SAPIENZA
UNIVERSITÀ DI ROMA

Sviluppo di un linguaggio visuale in grado di adattarsi autonomamente

Facoltà di Scienze Matematiche Fisiche e Naturali
Corso di laurea in Tecnologie Informatiche

Candidato
Jakub Czerwonko
n° matricola 1049198

Responsabile
Andrea Sterbini

Corresponsabile
Stefano Nolfi

A/A 2011/2012

Indice

1. Introduzione

Il lavoro di tesi ha riguardato lo sviluppo di un linguaggio di programmazione visuale per robot e della relativa interfaccia grafica. Tale linguaggio di programmazione è stato testato per implementare una serie di controllori di complessità crescente e per comparare i risultati ottenuti attraverso la programmazione esplicita e l'adattamento, realizzato attraverso un metodo evolutivo.

L'utente tramite delle "espressioni visuali" è in grado di programmare il comportamento dei robot. L' interfaccia grafica prende in input i valori dei sensori generati dal simulatore Evorobot* e calcola il valore di output in base "all'espressione visuale" realizzata dall'utente.

Tali espressioni visuali realizzate tramite "boxes and arrows" ooppure altrimenti note come "box"(triangoli, rettangoli) sono interpratate alla pari di funzioni connesse tra loro da "arrows" frecce.

La strada percorsa per arrivare allo sviluppo dell'interfaccia grafica è stata lunga e articolata. Prima di iniziare la scrittura del codice è stato, infatti, necessario prendere confidenza con il softwaere evorobot* e analizzare le soluzioni disponibili per procedere all'implementazione dell'applicazione richiesta.

Per la trasformazione di evorobot*, da addestratore di reti neurali in addestratore di espressioni visuali è stato neccessario eliminare totalmente il sistema di gestione della rete neurale.

L’attività di tirocinio è stata svolta per l'istituto di Scienze e Tecnologie della Cognizione del CNR. Il progetto è stato svolto utilizzando esclusivamente strumenti gratuiti e open source.
Il linguaggio di programmazione utilizzato per la gestione della rete è il C, mentre, l'interfaccia grafica è stata sviluppata con le librerie Qt.

2. La robotica

La parola robotica[1] proviene dal ceco robota, dove ha il significato di "lavoro pesante" o "lavoro forzato". Questo termine è stato introdotto dallo scrittore ceco Karel Čapek, nel 1920[2].

Karel Čapek definiva il robot un uomo artificiale organico privo di sentimenti votato esclusivamente al lavoro:

> *« Quale operaio è migliore dal punto di vista pratico? È quello che costa meno. Quello che ha meno bisogni. Il giovane Rossum inventò l'operaio con il minor numero di bisogni. Dovette semplificarlo. Eliminò tutto quello che non serviva direttamente al lavoro. Insomma, eliminò l'uomo e fabbricò il Robot. » .*

L'inizio di studi conformi all'obiettivo di sviluppare macchine, con caratteristiche analoghe agli organismi biologici, coincide con la nascita della *cibernetica*. Scienza affermatasi alla fine della prima metà del 900, nata dall'interscambio di teorie e idee elaborate nell'ambito della teoria del calcolo, della teoria dell'informazione e della biologia.

La robotica, in virtù della sua interdisciplinarietà, fluisce in differenti aree di studio: *domotica, robotica industriale, biorobotica, arte robotica, robotica militare, robotica evoluzionistica, microrobotica*.

L'impiego della robotica non è solo prettamente scientifico, ma educativo e ludico. Il primo ambiente di robotica educativa che ha conosciuto una larga diffusione è LEGO Mindstorm.

---

[1] Karel Čapek (1890-1938) il quale usò per la prima volta il termine nel 1920 nel suo dramma teatrale I robot universali di Rossum. Anche se esiste in realtà chi sostiene che l'inventore della parola robot, sia stato il fratello di l Capek, **Josef**, (1887- 1945) anche lui scrittore e pittore cubista, il quale utilizzò la parola "*automat*", (automa), in un suo racconto del **1917**, ***Opilec*** ("L'ubriacone").

[2] *Rossum's Universal Robots* tr. It. *I robot universali di Rossum*. Dramma fantascientifico articolato in tre atti del ceco Karel Čapek (1890-1938).

1 Robot realizzato con Lego Mindstorm

Tale ambiente è il risultato dell'attività congiunta del laboratorio di ricerca "Epistemology and Learning Group" dell'MIT e l'industria di giocattoli danese LEGO (Mindell et al., 2000). L'obiettivo che si proposero inizialmente fu quello di sviluppare un nuovo tipo di giocattolo. L'idea di partenza era di sviluppare, un set di costruzioni che potesse consentire ai bambini di costruire oggetti e macchine analoghe a quelle che potevano essere sviluppate con i kit LEGO dell'epoca, ma al tempo stesso in grado di fornire ai bambini la possibilità di animare le proprie costruzioni. Per tale ragione decisero di estendere i kit LEGO. Così cominciarono a sviluppare in collaborazione con la LEGO un mattoncino programmabile con il linguaggio LOGO.

Riguardo l'ambiente di programmazione, furono sperimentate diverse scelte, inclusi ambienti di programmazione testuali tradizionali che tuttavia, pur essendo molto potenti, richiedevano ai ragazzi uno sforzo

iniziale relativamente alto. Per questa ragione, Resnick e collaboratori implementarono LogoBlocks, una versione grafica del linguaggio in cui le istruzioni sono costituite da icone/comandi che possono essere posizionati sullo schermo.

2.1 I robot

I robot sono sistemi artificiali (dotati di un corpo, di attuatori, sensori, e di un sistema di controllo), situati in un ambiente fisico ed eventualmente sociale con il quale interagiscono.
L'azione è realizzata dagli attuatori, i quali azionano delle ruote, muovono un arto, aprono e chiudono una pinza. La percezione avviene per mezzo di sensori, alterati dall'ambiente esterno e capaci di registrare tale alterazione.
Il sistema di controllo, imposta lo stato degli attuatori in base allo stato dei sensori. Nei sistemi reattivi lo stato dei motori è determinato solamente dallo stato corrente dei sensori. In sistemi più complessi lo stato dei motori può essere determinato non solo dai sensori, ma da uno stato interno nel quale è memorizzato lo stato precedente dei sensori oppure dello stato interno stesso.
Il controllo passo-passo e la retroazione sono forniti da un programma eseguito da un computer esterno o interno al robot, o da un microcontroller .

2.2 Il sistema di controllo

Esistono diversi metodi per sviluppare robot autonomi: quelli basati sulla progettazione in cui il ricercatore gestisce l'organizzazione generale del robot e i dettagli di ciascuna componente; metodi *bioispirati* grazie ai quali il ricercatore individuando un organismo, realizza un robot con caratteristiche simili; metodi adattivi dove l'organizzazione generale del robot non è definita dall'utente, ma è gestita da un sistema di adattamento automatico.

Le tre metodologie (prima indicate), possono essere utilizzate in modo combinato. Infatti, alcune parti di robot sono istruite in modo automatico e altre, invece, possono esser progettate dal ricercatore.

