POLITECNICO DI MILANO

FACOLT DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea in Ingegneria informatica

RobotWII 2.0.1

Andrea Pontecovo
Matr. 702084

Indice

1 Introduzione

RoboWii 2.0.1 si basa sull'interazione tra un giocatore umano e il robot mobile Spykee tramite l'utilizzo di un controller Wiimote e di marker colorati indossati dal giocatore.
In questa versione del RoboWii, che deriva da RoboWii 1.0 si è scelto di utilizzare un robot commerciale (Spykee) per sviluppare un nuovo tipo di gioco basato sull'interazione tra robot e un giocatore umano.

2 Il gioco

2.1 Obiettivi

Il gioco RoboWii 2.0.1 si tratta di una variante del classico gioco del nascondino: in questo caso è il robot che cerca di nascondersi dal giocatore per un tempo sufficiente a fargli guadagnare un punto. Il giocatore dovrà invece cercare di "colpirlo" utilizzando il WiiMote guadagnando in questo modo un punto a suo favore.
Il gioco termina quando uno dei due partecipanti (umano o robot) raggiunge un punteggio prestabilito.

2.2 Descrizione generale e regole

Come detto precedentemente il gioco è una variante del nascondino, quindi le regole di base sono le stesse: il robot cerca di individuare il giocatore, tramite i marker che quest'ultimo indossa sul petto e sulla schiena, o rilevando di essere stato puntato con il WiiMote.
Quando il giocatore individua il robot dovrà puntarlo tramite il WiiMote e, tenendolo sotto tiro, tenere premuto il pulsante di fuoco per un tempo sufficiente a "caricare" il colpo vincendo il match e ottenendo un punto; in questo caso il robot inizierà un nuovo match, in caso contrario cercherà di nascondersi.
Il gioco è progettato per essere svolto principalmente al chiuso, in un ambiente che offra al robot diversi possibili nascondigli (ad esempio mobili), ben illuminato e non eccessivamente vasto.

2.3 Comportamenti del robot

In questa sezione vengono descritti nel dettaglio i comportamenti del robot durante il gioco e la sua interazione con giocatore e ambiente circostante.

2.3.1 Fuga per contato visivo

Quando il giocatore si trova davanti al robot, questo sarà in grado di vederlo (grazie al marker indossato) e reagirà girandosi per allontanarsi dal giocatore ed eventualmente nascondersi, cercando di tenere il giocatore fuori dal proprio campo visivo.

2.3.2 Fuga per rilevazione IR

Quando il giocatore punta Spykee con il WiiMote, il robot cercherà di scappare dando origine a un comportamento uguale a quello della regola precedente.

2.3.3 Ricerca giocatore

Questo comportamento viene eseguito all'avvio della partita, dopo che il robot viene colpito e ogni volta che ha raggiunto un nascondiglio. Spykee gira su se stesso in modo da individuare il giocatore tramite la telecamera. Se girando dovesse rilevare (tramite i sonar) che la telecamera andrà a guardare un ostacolo troppo vicino fermerà la rotazione e la riprenderà in senso opposto, in modo da osservare solo spazio libero.
Se durante la ricerca Spykee individua il giocatore tramite contatto visivo o se dovesse venire puntato cercherà di fuggire e nascondersi.R.

2.3.4 Il robot viene colpito (punto per il giocatore)

Quando il giocatore riesce a tenere sotto tiro il robot per un tempo sufficiente a caricare il colpo, il robot risulta colpito e di conseguenza il giocatore ottiene un punto, che viene indicato tramite i led blu del WiiMote. Una volta segnalato il punteggio e dopo un tempo di attesa di qualche secondo il robot si rimette alla ricerca del giocatore e il gioco riprende per il match successivo.

2.3.5 Spykee si nasconde

Quando il robot rileva il giocatore (tramite contatto visivo) o si accorge di essere puntato (rilevazione IR) cercherà di allontanarsi in una direzione casuale. Se durante la fuga, che avviene in linea retta, dovesse rilevare delle cavità (per la presenza di mobili o di altri ostacoli) sul lato destro o sinistro si fermerà e si dirigerà verso di esse in modo da essere almeno parzialmente coperto alla vista del giocatore. Una volta "nascosto" ricomincerà a cercare il giocatore.

3 Hardware

In questa sezione verrà illustrato l'hardware utilizzato per la realizzazione del progetto.

3.1 Spykee

Il robot utilizzato è un modello commerciale (modificato) prodotto dalla Meccano: Spykee.

Spykee è un robot autonomo mobile di forma umanoide dotato di due braccia fisse, di una luce bianca e di una telecamera a colori con una risoluzione di 320x240 px (entrambe montate sulla tesa). Il robot dispone inoltre di alcune luci colorate diffuse tramite fibre ottiche (non utilizzate per il progetto). E' possibile inoltre far emettere al robot alcuni suoni, ad esempio una sirena d'allarme o un esplosione.

Figura 1: Spykee

Il sistema di movimento del robot è di tipo differential drive: i due cingoli sono controllati indipendentemente l'uno dall'altro permettendo al robot di girare su se stesso o di effettuare curve con raggio variabile. La velocità massima in linea retta è di circa 30 cm al secondo.
Il robot ha un autonomia di circa 30 minuti e viene ricaricato tramite una base collegata alla rete elettrica. Per collegarsi al computer Spykee crea una connessione wi-fi di tipo ad hoc ed è controllabile anche tramite il software proprietario fornito col robot.

