POLITECNICO DI MILANO
SCUOLA DI INGEGNERIA DELL'INFORMAZIONE
CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Progetto di Ingegneria Informatica

RoboTower

Documentazione

Responsabile:
Prof. Andrea Bonarini

Progetto di:
Marcello Pogliani Matricola n. 742961
Davide Tateo Matricola n. 743013

ANNO ACCADEMICO 2011-2012

Indice

1. Introduzione

Questo lavoro ha come obiettivo il progetto e l'implementazione di un gioco appartenente alla categoria degli *Highly Interactive, Competitive Robogames* ("HI-CoRG"). Questi giochi prevedono l'utilizzo di robot autonomi e contengono marcati elementi di interazione attiva tra i giocatori umani e il robot (o i robot).

Il gioco che è stato ideato, *RoboTower*, è uno strategico in tempo reale che si basa sull'idea dei "Tower defense", in cui lo scopo del giocatore è difendere un oggetto, la "torre", dall'avanzata delle armate nemiche: in questo caso, è il giocatore umano che deve ostacolare l'avanzata del robot, ad esempio posizionando opportuni oggetti nel campo di gioco.

2. Progetto del gioco

In questo capitolo verranno descritte le linee guida e i passi seguiti per progettare il gioco.

2.1 Linee guida

Innanzitutto, si è cercato di seguire alcune linee guida per lo sviluppo di un Robogame competitivo di successo, ricavate a partire da alcuni articoli sull'argomento. Un Robogame deve avere alcune caratteristiche generali:

- consistere in almeno un robot e almeno un giocatore umano in grado di interagire tra di loro cooperativamente o competitivamente
- basso costo e alta efficienza dell'uso dei componenti
- sicuro da giocare
- semplice e divertente (per chi ci gioca)
- il robot deve essere visto dal giocatore come un agente razionale
- il gioco deve essere provato sul campo
- il gioco deve coinvolgere più sensi (del giocatore... e del robot) : vista, udito, tatto (ossia colori, suoni e forme)

In aggiunta a queste, il gioco che viene implementato deve avere un obiettivo chiaro e semplice, che deve poter essere suddiviso in sotto-obiettivi (punti) nel caso sia troppo lungo. La difficoltà del gioco deve essere adatta o adattabile al giocatore, le regole devono essere semplici da capire e imparare e le azioni facili da compiere. Inoltre, perché il gioco sia coinvolgente, tutto il sistema deve reagire reagisce prontamente alle azioni dell'utente.

L'efficacia dell'interazione con il giocatore viene influenzata notevolmente da robot, che dovrebbe reagire bene (a meno di semplificazioni) al comportamento umano, e avere alcune caratteristiche:

- Ricevere input nella maniera più affidabile e credibile possibile (anche a costo di semplificazioni)
- Aspetto adatto al gioco
- Comportamento che non appare casuale, ma razionale e pensato
- Funzionamento in tempo reale

L'obiettivo generale della progettazione dei Robogame è quello di portare il gaming verso la sua naturale evoluzione verso la "fisicità", strada cominciata da Nintendo con il wii, e ormai accettata dalle maggiori case di videogiochi. I Robogames vogliono inoltre introdurre nella vita quotidiana il robot come qualcosa di "familiare" e utile, nonché diffondere l'interesse per la robotica ad un pubblico più ampio.

2.2 Considerazioni preliminari al progetto del gioco

Partendo dal presupposto che Robogame sia l'evoluzione del gaming tradizionale, abbiamo quindi cominciato a considerare i generi più usati nei videogiochi attuali:

- Strategici
- Gestionali (esclusi in ottica robogame in quanto non adatti)
- Sparatutto (esclusi in quanto fin troppo sfruttati, sia dai robot che dai videogiochi)
- Giochi di ruolo (esclusi a causa del numero limitato di robot in gioco)
- Platform (fisicamente poco realizzabili da un robot attuale)
- Sportivi

Dopo aver escluso i generi non adatti, abbiamo considerato approfonditamente i due generi rimasti, e i loro limiti. Nel caso dei giochi sportivi, il limite è dato dalla necessità di avere un robot abbastanza dinamico, mentre i giochi strategici sono limitati dalla scarsa dinamicità di azione.

Uso della palla

Una prima distinzione all'interno dei giochi delle categorie considerate riguarda l'uso della palla. I giochi con palla rendono il robot più complesso, ma sono immediati e coinvolgenti, e non pongono gravi problemi di analisi dinamica del campo di gioco. Sono certamente semplici, intuitivi e dinamici. Permettono di variare strategia, soprattutto se il gioco avviene con più agenti intelligenti in campo. I giochi senza palla necessitano di robot in generale semplici, che richiedano al più velocità discrete. Sono meno coinvolgenti (anche se dipende dai gusti) e meno dinamici, o comunque se sono dinamici danno meno spazio a strategie pensate come "razionali" (se devo scappare da qualcuno... scappo) oppure paradossalmente prevedono l'introduzione di idee complesse, come quella di nascondiglio. Un grosso problema legato all'uso della palla riguarda la velocità. Questa può essere limitata, ad esempio, utilizzando palloncini, oppure palline da tennis sgonfie da far rimbalzare.

Giochi strategici

Dei giochi strategici abbiamo considerato:

Derivati dai giochi da tavolo Rientrano in questa categoria giochi come il labirinto magico (famoso gioco da tavolo in scatola). Il labirinto magico è un labirinto, fatto ad esempio con segnali luminosi o in altro modo, che può cambiare configurazione in cui bisogna trovare degli oggetti (oppure potrebbe essere l'uscita... oppure il robot). Altri esempi di giochi da tavolo che possono essere sfruttati sono giochi come "battaglia navale".

Tower Defense Sono giochi in cui il giocatore deve difendere un obiettivo sensibile da orde di nemici deboli, oppure da un unico nemico difficile da bloccare.

Da bambini o di intelligenza giochi "infantili" possono essere trova & nascondi (simili alla caccia al tesoro), distruggi & costruisci. Un'altra idea potrebbe essere uno spy-game, in cui il robot deve ottenere informazioni (degli oggetti) e il giocatore deve catturare il robot, magari ponendo gli oggetti che il robot ricerca in zone adatte a intrappolare l'avversario.

2.3 Design del gioco

Dalle considerazioni sopra esposte sono state ricavate tre bozze di gioco: una (*RoboTower*) basata sull'idea di Tower Defense, che è poi stata effettivamente implementata, e due bozze, entrambe focalizzate sull'uso della palla. Delle bozze scartate, la prima consiste in un gioco sportivo simile al tennis, in cui il robot deve svolgere azioni limitate rispetto all'umano, la seconda consiste in un gioco ispirato ai giochi infantili e basato su un palloncino.

In generale, è preferibile pensare a giochi in cui i ruoli di giocatore e robot siano diversi (come nella bozza Tower Defense), il che permette di aggirare eventuali limitazioni del robot, se non sfruttarle a vantaggio dell'esperienza di gioco. Un altro problema, oltre a quello già citato della velocità, legato ai giochi che utilizzano la palla, è la necessità da parte del robot di effettuare movimenti complessi: la palla deve infatti essere sollevata da terra, colpita (eventualmente al volo) e direzionata, causando problemi nell'implementazione con una precisione accettabile su un robot di tali comportamenti. Per tale motivo, le bozze che prevedono l'utilizzo della palla sono state scartate.

Nello sviluppo dello storyboard di RoboTower, è stata prestata particolare attenzione a quali aspetti del gioco siano completamente controllabili dalla logica di gioco e quali invece non sono controllabili in maniera efficiente eøsufficientemente precisa, lasciando il compito di rispettare queste ultime regole alla buona fede del giocatore. Il gioco è stato progettato per essere sia facilmente espandibile, aumentando ad esempio il numero di robot o di torri da utilizzare, di conseguenza è possibile implementare versioni del gioco con più giocatori umani che collaborino con o contro più robot.

3. Descrizione del gioco

In questo capitolo vengono descritti gli elementi principali e le regole complete di *RoboTower*, e vengono dettagliati i requisiti che devono soddisfare i vari componenti del gioco.

3.1 Obiettivo

Scopo del gioco è la difesa, da parte del giocatore, di un oggetto (la "torre") dall'assalto del (o dei) robot. Il giocatore, per conquistare la vittoria, deve posizionare in maniera opportuna i vari oggetti a sua disposizione (ostacoli), e riuscire a resistere per un determinato periodo di tempo all'assalto del robot, evitando quindi la distruzione della torre.

3.2 Destinatari

Nella sua versione di base, il gioco si svolge tra un robot e un giocatore. È adatto a un target abbastanza vario, essendo rivolto a bambini e adulti tra i 12 e i 60 anni.

RoboTower può essere facilmente esteso a più giocatori sia in senso cooperativo che in senso competitivo:

- È possibile introdurre diverse torri (e diversi set degli oggetti che verranno descritti in seguito), una per ogni giocatore. Ogni giocatore controlla la sua torre e tramite gli ostacoli cerca di allontanare il robot e "convincerlo" ad attaccare le altre torri (la strategia del robot potrebbe, ad esempio, cercare di attaccare la torre più vicina)

- Si possono introdurre diversi giocatori e/o robot che cooperano rispettivamente alla difesa e alla distruzione dell'unica torre

3.3 Requisiti di base

Ambiente di gioco

Il gioco può svolgersi sia in un ambiente chiuso che all'esterno, purché l'ambiente sufficientemente grande, in modo da permettere il posizionamento degli ostacoli e il movimento del robot. In particolare sono scosigliati oggetti come sedie, gambe dei tavoli, o altri tipi di ostacoli bassi che il robot non può rilevare coi sonar ed eventuali altri sensori. E' preferibile che il campo

sia delimitato da ostacoli verticali e lisci, in modo che il robot possa rilevarli correttamente, senza sbatterci contro.

