89   //cvEqualizeHist(curGray,curGray);
90
91   /*** ESTRAZIONE SURF ***/
```

```
92  CvSeq  *camKeypoints, *camDescriptors;    //keypoints e
        descrittori dell'immagine di spykee
93  try {
94    cvExtractSURF(curGray, curBackProject, &camKeypoints, &
          camDescriptors, mLoopStorage, cvSURFParams(SURF_THR,1))
          ;
95  } catch (cv::Exception &e){
96    //deallocateHelper(camDescriptors, camKeypoints,
          mLoopStorage);
97    cerr << "Error" << endl;
98    cvClearMemStorage(mLoopStorage);
99    return;
100 }
101
102 if (camDescriptors->total <= 0){
103   deallocateHelper(camDescriptors, camKeypoints, mLoopStorage
          );
104   return;
105 }
106
107 int length = (int)(camDescriptors->elem_size/sizeof(float));
108 cv::Mat camDescMatrix(camDescriptors->total, length, CV_32F);
109
110 CvSeqReader reader;
111 float *ptr = camDescMatrix.ptr<float>(0);
112 cvStartReadSeq(camDescriptors, &reader);
113
114 for(int i = 0; i < camDescriptors->total; i++){
115   const float *pDescriptor = (const float*)reader.ptr;
116   CV_NEXT_SEQ_ELEM(reader.seq->elem_size, reader);
117   memcpy(ptr, pDescriptor, length*sizeof(float));
118   ptr += length;
119 }
120
121 /*** ***/
122
123
124 for (mIndex = 0; mIndex < mModels.size(); mIndex++){
125   ModelData *pModel = mModels.at(mIndex);
126
127   CvPoint src_corners[4] = {{0,0}, {pModel->imgSize.width,0},
          {pModel->imgSize.width, pModel->imgSize.height}, {0,
          pModel->imgSize.height}};
128   CvPoint dst_corners[4];
129
130   if (camDescriptors->total <= 0)
131     break;
132
133   // Inizializzazione KNN
```

```
134     cv::flann::Index flann_index(camDescMatrix, cv::flann::
            KDTreeIndexParams(4));
135     flann_index.knnSearch(*(pModel->objDescriptors), *(pModel->
            indices), *(pModel->distances), 2, cv::flann::
            SearchParams(64));
136
137     /*** Confronto SURF modello - SURF immagine -> KNN ***/
138     int *indicesPtr = pModel->indices->ptr<int>(0);
139     float *distPtr = pModel->distances->ptr<float>(0);
140     int count = 0;
141
142     pModel->pt1.resize(pModel->indices->rows);
143     pModel->pt2.resize(pModel->indices->rows);
144
145     for (int i = 0; i < pModel->indices->rows; i++){
146       CvSURFPoint *spt1 = ((CvSURFPoint*)cvGetSeqElem(pModel->
              objKeypoints, i));
147       CvSURFPoint *spt2 = ((CvSURFPoint*)cvGetSeqElem(
              camKeypoints, indicesPtr[2*i]));
148
149       if (distPtr[2*i] < 0.6*distPtr[2*i + 1]){
150         pModel->pt1[count] = spt1->pt;
151         pModel->pt2[count++] = spt2->pt;
152
153         cvCircle(curFrame, ::cvPointFrom32f(spt2->pt), cvRound(
                spt2->size*1.2/9.*2), colors[mIndex], 1);
154       }
155     }
156
157     /*** ***/
158
159     /*** Se > 4 features -> homography ***/
160     // Se < 4 carsh!!!! //
161
162     if (count >= 4){
163       CvMat _pt1 = cvMat(1, count, CV_32FC2, &(pModel->pt1[0]))
              ;
164       CvMat _pt2 = cvMat(1, count, CV_32FC2,  &(pModel->pt2[0])
              );
165
166       if (cvFindHomography(&_pt1, &_pt2, &_hMatrix, CV_RANSAC,
              5)){
167         for(int i = 0; i < 4; i++){
168           double x = src_corners[i].x, y = src_corners[i].y;
169           double Z = 1./(hMatrix[6]*x + hMatrix[7]*y + hMatrix
                  [8]);   //scala ?
170           double X = (hMatrix[0]*x + hMatrix[1]*y + hMatrix[2])
                  *Z;
```

```
171           double Y = (hMatrix[3]*x + hMatrix[4]*y + hMatrix[5])
                  *Z;
172           dst_corners[i] = cvPoint(cvRound(X), cvRound(Y));
173         }
174
175         cvSeqPush((CvSeq*)mApproxedSquare, &dst_corners[0]);
176         cvSeqPush((CvSeq*)mApproxedSquare, &dst_corners[1]);
177         cvSeqPush((CvSeq*)mApproxedSquare, &dst_corners[2]);
178         cvSeqPush((CvSeq*)mApproxedSquare, &dst_corners[3]);
179
180         CvSeq *res = cvApproxPoly(mApproxedSquare,
                mApproxedSquare->header_size, mLoopStorage,
                CV_POLY_APPROX_DP, cvContourPerimeter(
                mApproxedSquare)*0.02);
181
182         // Se assomiglia a un rettangolo....
183         if ((cvMatchShapes(res, mSquareModel,
                CV_CONTOURS_MATCH_I1) < 0.1) &&
184               res->total == 4 && cvCheckContourConvexity(res))
                      {
185
186           CvRect rc = points2Rect(dst_corners);
187           cvDrawRect(curFrame, cvPoint(rc.x,rc.y), cvPoint(rc.x
                  +rc.width,rc.y+rc.height), colors[mIndex], 3);
188
189           if (mNumMatched < 3)
190             mNumMatched++;
191
192           if (!mInit && isDetected()){
193             try {
194               cvSetImageROI(curHUE, rc);
195               cvCalcHist(&curHUE, mHist, NULL, NULL);
196               cvResetImageROI(curHUE);
197               mInit = true;
198             } catch (cv::Exception &e){
199               cout << "OpenCV EXCEPTION: " << e.err << endl;
200               break;
201             }
202           }
203
204           float max_val = 0.0 ;
205           cvGetMinMaxHistValue(mHist, 0, &max_val, 0, 0);
206           cvConvertScale(mHist->bins, mHist->bins, max_val ?
                  255.0 / max_val : 0.0, 0);
207
208           // se è la prima volta che becco il marker creo anche
                   l'istogramma del colore,
209           // altrimenti uso quello del modello corrente
210           if (mInit){
```

```
211             cvCalcBackProject(&curHUE, curBackProject, mHist);
212             drawHist(mHist, histImage);
213           } else {
214             cvCalcBackProject(&curHUE, curBackProject, pModel->
                    pHist);
215             drawHist(pModel->pHist, histImage);
216           }
217
218           // calcolo il centro del rettangolo che contiene il
                marker, in px
219           if (isDetected())
220             mCenter = cvPoint(rc.x + (rc.width/2) , rc.y + (rc.
                    height/2));
221
222           cvShowImage("hist", histImage);
223
224           // pulisco i vector ---> poco efficente
225           pModel->pt1.clear();
226           pModel->pt2.clear();
227           cvClearSeq((CvSeq*)mApproxedSquare);
228           break;
229         }
230
231         cvClearSeq((CvSeq*)mApproxedSquare);
232
233       } else {  //cvFindHomography
234         if (mNumMatched > 0)
235           mNumMatched--;
236
237         if (mNumMatched == 0){
238           cvZero(curBackProject);
239           cvNot(curBackProject,curBackProject);
240           mInit = false;
241         }
242       }
243
244     } else  { // count >= 4
245       if (mNumMatched > 0)
246         mNumMatched--;
247
248       if (mNumMatched == 0){
249         cvZero(curBackProject);
250         cvNot(curBackProject,curBackProject);
251         mInit = false;
252       }
253     }
254
255     /*** ***/
256
```

```
257   } // Ciclo modelli
258
259   cvShowImage("back", curBackProject);
260
261   //svuoto le pile e pulisco la memoria
262   deallocateHelper(camDescriptors, camKeypoints, mLoopStorage);
263 }
264
265 void SURFModule::deallocateHelper(CvSeq *camDesc, CvSeq *
      camKeypts, CvMemStorage *pLoopStorage){
266   cvClearSeq(camDesc);
267   cvClearSeq(camKeypts);
268   cvClearMemStorage(pLoopStorage);
269 }
270
271
272 // Utility: calcolo immagini per ogni loop, sovrascrivo alle
      precedenti
273 void SURFModule::createImages(){
274   cvCvtColor(curFrame, curGray, CV_BGR2GRAY);
275   cvCvtColor(curFrame, curHSV, CV_BGR2HSV);
276   cvSplit(curHSV, curHUE, NULL, NULL, NULL);
277 }
278
279 CvRect SURFModule::points2Rect(CvPoint pts[4]){
280   int x1 = pts[0].x < pts[3].x ? pts[0].x : pts[3].x;
281   int x2 = pts[1].x > pts[2].x ? pts[1].x : pts[2].x;
282   int y1 = pts[0].y < pts[1].y ? pts[0].y : pts[1].y;
283   int y2 = pts[2].y < pts[3].y ? pts[2].y : pts[3].y;
284
285   return cvRect(x1, y1, x2 - x1, y2 - y1);
286 }
287
288 // Carica i modelli dell'immagine
289 IplImage *SURFModule::LoadModel(string file, CvSeq **
      objKeypoints, CvSeq **objDescriptors, CvSize &size){
290   IplImage *pImage = cvLoadImage(file.c_str(), 1);
291   Ptr<IplImage> pGray = cvLoadImage(file.c_str(), 0);
292
293   cvCvtColor(pImage, pGray, CV_BGR2GRAY);
294
295   size.height = pImage->height;
296   size.width = pImage->width;
297
298   cvExtractSURF(pGray, NULL, objKeypoints, objDescriptors,
        mStorage, cvSURFParams(SURF_THR,1));
299
300   for (int i = 0; i < (*objKeypoints)->total; i++){
```

