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

Un robot autonomo che può giocare a nascondino

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

Relatore: Prof. Andrea Bonarini

Tesi di Laurea di:
Davide Gadioli, matricola 701948

Anno Accademico 2009-2010

Sommario

L'espansione straordinaria dell'aspetto ludico dell'informatica ha portato da sempre grandisssime novità, basti pensare all'idea di fornire i computer della capacità di emettere suoni. Questa specializzazione dell'informatica ha spronato la costruzione di hardware sempre più performante e software più efficiente.

Lo scopo di questa tesi è quello di realizzare un gioco in cui un robot autonomo riesca a sfidare il giocatore in una partita a nascondino, avendo a disposizione solo pochi dati per modellizzare il contesto in cui si trova.

La realizzazione di questo gioco è iniziata focalizzandoci sulla parte di visone del robot, per riuscire a comprenderne le potenzialità e i limiti. Successivamente abbiamo creato l'archichettura software in grado di gestire tutte le componenti hardware e le meccaniche del gioco. L'ultimo passo è stato quello di fornire Spykee l'intelligenza per essere un degno avversario.

Ringraziamenti

Ringrazio innanzitutto miei genitori per avermi supportato in questi anni e per avermi fornito la possibilità di raggiungere questo traguardo. A volte anche letteralmente considerando la mia predisposizione a ignorare gli avvertimenti della sveglia.

Ringrazio specialmente il mio relatore per avermi fornito preziosi consigli durante lo sviluppo di questa tesi. Ringrazio tutte le persone incontrate durante la mia permanenza in Airlab sempre disponibili per chiarimenti, a condividere il loro sapere o per semplice compagnia. Ringrazio specialmente tutti i ragazzi che hanno lavorato con Spykee prima di me, perché senza il loro lavoro non avrei potuto fare quello che ho fatto.

Ripensando allo scrittore che ha riempito ossessivamente il suo libro con "All work and no play makes Jack a dull boy" prima di impazzire, mi sento di ringraziare anche tutti i miei amici, compresi i folli che mi hanno accompagnato e spronato in questi anni di studio. Ringrazio i miei compagni di avventure con cui ho passato piacevoli serate a esplorare mondi di misteri e magia. Ringrazio inoltre i miei commilitoni di Azeroth impegnati a combattere contro la legione infuocata dimostrando contemporaneamente la supremazia dell'orda. Mantenendo una vecchia promessa ringrazio esplicitamente anche una persona che é stata recentemente a LA.

Indice

Capitolo 1

Introduzione

"And now you dare enter MY realm? You are not prepared..."

Illidan Stormrage

Per noi esseri umani giocare ci risulta naturale, tanto da catalogarla come attività ricreativa o di svago. Da un punto di vista software, come dal punto di vista di un robot, un gioco è invece qualcosa di molto complesso. Spesso perchè ci obbliga a creare una simulazione sempre più accurata della realtà, sia a livello grafico, sia a livello di modellizzazione. In qualche occasione occorre prendere decisioni sulla base di informazioni che descrivono in modo incompleto il contesto in cui ci si trova. L'imprevedibilità e la creatività del giocatore spesso portano al limite il modello che abbiamo creato e utilizzato.

Lo scopo di questa tesi è quello di fornire la capacità di giocare a nascondino a un robot autonomo sulla base di scarse informazioni sul mondo esterno. Abbiamo scelto di utilizzare un robot commerciale a basso costo come antagonista del giocatore umano, modificandolo il meno possibile. Il nostro obiettivo é trasformare questo robot in un degno avversario per un essere umano.

Sicuramente nascondino è stato un gioco con cui ci siamo divertiti da piccoli. Per un bambino può sembrare banale giocarci, perchè lui vede direttamente alberi, scatoloni, muri, insomma un bambino vede immediatamente dei luoghi in cui è possibile nascondersi. Per un robot invece individuare un nascondiglio é tutt'altro che banale, sopratutto considerando che un robot vede solo immagini, ovvero sequenze ordinate di puntini colorati, senza nessuna connessione logica. In aggiunta un bambino sa esattamente dove si trova ed è consapevole delle distanze a cui si trovano tutti gli oggetti intorno a lui; il nostro robot invece si deve basare su ciò che vede da una camera

e le misure di quattro sonar che dispone per rilevare eventuali oggetti nelle sue vicinanze.

1.1 Descrizione del lavoro

Il gioco con cui sfidiamo il nostro avversario é una variante di nascondino. Anche un gioco come nascondino, in apperenza molto semplice, richiede un'attenta analisi dei concetti che racchiude e richiede anche l'elaborazione di una linea d'azione che conduca Spykee alla vittoria.

Come accennato precedentemente e poi spiegato nel dettaglio nei capitoli successivi, l'azione fondamentale che Spykee deve compiere è quella di nascondersi. Per iniziare ad analizzare il problema contenuto in questa tesi occorre quindi stabilire cos'è un nascondiglio e nel caso se ne individuiamo più di uno come scegliere in quale nasconderci, ma sopratutto come nasconderci. Le risposte a tutte queste domande sono racchiuse nei capitoli che seguono.

In questa variante di nascondino il giocatore deve riuscire a sparare a Spykee per poter vincere la partita. Il primo passo che abbiamo effettuato è quello di analizzare con cura le richieste di questa tesi descrivendo meglio il contesto ed evidenziando eventuali problemi nascosti. Inoltre abbiamo elaborato la strategia che Spykee deve utilizzare per massimizzare le probabilità di vittoria.

Il passo successivo che abbiamo compiuto è quello di verificare i limiti e le potenzialità della visione. Il motivo di questa decisione è che il sensore in grado di fornire abbastanza informazioni per poter consentire al nostro robot di nasconderci è la camera di Spykee. Per questo motivo abbiamo esplorato le opzioni che ci forniscono gli algoritmi di visione, utilizzando le librerie OpenCv.

Una volta che abbiamo risolto il problema di visione, riuscendo ad identificare il miglior nascondiglio in cui ci andremo a nascondere, ci siamo concentrati sullo sviluppo dello scheletro della nostra applicazione. Abbiamo realizzato l'architettura multi-agente richiesta dalla nostra applicazione e abbiamo implementato le meccaniche del gioco.

Una volta completato il software necessario per poter utilizzare tutti i componenti hardware e software utilizzati in questa tesi, abbiamo iniziato a fornire Spykee dell'intelligenza necessaria per eseguire la strategia che abbiamo elaborato.

Durante quest'ultima fase abbiamo realizzato che alcune parti dell'analisi erano lacunose o davano per scontato alcuni fatti che non lo erano per nulla; oppure semplicemente non abbiamo ottenuto i dati che speravamo.

Raggiunto questo stadio, il nostro software ha subito un'evoluzione a spirale: si riformula l'analisi tenendo conto dei risultati che abbiamo ottenuto, si modifica la soluzione in relazione alla modifica dell'analisi e si procede nuovamente a testare l'applicazione. L'iterazione di questo procedimento ha condotto alla nascita dell'applicazione che abbiamo sviluppato.

1.2 Struttura della tesi

Questo elaborato é stato scritto seguendo la seguente struttura logica:

- Nel secondo capitolo si illustra lo stato dell'arte, ovvero si descrivono le tecnologie che abbiamo utilizzato come punto di partenza per la realizzazione della nostra applicazione.
- Nel terzo capitolo abbiamo discusso l'analisi del problema evidenziandone tutti i suoi aspetti.
- Nel quarto capitolo presentiamo la soluzione logica che abbiamo utilizzato per risolvere i problemi emersi durante l'analisi.
- Nel quinto capito illustriamo nel dettaglio l'architettura che abbiamo utilizzato specificando la funzione che svolge ogni agente coinvolto.
- Nel sesto capitolo mostriamo il progetto dal punto di vista sperimentale, descrivendo le attività svolte e i risultati ottenuti. Inoltre vengono illustrati gli sviluppi futuri.
- Nell'appendice A sono presenti le regole fuzzy più importanti che abbiamo utilizzato.
- Nell'appendice B è presente il manuale utente per compilare e utilizzare la nostra applicazione.

Capitolo 2

Stato dell'arte

"Mai fidarsi di un computer che non è possibile gettare dalla finestra."

Steve Wozniak

In questo capitolo illustreremo tutte le tecnologie utilizzate come punto di partenza per la realizzazione di questa tesi. In particolare descriveremo tutte le componenti software e hardware utilizzate.

2.1 Componenti Hardware

In questa sezione vedremo quali sono i componenti hardware che compongono questa tesi. In particolare verranno descritti Spykee, il Wiimote, i sonar e gli Xbee. Inoltre verrà descritto il loro utilizzo in altre applicazioni.

2.1.1 Spykee

L'idea che portò all'origine di Spykee è quella di fornire a tutte le persone, indipendentemente dalla loro età, la possibilità di costruire il loro robot spia personale. Con questa idea in mente la Meccano ha costruito questo robot che può essere assemblato principalmente in tre differenti allestimenti. Per questo progetto è stato scelto l'allestimento a sembianze umanoidi come illustrato nella Figura 2.1.

Analizzando le caratteristiche tecniche di Spykee (disponibili per esteso sul sito ufficiale [6]) emergono delle funzionalità molto interessanti, in particolare:

Movimento: Spykee utilizza due cingoli gommati per muoversi, che gli permettono di raggiungere una velocità massima di circa 3.5 km/h.

Figura 2.1: Allestimento di Spykee utilizzato

Visione: Questa funzione è stata affidata ad una camera QVGA CMOS in grado di ottenere 15 immagini al secondo in locale in condizioni ottimali. Le immagini ottenute da remoto possono essere da 4 a 15 in condizioni ottimali. Spykee prima di inviare le immagini le comprime utilizzando lo standard JFIF.

Comunicazione: Il controllo remoto del robot avviene instaurando un canale WiFi utilizzando il protocollo 802.11 b/g.

Per utilizzare Spykee in questo progetto sono state apportate delle modifiche. In particolare è stata aggiunta una fascia sonar per individuare eventuali oggetti intorno a Spykee, alcuni led per indicare con maggior chiarezza la situazione di gioco e due led ad infrarossi sulla parte posteriore del robot, con lo scopo di permettere al Wiimote di individuare Spykee. Per poter interagire con Spykee, la Meccano mette a disposizione una sua applicazione proprietaria, in grado si essere eseguita su una piattaforma Windows o Mac.

2.1.2 Wiimote

Il Wiimote è il controller predefinito utilizzato dalla console Wii. Simile per aspetto ad un telecomando, come illustrato nella Figura 2.2, racchiude tuttavia molte caratteristiche interessanti. A differenza dei controller di altre console, il Wiimote interagisce con l'utente tramite il movimento e il puntamento, rendendo l'esperienza di utilizzo molto più dinamica e divertente. In particolare le sue caratteristiche di rilievo sono le seguenti:

Figura 2.2: Profili del Wiimote

Puntamento: Il Wiimote incorpora una telecamera ad infrarossi 480p (PAL/NTSC) o 576i (PAL/SECAM). L'elaborazione dell'immagine è effettuata direttamente dal controller utilizzando una gpu ATI Hollywood in modo da estrapolare da ogni immagine le coordinate di fino a quattro sorgenti luminose. Le coordinate dei quattro punti individuati vengono utilizzati dalla Wii per stabilire la posizione del Wiimote rispetto alla fascia di led normalmente posta sopra o sotto la televisione. In questo progetto vengono utilizzati per stabilire se l'utente stia puntando Spykee.

Movimento: Il controller della Wii integra al suo interno anche degli accelerometri in grado di riconoscere il movimento che l'utente sta eseguendo con il controller. Come si può osservare dalla Figura 2.3, il Wiimote è in grado di riconoscere le accelerazioni su tutti i tre assi.

Comunicazione: Il Wiimote per comunicare con la console non utilizza fili ma sfrutta il protocollo Bluetooth garantendo quindi la totale libertà di movimento. Questa caratteristica è molto utile in un gioco dinamico come quello che abbiamo sviluppato.

Vibrazione: Per migliorare la sensazione di feedback in alcuni frangenti, la Nintendo ha dotato il Wiimote della capacità di vibrare.

Grazie a queste sue caratteristiche che lo rendono unico, è stato utilizzato in molti altri progetti interessanti, come ad esempio:

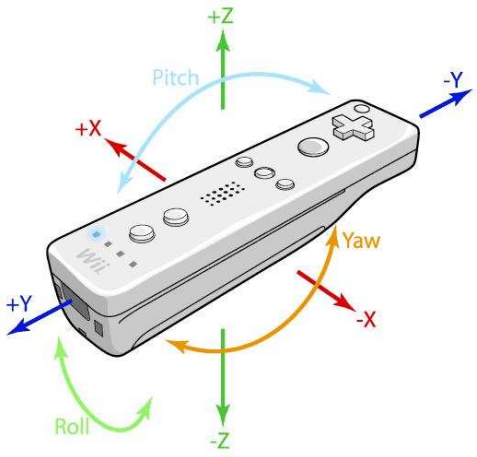


Figura 2.3: Rollio, beccheggio e imbardata riconusciuti dal Wiimote

Tracking Your Fingers with the Wiimote

Johnny Chung Lee, un ricercatore presso la Carnegie Mellon University, attraverso la camera IR del Wiimote riesce a riconoscere e tracciare i movimenti degli oggetti, come per esempio le dita della mano. In questo modo è possibile controllare un computer con le mani, come si vede nel film Minority report.

Guardando la presentazione del suo lavoro si vede che per far funzionare il suo progetto, ha costruito una sorta di lampada ad infrarossi unendo decine di led ad infrarossi in una basetta con un foro nel mezzo per inserire la camera del wiimote. Questo dispositivo deve essere posizionato sopra al monitor del computer da controllare. Per il corretto funzionamento della sua applicazione occorre applicare all'estremità degli indici della mano un poco di materiale riflettente. Ultimata anche quest'ultima fase, è possibile utilizzare le proprie mani per interagire con il computer servendosi della sua applicazione.

Come si vede dalla Figura 2.4, ha reso disponibile anche una demo in cui mostra le potenzialità della sua applicazione.

2.1.3 Sonar

Nasce per esigenze di localizzazioni sottomarine, il sonar costituisce un metodo semplice per ottenere la misura della distanza tra un oggetto e il trasmettitore sonar. L'idea dietro a questo strumento è quella di calcolare il tempo che impiega un segnale per colpire un oggetto e rimbalzare indietro fino al ricevitore. Il sonar infatti, emette un'onda sonora che si propaga attraver-

Figura 2.4: Applicazione dimostrativa realizzata da Johnny Chung Lee

so l'atmosfera, quando colpisce un oggetto, parte dell'onda viene riflessa generando un eco che si propaga verso il sonar. Quando il ricevitore rileva l'eco, calcola la misura della distanza dell'oggetto in base al tempo che l'onda ci ha impiegato a ritornare al ricevitore.

La scelta dei sonar in questo progetto è caduta sugli LV-MaxSonar-EZ2. Tale scelta è legata al fatto che questi sensori riescono a rilevare oggetti fino a 6,45 metri con un angolo di 30 gradi e hanno un ridottissimo consumo di potenza.

Per poter essere usati in questa tesi ci siamo serviti di una scheda di gestione dei sonar sviluppata internamente all'Airlab.

2.1.4 Xbee

La Digi ha sviluppato questi moduli per instaurare una connessione senza fili tra di loro. GLi Xbee infatti utilizzano il protocollo IEEE 802.15.4 per ottenere una rete veloce punto-punto oppure punto-multipunto.

L'idea alla base della realizzazione di questi moduli è quella di fornire un dispositivo di dimensioni molto contenute, che consumi poca energia e in grado di creare un collegamento senza fili fra i due moduli. Se opportunamente configurati, riescono a creare una rete in cui più Xbee riescono a comunicare fra loro.

Di seguito elenchiamo le caratteristiche tecniche principali che ci hanno spinto al loro utilizzo, per un elenco più completo si faccia riferimento al sito ufficiale [5].

- Potenza segnale emesso: 1mW (+0dBm)
- Distanza massima in ambiente chiuso: 30m
- Distanza massima in ambiente aperto: 90m
- Velocità di trasferimento massima: 250 Kbps

♦ Banda utilizzata: 2.4000 - 2.4835 GHz

2.2 Componenti Software

Le componenti software utilizzate comprendono l'architettura MRT, sviluppata all'interno dell'Airlab nel contesto della RoboCup; la libreria OpenCv dell'Intel per l'elaborazione delle immagini, la libreria Libxml2 realizzata da GNOME per l'analisi di messaggi xml, la libreria Wiiuse per poter comunicare con il Wiimote e Mr. Brian.

2.2.1 MRT

MRT, Modular Robotic Toolkit, è un'architettura software multiagente composta da moduli che permettono di risolvere molte problematiche tipiche della robotica mobile. Mrt è essenzialmente una raccolta di moduli utili per sviluppare applicazioni. Fra i moduli che contiene Mrt, si distinguono due grandi gruppi: il primo gruppo fornisce uno scheletro su cui appoggiare tutto il software, il secondo uno strumento in grado di far prendere al robot decisioni autonome. Il modulo principale che si occupa di fornire lo scheletro dell'applicazione si chiama DCDT, mentre il modulo principale del secondo gruppo si chiama Mr.Brian.

A volte per risolvere un problema complesso occorre scomporlo in sottoproblemi, tipicamente indipendenti fra loro, e risolvere ciascun sottoproblema. Per questo motivo l'idea a monte della realizzazione di DCDT è quella di fornire uno strumento per risolvere un compito complesso attraverso la risoluzione dei sottoproblemi. In particolare la responsabilità della risoluzione di ogni sottoproblema è affidata ad un agente (o esperto). Un tipico esempio potrebbe essere che sia presente un agente che si occupa della parte di visione, un altro esperto che si occupa di prendere le decisioni su cosa debba fare il robot (utilizzando Mr. Brian) e un ultimo esperto che si occupi di muovere il robot.

In un qualsiasi sistema di questo tipo occorre che i diversi agenti comunichino fra di loro per potersi coordinare e risolvere adeguatamente il problema complesso iniziale. MRT utilizza un sistema di tipo publish/subscriber per assolvere a questa funzionalità. In questo tipo di comunicazione i mittenti e i destinatari di messaggi comunicano solo attraverso un tramite chiamato dispatcher (o broker). Ogni messaggio ha un tipo, che ne identifica la natura. Quando un mittente pubblica un messaggio nel dispatcher, tutti i destinatari interessati al tipo del messaggio inviato ricevono automaticamente tale messaggio. Questo schema implica che i mittenti non sanno

quanti e quali destinatari esistono e viceversa, questo può contribuire alla scalabilità del sistema.

In MRT il ruolo di dispatcher lo assume l'Agorà, mentre gli esperti hanno il ruolo di mittenti e/o destinatari. Ogni esperto ha a disposizione un'interfaccia verso l'Agorà per iscriversi ad un determinato tipo di messaggio, per inviare messaggi e per ricevere messaggi del tipo richiesto. Per ricevere un messaggio di un certo tipo occorre che l'esperto ne abbia precedentemente fatto richiesta all'Agorà.

La dislocazione degli esperti o il canale di collegamento tra esperto e Agorà viene gestito dall'Agorà stessa in maniera del tutto trasparente rispetto all'esperto. La comunicazione tra più Agorà è possibile. Per questo motivo MRT costituisce una base molto solida da utilizzare per sistemi multiagente.

Come già accennato, Mr. Brian è il modulo che a fronte della situazione analizzata, determina il comportamento del robot. Questo modulo infatti si occupa di eseguire determinati comportamenti a seconda degli ingressi che ha ricevuto. La particolarità di questo modulo è che ragiona in logica fuzzy, garantendo quindi una maggiore fluidità nei suoi comportamenti. Per una spiegazione dettagliata su come abbiamo utilizzato questo modulo consultare il Capitolo 4.

MRT è stata sviluppata internamente all'Airlab nel contesto della RoboCup. Per un manuale più dettagliato sul funzionamento di DCDT e per esempi di codice, leggere [2]. Per maggiori informazioni su Mrt in generale e Mr. Brian consultare [3]

RoboCup

Un esempio in cui è stato utilizzato MRT è nella RoboCup. La RoboCup è un'iniziativa di ricerca internazionale che tenta di favorire lo sviluppo di IA e robot intelligenti dando un problema standard che tocca un vasto assortimento di tecnologie da esaminare e studiare.

Per questo motivo la RoboCup ha deciso di usare il gioco del calcio come principale dominio di ricerca. Questo perché una squadra di robot che volesse giocare a calcio dovrebbe includere una grande varietà di tecnologie: progettazione di agenti autonomi, collaborazione multiagente, acquisizione di una strategia, ragionamento in tempo reale, robotica, unificare i dati ricevuti dai sensori per comprendere meglio il mondo esterno.

Per questi motivi il Politecnico di Milano ha creato la propria squadra di robot calciatori, partecipando anche ad alcuni campionati della RoboCup. In questi robot è stato usato MRT come architettura dei robot, configuran-

dola in modo da essere in collegamento oltre tra gli esperti locali di un robot anche con le Agorà degli altri robot.