3.2 Sonar e led

Per poter essere utilizzato in diversi progetti sono state necessarie diverse aggiunte al robot Spykee. Il robot è stato equipaggiato con due led infrarossi, posizionati sulle spalle, in modo da poter essere individuato dal wiimote, inoltre essendo nota la distanza tra i due led è possibile stimare la distanza robot-giocatore. Oltre ai led IR sono presenti un led verde, che indica quando il robot viene colpito, e un gruppo di quattro led rossi (montati frontalmente) che indicano il livello di carica del colpo quando il giocatore tiene Spykee puntato. Per rilevare le distanze (e quindi gli ostacoli) sono stati montati

diversi sensori sonar sul robot con un raggio di circa 7 metri e un ampiezza di circa 30°. Questi sensori indicano la distanza dell'oggetto più vicino che si trova nel loro raggio d'azione.
Attualmente sono utilizzati sei sensori: tre frontali, utilizzati principalmente per rilevare gli ostacoli di fronte a Spykee e per eventualmente essere utilizzati insieme alla telecamera, due sensori laterali (a destra e a sinistra) utilizzati per la rilevazione delle cavità e quindi di possibili nascondigli e uno posteriore utilizzato per rilevare gli ostacoli.
Tutte le aggiunte sono alimentate tramite tre batterie stilo (indipendenti quindi dalla ricarica del robot) e comunicano i dati al computer tramite una scheda XBee.

3.3 XBee

Per poter trasmettere i dati dei sensori al computer e per poter comandare i led di Spykee è stata utilizzata una coppia di schede XBee (una collegata al computer e una a Spykee). Queste schede permettono di stabilire una connessione punto-punto tra di loro e si comportano come se fossero una porta seriale.
I dati vengono inviati e ricevuti tramite SonarExpert (4.3.1).

3.4 Wiimote

Il Wii Remote (wiimote) è il controller standard della console WII. Esso è dotato di una telecamera infrarssi, un accelerometro a tre assi, di quattro led blu controllabili separatamente e di diversi pulsanti. Al momento solo alcuni di essi sono utilizzati:

- Pulsante B: viene utilizzato per caricare il colpo e sparare al robot.
- Pulsante A: arresto di emergenza del robot.
- Pulsante -: passa al controllo manuale del robot e interrompe il gioco.
- Pulsante +: passa al gioco.
- Tasti direzionali: permettono il controllo del robot in modalità manuale.

Tramite la telecamera è possibile individuare i due led infrarossi presenti sul robot e quindi sapere quando esso è puntato oppure no. Le coordinate dei due led sono fornite come punti su un immagine' di 1024x768px. I led invece sono utilizzati per indicare il punteggio del giocatore e del robot, l'inizio e la fine di una partita.

Figura 2: Il controller wiimote

3.5 Computer

L'intero gioco viene gestito tramite un computer dotato di connettività Wi-Fi (per connettersi a Spykee) e bluetooth (per il Wiimote). E' inoltre necessario disporre di almeno una porta usb per poter collegare la scheda XBee.

3.6 Marker

Per poter essere riconosciuto dal robot il giocatore deve indossare almeno due marker colorati: uno sul torace e uno sulla schiena. Al momento il colore utilizzato è un rosa acceso in quanto facilmente riconoscibile dal robot.

4 Software

In questa sezione viene spiegato parte del codice sorgente del gioco e le librerie che sono state utilizzate per realizzarlo.

4.1 Librerie utilizzate

4.1.1 Wiiuse

Wiiuse è una libreria scritta in C/C++ che permette di interfacciare il computer con diversi controller wiimote, tramite collegamento Bluetooth.

Essa permette di rilevare la pressione dei tasti, controllare i led blu e inoltre dispone di funzioni per ottenere i valori registrati dall'accelerometro (permettendo cosi di conoscere l'orientamento del wiimote) e anche di ottenere, sotto forma di coordinate XY. E' inoltre possibile ottenere una stima della distanza dalla sorgente infrarossi (coordinata Z).
Per rilevare un wiimote bisogna utilizzare il seguente codice:

```
wiimotes = wiiuse_init(MAX_WIIMOTES);
found = wiiuse_find(wiimotes, MAX_WIIMOTES, CONNECTION_TIME);

if (!found) {
   printf ('No wiimotes found.');
   return 0;
}

connected = wiiuse_connect(wiimotes, MAX_WIIMOTES);

if (connected)
    printf('Connected to %i wiimotes.\n', connected);
else {
    printf('Failed to connect to any wiimote.\n');
    return 0;
}

wiiuse_motion_sensing(wiimotes[0],1);
wiiuse_set_ir(wiimotes[0], 1);
```

Mentre per rilevare gli eventi:

```
while (1) {
    if (wiiuse_poll(wiimotes, MAX_WIIMOTES)) {
       /* ... */
   }
}
wiiuse_cleanup(wiimotes, MAX_WIIMOTES);
```

Quindi il codice principale che controlla il robot e il gioco va inserito all'interno di questo ciclo while. Per una documentazione completa delle Wiiuse si veda [5].