Le metodologie utilizzate si differenziano rispetto agli obiettivi che si prefiggono: l'obiettivo può essere scientifico-tecnologico ossia si vorrebbe sviluppare robot in grado di risolvere complessità irrisolte; modellistico, come per esempio comprendere organismi naturali sviluppando sistemi artificiali con funzionalità analoghe agli organismi viventi; applicativo, in altre parole svolgere una funzione precisa determinata in precedenza.

I metodi basati sulla progettazione sono i più utilizzati per lo sviluppo di robot. Il punto di partenza per utilizzare questi metodi è:

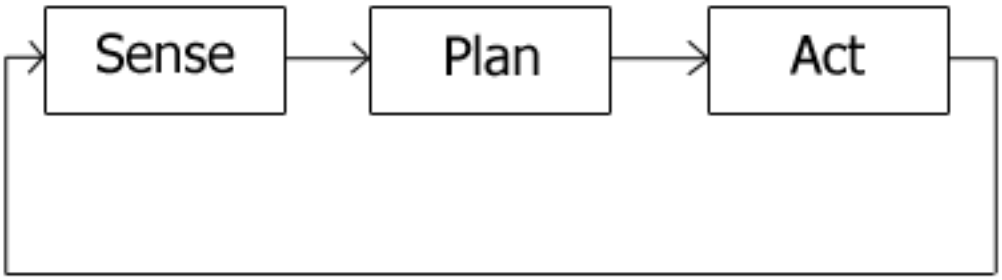
1) la scomposizione del problema in vari sotto problemi;

2) progettare moduli capaci di risolvere il sotto problema precedentemente individuato.

Due sono gli approcci che si pongono la problematica di come disegnare il sistema di controllo di un robot: ***l'approccio deliberativo*** e ***l'approccio behavior-based***.

2.2.1 Sistema deliberativo

Sviluppato nell'ambito dell'intelligenza artificiale, a partire dagli anni sessanta inizia a essere applicato alla robotica[3].
Nell'approccio deliberativo il sistema nervoso è composto di tre parti: *un sistema percettivo* capace di percepire l'ambiente esterno al fine di rappresentarlo; *un sistema di pianificazione* o *di ragionamento* in grado di pianificare un insieme di azioni da attuare in vista dell'obiettivo e della rappresentazione dell'ambiente infine, un *sistema motorio* capace di eseguire le azioni programmate in precedenza.



L'architettura del sistema nervoso, seguendo l'approccio deliberativo, si articola a più livelli:

- Il livello più alto o superiore, identifica i diversi sotto-obiettivi al fine di raggiungere l'obiettivo generale, del contesto in esame.
- I livelli sottostanti scompongono ciascun sotto-obiettivo in più obiettivi, fino ad arrivare a definire le singole azioni atomiche, a

[3]N. J. Nilsson, (1969), *A mobile automation. An application of artificial intelligence Techniques,* in *Proocedings of 1st International Joint Conference on artificial intelligence (IJCA) ,Washington, pp509-20*

livello degli attuatori.[4]

Questa tipologia di architettura non prevede l'influenza diretta del sensore sull'attuatore. Si scelgono tutte quelle azioni capaci di massimizzare la possibilità di raggiungere l'obiettivo prefissato. Questo richiede la capacità di individuare una rappresentazione dell'ambiente completa, aggiornata e di generare in tempo reale un piano di azioni efficace e dettagliato. Tutti aspetti che riscontrano, comunque problematiche: basta pensare ad esempio alla capacità di fornire una rappresentazione della realtà, in costante aggiornamento, impedisce ai robot dotati di questo tipo di architettura, di operare efficacemente in ambienti dinamici.

2.2.2 Sistemi reattivi

Dal finire degli anni 80 del *'900*, per la progettazione di robot, la robotica inizia ad avvalersi del *paradigma*[5] *reattivo:* il quale permette al controllore del robot di operare molto velocemente.

Alla base dei software reattivi c'è la concezione del computer come ambiente di apprendimento[6]. Un sistema è detto reattivo quando il suo comportamento è influenzato da eventi che hanno luogo nel mondo reale, al di fuori dei computer che governano il sistema stesso.

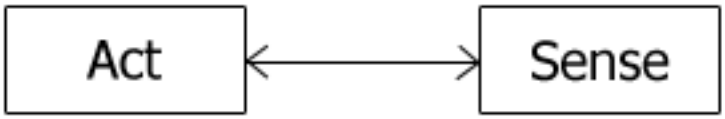
Può esser definito "*event-driven*", ovvero un sistema guidato dall'evento, capace di interagire continuamente con l'ambiente esterno, di

[4] Cfr. S. Nolfi, *Che cos'è la Robotica Autonoma,* Carrocci, 2009, pag. 96.

[5] Definiamo un paradigma come un insieme di assunzioni teoriche e di tecniche risolutive che caratterizza un approccio ad una classe di problemi.

[6] Per approfondimenti sui sistemi reattivi vedi: http:// disi.unige.it/

conseguenza, reagire agli stimoli che da esso pervengono.
*Nei **sistemi reattivi*** il robot interagisce con il mondo esterno tramite sensori ed attuatori.



La conoscenza che ne deriva non è modellata, non è memorizzata, ma è estratta 'on-line' dal mondo stesso tramite i sensori.

I comportamenti del robot sono definiti come reazione alle informazioni percepite dall'ambiente.
Nel caso di sistemi reattivi si considera che:

-l'ambiente può perdere consistenza temporale e stabilità

-il sistema sensoriale robotico è adeguato solo al compito specifico

-può essere difficile localizzare un robot rispetto ad un modello del mondo

-la conoscenza rappresentativa del mondo non è presa in considerazione.
Come ricorda Brooks[7] "*Non esiste una rappresentazione del mondo (...)* la conoscenza che abbiamo del mondo non è modellizzata tanomeno memorizzata nel robot, "*ma è estratta in tempo reale dal mondo stesso attraverso i sensori*[8]".
Poiché non esiste un modello del mondo, di conseguenza non esiste pianificazione a priori delle azioni del robot.

[7] R.A Brooks, "The world is the best model" 1986, cfr. http://rair.cogsci.rpi.edu/pai/restricted/logic/elephants.
[8] R.A.Brooks, *Cambrian Intelligence*, The Mit Press, 2005.

Nelle architetture di tipo reattive, i diversi moduli che compongono l'architettura hanno un compito specifico, ognuno dei moduli ha una specifica competenza e attua un determinato comportamento del robot. Il comportamento complessivo del robot è determinato dall'insieme dei comportamenti presenti.

L'organizzazione all'interno di tale architettura risponde a due principi: *principio di indipendenza* e *principio di località*[9].

Secondo il primo, i vari moduli devono essere mutuamente indipendenti tra loro, è chiara quindi, l'impossibilità di avere un modello del mondo completo, condivisibile tra tutti i moduli.

Il Principio di località prevede che ciascun sotto-compito, per completarsi tenga conto di una parte limitata di tutta l'informazione sensoriale disponibile. Il robot risponde solo ad eventi del mondo senza mantenere stati persistenti: la memoria di cui ha bisogno è realizzata leggendo direttamente la situazione ambientale che gli indica il modo operativo corrente.