```
301     CvSURFPoint *r = ((CvSURFPoint*)cvGetSeqElem(*objKeypoints,
            i));
302     CvPoint p = cvPointFrom32f(r->pt);
303     cvCircle(pImage, p, cvRound(r->size*1.2/9.*2), colors[
            mNModels]);
304   }
305
306   return pImage;
307 }
308
309 void SURFModule::AddModel(string file){
310   ModelData *data = new ModelData;
311   CvSeq *objDescriptors;
312
313   data->pImage = LoadModel(file, &data->objKeypoints, &
          objDescriptors, data->imgSize);
314   int length = (int)(objDescriptors->elem_size/sizeof(float));
315
316   mNModels++;
317
318   data->file = file;
319   data->area = data->imgSize.width * data->imgSize.height;
320   data->objDescriptors = new cv::Mat(objDescriptors->total,
          length, CV_32F);
321   data->indices = new cv::Mat(objDescriptors->total, 2, CV_32S)
          ;
322   data->distances = new cv::Mat(objDescriptors->total, 2,
          CV_32F);
323
324   data->pHist = cvCreateHist(1, &hdims, CV_HIST_ARRAY, &hranges
          , 1);
325
326   Ptr<IplImage> hsv = cvCreateImage(cvGetSize(data->pImage), 8,
           3);
327   IplImage *hue = cvCreateImage(cvGetSize(data->pImage), 8, 1);
328
329   cvCvtColor(data->pImage, hsv, CV_BGR2HSV);
330   cvSplit(hsv, hue, NULL, NULL, NULL);
331   cvCalcHist(&hue, data->pHist, NULL, NULL);
332   cvReleaseImage(&hue);
333
334   float max_val = 0.0 ;
335   cvGetMinMaxHistValue(data->pHist, 0, &max_val, 0, 0);
336   cvConvertScale(data->pHist->bins, data->pHist->bins, max_val
          ? 255. / max_val : 0., 0);
337
338   /* Copy model descriptors in a Mat class because of flann */
339     CvSeqReader reader;
340   float *ptr = data->objDescriptors->ptr<float>(0);
```

```
341
342     cvStartReadSeq(objDescriptors, &reader);
343
344     for(int i = 0; i < objDescriptors->total; i++) {
345         const float *pDescriptor = (const float*)reader.ptr;
346         CV_NEXT_SEQ_ELEM(reader.seq->elem_size, reader);
347         memcpy(ptr, pDescriptor, length*sizeof(float));
348         ptr += length;
349   }
350
351     cvClearSeq(objDescriptors);
352
353   mModels.push_back(data);
354
355   //printf("Modello: %s\n\t\n\tKeypoints: %d\n\tSize: %dx%d\n",
          file.c_str(), data->objKeypoints->total, data->imgSize.
        width, data->imgSize.height);
356   logger.info("Modello:%s", file.c_str());
357   logger.info("Keypoints: %d", data->objKeypoints->total);
358   logger.info("Size: %dx%d", data->imgSize.width, data->imgSize
        .height);
359   logger.info("");
360 }
361
362 //Utility: disegna l'istogramma del colore
363 void SURFModule::drawHist(CvHistogram *hist, IplImage *histimg)
      {
364   cvZero(histimg);
365   int bin_w = histimg->width / hdims;
366
367   for(int i = 0; i < hdims; i++) {
368     int val = cvRound(cvGetReal1D(hist->bins,i)*histimg->height
          /255);
369     CvScalar color = hsv2rgb(i*180.f/hdims);
370     cvRectangle(histimg, cvPoint(i*bin_w,histimg->height),
          cvPoint((i+1)*bin_w,histimg->height - val),
371     color, -1, 8, 0);
372   }
373 }
374
375 //Struttura dati per i modelli
376 ModelData::ModelData(){
377   pImage = NULL;
378   objKeypoints = NULL;
379   objDescriptors = NULL;
380   indices = NULL;
381   distances = NULL;
382   imgSize.height = imgSize.width = 0;
383   area = 0.0;
```

```
384 }
385
386 ModelData::ModelData(string _file, IplImage *image, CvSeq *
      objKey, Mat *objDesc, Mat *ind, Mat *dist, CvSize s, int a)
      {
387   file = _file;
388   pImage = image;
389   objKeypoints = objKey;
390   objDescriptors = objDesc;
391   indices = ind;
392   distances = dist;
393   imgSize = s;
394   area = a;
395 }
396
397 ModelData::~ModelData(){
398   if (pImage != NULL)
399     cvReleaseImage(&pImage);
400
401   cvClearSeq(objKeypoints);
402
403   if (objDescriptors)
404     delete objDescriptors;
405
406   if (indices)
407     delete indices;
408
409   if (distances)
410     delete distances;
411
412   if (pHist)
413     cvReleaseHist(&pHist);
414 }
```

B.4 BrainExpert

Listing B.6: src/BrianExpert.h

```
1 #ifndef brianExpert_h
2 #define brianExpert_h
3
4 #include <string.h>
5
6 #include <const.h>
7
8 #include <brian.h>
9 #include <getFuzzy.h>
10
11 #include "StringModuleMember.h"
```

```
12 #include <const.h>
13 //#include "mylogger.h"
14 #include <iomanip>
15
16 #ifdef DMALLOC
17 #include <dmalloc.h>
18 #endif
19
20 extern int MessageLineNum;
21 #define MOTOR_COMMAND   "MotorCommand"
22
23 #define SPYKEE_NORMAL   -1
24 #define SPYKEE_TURN_RIGHT  0
25 #define SPYKEE_TURN_LEFT   1
26 #define SPYKEE_TURN_BACK   2
27 #define SPYKEE_AUTO_FORWARD  3
28
29 #define STATUS_SEARCHING  0
30 #define STATUS_ESCAPING   1
31 #define STATUS_HIDING   2
32
33 typedef map< string, std::pair<float,float> > BrianMap;
34
35 class BrianExpert: public StringModuleMember{
36  public:
37   static const string msg_always;
38  public:
39   BrianExpert(ModuleDispatch* dis,long period);
40   ~BrianExpert();
41
42   void RunInit();
43   void RunDuty();
44   void RunClose();
45
46   void ParseStringMessages();
47   void FillMessages(command_list * cl, action_list * al,
       predicate_list * linkcadl, weight_want_list* linkwwl,
       predicate_list* linkpdl);
48
49   void SetValue(const char * name, float val, float rel);
50   void AddValueToMap(const char *name, float val, float rel =
       1.0);
51
52  private:
53   enum { MOTOR_AUTO_RIGHT = 0, MOTOR_AUTO_LEFT = 1,
       MOTOR_NORMAL = -1 };
54   int mMotorState;
55   int mGameStatus;
56   int mEscapeTimeout;
```

```
57   bool mOldClock;
58
59   // function used to keep values in cdl up to date
60   //void BuildData(command_list* cl);
61
62   struct timeval last_render[1];
63   long elapsed(int);
64   void settimer(int);
65
66   void debug();
67
68   MrBrian * TheBrian;
69
70   void AddBehaviorData(string name, float cando_value, float
         want_value);
71   void AddFuzzyData(string name,float value,float reliability);
72   void AddCommandData(string name, float value);
73   void RemoveMapValue(string name);
74
75   BrianMap mCL;
76   //MyLogger mLogger;
77   ofstream mLogger;
78
79   ofstream logfile;
80   ofstream brianmsglog;
81 };
82
83 #endif
```

Listing B.7: src/BrianExpert.cpp

```
1  #include "BrianExpert.h"
2  #include <iostream>
3  #include <stc.h>
4  #include <msg.h>
5  #include <MrtMisc.h>
6
7  #include "smdebug.h"
8  #include "Configurator.h"
9
10 extern void brianlex_memory(const char* src, BrianExpert* brian
       );
11
12 //#define BRIAN_DEBUG 1 //*wiim*
13 //#define EXPERTS_DEBUG
14
15 const string BrianExpert::msg_always = "Always";
16
17 BrianExpert::BrianExpert(ModuleDispatch* kernel,long period) :
18   StringModuleMember(kernel,period,"brian"){
```

```
19
20  //m_bTurning = false;
21
22 #ifdef EXPERTS_DEBUG
23   cout<<"\n----------------> Starting Brian" << endl;
24 #endif
25
26   mLogger.open("log/mylog.txt", ios::out);
27
28   TheBrian = new MrBrian(
29   (char*)Configurator::brian_ctof.c_str(),
30   (char*)Configurator::brian_ctof_shapes.c_str(),
31   (char*)Configurator::brian_predicate.c_str(),
32   (char*)Configurator::brian_predicate_actions.c_str(),
33   (char*)Configurator::brian_cando.c_str(),
34   (char*)Configurator::brian_behaviour.c_str(),
35   (char*)Configurator::brian_want.c_str(),
36   (char*)Configurator::brian_ftoc.c_str(),
37   (char*)Configurator::brian_ftoc_shapes.c_str());
38 }
39
40 BrianExpert::~BrianExpert(){
41   logfile.close();
42   mLogger.close();
43   delete TheBrian;
44 }
45
46 void BrianExpert::RunInit(){
47  mLogger << "*** BRIAN INIT ***" << endl;
48
49  AddMessageRequest(MSG_FROM_SONAR,LOCAL);
50  AddMessageRequest(MSG_FROM_WIIMOTE, LOCAL);
51  AddMessageRequest(MSG_FROM_VISION, LOCAL);
52
53  logfile.open(BRIANFULLLOG);
54  brianmsglog.open(BRIANMSGLOG, ios::out|ios::app);
55
56  AddValueToMap("GameStatus", STATUS_SEARCHING, 1);    //
       SEARCHING
57  AddValueToMap("Always", 1, 1);           // TRUE -- utility
58  AddValueToMap("RotSearch", 0, 1);            // LEFT
59
60  mMotorState = MOTOR_NORMAL;
61  mEscapeTimeout = Configurator::escape_timeout;
62  mOldClock = false;
63  settimer(0);
64 }
65
66 void BrianExpert::RunClose(){
```

```
67  mLogger << "\n*** BRIAN CLOSE ***" << endl;
68  logfile.close();
69  brianmsglog.close();
70 }
71
72 void BrianExpert::RunDuty(){
73  bool bReceived = false;
74  bool bStop = false;
75
76  unsigned char *buffer = NULL;
77
78  if (ReceiveLastMessage(MSG_FROM_SONAR, buffer, false) > 0){
79   brianmsglog << '\n' << buffer << endl;
80   brianlex_memory((const char*)buffer, this);
81   bReceived = true;
82   buffer = NULL;
83  }
84
85  if (ReceiveLastMessage(MSG_FROM_VISION, buffer, false) > 0){
86   brianmsglog << '\n' << buffer << endl;
87   brianlex_memory((const char*)buffer, this);
88   bReceived = true;
89   buffer = NULL;
90  }
91
92  if (ReceiveLastMessage(MSG_FROM_WIIMOTE, buffer, false) > 0){
93   brianmsglog << '\n' << buffer << endl;
94   brianlex_memory((const char*)buffer, this);
95   bReceived = true;
96   buffer = NULL;
97  }
98
99  // Se non ho inviato il segnale di gioco, non faccio nulla
100 if (mCL.find("GameStart") == mCL.end() || (mCL.find("GameStart
      ") != mCL.end() && mCL.find("GameStart")->second.first ==
      0)){
101  // resetto mappa e brian
102  mCL.clear();
103
104  // workaroud -> brian problems!
105  TheBrian->run();
106    TheBrian->flush();
107
108  // Imposto i valori iniziali
109  AddValueToMap("GameStatus", STATUS_SEARCHING, 1);   //
       SEARCHING
110  AddValueToMap("Always", 1, 1);              // TRUE -- utility
111  AddValueToMap("RotSearch", 0, 1);             // LEFT
112  AddValueToMap("GameStart", 0, 1);
```