2.2.2 OpenCv

Le librerie OpenCv (Open Source Computer Vision) sono state sviluppate dall'Intel con lo scopo di fornire le principali funzioni per l'elaborazione di immagini. In letteratura si trovano molti algoritmi riguardanti il campo dell'elaborazione delle immagini. Considerando che l'immagine stessa è solamente una matrice di interi, è naturale domandarsi perché utilizzare questa libreria al posto di un'altra o al posto di implementare da soli gli algoritmi che ci interessano. La risposta a questa domanda risiede nel fatto che l'Intel ha scritto questa libreria pensando all'ottimizzazione e alle prestazioni, oltre ovviamente alla validità degli algoritmi. In queste librerie l'Intel utilizza un loro sistema per memorizzare le immagini che abbatte il tempo di referenziazione e deferenziazione dei dati quando si accede alle immagini, inoltre ogni algoritmo implementato sfrutta la capacità di parallelizzazione delle moderne cpu.

Le OpenCv si possono dividere in tre macroparti: la prima, che è il cuore della libreria, racchiude il necessario per lavorare con le immagini (CxCore), la seconda comprende delle funzioni per visualizzare a schermo l'immagine in una finestra con la possibilità di aggiungere alcuni controlli grafici elementari (HighGui) e la terza si occupa di riconoscere un specifico pattern nell'immagine come per esempio le facce (HaarTraining).

Le CxCore offrono una serie di strutture dati e funzioni d'appoggio oltre a quelle di manipolazione classiche. Questa parte della libreria comprende infatti molte funzioni, dai metodi per la calibrazione della camera, al fornire strumenti per estrarre delle caratteristiche delle immagini per poterle tracciare in una seconda immagine individuando il flusso ottico, piuttosto che offrire la possibilità di segmentare le immagini per ottenere contorni su cui possono essere fatte altre elaborazioni, come per esempio l'approssimazione a un poligono, il calcolo dell'area e del perimetro. Le CxCore inoltre comprendono ovviamente le classiche funzioni per elaborare le immagini come per esempio scrivere un testo o disegnare una riga, un rettangolo o un'altra figura geometrica.

Una guida completa circa le potenzialità della libreria e il loro uso è [1].

2.2.3 libXml2

Il formato xml nasce alla fine degli anni 80 nell'ambito della SGML Activity del W3C, ma solo nel 1998 ne vennero definite le specifiche. L'xml è un

linguaggio di demarcazione, che permette di definire dati ben strutturati. All'origine era limitato al contesto web, in seguito si è notato che questo linguaggio era ben più generale e in grado di adattarsi in qualsiasi contesto. Infatti l'xml fornisce delle regole riguardo al modo di descrivere le informazioni e , se lo si ritiene necessario, validarne i contenuti. Per maggiori informazioni riguardo l'xml consultare il libro [4].

Il vantaggio principale del fatto di avere regole ben definite su come descrivere le informazioni è che ha reso possibile realizzare degli analizzatori di documenti xml generali senza che debbano saperne il contesto. Libxml2 è infatti una libreria in grado di analizzare, esplorare e ottenere le informazioni contenute in un file xml. Inoltre questa libreria offre anche una funzione di validazione del documento da analizzare, ovvero uno strumento che stabilisce se il messaggio che si sta analizzando rispetta la grammatica da noi stabilita. Quest'ultima funzione richiede in aggiunta anche il Document Type Definition (DTD), ovvero un file che descrive la grammatica che il documento da analizzare deve rispettare. Xml definisce le regole con cui scrivere anche quest'ultimo documento.

Nel nostro progetto le informazioni che si inviano gli esperti sono scritte rispettando il formato xml. Siccome le informazioni inviate sono prodotte dagli esperti senza utilizzare alcun ingresso direttamente immesso dall'utente, si è preferito evitare di convalidare i messaggi ricevuti agli esperti per evitare l'inevitabile spreco di risorse.

Esistono molte librerie che svolgono una funzione simile, ma il team di GNOME che ha realizzato questa libreria è riuscito a creare un valido software open source, utilizzato anche in ambito commerciale. Questo significa che oltre a funzionare correttamente, lo fa in maniera efficiente. Questa libreria può essere disponibile anche al di fuori dell'ambiente GNOME, ma sopratutto ha il pregio di essere thread safe. All'interno di un ambiente come quello di MRT, in cui sono presenti diversi thread (in Mrt ogni agente viene abbinato ad un thread), si intuisce la rilevanza di quest'ultima caratteristica.

2.2.4 Wiiuse

Per poter utilizzare il Wiimote all'interno di questa tesi abbiamo bisogno di una libreria che riesca a comunicare con questo strumento. Nonostante è noto che il wiimote utilizza il protocollo Bluetooth per instaurare il canale di comunicazione, i criteri e le modalità con cui trasmette e riceve dati non lo sono. Per questo motivo sono state scritte diverse librerie per utilizzare il Wiimote all'interno della propria applicazione, ma fra tutte queste Wiiuse ci è sembrata la più stabile e la più completa. Questa libreria supporta

infatti diversi dispositivi oltre al Wiimote, come per esempio i Nunchuck e il controller di Guitar Hero 3.

Wiiuse ha la capicità di connettersi a tutti i Wiimote che trova, il numero massimo di connessioni può essere il limite fisico del sistema oppure un limite fissato via software. Per poter essere trovata una Wiimote deve essere regolata in modalità esplora (basta premere i pulsanti "1" e "2" contemporaneamente). Una volta che il Wiimote si connette al pc, invia il suo status che comprende informazioni riguardo l'attivazione della camera IR, l'attivazione degli accelerometri e il livello della batteria.

Il comportamento predefinito della Wiimote è di tenere spente tutte le sue parti non essenziali per limitare il consumo batteria,per questo motivo dobbiamo essere noi a chiedergli di attivare le parti che ci interessano, come la camera IR e gli accelerometri. Ogni volta che si crea un evento, dalla pressione di un tasto alla variazione degli accelerometri, il Wiimote invia tale evento. E' necessario quindi utilizzare una sorta di polling per gestire in maniera corretta tutti gli eventi generati dal Wiimote.

Oltre la comunicazione diretta dal Wiimote al pc che abbiamo descritto fino a questo momento, può esistere un comunicazione nel verso opposto. Il Wiimote possiede infatti quattro led azzurri e lacapacità di vibrare con intensità e durata variabile. La libreria Wiiuse consente di accendere indipendentemente ogni led a bordo del Wiimote e consente anche di gestire la sua capacità di vibrare.

2.2.5 Mr. Brian

Mr. Brian, Multilevel Ruling Brian Reacts by Inferring ActioNs, è il modulo di controllo usato in MRT. Il compito di questo modulo è essenziale per qualsiasi architettura software, in quanto si occupa di gestire i processi decisionali che determinano il comportamento dei robot autonomi.

Mr. Brian è un motore per un agente complesso che sfrutta il paradigma basato sui comportamenti. In questa ottica un compito complesso, come il nostro gioco, viene svolto dalla cooperazione di comportamenti attivati in accordo con la situazione che si presenta. Mr. Brian è l'espansione del modulo Brian, in cui è stata introdotta una gerarchia fra i comportamenti da utilizzare. A differenza degli altri componenti software e hardware descritti in questo capitolo, per riuscire a comprendere la soluzione che abbiamo adottato, occorre descrivere in dettaglio come funziona Mr. Brian.

Architettura generale

Mr. Brian utilizza predicati fuzzy per esprimere le regole di attivazione di un comportamento, per valutare il contesto in cui si trova partendo dai dati in ingresso e in generale a modellizzare la sua conoscenza interna che rispecchia il mondo esterno. I predicati fuzzy sono a tutti gli effetti come i predicati logici classici, solamente che si basano sulla logica fuzzy.

Nella logica tradizionale un elemento può appartenere o non appartenere ad un insieme, non esistono degli stadi intermedi. Nella logica fuzzy un elemento può appartenere o non appartenere ad un insieme, ma può anche appartenere parzialmente a un insieme o a entrambi. Come suggerisce la parola stessa, i confini degli insiemi non sono rigidi e ben delineati come nella logica tradizionale, ma sono più sfumati.

Un predicato fuzzy all'interno di Mr. Brian è composto da tre parti: un'etichetta che indica il nome del predicato, il grado di verità del predicato e la sua affidabilità. Mentre per il nome e il grado di verità è palese la loro funzione, l'affidabilità esprime la fiducia che riponiamo nel valore che assume il predicato.

Come abbiamo già accennato Mr. Brian si basa su una gerarchia di comportamenti. Questo significa che i comportamenti posso appartenere a livelli diversi. Un comportamento di livello superiore può conoscere le azioni proposte dai comportamenti di livello inferiore e può decidere perfino di eliminarle. Per esempio i comportamenti di livello n propongono di muovere il robot avanti e leggermente a sinistra. Il comportamento "*evita ostacoli*" di livello n+1 rileva che c'è un ostacolo davanti al robot, quindi elimina le azioni proposte dai comportamenti di livello n che porterebbero il robot a impattare con l'ostacolo. Il comportamento "*evita ostacoli*" tiene in considerazione le azioni proposte dal livello inferiore decidendo infatti di aggirare l'ostacolo girando a sinistra.

Ogni comportamento viene eseguito se, e solo se, il suo predicato di CANDO sia vero. Questo significa che ogni comportamento deve avere associato un predicato fuzzy che ne determina l'esecuzione. Il predicato di WANT indica per ogni comportamento, quanto vogliamo che venga eseguito. Siccome più comportamenti dello stesso livello possono imporre valori differenti allo stesso dato in uscita, Mr. Brian calcola la media dei valori pesandoli per il livello di WANT. Per questo motivo occorre che comportamenti diversi svolgano funzioni differenti. Riprendiamo l'esempio descritto prima, se al posto del comportamento "*evita ostacoli*", fossero presenti due comportamenti diversi: "*evita ostacoli a sinistra*" e "*evita ostacoli a destra*", potrebbe succedere che vengano eseguiti entrambi, con il risultato di far muovere il

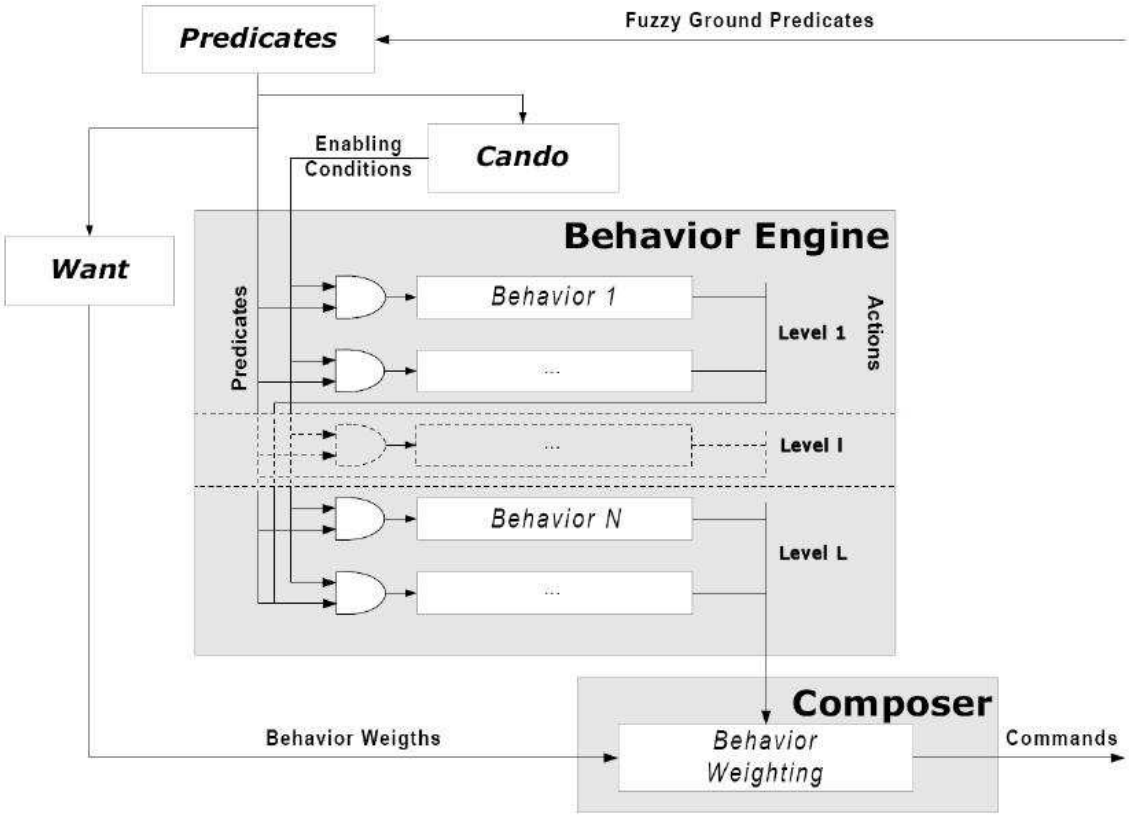


Figura 2.5: Flusso di lavoro di Mr. Brian

robot in avanti (supponendo che i loro predicati di WANT abbiano lo stesso valore), facendolo impattare con l'oggetto.

Nella Figura 2.5 viene mostrato quanto è stato descritto precedentemente. Nelle sottosezioni successive spigheremo meglio il processo di fuzzyficazione e defuzzyficazione. Per vedere nel dettaglio il funzionamento di Mr. Brian e come imparare a scrivere le regole per farlo funzionare nel modo corretto consultare il libro [3].

Fuzzyficazione

Per ogni dato in ingresso dobbiamo definire in che modo e a quali insiemi può appartenere. Per ogni variabile introdotta dobbiamo fornirne anche l'affidabilità. Il risultato di questa operazione è quello di produrre una serie di dati fuzzy. Se un elemento ha grado di appartenenza ad un insieme uno, significa che quest'ultimo appartiene totalmente all'insime; se un elemento ha grado di appartenenza zero invece non appartiene all'insieme. Nella Figura

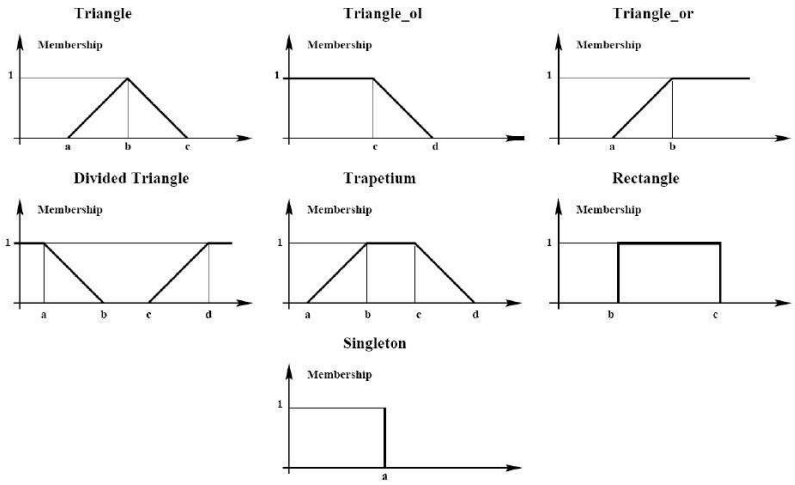


Figura 2.6: Forme disponibili per definire un insieme fuzzy

2.6 sono presenti le "forme" che Mr. Brian mette a disposizione per definire gli insiemi fuzzy. Come è possibile osservare, i segmenti obliqui che compongono le forme rappresentano il concetto di fuzzy; nella logica normale infatti non potrebbero esistere segmenti obliqui, ma solo segmenti perfettamente orizzontali o verticali.

Per comprendere meglio il processo di fuzzyficazione, consideriamo l'esempio in cui noi vogliamo fuzzyficare la variabile che contiene la distanza di un sonar. Abbiamo deciso che una distanza può appartenere a tre insiemi: *molto vicino*, *vicino*, *lontano*. Considerando che la misura del sonar è espressa in millimetri definiamo i tre insiemi in questo modo:

```
(DISTANZA_SONAR
(MOLTO_VICINO  (TRIANGLE_OL (200 300))),
(VICINO        (TRAPETIUM (200 300 600 700))),
(LONTANO       (TRIANGLE_OR (600 700)))
)
```

Questo stralcio di pseudocodice definisce gli insiemi per la DISTANZA SONAR. In particolare stabilisce per ogni insieme (*molto vicino*, *vicino*, *lontano*) il tipo di forma e la sua dimensione. Per comprendere i valori utilizzati riferirsi alla Figura 2.6. I predicati fuzzy si basano sui dati fuzzy generati in questo modo. Per esempio se in ingresso il sonar rileva una misura di 250 millimetri si origineranno tre dati fuzzy:

```
<MOLTO VICINO, grado di appartenenza 0.5, affidabilita' 1>
<VICINO, grado di appartenenza 0.5, affidabilita' 1>
<LONTANO, grado di appartenenza 0, affidabilita' 1>
```

Defuzzyficazione

Una volta ottenuta quindi la lista dei dati fuzzy in uscita si procede alla defuzzyficazione. Questo processo è l'inverso rispetto alla fuzzyficazione ma occorre definire anche in questa fase le forme di conversione tra dati fuzzy e valori da assegnare alle variabili. Come già descritto precedentemente i comportamenti assegnano dei valori a dei dati fuzzy in uscita. Questo implica che comportamenti diversi assegnano valori differenti allo stesso dato fuzzy. Occorre quindi che Mr. Brian trovi il modo di risolvere queste situazioni. Il procedimento utilizzato per defuzzyficare anche nel caso di valori multipli per lo stesso dato fuzzy, è quella di fare la media pesata con i valori dei predicati di WANT del comportamento che propone quel valore. La formula quindi usata è l'Equazione 2.1.

$$valore = \frac{\sum(peso \times valore)}{\sum(peso)} \tag{2.1}$$

Ricordiamo che la guida completa che contiene la sintassi che bisogna utilizzare e che spiega molto più nel dettaglio i procedimenti per utilizzare correttamente Mr. Brian è [3]. Questa spiegazione ha l'intenzione di fornire al lettore la conoscenza base per capire la soluzione che abbiamo adottato.

Capitolo 3

Analisi del problema

"Chiedersi se un computer possa pensare non è più interessante del chiedersi se un sottomarino possa nuotare. "

Edsger Wybe Dijkstra

In questo capitolo presentiamo l'analisi del problema, ovvero descriviamo dettagliatamente il gioco che dobbiamo realizzare e analizziamo i problemi che dobbiamo risolvere.

3.1 Descrizione del gioco

In questo gioco partecipano due attori: un'attore che deve cercare di sopravvivere nascondendosi, mentre l'altro deve cercare chi si nasconde e tentare di sparargli. Il ruolo di chi scappa lo interpreta Spykee, mentre è il giocatore umano che deve trovare Spykee e riuscire a sparargli. Il gioco inizia con Spykee e il giocatore schiena contro schiena in mezzo alla stanza. A questo punto il giocatore deve andare difronte al muro che ha davanti, far partire il gioco e contare fino a dieci. Dopo che il giocatore ha finito di contare può iniziare a cercare Spykee e tentare di sparargli.

Da quando il gioco parte Spykee ha dieci secondi per nascondersi al meglio delle sue capacità. Il costruttore di Spykee preoccupato della sorte del suo robot, gli ha installato quattro scudi energetici che lo proteggono dall'arma del giocatore. Colpire Spykee con anche solo uno scudo attivo è impossibile, per questo motivo il giocatore deve sovraccaricare tutti e quattro gli scudi prima di poter sparare a Spykee. Per sovraccaricare uno scudo il giocatore deve mantenere sotto tiro Spykee per 2.4 secondi. Attenzione, se si dovesse interrompere il contatto tra l'arma del giocatore e la schiena di Spykee gli scudi tenteranno di autorigenarsi uno alla volta. Per rigenerarsi

completamente, uno scudo ci mette 0.6 secondi. Spykee è dotato anche di un dispositivo in grado di sfruttare l'energia del proiettile dell'arma del giocatore per ricaricare completamente e istantaneamente tutti gli scudi. Un colpo affrettato può quindi vanificare tutti gli sforzi fatti fino ad ora.

Una volta che è iniziato il gioco Spykee deve dare il meglio di se per sopravvivere, infatti il giocatore acquista un punto se riesce a sparare a Spykee, viceversa se entro 20 secondi il giocatore non è ancora riuscito a colpirlo, è Spykee ad acquistare un punto. Il primo che arriva a tre punti si aggiudica la partita e il gioco può ricominciare.

Per maggiore chiarezza sono presenti dei led che indicano il numero di scudi sovraccaricati sia nel Wiimote, sia vicino alle spalle di Spykee, quindi prima di sparare occorre che tutti e quattro i led siano accesi. Quando il giocatore tiene sotto tiro Spykee, il led sopra la testa di Spykee si illumina.

3.2 Comportamenti

Dalla descrizione del gioco possiamo dedurre che l'obiettivo di Spykee è quello di riuscire a sopravvivere più tempo possibile evitando di essere colpito dal giocatore. Il comportamento che con maggiori probabilità garantisce a Spykee di sopravvivere è aspettare nel nascondiglio migliore che trova.

Per questo motivo l'obiettivo di Spykee appena inizia il gioco è quello di nascondersi il più velocemente possibile. Una volta che Spykee si è nascosto deve rimanere immobile fino a quando vince oppure fino a quando il giocatore lo trova e comincia a tenerlo sotto tiro.