Alcuni vantaggi:

-Le architetture di controllo reattive considerano che non sia possibile, avere un modello o rappresentazione del mondo.

- C'è alta adattabilità alle modifiche dell'ambiente (risposte in tempo reale)

-Bassa complessità di ogni livello e basso costo computazionale complessivo del sistema

-E' possibile avere parallelismo nel controllo

-L'estensione dei comportamenti è relativamente semplice.

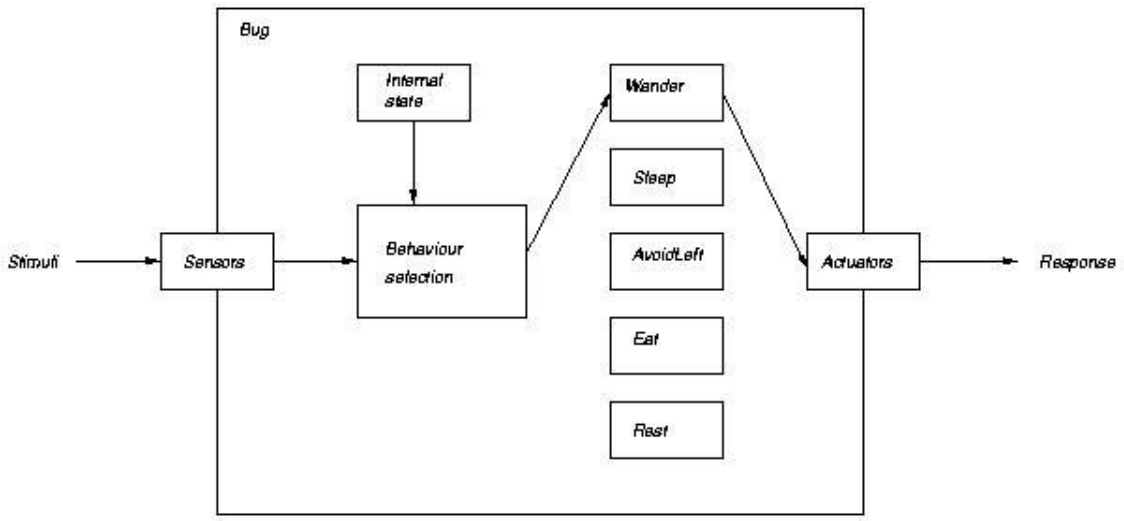
D'altro canto risulta però complicato prevedere a priori il comportamento

[9] Ibidem.

globale del robot e gestire la concorrenza tra moduli; è probabile che incrementando i comportamenti, aumenti la complessità della gestione, complicazioni si possono verificare data l'elevata concorrenza tra moduli, nella risoluzione di conflitti.

2.2.3 Sistemi behavior-based

Sviluppati da Rodney Brooks[10] negli anni 80. Tale approccio concepisce il sistema nervoso del robot, articolato in moduli o livelli comportamentali: che indirizzano il robot nell'adozione di comportamenti congrui al compito da svolgere e che agiscono in maniera asincrona e concorrente. Lo stato dei sensori, all'interno di ciascun modulo è utilizzato per regolare direttamente lo stato dei motori.



Differentemente dall'approccio deliberativo, il ***behavior-based*** non esige

[10] Rodney Allen Brooks, nato il 30 Dicembre 1954, Adelaide , in Australia. Professore di robotica presso il Massachusetts Institute of Technology . Prolifico autore di libri, membro fondatore della Associazione Americana per l'Intelligenza Artificiale (AAAI) e altro.

che il robot fornisca una rappresentazione dell'ambiente esterno, aggiornata e dettagliata, creando così una copia virtuale, anzi al contrario prevede che le informazioni, circa l'ambiente esterno siano estrapolate direttamente dall'ambiente stesso, quando è necessario.

Nell'ambito di architetture ***behavior-based***, sono state proposte dallo stesso Brooks, architetture *subsumption*.

Tali architetture hanno un'organizzazione gerarchica, articolata in: moduli di basso livello (che permettono al robot di eseguire compiti semplici, come evitare un ostacolo) e moduli di alto livello (fanno eseguire al robot compiti più complessi, come pulire un appartamento). Aspetto molto importante di questi ultimi moduli è che possono avvalersi dei moduli di livello basso, ma mai il contrario. Ad esempio un modulo che genera il comportamento di navigazione si avvale di un modulo di basso livello, che consente l'evitare ostacoli.

Il coordinamento dell'attività dei moduli è definito da uno schema esclusivo secondo il quale i moduli di livello superiore possono assumere il controllo e impedire che i segnali prodotti da moduli di livello più basso possano raggiungere l'attuatore.

Il progettista ha la possibilità di sviluppare il sistema nervoso del robot, in maniera incrementale, ovvero implementando i moduli di livello alto, solo dopo aver testato e implementato i moduli di livello elementare.

Altro esempio di architetture ***behavior-based*** sono le *motor schemas*, messe appunto nel 1989, da Ronald Arkin[11].

Nelle *motor schemas*, l'organizzazione dei moduli comportamentali non è gerarchica, anzi cooperano in parallelo e non in modo esclusivo come

[11] Ronald Arkin Craig, nato a New York nel 1949. Esperto di robotica e roboethicist, professore Regents ' nella School of Interactive Computing, College of Computing presso il Georgia Institute of Technology. Conosciuto maggiormente per lo schema motorio, meglio noto come tecnica di navigazione robotica e per il suo libro *Behavior-Based Robotics*.

nelle architetture *subsmption*. L'output finale e complessivo è ottenuto attraverso la somma degli output, prodotti dai vari moduli e moltiplicata per la forza di ciascun modulo.

2.3 Linguaggi di programmazione:

LabView- Urbi - Evorobot*

In questa sezione illustreremo tre diversi tools per lo sviluppo di sistemi di controllo per robot: (i) Labview, un linguaggio di programmazione visuale, (ii) Urbi un linguaggio di programmazione convenzionale di alto livello orientato alla programmazione robotica, (iii) evorobot*, un tool per lo sviluppo di sistemi di controllo per robot basati sulle reti neurali artificiali addestrate attraverso algoritmi evolutivi.

2.3.1 LabVIEW

LabVIEW (Laboratory Virtual Instrumentation Engineering Workbench) di *National Instruments*. Tale linguaggio grafico è anche conosciuto come **Linguaggio G** (Graphic Language).

Sviluppato dalla National Instruments a partire dal 1983 per la necessità di creare un software grafico dove poter testare i dispositivi hardware di tale industria.

LabVIEW realizzato per la Apple Machintosh nel 1986; nel 1992 la *National Instruments* ne realizza una versione multipiattaforma.
Utilizzato per l'aquisizione di dati, controllo di processi, generazione di

rapporti; LabVIEW è un ottimo strumento per l'automazione industriale e può essere anche utilizato per il processamento dei dati video, audio e del segnale.