```
113
114   mMotorState = MOTOR_NORMAL;
115   mEscapeTimeout = Configurator::escape_timeout;
116   mOldClock = false;
117
118   return;
119  }
120
121 //  cout << "RUN" << endl;
122
123  BrianMap::iterator clkIt = mCL.find("Clock");        // clock
       -- settato da WiimExpert
124  BrianMap::iterator statusIt = mCL.find("GameStatus");    //
       Stato del gioco
125  BrianMap::iterator irIt = mCL.find("IRCenter");       //
       Puntato ?
126  BrianMap::iterator vIt = mCL.find("VisionPlayerDetected");  //
        Player in vista
127
128  /* Se il robot sta scappando da piu di 10 secondi e non viene
       nuovamente puntato e non ha trovato un nascondiglio,
129   * ritorna in stato di ricerca.
130   * Se viene nuovamente puntato il timer si resetta.
131   */
132
133  if (irIt == mCL.end() || vIt == mCL.end()){ // Se non è
       puntato e non vede il giocatore
134    if (clkIt != mCL.end() && statusIt->second.first ==
         STATUS_ESCAPING){ // Se sta scappando
135     if (((bool)clkIt->second.first) != mOldClock){      // ed
          è passato un altro secondo
136      mOldClock = (bool)clkIt->second.first;         // imposto
          il clock attuale
137    mEscapeTimeout--;                    // decremento di un
       altro secondo
138    cout << "Escape reset in: " << mEscapeTimeout << endl; //
       debug
139     }
140    }
141  } else { // Se il robot è puntato o vede il giocatore
142    mEscapeTimeout = Configurator::escape_timeout; //continua a
       scappare
143  }
144
145  /*
146   * Timeout fuga: se dopo 10 (default) secondi il robot non
       viene nuovamente puntato o non trova un nascondiglio
147   * torno in àmodalit di ricerca
148   */
```

```
149
150 if (mEscapeTimeout <= 0){
151   mEscapeTimeout = Configurator::escape_timeout;
152   AddValueToMap("GameStatus", STATUS_SEARCHING, 1);
153   mLogger << "INFO: EscapeReset" << endl;
154 }
155
156
157 // Copio i valori della mappa nella crisp_data_list
158 for (BrianMap::iterator it = mCL.begin(); it != mCL.end(); it
      ++)
159  SetValue(it->first.c_str(), it->second.first, it->second.
       second); //name - value - rel = 1
160
161 // Mando lo stato del gioco al wiimote expert per
      visualizzarlo a console / gui
162 NewMessage(MSG_FROM_BRIAN,"info");
163 mCurrentMessage << "<D>GameStatus " << mCL.find("GameStatus")
      ->second.first << "</D>";
164 CloseCurrentMessage();
165 SendAllStringMessages();
166
167 // Eseguo Mr. Brian
168 TheBrian->run();
169
170 // Ottengo le variabili crisp in uscita
171 command_list *cmdList = TheBrian->getFuzzy()->
      get_command_singleton_list();
172
173 /*
174  * Retroazione dello stato di gioco (SEARCH, ESCAPE, HIDE)
175  */
176
177 command *cmdStatus = cmdList->get_command("GameStatus");
178
179 if (cmdStatus){
180  mGameStatus = cmdStatus->get_set_point();
181  AddValueToMap("GameStatus", (int)mGameStatus);
182 }
183
184 /*
185  * Retoazione della direzione di ricerca
186  */
187
188 command *cmdRotSearch = cmdList->get_command("RotSearch");
189
190 if (cmdRotSearch)
191  AddValueToMap("RotSearch", cmdRotSearch->get_set_point());
192
```

```
193 /*
194  * Gestion della rotazione a sx/dx di °90 gradi
195  */
196
197 BrianMap::iterator gsCl = mCL.find("GameStatus");
198
199 // Se il robot si sta nascondendo
200 if (gsCl != mCL.end() && gsCl->second.first == STATUS_HIDING){
201   command *cmdMotor = cmdList->get_command(MOTOR_COMMAND);
202   predicate *irExists = TheBrian->get_predicate_list()->get("
          IRExist");
203
204  // Se il robot è puntato
205   if (irExists){
206   mMotorState = MOTOR_NORMAL;
207   mEscapeTimeout = Configurator::escape_timeout;
208   // Distattiva l'attivazione delle regole Hides
209   RemoveMapValue("AutoTurn");
210   RemoveMapValue("GoForward");
211
212   //Riporta il robot nello stato di fuga
213   AddValueToMap("GameStatus", STATUS_ESCAPING);
214   cout << "End hide -> ESCAPE" << endl;
215
216   goto out;
217  }
218
219  // Il robot vuole nascondersi a destra
220  if (cmdMotor && cmdMotor->get_set_point() ==
          SPYKEE_TURN_RIGHT){
221  // Qui inizia la rotazione, MOTOR_NORMAL indica che il
          controllo dei motori è affidato a
222  // MotorExpert
223  if (mMotorState==MOTOR_NORMAL){
224      // Per tenere attiva la regola HideRight
225    AddValueToMap("AutoTurn", 1, 1);
226    mMotorState = SPYKEE_TURN_RIGHT;
227    //Inizializzo il timer per caloclare °90
228    settimer(0);
229    cout << "Start turning right" << endl;
230  }
231
232  /* Quando il timer dice che il tempo è passato, il robot
          smette di girare
233   * e avanza per nascondersi
234   */
235
236  if (elapsed(0) > Configurator::turn_time && (mMotorState==
          SPYKEE_TURN_RIGHT)){
```

```
237     mMotorState = SPYKEE_AUTO_FORWARD;
238     AddValueToMap("GoForward", 1, 1);
239     cout << "End turning right -> GoForward" << endl;
240     settimer(0);
241    }
242
243   } else if (cmdMotor && cmdMotor->get_set_point() ==
        SPYKEE_TURN_LEFT){
244    /*
245     * Come sopra, ma per girare a sinsitra
246     */
247     if (mMotorState==MOTOR_NORMAL){
248     //Per tenere attiva la regola HideLeft
249     AddValueToMap("AutoTurn", 0, 1);
250     mMotorState = SPYKEE_TURN_LEFT;
251     settimer(0);
252     cout << "Start turning left" << endl;
253     }
254
255     if (elapsed(0) > Configurator::turn_time && (mMotorState==
          SPYKEE_TURN_LEFT)){
256     mMotorState = SPYKEE_AUTO_FORWARD;
257     AddValueToMap("GoForward", 1, 1);
258     cout << "End turning left -> GoForward" << endl;
259     settimer(0);
260     }
261    }
262   } else if (gsCl != mCL.end() && gsCl->second.first ==
        STATUS_SEARCHING){
263     command *cmdMotor = cmdList->get_command(MOTOR_COMMAND);
264     /*
265      * Il robot si è nascosto
266      */
267     if ((cmdMotor && cmdMotor->get_set_point() == 6) && (
          mMotorState == SPYKEE_AUTO_FORWARD)){
268      /*
269       * Restituisco il controllo dei motori al MotorExpert e
            torno in àmodalit di ricerca
270       */
271      mMotorState = MOTOR_NORMAL;
272      RemoveMapValue("AutoTurn");
273    RemoveMapValue("GoForward");
274    cout << "End hide -> Search" << endl;
275
276    AddValueToMap("GameStatus", STATUS_SEARCHING, 1.0);
277
278    /*
279     * Se non si è nascosto bene ricomincia a cercare un
          nascondiglia
```

```
280     */
281    predicate *pRight = TheBrian->get_predicate_list()->get("
           RightBusy");
282    predicate *pLeft = TheBrian->get_predicate_list()->get("
           LeftBusy");
283
284    if (pRight == NULL && pLeft == NULL){
285      AddValueToMap("GameStatus", STATUS_ESCAPING, 1.0);
286      this->mEscapeTimeout = Configurator::escape_timeout;
287      cout << "HIDE FAILED -> Escape again" << endl;
288    }
289
290     }
291  }
292
293  // Se ricevo qualcosa da qualche expert lo loggo
294 /*  if (bReceived){
295
296  }
297 */
298  // Creazione file di log
299  TheBrian->log();
300  debug();
301
302  // Se ho qualcosa da inviare lo mando ai motori
303  if (cmdList->size() > 0){
304    NewMessage(MSG_TO_MOTION, COMMAND_DATA);
305
306    for (command_list::iterator it = cmdList->begin(); it !=
           cmdList->end(); it++)
307    AddCommandData(it->first, it->second->get_set_point());
308
309    CloseCurrentMessage();
310    SendAllStringMessages();
311  }
312
313 out:
314  // Pulisco la memoria di Mr. Brian
315  TheBrian->flush();
316
317  /* Cancello i valori che indicano se il robot è puntato,
318   * altrimenti una volta puntato rimarrebbe sempre in questo
           stato
319   */
320
321  RemoveMapValue("IRCenter");
322  RemoveMapValue("IRDist");
323 }
324
```

```
325 void BrianExpert::RemoveMapValue(string name){
326  mCL.erase(name);
327  mLogger << "MANUAL: " << name << " removed" << endl;
328 }
329
330 // Utility
331 void BrianExpert::ParseStringMessages(){
332  unsigned char *buffer = NULL;
333
334  if (ReceiveLastMessage(MSG_TO_BRIAN,buffer,true)>0){
335   FILE* fp=fopen(BRIANMSGLOG,"a");
336   fprintf(fp,"\n%s\n",buffer);
337   fclose(fp);
338   brianlex_memory((const char*)buffer,this);
339  }
340 }
341
342 void BrianExpert::FillMessages(command_list * cl, action_list *
       al, predicate_list* linkcadl, weight_want_list* linkwwl,
      predicate_list* linkpdl){
343  // create the message to return
344
345  command_multimap::iterator c_it;
346  command* c = 0;
347  for (c_it = cl->begin(); c_it != cl->end(); c_it++){
348   c = c_it->second;
349   AddCommandData(c_it->first,c->get_set_point());
350   cout << c_it->first << ": " << c->get_set_point() << ' ';
351  }
352
353  cout << endl;
354 }
355
356 void BrianExpert::SetValue(const char * name, float val, float
      rel){
357  // search for the crisp_data whose naAddCommandData("RotSpeed
       ", 0.0);me is "name" and sets its value and reliability to
        "val" and "rel"
358  crisp_data_list * cdl = (TheBrian->getFuzzy())->
       get_crisp_data_list();
359  crisp_data_list::iterator i=cdl->find(name);
360
361  if (i != cdl->end()){
362   (*i).second->set_value(val);
363   (*i).second->set_reliability(rel);
364  } else {
365    cdl->add(new crisp_data(name,val,rel));
366   }
367 }
```