Robot

Il robot deve essere dotato di una telecamera per riconoscere gli ostacoli "fisici", la torre e le trappole (costituite da marker a cui sono collegati contenuti concettuali). Sulla velocità non ci sono particolari requisiti, tuttavia deve essere sufficiente a rendere – in una certa misura – dinamico il gioco.

Il robot deve comunque disporre di un collegamento wireless per comunicare dati al computer e, nel caso non sia dotato di un processore sufficientemente potente per effettuare il riconoscimento delle immagini e gestire la logica di gioco, ricevere i comandi destinati agli attuatori.

Per questi motivi, è possibile utilizzare sia robot commerciali, come lo Spykee della Meccano, che un robot appositamente costruito allo scopo.

Per rendere più coinvolgente e realistico il gioco, è preferibile - anche se non indispensabile - che il robot sia dotato di una o più armi (un lanciatore di palle, ad esempio) per colpire gli oggetti presenti nel gioco.

Computer

Un computer svolge la funzione di arbitro e segnapunti. Viene utilizzato per dare inizio al gioco, calcolare i crediti del giocatore, e visualizzare lo stato di attivazione delle trappole. Inoltre, fornisce al giocatore le informazioni sullo stato del gioco (ad esempio il tempo che rimane alla fine del round, i crediti di gioco acquisiti) e alcune statistiche (il numero di partite giocate, il numero di vittorie e sconfitte, . . .).

3.4 Oggetti coinvolti

Oltre al robot e al computer, già descritti nella sezione precedente, lo svolgimento del gioco prevede la presenza di altri oggetti: la torre, gli ostacoli, e le fabbriche.

Torre

La torre è un oggetto di un colore uniforme e particolare (ad esempio un tubo di cartone rosso). Il colore della torre deve essere differente da quello degli altri oggetti presenti nel campo di gioco, in modo che il robot possa riconoscerla tramite una telecamera, ed eventualmente anche calcolare la distanza a cui si trova. Inoltre la torre è dotata di un dispositivo di segnalazione radio, per indicare al robot l'avvenuto abbattimento della torre.

La torre - realizzata in un materiale sufficientemente leggero - dev'essere caricata dal robot e fisicamente abbattuta urtandola.

Fabbriche

Come le torri, anche le fabbriche sono oggetti di un colore uniforme e differente da quello di altri oggetti presenti nel campo di gioco (ad esempio, tubi di cartone giallo). Anche le fabbriche devono essere riconosciute e distrutte dal robot, e devono poter segnalare l'avvenuta distruzione.

Tutte le fabbriche vengono posizionate nel campo di gioco all'inizio della partita, e ogni fabbrica non ancora distrutta produce costantemente *crediti di gioco*.

Ostacoli

All'interno del gioco, vengono utilizzati due tipologie di ostacoli: gli ostacoli fisici e le trappole.

- Gli *ostacoli fisici* rappresentano ostacoli insormontabili (come torrette difensive, barriere, montagne, mura). Sono una serie di oggetti a disposizione del giocatore che il robot non può spostare né abbattere, ma solo evitare. A differenza delle trappole, e come per le fabbriche, possono essere posizionati solo a inizio partita.

- Le *trappole* sono ostacoli che modificano il comportamento del robot. Quando il robot passa sopra a questi ostacoli, svolge delle azioni differenti a seconda della trappola in cui è incappato, ad esempio:

  - il robot viene obbligato a compiere un particolare movimento, ad esempio girare a destra o a sinistra, oppure tornare indietro (la direzione deve essere mantenuta per alcuni secondi)
  - il robot viene bloccato completamente per un certo numero di secondi
  - alcune funzionalità del robot vengono temporaneamente disabilitate o modificate per un certo periodo. Il robot può perdere la vista, diminuire la velocità o la capacità di cambiare direzione.

  Le trappole possono avere un effetto ben noto al giocatore, oppure possono avere effetti casuali, e anche negativi nei confronti del giocatore, come l'aumento di velocità del robot, oppure dell'energia del robot (ossia la durata massima del round).

3.5 Regole del gioco

Il gioco è strutturato in vari *round*, ognuno dei quali può essere vinto dal robot oppure dal giocatore. Al termine della partita viene dichiarato vincitore chi, tra il robot e il giocatore, vince il maggior numero di round. Inoltre il giocatore può misurare le sue prestazioni, e confrontarle con quelle di altri giocatori, acquisendo crediti di gioco.

Ogni round ha una durata costante (circa 5 minuti). In caso di problemi al robot (ad esempio se il robot cade a causa di ostacoli non idonei sul campo di gioco), può essere fermato il tempo. Inoltre, la durata del round può essere eventualmente modificata dalle trappole.

All'inizio del gioco, il robot è posizionato nella propria base. Gli ostacoli sono posizionati in un luogo (il "deposito ostacoli") posto, a discrezione del giocatore, a una certa distanza dall'area in cui verrà posizionata la torre, per rendere il gioco più (o meno) dinamico.

Preparazione

Il giocatore, prima dell'inizio di ogni round, posiziona la torre, poi dà avvio al gioco e posiziona gli ostacoli fisici e le fabbriche.

A seguito dell'avvio del gioco, dopo 30 secondi, il robot comincia a dirigersi verso la torre, cercando di evitare gli ostacoli.

Svolgimento

Durante lo svolgimento del gioco, il giocatore può posizionare le trappole (che possono essere costituite, ad esempio, da carte di plastica), cercando di evitare l'avanzata del robot. Il robot cerca di dirigersi verso la torre e abbatterla, evitando gli ostacoli fisici e rispettando gli ordini di quelli concettuali.

Il robot è dotato di una certa quantità di *energia*. L'energia residua del robot è rappresentata dal tempo residuo del round, terminato il quale il robot si disattiva e non è più in grado di continuare, consegnando la vittoria al giocatore.

All'inizio del round il giocatore avrà a disposizione l'intero mazzo di carte trappola. Una volta attivata la trappola, essa viene disabilitata e sarà necessario attendere un tempo di rigenerazione, che dipende dalla quantità di fabbriche sul campo attive sul campo di gioco: più fabbriche sono attive, più le trappole si rigeneranno velocemente.

Conclusione

Il round termina quando il robot esaurisce tutta la sua energia oppure riesce ad abbattere la torre. In particolare, il giocatore vince il round quando il robot esaurisce tutta la sua energia, ossia finisce il tempo senza che venga abbattuta la torre, mentre il robot vince il round se riesce ad abbattere la torre prima del termine del tempo. Al termine di ogni round gli ostacoli vengono riportati nel deposito ostacoli. I *crediti di gioco* guadagnati, che misurano le prestazioni del giocatore nel difendere torri e fabbriche il pià a lungo possibile, andranno a sommarsi all punteggio del giocatore, sia che il giocatore vinca la partita, sia che sia stato sconfitto dal robot. A questo punto, il giocatore è pronto per iniziare un altro round.

Nota: Il gioco qui presentato può essere espanso in varie direzioni. Ad esempio, può essere articolato in diversi livelli di difficoltà, scelti dal giocatore all'inizio della partita, oppure selezionati automaticamente all'inizio di ogni round in base al punteggio raggiunto dal giocatore. Il livello di difficoltà può essere utilizzato sia per variare i comportamenti assunti dal robot durante il gioco, che per variare i parametri definiti dalle regole (l'energia del robot, il tempo di ricarica delle trappole, ...).

3.6 Scenari di gioco

In questo paragrafo viene riassunto lo schema generale di un round del gioco, specificati ulteriori dettagli del funzionamento, e presentati alcuni scenari d'uso.

1. Il giocatore posiziona il robot e la torre nella stanza
2. Il giocatore sceglie gli ostacoli fisici da usare e da avvio al gioco
3. Il giocatore piazza gli ostacoli fisici fino a che un segnale lo avvisa che la partita comincia.
4. Il robot si mette in movimento verso la torre nel tentativo di abbatterla.
5. Il giocatore può posizionare le trappole dove e quando vuole. La trappola avrà effetto solo se attiva.
6. Il robot che cada in una trappola, subirà gli effetti imposti da essa, che possono essere:
   (a) Perdere energia vitale (si accorcia il suo "time to live")
   (b) Aumentare la sua energia vitale (si allunga il suo "time to live")
   (c) Venire immobilizzato per alcuni secondi
   (d) Non potere vedere nulla nel suo campo visivo per alcuni secondi
   (e) Subire il blocco di un cingolo per alcuni secondi
   (f) Compiere una retromarcia obbligata
7. Il robot può, se lo ritiene opportuno, distruggere le fabbriche, e diminuire la produzione di crediti del giocatore. Inoltre un numero minore di fabbriche comporta una velocità di riattivazione delle carte minore.
8. Se il robot distrugge la torre, ha vinto il round.
9. Se il giocatore resiste all'assalto del robot fino a far consumare la sua energia vitale completamente (il suo "Time to live") vince il round.
10. Se il robot si trova in impossibilità di muoversi o ha qualche problema hardware, il gioco deve essere messo in pausa e si deve ovviare al problema prima di riprendere.