Nel caso che il giocatore lo tenga sotto tiro, il nuovo obiettivo di Spykee diventa quello di sfuggire al giocatore. Per farlo occorre che Spykee si vada a nascondere ancora, in modo da guadagnare tempo per permettere agli scudi di rigenerarsi. In quest'ultimo caso Spykee può cercare un nuovo nascondiglio oppure cercare di sfruttare lo stesso nascondiglio per sfuggire dal giocatore.

3.3 Nascondersi

Leggendo con attenzione la strategia di gioco, si intuisce che nascondersi è il compito principale di Spykee. Per un robot l'atto di nascondersi è molto complesso. Per prima cosa deve individuare il nascondiglio dove andrà a nascondersi. Anche questa singola operazione è abbastanza impegnativa. Un robot per nascondersi ha bisogno di un solo nascondiglio, tuttavia quando analizza i dati provenienti dal mondo esterno, è possibile che trovi anche più

di un nascondiglio. Per questo motivo è necessario definire un metodo che indichi quale nascondiglio è migliore di un altro. Di seguito verranno quindi analizzati meglio l'individuazione di un nascondiglio e il metodo di giudicare un nascondiglio; il modo per poter sfruttare il nascondiglio verrà descritto nel Capitolo 4 come parte della soluzione del problema.

3.3.1 Individuazione dei nascondigli

Per nascondiglio si intende un luogo in cui ci si è sottratti alla vista della persona da cui ci vogliamo nascondere. L'unico sensore utile per riconoscere un nascondiglio è la camera di Spykee, in quanto i sonar non forniscono abbastanza informazioni riguardo all'ambiente esterno per questo scopo. Quando il gioco inizia, la camera di Spykee è rivolta dalla parte opposta rispetto allo sguardo del giocatore, per questa ragione possiamo assumere che il giocatore si trovi dietro a Spykee. Questa assunzione può essere estesa anche durante il gioco, perché il giocatore per poter sparare a Spykee deve essere alle sue spalle. Per questo motivo possiamo assumere che trovare un nascondiglio che permetta a Spykee di sottrarsi alla vista del giocatore significa trovare un nascondiglio presente nelle immagini della camera di Spykee.

Considerando che Spykee si può muovere solamente tramite cingoli e che si trova a terra, l'unico modo per poter non essere visto dal giocatore é trovarsi dietro ad un oggetto abbastanza grande da nascondere Spykee. In ultima istanza abbiamo affermato che individuare un nascondiglio significa individuare un oggetto che Spykee possa aggirare e nascondersi dietro inquadrato dalla camera di Spykee.

3.3.2 Valutazione dei nascondigli

Nella sezione precedente abbiamo stabilito che il nascondiglio è un oggetto che dobbiamo sfruttare per sottrarci alla vista del giocatore. Le caratteristiche che risultano essere di maggiore rilevanza sono quindi la dimensione dell'oggetto e la distanza da Spykee. Potrebbe essere utile anche sapere se l'oggetto ha componenti maggiormente orizzontali o verticali, ovvero se è più largo che alto o viceversa.

Ognuna di queste caratteristiche descrive in maniera parziale la qualitá di un nascondiglio, quindi é necessario creare un indice esprimibile con un solo valore che indica il punteggio di un oggetto come nascondiglio. In questo elaborato ci riferiremo a tale valore come punteggio di rank o rank di un oggetto. Per poter costruire il rank di un oggetto occorre tenere conto di tutte le sue caratteristiche, quindi occorre farne una media. La media semplice presuppone che tutti i fattori incidano in maniera uguale sul

punteggio di rank, ma questo non possiamo deciderlo deciderlo a priori. È utile quindi decidere il peso che ogni fattore apporta al rank di un oggetto.

Il punteggio di rank esprime quanto ci è utile un nascondiglio, ma l'utilità di un nascondiglio potrebbe variare in funzione del contesto in cui si trova Spykee; per esempio se libero di nascondersi potrebbe favorire oggetti di grandi dimensioni, mentre quando il giocatore lo sta tenendo sotto tiro potrebbe favorire nascondigli vicini.

Per valutare un nascondiglio occorre quindi estrarre tutte le informazioni necessarie dall'oggetto che vogliamo valutare, trovare un modo per esprimere queste informazioni in modo da essere numericamente confrontabili e infine decidere il peso di ogni informazione in funzione della minaccia che Spykee si trova a fronteggiare.

Capitolo 4

Progetto logico della soluzione del problema

"Un giorno le macchine riusciranno a risolvere tutti i problemi, ma mai nessuna di esse potrà porne uno."

Albert Einstein

In questo capitolo descriveremo la soluzione che abbiamo utilizzato per risolvere i due principali problemi presenti in questa tesi: individuare il nascondiglio migliore e riuscire a rendere Spykee autonomo. Il primo é un problema di visione, mentre per risolvere il secondo problema abbiamo scelto di usare Mr. Brian. Di seguito spiegheremo nel dettaglio le soluzioni che abbiamo adottato.

4.1 Il problema di visione

In questa sezione descriveremo in dettaglio la soluzione logica dell'algoritmo di visione che si occupa di individuare e analizzare i nascondigli trovati e che si fa carico di valutare la qualitá dei nascondigli individuati al fine di selezionare il nascondiglio migliore.

4.1.1 Individuazione dei nascondigli

Come accennato nel Capitolo 3, l'unico sensore in grado di individuare un nascondiglio è la camera di Spykee. Per questo motivo il problema di individuazione del nascondiglio è un problema di visione. La soluzione ideale per identificare un oggetto da utilizzare come nascondiglio sarebbe ricostruire l'ambiente in tre dimensioni e operare su tale ricostruzione. Questa

operazione richiede la presenza di almeno due camere per poter essere effettuata, ma Spykee ne possiede una sola. In letteratura è presente anche una soluzione alternativa, che prevede l'utilizzo di una sola camera in movimento; per piccoli movimenti infatti si possono utilizzare due fotogrammi dalla camera di Spykee come due immagini catturate da due camere differenti e si utilizza la rototraslazione della camera per ricostruire l'ambiente in tre dimensioni. Purtroppo non abbiamo potuto seguire neanche questo approccio perché non potendo conoscere di quanto si muove Spykee (non ha odometria), non siamo riusciti a ricavare la rototraslazione. Per queste ragioni ci siamo focalizzati nell'utilizzare un'analisi cromatica dell'immagine per individuare i nascondigli.

La nostra soluzione è efficace se il mondo esterno rispetta due vincoli. Il primo vincolo richiede che il pavimento, o comunque la superficie su cui si muove Spykee, sia omogeneo e quindi che non presenti eccessive variazioni di colore. Il secondo vincolo richiede che gli oggetti da utilizzare come nascondiglio siano carichi di colore o molto scuri, mentre le pareti, o comunque lo sfondo dello scenario, sia di colore chiaro.

Prima di spiegare l'elaborazione effettuata sulle immagini, è necessario conoscere come sono rappresentate all'interno di un computer. Tutte le immagini sono composte da una serie ordinata di punti colorati, chiamati pixel: piú precisamente questi punti sono disposti in una tabella, o matrice, con determinate righe e colonne, il loro numero dipende dalla risoluzione. Per esempio un'immagine con risoluzione 1024x768 pixel significa che l'immagine è racchiusa in una matrice con 1024 colonne e 768 righe. Nella codifica RGB (utilizzato in quasi tutti i tipi di immagini, compreso quello della camera di Spykee) il colore di un pixel è scomposto in tre colori fondamentali: il rosso, il verde e il blu. Infatti combinando opportunamente le quantità di ciascuno questi tre colori si possono ottenere tutti i colori che si desiderano. Un'altra caratteristica delle immagini digitali è la profondità di colore, questa misura indica in quanti bit viene memorizzata la quantità del colore. Nelle immagini attuali la profondità di colore è 24 bit, questo significa che 8 bit sono dedicati per rappresentare il rosso, altri 8 per il verde e gli ultimi 8 bit per il blu.

Il nostro algoritmo riceve in ingresso le immagini catturate da Spykee, in RGB e con la risoluzione di 320x240 pixel. Quindi il nostro algoritmo di visione deve operare solamente con tre matrici con 320 colonne e 240 righe, che rappresentano in una scala da 0 a 255 la quantità del colore. Ogni matrice (o piano) rappresenta quindi uno dei tre colori fondamentali. Valori tendenti a zero indicano l'assenza di colore, mentre valori tendenti a 255 indicano la massima presenza di colore. L'idea generale alla base di questo

algoritmo è cercare e analizzare le parti dell'immagine che hanno valori bassi, perché sotto le nostre ipotesi rappresentano gli oggetti da utilizzare come nascondigli.

Nella codifica RGB i colori chiari o tendenti al giallo vengono espressi con valori alti, quindi per aumentare l'efficacia del nostro algoritmo è conveniente convertire l'immagine nella codifica HSB (Hue Saturation Brightness), in cui il colore del pixel non viene più scomposto in tre colori, ma viene diviso in tonalità saturazione e luminosità. Il piano che indica la tonalità, per come viene costruito, è esattamente quello che serve ai nostri scopi. I restanti piani contengono delle informazioni non rilevanti per trovare un nascondiglio, per cui utilizzeremo solo il piano della tonalità.

Vogliamo far notare che solo preparando l'immagine ad essere analizzata abbiamo trasformato un'immagine a colori in un'immagine in scala di grigio, perdendo in questo modo molte informazioni.

Il passo successivo è quello di trovare i contorni degli oggetti trovati nell'immagine per poterne determinare le informazioni necessarie, come per esempio la posizione e la dimensione. Le librerie OpenCv mettono a disposizione un'unica funzione in grado di fare questo: cvFindContours (derivato da Suzuki [Suzuki85]); tale funzione infatti analizza un'immagine binaria, ovvero in bianco e nero (che non significa in scala di grigio), e restituisce i contorni trovati.

Prima di parlare di contorni è necessario definire esattamente cosa sono. Un contorno è una lista di punti che rappresenta, in un modo o nell'altro, una curva nell'immagine. Nelle OpenCv i contorni sono rappresentati da una sequenza che contiene (oltre al contorno stesso) il riferimento al contorno successivo e al contorno inferiore. Per capire meglio come questo algoritmo funzioni è necessario guardare la Figura 4.1.

La parte superiore della figura rappresenta delle regioni bianche (etichettate da A a E) su uno sfondo grigio. La parte inferiore dell'immagine rappresenta la stessa immagine ma con evidenziati i contorni che rileva cvFindContours. Questi contorni sono etichettati cX o hX, dove c sta per "contour" (contorno), h sta per "hole" (buche) e "X" per qualche numero. Alcuni di questi contorni sono linee tratteggiate; esse rappresentano i limiti esterni delle regioni bianche. Gli altri contorni sono linee puntinate e rappresentano i limiti interni delle regioni bianche oppure i limiti esterni delle buche presenti nella regione bianca che li contiene.

Il concetto di contenimento è molto importante in questo caso perchè permette di comprendere meglio come esplorare le varie regioni bianche all'interno della figura. CvFindContours restituisce infatti i contorni trovati come sequenze (cvSeq). Questa struttura oltre a contenere una sequenza

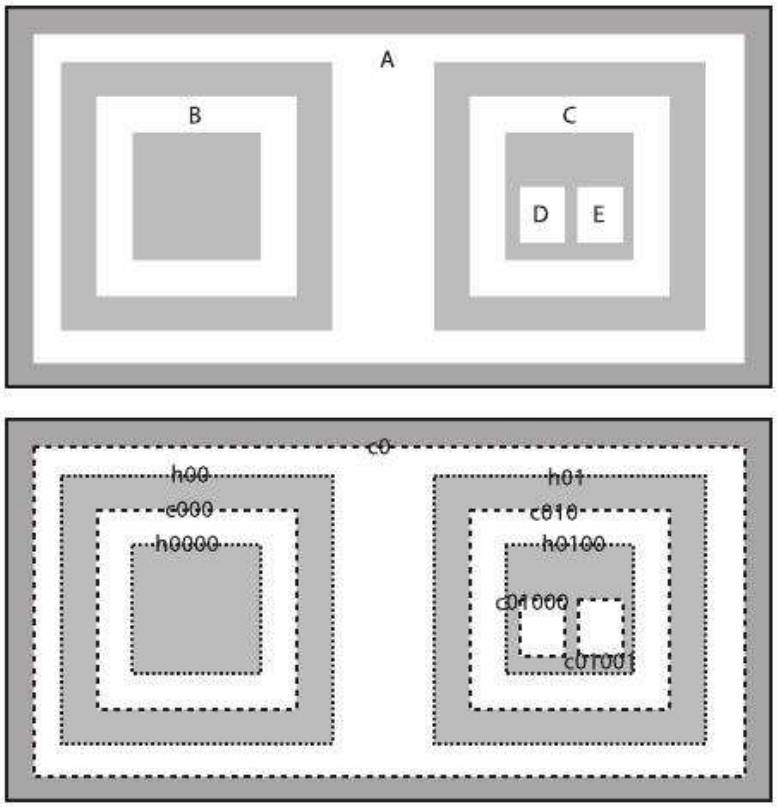


Figura 4.1: I contorni inividuati dalla funzione cvFindContours

di punti (in questo caso un contorno), contiene i riferimenti alla sequenza successiva e alla sequenza inferiore; per ulteriori dettagli su come viene memorizzata e utilizzata dalle OpenCv consultare [1]. Per esempio se noi esaminassimo la sequenza che contiene il contorno h00 (o sequenza h00 per brevità), avremmo come riferimento successivo h01 e come riferimento inferiore c000; mentre se esaminassimo la sequenza c0, non avremmo alcun riferimento successivo e la sequenza h00 come riferimento inferiore.

Quello che occorre fare per individuare i nascondigli è quindi creare un'immagine binaria, partendo da quella in scala di grigi che abbiamo ottenuto dall'immagine che ha acquisito la camera di Spykee. Per ottenere un'immagine binaria con lo scopo di trovare i contorni degli oggetti da utilizzare come nascondiglio ci sono due modi. Il primo metodo utilizza l'immagine restituita dall'algoritmo di Canny. Questo algoritmo è usato per individuare tutti i punti che interpreta come bordo presenti nell'immagine, una volta individuati li disegna nell'immagine che restituisce. Il secondo metodo per ottenere un'immagine binaria è quello di fissare una soglia e

forzare il colore del pixel a zero se sotto soglia altrimenti a 255. Le OpenCv mettono a disposizione due tipi di soglie differenti. Il primo tipo analizza un solo pixel alla volta, confrontandolo con la soglia e agendo di conseguenza sul pixel. Il secondo tipo di soglia non valuta un singolo pixel, ma valuta anche l'intorno di quel pixel. Il valore di ogni pixel analizzato viene infatti mediato con il valore dei pixel attorno a lui. Il numero e il peso da attribuire ai pixel considerati può essere scelto a piacere. In un caso generale utilizzare una soglia per costruire un'immagine binaria non garantisce di trovare i contorni degli oggetti, tuttavia considerando i vincoli in cui il nostro algoritmo opera, anche questo modo di realizzare un'immagine binaria risulta efficace.

Potenzialmente tutti questi metodi possono garantire un buon risultato, per questo motivo abbiamo deciso di realizzare ed affinare al meglio tutte le alternative che abbiamo a disposizione. La scelta che è risultata essere più efficace è quella di utilizzare il filtro binario semplice ed elaborare successivamente l'immagine binaria. Di seguito illustreremo in dettaglio l'algoritmo utilizzato, dalla cattura dell'immagine dalla camera di Spykee fino al ritrovamento dei singoli oggetti utilizzando la funzione CvFindContours, il tutto diviso in cinque passi.

Primo passo: acquisire l'immagine

Il primo passo è quello di ottenere l'immagine da Spykee e convertirla nel formato HSV, i motivi di tale conversione sono stati precedentemente esposti. Da questo punto in avanti l'immagine che utilizziamo si riduce al piano della tonalità dell'immagine acquisita.

Secondo passo: eliminare il pavimento

Con il termine pavimento indichiamo la superficie in cui Spykee si muove. Il secondo vincolo che limita la generalità della nostra applicazione, impone che il pavimento deve essere uniforme, non entra nel merito del colore del pavimento. Il nostro algoritmo deve riuscire ad essere efficace anche in presenza di un pavimento dal colore intenso o scuro. Un pavimento di questo tipo renderebbe inefficace il nostro algoritmo, in quanto il pavimento verrebbe considerato come un possibile nascondiglio.

Per questo motivo è necessario assicurarci che l'algoritmo di visione riconosca il pavimento come elemento di sfondo. La soluzione più semplice che abbiamo ideato è quello di colorarlo di bianco. Siccome la camera di Spykee si trova ad una bassa altezza, é lecito affermare che l'estrema zona inferiore dell'immagine inquadra il pavimento.

Questa considerazione ci consente di colorare il pavimento utilizzando la funzione cvFloodFill. Questa funzione inizia selezionando un punto dell'immagine, confronta il colore di ogni pixel adiacente al punto selezionato e include nella selezione i pixel che hanno il colore simile al punto selezionato in partenza. Successivamente prende in considerazione tutti i pixel appena aggiunti e itera il procedimento appena descritto fino a quando non riesce ad aggiungere piú pixel alla selezione. DopodichÃ© disegna la selezione ottenuta alla fine del processo del colore desiderato. Noi possiamo decidere quanto un pixel deve essere simile ad un altro per essere incluso nella selezione. Come suggerisce il nome, questa funzione agisce come un'allagamento che parte da un punto e si propaga all'interno dell'immagine.

Secondo il nostro vincolo il pavimento deve essere uniforme, quindi basterebbe applicare questa funzione ad un solo punto per eliminare il pavimento, ma in questo caso un'illuminazione non omogenea vanificherebbe i nostri sforzi. La soluzione che abbiamo utilizzato è quella di eseguire cvFloodFill su tre punti: l'angolo in inferiore sinistro, inferiore centrale e l'angolo in inferiore destro.

L'eliminazione del pavimento effettuata in questo modo cela un'insidia. Quando Spykee si avvicina ad un oggetto, può capitare che l'oggetto raggiunga la zona inferiore dell'immagine. Se questo succede in corrispondenza di uno dei punti di floodfill, rischiamo di riservare all'oggetto la stessa sorte del pavimento. Considerando che solitamente un oggetto è prevalentemente monocromatico, il nostro è qualcosa in più di un rischio. La soluzione che abbiamo adottato è quella di prelevare il colore dei tre punti di floodfill e confrontarli. Se tutti e tre i punti hanno lo stesso colore eseguiamo il floodfill su tutti i punti, se almeno due punti hanno lo stesso colore eseguiamo il floodfill solo su quei due punti, mentre se tutti i punti hanno dei colori diversi non eseguiamo il floodfill.

Per aumentare l'efficacia del nostro algoritmo in questo passo, il controllo del colore, per essere sicuri di evitare di eliminare un'oggetto, viene effettuato sull'immagine a colori che ha acquisito Spykee. Il floodfill viene invece effettuato normalmente sul piano della tonalità come il resto delle altre elaborazioni. Riassumendo, arrivati a questo punto abbiamo quindi colorato il pavimento di bianco, rendendolo di fatto un elemento di sottofondo.

Terzo passo: creiamo l'immagine binaria

Una volta eliminato il pavimento siamo pronti per creare la nostra immagine binaria. In questo punto dell'algoritmo abbiamo un'immagine i cui pixel con valore basso rappresentano oggetti, mentre i pixel con valore alto rapp-

resentano le pareti o comunque elementi di sfondo che non ci sono di alcuna utilità. Il filtro che utilizziamo è quello semplice che confronta punto per punto, solamente che lo facciamo funzionare al contrario: se il colore del pixel analizzato è maggiore della soglia viene forzato a zero, altrimenti viene forzato a 255.

Utilizzando questo accorgimento abbiamo quindi realizzato un'immagine esattamente come richiesto dalla funzione cvFindContours, ovvero un'immagine in cui le regioni bianche rappresentano gli oggetti da utilizzare come nascondigli, mentre le regioni nere rappresentano lo sfondo dell'immagine.

Quarto passo: prepariamo l'immagine binaria

L'immagine come l'abbiamo lasciata al terzo passo non è ancora pronta per essere utilizzata per trovare i contorni degli oggetti: il rumore presente nell'immagine, le fughe delle piastrelle presenti nel pavimento e altre curve non volute vanificherebbero gli sforzi effettuati da cvFindContours.

Per pulire l'immagine occorre effettuare processi di erosione e di dilatazione. La dilatazione è un processo che permette alle regioni chiare di aumentare di volume; viceversa l'erosione è un processo che permette alle zone chiare di diminuire di volume, eventualmente anche a sparire se sono regioni che occupano una superficie inferiore a quella che viene erosa dal processo.

Il primo intervento consiste nell'eliminare i rumori e le curve che disturbano l'immagine. A questo scopo applichiamo un processo di erosione e dilatazione abbastanza incisivo. Il risultato di questa operazione è quello di eliminare piccole regioni chiare, come per esempio le fughe delle piastrelle del pavimento e di mantenere invariate le regioni più grandi. Analogamente applichiamo un processo di dilatazione ed erosione altrettanto incisivo per eliminare alcuni "buchi" all'interno degli oggetti, generalmente originati da ombre o illuminazioni artificiali.