```
368
369 void BrianExpert::AddBehaviorData(string name, float
        cando_value, float want_value){
370  mCurrentMessage << "<B>" << name << " " << cando_value << " "
         << want_value << "</B>\n";
371 }
372
373 void BrianExpert::AddFuzzyData(string name, float value, float
        reliability){
374  mCurrentMessage << "<F>" << name << " " << value << " " <<
         reliability << "</F>\n";
375 }
376
377
378 void BrianExpert::AddValueToMap(const char *name, float val,
        float rel){
379  mCL[name] = std::pair<float,float>(val, rel);
380 }
381
382 void BrianExpert::AddCommandData(string name, float value){
383  if (fabs(value) < 0.001) //*wiim* evito che vengano mandati
         valori in notazione scientifica
384   value = 0;
385
386  mCurrentMessage << "<C>" << name << ' ' << value << "</C>\n";
387 }
388
389 long BrianExpert::elapsed(int i){
390  struct timeval now;
391  long elapsed_usec = 0;
392
393  gettimeofday(&now, NULL);
394
395  if (now.tv_usec > 1000000) {
396   now.tv_usec -= 1000000;
397   ++now.tv_sec;
398  }
399
400  if (now.tv_sec > last_render[i].tv_sec){
401   elapsed_usec = ((now.tv_sec - last_render[i].tv_sec) *
         1000000);
402  }
403
404  if (now.tv_usec > last_render[i].tv_usec)
405   elapsed_usec += now.tv_usec - last_render[i].tv_usec;
406  else
407   elapsed_usec -= last_render[i].tv_usec - now.tv_usec;
408
409  return elapsed_usec ;
```

```
410 }
411
412 void BrianExpert::settimer(int i){
413  gettimeofday(&(last_render[i]),NULL);
414 }
415
416 void BrianExpert::debug(){
417  struct timeval now;
418  gettimeofday(&now,NULL);
419
420  //ostringstream ostr;
421  mLogger << "\n\n" << now.tv_sec << "_" << now.tv_usec << "
        Brian debug start **********************\n" << endl;
422  mLogger.setf(ios::left, ios::adjustfield);
423  //mLogger.setf(ios::fixed, ios::floatfield);
424
425  crisp_data_list *cl = TheBrian->getFuzzy()->
        get_crisp_data_list();
426  mLogger << "--------- Crisp Data - Length: " << cl->size() <<
        "---------" << endl;
427  for (crisp_data_list::iterator it = cl->begin(); it != cl->end
        (); it++){
428   mLogger << "CRISP: " << setw(20) << it->second->get_name()
429       << " = " << fixed << setw(4) << it->second->get_value()
              << "\t"
430       << setw(4) << it->second->get_reliability() << endl;
431  }
432  mLogger << "----- End Crisp Data -----\n" << endl;
433
434
435  fuzzy_data_list *fl = TheBrian->getFuzzy()->
        get_fuzzy_data_list();
436  mLogger << "--------- Fuzzy Data - Length: " << fl->size() <<
        " ---------" << endl;
437  for (fuzzy_data_list::iterator it = fl->begin(); it != fl->end
        (); it++){
438   mLogger << "FUZZY: " << setw(20) << it->second->get_name()
439       << " = " << setw(15) << it->second->get_label()
440       << setw(4) << it->second->get_membership_value() << endl;
441  }
442  mLogger <<  "--------- Fuzzy Data End -----\n" << endl;
443
444
445  predicate_list *pl = TheBrian->get_predicate_list();
446  mLogger << "--------- Predicates - Length: " << pl->size() <<
        " ---------" << endl;
447  for (predicate_list::iterator it = pl->begin(); it != pl->end
        (); it++){
```

```
448   mLogger << "PREDICATE: " << setw(20) << it->second->get_name
          ()
449       << " = " << setw(4) << it->second->get_value()
450       << " rel = " << setw(4) << it->second->get_reliability()
              << endl;
451   }
452   mLogger << "----- End Predicates -----\n" << endl;
453
454
455   predicate_list *canl = TheBrian->get_cando_list();
456   mLogger << "--------- Cando - Length: " << canl->size() << "
          ---------" << endl;
457   for (predicate_list::iterator it = canl->begin(); it != canl->
          end(); it++){
458    mLogger << "CANDO: " << setw(20) << it->second->get_name()
459       << " = " << setw(4) << it->second->get_value()
460       << " rel = " << setw(4) << it->second->get_reliability()
              << endl;
461   }
462   mLogger << "----- End Cando -----\n" << endl;
463
464
465   proposed_action_list *pal = TheBrian->get_proposed_action_list
          ();
466   mLogger << "--------- Proposed Action - Length: " << pal->size
          () << " ---------" << endl;
467   for (proposed_action_list::iterator it = pal->begin(); it !=
          pal->end(); it++){
468    mLogger << "PAL: " << setw(20) << it->second->get_name() << "
           FROM " << setw(10) << it->second->get_behavior_name()
469       << " = " << setw(15) << it->second->get_label()
470       << " rel = " << setw(4) << it->second->
              get_membership_value() << endl;
471   }
472   mLogger << "----- End Proposed Action -----\n" << endl;
473
474
475   weight_want_list *wl = TheBrian->get_weight_want_list();
476   mLogger << "--------- Want - Length: " << wl->size() << "
          ---------" << endl;
477   for (weight_want_list::iterator it = wl->begin(); it != wl->
          end(); it++){
478    mLogger << "WANT: " << setw(20) << it->second->get_name()
479       << " = " << setw(4) << it->second->get_value() << endl;
480   }
481   mLogger << "----- End Want -----\n" << endl;
482
483
484   action_list *al = TheBrian->getFuzzy()->get_action_list();
```

```
485  mLogger << "--------- Action List - Length: " << al->size() <<
         " ---------" << endl;
486  for (action_list::iterator it = al->begin(); it != al->end();
         it++){
487   mLogger << "ACTION: " << setw(20) << it->second->get_name()
488        << " = " << setw(15) << it->second->get_label()
489        << " rel = " << setw(4) << it->second->
             get_membership_value();
490
491   if (it->second->get_membership_value() == 0)
492     mLogger << "\t" << "DISABLED";
493
494   mLogger << endl;
495  }
496  mLogger << "----- End Action -----\n" << endl;
497
498  command_list *cmdl = TheBrian->getFuzzy()->
         get_command_singleton_list();
499  mLogger << "--------- Commnad List - Length: " << cmdl->size()
         << " ---------" << endl;
500  for (command_list::iterator it = cmdl->begin(); it != cmdl->
         end(); it++){
501   mLogger << "COMMAND: " << setw(20) << it->second->get_label()
         << ' '
502        << setw(4) << it->second->get_set_point() << endl;
503  }
504  mLogger << "----- End Command -----\n" << endl;
505
506  mLogger << "***** Brian debug end
         ************************************************************\
         n" << endl;
507 }
```

B.5 WiimExpert

Listing B.8: src/WiimExpert.h

```
1  #ifndef WiimExpert_h
2  #define WiimExpert_h
3
4  #include "StringModuleMember.h"
5
6  #include <sys/time.h>
7  #include <cstdlib>
8  #include <cstdio>
9  #include <const.h>
10 #include <msg.h>
11 #include <wiim.h>
12 #include <wiiuse.h>
```

```
13 #include <Spykee.h>
14
15 typedef map<string,float> FloatDataListType;
16
17 class WiimExpert : public StringModuleMember{
18   public:
19       WiimExpert (ModuleDispatch* kernel, int period, Spykee*);
20       ~WiimExpert ();
21       void RunInit();
22       void RunClose();
23       void RunDuty();
24       void setRotDirection(int dir);
25
26     //int distances[SENSORS_NUMBER * 2];
27     bool m_bAuto;
28
29     inline void InsertData(string name, float value){
30       mCommandData[name] = value;
31     }
32
33   private:
34
35       Wiim* wiim;
36       int canDo();
37       void calculateposition();
38       long elapsed(int);
39       void settimer(int);
40       void AddBrianData(char*,float,float);
41       void printstatus();
42
43       struct timeval last_render[10];
44       float charged;
45     int Hit;
46     int gHome;
47     int restartflag;
48     int rotflag;
49     int fps;
50     int random_sign;
51     int goForward;
52
53       pos home;
54       pos goal;
55       pos center;
56       pos rob;
57       int angoloOdo;
58
59       ppoint position;
60
61     int robotpoint;
```

```
62      int humanpoint;
63      int pointrobotflag;
64
65      float maxx,maxy,minx,miny;
66
67      weightfloat ir_centerarray_x[IRCENTERFILTER];
68      weightfloat ir_centerarray_y[IRCENTERFILTER];
69      int distances_buffer[4];
70
71      float base_time;
72      FILE* fp;
73      FILE* f;
74
75      bool playing;
76      int clock;
77      bool cameraLightOn;
78
79      Spykee* pSpykee;
80      FloatDataListType mCommandData;
81
82    };
83  #endif
```

Listing B.9: src/WiimExpert.cpp

```
1  #include "WiimExpert.h"
2  #include <msg.h>
3
4  #include <iostream>
5  #include <math.h>
6  #include <string>
7  #include <stdlib.h>
8
9
10 #include "Configurator.h"
11
12 #define PI 3.14159265358979323846
13
14 using namespace std;
15
16 #define FPS_TIMER     0
17 #define CLOCK_TIMER   1
18 #define GAME_TIMER    5
19 #define GAMESTART_TIMER 6
20
21 // la parola extern non dichiara un nuovo identificatore,
22 // ma dice "La variabile intera a e' dichiarata da qualche
       altra parte,
23 // lascia solo lo spazio per risolvere il riferimento"
24 extern void wiimlex_memory(const char* src, WiimExpert* pWiim);
```