4.1.2 Libreria Spykee

Questa libreria permette di controllare il robot Spykee; permette inoltre di ricevere il video della telecamera sotto forma di una sequenza di immagini

jpeg con una risoluzione 320x240px con una profondità di colore di 24bpp.
Le funzioni principali sono:

Spykee(char*username, char *password);

void Move (int right, int left)

void startCamera();

static byte *pharseMessage(byte *m, int *lm, byte **image, enum operations op)

Per una documentazione completa della libreria si veda [1].

4.1.3 OpenCV

OpenCV è una libreria open source [2] che consiste in un insieme di funzioni e classi scritte in C/C++ sviluppate e supportate da Intel per la visione artificiale. Questa libreria è orientata alla visione artificiale in tempo reale utilizzando quindi funzioni ad alte prestazione, ottimizzate soprattutto per architettura Intel. OpenCV comprende numerose funzioni di alto livello: tecniche di calibrazione, features detection, tracciamento tramite flusso ottico, analisi sulla geometria dei contorni, identificazione del movimento, ricostruzione 3D, riconoscimento dei gesti della mano e segmentazione dell'immagine. Permette inoltre di lavorare sui colori delle immagini riconoscendo aree con un determinato colore, permettendone il tracciamento.

4.1.4 libjpeg

Questa è la libreria standard per la lettura e scrittura di immagini in formato jpeg. E' stato necessario utilizzarla per poter convertire il flusso di immagini in formato jpeg acquisito da Spykee e passarlo alle OpenCV come sequenza di immagini non compresse (RGB).

4.2 Visione

4.2.1 Istogramma del colore

L'istogramma immagine è una rappresentazione della distribuzione delle diverse tonalità di colore in una immagine.
Sull'asse orizzontale vi sono le differenti tonalità di colore, mentre su quello verticale vi è il conteggio dei pixel.
L'altezza delle barre dell'istogramma rappresenta il numero di pixel presenti nell'immagine che possiedono quel particolare colore.

Questo istogramma è particolarmente utilie nel riconoscimento delle immagini (e di conseguenza degli oggetti che vi sono rappresentati) in quanto ogni immagine possiede un istogramma del colore specifico. Inoltre se si lavora in uno spazio dei colori in cui il colore dei pixel (tonalità) è indipendete dalla luminosità (4.2.2) degli stessi l'istogramma diventa un ottimo modo per riconoscere un oggetto collocato anche in ambienti diversi.
Trammite le librerie OpenCv è possibile 'filtrare' un'immagine con un determinato istogramma in modo da ottenere una rappresentazione in scala di grigi, chiamata backproject, dell'immagine originale. Più un pixel è bianco più questo rientra nel range di colori dell'istogramma. (Figura 5)

4.2.2 Spazio colore HSV

Lo spazio del colore HSV (Hue, Saturation, Value) è un modo, differente dal RGB, per rappresentare il colore. La tonalità (hue) indica il colore stesso ed ha un valore compreso tra 0 (rosso) e 360 (ancora rosso), infatti nella rappresentazione "a cono" corrisponde all'angolo lungo l'asse verticale. La saturazione corrisponde alla distanza dall'asse del cono e varia tra 0 e 1. La luminosità (Value) varia anch'essa tra 0 e 1 e corrisponde all'altezza del cono (1 in cima corrisponde al bianco, 0 al vertice inferiore al nero). L'utilizzo di questa rappresentazione del colore è molto comoda in quanto permette di tenere separati la tonalità di una determinata regione dell'immagine da altre sue caratteristiche che in questo caso risultano meno utili (la saturazione e la luminosità).
Nell'utilizzare questa rappresentazione con le OpenCV bisogna porre attenzione al fatto che l'hue varia tra 0 e 180, mentre Saturation e Value variano tra 0 e 255.

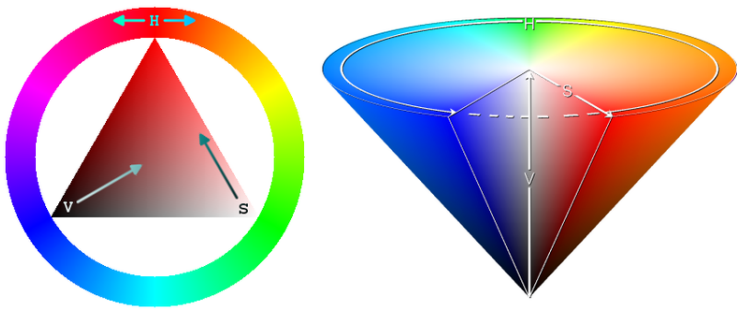


Figura 3: Rappresentazione dello spazio dei colori in HSV

4.2.3 Algoritmo di visione

Il modulo che si occupa del riconoscimento visivo è VisionExpert, che composto dalle seguenti funzioni ausiliarie:

VisionExpert(): È il costruttore della classe qui sono inizializzate le variabili e i buffer necessari all'acquisizione e all'elaborazione delle immagini (si vedano i commenti del codice per la descrizione delle singole variabili).