Alcune regole fondamentali del gioco risultano impossibili da verificare da parte della logica di gioco, o quantomeno risultano impossibili da verificare in maniera efficiente. Il rispetto di queste regole da parte del giocatore è pertanto basato sulla fiducia nei suoi confronti:

- Il giocatore non può colpire il robot.
- Il giocatore non può chiudere, con le trappole fisiche, ogni percorso che il robot potrebbe seguire per raggiungere la torre.
- Il giocatore non può spostare trappole fisiche posizionate in precedenza, ne posizionare trappole fisiche una volta che il robot sia in movimento.
- Il giocatore non può frapporsi tra il robot e il suo obbiettivo.

4. Architettura hardware

In questo capitolo viene descritta in dettaglio la componentistica hardware utilizzata per l'implementazione del gioco.

4.1 Unità di elaborazione

L'elaborazione dei dati trasmessi dal robot e la definizione dei comportamenti in base alle logiche di gioco sono gestiti da un elaboratore esterno, che viene anche utilizzato dal giocatore per controllare lo stato del gioco (punteggio, tempo rimanente, ...) e gestirne l'avvio.

4.2 Il robot

L'attore principale del gioco è il robot autonomo *Spykee*, un modello commerciale della Meccano [2], disponibile nel laboratorio di Intelligenza Artificiale e Robotica (AIRLab) del Politecnico di Milano. Questo modello è già stato utilizzato con successo in precedenti progetti nell'ambito dei Robogame[1].

Spykee si muove tramite due cingoli (con tecnica differential drive). Una telecamera montata sulla testa permette di catturare immagini fortemente compresse in formato JPEG con una risoluzione di 320×240 pixel a una frequenza di circa 20 frame al secondo. Per comunicare con il computer che effettua il controllo, viene utilizzata una rete Wi-Fi. Delle varie modalità di connessione messe a disposizione dal robot, viene utilizzata la modalità "local ad hoc mode": il robot crea all'accensione una rete ad-hoc a cui il computer deve essere connesso.

Aggiunte hardware

A partire da quanto già realizzato nei precedenti progetti, Spykee è stato dotato di ulteriori sensori che lo rendono adatto all'utilizzo all'interno del gioco. Tutti i componenti hardware aggiunti al robot sono controllati da un microcontrollore. A questo scopo è stata utilizzata una scheda STM32F4 Discovery Board della ST [10], dotata di un processore ARM Cortex M4, scelta per le sue caratteristiche: in particolare il grande numero di periferiche (GPIO, timer, interfacce seriali) utilizzabili.

Il firmware realizzato per controllare i vari dispositivi è stato sviluppato utilizzando il sistema operativo ChibiOS/RT [3]. L'utilizzo di un sistema

[1] in particolare nelle varie versioni di Robowii [5]

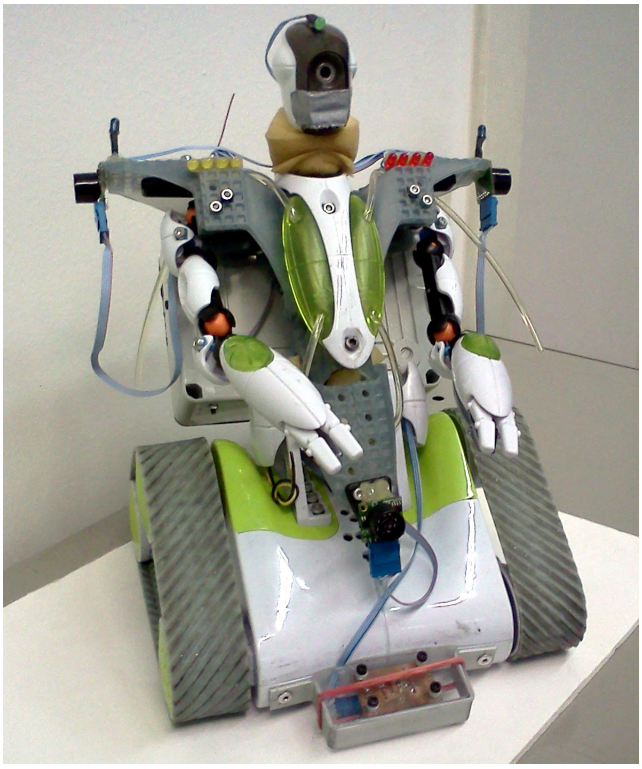

Figura 4.1: *Il robot Spykee*

operativo per microcontrollori permette sia di utilizzare concetti di programmazione concorrente quali thread e mutex, che di astrarre l'hardware sottostante, consentendo quindi una certa modularità (ottenuta isolando ogni funzione in un thread indipendente) e quindi un rapido sviluppo del firmware.

La scheda comunica con l'unità di elaborazione tramite un collegamento wireless di tipo Zigbee (utilizzando un modulo XBee della Digi). Il protocollo è caratterizzato da una bassa velocità di trasmissione (comunque ampiamente sufficiente per gli scopi del progetto), ma da una buona facilità di utilizzo, specialmente in ambito embedded: nella modalità più semplice viene infatti utilizzato come un collegamento seriale punto-punto tra microcontrollore e computer.

Tutti i vari componenti sono state raccolti in una apposita scatola e sono alimentati direttamente dal pacco batteria di Spykee tramite un regolatore di tensione, che permette di ridurre la tensione proveniente dalla batteria (circa 9 V) a quella di 5 V. Un apposito interruttore, posto a valle dell'interruttore di alimentazione di Spykee, permette di spegnere o accendere le aggiunte hardware[2].

[2]Durante la carica delle batterie, si consiglia di tenere spente le aggiunte hardware

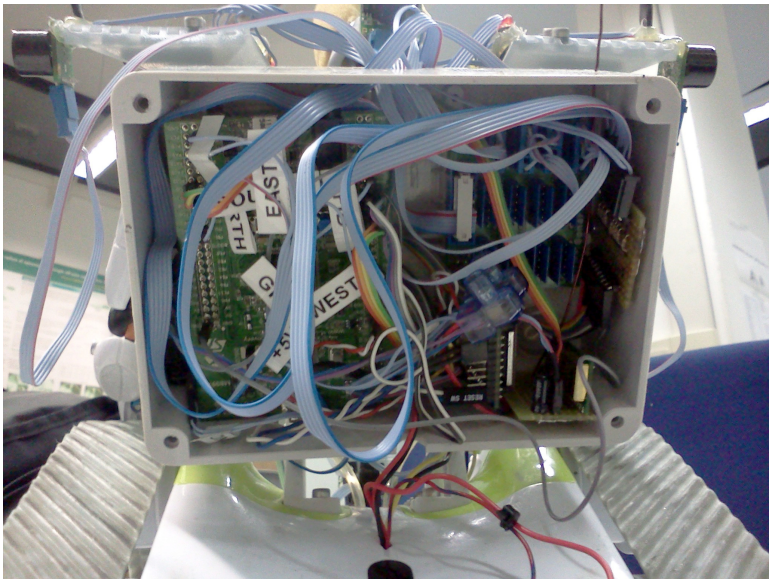

Figura 4.2: *La scatola con i componenti che sono stati aggiunti al robot*

Sonar Per rilevare, e quindi evitare, eventuali ostacoli incontrati durante il movimento del robot, sono stati montati quattro sonar MaxSonar®-EZ della MaxBotix, uno per ognuno dei punti cardinali (nord, sud, ovest, est). L'aggiunta di un certo numero di sensori è necessaria in quanto il robot non è progettato per muoversi autonomamente, ma soltanto per ricevere comandi impartiti manualmente.

Led Spykee è stato dotato di due strisce di LED (quattro LED gialli e quattro rossi) e di un LED verde che permettono di mostrare alcune informazioni relative allo stato del robot e/o del gioco nel complesso. Sul robot sono anche presenti due led a infrarossi, che venivano utilizzati insieme a un controller Wiimote in precedenti progetti, ma che non sono di interesse per questo gioco.

Le altre aggiunte hardware sono strettamente collegate ad altri componenti del gioco, pertanto verranno descritte nei paragrafi successivi.

4.3 Ostacoli attivi

Per quanto riguarda l'implementazione delle *trappole* (gli ostacoli che modificano il comportamento del robot) sono state valutate diverse soluzioni.

Poiché nel gioco viene già utilizzata una telecamera per poter riconoscere alcuni oggetti, inizialmente si è pensato di implementare anche le trappole utilizzando meccanismi di visione artificiale. In particolare, sono stati presi in considerazione vari tipi di marker bidimensionali, in particolare i "Data

Figura 4.3: *Spykee, con due fabbriche, una torre e alcune delle carte trappola (tag RFID)*

Matrix" e i tag della libreria "ARToolkitPlus"[3]. Purtroppo, tutti questi meccanismi hanno dato scarsi risultati in termini di velocità oppure di qualità del riconoscimento. I difetti riscontrati sono, ad esempio, un'enorme dipendenza dalle condizioni di luce, scarsi risultati in movimento, dovuti anche alla lentezza degli algoritmi (specialmente nel caso dei Data Matrix), e alle pessime condizioni delle immagini catturate. Per questi motivi, le soluzioni basate sul riconoscimento di tag mediante visione sono state scartate.