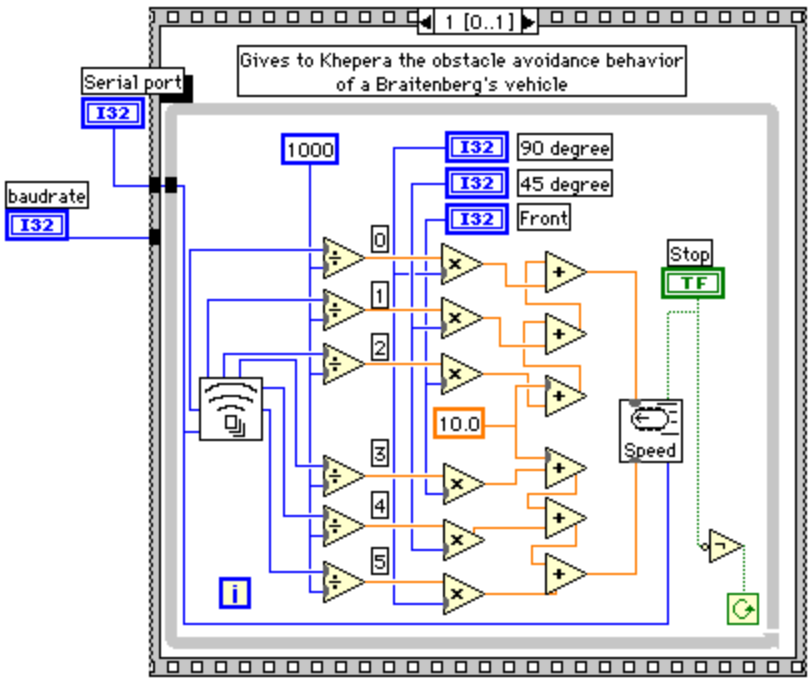
La programmazione nel **Linguaggio G** avvviene con icone e altri oggetti grafici (che rapresentano diverse funzioni), uniti da linee (wire), creano cosi un dataflow. Tali oggetti grafici possono gestire tutti i costrutti standard, come i tipi di dati, cicli, gestione eventi, variabili, ricorsione, programmazione orientata agli oggetti.

Con il **dataflow** l'esecuzione dei programmi è definita dal flusso dei dati atraverso le linee direzionali che collegano gli oggetti grafici (o funzioni).

Il flusso dei dati tra i nodi del programma determina l'ordine di esecuzione, non le linee sequenziali di comandi presenti in un tradizionale linguaggio procedurale.

National Instruments (uno dei partner ufficiali della Lego) sulla base di LabVIEW ha sviluppato un linguaggio di programmazione e una interfaccia di programmazione chiamata NXT-G che serve per programmare i Lego Mindstorm NXT.

NXT-G rispetto a LabVIEW dispone di un'interfaccia semplificata così da permettere ai più giovani di poter programmare con facilità i propri dispositivi robotici.



2 esempio di obstacle avoidance realizzato con LABView per Khepera

2.3.2 Urbi: orchestration language

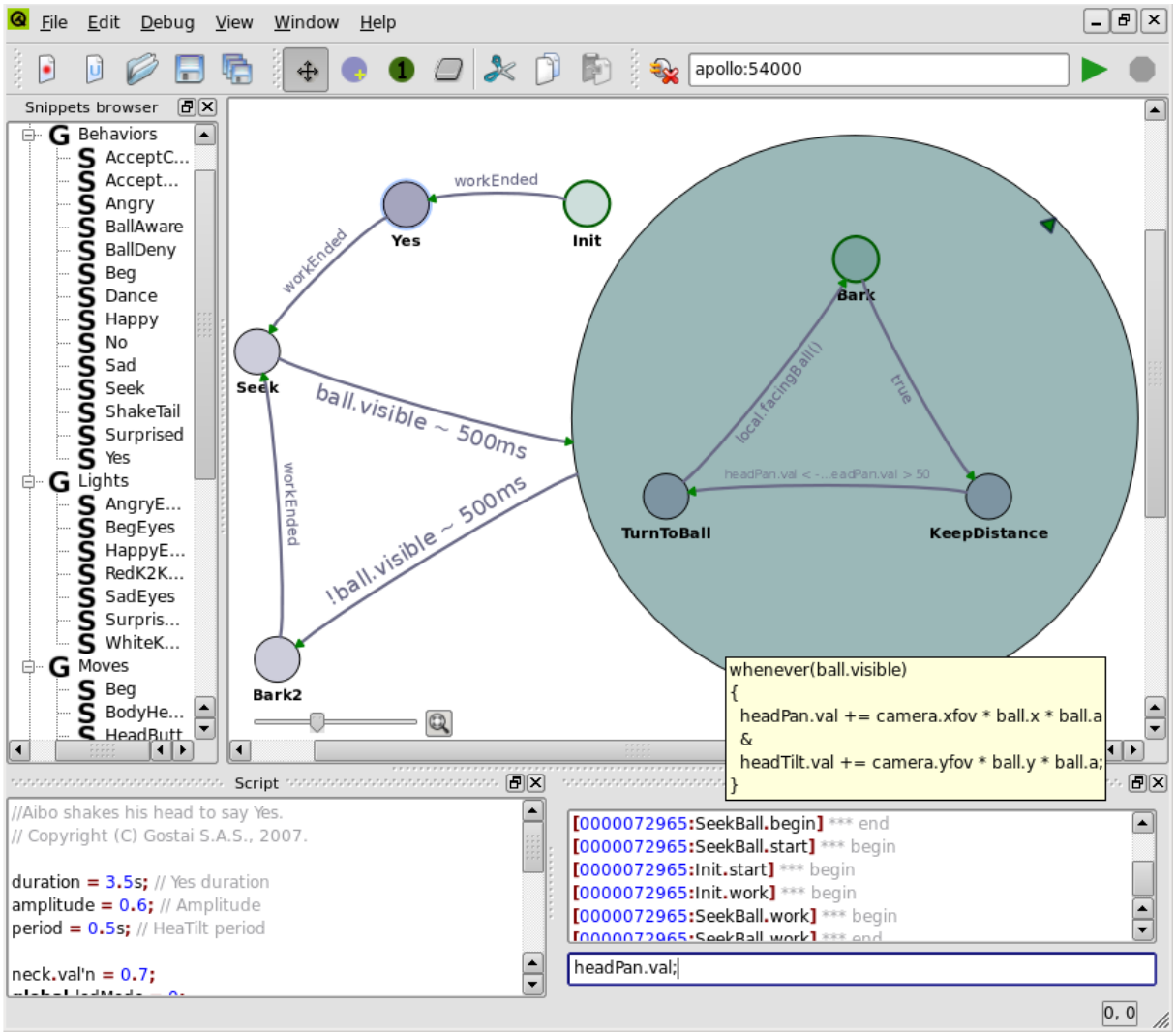
Urbi è una piattaforma software open surce nata per controllare i robot; rappresenta una sorta di sistema operativo per la robotica. Comprende una libreria di C++ chiamata Uobject.

Il linguaggio urbiscript è innovativo perchè integra nuovi metodi per gestire la programmazione parallela e la programmazione a eventi che sono elementi essenziali nella programmazione del robot.

URBI può controllare diversi tipi di robot tra cui: Lego NXT, Bioloid, Spykee, Aibo, Nao e Segway RMP. Urbi supporta diverse piattaforme tra le quali Windows, Linux, MacOSX.