*unsigned char *jpeg2bmp(unsigned char *jpeg, int len)*: Qusta funzione si occupa di convertire un immagine in formato jpeg, rappresentata come un vettore di byte, in un immagine non compressa in formato bmp anch'essa rappresentata come un array di byte, in cui ogni tripletta di byte indica il contenuto di rosso, verde e blu di un singolo pixel. L'immagine è restituita come valore di ritorno della funzione. Non è necessario specificare la lunghezza del vettore in quanto essa sarà sempre: altezza x larghezza x 3. La conversione è effettuata mediante le librerie libjpeg.

*CvHistogram *CreateMatchingHist(string file)*: Questa funzione crea l'istogramma del colore di un immagine campione (che si suppone rappresenti il marker) fornita dall'utente. Questo istogramma verrà poi utilizzato dall'algoritmo per identificare il giocatore.
Questa funzione viene richiamata solo una volta all'interno del costruttore della classe in quanto il marker utilizzato nella sessione di gioco sarà sempre lo stesso.

L'algoritmo vero e proprio è contenuto nella funzione RunDuty (eseguita in loop dall'MRT).
I passi principali dell'algoritmo sono:

1. righe 109-118: qui l'immagine viene acquisita, tramite la funzione Spykee::pharseMessage, che restituisce l'immagine in formato jpeg memorizzata nella variabile, di tipo unsigned char*, 'jpeg' e che memorizza in imgLen la dimensione del vettore restituito.

2. riga 123: l'immagine viene convertita in formato bmp utilizzando la funzione jpegToBmp. Questo è stato necessario dato che le openCV non permettono la decodifica di un immagine jpeg già caricata in memoria. L'immagine restituita è memorizzata nella variabile 'pBmp' di tipo unsigned char*.

3. riga 126: il vettore dell'immagine bmp viene 'incollato' (con la funzione standard C 'memcpy') sul campo imageData della variabile membro

m_image (variabile di tipo CvImage, struttra dati che nelle opencv rapprenta un immagine con tutte le sue caratteristiche), che rappresenta il fotogramma corrente su cui si sta effettuando l'elaborazione.

4. righe 128-130: grazie alla funzione delle opencv 'cvCvtColor' l'immagine viene convertita da una rappresentazione RGB in una HSV.
   In seguito grazie a 'cvInRangeS' viene effettuata una sogliatura della saturazione e della luminosità dell'immagine per eliminare le zone troppo chiare e troppo scure che disturberebbero il riconoscimento del colore. Vengono inoltre escluse le tonalità di colori molto differenti da quelle del marker. Il risultato di queste operazioni è un immagine in bianco e nero memorizzata nella variabile m_mask (anch'essa di tipo CvImage*), in cui le zone nere sono le parti di immagine che non soddisfano i valori di soglia (quindi troppo chiare, troppo scure o di colore palesemente diverso da quello del marker).

5. riga 132: il canale hue (tonalità del colore) viene separato dall'immagine tramite la funzione delle opencv 'cvSplit'.

6. righe 134-138: viene calcolato il backproject utilizzando l'istogramma del colore dell'immagine di rifermento (calcolato con CreateMatchingHist). La funzione che si occupa di questo processo è la 'cvCalcBackProject'.
   Il backproject è una rappresentazione probabilistica dell'immagine originale che mostra per ogni pixel la probabilità di appartenere o meno alla gamma di colori contenuti nell'istogramma (e di conseguenza al marker). Per una migliore precisione il backproject viene poi messo in AND con la variabile m_mask, in modo da non considerare zone già escluse precedentemente, e sogliato per ottenere solo la zona in cui il colore è più "evidente" (mediante cvThreshold).

7. riga 145: tramite la funzione opencv 'cvCountNonZero' vengono conteggiati i pixel bianchi nel backproject.

8. righe 165-184: se il numero di pixel bianchi è superiore a una certa soglia (quindi ci sono buone probabilità che il giocatore si trovi nel campo visivo), vengono estratte una per volte (grazie a cvGetCol, che restituisce il una singola colonna di pixel in una immagine) le colonne di pixel del backproject. Su ogni colonna viene contato il numero di pixel bianchi (funzione cvCountNonZero). La colonna con il maggior numero di pixel bianchi identifica il marker. Conoscendo la posizione della colonna sull'asse x dell'immagine e conoscendo l'apertura della

telcamera (46°) viene calcolata, con una proporzione) la posizione angolare del giocatore rispetto alla direzione del robot.
Alla fine di ogni ciclo, che analizza un singolo fotogramma, viene inviato al modulo BrianExpert un messaggio di tipo VISION_DATA; esso contiene due parametri che indicano se il giocatore rientra nel campo visivo e la sua posizione, in gradi, rispetto all'asse della telecamera.

9. righe 185-194: nel caso in cui il giocatore esca dal campo visivo non verrà inviato a BrianExpert il messaggio di tipo VISION_DATA indicante questo evento.

Inizialmente si era pensato di utilizzare l'algoritmo CamShift per il tracking del giocatore. Tuttavia questo algoritmo funziona bene per tracciare il giocatore senza che questo esca dal campo visivo della telecamera. Se questo dovesse avvenire c'è il rischio che venga "agganciato" un oggetto sullo sfondo che ha un colore simile a quello del marker del giocatore. Quindi alla fine si è deciso di utilizzare l'algoritmo precedente per ottenere la posizione del giocatore.