Una soluzione che ha fornito prestazioni migliori, e che è stata adottata per l'implementazione del gioco, è l'uso di tag RFID passivi a 125 KHz, che tuttavia richiede l'aggiunta di ulteriore hardware. I tag vengono correttamente riconosciuti quando il robot passa sopra di essi, con un margine di errore accettabile. Inoltre, il loro formato (tessere di plastica della dimensione di una carta di credito, che eventualmente l'utente può facilmente identificare con delle etichette) li rende abbastanza comodi per l'utilizzo da parte del giocatore. Per la lettura dei tag, è stato montato sul robot il lettore ID-12 della ID-Innovations. Il lettore trasmette alla scheda di controllo presente sul robot i dati (id del tag letto più un checksum) in formato testuale tramite un collegamento seriale a 9600 bps. Una limitazione di questo specifico modello di lettore è la presenza di un'antenna interna, che ne limita pesantemente il posizionamento.

4.4 Torri e fabbriche

Le torri e le fabbriche sono costituite semplicemente da tubi di PVC leggero di diametro 100 mm. Le fabbriche, di altezza 45 cm, sono state verniciate di giallo, mentre le torri, alte 60 cm, sono di colore rosso.

La base dei cilindri è costituita da un tappo su cui è montato un interruttore che risulta chiuso quando il tubo è appoggiato a terra, e viene aperto quando il cilindro viene abbattuto. L'apertura dell'interruttore attiva un trasmettitore (del tipo di quelli utilizzati per comandare i cancelli elettrici) che

[3] A differenza dei codici Data Matrix, che sono pensati per contenere informazioni arbitrarie, ad esempio URL oppure informazioni di spedizione, i tag ARToolkitPlus codificano solo un id, e sono pensati per applicazioni di identificazione, localizzazione, ... (anche in presenza di distorsioni nell'immagine o altre condizioni non ottimali).

invia un segnale radio a un ricevitore posto sul robot. Quest'ultimo segnala l'evento all'unità di elaborazione.

Il ricevitore montato sul robot è il RX-4M-HCS prodotto dall'Aurel, ed è dotato di quattro canali separati con uscita open-drain (corrispondenti ai quattro pulsanti presenti sui corrispondenti trasmettitori, sempre della stessa marca). Pertanto, si sono impostati i relativi ingressi del microcontrollore come ingressi pull-up. Le uscite del ricevitore sono state impostate in modalità monostabile: in questo modo, ogni volta che una torre o una fabbrica cade, viene inviato un segnale al microcontrollore presente sul robot, e quindi all'unità di elaborazione.

5. Architettura software

Per semplificare l'implementazione e agevolare il riuso del codice esistente, il software è stato realizzato utilizzando ROS (Robot Operating System) [1]. Si tratta di un middleware pensato per applicazioni robotiche costituite da un elevato numero di moduli, relativamente indipendenti, che interagiscono tra loro.

Un sistema realizzato con ROS è suddiviso in un certo numero di *nodi*, ovvero semplici processi[1] indipendenti, che vengono eseguiti in parallelo e svolgono ciascuno una funzione elementare. Per migliorare l'organizzazione (e la distribuzione) del codice sorgente, i nodi possono poi essere raggruppati in *package*. A loro volta, un gruppo di package può formare uno *stack*. ROS mette a disposizione due differenti tecniche con cui i nodi possono comunicare tra loro:

- un paradigma "publish-subscribe": i nodi pubblicano (*publish*) messaggi su un certo argomento (*topic*). I destinatari della comunicazione sono i nodi iscritti (*subscribe*) a quel topic.

- l'invocazione di *servizi* esposti da altri nodi, con la possibilità di ricevere una risposta, secondo una semantica simile a quella di una chiamata di procedura remota (RPC)

Oltre a funzionare come middleware di comunicazione interprocesso, ROS mette a disposizione alcuni programmi grafici e testuali, e funzioni di libreria che forniscono funzioni utili allo sviluppo di applicazioni robotiche.

Il sistema da noi realizzato risulta quindi composto da vari nodi. Ciascuno di essi è contenuto in un package omonimo all'interno della directory principale del progetto. In figura 5.1 è rappresentata l'architettura generale del sistema: i vari nodi e le dipendenze dalle librerie esterne a ROS. Le frecce rappresentano la comunicazione tra nodi (messaggi scambiati e servizi invocati) e i canali di comunicazione tra il software presente sull'unità di elaborazione e il robot.

5.1 SpyKee

SpyKee si occupa di interfacciare l'unità di elaborazione con le funzioni del robot che comunicano via Wi-Fi: permette di fornire i comandi ai cingoli e di ricevere le immagini catturate dalla telecamera. Questo nodo è l'adattamento a

[1] attualmente ROS supporta C++ e Python

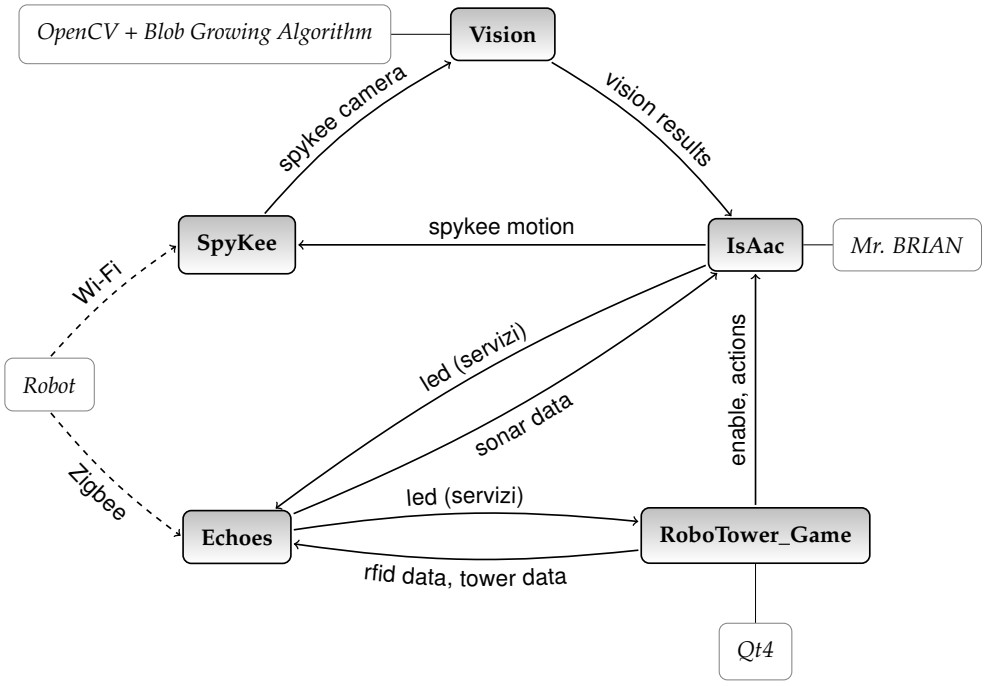

Figura 5.1: *Struttura generale del sistema*

ROS di una libreria realizzata nei precedenti progetti, e ottenuta analizzando il codice sorgente del firmware di SpyKee, disponibile sul sito [2].

SpyKee pubblica messaggi di tipo `sensors_msgs::CompressedImage` sull'argomento `spykee_camera`, che contengono le immagini ricevute dalla telecamera compresse in formato JPEG e messaggi di tipo `std_msgs::Int8` sull'argomento `spykee_battery` contenenti gli aggiornamenti sullo stato della carica della batteria del robot (i valori sono negativi se il robot è in carica). Inoltre sottoscrive messaggi di tipo `SpyKee::Motion` sull'argomento `spykee_motion`, contenenti i comandi da inviare ai cingoli. Tali comandi sono costituiti da due interi compresi tra -90 e 90, che rappresentano la velocità tangenziale e angolare del robot, e vengono convertiti dal nodo nei corrispondenti comandi al cingolo destro e sinistro.

5.2 Echoes

Echoes comunica via Zigbee con l'hardware aggiunto a posteriori al robot: sonar, led, lettore RFID, e ricevitori dei comandi inviati dagli interruttori posti sulle torri e sulle fabbriche.

Il nodo riceve i dati dalla porta seriale[2], analizza le stringhe ricevute, e

[2]di default viene utilizzato il device `/dev/ttyUSB0`, altrimenti il percorso deve essere fornito da riga di comando come primo argomento

pubblica i messaggi contenenti i dati rilevati:

- i messaggi di tipo `Echoes::Sonar` sull'argomento `sonar_data` contengono i dati ricevuti dai sonar (valori delle distanze dei quattro sonar montati sul robot, espresse in millimetri).
- i messaggi di tipo `Echoes::Rfid` sull'argomento `rfid_data` contengono semplicemente una stringa identificativa del tag RFID rilevato
- i messaggi di tipo `Echoes::Tower` sull'argomento `towers_data` contengono un intero che corrisponde all'id della torre (o della fabbrica) abbattuta

Inoltre, il nodo espone alcuni servizi che permettono di comandare i led presenti sul robot (`green_led`, `yellow_led` e `red_led`), e un ulteriore servizio che consente di spegnere tutti i led (`reset_led`). I led possono essere impostati nello stato di acceso, spento o lampeggiante. In particolare, per i led gialli e i led rossi, lo stato lampeggiante viene definito a livello dell'intero gruppo di led.