Urbiscript è un "linguaggio di orchestrazione" (orchestration language), ciò significa che, il suo ruolo è coordinare i vari componenti, organizzare

le loro interazioni e i dati che loro si scambiano.
L'orchestrazione si verifica in tempo reale, con un approcio dinamico (tutto si puo riconfigurare a tempo di esecuzione), e con una capacità di controllo parallela e basata sugli eventi.
Questa ultima parte, distingue Urbi dagli altri linguaggi script come Phyton, dove la concorrenza e gli eventi sono gestiti in modo parallelo.
C++ e urbiscript comunicano tra loro tramite la libreria Uobject, la quale è una libreria di C++ capace di collegare in modo trasparente qualsiasi oggetto C++ a urbiscript.
Con Uobject , si può utilizzare qualsiasi classe C++ che apparirà come una classe nativa di urbiscript.
Urbi include in Gostai Studio un'interfaccia di prorammazione grafica che gestisce un'automa a stati finiti (la maggior parte degli altri approci grafici offre diagrammi di flusso).
Una macchina a stati finiti è un modo per rappresentare i comportamenti di un robot, come una serie di stati connessi tra loro da transizioni, le quali sono assocciate a determinate condizioni. Per esempio, un robot può essere in uno stato "cerca la palla", e passare allo stato "segui la palla" attraverso la transizione che ha come condizione "vede la palla".



3 esempio di automa a stati finiti realizato con Urbi

2.3.3 Evorobot*

Evorobot* è un tool che permette di creare dei sistemi di controllo per robot costituiti da reti neurali e di addestrarli attraverso algoritmi evolutivi[12]. Il tool contiene inoltre un simulatore che consente di eseguire esperimenti virtuali basati sui robot e-puck, khepera, e LEGO.
Sviluppato presso il CNR-ISTC da Stefano Nolfi e Onofrio Gigliotta; Il

12 S.Nolfi, M. Mirolli, *Evolution of Communication and Language in Embodied Agents*, Springer, 2010.

software è basato sulla piattaforma robotica e-puck[13](messa a punto dall'Ecole Politechnique Federale de Lausanne).

Evorobot* è scritto in C e C++ , l'interfaccia grafica è sviluppata con la libreria grafica QT; tale programma può essere utilizzato nei sistemi operativi quali, Windows, Linux e Mac.

Evorobot* permette di evolvere robot o un gruppo di robot affinché siano in grado di svolgere determinati compiti in specifici ambienti.

I robot addestrati possono avere differenti tipi di sensori (infrarosso, luce, visione, suono) e diversi tipi di attuatori (motori, luci, emettitore di suoni).

I robot sono determinati da reti neurali, composte da neuroni sensoriali, neuroni interni e neuroni motori. L'ambiente di addestramento prevede aree nelle quali, si possono inserire muri, cilindri di diversi spessori, colori e luci.

Il compito dei robot è specificato nella funzione fitness, la quale a ogni iterazione dell'addestramento, calcola la quantità di lavoro svolto correttamente.

Evorobot* include i seguenti strumenti:

-algoritmi di evoluzione,

-simulatore di reti neurali,

-simulatore di robot, dell'ambiente e le loro interazioni,

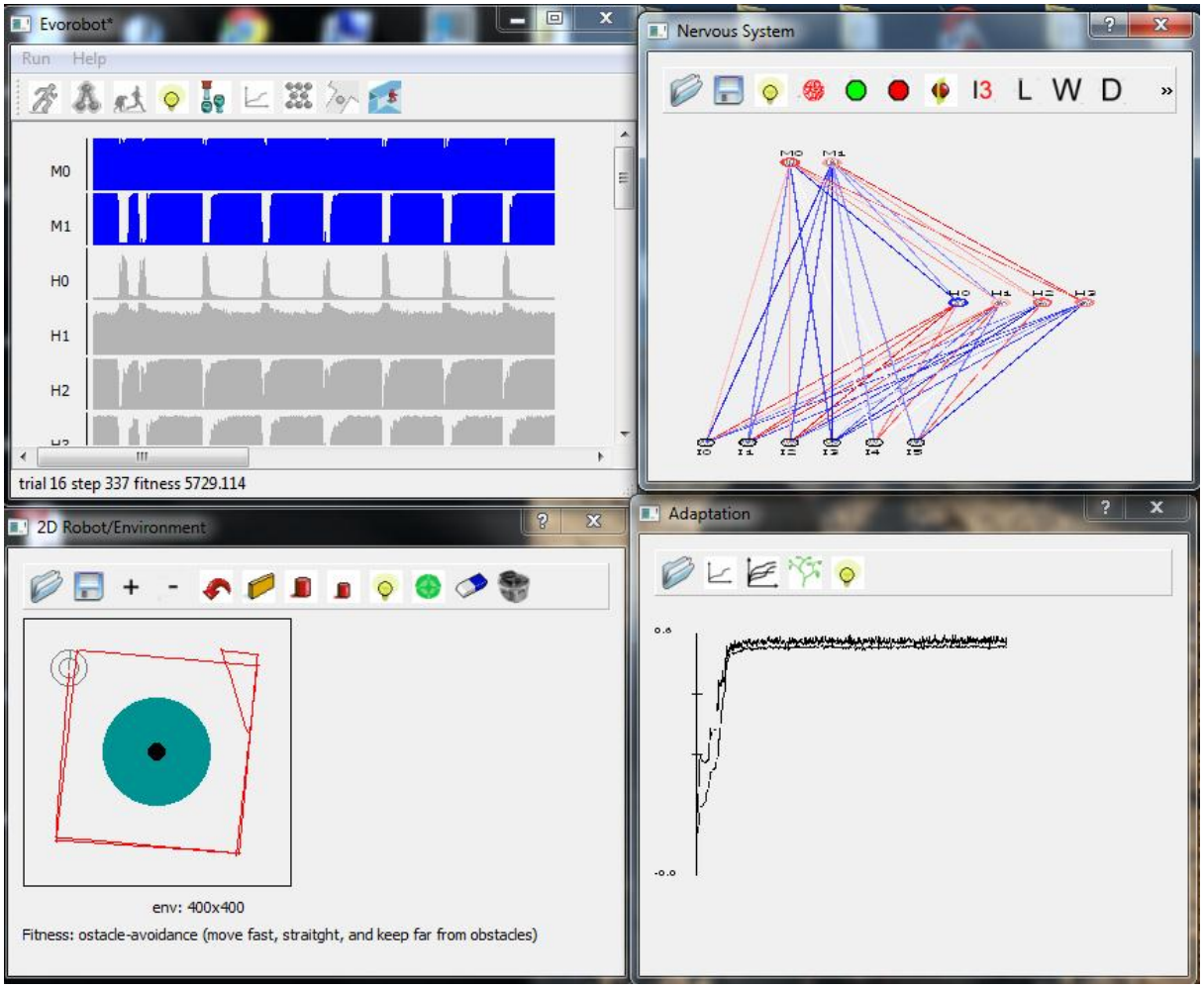
-un'interfaccia grafica con comandi per salvare e analizzare i dati,

-uno strumento che permette all'utente di testare e/o evolvere i robot in simulazione e *in real time* .