L'immagine ora risulta essere molto migliorata, ma sussiste ancora un problema che potrebbe portare cvFindContours ad ignorare un oggetto. Se un oggetto non termina all'interno dei limiti dell'immagine, ma continua oltre il bordo destro o sinistro, cvFindContours non riesce a rilevarlo correttamente. Per questo motivo coloriamo di nero il bordo esterno dell'immagine con uno spessore di qualche pixel. In questo modo siamo sicuri di riconoscere tutti gli oggetti all'interno dell'immagine di Spykee.

Quinto passo: trovo i contorni degli oggetti

Utilizzando l'immagine binaria che abbiamo ricavato dai passi precedenti, possiamo utilizzare la funzione cvFindContours per ottenere i contorni degli oggetti presenti. Quello che abbiamo ottenuto dopo questa operazione è un albero che contiene tutti i contorni trovati. Per come elaboriamo l'immagine prima di trovare i contorni, l'albero ottenuto ha una struttura logica fissa. La radice dell'albero rappresenta il contorno dello sfondo e tutti i suoi figli sono gli oggetti trovati.

Nella Figura 4.2 mostriamo il risultato dell'elaborazione che viene eseguita seguendo i passi descritti sopra. Abbiamo scelto di mostrare questa figura perché è evidente l'effetto dell'illuminazione. Come è possibile osservare, l'algoritmo individua anche due oggetti fantasma che sono originati dalle zone d'ombra presenti nell'immagine. Questi oggetti occupano sempre una piccola superficie e solitamente compaiono agli estremi dell'immagine, per questi motivi non vengono mai selezionati come nascondiglio migliore, sempre se valgono le ipotesi formulate precedentemente sul mondo esterno.

I riquadri blu che vedete nell'Immagine 4.2f sono ottenuti tramite le OpenCv; in particolare sono i più piccoli rettangoli in grado di contenere i contorni trovati. Nella scatola di sinistra si vede come la sua ombra aumenti di parecchio la dimensione del rettangolo che la racchiude. Questo è un altro problema da affrontare nella valutazione dei nascondigli.

Nell'angolo inferiore sinistro della Figura 4.2e è presente una regione bianca che non risulta essere presente negli ostacoli trovati nella Figura 4.2f, il motivo di questo comportamento lo spiegheremo nella sezione sulla valutazione dei nascondigli.

4.1.2 Valutazione dei nascondigli

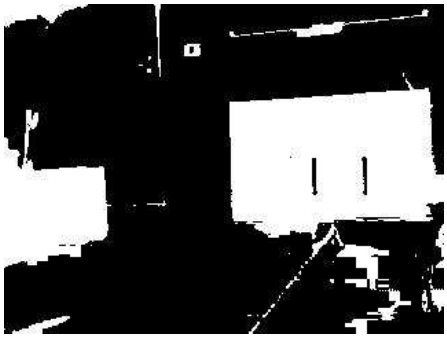
La prima parte del nostro algoritmo di visione ha quindi individuato tutti gli oggetti presenti nell'immagine da sfruttare come nascondiglio. L'obiettivo di questa seconda parte è diviso in due fasi: nella prima occorre analizzare tutti gli oggetti e ricavarne tutte le informazioni necessarie, nella seconda occorre trovare un modo per unire in un unico indice, tutte le informazioni estrapolate in funzione della minaccia che Spykee dovrà affrontare.

Per poter analizzare i nascondigli abbiamo a disposizione solamente l'immagine originale catturata dalla camera di Spykee e i contorni individuati nella fase precedente. Utilizziamo una classica funzione ricorsiva per esplorare l'albero dei contorni allo scopo di analizzare tutti gli oggetti predenti

(a) Immagine catturata da Spykee.

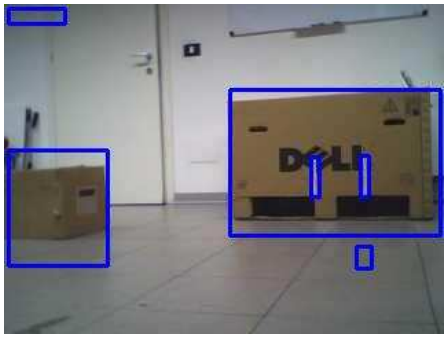
(b) Piano tonalitá dell'immagine.

(c) Eliminazione del pavimento.



(d) Creazione immagine binaria.

(e) Elaborazione immagine binaria.



(f) Ostacoli trovati

Figura 4.2: L'algoritmo di individuazione dei nascondigli.

nell'immagine. Occorre evitare di analizzare il contorno che racchiude lo sfondo dell'immagine. Per evitare di sprecare risorse evitiamo anche di analizzare gli oggetti troppo piccoli. È per questo motivo che nella Figura 4.2f non compariva l'ombra in basso a destra. Esplorando indistintamente tutto l'albero dei contorni, corriamo il rischio di processare dei contorni interni di un oggetto come un altro possibile oggetto da sfruttare come nascondiglio. Un esempio é proprio nella Figura 4.2 dove le strisce di nastro adesivo bianco nello scatolone di destra vengono individuati come nascondigli. A priori non possiamo affermare che gli oggetti da poter sfruttare come nascondigli sono solo quelli rappresentati dai contorni contenuti nei figli della radice dell'albero; tuttavia possiamo affermare che un contorno padre sarà sempre più grande e piú vicino a Spykee di qualsiasi contorno che contiene, quindi il contorno padre sarà sicuramente migliore. Per questo motivo possiamo tranquillamente esplorare ricorsivamente l'albero dei contorni e analizzare ogni suo nodo interpretandolo come possibile nascondiglio.

Dall'analisi abbiamo dedotto che la qualità di un nascondiglio (chiamata rank) è influenzata dalla sua dimensione, dalla sua distanza, e in minor parte dalle sue componenti spaziali. Di seguito elencheremo come ricaviamo i singoli fattori, oltre come abbiamo utilizzato questi valori per calcolare il rank di un oggetto.

Dimensione

Per calcolare la dimensione dell'oggetto utilizzavamo in prima istanza l'area del rettangolo che racchiude il contorno. Abbiamo notato che tale approccio non forniva risultati molto attendibili in quanto il rettangolo che racchiude il contorno può essere ingrandito dall'ombra dell'oggetto, oppure la geometria concava dell'oggetto falserebbe la dimensione del riquadro. Un esempio di questa situazione è presente anche nell'immagine di esempio. Per queste ragioni abbiamo deciso di utilizzare una funzione delle OpenCv che calcolava esattamente l'area che racchiude un contorno.

Distanza

Il calcolo di questo fattore è meno diretto rispetto alla dimensione, perchè non esiste una funzione in grado di calcolarla. Per trovare la distanza a cui si trova un oggetto abbiamo assunto che la zona inferiore dell'immagine sia il pavimento e che Spykee si trovi in centro all'immagine; possiamo assumere che Spykee si trovi al centro dell'immagine perché la camera di Spykee è montata al centro di Spykee e possiamo regolare l'angolo della camera in modo che la zona inferiore dell'immagine inquadri il pavimento.

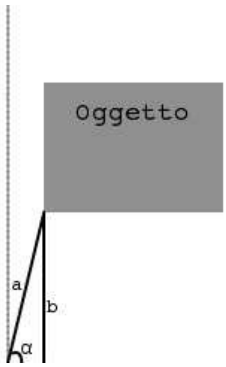


Figura 4.3: Definizione della distanza di un oggetto

La distanza è definita come la misura dello spazio fra una sorgente e una destinazione. Nel nostro caso è sensato considerare come origine Spykee; questo significa che l'origine nell'immagine è rappresentata dal pixel centrale nell'ultima riga in basso. Fissando in questo modo il nostro punto di riferimento abbiamo diviso l'immagine in due parti: una parte a destra di Spykee e una parte a sinistra. La destinazione deve essere un punto significativo dell'oggetto che vogliamo utilizzare come nascondiglio. Come spiegheremo più dettagliatamente in questo capitolo a noi interessa aggirare l'oggetto per andare a nasconderci dietro ad esso. Per questo motivo ci interessa la distanza dell'estremo destro o sinistro dell'oggetto. Per questo motivo se il punto centrale dell'oggetto è nella parte sinistra dell'immagine cercheremo di avvicinarci al lato destro dell'oggetto, viceversa cercheremo di avvicinarci al lato sinistro dell'oggetto.

Abbiamo appena affermato che la destinazione deve essere il punto inferiore del lato dell'oggetto che stiamo analizzando. In prima istanza utilizzavamo lo spigolo del rettangolo che conteneva il contorno, tuttavia come la Figura 4.2 mostra, il contorno di un oggetto non è detto che abbia una forma regolare, ma al contrario zone d'ombra o la geometria dell'oggetto rendono inadeguato tale metodo. Per questo motivo la destinazione per il calcolo della distanza non è immediato ma deve essere individuato.

Una volta individuato il lato dell'oggetto che ci interessa, ci spostiamo di pochi pixel verso il centro dell'immagine per quanto riguarda l'ascisse della destinazione. L'ordinata della destinazione la facciamo partire un pixel al di sotto del lato inferiore del rettangolo che delimita il contorno. A questo punto iniziamo ad aumentare l'ordinata della destinazione fino a quando rileviamo un cambiamento nel colore dell'immagine, che rappresenta l'inizio dell'oggetto. Il confronto avviene utilizzando l'immagine originale catturata dalla camera di Spykee.

Come illustrato nella Figura 4.3 la distanza che ci interessa è rappresentata dal segmento a. Per poter calcolare questa distanza espressa in centimetri dobbiamo trovare un modo di ottenere una distanza in pixel convertibile in maniera semplice. Per questo motivo otteniamo la distanza in pixel rappresentata nell'immagine dal segmento b e la convertiamo in centimetri. Questa conversione non è lineare: se ci si allontana dal bordo inferiore dell'immagine un pixel rappresenta sempre più centimetri. Questo è dovuto al fatto che le camere deformano la realtà che riprendono. Per riuscire a convertire la misura, attraverso dati sperimentali abbiamo suddiviso l'immagine in segmenti e abbiamo approssimato in tali segmenti che la conversione sia lineare. Questo stratagemma ci ha permesso di ottenere una funzione che converte la distanza espressa in pixel in una misura espressa in centimetri. Questa conversione fornisce risultati con un errore di pochi centimetri se l'ostacolo si trova entro tre metri; oltre i tre metri l'errore aumenta esponenzialmente.

Guardando La Figura 4.3 possiamo notare come i segmenti a e b formino un triangolo rettangolo. Per questo motivo riusciamo a calcolare la misura del segmento a espressa in centimetri partendo dalla misura in centimetri del segmento b. Il primo passo richiede il calcolo dell'angolo α, considerando che abbiamo la misura in pixel di tutti i lati del triangolo che si viene a formare, possiamo calcolare l'angolo α tramite la Formula 4.1, dove c è il lato rimanente del triangolo rettangolo.

$$\alpha = \arctan(\frac{b}{c}) \tag{4.1}$$

Utilizzando la Formula 4.2 è possibile ottenere la misura del segmento a espressa in centimetri. Data la simmetria del nostro problema, si può considerare equivalente trovare la distanza di un oggetto posto a sinistra di Spykee.

$$a = \frac{b}{\sin(\alpha)} \tag{4.2}$$

Come si può intuire dal procedimento utilizzato, il calcolo della distanza dell'oggetto si basa su precise assunzioni, ed è derivata da parecchie approssimazioni dovute alla deformazione della realtà che effettua la camera di Spykee. Le misura della distanza che otteniamo risulta avere una precisione sufficiente per il nostro utilizzo.

Un oggetto è raramente perfettamente allineato con Spykee, quindi oltre alla misura del segmento a è utile affiancargli anche il valore dell'angolo α. Per come abbiamo costruito il nostro sistema di riferimento, valori negativi dell'angolo indicano che la destinazione si trova a sinistra di Spykee, valori

positivi indicano che la destinazione si trova a destra. Tale valore può variare in modulo da 0 a $\frac{\pi}{2}$.

Vogliamo sottolineare il fatto che per poter essere attendibile occorre posizionare la camera di Spykee esattamente nella posizione che abbiamo utilizzato per convertire la misura della distanza, altrimenti le misure non saranno veritiere. Per effettuare questa operazione abbiamo messo a disposizione un'applicazione apposita, tale procedure è riportata nell'Appendice B.

Per essere utilizzato nel calcolo del rank, ogni fattore deve essere normalizzato. Il primo approccio è stato quello di utilizzare una normalizzazione assoluta, in cui si usavano delle misure di aree o di distanze campione. Il pensiero alla base du questa scelta, era che Spykee continuasse a girare su stesso fino a quando rilevasse il nascondiglio migliore in assoluto. Siccome Spykee non ha odometria e considerando che non riconosce il giocatore, questo comportamento potrebbe produrre risultati pessimi. A utilizzare una normalizzazione assoluta si rischia di incappare in un altro tipo di problema. L'idea del punteggio di rank di un nascondiglio, espresso con un unico valore, calcolato come la media pesata dei contributi dei singoli fattori, come dimensione o distanza, implica che, questi fattori, devono essere numericamente confrontabili. Per normalizzare la dimensione, la si divide per la dimensione massima possibile di un oggetto. A rigor di logica la dimensione massima possibile di un oggetto è la dimensione dell'immagine. La scelta di tale valore comporta peró valori bassissimi in quanto mediamente l'area di un oggetto è molto minore rispetto all'area totale. Un discorso analogo può essere fatto per la distanza. Per questo motivo abbiamo optato per effettuare una normalizzazione relativa. Il calcolo del rank di un oggetto avviene quindi in due fasi. Nella prima fase viene esplorato l'albero dei contorni, viene creata una lista di nascondigli specificandone tutte le qulità come dimensione, distanza e angolo; infine vengo memorizzate la distanza e la dimensione massima rilevate in oggetti presenti nell'immagine. Nella seconda fase si esplora la lista di nascondigli e viene normalizzato ogni fattore di ogni nascondiglio, utilizzando i valori massimi trovati nella prima fase, inoltre a seconda del livello di minaccia vengono variati i coefficienti per il calcolo del rank ed infine si calcola il rank finale di un nascondiglio. Se Spykee si trova sotto fuoco si preferiscono ostacoli vicini, altrimenti si prediligono ostacoli grandi. L'ostacolo restituito dal nostro algoritmo é l'ostacolo con rank maggiore, che risulta essere quindi l'ostacolo migliore.

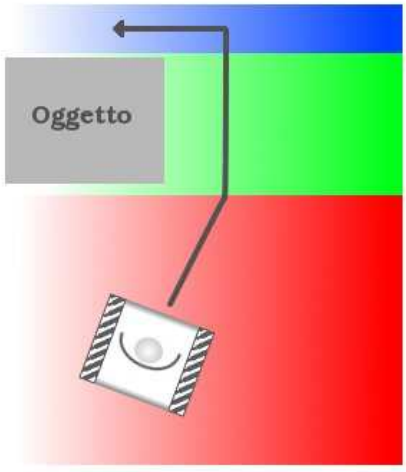


Figura 4.4: Manovra ideale per sfruttare un oggetto come nascondiglio

4.2 Il problema di controllo

Abbiamo incaricato Mr. Brian di decidere le azioni di Spykee per permettergli di diventare un degno avversario per il giocatore umano. Mr. Brian è un motore basato sul paradigma dei comportamenti, quindi il nostro obiettivo è quello di fornire a Mr. Brian un modello adeguato per rappresentare il contesto in cui si trova e scrivere i comportamenti che conducano Spykee alla vittoria.

Per riuscire a comprendere la soluzione che abbiamo adottato è necessario possedere un'infarinatura sul funzionamento di Mr. Brian; la lettura del Capitolo 2 fornisce tali conoscenze basilari, ma per una lettura approfondita su Mr. Brian leggere [3].

Dall'analisi abbiamo concluso che per sfruttare un oggetto come nascondiglio occorre aggirarlo e fermarsi dietro al suo lato posteriore, in modo da riuscire a sottrarci dallo sguardo del giocatore. La traiettoria teorica per riuscire a eseguire questa manovra é rappresentata in Figura 4.4. Questa manovra è divisa in tre fasi precise: l'avvicinamento (zona colorata di rosso), il costeggiamento dell'ostacolo (zona colorata di verde) e l'aggirare l'ostacolo (zona colorata di blu).

L'atto di nascondersi comprende otto stati:

1. Cerco un nuovo nascondiglio.

2. Avvicinamento al nascondiglio.

3. Avvicinamento al nascondiglio alla cieca.
4. Costeggiamento nascondiglio.
5. Aggira il nascondiglio - fase 1.
6. Aggira il nascondiglio - fase 2.
7. Aggira il nascondiglio - fase 3.
8. Nascosto.

La spiegazione dettagliata di ogni stato e il passaggio da uno stato all'altro viene spiegata nel dettaglio nei comportamenti che l'effettuano. Questa lista è stata inserita per fornire al lettore un quadro di insieme degli stati che abbiamo utilizzato.

Noi abbiamo progettato i comportamenti necessari a Spykee per nascondersi, in modo che la geometria dell'oggetto da utilizzare come nascondiglio influenzi il meno possibile l'esito della manovra. Oltre ai comportamenti che consentono a Spykee di nascondersi, occorrono altri due comportamenti: un comportamento per cercare i nascondigli e uno per disimpegnarsi nel caso Spykee risulti essere sotto tiro da parte del giocatore.

Prima di eseguire i comportamenti Mr. Brian crea la sua rappresentazione del mondo in base agli ingressi che gli vengono forniti. Ogni volta viene chiesto a Mr. Brian di decidere che azioni intraprendere, gli dobbiamo fornire le seguenti informazioni riguardo il contesto in cui si trova:

Distanze sonar Rappresenta le distanze che rilevano i sonar nelle quattro direzioni.

Differenze sonar Ogni volta che cambia il valore della distanza di un sonar, viene calcolata la differenza del valore precedente rispetto al nuovo valore. Quando non cambia il valore della distanza tale differenza viene posta a zero. Vogliamo precisare che chiediamo a Mr. Brian di prendere decisioni diecimila volte al secondo, mentre i sonar riescono a ottenere tre, raramente quattro, valori al secondo.

Nascondiglio Rappresenta l'ultimo nascondiglio che l'algoritmo di visione ci fornisce. Le uniche informazioni che Mr. Brian possiede, sono la distanza dell'oggetto, l'angolo rispetto al centro dell'immagine e se l'angolo si riferisce allo spigolo destro o sinistro dell'oggetto. Come spiegato nella sezione precedente le misure dell'angolo e della distanza si riferiscono allo spigolo più vicino al centro dell'immagine. Nel nostro

sistema di riferimento il centro dell'immagine rappresenta l'origine, e l'angolo a cui si trova lo spigolo del nascondiglio assume valori positivi se si trova destra e valori negativi se si trova sinistra. Il segno dell'angolo tuttavia non ci garantisce che valori positivi dell'angolo indichino che l'oggetto si trova a destra e quindi Mr. Brian consideri lo spigolo sinistro per costeggiarlo o viceversa nel caso di valori negativi. Per questo motivo occorre specificare lo spigolo che prendiamo come riferimento. Considerando che l'obiettivo di Spykee è quello di costeggiare il nascondiglio, occorre tenere presente la dimensione che occupa Spykee. Per questo motivo occorre compensare l'angolo che ci fornisce l'algoritmo di visione. Tale compensazione non è un valore fisso, ma varia in funzione della distanza dell'oggetto. Questa funzione non è lineare ma tende a convergere a novanta gradi per piccole distanze, mentre tende a stabilizzarsi a trenta gradi per distanze elevate. Anche in questo caso abbiamo ripartito la distanza in sottoinsiemi e abbiamo assunto che fosse lineare in quei sottoinsiemi. Per questo motivo prima di utilizzare la nostra applicazione occorre calibrare la camera come illustrato nell' Appendice B. Vogliamo precisare che la compensazione dell'angolo è in funzione di una misura approssimata ottenuta dalla camera, quindi se la camera non è perfettamente allineata, la compensazione non risulta essere efficace.

Differenza del nascondiglio Quando l'algoritmo di visione identifica un nuovo nascondiglio, viene calcolata la differenza di dimensione e dell'angolo. La differenza di dimensione rappresenta il rapporto fra le due dimensioni moltiplicato per cento. Quindi per valori tendenti al 100% si hanno due oggetti di dimensione identica. La differenza dell'angolo indica di quanti gradi l'angolo del primo oggetto differisce dal secondo. Se lo stato è *Avvicinamento al nascondiglio alla cieca* entrambe le differenze si riferiscono all'ultimo nascondiglio individuato prima di entrare in quello stato.

Sotto fuoco Questo ingresso rappresenta il fatto che il giocatore sta tenendo sotto tiro Spykee. Per maggiore chiarezza questa informazione viene evidenziata anche dall'accensione del led sopra la testa di Spykee.

Rilevato oggetto Questo ingresso indica se i sonar hanno rilevato che abbiamo superato l'oggetto che vogliamo utilizzare come nascondiglio. Siccome Mr. Brian prende decisioni solo alla base degli ingressi che riceve in quel momento, questo tipo di ingressi serve a mantenere la memoria degli eventi principali che sono stati effettuati.