```
25 extern int MessageLineNumWiim;
26
27 extern struct wiimote_t* wm0;
28
29 WiimExpert::WiimExpert(ModuleDispatch* kernel,int period,
       Spykee* s) : StringModuleMember(kernel,period,"WiimExpert")
       {
30   pSpykee = s;
31   printf("\nWiimExpert built");
32   fflush(stdout);
33 };
34
35 WiimExpert::~WiimExpert()
36 {
37   fclose(fp);
38   fclose(f);
39   printf("\nWiimExpert destroyed");
40   fflush(stdout);
41 };
42
43 void WiimExpert::RunInit(){
44   AddMessageRequest(MSG_FROM_BRIAN, LOCAL);
45
46   int i=0;
47
48   m_bAuto = Configurator::initAuto;
49
50   charged=0;
51   Hit=0;
52   fps=0;
53   random_sign=2;
54
55   settimer(0);
56   settimer(1);
57   settimer(2);
58   settimer(3);
59
60   center.x=0;
61   center.y=0;
62
63   robotpoint=0;
64   humanpoint=0;
65   pointrobotflag=0;
66
67   srand(time(NULL));
68
69   for(i=0;i<IRCENTERFILTER;i++){
70     ir_centerarray_x[i].v=0;
71     ir_centerarray_x[i].w=0;
```

```
72      ir_centerarray_y[i].v=0;
73      ir_centerarray_y[i].w=0;
74    }
75
76    wiim = new Wiim(); //inizializza (connessione al wiimote)
77
78    struct timeval now;
79    gettimeofday(&now,NULL);
80      base_time=now.tv_sec;
81
82    playing = false;
83    clock = 0;
84    cameraLightOn = false;
85    settimer(GAMESTART_TIMER);
86
87    if (pSpykee != NULL)
88      pSpykee->turnOffCameraLight();
89
90      printf("\nWiimExpert initialized");
91      fflush(stdout);
92 };
93
94 /* **********************************
95
96     DoYourDuty()
97
98     ********************************** */
99 void WiimExpert::RunDuty(){
100   ppoint ir_center;
101   float ir_centerx,ir_centery;
102   int ir_centervalid;
103   weightfloat newdata;
104   weightfloat ir_newcenterx,ir_newcentery;
105   float ir_center_distance=0;
106   ostringstream oss;
107
108   // --- blocco d'emergenza ---
109   if (wiim->getPButtonTwo()){
110     NewMessage(MSG_TO_MOTION,WIIMOTE_DATA);
111     mCurrentMessage << "<C>TanSpeed 0.0</C>\n" << "<C>RotSpeed
          0.0</C>\n";
112     CloseCurrentMessage();
113     SendAllStringMessages();
114
115     int events = wiim->run();
116     if(wm0==NULL || !events)
117       return;
118   }
119
```

```
120   // --- End game --- //
121   if (wiim->getPButtonHome()){
122     cout << "Shutting down" << endl;
123
124     NewMessage(MSG_TO_MOTION, WIIMOTE_DATA);
125     mCurrentMessage << "<C>TanSpeed 0.0 </C>\n" << "<C>RotSpeed
          0.0</C>\n";
126     CloseCurrentMessage();
127     SendAllStringMessages();
128
129     NewMessage(MSG_TO_SONAR, LED_DATA);
130     mCurrentMessage << "Red " << 0 << " Green " << 0 << "\n";
131     CloseCurrentMessage();
132     SendAllStringMessages();
133     this->Shutdown(1);
134
135     return;
136   }
137
138   /*
139    * Passo in automatico (gioco)
140    */
141   if (!m_bAuto && wiim->getPButtonPlus()){
142     m_bAuto = true;
143     settimer(GAMESTART_TIMER);
144     settimer(CLOCK_TIMER);
145     settimer(GAME_TIMER);
146     settimer(FPS_TIMER);
147   }
148
149   /*
150    * Controllo manuale
151    */
152   if (m_bAuto && wiim->getPButtonMinus()){
153     m_bAuto = false;
154     playing = false;
155     cout << "Manual mode\nBrian deactivated" << endl;
156
157     // Reset brian
158     NewMessage(MSG_TO_SONAR, LED_DATA);
159     mCurrentMessage << "Red " << 4 << "\nGreen " << 8 << "
          Yellow " << 255 << "\n";
160     CloseCurrentMessage();
161
162     NewMessage(MSG_FROM_WIIMOTE, WIIMOTE_DATA);
163     AddBrianData("GameStatus", 0, 1);
164     CloseCurrentMessage();
165
166     SendAllStringMessages();
```

```
167   }
168
169   // --- Controllo manuale --- //
170   if (!m_bAuto){
171     NewMessage(MSG_FROM_WIIMOTE, WIIMOTE_DATA);
172     AddBrianData("GameStart", 0, 1);
173     CloseCurrentMessage();
174     SendAllStringMessages();
175
176     NewMessage(MSG_TO_MOTION, WIIMOTE_DATA);
177
178     if (wiim->getPButtonUp())
179       mCurrentMessage << "<C>TanSpeed 75.0</C>\n";
180     else if (wiim->getPButtonDown())
181       mCurrentMessage << "<C>TanSpeed -75.0</C>\n";
182     else if (wiim->getPButtonRight())
183       mCurrentMessage << "<C>RotSpeed 50.0</C>\n";
184     else if (wiim->getPButtonLeft())
185       mCurrentMessage << "<C>RotSpeed -50.0</C>\n";
186     else
187       mCurrentMessage << "<C>TanSpeed 0.0</C>\n<C>RotSpeed
              0.0</C>\n";
188
189     CloseCurrentMessage();
190     SendAllStringMessages();
191
192     settimer(GAMESTART_TIMER);
193
194     if((wm0==NULL) || (!wiim->run()))
195       return;
196
197     return;
198   }
199
200   if (elapsed(CLOCK_TIMER) > 1000000){    // Ogni secondo mando
        il clock
201     clock = (clock == 0 ? 1 : 0);
202     random_sign = rand() % 6;
203     settimer(CLOCK_TIMER);
204   }
205
206   if (m_bAuto && ((elapsed(GAMESTART_TIMER) > 5000000) && (
        playing == false))){
207     printf("\n--- *** START PLAY *** ---");
208
209     wiim->WeakRumble(1);
210     playing = true;
211     Hit = 0;
212     charged=0;
```

```
213
214     settimer(GAME_TIMER); // Timer per il tempo di durata del
            gioco
215
216     NewMessage(MSG_FROM_WIIMOTE, WIIMOTE_DATA);
217     AddBrianData("GameStart", 1, 1);
218     CloseCurrentMessage();
219     SendAllStringMessages();
220   }
221
222   int events = wiim->run();
223   if((wm0==NULL) || (!events))
224     return;
225
226   ir_center=wiim->getcenter();
227
228   if (ir_center.valid == 0){
229     newdata.w = 0;
230
231     ir_newcenterx=getFilteredData(newdata,ir_centerarray_x,
            IRCENTERFILTER,IRCENTERATT,NULL);
232     ir_newcentery=getFilteredData(newdata,ir_centerarray_y,
            IRCENTERFILTER,IRCENTERATT,NULL);
233
234     if(ir_newcenterx.w==0 || ir_newcentery.w==0)
235       ir_centervalid = 0;
236
237   } else {
238     newdata.w = 1;
239
240     newdata.v = ir_center.x;
241     ir_newcenterx=getFilteredData(newdata,ir_centerarray_x,
            IRCENTERFILTER,IRCENTERATT,NULL);
242
243     newdata.v=ir_center.y;
244     ir_newcentery=getFilteredData(newdata,ir_centerarray_y,
            IRCENTERFILTER,IRCENTERATT,NULL);
245
246     ir_centervalid = 1;
247   }
248
249   ir_centerx=ir_newcenterx.v;
250   ir_centery=ir_newcentery.v;
251
252   if(ir_centervalid==1)
253     ir_center_distance=sqrt(pow((ir_centerx-1024/2),2)+pow((
            ir_centery-768/2),2));
254
255   // --- messaggi al log ---
```

```
256   NewMessage(MSG_TO_LOG,WIIMOTE_DATA);
257   if(ir_centervalid==1)
258     mCurrentMessage << "valid 1 xcenter "<<ir_centerx<<"
          ycenter "<<ir_centery;
259   else
260     mCurrentMessage << "valid 0 xcenter 0 ycenter 0";
261
262   CloseCurrentMessage();
263   SendAllStringMessages();
264
265   if(elapsed(FPS_TIMER) > 1000000){
266     settimer(0);
267     printf("\nWiimExpert (%d) fps:%d     ", m_bAuto, fps);
268     printstatus();
269     fps = 0;
270   } else
271     fps++;
272
273   //carica
274   wiim->SetCharge(charged);
275
276   if (ir_centervalid){
277     if(ir_center_distance<IRCENTERLIMIT){
278
279       NewMessage(MSG_TO_SONAR, LED_DATA);
280       mCurrentMessage << "Red " << charged << " Green " << 8 <<
            "\n";
281       CloseCurrentMessage();
282       SendAllStringMessages();
283
284       if(charged < 4.5)
285         charged += 0.1;
286
287     } else {
288       NewMessage(MSG_TO_SONAR, LED_DATA);
289       mCurrentMessage << "Red " << charged << " Green " << 0 <<
            "\n";
290       CloseCurrentMessage();
291       SendAllStringMessages();
292
293       if(charged >=0)
294         charged -= 0.02;
295     }
296   }else{
297     NewMessage(MSG_TO_SONAR, LED_DATA);
298     mCurrentMessage << "Red " << charged << " Green " << 0 << "
          \n";
299     CloseCurrentMessage();
300     SendAllStringMessages();
```

```
301
302     if(charged >=0)
303       charged -= 0.02;
304   }
305
306   if ((charged > 4) && (ir_centervalid) && (ir_center_distance
          < IRCENTERLIMIT)){
307     if(wiim->getPButtonB()){
308       if(Hit == 0){
309         Hit = 1;
310         charged = 0;
311         cout << "\n--- *** COLPITO! *** ---" << endl;
312         humanpoint++;
313         playing = false;
314
315         if (pSpykee != NULL)
316           pSpykee->soundAllarm();
317
318         wiim->SetpointDiff(humanpoint-robotpoint);
319         wiim->BlinkRight(1);
320         wiim->StrongRumble(1);
321
322         NewMessage(MSG_FROM_WIIMOTE, WIIMOTE_DATA);
323         AddBrianData("GameStatus", 0, 1);
324         AddBrianData("GameStart", 0, 1);
325         CloseCurrentMessage();
326         SendAllStringMessages();
327
328         NewMessage(MSG_TO_MOTION, COMMAND_DATA);
329         mCurrentMessage << "<C>TanSpeed " << 0.0 << "</C>\n";
330         mCurrentMessage << "<C>RotSpeed " << 0.0 << "</C>\n";
331         CloseCurrentMessage();
332         SendAllStringMessages();
333
334         settimer(GAMESTART_TIMER);
335       }
336     }
337   }
338
339   NewMessage(MSG_TO_SONAR, LED_DATA);
340   mCurrentMessage << "Yellow " << humanpoint << "\n";
341   CloseCurrentMessage();
342   SendAllStringMessages();
343
344
345   if (humanpoint == 3){
346     cout << "Vittoria! Hai colpito il robot per 3 volte!\n"
347        << "Premi \'+\' sul Wiimote per iniziare una nuova
              partita" << endl;
```