-l'Evorobot* firmware può essere installato su ogni robot, i seguenti potranno comportarsi sulla base delle istruzioni ricevute dalla rete neurale del Pc. La comunicazione tra computer e i robot, avviene per mezzo di una connessione Bluetooth.

[13] http://www.e-puck.org/

In evorobot* il processo di adattamento è un processo iterativo che si attua alle reti neurali[14], in modo che l'output prodotto dalle rete stessa, di volta in volta sia migliorato.



4 Interfaccia grafica di Evorobot*

Il processo di Apprendimento si sviluppa tecnicamente grazie ad una famiglia di algoritmi, chiamati algoritmi evolutivi.

Si tratta di un complesso di metodi, ispirati all'evoluzione biologica, che

[14]Sistema computazionale, ispirato al sistema nervoso, formato da neuroni (nodi) uniti fra loro da connessioni. I neuroni sono caratterizzati da uno stato di attivazione continuo e da un output continuo o binario. Le connessioni hanno un valore numerico continuo negativo o positivo detto peso sinaptico, che determina quanto una connessione aumenta o diminuisce l'attivazione del neurone sinaptico. Cfr S.Nolfi, *Che Cos'è la robotica Autonoma*,Carrocci, 2009, pag 40.

generano automaticamente sistemi capaci di eseguire compiti prestabiliti. Gli algoritmi evolutivi si basano su un processo di riproduzione selettiva e variazione, operano su una popolazione di fenotipi e genotipi, che individuano soluzioni alternative ai compiti prestabiliti e evolvono attraverso un processo iterativo di riproduzione e selezione.

Per *fenotipo* si intende un sistema in grado di svolgere la funzione data, può trattarsi di un robot, di un circuito elettrico o altresì di un programma software.

Parlando di *genotipo*, invece, si indica quella parte del sistema (oppure fenotipo), che codifica le caratteristiche del sistema stesso, sottoposte successivamente, al processo di adattamento evolutivo.

Lo sperimentatore per utilizzare questa classe di algoritmi, stabilirà:

-i parametri del fenotipo codificati poi dal genotipo ed evoluti,

-la relazione tra fenotipo e genotipo,

-infine deciderà la *funzione di fitness*.

Quest'ultima è una funzione, che assegna in output un unico valore, capace di definire "quanto" il fenotipo può risolvere il problema dato.

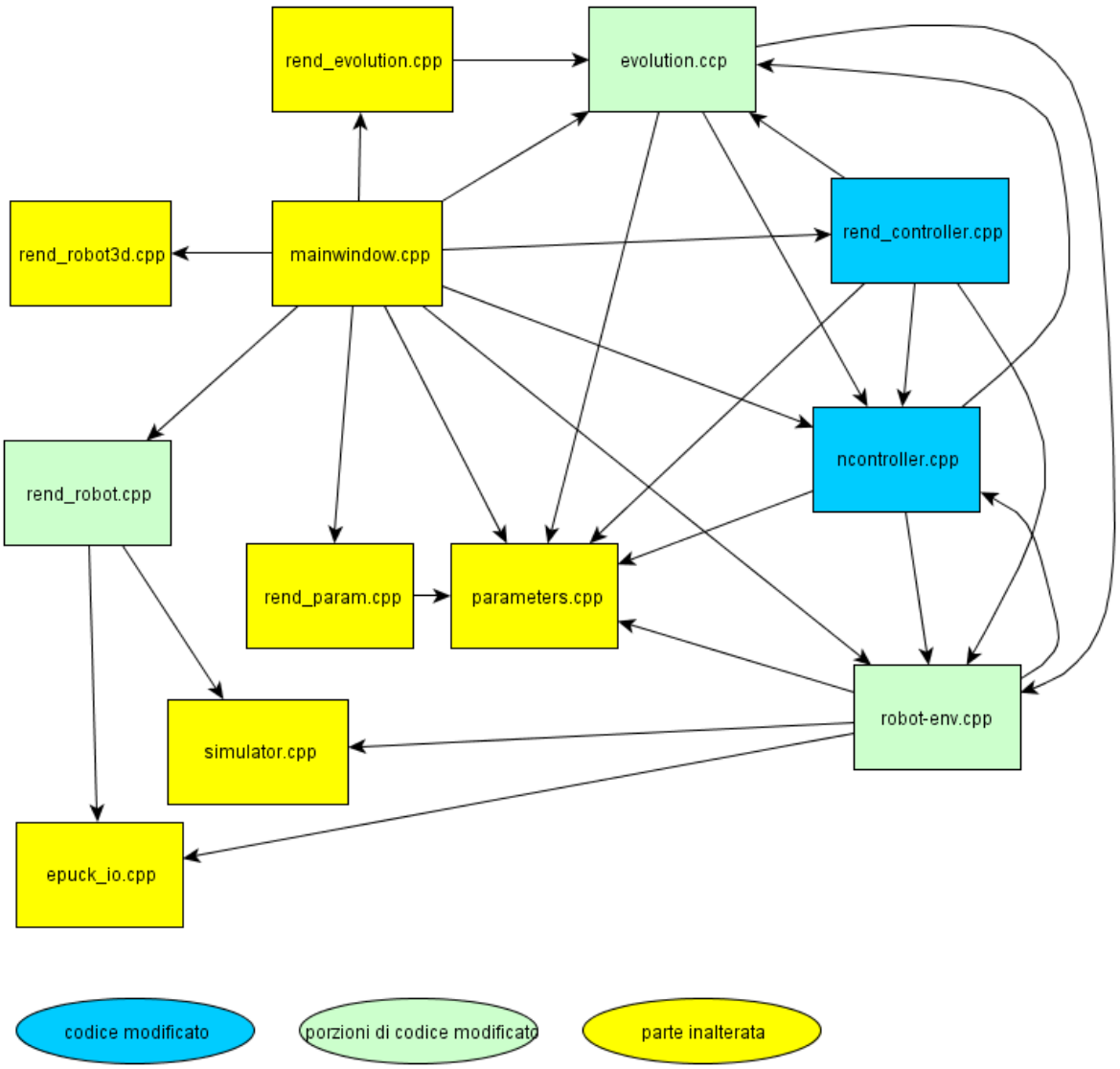
L'efficacia di tali algoritmi, la si deve alla possibiltà di tener conto degli effetti delle iterazioni dal punto di vista della funzione di fitness.

Partendo dall'assunto che gli algoritmi evolutivi si fondano su un processo di riproduzione e variazione, si rintracciano algoritmi che variano rispetto ai seguenti aspetti:

-*Relazione tra genotipo e fenotipo*: in alcuni casi la relazione è diretta, in altri casi il genotipo codifica elementi di base che combinati tra loro e maturati danno luogo al fenotipo.

-*Modalità di variazione e selezione*: un genotipo può ereditare parte del proprio genotipo da due diversi individui. La capacità degli individui di riprodursi e morire è proporzionale al proprio livello di fitness.

-*Socialità:* può capitare che un'insieme di individui sottoposti al processo di selzione, possano interagire tra loro in collaborazione o in competizione. La fitness in questo caso non dipenderà solo dalle caratteristiche del singolo individuo, ma anche da quella degli altri individui coinvolti nell'interazione.