4.3 MRT

Il software principale che controlla il gioco è sviluppato basandosi su MRT, un framework ideato per permettere l'esguzione parallela di diversi moduli, chiamati expert (che sono implementate come classi C++). Questi moduli possono essere sulla stessa macchina o anche distribuiti su differenti macchine collegate tramiete una rete LAN.
MRT permette inoltre di creare comportamenti e regole di gioco mediante l'utilizzo di logica fuzzy.

4.3.1 Experts

Gli experts non sono altro che classi C++ che ereditano alcuni metodi da una classe base. I metodi più importanti sono:

```
void RunInit(); //Viene chiamato alla creazione della
   classe per inizializzarla
void RunClose(); //Viene chiamato alla distruzione
   della classe
void RunDuty(); //Metodo che viene eseguito con un
   intervallo di x ( >= 0 ) millisecondi
```

Di seguito i moduli utilizzati nello sviluppo del gioco.

VisionExpert

E' il modulo che si occupa di getire la visione artificiale.
La funzione RunDuty() eseguita il loop (senza intervallo di tempo) contiene l'algorimo di visione (vedi: 4.2.3)
Il modulo si occupa anche di inviare a BrianExpert le informazioni aggiornate sulla posizione del giocatore quando entra nel campo visivo trmite (Vedi: 4.3.2)

SonarExpert

Questo modulo si occupa di di ricevere i valori letti dai sonar e controllare i led rossi e verdi montati su Spykee.
Invia i valori letti a WiimoteExpert (messaggio DISTANCE_DATA) e riceve messaggi che accendono o spengono i led montati su Spykee.

MotorExpert

Questo modulo si occupa, tramite la libreria Spykee di controllare i motori del robot. Riceve messaggi da BrianExpert e da WiimoteExpert.

WiimoteExpert

E' il modulo principale che controlla il gioco e l'interazione con l'utente.
Si occupa di gestire la comunicazione con il WiiMote (tramte le wiiuse) e contiene il ciclo principale di gioco. Invia messaggi di tipo WIIMOTE_DATA a diversi altri Expert: a BrianExpert per trasmettergli le distanze dei sonar, a SonarExpert per controllare lo stato dei led e a MotorExpert per controllare il movimento del robot (sia in controllo manuale che in fase di gioco).

4.3.2 Comunicazione tra moduli

I diversi Expert comunicano tra loro mediante messaggi in formato XML, ognuno dei quali è identificato da un nome univoco. Ogni Expert inoltre si mette in 'ascolto' per ricevere i messaggi che sono indirizzati a lui.
Alla creazione del messaggio è quindi necessario specificare: nome del messaggio e destinatario.
Ad esempio nella funzione RunInit() di BrianExpert viene richiamata:

```
AddMessageRequest(MSG_TO_BRIAN,LOCAL);
```

Che permette al modulo BrianExpert di ricevere tutti i messaggi indirizzati a lui.
Mentre in un altro modulo (ad esmepio nel VisionExpert) vi sarà:

```
NewMessage( MSG_TO_BRIAN, VISION_DATA );
```

Che crea un messaggio di tipo VISION_DATA per MSG_TO_BRIAN, che in questo caso identifica il BrianExpert.

4.3.3 Struttura dei messaggi

Una volta creato un nuovo messaggio questo andrà riempito coi dati da trasmettere.
Il messaggio appena creato viene memorizzato in mCurrentMessage. I dati da inviare possono essere aggiunti utilizzando l'operatore di stream del C++ nel modo seguente:

```
mCurrentMessage << '<D>' << nome << ' ' << valore
<< ' ' << relaiability << '</D>\n'
```

Dove 'nome' è il nome, in formato stringa, del dato da trasmettere (univoco), valore è il dato e relaiability ne indica l'affidabilità (entrambi valori in virgola mobile).
In un singolo messaggio si possono aggiungere dati a piacere; una volta completo lo si può chiudere e inviare mediante la funzione SendAllStringMessages(); .

4.3.4 Messaggi utilizzati per il gioco

Di seguito sono elencati i principali messaggi, con i relativi parametri, che i diversi moduli dell'MRT si scambiano per il corretto funzionamento del gioco.

Tipo	**Destinatario**	**Parametri**	**Descrizione**
VISION_DATA	BriandExpert	EnemyDetecter	Se il giocatore è nel campo visivo del robot è a 1, 0 altrimenti.
		EnemyAngle	Indica l'angolo a cui è stato rilevato il giocatore (0-47)
		Status	Indica in che stato si trova il robot.
WIIM_DATA	MotorExpert	TanSpeed	Indica la velocità in linea reatta
		RotSpeed	Indica la velocità di rotazione del robot
	BrianExpert	SonarLeft	Distanza sinistra
		SonarRight	Distanza destra
		SonarFront	Distanza frontale
		SonarBack	Distanza posteriore
		SonarLeftOld	Retroazione distanza sx
		SonarRightOld	Retroazione distanza dx
		Hit	Indica quando il robot viene colpito
LED_DATA	SonarExpert	Charge	Indica il numero di led rossi accessi
		Point	Comanda il led verde (0,1)

4.4 Logica Fuzzy

I comportamenti del robot sono decisi utilizzando la logica fuzzy: un tipo di logica in cui le proposizioni non assumono solo il valore vero o falso, ma hanno un grado di verità compreso tra 0 (falso) e 1 (vero).
In questo tipo di logica esistono anche gli operatori AND, OR e NOT che, a differenza di quelli booleani, sono definiti così:

NOT(X) = 1 - val(X)

AND(X Y) = min(val(X), val(Y))

OR(X Y) = max(val(X), val(Y))

Il software che si occupa di gestire la logica fuzzy è Mr. Brian sviluppato dal Politecnico di Milano per il progetto MRT (Milano RoboCup Team).