Stato Questo ingresso indica lo stato in cui si trova Spykee. Come l'ingresso *Rilevato oggetto* viene riportato lo stato che Mr. Brian ha stabilito all'esecuzione precedente.

Oggetto perso Questo ingresso assume valore positivo se l'ultima iterazione di Mr. Brian ha stabilito che Spykee ha finito di costeggiare l'oggetto e l'ha superato.

Sulla base di queste informazioni Mr. Brian associa ai predicati fuzzy il loro grado di verità attivando i comportamenti adeguati alla situazione in cui si trova. Per spiegare l'atto di nascondersi mostro in dettaglio tutti i comportamenti utilizzati, nell'ordine in cui vengono eseguiti nella maggior parte dei casi.

4.2.1 Cercare un nuovo ostacolo

Nelle immagini catturate da Spykee non è scontato che siano presenti oggetti da sfruttare come nascondigli. Per questo motivo occorre implementare un comportamento che permetta a Spykee di muoversi fino a quando riesce a individuare un nascondiglio.

Questo comportamento propone a Spykee di girare su se stesso fino a quando riconosce un nascondiglio. La direzione predefinita di movimento è verso sinistra. Se a sinistra il sonar rileva un ostacolo gira verso destra. Se anche a destra è presente un ostacolo vicino propone a Spykee di muoversi in avanti. Se anche davanti c'è un ostacolo propone di muoversi all'indietro. Se è circondato da ostacoli non propone alcun movimento.

Vogliamo precisare che se le condizioni di attivazione rimangono immutate, un comportamento viene continuamente eseguito. Se noi prendiamo come esempio la situazione in cui Spykee ha rilevato due ostacoli, uno a destra e uno a sinistra, e la camera non ha ancora rilevato nessun ostacolo. All'inizio SPykee andrà avanti, quando i sonar si accorgeranno che a sinistra è scomparso l'ostacolo, quindi quando rivelerà una grande distanza, Spykee inizierà a girare verso sinistra; probabilmente a questo punto l'algoritmo di visione valuterà l'ostacolo percepito dai sonar come oggetto in cui nascondersi.

La condizione di attivazione di questo comportamento è la mancanza di nascondigli individuati dall'algoritmo di visione e lo stato di Spykee deve essere *Cerco un nuovo nascondiglio.*

4.2.2 Avvicinamento al nascondiglio

Quando l'algoritmo di visione fornisce un nascondiglio a Mr. Brian, incomincia la fase di avvicinamento. Se siamo nello stato *Cerco un nuovo nascondiglio* proponiamo di passare allo stato *Avvicinamento al nascondiglio* e le velocità che deve assumere Spykee per intercettare lo spigolo del nascondiglio.

Se siamo nello stato *Avvicinamento al nascondiglio*, stabiliamo le velocità che deve assumere Spykee per raggiungere lo spigolo dell'oggetto. Questo comportamento si occupa di selezionare se l'angolo dell'ostacolo attuale si riferisce allo spigolo sinostro o destro del nascondiglio. Quando il sonar laterale in accordo con il lato dell'oggetto rileva un'elevata riduzione della distanza e la distanza attuale è esigua, deduce che Spykee ha intercettato l'oggetto e propone di cambiare lo stato in *Costeggiamento nascondiglio* e notifica la rivelazione dell'oggetto.

Anche se l'algoritmo di visione è greedy, ovvero restituisce il nascondiglio migliore senza tenere conto dei nascondigli migliori precedenti, la fase di avvicinamento è abbastanza coerente. Questo perchè se Spykee si muove verso l'ostacolo migliore, esso sarà sempre più vicino e più grande, diventando quindi sempre con più alta probabilità il migliore rispetto agli altri oggetti.

Nella fase di avvicinamento dobbiamo stare attenti ad avvicinarsi allo stesso nascondiglio: quando ci stiamo avvicinando allo spigolo dell'oggetto, spesso capita che la camera di Spykee non riesce più ad inquadrare l'oggetto a cui siamo interessati. Quindi siccome i sonar non hanno ancora rilevato l'oggetto, cerchiamo di avvicinarci a un altro nascondiglio che l'algoritmo di visione ci fornisce. Potenzialmente potremmo quindi rimbalzare da un nascondiglio all'altro senza mai nasconderci completamente.

Per questo motivo occorre riconoscere quando l'algoritmo di visione cambia nascondiglio. Mr. Brian decide che il nascondiglio è cambiato in due casi: sia se la differenza dell'angolo che della dimensione sono cambiati leggermente, oppure se almeno una delle due differenze è cambiata drasticamente. Mentre ci si avvicina la dimensione e l'angolo del nascondiglio variano, ma siccome l'algoritmo di visione è molto veloce, le differenze dovute al movimento sono trascurabili rispetto al caso in cui cambia l'oggetto in cui nascondersi.

Questo problema limita la nostra capacità di curvare durante l'avvicinamento, perché Spykee ha un'alta velocità rotazionale rispetto a quella tangenziale, quindi se gli facciamo effettuare curve troppo accentuate corriamo il rischio che interpreti il brusco movimento come cambiamento del nascondiglio. Al contrario se rilassiamo i vincoli che determinano il

cambiamento del nascondiglio rischiamo di rendere inefficace la nostra applicazione. In questi comportamenti abbiamo tarato i parametri in modo da avere la maggior libertà di sterzo possibile, senza influenzare l'efficacia dell'applicazione.

Nel caso Mr. Brian decide che l'ostacolo è cambiato viene proposto lo stato *Avvicinamento al nascondiglio alla cieca*.

Questo comportamento si attiva se siamo nello stato *Avvicinamento al nascondiglio* o *Cerco un nuovo nascondiglio*.

4.2.3 Avvicinamento al nascondiglio alla cieca

Questo comportamento si occupa di risolvere il problema della fase dell'avvicinamento nell'ultimo miglio, ovvero quando la telecamera non inquadra più l'ostacolo a cui ci stavamo avvicinando e i sonar non l'hanno ancora rilevato. Considerando che quando sopraggiunge questo problema, Spykee si trova già posizionato correttamente e a pochi centimetri dal rilevare coi sonar il nascondiglio, abbiamo pensato di aggiungere questo comportamento che propone a Spykee di andare avanti con la speranza di rilevare con i sonar lo spigolo del nascondiglio.

Raramente capita che grazie a un'illuminazione particolare, in un fotogramma l'algoritmo di visione riconosca un altro nascondiglio come migliore (tipicamente un bagliore, non un vero oggetto), per poi ritrovare nel fotogramma successivo il vecchio oggetto a cui ci stavamo avvicinando. Per questo motivo se le differenze dei nascondigli rientrano a valori nella norma, questo comportamento propone di ritornare allo stato *Avvicinamento al nascondiglio*. Vogliamo ricordare al lettore che durante lo stato *Avvicinamento al nascondiglio alla cieca* le differenze si calcolano rispetto all'ultimo nascondiglio valido.

Se i sonar rilevano l'inizio del nascondiglio, il comportamento propone di passare allo stato *Costeggiamento nascondiglio*.

Questo comportamento è sempre andato a buon fine, tuttavia data la sua natura imprevedibile, abbiamo ritenuto utile fissare un numero limite di iterazioni che può effettuare Mr. Brian in questo stato. Nel caso si superi questo limite si forza lo stato in *Cerco un nuovo nascondiglio*. Questa forzatura avviene all'esterno del contesto di Mr. brian.

Questo comportamento viene eseguito solo se il nascondiglio è cambiato oppure lo stato è *Avvicinamento al nascondiglio alla cieca* e i sonar non devono aver rilevato il nascondiglio.

4.2.4 Costeggiamento nascondiglio

In questa fase della manovra le informazioni della camera sono inutili. Da questa fase fino a quando ci nascondiamo sono i sonar gli unici strumenti che abbiamo per muoverci. Il compito di questo comportamento infatti è quello di seguire l'andamento dell'oggetto.

Per riuscire a svolgere questo compito, sono state pensate ed implementate veramente molte soluzioni. Il motivo di questa difficoltà è che i valori che otteniamo dai sonar sono molto diversi da quelli che ci aspettavamo e la velocità di lettura è sensibilmente più lenta di quella prevista.

La soluzione che ci ha garantito dei risultati è quella di far effettuare a Spykee rapide sterzate verso il nascondiglio ogni volta che ci sono delle differenze di misure. L'intensità delle sterzate varia a seconda di quanto ci stiamo allontanando o avvicinando all'oggetto, ma in ogni caso la direzione rimane verso il nascondiglio.

Vogliamo ricordare che un sonar viene letto poche volte al secondo, quindi il risultato che otteniamo è di mantenere la direzione di Spykee, ma sporadicamente gli facciamo eseguire una breve sterzata verso l'ostacolo. Ricordiamo che questo comportamento non ha il compito di evitare di impattare i possibili oggetti.

Se i sonar rilevano un grande aumento della distanza oppure che la distanza è considerevole, significa che abbiamo superato la fine dell'ostacolo. Per questo motivo questo comportamento segnala che il nascondiglio è stato perso e propone di passare allo stato *Aggira il nascondiglio - fase 1*.

Se non è possibile aggirare l'ostacolo perché il sonar a nord rileva che c'è un ostacolo vicino, propone di passare allo stato *Nascosto*.

Questo comportamento viene eseguito solo se i sonar hanno rilevato l'inizio del nascondiglio oppure se siamo nello stato *Costeggiamento nascondiglio*.

4.2.5 Aggirare l'ostacolo

Questo comportamento ha l'obiettivo di aggirare l'ostacolo e posizionare Spykee dietro lo stesso. Questa manovra si divide in tre fasi. Le prime due si occupano di allinearsi verso il lato posteriore dell'oggetto, la terza di andare avanti fino a posizionarsi dietro.

La prima fase propone di far girare Spykee attorno al proprio asse centrale nella direzione del nascondiglio, fino a quando i sonar rilevano ancora l'oggetto, a questo punto propone di passare allo stato *Aggira il nascondiglio - fase 2*.

La seconda fase propone di continuare lo stesso moto iniziato nella fase uno. Quando i sonar rilevano che hanno perso di nuovo il nascondiglio il

comportamento propone di passare allo stato *Aggira il nascondiglio - fase 3*.

Giunti alla terza fase siamo allineati con il lato posteriore del nascondiglio, quindi il comportamento propone di andare dritti in avanti. Quando i sonar rilevano un'ulteriore volta il nascondiglio, il comportamento propone di passare allo stato *Nascosto*.

Per alcune traiettorie i sonar non riescono a rilevare l'ostacolo durante la fase 1, con il risultato di continuare a far girare Spykee su se stesso. Per questo motivo abbiamo ritenuto necessario effettuare un controllo sul numero di iterazioni in questa fase. In maniera analoga al comportamento *Avvicinamento al nascondiglio alla cieca* la forzatura dello stato avviene fuori dal contesto di Mr. Brian.

Questo comportamento viene attivato solo se si è perso l'oggetto oppure si è nei tre stati *Aggira il nascondiglio*.

4.2.6 Resta fermo

Questo comportamento non ha un ruolo attivo come gli altri, infatti impone a Spykee di restare immobile; è possibile immaginarlo come conclusione dell'atto di nascondersi. Questo comportamento viene eseguito in due occasioni: quando lo stato è *Nascosto* oppure se lo stato è *Aggira il nascondiglio - fase 3* e l'oggetto è stato nuovamente rilevato dei sonar.

Questo comportamento perturba questa situazione di immobilità solo se Spykee rileva che è sotto fuoco da parte del giocatore. Intuitivamente fincheè Spykee è nascosto deve restare fermo, appena il giocatore l'individua deve cercare di fuggire.Per questo motivo e i sonar rilevano che lo spazio è sufficiente, Spykee cerca di sfruttare lo stesso nascondiglio, proponendo di tornare allo stato *Costeggiamento nascondiglio* e notificando la rilevazione del nascondiglio. Altrimenti se lo spazio difronte a Spykee non consente quest'ultima manovra, questo comportamento riporta tutto come all'inizio della partita, ripristinando lo stato *Cerco un nuovo nascondiglio*.

Quando si trova sotto la minaccia del giocatore, Spykee può recarsi in un altro nascondiglio, oppure tentare di girare attorno all'oggetto in cui si era già nascosto.

4.2.7 Evita ostacoli

Questo comportamento ha la responsabilità di evitare che Spykee impatti con qualche ostacolo. A differenza degli altri comportamenti, questo è gerarchicamente superiore di livello. Per questo motivo possiede come dati in

ingresso tutte le azioni proposte dagli altri livelli, oltre ovviamente ai dati in ingresso a Mr. Brian.

Considerando che lo chassis di Spykee è quadrato, se gira su se stesso non può urtare contro alcun ostacolo. L'unico modo che ha di urtare qualcosa è aggiungendo una componente tangenziale al movimento rotazionale.

Questo comportamento si attiva se almeno un sonar individua che é troppo vicino ad un oggetto. Il fatto che viene attivato questo comportamento non implica tuttavia che debba intervenire. Abbiamo ideato questo comportamento per intervenire solo se il suo non agire comporta l'impatto di Spykee con un oggetto.

È possibile evidenziare due modi molto differenti in cui Spyke può impattare con un oggetto: urtare l'oggetto con la parte frontale (o posteriore) oppure urtare l'oggetto con la parte laterale.

Nel primo caso significa che Spykee si sta avvicinando ad un oggetto abbastanza grande da essere rilevato dal sonar, ma non abbastanza grande da essere sfruttato come nascondiglio. La strategia migliore sarebbe modificare la traiettoria per evitarlo e in seguito procedere nuovamente alla manovra che si stava effettuando, purtroppo la mancanza di odometria non permette di riprendere la traiettoria precedente; per questo motivo evitiamo l'ostacolo in accordo con la direzione che ci ha fornito il comportamento di livello inferiore e fissiamo lo stato di SPykee in *Cerco un nuovo nascondiglio* sperando di individuare nuovamente l'ostacolo precedente.

Nel secondo caso significa che abbiamo un oggetto ad un nostro lato e la nostra traiettoria porterebbe il lato di Spykee a urtare contro l'oggetto. Questo caso si verifica sopratutto quando tentiamo di costeggiare il nascondiglio. Per evitare l'impatto agiamo come per il costeggiamento, ovvero effettuiamo una breve ma intensa virata nel senso opposto rispetto all'ostacolo, questa soluzione evita di interrompere la manovra proposta dai livelli inferiori ma è sufficiente per evitare che Spykee urti contro l'oggetto. In quest'ultimo caso non alteriamo lo stato attuale di Spykee.

Come avete potuto notare leggendo i comportamenti, se Spykee viene minacciato mentre si sta avvicinando al nascondiglio non effettua nessuna manovra per disimpegnarsi. Questa scelta è legata a due motivi. Il primo motivo è che l'azione di nascondersi è molto critica anche in condizioni normali, se alteriamo il percorso, non avendo a disposizione odometria, non riusciamo a portare a buon fine l'azione di nascondersi. Il secondo motivo è che implicitamente mentre l'algoritmo di visione calcola il punteggio di

rank dei nascondigli tiene già conto del livello di minaccia; quindi avremo che Spykee normalmente si nasconde dietro ad oggetti grandi, se minacciato cercarà di nascondersi dietro all'oggetto più vicino che ha rilevato l'algoritmo di visione.

Nell'appendice A è presente la lista di tutti i predicati fuzzy che servono per dedurre tutte le informazioni di alto livello che abbiamo utilizzato, come per esempio il rilevare il nascondiglio utilizzando i sonar. Per fornire la visione completa della logica utilizzata includiamo anche la fuzzyficazione degli ingressi.

Per dedurre se Spykee è sotto la minaccia del giocatore non abbiamo utilizzato Mr. Brian, ma come leggerete nel Capitolo 5 un altro agente ha questo incarico.

Capitolo 5

Architettura del sistema

"Qualunque bug sufficientemente avanzato è indistinguibile da una caratteristica del software"

Rich Kulawiec

In questo capitolo descriviamo dettagliatamente lo sviluppo pratico della nostra applicazione. Il linguaggio che abbiamo utilizzato nell'implementazione della soluzione del problema presentato in questa tesi è C/C++. Quando abbiamo progettato la struttura dell'implementazione, abbiamo cercato di mantenere la modularità che contraddistingue MRT. Le librerie necessarie agli agenti sono infatti contenute in moduli separati dal codice degli esperti; per esempio la libreria che permette di utilizzare Mr. Brian è contenuta nei moduli brian e fuzzy.

Il nostro spazio di lavoro contiene solo cartelle e un makefile. Il makefile permette la compilazione della nostra applicazione. L'Appendice B contiene una guida per la compilazione dei nostri sorgenti. Le cartella bin, conterrà i file binari della nostra applicazione, i file di configurazione e i file di log.

La cartella robot contiene il codice sorgente di tutti gli esperti che abbiamo utilizzato nella nostra applicazione, possiamo affermare che rappresenta il cuore della nostra applicazione. Le altre cartelle contengono moduli e librerie utilizzate dagli esperti. La libreria essenziale che costituisce lo scheletro della nostra applicazione è contenuta nel modulo DCDT, come descritto nel Capitolo 2 incorpora i meccanismi di comunicazione fra esperti; i moduli Middleware e Xware ne aumentano le potenzialità. Mr. Brian è contenuto nei moduli brian e fuzzy. La libreria che si occupa di comunicare con il Wiimote è contenuta nel modulo wiiuse. Per poter comunicare con Spykee è necessaria la libreria contenuta nel modulo spykee. Per poter leggere le misure dei sonar è necessaria la libreria contenuta nel modulo sonar.

Nel modulo shared sono contenuti tutte le informazioni che sono condivise dai moduli, come le costanti di gioco.

Per poter essere eseguita la nostra applicazione ha bisogno di ulteriori librerie non incluse nel nostro spazio di lavoro: OpenCv e libXml2; il motivo di tale scelta è che libXml2 risulta essere già preinstallata nelle ultime distribuzioni GNU/Linux, mentre la libreria OpenCv offre una libertà di personalizzazione talmente elevata, che abbiamo ritenuto migliore l'idea di lasciare all'utente la scelta di compilarla da sorgenti oppure ottenerne una versione già compilata dai repository ufficiali.

In questo capitolo illustreremo solo i moduli che abbiamo realizzato e gli esperti che abbiamo utilizzato per risolvere il problema presentato in questa tesi. Escludendo l'esperto dei sonar, gli altri agenti li abbiamo riscritti da zero perchè utilizzando Xware è necessario cambiare anche la struttura oltre che ai contenuti. I moduli dcdt, middleware, brian, fuzzy e sonar fanno parte della raccolta di MRT; i moduli wiiuse, opencv, libxml2 provengono da progetti di terze parti.

5.1 Xware

Come accennato nel capitolo 2, MRT utilizza un sistema public-subscriber per gestire lo scambio di messaggi fra esperti. Il modulo base che si occupa di questa funzione è dcdt. Questo modulo invia ai richiedenti del messaggio le informazioni racchiuse nella zona di memoria indicata dal mittente. In questo modo è possibile trasferire qualsiasi tipo di dato. DCDT non prevede politiche di serializzazione, per questo motivo è l'esperto ad assicurarsi che il messaggio che vogliamo inviare sia memorizzato in un'area contigua di memoria.

Con l'obiettivo di migliorare questo aspetto di MRT hanno sviluppato Middleware; questo modulo infatti estende DCDT e offre funzioni di più alto livello. Middleware permette agli esperti di scambiarsi messaggi che contengono solamente del testo senza lasciare all'agente l'onere di gestire direttaemnte la memoria. La scelta di scambiarsi solo del testo rende molto più chiaro osservare che informazioni si scambiano gli esperti, anche a costo di aumentare le dimensioni del messaggio.

Le applicazioni che hanno utilizzato Middleware hanno associato ad ogni esperto uno scanner lessicale in grado di interpretare il contenuto dei messaggi che ricevono. Per questo motivo abbiamo pensato di comporre invece messaggi in xml e utilizzare la libreria di sistema libXml2 per esplorare il contenuto dei messaggi, evitando di utilizzare tali scanner, alleggerendo la nostra applicazione.

Con questo obiettivo in mente abbiamo scritto il modulo xware. Questo modulo estende Middleware e fornisce delle funzioni di più alto livello per scambiarsi le informazioni. Considerando che i tipi di informazioni che gli esperti si devono scambiare sono note a priori, come per esempio il nascondiglio individuato dalla visione o le distanze sonar, abbiamo scritto delle funzioni che a partire dall'informazione che dobbiamo inviare, compongono da sole il messaggio xml appropriato. Analogamente questo modulo contiene le funzioni necessarie per interpretare il messaggio xml in arrivo e restituiscono direttamente le informazioni necessarie.

Questo modulo nella maggior parte dei casi rende del tutto trasparente rispetto agli esperti la composizione del messaggio. L'unico onere a carico dell'agente è infatti quello di utilizzare la funzione appropriata per le informazioni che vuole inviare, ed è Xware a comporre e inviare il messaggio. Un discorso analogo può essere fatto anche per quanto riguarda la ricezione; in questo ultimo caso infatti Xware capisce dal tag principale del messaggio ricevuto che tipo di informazioni contiene e ritorna l'oggetto appropriato contenente le informazioni ricevute.