5 Schema a blocchi di Evorobot con chiamate alle funzioni dei vari sorgenti

3. Sviluppo di un ambiente di programmazione visuale.

In questa tesi ci siamo posti come obiettivo quello di sviluppare un nuovo tool di programmazione visuale che consentisse sia di creare e modificare facilmente dei sistemi di controllo behavior-based sia di sottoporre questi sistemi ad un processo di adattamento evolutivo.
Tale sistema è stato sviluppato estendendo le funzionalità del tool evorobot* descritto sopra.

3.1 Elementi logico/grafici

Nel percorso di creazione dell'interfaccia grafica per la programmazione visuale di Evorobot*, sono state sviluppate diverse unità che rendono possibile definire i comportamenti di un robot.
Le unità nel complesso rappresentano un sistema di controllo, il quale assume un insieme di comportamenti in risposta agli stimoli ricevuti.
Alcuni comportamenti che possono presentarsi :
-Se i sensori posti sulla parte sinistra del robot sono stimolati, il comportamento di risposta per il robot(come mostrato nella figura 6) sarà girare a destra ,
- se i sensori frontali non sono stimolati il robot avrà un comportamento di proseguire in avanti.
Questi comportamenti per essere realizzati hanno bisogno di diversi tipi di

unità.

I nodi sono costituti da sensori,che si comportano come unità di input, attuatori, che si comportano come unità di output infine, le unità di elaborazione del linguaggio di programmazione visuale che in base allo stato dei sensori attivano o disattivano lo stato dei attuatori.

La figura 6 mostra un esempio di sistema di controllo che consente al robot di evitare degli ostacoli.

Le unità in basso sono i sensori a infrarosso localizzati, il primo sul lato sinistro, il secondo a 45 gradi il terzo e il quarto frontalmente, il quinto a 45 gradi verso destra e il sesto sul lato destro. Tali unità hanno uno stato di attivazione tra 0 e 1. Lo stato di attivazione corrisponde a 0 in assenza di oggetti e tende a 1 avvicinandosi a un oggetto.

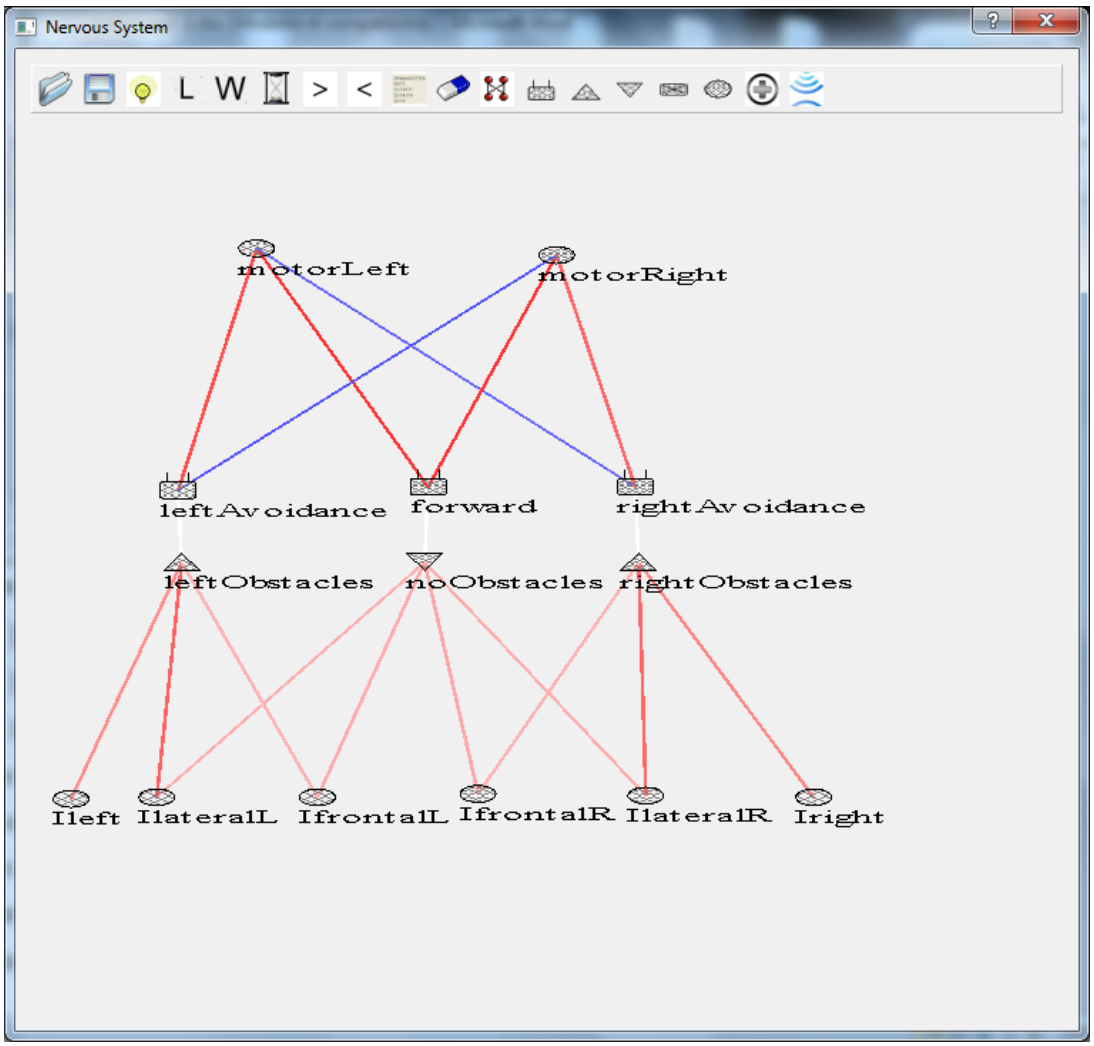
Le unità del secondo strato dal basso, sono delle unità if che si attivano se la soglia di accensione-spegnimento è superata dalla somma dei valori delle connessioni in input per lo stato di attivazione dei sensori. Il valore della soglia è impostato dall'utente come anche il peso delle connessioni. Quindi il nodo leftAvoidance (della figura 6) con una soglia pari a 1,0 e con il peso delle connessioni in entrata uguali a 1.0, si attiva se i tre sensori connessi si attivano a 0,4.

Le unità del terzo strato sono delle motor-unit, queste decidono i valori di attivazione degli attuatori, sulla base di quante if-unit e if-unitmin connesse sono attive.

La motor-unit turnRight si attiva se il nodo leftAvoidance è attivo. Tale nodo ha due connessioni con due attuatori, che sono il motore sinistro e il destro: con il motore sinistro ha una connessione con valori positivi e con il motore destro ha una connessione con valori negativi inducendo cosi il robot a girare a destra.

Le unità in alto (vedi figura 6) sono i motori "sinistro e destro " che

possono avere uno stato di attivazione da 0 a 1 con 1 che significa avanti e 0 indietro. I motori si attivano se si ha almeno un motor-unit attivo; se ci sono più di un motor- unit attivi, la somma dei pesi delle connessioni in entrata per gli stati di attivazione di ciascun motor-unit, è divisa per il numero di motor- unit attive.