```
348
349     playing = m_bAuto = false;
350     humanpoint = robotpoint = 0;
351
352     if (pSpykee)
353       pSpykee->soundAllarm();
354
355     wiim->BlinkRight(1);
356     wiim->StrongRumble(1);
357     return;
358   }
359
360   if(wiim->getPButtonB()){
361     if(charged > 2)
362       wiim->WeakRumble(1);
363
364     charged = 0;
365   }
366
367   if (playing == true){
368     NewMessage(MSG_FROM_WIIMOTE, WIIMOTE_DATA);
369     AddBrianData("Random",random_sign,1);
370     AddBrianData("Hit", Hit, 1);
371     AddBrianData("Clock", clock, 1);
372
373     //messaggio distanza e direzione dal centro IR (per brian)
374
375     if(ir_centervalid){
376       if(ir_centerx<1024/2)
377         AddBrianData("IRCenter",-1*ir_center_distance,1.0);
378       else
379         AddBrianData("IRCenter",+1*ir_center_distance,1.0);
380
381       AddBrianData("IRDist", wm0->ir.z, 1.0);
382     }
383     CloseCurrentMessage();
384     //invio tutti i msgs
385     SendAllStringMessages();
386   }
387 }
388
389
390 void WiimExpert::RunClose(){
391   printf("\nWiimExpert closed");
392   fflush(stdout);
393 };
394
395 void WiimExpert::AddBrianData(char* name,float value,float rel)
      {
```

```
396   float v;
397   float r;
398
399   v=value;
400   r=rel;
401   if(fabs(v)<0.001){ //evito che vengano mandati valori in
          notazione scientifica
402     v=0;
403   }
404   if(fabs(r)<0.001){
405     v=0;
406   }
407
408   mCurrentMessage<<"<D>"<<name<<" "<<v<<" "<<rel<<" </D>\n";
409   return;
410
411 }
412
413 long WiimExpert::elapsed(int i){
414   struct timeval now;
415   long elapsed_usec = 0;
416
417   gettimeofday(&now, NULL);
418
419   if (now.tv_usec > 1000000) {
420     now.tv_usec -= 1000000;
421     ++now.tv_sec;
422   }
423
424   if (now.tv_sec > last_render[i].tv_sec)
425     elapsed_usec = ((now.tv_sec - last_render[i].tv_sec) *
          1000000);
426
427   if (now.tv_usec > last_render[i].tv_usec)
428     elapsed_usec += now.tv_usec - last_render[i].tv_usec;
429   else
430     elapsed_usec -= last_render[i].tv_usec - now.tv_usec;
431
432
433   return elapsed_usec ;
434
435 }
436
437 void WiimExpert::settimer(int i){
438   gettimeofday(&(last_render[i]),NULL);
439 }
440
441 void::WiimExpert::printstatus(){
442   unsigned char *buffer = NULL;
```

```
443   float v = -1;
444
445   if (ReceiveLastMessage(MSG_FROM_BRIAN, buffer, false) > 0)
446     strtovalue((char*)buffer, "GameStatus ", &v);
447
448   printf("carica: %f\t",charged);
449
450   if(Hit)
451     printf("Colpito!\t");
452
453   printf(" hum:%d rob:%d\t",humanpoint,robotpoint);
454
455   switch ((int)v){
456     case 0:
457       printf("S:Search");
458       break;
459     case 1:
460       printf("S:Escape");
461       break;
462     case 2:
463       printf("S:Hide");
464       break;
465     default:
466       printf("E:?");
467   }
468
469   printf("\n");
470 }
471
472 void::WiimExpert::setRotDirection(int dir){
473   //rotDirection = dir;
474 }
```

B.6 SonarExpert

Listing B.10: src/WiimExpert.h

```
1  #ifndef WiimExpert_h
2  #define WiimExpert_h
3
4  #include "StringModuleMember.h"
5
6  #include <sys/time.h>
7  #include <cstdlib>
8  #include <cstdio>
9  #include <const.h>
10 #include <msg.h>
11 #include <wiim.h>
12 #include <wiiuse.h>
```

```
13 #include <Spykee.h>
14
15 typedef map<string,float> FloatDataListType;
16
17 class WiimExpert : public StringModuleMember{
18   public:
19       WiimExpert (ModuleDispatch* kernel, int period, Spykee*);
20       ~WiimExpert ();
21       void RunInit();
22       void RunClose();
23       void RunDuty();
24       void setRotDirection(int dir);
25
26     //int distances[SENSORS_NUMBER * 2];
27     bool m_bAuto;
28
29     inline void InsertData(string name, float value){
30       mCommandData[name] = value;
31     }
32
33   private:
34
35       Wiim* wiim;
36       int canDo();
37       void calculateposition();
38       long elapsed(int);
39       void settimer(int);
40       void AddBrianData(char*,float,float);
41       void printstatus();
42
43       struct timeval last_render[10];
44       float charged;
45     int Hit;
46     int gHome;
47     int restartflag;
48     int rotflag;
49     int fps;
50     int random_sign;
51     int goForward;
52
53       pos home;
54       pos goal;
55       pos center;
56       pos rob;
57       int angoloOdo;
58
59       ppoint position;
60
61     int robotpoint;
```

```
62      int humanpoint;
63      int pointrobotflag;
64
65      float maxx,maxy,minx,miny;
66
67      weightfloat ir_centerarray_x[IRCENTERFILTER];
68      weightfloat ir_centerarray_y[IRCENTERFILTER];
69      int distances_buffer[4];
70
71      float base_time;
72      FILE* fp;
73      FILE* f;
74
75      bool playing;
76      int clock;
77      bool cameraLightOn;
78
79      Spykee* pSpykee;
80      FloatDataListType mCommandData;
81
82    };
83  #endif
```

Listing B.11: src/WiimExpert.cpp

```
1  #include "WiimExpert.h"
2  #include <msg.h>
3
4  #include <iostream>
5  #include <math.h>
6  #include <string>
7  #include <stdlib.h>
8
9
10 #include "Configurator.h"
11
12 #define PI 3.14159265358979323846
13
14 using namespace std;
15
16 #define FPS_TIMER     0
17 #define CLOCK_TIMER   1
18 #define GAME_TIMER    5
19 #define GAMESTART_TIMER 6
20
21 // la parola extern non dichiara un nuovo identificatore,
22 // ma dice "La variabile intera a e' dichiarata da qualche
       altra parte,
23 // lascia solo lo spazio per risolvere il riferimento"
24 extern void wiimlex_memory(const char* src, WiimExpert* pWiim);
```

```
25 extern int MessageLineNumWiim;
26
27 extern struct wiimote_t* wm0;
28
29 WiimExpert::WiimExpert(ModuleDispatch* kernel,int period,
       Spykee* s) : StringModuleMember(kernel,period,"WiimExpert")
       {
30   pSpykee = s;
31   printf("\nWiimExpert built");
32   fflush(stdout);
33 };
34
35 WiimExpert::~WiimExpert()
36 {
37   fclose(fp);
38   fclose(f);
39   printf("\nWiimExpert destroyed");
40   fflush(stdout);
41 };
42
43 void WiimExpert::RunInit(){
44   AddMessageRequest(MSG_FROM_BRIAN, LOCAL);
45
46   int i=0;
47
48   m_bAuto = Configurator::initAuto;
49
50   charged=0;
51   Hit=0;
52   fps=0;
53   random_sign=2;
54
55   settimer(0);
56   settimer(1);
57   settimer(2);
58   settimer(3);
59
60   center.x=0;
61   center.y=0;
62
63   robotpoint=0;
64   humanpoint=0;
65   pointrobotflag=0;
66
67   srand(time(NULL));
68
69   for(i=0;i<IRCENTERFILTER;i++){
70     ir_centerarray_x[i].v=0;
71     ir_centerarray_x[i].w=0;
```

```
72      ir_centerarray_y[i].v=0;
73      ir_centerarray_y[i].w=0;
74    }
75
76    wiim = new Wiim(); //inizializza (connessione al wiimote)
77
78    struct timeval now;
79    gettimeofday(&now,NULL);
80      base_time=now.tv_sec;
81
82    playing = false;
83    clock = 0;
84    cameraLightOn = false;
85    settimer(GAMESTART_TIMER);
86
87    if (pSpykee != NULL)
88      pSpykee->turnOffCameraLight();
89
90      printf("\nWiimExpert initialized");
91      fflush(stdout);
92 };
93
94 /* **********************************
95
96     DoYourDuty()
97
98     ********************************** */
99 void WiimExpert::RunDuty(){
100   ppoint ir_center;
101   float ir_centerx,ir_centery;
102   int ir_centervalid;
103   weightfloat newdata;
104   weightfloat ir_newcenterx,ir_newcentery;
105   float ir_center_distance=0;
106   ostringstream oss;
107
108   // --- blocco d'emergenza ---
109   if (wiim->getPButtonTwo()){
110     NewMessage(MSG_TO_MOTION,WIIMOTE_DATA);
111     mCurrentMessage << "<C>TanSpeed 0.0</C>\n" << "<C>RotSpeed
          0.0</C>\n";
112     CloseCurrentMessage();
113     SendAllStringMessages();
114
115     int events = wiim->run();
116     if(wm0==NULL || !events)
117       return;
118   }
119
```

```
120   // --- End game --- //
121   if (wiim->getPButtonHome()){
122     cout << "Shutting down" << endl;
123
124     NewMessage(MSG_TO_MOTION, WIIMOTE_DATA);
125     mCurrentMessage << "<C>TanSpeed 0.0 </C>\n" << "<C>RotSpeed
          0.0</C>\n";
126     CloseCurrentMessage();
127     SendAllStringMessages();
128
129     NewMessage(MSG_TO_SONAR, LED_DATA);
130     mCurrentMessage << "Red " << 0 << " Green " << 0 << "\n";
131     CloseCurrentMessage();
132     SendAllStringMessages();
133     this->Shutdown(1);
134
135     return;
136   }
137
138   /*
139    * Passo in automatico (gioco)
140    */
141   if (!m_bAuto && wiim->getPButtonPlus()){
142     m_bAuto = true;
143     settimer(GAMESTART_TIMER);
144     settimer(CLOCK_TIMER);
145     settimer(GAME_TIMER);
146     settimer(FPS_TIMER);
147   }
148
149   /*
150    * Controllo manuale
151    */
152   if (m_bAuto && wiim->getPButtonMinus()){
153     m_bAuto = false;
154     playing = false;
155     cout << "Manual mode\nBrian deactivated" << endl;
156
157     // Reset brian
158     NewMessage(MSG_TO_SONAR, LED_DATA);
159     mCurrentMessage << "Red " << 4 << "\nGreen " << 8 << "
          Yellow " << 255 << "\n";
160     CloseCurrentMessage();
161
162     NewMessage(MSG_FROM_WIIMOTE, WIIMOTE_DATA);
163     AddBrianData("GameStatus", 0, 1);
164     CloseCurrentMessage();
165
166     SendAllStringMessages();
```