Xware utilizza Middleware per inviare e ricevere i messaggi, quindi è possibile comporre manualmente i messaggi da inviare oppure interpretarli manualmente in ricezione senza alcuna controindicazione. Utilizzando il linguaggio xml è possibile validare la sintassi del messaggio in arrivo attraverso l'uso del dtd (Document Type Definition); tuttavia considerando che è il modulo stesso a comporre i messaggi, abbiamo preferito evitare di sprecare risorse verificando il messaggio che ha scritto il modulo stesso. Se si dovesse corrompere il messaggio oppure se si dovesse verificare un errore invece dell'oggetto contente l'informazione viene restituito NULL.

I messaggi che si scambiano gli esperti sono inseriti in un tag che contiene le informazioni necessarie a descrivere la natura del messaggio. Ogni messaggio infatti è composto come:

```
<?xml version="1.0" ?>
<message object="" realm="" timestamp="" sender="">

    ... corpo del messaggio ...

</message>
```

Il campo "object" identifica l'oggetto del messaggio, ovvero la natura del contenuto del messaggio; il nostro modulo utilizza questo campo per decidere come interpretare il contenuto del messaggio. Il campo "realm"

identifica l'Agorà a cui appartiene il mittente. Il campo "sender" identifica il mittente del messaggio. Il campo "timestamp" contiene la data temporale della composizione del messaggio. Gli ultimi tre campi vengono riempiti dal mittente, ma attualmente non vengono utilizzati in questa applicazione.

Il modulo Xware offre anche la possibilità di utilizzare un file di log. Utilizzando il nome dell'esperto, Xware crea un file di log per ogni esperto. Xware mette a disposizione lo stream di scrittura al file di log e attraverso l'uso di tre funzioni permette di mantenere una formattazione coerente all'interno del file di log per tre livelli di logging; è possibile infatti inserire una riga di log come *informazione*, come *avvertimento* o come *errore*. È Xware che si occupa di aprire e chiudere il file di log.

5.2 Spykee

Come accennato nel Capitolo 2, la Meccano mette a disposizione una sua applicazione per poter comandare Spykee. Questa applicazione risulta inservibile perché non prevede alcun meccanismo per essere interfacciata con software di terze parti, come la nostra applicazione; non sono presenti infatti né interfacce da poter includere nel nostro codice sorgente né applicazioni a riga di comando.

Per questo motivo le persone che hanno precedentemente lavorato con Spykee hanno effettuato del reverse-engeneering sui pacchetti che l'applicazione della Meccano si scambia con Spykee e hanno realizzato una libreria per riuscire a pilotare Spykee.

Siccome la funzione che ottiene le immagini della camera di Spykee non era adatta ai nostri scopi, l'abbiamo riscritta. Considerando che alcune funzioni non le utilizzavamo le abbiamo escluse nella nuova libreria che abbiamo adottato. In particolare abbiamo tenuto la funzione che permette a Spykee di sganciarsi dalla piattaforma di caricamento delle batterie, la funzione che accende la camera di Spykee e la funzione che permette a Spykee di muoversi.

La funzione che fa muovere Spykee necessita di due parametri: la velocità del cingolo sinistro e la velocitá del cingolo destro. Le velocità dei cingoli sono espresse mediante interi e variano da -90 a +90. Se la velocià è positiva, i cingoli si muovono in avanti, se è negativa si muovono all'incontrario.

Prima che Spykee inizi ad inviare le immagini che cattura la sua camera è necessario che sia sganciato dalla piattaforma di caricamento e che gli sia stato comandato di accendere la camera. Una volta che si verificano questi due eventi, Spykee inizia ad inviare le immagini che cattura la sua camera. Un'immagine solitamente occupa più di un pacchetto e Spykee non invia solo immagini ma anche altri pacchetti di controllo. Quando chiamiamo la

nostra funzione per catturare un'immagine, si mette ad analizzare tutti i pacchetti che invia Spykee. Quando rileva che un pacchetto contiene l'immagine inizia a salvare il contenuto di tutti i pacchetti ricevuti (se sono più di uno) fino a quando raggiunge la fine dell'immagine. Di seguito mostriamo in pseudocodice la funzione che abbiamo implementato.

```
stato ← nonTrovata
while stato ≠ immagineFinita do
  leggi pacchetto
  if pacchetto contiene una nuova immagine then
    inizia acquisire immagine
    if immagine finita then
      stato ← immagineFinita
    else
      stato ← acquisizioneInCorso
    end if
  end if
  if stato = acquisizioneInCorso then
    continua ad acquisire
    if immagine finita then
      stato ← immagineFinita
    end if
  end if
end while
restituisci immagine acquisita
```

Ricordiamo che per comunicare con Spykee è necessario instaurare un canale di comunicazione wireless, quindi occorre connetterci alla rete che crea quando viene acceso. Inoltre prima di poter comandare Spykee occorre eseguire l'accesso fornendo nome utente e password.

5.3 Agenti del sistema

La nostra applicazione sfrutta la cooperazione fra sei esperti per risolvere il problema proposto in questa tesi. Abbiamo creato un esperto che si occupa del problema della visione. Abbiamo creato un esperto che si occupa di comandare i motori di Spykee per farlo muovere. Abbiamo realizzato un esperto che si occupa di comunicare con il Wiimote. Abbiamo affidato a un esperto l'uso di Mr. Brian per prendere le decisioni. Abbiamo affidato a un esperto la lettura dei sonar e la gestione dei led aggiunti a Spykee.L'ultimo esperto che abbiamo creato si occupa di gestire le meccaniche del gioco.

In questa sezione descriveremo nel dettaglio come abbiamo implementato ciascun esperto.

Come abbiamo precedentemente scritto, tutti gli esperti risiedono nella cartella robot. In questa cartella è presente anche il cuore dell'applicazione, ovvero il file che contiene il "main" dell'applicazione: il kernel. In questa sezione spiegheremo anche l'implementazione di questo file per avere una visione più chiara di come funziona la nostra applicazione.

5.3.1 Il kernel

Il kernel non è un esperto ma è il punto di partenza della nostra applicazione. In questo file infatti viene creata l'Agorá che conterrà tutti gli esperti che abbiamo utilizzato. Per la nostra applicazione abbiamo configurato l'Agorà per funzionare in locale, considerando che tutti gli esperti sono processi che risiedono nel nostro computer.

Dopo essersi avviato il kernel inizializza il parser Xml, si connette alla rete di Spykee fornendo nome utente e passwor e crea l'Agorà. Dopo questa prima fase crea tutti gli agenti utilizzati, fornendogli se necessario il riferimento per poter comunicare con Spykee. Dopo aver creato gli esperti li aggiunge all'Agorà e li attiva. In questo momento, gli esperti si sono inizializzati ma non hanno ancora agito.

L'ultimo passo è avviare gli esperti dando inizio alla nostra applicazione. Questo processo viene effettuato chiamando la fatidica istruzione "dispatch-¿LetsWork();"; questa chiamata è bloccante. Quando il gioco finisce, ovvero quando un esperto spegne l'Agorà, il flusso del programma ritorna al kernel che libera la memoria utilizzata e termina l'esecuzione della nostra applicazione.

5.3.2 Wiimote Expert

Questo esperto si occupa di gestire la comunicazione con il Wimmote. Il Wiimote expert è uno dei due agenti che ha la facoltà di far terminare il gioco. Durante la sua inizializzazione cerca dei Wiimote che si sono messi in modalità esplorazione. Se non trova alcun Wiimote registra l'errore sul file di log e spegne l'Agorà, questo comportamento è dettato dalla mancanza di controllo sulla nostra applicazione. Se l'esperto trova una sola Wiimote procede alla connessione. Se trova più di una Wiimote avvisa nel file di log questa situazione potenzialmente dannosa ma continua comunque a connettersi a tutte le Wiimote trovate e non spegne l'Agorà. Se l'esperto si connette ad almeno un Wiimote fa leggermente vibrare i controller a cui si è connesso. Inoltre in questa fase di inizializzazione abilita gli accelerometri

e la camera del Wiimote. Anche se in questa applicazione non vengono utilizzati direttamente i valori degli accelerometri, nella documentazione della libreria si afferma che la camera del Wiimote individua con più precisione le fonti luminose se sono attivi gli accelerometri. Durante l'inizializzazione l'esperto comunica all'Agorà che è interessato a ricevere i messaggi contenenti lo status dei led, ovvero quali led devono essere accesi.

Il corpo di questo esperto è suddivso in due parti principali. La prima parte chiede al dispatcher l'ultimo messaggio riguardante lo status dei led. Se è presente un nuovo messaggio illumina il numero di led sul controller in accordo con il messaggio arrivato. La seconda parte chiede alla libreria Wiiuse se il Wiimote ha generato degli eventi, come la pressione di uno o più bottoni. Anche il numero e la posizione di sorgenti luminose che la camera del Wiimote individua è considerato un evento.

Possiamo dividere gli eventi che genera il Wiimote in tre grandi tipologie: la pressione di almeno un bottone, il numero e la posizione delle sorgenti luminose e lo status del Wiimote. La gestione degli eventi comprende solamente i primi due casi, lo status del Wiimote viene registrato sul file di log ma non viene utilizzato per altri scopi. Lo status del Wiimote infatti ci indica quali componenti del controller sono attualmente in uso e il livello delle batterie.

Se l'evento generato dal Wiimote é la pressione di almeno un tasto, inviamo un messaggio contenente tutti i tasti che sono stati premuti, rilasciati o mantenuti pigiati. Se l'evento generato riguarda le fonti luminose individuate dal Wiimote, è questo l'esperto a controllare se Spykee si trova sotto fuoco o meno, e ogni volta che il giocatore inizia o finisce di tenere sotto tiro Spykee, l'esperto invia un messaggio per notificare l'evento.

L'esperto controlla gli eventi di tutte le Wiimote a cui si è connesso. In questo modo abbiamo un controllo sulla nostra applicazione anche se non ne abbiamo l'esclusiva. Se un Wiimote si disconnette l'esperto spegne l'Agorà, questo perchè non avremo più modo per interagire con l'applicazione. Una Wiimote si può disconnettere premendo l'apposito tasto, oppure se si scaricano le batterie.

5.3.3 Vision Expert

Questo esperto ha il compito di individuare e valutare i nascondigli trovati. L'agente di visione necessita del riferimento a Spykee per ottenere le immagini della sua camera. L'inizializzazione di questo esperto attiva la camera di Spykee e richiede al dispatcher i messaggi che indicano il livello di minaccia, ovvero se Spykee è sotto tiro da parte dall'avversario o meno. Il motivo della

richiesta di questo tio di messaggi risiede nel fatto che ci serve questa informazione per calcolare il rank dei nascondigli. Originariamente questo era il compito dell'esperto del gioco, tuttavia per diminuire il flusso di messaggi scambiati e velocizzare l'operazione, abbiamo deciso di assegnare all'esperto della visione anche questo compito.

Il corpo dell'esperto è diviso in tre parti principali. Nella prima l'agente controlla ed eventualmente aggiorna il livello di minaccia di Spykee. Nella seconda parte acquisisce l'immagine della camera di Spykee e utilizzando l'algoritmo di visione spiegato nel Capitolo 4 trova la lista dei nascondigli, la dimensione dell'oggetto più grande e la distanza dell'oggetto più lontano. L'ultima parte calcola il rank di ogni nascondiglio dell'immagine e ne trova il migliore. Una volta effettuati questi passi invia l'ostacolo migliore individuato nell'immagine. Il procedimento per valutare i nascondigli è spiegato dettagliatamente nel Capitolo 4.

5.3.4 Motor Expert

Questo esperto ha il compito di inviare a Spykee le velocitá che devono assumere i suoi cingoli. Nell'inizializzazione informa l'Agorà che é interessato a ricevere i messaggi che riguardano la velocità che Spykee deve assumere e lo stato del gioco, ovvero se si sta giocando oppure si è in pausa. Questo esperto necessita ovviamente della possibilitá di comunicare con Spykee.

La libreria che utilizziamo per comunicare con Spykee mette a disposizione un metodo per inviare a Spykee la velocitá a cui deve muoversi. La gestione dell'accelerazione e della decelerazione viene gestita internamente da Spykee, noi possiamo solamente imporgli la velocità da raggiungere. Questo metodo utilizza la velocità espressa in velocità del cingolo sinistro e velocità del cingolo destro e prevede valori interi, compresi fra -90 e +90. Valori positivi fanno muovere Spykee in avanti, valori negativi fanno muovere Spykee indietro.

All'interno della nostra applicazione noi ragioniamo invece in termini di velocità tangenziale e velocità rotazionale. Utilizzando l'Equazione 5.1 e l'Equazione 5.2 è possibile passare da una rappresentazione della velocità all'altra.

Spykee non mantiene la velocitá che gli inviamo, infatti appena raggiunge tale velocitá inizia a decelerare fino a fermarsi. Occorre quindi continuare a inviargli la velocità desiderata, anche se essa non cambia.

$$\text{velocità cingolo destro} = \text{velocità tangenziale} - \text{velocità rotazionale} \tag{5.1}$$

velocità cingolo sinistro = velocità tangenziale + velocità rotazionale (5.2)

Questo è l'unico esperto in cui non abbiamo utilizzato Xware per lo scambio di messaggi, ma abbiamo interpretato manualmente i messaggi in arrivo. Il motivo di questa scelta risiede nella criticitá della funzione che svolge questo agente. Utilizzando Xware non è possibile discriminare il caso in cui non arriva un messaggio, dal caso in cui si rilevano errori in lettura. Considerando che per far muovere Spykee occorre mantenere la velocità desiderata, abbiamo preferito implementare un sistema per rallentare Spykee nel caso arrivi un messaggio corrotto. A ogni messaggio corrotto che arriva, questo agente dimezza le velocitá che sta utilizzando. In questo modo se si dovesse corrompere un solo messaggio l'effetto sarebbe solamente un rallentamento, mentre se per qualche ragione dovessero arrivare una serie di messaggi corrotti, le velocità tenderebbero ad annullarsi rapidamente, facendo fermare Spykee.

Quando il giocatore mette in pausa il gioco, l'esperto riceve il messaggio e ferma immediatamente Spykee. Abbiamo accuratamente scelto il periodo attivazione di questo esperto in modo da rendere il movimento di Spykee fluido.

5.3.5 Mr. Brian Expert

Questo agente ha il compito di decidere e inviare la velocità che deve assumere Spykee. Per svolgere questo compito si affida a Mr. Brian. L'inizializzazione dell'esperto crea un'istanza di Mr. Brian indicandogli il percorso ai file di configurazione e notifica all'Agorà che è interessato a ricevere i messaggi che contengono il livello di minaccia del giocatore, il nascondiglio individuato da vision, le distanze dei sonar e lo status del gioco ovvero se il giocatore ha messo il gioco in pausa o meno.

Il corpo dell'esperto è l'implementazione fedele della soluzione logica descritta nel Capitolo 4. Se il giocatore mette in pausa il gioco, non viene chiesto l'intervento di Mr. Brian. Abbiamo deciso di eseguire questo agente molto velocemente perché alcuni meccanismi di passaggio di stato possono richiedere più di un'esecuzione di Mr. Brian.

5.3.6 Game Expert

Questo agente si occupa di gestire tutte le meccaniche del gioco. L'inizializzazione chiede al dispatcher di ricevere messaggi che contengono gli eventi

generati dalla Wiimote e il livello di minaccia di Spyke, ovvero se il giocatore stia puntando o meno Spykee.

Quando l'Agorà chiede all'esperto di eseguire il suo dovere, questo agente svolge essenzialmente tre compiti. Il primo compito che svolge è controllare se Spykee o il giocatore hanno raggiunto i punti necessari a conseguire la vittoria. Nel caso qualcuno abbia vinto questo esperto notifica la vittoria nel file di log e si appresta a spegnere l'Agorà. Nel caso non ha vinto nessuno si preoccupa di gestire i timer per il punto di Spykee e il timer che indica il numero di scudi che il giocatore ha sovraccaricato, inviando un messaggio ogni qual volta cambi questo numero. Il secondo compito di questo esperto è quello di interpretare gli eventi generati dal Wiimote nel contesto di questo gioco. È infatti compito di questo esperto interpretare per esempio la pressione del tasto "A" come "mettere in pausa il gioco". Per ulteriori informazioni sull'interazione del giocatore con il gioco consultare l'Appendice B.

Questo gioco prevede la possibilità di controllare Spykee manualmente attraverso l'utilizzo del Wiimote. Il terzo ed ultimo compito di questo agente é infatti quello di inviare le velocità corrette all'esperto dei motori in accordo con le scelte effettuate dell'utente utilizzando il Wiimote.

5.3.7 Sonar Expert

Questo esperto si occupa di leggere le misure rilevate dai sonar e di illuminare i led presenti su Spykee. L'inizializzazione di questo esperto chiede all'Agorá di ricevere i messaggi che indicano il numero di scudi che il giocatore ha sovraccaricato e se il giocatore stia tenendo sotto tiro o meno il giocatore.

Il corpo di questo esperto è composto da due parti differenti. La prima parte si occupa di leggere ed inviare le misure dei sonar. La lettura dei sonar avviene ad opera di una scheda a bordo di Spykee che invia i dati alla nostra applicazione utilizzando gli Xbee. La scheda legge le misure fornite da una coppia di sonar, quindi la nostra applicazione ogni volta che arriva una coppia di valori dai sonar invia immediatamente tali informazioni senza aspettare che la scheda a bordo di Spykee legga tutti e quattro i sonar. Abbiamo operato tale scelta perché la scarsa frequenza con cui arrivano le letture dei sonar fa da collo di bottiglia nella nostra applicazione.

La seconda parte si occupa di inviare allo Xbee di Spykee iled che occorre illuminare e quali invece occorre siano spenti. A differenza di quanto succede con Spykee, occorre inviare le informazioni sui led solo quando si presenta qualche variazione.

Capitolo 6

Realizzazioni sperimentali e valutazione

"La disumanità del computer sta nel fatto che, una volta programmato e messo in funzione, si comporta in maniera perfettamente onesta."

Isaac Asimov

Giunti alla lettura di questo capitolo avrete chiaro come abbiamo realizzato la nostra applicazione e quali sono le ipotesi che devono verificarsi affinchè diventi efficace. Per questo motivo prima di poterla provare occorre individuare un luogo adatto. Tutte le prove che abbiamo fatto sono state effettuate all'interno del laboratorio. Considerando che lo sviluppo dell'applicazione è principalmente l'atto di nascondersi, abbiamo focalizzato la nostra attenzione su quell'aspetto. Quindi la maggioranza delle nostre prove consistevano nel far nascondere Spykee costruendogli attorno un contesto mirato a testare come si comportava la nostra applicazione in determinati frangenti.

Una volta realizzata la struttura dell'applicazione abbiamo implementato subito la possibilità di pilotare Spykee manualmente attraverso il Wiimote. In questo modo abbiamo provato a simulare manualmente le traiettorie che avrebbe dovuto compiere Spykee per verificare se le informazioni che poteva utilizzare coincidevano con quelle che avevamo previsto nell'analisi. È durante questa fase che abbiamo individuato alcuni problemi nell'algoritmo di visione che abbiamo provveduto poi a risolvere. Per esempio abbiamo aggiunto l'eliminazione del pavimento nell'algoritmo di visione perché quando il sole tramontava, l'illuminazione artificiale era tale che il pavimento grigio del laboratorio risultava essere sotto la soglia del filtro binario, in questo

modo l'algoritmo di visione interpretava come oggetto anche il pavimento, vanificando quindi ogni speranza di nascondersi che aveva Spykee.

Quando abbiamo reso operativo anche Mr. Brian abbiamo implementato le regole base che pensavamo fossero sufficienti a farlo nascondere. In questa fase ci siamo resi conto che l'analisi che avevamo fatto era leggermente ottimistica, infatti durante questi primi esperimenti ci siamo resi conto per esempio che la compensazione dell'angolo dell'ostacolo durante l'avvicinamento, che serve a far passare Spykee di fianco all'oggetto, non era costante; ci siamo resi conto anche che era necessario implementare il comportamento *Avvicinamento al nascondiglio alla cieca*.

Fino a questo punto avevamo sempre utilizzato un oggetto con cui Spykee doveva nascondersi di piccole dimensioni, perchè aveva caratteristiche tali da essere ottimamente individuato dal nostro algoritmo di visione. Tuttavia avendo dimensioni ridotte non avevamo mai testato effettivamente il costeggiamento del nascondiglio. A questo scopo abbiamo iniziato ad utilizzare un oggetto più largo che si prestava meglio ad essere costeggiato. Anche questo cambiamento ha portato alla luce alcuni difetti della nostra applicazione, per esempio ci siamo resi conto che il comportamento *Costeggiamento nascondiglio*, per come l'avevamo implementato, era del tutto inefficace. Dopo svariati tentativi abbiamo realizzato il comportamento che abbiamo descritto nel Capitolo 4.