A causa dell'alto rumore presente nei valori provenienti dai sonar, il valore x_k che viene pubblicato sul topic `sonar_data` è filtrato attraverso un filtro a media mobile esponenziale. Questo significa che il dato pubblicato all'arrivo del k-esimo campione x_{cur} registrato dal sonar è dato da

$$x_k = \alpha x_{\text{cur}} + (1 - \alpha) x_{k-1}$$

dove $\alpha = 0.3$, valore sufficientemente basso da ridurre il rumore ad alta frequenza presente nel segnale, ma comunque abbastanza alto da rendere l'aggiornamento dei valori reattivo, e riuscire ad evitare efficacemente gli ostacoli.

5.3 Vision: identificazione degli oggetti

Vision si occupa di analizzare le immagini provenienti dalla telecamera di Spykee, per rilevare la presenza di una torre o di una fabbrica.

Il nodo riceve da *SpyKee* i messaggi contenenti le immagini e, analizzata l'immagine, pubblica un messaggio di tipo `Vision::Results` sull'argomento `vision_results`. Questo messaggio contiene i dati riguardanti gli oggetti trovati: la posizione rispetto al centro dell'immagine, una stima della distanza in centimetri, e l'altezza e la larghezza del blob in pixel.

Il cuore del nodo è un algoritmo, già utilizzato nel progetto [8], che si occupa di identificare all'interno dell'immagine dei blob sufficientemente uniformi e di colore "simile" a quello degli oggetti che si stanno cercando. L'analisi viene effettuata basandosi su un algoritmo di tipo KNN, che necessita di una prima fase di addestramento. Al termine di questa fase, viene generato un classificatore (contenuto in un file `.kcc`), tramite il quale l'algoritmo è

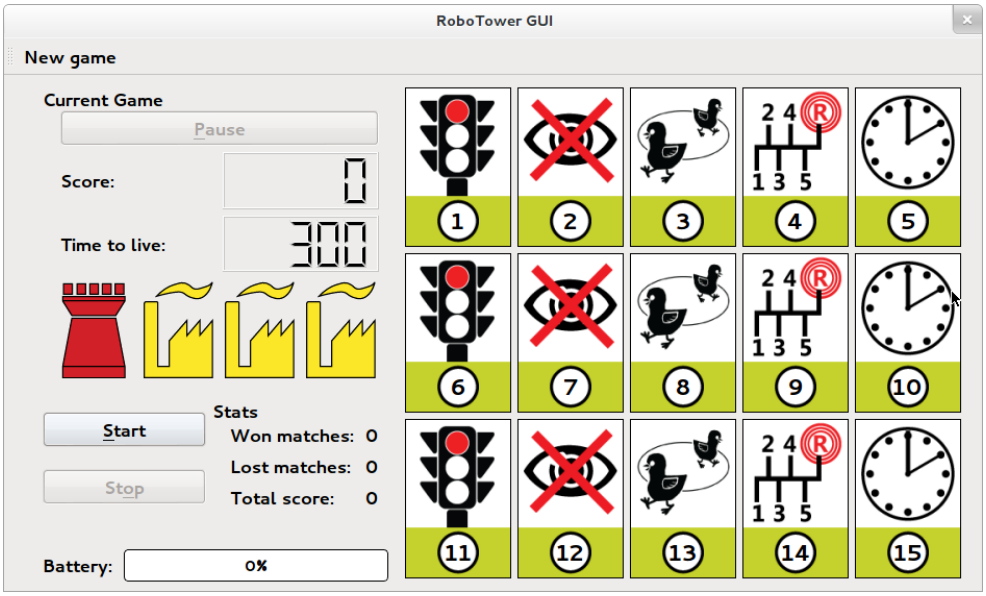

Figura 5.2: *Schermata principale dell'interfaccia di controllo del gioco*

in grado di associare ad ogni pixel dell'immagine una classe di identificazione. Le classi riguardano i pixel di colore simile a quello di una torre (classe `R`), oppure di una fabbrica (classe `G`). Una volta effettuata la classificazione dei pixel dell'immagine, l'algoritmo rileva all'interno dell'immagine i blob di interesse, e quindi stabilisce se è presente una torre o una fabbrica.

Il classificatore viene generato a partire da un file che contiene semplicemente un elenco di valori `BGR` di pixel che si considerano del colore cercato. Per generare questo file è possibile utilizzare il nodo *LittleEndian*: questo nodo carica le immagini da file oppure le riceve direttamente da *SpyKee*, e consente di evidenziare nell'immagine le aree che corrispondono agli oggetti cercati. Alla fine, LittleEndian genera un file `.dts` che contiene l'insieme dei valori. Per generare il classificatore, è sufficiente lanciare una volta *Vision* indicando come parametro `-L <filename.dts>`.

Nota: È necessario prestare particolare attenzione nel training del classificatore, in quanto è una fase critica per il corretto funzionamento del gioco e il corretto riconoscimento degli oggetti.

5.4 RoboTower_Game: la logica di gioco

RoboTower_Game gestisce la logica ad "alto livello" del gioco e la comunicazione con l'utente tramite un'apposita interfaccia grafica, realizzata con le librerie Qt [4]. Si occupa di avviare e arrestare le partite, gestire lo stato del gioco, contare i punti, e tenere alcune semplici statistiche. Il nodo interagisce con gli altri processi:

- avviando e fermando il comportamento di basso livello del robot, a seconda dello stato corrente del gioco (pubblicando messaggi di tipo `std_msgs::Bool` sull'argomento `isaac_enable`)
- ricevendo da `Echoes` gli ID dei tag RFID letti e le informazioni riguardanti eventuali abbattimenti di torri o fabbriche
- pubblicando messaggi di tipo `std_msgs::String` relativi alle azioni speciali che devono essere eseguite dal robot (topic `rfid_actions`), come spiegato nel seguito
- invocando i servizi relativi all'accensione e allo spegnimento dei led relativi al punteggio

```
<robotower>
      <config>
            <time timetolive="300" setuptime="30" />
            <points tower="100" factory="30" />
            <goals towerid="4" factories="3" />
            <recharge basic="3" factory="2"/>
      </config>

      <rfid>
            <action name="lock_all">
                  <tag id="4400F56CD1" num="1" />
                  <tag id="4400F59195" num="2" />
                  <tag id="4400F58B6D" num="3" />
            </action>

            <action name="disable_vision">
                  <tag id="4400BDB1D9" num="4" />
                  <tag id="4400BDC253" num="5" />
                  <tag id="4B00DA3279" num="6" />
            </action>
      </rfid>
</robotower>
```

Figura 5.3: *Il file di configurazione*

La maggior parte delle impostazioni di RoboTower_Game possono essere configurate modificando il file `robotower.xml`, che si trova nella directory `RoboTower_Game`. Un esempio di file di configurazione corretto è riportato in figura 5.3[3]. Nel file è possibile specificare:

[3]per semplicità sono stati omessi alcuni tag RFID e le relative azioni

- I tempi del gioco (tag `<time>`): l'attributo `timetolive` permette di modificare la durata massima di un round del gioco, mentre l'attributo `setuptime` controlla il tempo che deve trascorrere dall'inizio della partita (pressione del pulsante "start") e l'inizio del movimento del robot

- I parametri con cui viene calcolato il punteggio (tag `<points>`): durante la partita, ogni secondo viene aggiunto un certo numero di punti per ogni torre e fabbrica che non sono stati ancora distrutti, definiti dagli attributi `tower` e `factory`

- I parametri relativi a torre e fabbriche (tag `<goals>`): in particolare, il numero di fabbriche (`factories`) e l'id dell'obiettivo che dev'essere considerato come torre (`towerid`)

- I parametri relativi al tempo di ricarica delle trappole (tag `<recharge>`). Le trappole vengono completamente ricaricate dopo

  $$\frac{100}{\texttt{basic} + \texttt{factories} \cdot \text{numero fabbriche attive}}$$

  secondi da quando vengono attivate.

Il nodo si occupa inoltre di gestire i comportamenti "speciali" del robot, collegati alle trappole (tag RFID). Per ogni trappola, è necessario specificare, all'interno del blocco `<action>` relativo all'azione associata, l'id e un numero (univoco) che viene utilizzato per identificarlo nell'interfaccia grafica. Le azioni che possono essere specificate (parametri `name`) sono:

- `lock_all`: blocca per 5 secondi i motori del robot

- `force_rotate`: costringe per 5 secondi il robot a ruotare su se stesso, inibendo i comandi diretti a uno dei due cingoli

- `disable_vision`: disabilita la visione del robot per 5 secondi

- `go_back`: blocca per 5 secondi l'avanzamento del robot, costringendolo a tornare indietro

- `modify_time`: modifica casualmente il tempo rimanente alla fine del round (sia in positivo che in negativo)

Una volta che viene letto un tag, questo viene disabilitato. I tag vengono ricaricati, uno alla volta, dopo un periodo di tempo che dipende dai parametri di configurazione impostati.

5.5 IsAac: il comportamento del robot

IsAac si occupa di controllare il comportamento di "basso livello" del robot durante il gioco. In base ai dati provenienti dai sensori, gestisce il comportamento del robot, impostando i set-point per i cingoli e controllando l'accensione di alcuni led.