```
167   }
168
169   // --- Controllo manuale --- //
170   if (!m_bAuto){
171     NewMessage(MSG_FROM_WIIMOTE, WIIMOTE_DATA);
172     AddBrianData("GameStart", 0, 1);
173     CloseCurrentMessage();
174     SendAllStringMessages();
175
176     NewMessage(MSG_TO_MOTION, WIIMOTE_DATA);
177
178     if (wiim->getPButtonUp())
179       mCurrentMessage << "<C>TanSpeed 75.0</C>\n";
180     else if (wiim->getPButtonDown())
181       mCurrentMessage << "<C>TanSpeed -75.0</C>\n";
182     else if (wiim->getPButtonRight())
183       mCurrentMessage << "<C>RotSpeed 50.0</C>\n";
184     else if (wiim->getPButtonLeft())
185       mCurrentMessage << "<C>RotSpeed -50.0</C>\n";
186     else
187       mCurrentMessage << "<C>TanSpeed 0.0</C>\n<C>RotSpeed
              0.0</C>\n";
188
189     CloseCurrentMessage();
190     SendAllStringMessages();
191
192     settimer(GAMESTART_TIMER);
193
194     if((wm0==NULL) || (!wiim->run()))
195       return;
196
197     return;
198   }
199
200   if (elapsed(CLOCK_TIMER) > 1000000){    // Ogni secondo mando
        il clock
201     clock = (clock == 0 ? 1 : 0);
202     random_sign = rand() % 6;
203     settimer(CLOCK_TIMER);
204   }
205
206   if (m_bAuto && ((elapsed(GAMESTART_TIMER) > 5000000) && (
        playing == false))){
207     printf("\n--- *** START PLAY *** ---");
208
209     wiim->WeakRumble(1);
210     playing = true;
211     Hit = 0;
212     charged=0;
```

```
213
214     settimer(GAME_TIMER); // Timer per il tempo di durata del
            gioco
215
216     NewMessage(MSG_FROM_WIIMOTE, WIIMOTE_DATA);
217     AddBrianData("GameStart", 1, 1);
218     CloseCurrentMessage();
219     SendAllStringMessages();
220   }
221
222   int events = wiim->run();
223   if((wm0==NULL) || (!events))
224     return;
225
226   ir_center=wiim->getcenter();
227
228   if (ir_center.valid == 0){
229     newdata.w = 0;
230
231     ir_newcenterx=getFilteredData(newdata,ir_centerarray_x,
            IRCENTERFILTER,IRCENTERATT,NULL);
232     ir_newcentery=getFilteredData(newdata,ir_centerarray_y,
            IRCENTERFILTER,IRCENTERATT,NULL);
233
234     if(ir_newcenterx.w==0 || ir_newcentery.w==0)
235       ir_centervalid = 0;
236
237   } else {
238     newdata.w = 1;
239
240     newdata.v = ir_center.x;
241     ir_newcenterx=getFilteredData(newdata,ir_centerarray_x,
            IRCENTERFILTER,IRCENTERATT,NULL);
242
243     newdata.v=ir_center.y;
244     ir_newcentery=getFilteredData(newdata,ir_centerarray_y,
            IRCENTERFILTER,IRCENTERATT,NULL);
245
246     ir_centervalid = 1;
247   }
248
249   ir_centerx=ir_newcenterx.v;
250   ir_centery=ir_newcentery.v;
251
252   if(ir_centervalid==1)
253     ir_center_distance=sqrt(pow((ir_centerx-1024/2),2)+pow((
            ir_centery-768/2),2));
254
255   // --- messaggi al log ---
```

```
256   NewMessage(MSG_TO_LOG,WIIMOTE_DATA);
257   if(ir_centervalid==1)
258     mCurrentMessage << "valid 1 xcenter "<<ir_centerx<<"
          ycenter "<<ir_centery;
259   else
260     mCurrentMessage << "valid 0 xcenter 0 ycenter 0";
261
262   CloseCurrentMessage();
263   SendAllStringMessages();
264
265   if(elapsed(FPS_TIMER) > 1000000){
266     settimer(0);
267     printf("\nWiimExpert (%d) fps:%d     ", m_bAuto, fps);
268     printstatus();
269     fps = 0;
270   } else
271     fps++;
272
273   //carica
274   wiim->SetCharge(charged);
275
276   if (ir_centervalid){
277     if(ir_center_distance<IRCENTERLIMIT){
278
279       NewMessage(MSG_TO_SONAR, LED_DATA);
280       mCurrentMessage << "Red " << charged << " Green " << 8 <<
            "\n";
281       CloseCurrentMessage();
282       SendAllStringMessages();
283
284       if(charged < 4.5)
285         charged += 0.1;
286
287     } else {
288       NewMessage(MSG_TO_SONAR, LED_DATA);
289       mCurrentMessage << "Red " << charged << " Green " << 0 <<
            "\n";
290       CloseCurrentMessage();
291       SendAllStringMessages();
292
293       if(charged >=0)
294         charged -= 0.02;
295     }
296   }else{
297     NewMessage(MSG_TO_SONAR, LED_DATA);
298     mCurrentMessage << "Red " << charged << " Green " << 0 << "
          \n";
299     CloseCurrentMessage();
300     SendAllStringMessages();
```

```
301
302     if(charged >=0)
303       charged -= 0.02;
304   }
305
306   if ((charged > 4) && (ir_centervalid) && (ir_center_distance
          < IRCENTERLIMIT)){
307     if(wiim->getPButtonB()){
308       if(Hit == 0){
309         Hit = 1;
310         charged = 0;
311         cout << "\n--- *** COLPITO! *** ---" << endl;
312         humanpoint++;
313         playing = false;
314
315         if (pSpykee != NULL)
316           pSpykee->soundAllarm();
317
318         wiim->SetpointDiff(humanpoint-robotpoint);
319         wiim->BlinkRight(1);
320         wiim->StrongRumble(1);
321
322         NewMessage(MSG_FROM_WIIMOTE, WIIMOTE_DATA);
323         AddBrianData("GameStatus", 0, 1);
324         AddBrianData("GameStart", 0, 1);
325         CloseCurrentMessage();
326         SendAllStringMessages();
327
328         NewMessage(MSG_TO_MOTION, COMMAND_DATA);
329         mCurrentMessage << "<C>TanSpeed " << 0.0 << "</C>\n";
330         mCurrentMessage << "<C>RotSpeed " << 0.0 << "</C>\n";
331         CloseCurrentMessage();
332         SendAllStringMessages();
333
334         settimer(GAMESTART_TIMER);
335       }
336     }
337   }
338
339   NewMessage(MSG_TO_SONAR, LED_DATA);
340   mCurrentMessage << "Yellow " << humanpoint << "\n";
341   CloseCurrentMessage();
342   SendAllStringMessages();
343
344
345   if (humanpoint == 3){
346     cout << "Vittoria! Hai colpito il robot per 3 volte!\n"
347        << "Premi \'+\' sul Wiimote per iniziare una nuova
              partita" << endl;
```

```
348
349     playing = m_bAuto = false;
350     humanpoint = robotpoint = 0;
351
352     if (pSpykee)
353       pSpykee->soundAllarm();
354
355     wiim->BlinkRight(1);
356     wiim->StrongRumble(1);
357     return;
358   }
359
360   if(wiim->getPButtonB()){
361     if(charged > 2)
362       wiim->WeakRumble(1);
363
364     charged = 0;
365   }
366
367   if (playing == true){
368     NewMessage(MSG_FROM_WIIMOTE, WIIMOTE_DATA);
369     AddBrianData("Random",random_sign,1);
370     AddBrianData("Hit", Hit, 1);
371     AddBrianData("Clock", clock, 1);
372
373     //messaggio distanza e direzione dal centro IR (per brian)
374
375     if(ir_centervalid){
376       if(ir_centerx<1024/2)
377         AddBrianData("IRCenter",-1*ir_center_distance,1.0);
378       else
379         AddBrianData("IRCenter",+1*ir_center_distance,1.0);
380
381       AddBrianData("IRDist", wm0->ir.z, 1.0);
382     }
383     CloseCurrentMessage();
384     //invio tutti i msgs
385     SendAllStringMessages();
386   }
387 }
388
389
390 void WiimExpert::RunClose(){
391   printf("\nWiimExpert closed");
392   fflush(stdout);
393 };
394
395 void WiimExpert::AddBrianData(char* name,float value,float rel)
      {
```

```
396   float v;
397   float r;
398
399   v=value;
400   r=rel;
401   if(fabs(v)<0.001){ //evito che vengano mandati valori in
          notazione scientifica
402     v=0;
403   }
404   if(fabs(r)<0.001){
405     v=0;
406   }
407
408   mCurrentMessage<<"<D>"<<name<<" "<<v<<" "<<rel<<" </D>\n";
409   return;
410
411 }
412
413 long WiimExpert::elapsed(int i){
414   struct timeval now;
415   long elapsed_usec = 0;
416
417   gettimeofday(&now, NULL);
418
419   if (now.tv_usec > 1000000) {
420     now.tv_usec -= 1000000;
421     ++now.tv_sec;
422   }
423
424   if (now.tv_sec > last_render[i].tv_sec)
425     elapsed_usec = ((now.tv_sec - last_render[i].tv_sec) *
          1000000);
426
427   if (now.tv_usec > last_render[i].tv_usec)
428     elapsed_usec += now.tv_usec - last_render[i].tv_usec;
429   else
430     elapsed_usec -= last_render[i].tv_usec - now.tv_usec;
431
432
433   return elapsed_usec ;
434
435 }
436
437 void WiimExpert::settimer(int i){
438   gettimeofday(&(last_render[i]),NULL);
439 }
440
441 void::WiimExpert::printstatus(){
442   unsigned char *buffer = NULL;
```