4.4.1 Fuzzificazione dei dati

Per prima cosa è necessario definire i valori in ingresso e la loro forma come variabili fuzzy. Queste vano definite nel file shape_ctof.txt
Un esempio di dichiarazione è:

```
(ENEMYANGLE
(TOL (LEFT 8 15))
(TRA (CENTER 8 15 31 38))
(TOR (RIGHT 31 38))
)
```

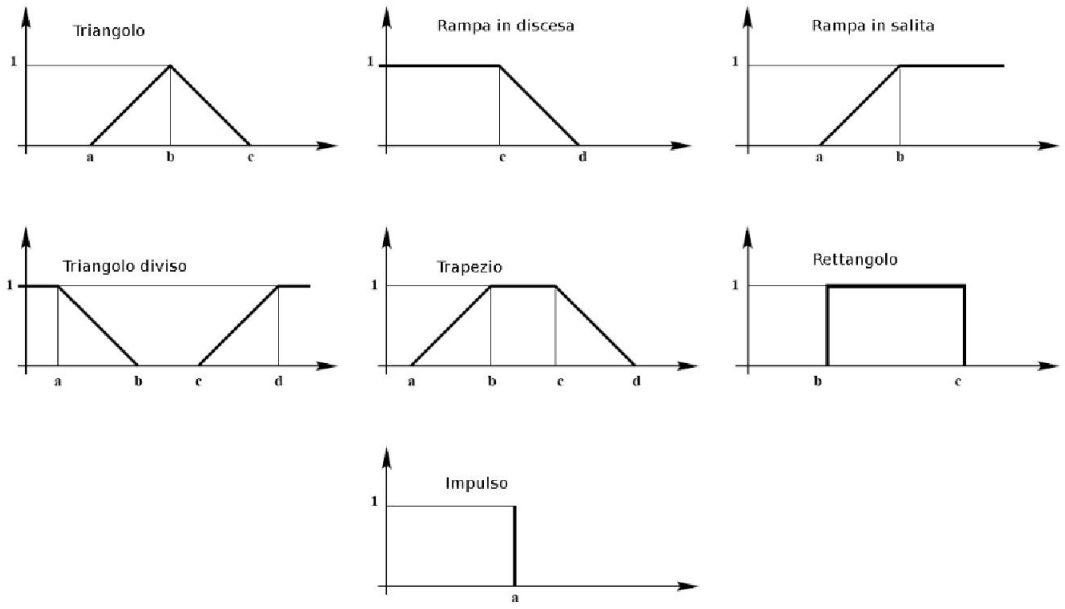


Figura 4: Le principali forme fuzzy

4.4.2 Predicati

Una volte definite le variabili fuzzy è necessario definire i predicati. Questi assumeranno un valore di verità compreso tra 0 e 1 in base al valore della variabili (collegate tra loro mediante connettivi AND, OR e NOT) a cui sono associati. Un esempio è:

```
EnemyFront = (AND (P ViewEnemy)(D EnemyAngle CENTER));
```

La variabile EnemyFront assumerà quindi un valore compreso tra 0 e 1 in base al valore assunto dall' AND tra le due variabili ViewEnemy e EnemyCenter.

4.4.3 Modulo CANDO

Questo modulo si occupa di decidere quali comportamenti attivare. Per la precisione il modulo CANDO decide quali comportamenti il robot puà attivare (ad esempio se il sonar posteriore rileva spazio libero Spykee può indietreggiare).
Le regole hanno la struttura:

```
EscapeIR = (P IRExist)
```

In questo caso il comportamento EscapeIR viene attivato solo se il predicato IRExists è vero

4.4.4 Modulo WANT

Il modulo WANT si comporta esattamente come il modulo CANDO. Viene usato per definire quali regole vogliano essere attivate in base a delle condizioni (fornite da predicati). Anche la struttura delle regole è identica a quella del modulo CANDO.

4.4.5 Regole

La lista delle regole disponibili è definita nel file 'behaviour.txt'. L'ordine con cui vi sono inserite è molto importante in quanto una regola di un livello superiore può modificare o anche annullare i comportamenti proposti dalle regole inferiori.
Ogni regola è definita in un file .rul ed è costituita da una o più righe strutturate così:

```
(pIn) => (var1 val1)(var2 val2)
```

dove pIn è un insieme di predicati in ingresso. Var1, var2 sono le variabili in uscita che assumono rispettivamente il valore val1 e val2.

4.4.6 Defuzzificazione dei dati

Le variabili in uscita sono definite nel file s_ftoc nella forma (nome tipo). Il tipo è specificato nel file s_shape.
Il valore di queste variabili verrà poi trasferito al codice di Mr Brian quindi possono essere definite solo come singleton. L'effettivo valore in uscita sarà una media pesata del valore imposto a quella variabile dalle varie regole attivate.