Il nodo riceve i messaggi pubblicati da *Echoes* e *Vision*, e invia comandi a *SpyKee* (messaggi `Spykee::Motion` sull'argomento `spykee_motion`) riguardanti il controllo dei cingoli. Inoltre invoca il servizio `led_data` esposto da Echoes per il controllo dei led gialli e del led verde posti sul robot. I led gialli lampeggiano quando *IsAac* è attivo, sono spenti quando non è attivo, e rimangono accesi quando il robot è bloccato in una trappola. Il led verde lampeggia quando viene rilevata una torre o una fabbrica, altrimenti è spento.

IsAac riceve da *RoboTower_Game* i comandi di attivazione e disattivazione, e le informazioni riguardanti il blocco del robot in una trappola (quando l'azione a cui è associata richiede la modifica dei dati provenienti dai sensori oppure dei comandi diretti agli attuatori, in caso contrario l'azione è implementata direttamente in *RoboTower_Game*).

Per l'implementazione di questo nodo, è stata sfruttata la libreria Mr. Brian, sviluppata dal Politecnico di Milano nel contesto di di MRT (Modular Robotic Toolkit) [7]. Questa libreria consente di definire il comportamento da applicare al robot come un insieme di regole scritte in *logica fuzzy*, utilizzando anche predicati che risultano dalla fuzzyficazione degli ingressi. La configurazione di Brian è basata su un insieme di file di testo: in questo modo è possibile modificare totalmente i comportamenti senza bisogno di ricompilare il codice sorgente. I file di configurazione, presenti nella cartella `IsAac/config`, sono:

- `behaviour.txt` contiene l'elenco, suddiviso per livelli, dei comportamenti (regole) che sono stati definiti. Ogni comportamento, costituito da un insieme di regole fuzzy, è contenuto in un file `.rul`

- `ctof.txt` associa ad ogni dato crisp in ingresso un'insieme fuzzy, che verrà utilizzato per la fase di fuzzyficazione. Gli insiemi fuzzy sono definiti nel file `shape_ctof.txt`.

- `s_ftoc.txt` definisce i valori in uscita da Brian, associandoli agli insiemi fuzzy da utilizzare per la defuzzyficazione. Gli insiemi sono definiti nel file `s_shape.txt`

- `Predicate.ini` definisce i predicati fuzzy sulla base di variabili fuzzy e/o di altri predicati

- `Cando.ini` definisce le condizioni di attivazione per un determinato comportamento (le condizioni necessarie per cui eseguire quel comportamento sia sensato)

- `Want.ini` definisce le condizioni per cui è opportuno attivare un determinato comportamento

La suddivisione in livelli nel file `behaviour.txt` permette di utilizzare come ingresso dati in uscita dai livelli precedenti, ed eventualmente cancellare anche le uscite impostate dai livelli precedenti. Nelle regole, infatti, è possibile utilizzare anche i predicati, definiti nel file `PredicateActions.ini`, in cui è possibile far riferimento ai dati in uscita dai livelli precedenti a quello corrente utilizzando etichette della forma

```
Proposed<nome del dato in uscita>
```

dopo averle associate ad opportuni insiemi fuzzy nel file `ctof.txt`. La suddivisione in livelli permette di rendere maggiormente modulari le regole fuzzy. All'interno del manuale di MRT [6] è presente la descrizione completa del funzionamento di Mr. Brian e della sintassi dei file di configurazione.

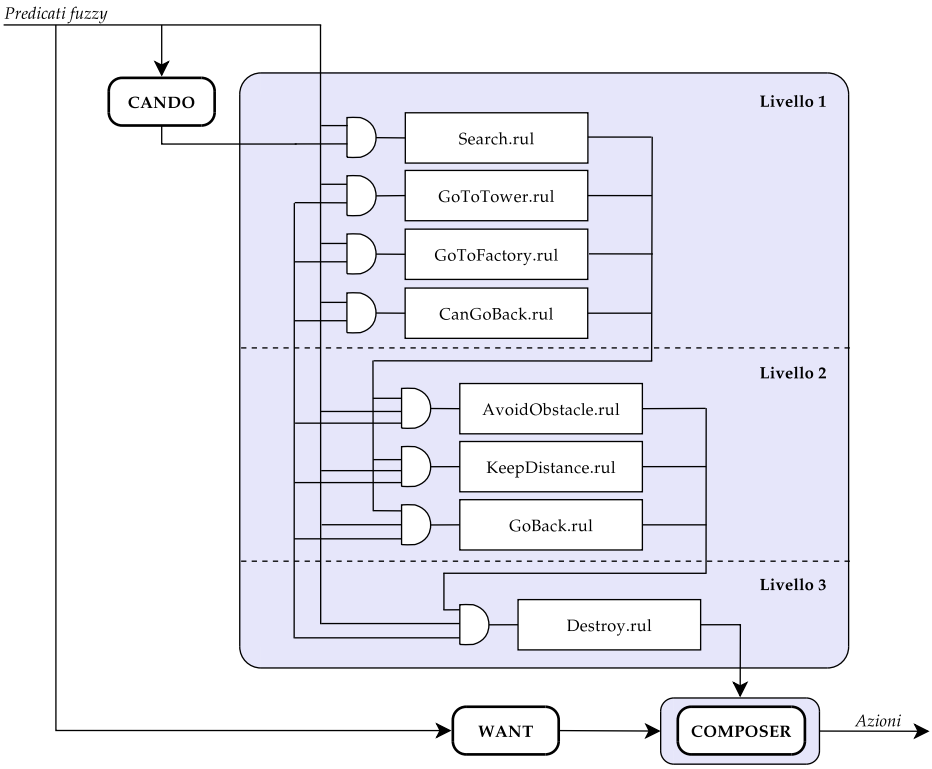

Figura 5.4: *I comportamenti utilizzati da IsAac*

Sono stati implementati vari comportamenti, che permettono al robot di:

- Cercare casualmente la torre o le fabbriche (*Search*)
- Raggiungere la torre o la fabbrica, una volta che è stata trovata (*GoToTower* e *GoToFactory*)

- Evitare gli ostacoli: in particolare, *AvoidObstacle* serve per evitare ostacoli posti frontalmente, *KeepDistance* evita che il robot colpisca ostacoli al lato, mentre *GoBack* è una regola utilizzata in situazioni di emergenza, dove avanzare frontalmente non porterebbe a uscire da una situazione di blocco
- Distruggere la torre o una fabbrica (*Destroy*)

Inoltre, la regola *CanGoBack* permette di ottenere in uscita un valore che indica, in base ai valori ricevuti dal sonar posteriore, se il robot può o meno muoversi all'indietro senza andare a sbattere. Questo valore viene utilizzato da IsAac durante l'attivazione dell'azione `go_back` associata ad alcune trappole, in modo da fermare il robot se non è possibile tornare indietro.

6. Conclusioni

Da alcuni test che abbiamo effettuato, *RoboTower* risulta giocabile e coinvolgente per il giocatore. Variando il numero e la dimensione degli ostacoli fissi, oppure agendo sui parametri di configurazione, quali la durata del singolo round oppure il tempo di ricarica delle carte, è possibile rendere il gioco più o meno difficile, e quindi adattarlo al giocatore. Agendo sulla dimensione del campo di gioco, invece, è possibile porre maggiormente l'accento sull'aspetto strategico del gioco (posizionamento corretto degli ostacoli, delle torri e delle fabbriche all'inizio del gioco), oppure su quello dinamico (posizionamento delle trappole durante l'avanzata del robot). Perché il gioco funzioni correttamente, comunque, il campo di gioco, deve avere una dimensione abbastanza piccola (ad esempio, un quadrato di lato 4 m).

Le maggiori problematiche relative all'implementazione del gioco riguardano alcuni limiti degli strumenti a disposizione, in particolare del sistema di visione e del controllo del movimento. Per quanto riguarda la visione, il funzionamento degli algoritmi è fortemente inficiato dalla qualità dell'immagine proveniente dalla telecamera. Le immagini, a causa della forte compressione JPEG cui sono sottoposte, hanno visibili artefatti che influenzano negativamente il riconoscimento dei blob colorati. Purtroppo, essendo la compressione effettuata a bordo del robot, non è possibile ridurne il livello. Altri problemi sono legati alla forte dipendenza alle condizioni di luminosità dell'algoritmo di visione utilizzato, che rende necessario effettuare più volte il training del classificatore, a seconda delle condizioni di luce.

Per quanto riguarda il controllo del movimento, l'imprecisione dei comandi ricevuti dal robot e la strategia di controllo attuata (ad anello aperto) rendono impossibile controllarne l'effetto sull'attuale hardware. Questo provoca dipendenza della risposta del robot ai comandi da molteplici fattori, come la carica della batteria e la superficie su cui si muove. I frequenti ritardi nella trasmissione dei comandi, e il fatto che questi vengano eseguiti in coda, rendono difficile far agire il robot in maniera istantanea. A causa della mancanza di opportuni sensori, risulta impossibile creare una mappa dell'ambiente, o più semplicemente mantenere informazioni riguardo le posizioni di alcuni elementi rispetto al robot: questo non permette di attuare strategie complesse, che tengano conto della posizione degli oggetti già visti, e di pianificare in maniera adeguata l'aggiramento degli ostacoli.