```
443   float v = -1;
444
445   if (ReceiveLastMessage(MSG_FROM_BRIAN, buffer, false) > 0)
446     strtovalue((char*)buffer, "GameStatus ", &v);
447
448   printf("carica: %f\t",charged);
449
450   if(Hit)
451     printf("Colpito!\t");
452
453   printf(" hum:%d rob:%d\t",humanpoint,robotpoint);
454
455   switch ((int)v){
456     case 0:
457       printf("S:Search");
458       break;
459     case 1:
460       printf("S:Escape");
461       break;
462     case 2:
463       printf("S:Hide");
464       break;
465     default:
466       printf("E:?");
467   }
468
469   printf("\n");
470 }
471
472 void::WiimExpert::setRotDirection(int dir){
473   //rotDirection = dir;
474 }
```

B.7 MotorExpert

Listing B.12: src/MotorExpert.h

```
1  #ifndef MotorExpert_h
2  #define MotorExpert_h
3
4  #include "StringModuleMember.h"
5
6  #include <sys/time.h>
7  #include <stdlib.h>
8  #include <stdio.h>
9  #include <const.h>
10 #include <msg.h>
11 #include <stl.h>
12
```

```
13
14 #include <Spykee.h>
15
16 typedef map<string,float> FloatDataListType;
17
18 #define TAN_SPEED     "TanSpeed"
19 #define ROT_SPEED     "RotSpeed"
20 #define MOTOR_COMMAND   "MotorCommand"
21
22 #define SPYKEE_NORMAL   -1
23 #define SPYKEE_TURN_RIGHT  0
24 #define SPYKEE_TURN_LEFT   1
25 #define SPYKEE_TURN_BACK   2
26
27   class MotorExpert : public StringModuleMember{
28    public:
29       MotorExpert (ModuleDispatch* kernel, int period, Spykee*
           s);
30       ~MotorExpert ();
31
32       void RunInit();
33       void RunClose();
34       void RunDuty();
35       long elapsed(int);
36       void settimer(int);
37
38       void SendMotorMessage();
39
40     inline void InsertData(string name, float value){
41       mCommandData[name] = value;
42     }
43
44     private:
45     struct timeval last_render[10];
46       void AddOdometryData(string name, float value);
47       float old_Tan;
48       float old_Rot;
49     Spykee *pSpykee;
50     int m_state;
51     int dxSpeed,
52       sxSpeed;
53
54     FloatDataListType mCommandData;
55   };
56
57 #endif
```

Listing B.13: src/MotorExpert.cpp

```
1 #include "MotorExpert.h"
```

```
2 #include "msg.h"
3
4 #include <iostream>
5 using namespace std;
6
7 extern void motorlex_memory(const char* src, MotorExpert*
      pMotor);
8 extern int MessageLineNum;
9 int needodo=0;
10
11 // la parola extern non dichiara un nuovo identificatore,
12 // ma dice "La variabile intera a e' dichiarata da qualche
      altra parte,
13 // lascia solo lo spazio per risolvere il riferimento"
14
15 //extern void motorlex_memory(const char* src, MotorExpert*
      pMotor);
16 //extern int MessageLineNum;
17
18 MotorExpert::MotorExpert(ModuleDispatch* kernel,int period,
      Spykee *s) : StringModuleMember(kernel,period, "MotorExpert
      "){
19   pSpykee = s;
20   cout << "\nMotorExpert built" << endl;
21 };
22
23 MotorExpert::~MotorExpert(){
24   cout << "\nMotorExpert destroyed" << endl;
25 };
26
27 void MotorExpert::RunInit(){
28   AddMessageRequest( MSG_TO_MOTION, LOCAL);
29   cout << "\nMotorExpert initialized" << endl;
30 };
31
32 void MotorExpert::RunDuty(){
33   float Tan,Rot;
34   int flag=0;
35   unsigned char* buffer = NULL;
36
37   if(ReceiveLastMessage(MSG_TO_MOTION, buffer,false) > 0){
38     mCommandData.clear();
39
40     motorlex_memory((const char *)buffer,this);
41     //cout << buffer << endl;
42
43     dxSpeed = (int)(mCommandData[TAN_SPEED] - mCommandData[
          ROT_SPEED]);
```

```
44      sxSpeed = (int)(mCommandData[TAN_SPEED] + mCommandData[
            ROT_SPEED]);
45
46      if (dxSpeed > 100) dxSpeed = 100;
47      if (sxSpeed > 100) sxSpeed = 100;
48
49      if (dxSpeed < -100) dxSpeed = -100;
50      if (sxSpeed < -100) sxSpeed = -100;
51
52  /*    if (dxSpeed < 25 && dxSpeed > 0 ) dxSpeed = 25;
53      if (sxSpeed < 25 && sxSpeed > 0 ) sxSpeed = 25;
54
55      if (dxSpeed > 25 && dxSpeed < 0 ) dxSpeed = -25;
56      if (sxSpeed > -25 && sxSpeed < 0 ) sxSpeed = -25;
57  */
58    }
59
60    if ( pSpykee ){
61      pSpykee->move(sxSpeed, dxSpeed);
62      //cout << sxSpeed << " - " << dxSpeed << endl;
63    } else {
64      cout << sxSpeed << " - " << dxSpeed << endl;
65    }
66  };
67
68  void MotorExpert::RunClose(){
69    pSpykee->move(0,0);
70    cout << "\nMotorExpert closed" << endl;
71  };
72
73
74  void MotorExpert::SendMotorMessage(){
75  }
76
77  void MotorExpert::AddOdometryData(string name, float value){
78    mCurrentMessage << "<" << value << ">";
79  }
80
81
82  long MotorExpert::elapsed(int i){
83    struct timeval now;
84    long elapsed_usec = 0;
85
86    gettimeofday(&now, NULL);
87
88    if (now.tv_usec > 1000000) {
89      now.tv_usec -= 1000000;
90      ++now.tv_sec;
91    }
```

```
92
93   if (now.tv_sec > last_render[i].tv_sec){
94     elapsed_usec = ((now.tv_sec - last_render[i].tv_sec) *
         1000000);
95   }
96
97   if (now.tv_usec > last_render[i].tv_usec)
98     elapsed_usec += now.tv_usec - last_render[i].tv_usec;
99   else
100    elapsed_usec -= last_render[i].tv_usec - now.tv_usec;
101
102
103  return elapsed_usec ;
104
105 }
106
107 void MotorExpert::settimer(int i){
108   gettimeofday(&(last_render[i]),NULL);
109 }
```

Appendice C

Il manuale utente

C.1 Installazione

C.1.1 Dipendenze

Il software è stato sviluppato con computer dotati di Ubuntu 9.10/10.04 a 34 bit.
Per compilarlo sono necessari i seguenti pacchetti, ciascuno da installare con le proprie dipendenze:

Pacchetto	Versione	Utilizzo
bison	2.4.1	creazione parser
flex	2.5.35	creazione parser
g++	4.4.3-4ubuntu5	compilatore
libbost-program-options-dev	1.4.0-4ubuntu4	gestione opzioni
libbluetooth-dev	4.60-0ubuntu8	gestione bluetooth
mesa-common-dev	7.7.1-1ubuntu3	finestra grafica
libsdl1.2-dev	1.2.14-4ubuntu1.1	finestra grafica
freeglut3-dev	2.6.0-0ubutnu2	finstra grafica
OpenCV	2.1	eleborazione immagini
Wiiuse	0.12	gestione WiiMote

Tutti i pacchetti, ad eccezione delle librerie OpenCV e Wiiuse, si trovano nei repository ufficiali di Ubuntu. Per quanto riguarda le OpenCV vanno compilate e installate manualmente dopo averle scaricate dal sito dello sviluppatore.
La versione 0.12 delle librerie Wiiuse, da comilare manualmente, è invece inculsa nel software; per compilarla è necessario dare da terminale i comandi: *make wiiuse* seguito da *sudo make install*.

C.1.2 Compilazione

Per compilare il software si può utilizzare lo script *run.sh* presente nella cartella princiaple e nella cartella *spykee*. In alternativa si può utilizzare il comando *make ROBOT=spykee* per compilare e *make clean* per cancellare i file intermeti ed ottenere i sorgenti puliti prima di una ricompilazione.
Una volta compilato l'eseguibile si trova nella cartella robowii.

C.2 Configurazione

Il software è stato progettato in modo che la maggior parte dei parametri di gioco sia impostabile senza dover ricompilare il codice, ma utilizzando un apposito file di configurazione.
Il file di configurazione spykee.cfg si trova in spykee/config e viene letto dalla classe *Configurator*. Per modificare questa classe o derivarne una nuova fare riferimento al codice sorgente.
I parametri configurabili sono:

Nome parametro	Descrizione
spykee-user	Nome utente per collegarsi a Spykee
spykee-password	Password per collegarsi a Spykee
auto	Falso avvio gioco in manuale. Vero in automatico.
sim	Usata per il debug, simulazione a terminale
escape-timeout	Tempo massimo di fuga
sonar-delay	Lunghezza del buffer circolare dei sonar, in μs
sonar-front	Indice del sonar anteriore
soanr-back	Indice del sonar posteriore
sonar-right	Indice del sonar destro
sonar-left	Indice del sonar sinistro
detection-threshold	Fotogrammi per considerare il giocatre in vista
brian-ctof	File di configurazione del fuzzyficatore
brian-ctof-shapes	Forme fuzzy utilizzate dal fuzzyficatore
brian-cando	File di configurazione di CANDO
brian-want	File di configurazione di WANT
brian-behviour	File di definizione delle regole
brian-predicate	Definizione dei predicati
brian-predicate-actions	Predicati utilizzati tra livelli di uno stesso ciclo
brian-ftoc	Definizione variabili numeriche defuzzyficate
brian-ftoc-shaps	Forme utilizzate nella defuzzyficazione

C.3 Come giocare

Prima di iniziare a giocare è importante che il giocatore indossi sul petto il marker per permettere a Spykee di individuarlo.
Per iniziare a giocare bisogna prima di tutto accendere Spykee e, tramite pc, stabilre la connessione wireless. A questo punto si può far prarire il software e se la connessione avveiene correttamente Spykee dovrebbe suonare. Una volta fatto questo vanno premuti i tasti 1 e 2 sul Wiimote per connetterlo al pc e dare il via al gioco.
Per totalizzare un punto bisogna puntare Spykee alle spalle e quando il colpo è carico premere il tasto B del Wiimote. Se si totalizza il punto, il Wiimote vibra e il punto verrà sommato a quelli indicati dai led gialli di Spykee e da quelli blu sul Wiimote, quindi il gioco si ferma per qualche secondo e poi riprende (la ripresa è indicata da una vibrazione del Wiimote).
Quando Spykee vede il giocatore o rileva di essere puntato cercherà di nascondersi.
Il gioco termina dopo aver totalizzato tre punti.

Tasti:

Tasto	Funzione
A	Stop di emergenza
B	Fuoco
1 & 2	Connessione al pc
-	Controllo manuale
+	Controllo automatico (gioco)