Un esempio è:

```
(ROTSPEED
(SNG (VERY_FAST_RIGHT -40))
(SNG (FAST_RIGHT -20))
(SNG (RIGHT -10))
(SNG (SLOW_RIGHT -5))
(SNG (AHEAD 0))
(SNG (SLOW_LEFT 5))
(SNG (LEFT 10))
(SNG (FAST_LEFT 20))
(SNG (VERY_FAST_LEFT 40))
)
```

4.5 Regole utilizzate

Per realizzare i comportamenti descritti nel capitolo 2.3 sono state utilizzate le seguenti regole (per la definizione delle variabili fuzzy si veda l'apendice A):

4.5.1 Fuga per contatto visivo

```
(ViewEnemy) => (&DEL.RotSpeed ANY)(&DEL.TanSpeed ANY)(&
    DEL.Status ANY);
(AND(EnemyLeft)(BackFree)) => (RotSpeed FAST_RIGHT)(
    MovingFlag DX180)(Status ESCAPE);
(AND(EnemyRight)(BackFree)) => (RotSpeed FAST_LEFT)(
    MovingFlag SX180)(Status ESCAPE);
(AND(EnemyFront)(BackFree)) => (&DEL.RotSpeed ANY)(
    TanSpeed FAST_BACKWARD)(Status ESCAPE);
(AND(EscapeOn)(NOT(ViewEnemy))) => (TanSpeed
    FAST_FORWARD);
```

4.5.2 Fuga per rilevazione IR

```
(IRDirClose) => (&DEL.RotSpeed ANY)(&DEL.TanSpeed ANY);
(IRDirClose) => (TanSpeed FAST_BACKWARD);
(IRDirNear) => (TanSpeed BACKWARD);
(AND(IRDirCloseR)(NOT(TurnRandomLeft))) => (RotSpeed
    FAST_RIGHT);
(AND(IRDirCloseL)(NOT(TurnRandomRight))) => (RotSpeed
    FAST_LEFT);
(AND(IRDirNearR)(NOT(TurnRandomLeft))) => (RotSpeed
    RIGHT);
(AND(IRDirNearL)(NOT(TurnRandomRight))) => (RotSpeed
    LEFT);
```

4.5.3 Ricerca giocatore

```
(SearchOn)=>(&DEL.TanSpeed ANY)(&DEL.RotSpeed ANY)(
    RotSpeed FAST_RIGHT)(Status SEARCH)(MovingFlag STOP)
```

4.5.4 Spykee si nasconde

```
(AND(Escape)(VoidOnLeft)) => (&DEL.TanSpeed ANY)(&DEL.
    RotSpeed ANY)(RotSpeed RIGHT);
(AND(Escape)(VoidOnRight)) => (&DEL.TanSpeed ANY)(&DEL.
    RotSpeed ANY)(RotSpeed LEFT);
```

4.5.5 Robot colpito

```
(OR(HumanPoint)(RobotPoint)) => (&DEL.RotSpeed ANY)(&
    DEL.TanSpeed ANY)(Status SERACH);
```

5 Analisi dei risultati

L'algoritmo utilizzato è in grado di distinguere un giocatore che indossa un marker di un colore che "risalta" rispetto allo sfondo; in questa situazione il giocatore risulta facilmente distinguibile ed è quindi possibile determinarne con buona approssimazione l'angolo (e di conseguenza la posizione) del giocatore rispetto al robot. Se nell'ambiente di gioco sono presenti oggetti di un colore uguale (o anche molto simili) al marker, l'algoritmo, basandosi solo sul riconoscimento dei colori, non è in grado di distinguere il giocatore (che indossa il marker) dall'oggetto esterno. Di conseguenza si possono verificare due situazioni:

1. Se l'oggetto esterno entra nel campo visivo del robot, verrà scambiato per il giocatore e il robot agirà di conseguenza.

2. Se sia l'oggetto esterno che il marker si trovano nel campo visivo del robot, non si può sapere a priori quale dei due verrà identificato come giocatore.

Eseguendo delle prove si è riscontrato che un buon marker deve avere le seguenti caratteristiche:

- Un colore che risalta rispetto all'ambiente circostante

- Non deve essere composto d un materiale che rifletta eccessivamente la luce, in quanto esso apparirebbe bainco alla telecamera e ignorato (Figura 5)

- Deve essere il più possibile una figura priva di concavità in modo da risultare un unico oggetto uniforme agli occhi del robot.