I maggiori successi nell'implementazione del robot provengono dallo sfruttamento della logica fuzzy e da MrBrian, che ben si adattano a situazioni poco definite, come lo sono i dati che vengono estratti dai sensori del robot. Tramite regole fuzzy, si è riusciti a implementare un comportamento che risponde abbastanza bene agli stimoli dell'ambiente, rendendo il robot in grado di evi-

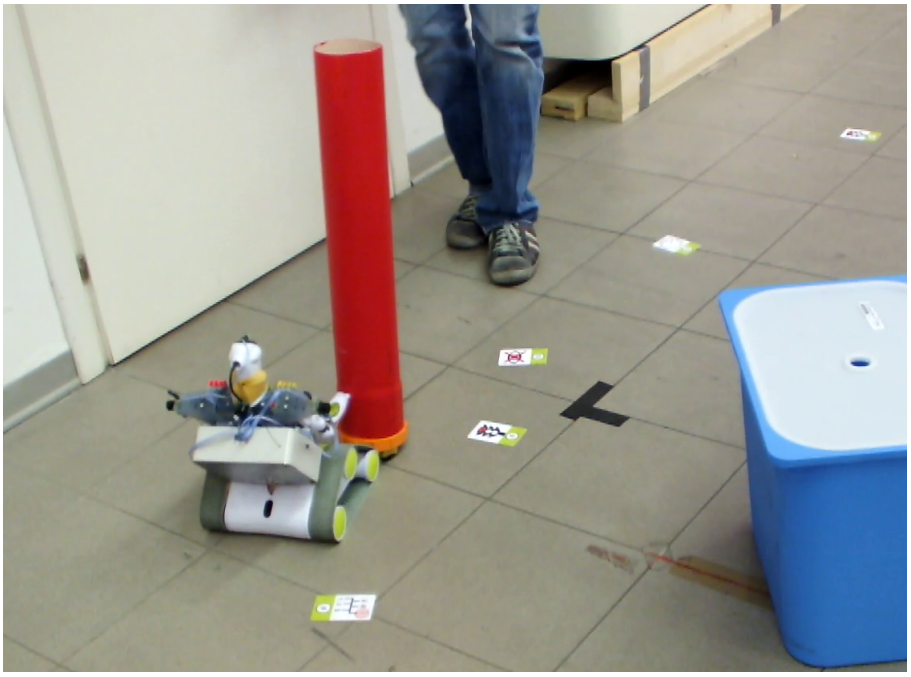

Figura 6.1: *Spykee va all'attacco della torre*

tare ostacoli in maniera abbastanza soddisfacente e di puntare gli obiettivi seguendo delle traiettorie abbastanza pulite e razionali, nonostante il controllo impreciso dei motori.

Altri ottimi risultati sono arrivati grazie al middleware ROS, che ha consentito un forte disaccoppiamento delle parti del sistema, favorendo la riusabilità dei componenti. È facile quindi re-implementare parte del codice per portare RoboTower su altri robot, e il debug viene facilitato dai tool grafici e a riga di comando forniti.

Ringraziamenti

Ringraziamo Davide Perego per averci fornito le immagini, sia per gran parte delle carte, sia per il logo dell'interfaccia grafica del gioco RoboTower.

Un grosso ringraziamento va inoltre a tutto lo staff dell'AIRlab per il loro grandissimo supporto durante tutto lo svolgimento del progetto, in particolare al prof. Bonarini, a Davide Rizzi e a Martino Migliavacca: il loro aiuto ci è stato fondamentale!

A. Installazione del software

Il software realizzato è stato sviluppato per un sistema Linux. Le istruzioni seguenti si riferiscono principalmente alla distribuzione Ubuntu (in particolare, sono state testate con la versione 12.04) in quanto è l'unica supportata da ROS. Il software è stato testato anche con Debian: l'unica differenza risiede nell'installazione di ROS, che va fatta manualmente.

A.1 Installazione del software lato PC

Installazione di ROS Le istruzioni di installazione di ROS sono presenti nel sito ufficiale, `www.ros.org`, e variano a seconda della distribuzione. In particolare, per Ubuntu, è necessario aggiungere i repository di ROS alle sorgenti software di sistema, dopodiché è possibile installare ROS direttamente dal package manager di Ubuntu.

Nota: I comandi per installare ROS su Ubuntu, riferiti all'ultima versione di ROS (`fuerte`) e del sistema operativo (`precise`) disponibile al momento della scrittura della documentazione, sono:

```
$ sudo sh -c `
    echo "deb http://packages.ros.org/ros/ubuntu precise main"
    > /etc/apt/sources.list.d/ros-latest.list `
$ wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
$ sudo apt-get update
$ sudo apt-get install ros-fuerte-desktop
```

Una volta installati i pacchetti, è necessario fare in modo che le variabili di ambiente di ROS vengano impostate all'avvio:

```
$ echo "source /opt/ros/fuerte/setup.bash" >> ~/.bashrc
$ . ~/.bashrc
```

Da ultimo, è necessario installare alcuni componenti aggiuntivi di ROS che verranno utilizzati in seguito:

```
$ sudo easy_install -U rosinstall vcstools rosdep
```

Installazione delle dipendenze Per poter compilare correttamente il software realizzato, è necessario che siano presenti sul sistema, oltre a ROS, i seguenti pacchetti installabili dai repository della distribuzione Linux in uso:

- `flex` e `bison`, utilizzate per generare i parser di Mr. Brian
- le librerie OpenCV (`libopencv-dev`), utilizzate per elaborare le immagini in Vision e in LittleEndian

- le librerie Qt4[1] (`libqt4-dev`), utilizzate per l'interfaccia grafica
- le librerie Qt Mobility (`qtmobility-dev`), che contengono QtMultimediaKit, utilizzato per la riproduzione dei suoni

È inoltre necessario installare il pacchetto di ROS `ann`[2], contenuto nello stack `ias_perception`. Attualmente (marzo 2013), questo stack non è più disponibili sul sito di ROS (sono stati deprecati). Ann è parte del package ias-perception, che si può scaricare dal repository git seguente: `http://code.in.tum.de/git/ias-perception.git`, accessibile anche dall'interfaccia web che si trova al link `http://code.in.tum.de/indefero/index.php//p/ias-perception/source/tree/master/`. Una volta scaricato è sufficiente compilarlo con il comando `make` sia dalla cartella principale `ias-perception` che dalla sottocartella `ann`, e successivamente copiarlo in una cartella presente nel PATH degli stack di ROS (ad esempio `/opt/ros/fuerte/stacks`).

Inoltre, per il corretto funzionamento di alcuni tool di debug, è necessario avere installato l'interprete Python (versione 2), e le librerie PyQt (bindings delle Qt per Python).

Compilazione del software Innanzitutto, scaricare i sorgenti da git:

```
$ git clone git://github.com/pogliamarci/robotower.git
```

oppure dalla pagina del progetto sull'airwiki in una cartella qualunque. Una volta scaricati i file, è necessario aggiungere la cartella del progetto alla variabile d'ambiente `ROS_PACKAGE_PATH`, inserendo in fondo al file `~/.bashrc` la riga

```
export ROS_PACKAGE_PATH=path:$ROS_PACKAGE_PATH
```

dove `path` va sostituito con il percorso completo della cartella che contiene i sorgenti del progetto. Dopo aver aggiornato il file `~/.bashrc`, eseguire

```
$ source ~/.bashrc
```

A questo punto, dovrebbe essere possibile compilare i sorgenti eseguendo semplicemente

```
$ make
```

dalla cartella principale del progetto.

Nota: In alternativa, è possibile utilizzare il comando `rosmake <nome_package>` per ognuno dei package realizzati. Quest'ultimo comando dovrebbe tentare di compilare e/o risolvere automaticamente le dipendenze, ma in generale risulta più lento e fornisce meno indicazioni sullo standard output rispetto all'utilizzo di `make`.

[1] Probabilmente le librerie Qt sono già installate da ROS come dipendenza
[2] `http://ros.org/wiki/ann`

A.2 Avvio del gioco

Una volta collegato al computer lo XBee e acceso il robot, collegarsi alla rete wireless creata da Spykee[3]. A questo punto, per avviare ROS e tutti i nodi che controllano il robot, è sufficiente eseguire il comando

```
$ roslaunch spykee.launch
```

dalla cartella del progetto.

Nota: Se il nodo Echoes non riesce ad accedere alla porta seriale, è possibile che i permessi per l'utente corrente non consentano di aprire il dispositivo: per risolvere questo problema è sufficiente aggiungere l'utente corrente al gruppo `dialout` oppure impostare i permessi corretti tramite regole di `udev`.

A.3 Firmware del microcontrollore

Il firmware è stato realizzato per la scheda (Evaluation Board) STM32F4 Discovery della ST. Nelle istruzioni che seguono, verrà utilizzato il programmatore integrato sulla scheda (ST-Link v2).

Prerequisiti

1. Installare una toolchain GNU per ARM. Per la realizzazione del firmware, è stata utilizzata CodeSourceryLite[4], che fornisce una versione precompilata della toolchain GNU (gcc, gdb, ...). È sufficiente scaricare l'archivio, decomprimerlo in una cartella, e aggiungere la sottocartella `bin` dell'archivio nel `PATH` dell'utente. Una volta che la toolchain è installata, il compilatore è raggiungibile dal comando

   ```
   $ arm-none-eabi-gcc
   ```

   Se si usa una toolchain diversa, potrebbe essere necessario cambiare i nomi degli eseguibili nel `Makefile`.