Giunti alla versione finale della nostra applicazione, l'abbiamo testata per verificarne il comportamento generale. Da questi esperimenti abbiamo dedotto che una parte critica del nascondersi è quando Spykee è nello stato *Aggira il nascondiglio - fase 1* e *Aggira il nascondiglio - fase 2*; in questi due stati Spykee ruota su se stesso fino a quando rileva il nascondiglio, per poi perderlo nuovamente e quindi entrare nello stato *Aggira il nascondiglio - fase 3*. Se i sonar fornissero letture istantanee Spykee riuscirebbe a nascondersi indipendente dalla posizione in cui termina il costeggiamento. Purtroppo le letture che forniscono i sonar avvengono a una frequenza troppo lenta, per questo motivo qualche volta i sonar non fanno in tempo a rilevare l'oggetto, con la conseguenza che Spykee inizia a girare su se stesso. A questo punto può succedere che i sonar rilevino l'oggetto nei giri successivi, oppure può succedere che l'esperto che utilizza Mr. Brian si accorge che è in questa fase da troppo tempo e forzi lo stato di Spykee a *Cerco un nuovo nascondiglio*.

Un'altra fase critica è l'avvicinamento all'oggetto selezionato come nascondiglio. La nostra capacitá di effettuare manovre accentuate è limitata dalla scelta dei vincoli per determinare se la camera non riesce più ad inquadrare l'oggetto a cui ci stavamo avvicinando, inquadrandone quindi uno diverso. Mentre determinavamo quei vincoli abbiamo cercato di trovare il vincolo che rius-

cisse correttamente a rilevare quella situazione, dandoci allo stesso tempo la maggiore libertà di movimento possibile. Nonostante i nostri sforzi esistono tuttavia alcune traiettorie che Spykee non può effettuare per nascondersi, in particolare se si trova ad agire in ambienti ristretti.

Un altro motivo per cui l'avvicinamento all'ostacolo è un operazione critica, è data dal fatto che il nostro algoritmo di visione fornisce il punto obiettivo che Spykee deve raggiungere per riuscire a iniziare a costeggiare l'oggetto, tuttavia non è in grado di fornirci l'orientazione dell'oggetto rispetto a Spykee. La conseguenza diretta di questo fatto è che esistono alcune orientazioni particolari di oggetti per cui Spykee va ad impattare contro l'oggetto senza che la nostra applicazione riesca a prevenirlo. Quando capitano questi episodi, generalmente l'algoritmo di visione ci indica che il punto obbiettivo è a pochi gradi rispetto al centro dell'immagine (per esempio due gradi a destra), proponendo quindi all'esperto dei motori di proseguire dritti, e i sonar non rilevano alcun ostacolo perchè l'oggetto è al di fuori dei loro coni di rilevamento. Queste situazioni, derivate da particolari orientazioni degli oggetti non sono frequenti, ma possono accadere.

Se non si verificano le situazioni che ho appena descritto la nostra applicazione è correttamente in grado di far nascondere Spykee dietro ad un oggetto. La sperimentazione delle meccaniche di gioco non ha rilevato che la nostra applicazione ha particolari problemi, questo è dovuto al fatto che il regolamento del gioco che abbiamo scelto non risulta essere molto articolato.

L'implementazione che abbiamo utilizzato in accordo con la soluzione descritta in questa tesi, non prevede algoritmi con complessità di calcolo elevata, tuttavia al fine di modellizzare con più accuratezza l'evoluzione del mondo attorno a noi richiediamo spesso l'esecuzione di tali algoritmi. Per questo motivo la nostra applicazione richiede un uso considerevole della capacità di calcolo del processore. La nostra applicazione purtroppo non è possibile compilarla su piattaforme Windows, in quanto alcuni componenti che abbiamo utilizzato, come per esempio la gestione dell'orario di sistema, vengono implementate diversamente nelle due piattaforme.

6.1 Sviluppi futuri e conclusioni

Analizzando i risultati che abbiamo ottenuto, si osserva come riuscire ad ottenere l'orientazione di un oggetto possa aiutare a rendere più robusta la nostra applicazione. Anche se sembra un'operazione semplice, determinare l'orientazioni di alcuni oggetti può essere veramente difficile. L'applicazione del filtro binario unito alle operazioni di erosione e dilatazione che eseguiamo per pulire l'immagine hanno la tendenza a far diventare i contorni di un

oggetto un guazzabuglio di pixel. Per esempio ombre e giochi di luce possono rendere difficile anche per un essere umano riconoscere l'orientamento di un oggetto dopo la nostra elaborazione.

Un'altra strada che si può tentare di seguire senza modificare Spykee è quella di ricostruire la rototraslazione della sua camera utilizzando il flusso ottico. Utilizzando le OpenCv è possibile estrarre dei punti caratteristici di un'immagine e tentare di ritrovarli nell'immagine successiva. Se questi punti si riescono a trovare nella seconda immagine è possibile calcolare la rototraslazione fra le due immaigni. Ottenendo la rototraslazione sarebbe possibile effettuare un ricostruzione in tre dimensione partendo da una camera in movimento, migliorando quindi moltissimo l'algoritmo di visione. C'è da considerare anche che la camera ha una bassissima risoluzione e la compressione dell'immagine effettuata a bordo di Spykee peggiora la situazione; considerando anche che la ricostruzione usando una sola telecamera non fornisce ottimi risultati è interessante scoprire che risultati si otterrebbero.

Se si considera l'eventualità di poter modificare Spykee, ci sono due interventi che avrebbero la massima priorità. Si potrebbe rimpiazzare la camera di Spykee con due camere migliori, in modo da poter effettuare la ricostruzione in tre dimensioni dell'ambiente, migliorando notevolmente la capacità di Spykee di individuare i nascondigli. Un'altra modifica che comporterebbe moltissimi miglioramenti sarebbe fornire Spykee di odometria, in modo che Spykee si possa muovere nel mondo esterno con una maggiore consapevolezza.

Lo scopo della nostra tesi è quello di realizzare un'applicazione in grado di fornire ad un robot autonomo la capacità di giocare a nascondino sulla base di scarse informazioni sul mondo esterno. L'applicazione che abbiamo realizzato non è perfetta, ma considerando che le uniche informazioni che possediamo sono le immagini di una camera con la risoluzione di 320x240 pixel, già compresse all'interno del robot e la lettura di quattro sonar posizionati ai lati di Spykee, siamo soddisfati dei risultati che abbiamo ottenuto.

Bibliografia

[1] Adrian Kaehler Gary Bradski. *Learning OpenCV, Computer Vision with the OpenCV Library*. O'Reilly Media, 2008.

[2] Cristian Giussani. *DCDT - Guida per l'utente*, 2007.

[3] Luigi Malagò. *MRT User Manual*, 2007.

[4] Erik T. Ray. *Learning XML*. O'Reilly & Associates, Inc., 2003.

[5] sito del produttore. Xbee. `http://www.digi.com/products/wireless/point-multipoint/xbee-series1-module.jsp`.

[6] sito ufficiale. Spykee. `http://www.spykeeworld.com/spykee/UK/index.html`.

Appendice A

Predicati Fuzzy

In questa appendice abbiamo inserito i file di configurazione che abbiamo utilizzato in questa applicazione. Abbiamo deciso di omettere i predicati che riguardano i comportamenti di primo livello perchè occuperebbero un volume molto elevato e ne abbiamo parlato dettagliatamente nel Capitolo 4. L'unico comportamento che abbiamo deciso di inserire è quello che permette a Spykee di evitare gli ostacoli. Abbiamo inserito inoltre i predicati che indicano i concetti che abbiamo sempre espresso senza indicare mai come li otteniamo, per esempio come facciamo a rilevare con i sonar l'oggetto dietro al quale ci vogliamo nascondere. Abbiamo deciso di inserire anche i predicati riguardo la fuzzyficazione e defuzzyficazione dei dati per rendere maggiormente chiaro il passaggio nel mondo fuzzy.

A.1 Fuzzyficazione degli ingressi

In questa sezione includiamo i due file che descrivono a MR. Brian come fuzzyficare i dati in ingresso. In questi file di configurazione abbiamo utilizzato una terminologia leggermente differente rispetto a quella che abbiamo utilizzato nella tesi. L'ostacolo target è l'oggetto in cui ci vogliamo nascondere e il suffisso "precedente" indica che quel predicato si riferisce al valore ottenuto da Mr. Brian all'iterazione precedente.

Il primo file che includiamo descrive le forme degli insiemi da utilizzare per fuzzyficare gli ingressi.

```
(VELOCITAININGRESSO
(TOL (VELOCEINDIETRO -8 -7))
(TRA (INDIETRO -8 -7 -5 -4))
(TRA (LENTOINDIETRO -5 -4 -2 -1))
(TRA (FERMO -2 -1 1 2))
```

```
(TRA (LENTOAVANTI 1 2 4 5))
(TRA (AVANTI 4 5 7 8))
(TOR (VELOCEAVANTI 7 8))
)

(DIFFERENZAMISURESONAR
(TOL (MOLTORIDOTTA -70 -60))
(TRA (RIDOTTA -70 -60 -20 -10))
(TRA (INVARIATA -20 -10 10 20))
(TRA (AUMENTATA 10 20 60 70))
(TOR (MOLTOAUMENTATA 60 70))
)

(DIFFERENZAMISURADIMENSIONE
(TOL (VARIATA 50 60))
(TRA (LEGGERMENTEVARIATA 50 60 70 75))
(TOR (INVARIATA 70 75))
)

(DIFFERENZAMISURAANGOLO
(TOL (INVARIATA 6 10))
(TRA (LEGGERMENTEVARIATA 6 10 15 20))
(TOR (VARIATA 15 20))
)

(ANGOLOOSTACOLO
(TOL (OVEST -70 -60))
(TRA (NORDOVEST -70 -60 -40 -30))
(TRA (NORDNORDOVEST -40 -30 -10 0))
(TRI (NORD -10 0 10 ))
(TRA (NORDNORDEST 0 10 30 40))
(TRA (NORDEST 30 40 60 70))
(TOR (EST 60 70))
)

(DISTANZASONAR
(TOL (MOLTOVICINO 150 220))
(TRA (VICINO 150 220 280 300))
(TRA (LONTANO 280 300 450 500))
(TOR (MOLTOLONTANO 450 500))
)

(DISTANZAOSTACOLO
(TOL (MOLTOVICINO 40 60))
(TRA (VICINO 40 60 80 100))
(TOR (LONTANO 80 100))
)

(STATOGIOCO
```

```
(SNG (CERCONUOVOOSTACOLO 1))
(SNG (AVVICINAMENTONUOVOOSTACOLO 2))
(SNG (AVVICINAMENTONUOVOOSTACOLOALLACIECA 3))
(SNG (COSTEGGIAMENTOOSTACOLO 4))
(SNG (GIRODIETROALLOSTACOLOTROVATO 5))
(SNG (GIRODIETROALLOSTACOLOTROVATOFASE2 6))
(SNG (GIRODIETROALLOSTACOLOTROVATOFASE3 7))
(SNG (NASCOSTO 8))
)

(FLAG
(SNG (TRUE 1))
)
```

Il secondo file associa ad ogni variabile in ingresso la forma che meglio la descrive. Le variabili che hanno il prefisso "Proposed" sono associate alle azioni proposte dal livello inferiore.

```
(ProposedVelTan VELOCITAININGRESSO)
(ProposedVelRot VELOCITAININGRESSO)
(ProposedOstacoloTargetRilevato FLAG)
(ProposedOstacoloTargetRilevatoEADestra FLAG)
(ProposedStatoSpykee STATOGIOCO)

(DistanzaOstacolo DISTANZAOSTACOLO)
(AngoloOstacolo ANGOLOOSTACOLO)
(SonarNord DISTANZASONAR)
(SonarSud DISTANZASONAR)
(SonarEst DISTANZASONAR)
(SonarOvest DISTANZASONAR)
(Puntato FLAG)
(DiffNord DIFFERENZAMISURESONAR)
(DiffSud DIFFERENZAMISURESONAR)
(DiffEst DIFFERENZAMISURESONAR)
(DiffOvest DIFFERENZAMISURESONAR)
(OstacoloTargetPrecedentementeRilevato FLAG)
(StatoGiocoPrecedente STATOGIOCO)
(OstacoloTargetADestra FLAG)
(OstacoloSinistraDaVision FLAG)
(VelocitaRotPrecedente VELOCITAININGRESSO)
(VelocitaTanPrecedente VELOCITAININGRESSO)
(DifferenzaAngoloOstacoloTarget DIFFERENZAMISURAANGOLO)
(DifferenzaDimensioneOstacoloTarget DIFFERENZAMISURADIMENSIONE)
(OstacoloTargetPersoIn FLAG)
```

A.2 Predicati fuzzy

In questa sezione includiamo i due file di configurazione che rappresentano la conoscenza del contesto in cui si trova Spykee. Il primo file che includiamo rappresenta la conoscenza di Spykee sulla base dei dati fuzzy provenienti dall'esperto. Siccome questo file è molto lungo, abbiamo omesso la prima parte, ovvero i predicati ottenuti dalla combinazione del dato fuzzy con tutti i valori dell'insieme ad esso associato, perché abbiamo utilizzato dei nomi autoesplicativi. Abbiamo omesso anche l'ultima parte del file in cui erano presenti i predicati utilizzati dalle regole fuzzy. La parte che abbiamo inserito è la parte chiave che cerca di intuire cosa sia successo nel mondo esterno in base agli ingressi ottenuti dai sensori.

```
##############################################################################
###################Predicati derivati da quelli in ingresso###################
##############################################################################

#Predicati che indicano se e' presente un'ostacolo target da vision#############
##############################################################################
OstacoloTargetTrovato =
    (OR (P OstacoloTargetMoltoVicino)
            (OR (P OstacoloTargetVicino) (P OstacoloTargetLontano)));
OstacoloTargetNonTrovato = (NOT (P OstacoloTargetTrovato));


#Predicati che cercano di capire se i sonar hanno rilevato l'ostacolo target#####
##############################################################################

#nel caso sono negli stati di avvicinamento
OstacoloRilevatoOvestAvvicinamentoNormale =
    (AND (AND (P StatoSpykeePrecedenteAvvicinamentoNuovoOstacolo)
              (P OstacoloTargetMoltoVicino))
                 (AND (P OstacoloSinistraSpigoloDestraUltimoDiVision)
                      (AND (OR (P DiffOvestMoltoRidotta) (P DiffOvestRidotta))
                           (NOT (P SonarOvestMoltoLontano))))));

OstacoloRilevatoEstAvvicinamentoNormale =
    (AND (AND (P StatoSpykeePrecedenteAvvicinamentoNuovoOstacolo)
              (P OstacoloTargetMoltoVicino))
         (AND (P OstacoloDestraSpigoloSinistraUltimoDiVision)
              (AND (OR (P DiffEstMoltoRidotta) (P DiffEstRidotta))
                   (NOT (P SonarEstMoltoLontano))))));

OstacoloRilevatoOvestAvvicinamentoCieca =
    (AND (P StatoSpykeePrecedenteAvvicinamentoNuovoOstacoloAllaCieca)
                 (AND (P OstacoloTargetASinistraPrecedente)
```

```
                    (AND (OR (P DiffOvestMoltoRidotta) (P DiffOvestRidotta))
                         (NOT (P SonarOvestMoltoLontano)))))) ;

OstacoloRilevatoEstAvvicinamentoCieca =
    (AND (P StatoSpykeePrecedenteAvvicinamentoNuovoOstacoloAllaCieca)
                (AND (P OstacoloTargetADestraPrecedente)
                     (AND (OR (P DiffEstMoltoRidotta) (P DiffEstRidotta))
                          (NOT (P SonarEstMoltoLontano))))) ;

OstacoloRilevatoOvestAvvicinamento =
    (OR (P OstacoloRilevatoOvestAvvicinamentoNormale)
        (P OstacoloRilevatoOvestAvvicinamentoCieca)) ;

OstacoloRilevatoEstAvvicinamento =
    (OR (P OstacoloRilevatoEstAvvicinamentoNormale)
        (P OstacoloRilevatoEstAvvicinamentoCieca)) ;

OstacoloRilevatoAvvicinamento = (OR (P OstacoloRilevatoOvestAvvicinamento)
                                    (P OstacoloRilevatoEstAvvicinamento)) ;

#predicati generali (da usare salvo casi particolari)

OstacoloRilevatoEst = (P OstacoloRilevatoEstAvvicinamento) ;
OstacoloRilevatoOvest = (P OstacoloRilevatoOvestAvvicinamento) ;
OstacoloRilevato = (OR (P OstacoloRilevatoEst) (P OstacoloRilevatoOvest)) ;

OstacoloNonRilevato = (NOT (P OstacoloRilevato)) ;


#predicati che servono per capire se il costeggiamento va a buon fine
##########################################################################

MiAvvicinoAllOstacoloDaCosteggiareOvestMolto =
(AND (OR (P StatoSpykeePrecedenteCosteggiamentoOstacolo) (P OstacoloRilevato))
     (AND (P OstacoloTargetASinistraPrecedente) (P DiffOvestMoltoRidotta))) ;

MiAvvicinoAllOstacoloDaCosteggiareOvest =
(AND (OR (P StatoSpykeePrecedenteCosteggiamentoOstacolo) (P OstacoloRilevato))
     (AND (P OstacoloTargetASinistraPrecedente) (P DiffOvestRidotta))) ;

MiAvvicinoAllOstacoloDaCosteggiareEstMolto =
(AND (OR (P StatoSpykeePrecedenteCosteggiamentoOstacolo) (P OstacoloRilevato))
     (AND (P OstacoloTargetADestraPrecedente) (P DiffEstMoltoRidotta))) ;

MiAvvicinoAllOstacoloDaCosteggiareEst =
(AND (OR (P StatoSpykeePrecedenteCosteggiamentoOstacolo) (P OstacoloRilevato))
     (AND (P OstacoloTargetADestraPrecedente) (P DiffEstRidotta))) ;

MiAllontanoDallOstacoloDaCosteggiareOvestMolto =
```

```
(AND (OR (P StatoSpykeePrecedenteCosteggiamentoOstacolo) (P OstacoloRilevato))
     (AND (P OstacoloTargetASinistraPrecedente) (P DiffOvestMoltoAumentata)));

MiAllontanoDallOstacoloDaCosteggiareOvest =
(AND (OR (P StatoSpykeePrecedenteCosteggiamentoOstacolo) (P OstacoloRilevato))
     (AND (P OstacoloTargetASinistraPrecedente) (P DiffOvestAumentata)));

MiAllontanoDallOstacoloDaCosteggiareEstMolto =
(AND (OR (P StatoSpykeePrecedenteCosteggiamentoOstacolo) (P OstacoloRilevato))
     (AND (P OstacoloTargetADestraPrecedente) (P DiffEstMoltoAumentata)));

MiAllontanoDallOstacoloDaCosteggiareEst =
(AND (OR (P StatoSpykeePrecedenteCosteggiamentoOstacolo) (P OstacoloRilevato))
     (AND (P OstacoloTargetADestraPrecedente) (P DiffEstAumentata)));

CosteggiamentoOkOvest =
    (AND (OR (P StatoSpykeePrecedenteCosteggiamentoOstacolo) (P OstacoloRilevato))
         (AND (P OstacoloTargetASinistraPrecedente) (P DiffOvestInvariata)));

CosteggiamentoOkEst =
    (AND (OR (P StatoSpykeePrecedenteCosteggiamentoOstacolo) (P OstacoloRilevato))
         (AND (P OstacoloTargetADestraPrecedente) (P DiffEstInvariata)));

CosteggiamentoOk =
    (AND (OR (P StatoSpykeePrecedenteCosteggiamentoOstacolo) (P OstacoloRilevato))
         (OR (P CosteggiamentoOkOvest) (P CosteggiamentoOkEst)));


#predicati che indicano se i sonar rilevano un'ostacolo vicino
#################################################################
OstacoloTroppoVicino = (OR (OR (P SonarNordMoltoVicino) (P SonarEstMoltoVicino))
                           (OR (P SonarSudMoltoVicino) (P SonarOvestMoltoVicino)));


#predicati che indicano se i sonar hanno perso l'ostacolo obbiettivo
#######################################################################
OstacoloTargetPersoOvest =
    (AND (P StatoSpykeePrecedenteCosteggiamentoOstacolo)
              (AND (P OstacoloTargetASinistraPrecedente)
                   (AND (P DiffOvestMoltoAumentata)
                        (OR (P SonarOvestMoltoLontano) (P SonarOvestLontano))))));

OstacoloTargetPersoEst =
    (AND (P StatoSpykeePrecedenteCosteggiamentoOstacolo)
                  (AND (P  OstacoloTargetADestraPrecedente)
                       (AND (P DiffEstMoltoAumentata)
                            (OR (P SonarEstMoltoLontano) (P SonarEstLontano))))));

OstacoloTargetPerso = (OR (P OstacoloTargetPersoOvest) (P OstacoloTargetPersoEst));
```

```
#predicati che indicano se vision ha trovato un altro ostacolo target
#########################################################################
GrandiVariazioniOstacoloTarget = (OR (P AngoloOstacoloTargetCambiato)
                                     (P DimensioneOstacoloTargetCambiata));

PiccoleVariazioniOstacoloTarget =
    (AND (P AngoloOstacoloTargetLeggermenteCambiato )
         (P DimensioneOstacoloTargetLeggermenteCambiata));

OstacoloVisionCambiato = (OR (P GrandiVariazioniOstacoloTarget)
                             (P PiccoleVariazioniOstacoloTarget));

OstacoloVisionNonCambiato = (NOT (P OstacoloVisionCambiato));


#predicato che indica quali sono gli stati per girare intorno
#############################################################
StatiGiraIntorno =
(OR (OR (P StatoSpykeePrecedenteGiraDietroOstacoloTrovato)
        (P StatoSPykeePrecedenteGiraDietroOstacoloTrovatoFase2))
    (P StatoSPykeePrecedenteGiraDietroOstacoloTrovatoFase3));


#predicati utili al comportamento GiraIntornoOstacolo target
############################################################

VariazioneSonarOvest = (AND (OR (P DiffOvestMoltoRidotta) (P DiffOvestRidotta))
                            (NOT (P SonarOvestMoltoLontano)));

VariazioneSonarEst = (AND (OR (P DiffEstMoltoRidotta) (P DiffEstRidotta))
                          (NOT (P SonarEstMoltoLontano)));

OstacoloTargetRilevatoAncoraSinistra =
    (AND (AND (OR (P StatoSpykeePrecedenteGiraDietroOstacoloTrovato)
                  (P StatoSPykeePrecedenteGiraDietroOstacoloTrovatoFase3))
              (P OstacoloTargetASinistraPrecedente))
         (P VariazioneSonarOvest));

OstacoloTargetRilevatoAncoraDestra =
    (AND (AND (OR (P StatoSpykeePrecedenteGiraDietroOstacoloTrovato)
                  (P StatoSPykeePrecedenteGiraDietroOstacoloTrovatoFase3))
              (P OstacoloTargetADestraPrecedente))
             (P VariazioneSonarEst));

OstacoloTargetRilevatoAncora =
    (OR (P OstacoloTargetRilevatoAncoraSinistra)
        (P OstacoloTargetRilevatoAncoraDestra));
```

```
VariazioneSonarOvest2 = (OR (OR (P DiffOvestAumentata)
                                (P DiffOvestMoltoAumentata))
                            (P SonarOvestMoltoLontano)) ;

VariazioneSonarEst2 = (OR (OR (P DiffEstAumentata)
                              (P DiffEstMoltoAumentata))
                          (P SonarEstMoltoLontano)) ;

OstacoloTargetPersoAncoraSinistra = (
    AND (AND (P StatoSPykeePrecedenteGiraDietroOstacoloTrovatoFase2)
             (P OstacoloTargetASinistraPrecedente))
         (P VariazioneSonarOvest2)) ;

OstacoloTargetPersoAncoraDestra =
    (AND (AND (P StatoSPykeePrecedenteGiraDietroOstacoloTrovatoFase2)
              (P OstacoloTargetADestraPrecedente))
              (P VariazioneSonarEst2)) ;

OstacoloTargetPersoAncora = (OR (P OstacoloTargetPersoAncoraSinistra)
                                (P OstacoloTargetPersoAncoraDestra)) ;
```