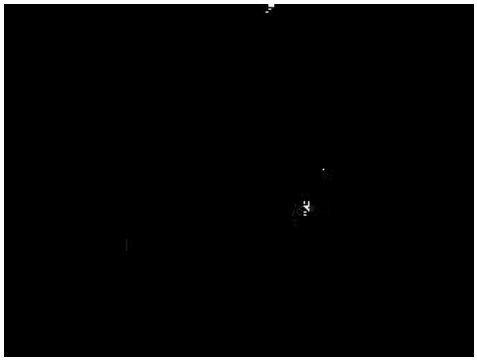
Ad esempio una maglietta colorata, o un grosso rettangolo di carta (sempre colorata) incollata sul petto del giocatore rappresentano dei buoni marker (il giubbino utilizzato in figura non è un ottimo marker).
Una limitazione imposta invece dalla telecamera utilizzata è quella di giocare in una stanza ben illuminata; questo problema potrebbe tuttavia essere risolto utilizzandone una diversa.
E' inoltre possibile che in alcune situazioni sia presente un "rumore di fondo" (Figura 6) dovuto a riflessioni da parte di oggetti prensenti nel campo

(a) Giocatore con marker

(b) Backproject

Figura 5: Come un giocatore appare al robot

(a) Immagine reale



(b) Backproject con falso positivo

Figura 6: Rumore di fondo

di gioco o, come detto precedentemente, da oggetti dello stesso colore del marker, che potrebbe provocare false letture nell'algoritmo e di conseguenza individuare il giocatore in una posizione errata.

5.1 Possibili miglioramenti

La maggior parte dei problemi sopra elencati dipendono dal fatto che ci si basa esclusivamente sul colore per identificare il giocatore. Bisognerebbe quindi modificare l'algoritmo in modo che usi l'identificazione del colore in coppia con altri metodi di analisi delle immagini in modo da poter rendere l'algoritmo più robusto.
Alcuni di questi metodi potrebbero essere:

- Utilizzare un marker di una forma ben definita e di due colori diversi

e modificare l'algoritmo in modo che identifichi un oggetto di quella particolare forma (template-matching) e che abbia esattamente quei due colori.

- Effettuare, prima di iniziare una partita, una fase di "training" del sistema di visione, durante la quale si insegna al robot a riconoscere il marker del giocatore utilizzando tecniche di template-matching.
- Un ulteriore miglioramento che si potrebbe fare è insegnare al robot a riconoscere la forma di una persona (in particolare il viso) in corrispondenza del marker, di modo da ridurre il rischio di confusione con oggetti esterni al gioco.

Riferimenti bibliografici

[1] Antonio Micali, Documentazione libreria Spykee (AirWiki – Spykee)

[2] Sito web OpenCV: http://sourceforge.net/projects/opencvlibrary/

[3] Documentazione MRT (AirWiki)

[4] Documentazione MrBrian (AirWiki)

[5] Wiiuse: http://www.wiiuse.net

A Variabili fuzzy

```
(Random NUMBER)
(GoForward NUMBER)

(SonarLeft OBJECTDISTANCE)
(SonarRight OBJECTDISTANCE)
(SonarBack OBJECTDISTANCE)
(SonarFront OBJECTDISTANCE)
(SonarNorthWest OBJECTDISTANCE)
(SonarNorthEast OBJECTDISTANCE)

(SonarLeftOld OBJECTDISTANCE)
(SonarRightOld OBJECTDISTANCE)

(ProposedTanSpeed TAN2SPEED )

(Status STATUS)
(ProposedStatus STATUS)
(EnemyDetected ENEMYDETECTED)
(EnemyAngle ENEMYANGLE)
(MovingFlag MOVINGFLAG)
(ProposedMovingFlag MOVINGFLAG)
```

B Predicati

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

IRDirClose = (OR (P IRDirCloseR) (P IRDirCloseL));
IRDirNear = (OR (P IRDirNearR) (P IRDirNearL));
IRExist = (D IRCenter EXIST);
HumanPoint = (D Hit V);
###Sonar#############################################
LeftFree = (NOT (D SonarLeft NEAR));
LeftBusy = (D SonarLeft NEAR);
```

```
LeftSemiBusy = (D SonarLeft MEDIUM);
RightFree = (NOT (D SonarRight NEAR));
RightBusy = (D SonarRight NEAR);
RightSemiBusy = (D SonarRight MEDIUM);
BackFree = (NOT (D SonarBack NEAR));
BackBusy = (D SonarBack NEAR);
BackSemiBusy = (D SonarBack MEDIUM);
FrontBusy = (D SonarFront NEAR);
NorthEastFree = (NOT (D SonarNorthEast NEAR));
NorthEastBusy = (D SonarNorthEast NEAR);
NorthEastSemiBusy = (D SonarNorthEast MEDIUM);
NorthWestFree = (NOT (D SonarNorthWest NEAR));
NorthWestBusy = (D SonarNorthWest NEAR);
NorthWestSemiBusy = (D SonarNorthWest MEDIUM);

LeftBusyOld = (NOT (D SonarLeftOld NEAR ));
RightBusyOld = (NOT (D SonarRightOld NEAR));
##### Vision - Enemy position ########################################
ViewEnemy = (D EnemyDetected YES);

EnemyFront = (AND (P ViewEnemy)(D EnemyAngle CENTER));
EnemyLeft = (AND (P ViewEnemy)(D EnemyAngle LEFT));
EnemyRight = (AND (P ViewEnemy)(D EnemyAngle RIGHT));

MovingStop = (D MovingFlag STOP);

SearchOn = (D Status SEARCH);
EscapeOn = (D Status ESCAPE);

Trapped = ( AND(P BackBusy)( AND(P LeftBusy)(P RightBusy) ) );
FightOn = (P Trapped);

VoidOnLeft = ( AND(P LeftBusyOld)(P LeftFree) );
VoidOnRight = ( AND(P RightBusyOld)(P RightFree) );
```