   Nota: Attualmente CodeSourceryLite viene rilasciato compilato per processori x86 a 32bit. Nel caso il sistema operativo in uso sia a 64 bit, è necessario installare le librerie `ia32-libs`.

2. Installare OpenOCD, scaricabile da `http://openocd.sourceforge.net`. Potrebbe essere necessario installare alcune dipendenze (autotools, automake, libusb ed altre), elencate comunque nella documentazione del programma.

[3] il SSID è SPYKEE seguito da un codice identificativo del robot. Non è necessario inserire manualmente i parametri della connessione (indirizzo IP, ...), in quanto il robot funziona da server DHCP

[4] scaricabile da `http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/editions/lite-edition/`

Nota: Al momento della scrittura del presente manuale (luglio 2012), il programmatore ST-Link v2 è ufficialmente supportato solo sulla piattaforma Windows. Tuttavia, è presente un supporto sperimentale in OpenOCD: per poterlo utilizzare, è necessario scaricare e compilare la release di sviluppo dal repository git[5]. Una volta scaricati i sorgenti, è sufficiente compilarli tramite i seguenti comandi (dalla cartella in cui si è clonato il repository):

```
$ ./configure --enable-maintainer-mode --enable-stlink
$ make
# make install
```

L'ultimo comando serve per installare OpenOCD nelle cartelle di sistema e configurare alcune regole di udev: dev'essere quindi eseguito con i privilegi di root.

3. Scaricare i sorgenti di ChibiOS da `http://chibios.org`. Una volta scaricato e decompresso l'archivio, aprire il `Makefile` presente nella cartella `FirmwareSpykee` e modificare la riga

   ```
   CHIBIOS = /opt/ChibiOS/ChibiOS_2.4.2
   ```

   inserendo il percorso corretto. Il firmware è stato testato con la versione 2.4.2 di ChibiOS.

Compilazione ed installazione

Compilazione Posizionarsi nella cartella `FirmwareSpykee` ed eseguire

```
$ make
```

Se la compilazione ha successo, viene creato nella sottocartella `build` il file `ch.elf`.

Installazione Per caricare il firmware sulla scheda, è necessario prima di tutto spegnere l'alimentazione[6] (tramite l'apposito interruttore) e collegare la scheda al computer tramite USB. Si deve quindi avviare OpenOCD con il comando

```
$ openocd -f board/stm32f4discovery.cfg
```

Quindi, mantenendo aperto OpenOCD, collegare il debugger gdb:

```
$ arm-none-eabi-gdb ch.elf
(gdb) target extended-remote localhost:3333
```

[5] `http://sourceforge.net/scm/?type=git&group_id=274635`

[6] Attenzione! Collegare l'alimentazione della scheda quando è collegata al computer tramite USB potrebbe causare seri danni alla stessa e renderla inutilizzabile

dove `ch.elf` è il percorso del firmware compilato che si vuole caricare sulla scheda. Una volta che `gdb` è connesso ad OpenOCD, la seguente sequenza di comandi cancella il contenuto della memoria FLASH, carica il nuovo firmware, e quindi lo avvia:

```
(gdb) monitor reset halt
(gdb) monitor flash probe 0
(gdb) monitor stm32f2x mass_erase 0
(gdb) load
(gdb) monitor reset halt
(gdb) continue
```

Nota: A volte la scheda non viene riconosciuta da OpenOCD. Questo problema a volte viene risolto tenendo premuto il pulsante RESET sulla scheda mentre si avvia openOCD. Se anche così non funziona, è necessario cancellare il contenuto della memoria flash con l'utility ST Visual Programmer (disponibile solo per Windows).

A.4 Collegamenti hardware

Attualmente, i pin presenti sulla Discovery Board montata sul robot sono assegnati in questo modo:

- Collegamento con i sonar (timer): `PA8` (nord), `PB4` (sud), `PA0` (ovest), `PC6` (est)
- Seriale per collegamento con lo XBee, bitrate 115200 bps: `PA2` (TX, va collegato al terminale `RX` dello XBee) e `PA3` (RX, va collegato al terminale `TX` dello XBee)
- Ricevitore dei segnali inviati
  - Dalle fabbriche: `PD0`, `PD1` e `PD2`
  - Dalla torre: `PD3`
- Led presenti sul robot:
  - Led rossi: `PE7` (0), `PE8` (1), `PE9` (2), `PE10` (3)
  - Led gialli: `PE11` (0), `PE12` (1), `PE13` (3), `PE14` (4)
  - Led verde: `PE15`
  - Led a infrarossi (sulle spalle del robot): `PD11`
- Lettore RFID (collegamento seriale a 9600 bps): `PB11` (RX)

Nota: Il file `board.h` permette di configurare le funzioni assegnate ai vari pin (input, output, alternate mode), secondo le informazioni presenti sul datasheet. Per modificare l'assegnamento dei pin, potrebb eessere necessario modificare - oltre a `board.h` - il sorgente dei moduli del firmware che ne fanno uso.

Alimentazione Tutti i dispositivi (discovery board, XBee, lettore RFID, sonar, ricevitore Atmel, driver per i led a infrarossi) vanno alimentati a +5V DC tramite il regolatore di tensione collegato alla batteria principale di Spykee. La scheda contenente le resistenze necessarie al funzionamento dei led necessita soltanto del collegamento a massa.

Configurazione Zigbee I dispositivi Xbee vengono utilizzati come un collegamento seriale punto-punto tra l'unità di elaborazione e il robot, pertanto per il debugging di problemi di configurazione è sufficiente utilizzare un terminale seriale, come `screen` o `minicom` su Linux. Per configurare i dispositivi Xbee è necessario utilizzare il programma X-CTU scaricabile dal sito Digi. Una volta aperta la scheda di configurazione, è sufficiente impostare su entrambi i dispositivi il bitrate di 115200 bps, e impostare come Destination Address (High e Low) il Serial Number dell'altro Xbee.

A.5 Comandi per il firmware del robot

Il firmware installato sulla discovery board presente nel robot accetta alcuni comandi che vengono inviati dal nodo `Echoes` attraverso il collegamento seriale, e permettono di controllare i led:

- `reset`: spegne tutti i led
- `led`: comanda i led colorati. Il comando è seguito da `R`, `Y` oppure `G` a seconda che si vogliano comandare rispettivamente i led rossi, i led gialli oppure il led verde che c'è sulla testa di Spykee. Il secondo argomento del comando può essere `B` per attivare la modalità lampeggiante del gruppo di led selezionato, oppure una stringa di cifre binarie che indicano se ognuno dei led del gruppo deve essere spento (`0`) oppure acceso (`1`). Ad esempio, il comando

  ```
  led R 1101
  ```

  accende il primo, il secondo e l'ultimo dei led rossi e spegne il terzo, mentre il comando

  ```
  led G B
  ```

  fa lampeggiare il led verde sulla testa del robot.
- `infrared on` accende i led a infrarossi, mentre `infrared off` li spegne.

I messaggi inviati dal robot al computer attraverso il collegamento seriale sono composti da una singola linea (sono terminati da `CR LF`) e contengono all'inizio la tipologia tra parentesi quadre. Le tipologie di messaggio che sono state definite, insieme con un esempio del loro formato, sono:

- `[SONAR] N:1443,S:161,W:3444,E:3458` contiene per i quattro sonar installati (nord, sud, ovest, est) la distanza rilevata in millimetri
- `[RFID] <id>` contiene l'ID del tag RFID rilevato. Contiene anche il checksum, calcolato come da datasheet del lettore ID-12 (la correttezza viene calcolata da Echoes e non a bordo del firmware).
- `[TOWER] destroyed <N>` indica che una torre o fabbrica è stata distrutta. `<N>` è compreso tra 1 e 4 e identifica la torre o la fabbrica che è stata distrutta.

Bibliografia

[1] ROS Wiki, www.ros.org

[2] Meccano, Spykee: the Spy Robot, http://www.spykeeworld.com

[3] The ChibiOS/RT Project, http://chibios.org

[4] Qt - Cross-platform application and UI framework, http://qt.nokia.com

[5] Progetto RoboWii, http://airlab.elet.polimi.it/index.php/ROBOWII

[6] MRT: Modular Robotics Toolkit, http://airlab.elet.polimi.it/index.php/MRT

[7] A. Bonarini, M. Matteucci, M. Restelli, *MRT: Robotics Off-the-Shelf with the Modular Robotic Toolkit*, in D. Brugali (ed.), *Software Engineering for Experimental Robotics*, STAR 30, Berlin, Springer-Verlag, 2007, 345-364

[8] C. Mandelli, D. Zamponi, *RoboWII 2.1: Implementazione di un gioco robotico*, http://airlab.elet.polimi.it/images/1/1f/RoboWii_2.1-ZamponiMandelli.pdf

[9] A. Bonarini, M. Matteucci, M. Restelli, *Concepts and fuzzy models for behavior-based robotics*, International Journal of Approximate Reasoning **41** (2006), 110-127

[10] Discovery kit for STM32 F4 series, http://www.st.com/internet/evalboard/product/252419.jsp