Il secondo file che includo rappresenta la conoscenza offerta dai comportamenti di livello inferiore. Nella nostra applicazione quasi tutti i comportamenti appartengono allo stesso livello, quindi l'unico comportamento che beneficia di queste informazioni è quello che permette a Spykee di evitare gli ostacoli.

```
propIndietroVeloce = (D ProposedVelTan VELOCEINDIETRO) ;
propIndietro = (D ProposedVelTan INDIETRO) ;
propIndietroLento = (D ProposedVelTan LENTOINDIETRO) ;
propFermo = (D ProposedVelTan FERMO) ;
propAvantiLento = (D ProposedVelTan LENTOAVANTI) ;
propAvanti = (D ProposedVelTan AVANTI) ;
propAvantiVeloce = (D ProposedVelTan VELOCEAVANTI) ;


propSinistraVeloce  = (D ProposedVelRot VELOCEINDIETRO) ;
propSinistra = (D ProposedVelRot INDIETRO) ;
propSinistraLento = (D ProposedVelRot LENTOINDIETRO) ;
propDritto = (D ProposedVelRot FERMO) ;
propDestraLento = (D ProposedVelRot LENTOAVANTI) ;
propDestra = (D ProposedVelRot AVANTI) ;
propDestraVeloce = (D ProposedVelRot VELOCEAVANTI) ;

propOstacoloTargetRilevato = (D ProposedOstacoloTargetRilevato TRUE) ;
propOstacoloTargetEADestra =
    (D ProposedOstacoloTargetRilevatoEADestra TRUE) ;
propOstacoloTargetEASinistra = (NOT (P propOstacoloTargetEADestra)) ;
```

```
propStatoSpykeeCosteggiamento =
    (D ProposedStatoSpykee COSTEGGIAMENTOOSTACOLO);

propVelA = (OR (OR (P propAvantiLento) (P propAvanti))
               (P propAvantiVeloce));

propVelI = (OR (OR (P propIndietroLento) (P propIndietro))
               (P propIndietroVeloce));

propVelS = (OR (OR (P propSinistraLento) (P propSinistra))
               (P propSinistraVeloce));

propVelD = (OR (OR (P propDestraLento) (P propDestra))
               (P propDestraVeloce));
```

A.3 Evita ostacoli

Questo comportamento è l'unico che abbiamo inserito in questo elaborato per due ragioni. La prima ragione che ci ha spinto ad inserirlo in questo elaborato è il fatto che questo comportamento è l'unico di secondo livello e inoltre svolge una funzione molto critica nella nostra applicazione. La seconda ragione è per dare un esempio di un comportamento utilizzato da Mr. Brian.

```
################################################
# Cerca di evitare gli ostacoli troppo vicini#
################################################

#Nel caso stiamo andando avanti o indietro e
#ci sia un ostacolo di fronte o dietro
################################################

#nel caso c'e' un ostacolo a nord e io sto andando avanti
(AND (AND (propVelA) (SonarNordMoltoVicino))
          (AND (propDritto) (SonarOvestMoltoLontano))) =>
    (&DEL.VelTan ANY) (&DEL.VelRot ANY) (VelTan LENTOINDIETRO)
        (VelRot LENTOINDIETRO) (StatoSpykee CERCONUOVOOSTACOLO);



(AND (AND (propVelA) (SonarNordMoltoVicino))
     (AND (propDritto) (AND (NOT (SonarOvestMoltoLontano))
                            (SonarEstMoltoLontano)))) =>
    (&DEL.VelTan ANY) (&DEL.VelRot ANY) (VelTan LENTOINDIETRO)
    (VelRot LENTOAVANTI) (StatoSpykee CERCONUOVOOSTACOLO);
```

```
(AND (AND (propVelA) (SonarNordMoltoVicino))
     (AND (propDritto)
          (AND (AND (NOT (SonarOvestMoltoLontano))
                    (NOT (SonarEstMoltoLontano)))
               (SonarSudMoltoLontano)))) =>
    (&DEL.VelTan ANY) (&DEL.VelRot ANY) (VelTan LENTOINDIETRO)
    (VelRot FERMO) (StatoSpykee CERCONUOVOOSTACOLO);

#nel caso ci sia un ostacolo a nord e
#noi vogliamo andare a nord, sinistra
(AND (AND (propVelA) (SonarNordMoltoVicino))
     (AND (propVelS) (SonarOvestMoltoLontano))) =>
    (&DEL.VelTan ANY) (VelTan FERMO)
    (StatoSpykee CERCONUOVOOSTACOLO);

(AND (AND (propVelA) (SonarNordMoltoVicino))
     (AND (propVelS) (AND (NOT (SonarOvestMoltoLontano))
                          (SonarEstMoltoLontano)))) =>
    (&DEL.VelTan ANY) (VelTan FERMO) (&DEL.VelRot ANY)
    (VelRot LENTOAVANTI) (StatoSpykee CERCONUOVOOSTACOLO);

(AND (AND (propVelA) (SonarNordMoltoVicino))
     (AND (propVelS) (AND (AND (NOT (SonarOvestMoltoLontano))
                               (NOT (SonarEstMoltoLontano)))
                          (SonarSudMoltoLontano)))) =>
    (&DEL.VelTan ANY) (VelTan LENTOINDIETRO) (&DEL.VelRot ANY)
    (VelRot FERMO) (StatoSpykee CERCONUOVOOSTACOLO);


#nel caso ci sia un ostacolo a nord e
#noi volgiamo andare a nord, destra
(AND (AND (propVelA) (SonarNordMoltoVicino))
     (AND (propVelD) (SonarEstMoltoLontano))) =>
    (&DEL.VelTan ANY) (VelTan FERMO)
    (StatoSpykee CERCONUOVOOSTACOLO);

(AND (AND (propVelA) (SonarNordMoltoVicino))
     (AND (propVelD) (AND (NOT (SonarEstMoltoLontano))
                          (SonarOvestMoltoLontano)))) =>
    (&DEL.VelTan ANY) (VelTan FERMO) (&DEL.VelRot ANY)
    (VelRot LENTOINDIETRO) (StatoSpykee CERCONUOVOOSTACOLO);

(AND (AND (propVelA) (SonarNordMoltoVicino))
     (AND (propVelS) (AND (AND (NOT (SonarOvestMoltoLontano))
                               (NOT (SonarEstMoltoLontano)))
                          (SonarSudMoltoLontano)))) =>
    (&DEL.VelTan ANY) (VelTan LENTOINDIETRO) (&DEL.VelRot ANY)
    (VelRot FERMO) (StatoSpykee CERCONUOVOOSTACOLO);
```

```
#nel caso ci sia un ostacolo a sud
#e noi vogliamo andare a sud
(AND (AND (propVelI) (SonarSudMoltoVicino))
     (propDritto)) => (&DEL.VelTan ANY) (VelTan FERMO)
                      (StatoSpykee CERCONUOVOOSTACOLO);


#nel caso ci sia un ostacolo a sud
#e vogliamo andare a sud, sinistra
(AND (AND (propVelI) (SonarSudMoltoVicino))
     (AND (propVelS) (SonarOvestMoltoLontano))) =>
    (&DEL.VelTan ANY) (VelTan FERMO) (&DEL.VelRot ANY)
    (VelRot LENTOINDIETRO) (StatoSpykee CERCONUOVOOSTACOLO);

(AND (AND (propVelI) (SonarSudMoltoVicino))
     (AND (propVelS) (AND (NOT (SonarOvestMoltoLontano))
                          (SonarEstMoltoLontano)))) =>
    (&DEL.VelTan ANY) (VelTan FERMO) (&DEL.VelRot ANY)
    (VelRot LENTOAVANTI) (StatoSpykee CERCONUOVOOSTACOLO);

(AND (AND (propVelI) (SonarSudMoltoVicino))
     (AND (propVelS) (AND (NOT (SonarOvestMoltoLontano))
                          (NOT (SonarEstMoltoLontano))))) =>
    (&DEL.VelTan ANY) (VelTan FERMO) (&DEL.VelRot ANY)
    (VelRot FERMO) (StatoSpykee CERCONUOVOOSTACOLO);


#nel caso ci sia un ostacolo
#a sud e vogliamo andare a sud, destra
(AND (AND (propVelI) (SonarSudMoltoVicino))
     (AND (propVelD) (SonarEstMoltoLontano))) =>
    (&DEL.VelTan ANY) (VelTan FERMO) (&DEL.VelRot ANY)
    (VelRot LENTOAVANTI) (StatoSpykee CERCONUOVOOSTACOLO);

(AND (AND (propVelI) (SonarSudMoltoVicino))
     (AND (propVelD) (AND (NOT (SonarEstMoltoLontano))
                          (SonarOvestMoltoLontano)))) =>
    (&DEL.VelTan ANY) (VelTan FERMO) (&DEL.VelRot ANY)
    (VelRot LENTOINDIETRO) (StatoSpykee CERCONUOVOOSTACOLO);

(AND (AND (propVelI) (SonarSudMoltoVicino))
     (AND (propVelD) (AND (NOT (SonarOvestMoltoLontano))
                          (NOT (SonarEstMoltoLontano))))) =>
    (&DEL.VelTan ANY) (VelTan FERMO) (&DEL.VelRot ANY)
    (VelRot FERMO) (StatoSpykee CERCONUOVOOSTACOLO);


#ma c'e' un ostacoloa sinistra o a destra che si avvicina,
```

```
#per esempio in fase di costeggio
#################################################################


(AND (propVelA)
      (AND (SonarOvestMoltoVicino)
           (OR (DiffOvestRidotta) (DiffOvestMoltoRidotta)))) =>
    (&DEL.VelRot ANY) (VelRot AVANTI);

(AND (propVelA)
      (AND (SonarEstMoltoVicino)
           (OR (DiffEstRidotta) (DiffEstMoltoRidotta)))) =>
    (&DEL.VelRot ANY) (VelRot INDIETRO);


#Nel caso ci sono tutti e quattro i sonar che
#indicano un ostacolo il robot deve rimanere fermo
#################################################################
(AND (AND (SonarOvestMoltoVicino) (SonarEstMoltoVicino))
      (AND (SonarSudMoltoVicino) (SonarNordMoltoVicino))) =>
    (&DEL.VelTan ANY) (&DEL.VelRot ANY)
    (VelTan FERMO) (VelRot FERMO);
```

A.4 Defuzzyficazione

In questa sezione inseriamo la parte mancante alla descrizione delle regole per utilizzare Mr. Brian, ovvero la defuzzyficazione dei dati fuzzy che i comportamenti hanno proposto. La sintassi è molto simile alla defuzzyficazione, con la differenza che la direzione è inversa. Questo è il file che definisce le forme degli insiemi per la defuzzyficazione.

```
(VELOCITAUSCITA
(SNG (VELOCEINDIETRO -9))
(SNG (INDIETRO -6))
(SNG (LENTOINDIETRO -3))
(SNG (FERMO 0))
(SNG (LENTOAVANTI 3))
(SNG (AVANTI 6))
(SNG (VELOCEAVANTI 9))
)

(STATOGIOCOOUT
(SNG (CERCONUOVOOSTACOLO 1))
(SNG (AVVICINAMENTONUOVOOSTACOLO 2))
(SNG (AVVICINAMENTONUOVOOSTACOLOALLACIECA 3))
(SNG (COSTEGGIAMENTOOSTACOLO 4))
(SNG (GIRODIETROALLOSTACOLOTROVATO 5))
```

```
(SNG (GIRODIETROALLOSTACOLOTROVATOFASE2 6))
(SNG (GIRODIETROALLOSTACOLOTROVATOFASE3 7))
(SNG (NASCOSTO 8))
)

(FLAG
(SNG (TRUE 1))
(SNG (FALSE 0))
)
```

Mentre il file sottostante racchiude le variabili fuzzy in uscita.

```
(VelTan VELOCITAUSCITA)
(VelRot VELOCITAUSCITA)
(OstacoloTargetRilevato FLAG)
(OstacoloTargetEADestra FLAG)
(StatoSpykee STATOGIOCOOUT)
(OstacoloTargetPerso FLAG)
```

Appendice B

Manuale utente

Questa appendice racchiude il manuale utente, ovvero tutte le informazioni necessarie ad utilizzare la nostra applicazione. La prima parte ricopre la compilazione e l'installazione della nostra applicazione. La seconda parte spiega come calibrare correttamente la camera di Spykee. La terza parte mostra l'interazione con il Wiimote che offre la nostra applicazione.

B.1 Compilazione e installazione

La nostra applicazione è stata compilata sulla piattaforma GNU/Linux. La distribuzione su cui abbiamo testato la nostra applicazione è Ubuntu, sia la versione 9.10 che la 10.04, entrambe a 32 bit. Prima di poter compilare la nostra applicazione occorre assicurarsi di avere installato tutto il software necessario. Il pacchetto *build-essential* comprende i compilatori che abbiamo utilizzato per ottenere i binari dell'applicazione. La libreria OpenCv è possibile compilarla dai sorgenti (presenti nel sito ufficiale), noi abbiamo utilizzato la versione 2.1. Per la guida ufficiale all'installazione andare all'indirizzo http://opencv.willowgarage.com/wiki/InstallGuide. Per riuscire a compilare la nostra applicazione è necessario compilare questa libreria nella cartella /usr/local come suggerisce la guida. Se lo desidera, l'utente può modificare i makefile specificando un diverso percorso di installazione. In alternativa è possibile installarla dai repository ufficiali installando i pacchetti *libcv4*, *libhighgui4* e *libcvaux4*.

Occorre verificare di avere installato il pacchetto *libxml2*, *libxml2-dev* e *libxml2-utils* per assicurarsi di avere la libreria libXml2. Occorre verificare di aver installato anche i pacchetti *libbluetooth-dev*, *libxtst-dev*, *libsdl1.2-dev* e *freeglut3-dev* per poter utilizzare la libreria Wiiuse.

Una volta assicurati di avere tutto il software necessario, è sufficiente posizionarsi nella cartella di lavoro della nostra applicazione e da terminale lanciare il comando "make". Per verificare che la compilazione abbia avuto buon esito, nella cartella bin devono essere presenti due file binari: hs2 e calibrazione. Se non sono presenti questi file significa che la compilaizone non è andata a buon fine, occorre quindi assicurarsi di aver installato correttamente tutti i prerequisiti che abbiamo elencato.

B.2 Calibrazione della camera

Prima di poter giocare utilizzando la nostra applicazione è necessario calibrare la camera di Spykee. Per effettuare questa operazione occorre seguire quattro semplici passi:

1. Accendere Spykee e assicurarsi che non sia inserito nella sua postazione caricabatterie.
2. Connettersi alla rete che ha creato Spykee, per riconoscerla occorre cercare una rete denominata SPYKEE e una sequenza alfanumerica.
3. Avviare il nostro tool di calibrazione, scrivendo a terminale "./calibrazione".
4. Posizionare la camera di Spykee in modo che la linea rossa orizzontale sia posizionata a 90 centimetri dall'inizio dello chassis di Spykee, contemporaneamente la linea rossa verticale deve rimanere perfettamente perpendicolare rispetto al pavimento. Una volta posizionata la camera nella maniera corretta premere il tasto "k" per uscire dall'applicazione.

B.3 Guida all'uso del Wiimote

L'interazione con la nostra applicazione è prevista essere fatta solamente utilizzando il Wiimote. Occorre che quest'ultimo sia in modalità esplorazione affinchè la nostra applicazione possa connetersi. Per entrare in questa modalità è necessario premere contemporaneamente i tasti "1" e "2". Fare riferimento alla Figura B.1 per la dislocazione dei pulsanti.

Per sparare a Spykee occorre solamente premere il grilletto del controller (tasto "B"). Premendo il tasto "A" si mette in pausa il gioco, mentre se il gioco è già in pausa o in controllo manuale si ricomincerà a giocare. Premendo il tasto "Home" è possibile attivare il controllo manuale. Attraverso l'uso dei pulsanti direzionali è possibile controllare manualmente il robot. Se si mantiene premuto il tasto "Power" si uscirà dall'applicazione.

Figura B.1: Ingrandimento del Wiimote

B.4 Istruzioni di gioco

Per poter giocare utilizzando la nostra applicazione occorre effettuare cinque semplici passi:

1. Accendere Spykee e assicurarsi che non sia inserito nella sua postazione caricabatterie.

2. Accendere lo Xbee a bordo di Spykee e inserire il connettore USB dell'altro Xbee nel nostro computer.

3. Connettersi alla rete che ha creato Spykee, per riconoscerla occorre cercare una rete denominata SPYKEE e una sequenza alfanumerica.

4. Avviare la modalità esplorazione del Wiimote. In questa modalità i led del Wiimote inizieranno a lampeggiare.

5. Avviare la nostra applicazione eseguendo a terminale "./hs2"

Da questo momento è possibile giocare utilizzando il regolamento presente nel Capitolo 3 di questo elaborato. Appena viene eseguita l'applicazione Spykee non ha ancora iniziato a giocare. Premendo il tasto "A" si avvia il gioco. Quando Spykee o il giocatore guadagna un punto, il gioco non

si ferma. Abbiamo effettuato tale scelta per rendere il gioco più dinamico. Se il giocatore desidera ricominciare a giocare dalla posizione di partenza quando qualcuno segna un punto, è sufficiente entrare in modalità manuale e posizionare Spykee nella posizione desiderata. Utilizzando questo sistema è possibile ricominciare la partita senza azzerare il punteggio.

B.5 Modalità controllo manuale

La modalitá manuale offre la possibilità al giocatore di poter comandare Spykee attraverso il Wiimote. Da quando si entra in modalità controllo manuale, i pulsanti direzionali del Wiimote controlleranno infatti i motori di Spykee. Per riprendere il gioco basterà premere nuovamente il pulsante “A”. La differenza sostanziale fra mettere in pausa il gioco ed entrare in modalità manuale è la situazione che si presenta quando il gioco riparte. Quando il gioco riprende dopo che lo si era messo in pausa, Spykee riprenderà il gioco esattamente da dove l’aveva interrotto. Al contrario se il gioco è stato messo in modalità manuale, quando riprende la partita, Spykee ritorna alla situazione di partenza cercando di nascondersi nel nascondiglio migliore che trova.