6 esempio di rete realizzata per risolvere il problema Obstacle Avoidance

Nel caso di un if-unit se l'attivazione di un sensore per il peso della connessione è maggiore rispetto alla soglia della suddetta unità, l'if-unit attiva la motor-unit. Per ciò che concerne le motor-unit queste si attivano sulla base di quante if-unit e di if-unitmin connesse, sono attive.
All'attivazione delle motor-unit si avviano gli attuatori connessi.

Gli attuatori si attivano in base al numero delle motor-unit connesse, ai pesi delle connessioni e sulla base dello stato di attivazione di tali unità. Inoltre gli attuatori dividono il valore ottenuto dal prodotto delle connessioni per lo stato di attivazione delle motor-unit per il numero di motor-unit che hanno uno stato di attivazione superiore allo zero.

 Sensore.

Un sensore è uno strumento capace di percepire la presenza di oggetti, le luci, le immagini, i diversi stati della pinza. La percezione di questi oggetti esposti lungo il suo campo visivo, è espressa in un valore numerico chiamato stato di attivazione.

Lo stato di attivazione di tale nodo varia tra 0.0 e 1.0.

La funzione update_ncontroller è chiamata dalla funzione trial presente nel file robot-env.cpp; tale funzione come parametro ha l'array input[].

Per ogni sensore si procede con la seguente assegnazione:

```
act[i] = input[i];
```

 If-unit.

Si tratta di un'unità che permette l'attivazione delle motor-unit sulla base del netinput, definito quest'ultimo dai sensori connessi agli if-unit e dalle connessioni con questi sensori.

Tale unità si attiva se la soglia inpostata è superata. Il valore che è confrontato con la soglia dell' if-unit è uguale alla somma dei valori dei sensori connessi per le connessioni .

In tale unità può essere cambiata la soglia.

```
for(j=0 ;j <nodes[id][i].inputnn ;j++){
     netinput[i] += act[nodes[id][i].inputid[j]] *
     nodes[id][i].inputv[j];
}
if ( netinput[i] > nodes[id][i].valueNGBD[0] ){
     act[i]=1.0;
```

```
}else{
      act[i]=0.0;
}
```

If-unitmin.

Si tratta di un'unità che permette l'attivazione delle motor-unit sulla base del netinput, definito quest'ultimo dai sensori connessi agli if-unitmin e dalle connessioni con questi sensori.

Tale unità si disattiva se la soglia inpostata è superata. Il valore che è confrontato con la soglia dell if-unitmin è uguale alla somma dei valori dei sensori connessi per le connessioni .

In tale unità può essere cambiata la soglia.

```
for(j=0 ;j <nodes[id][i].inputnn ;j++){
      netinput[i] += act[nodes[id][i].inputid[j]] *
      nodes[id][i].inputv[j];
}
if ( netinput[i] < nodes[id][i].valueNGBD[0] ){
      act[i]=1.0;
}else{
      act[i]=0.0;
}
```

Motor-unit.

La motor-unit è l'unità che controlla gli attuatori in base alla sua attivazione che è dovuta alle if-unit e alle if-unitmin.

Tale unità si attiva in percentuale a quante if-unit / if-unitmin connesse, si attivano. Inoltre è possibile impostare la durata dell'attivazione tramite il timer.

```
for(j=0 ;j <nodes[id][i].inputnn ;j++){
      if(nodes[id][nodes[id][i].inputid[j]].tipo == 4 ||
      nodes[id][nodes[id][i].inputid[j]].tipo == 6){
            if(act[nodes[id][i].inputid[j]]>0.5)iftrue++;
            else iffalse++;
      }
}
act[i]=(float)iftrue/(float)(iftrue + iffalse);
```

- Motor-block-unit.

  Serve per attivare un motor-unit alla volta, tra quelli connessi a lei. Permette così di far compiere lavori esclusivi al robot.
  Quest'unità ricerca tra le motor-unit connesse quella più attiva, imposta così lo stato di attivazione di tutte le altre a 0.0.

- Attuatore.

  Può essere un motore, una pinza, uno speaker. Si attiva in base alle motor-unit, compiendo così spostamenti del robot oppure può emettere suoni e può prendere oggetti.
  Ha come netinput la somma delle connessioni per lo stato di attivazione del motor-unit, tale somma è normalizzata nel rango 0.0 1.0.

```
for(j=0 ;j <nodes[id][i].inputnn ;j++){
     if(nodes[id][nodes[id][i].inputid[j]].tipo == 3){
          netinput[i] += act[nodes[id][i].inputid[j]] *
          nodes[id][i].inputv[j];
          if(act[nodes[id][i].inputid[j]]>0.0){
               motoruactive++;
          }
     }
}
act[i]=logistic((float)netinput[i]/(float)motoruactive);
```

3.2 Interfaccia grafica

L'interfaccia grafica è stata realizzata con i seguenti linguaggi di programmazione: C e la libreria grafica di QT 4.7.4.
L'ambiente di sviluppo utilizzato è Microsoft Visual C++ Studio 2010 Express e come editor di testo si è utilizzato ConTEXT.
L'interfaccia grafica per la programmazione visuale, permette la

programmazione dei comportamenti dei robot. La programmazione si realizza per mezzo della connessione dei sensori con le if- unit, quest'ultime con le motor -unit e infine le motor -unit con gli attuatori.

La realizzazione della rete è possibile grazie all'aggiunta di nodi interni che possono diventare di quattro tipi: if-unit, if-unitmin, motor-unit, motor-block-unit. Tali elementi possono essere connessi da archi con pesi definibili dall'utente. Per quanto concerne i nodi sensori e i nodi attuatori, il loro numero è definito dal file di configurazione **.cf** .

In questo file è possibile definire il tipo di sensori(infrarossi, luce, sonori, visivi e di ground)che vogliamo aggiungere e che sono visualizzati come cerchi e i tipi di attuatori che possono essere motori, pinze, sorgenti luminose e sorgenti sonore. I motori sono visualizzati nella forma di cerchi.

L'interfaccia grafica permette di creare, modificare, visualizzare le unità, interconnetterle tra di loro per poi modificarne e visualizzarne i parametri.

Un sensore può essere connesso tramite un arco con una o più if-unit o una o più if-unitmin. L'arco di connessione avrà un peso definibile dall'utente.

All'accensione dell'interfaccia di programmazione visuale, i cerchi che rappresentano i motori si trovano nella parte superiore della finestra, al contrario dei sensori che si trovano nella parte inferiore.

I motor-block-unit, motor-unit, le if-unit, if-unitmin sono creati in seguito alla trasformazione di un item avente la forma di cerchio oppure attraverso la trasformazione di qualsiasi altro item.

Per far sì che la trasformazione avvenga, è necessario selezionare

l'item che s'intende trasformare in motor-block-unit e cliccare il bottone  che si trova nella barra dei strumenti. Tale item è visualizzato nella seguente forma  , una volta connesso alle motor-unit è visualizzato nella forma di un rettangolo rosso che include tutte le motor-unit connesse.

Per trasformare un item in motor-unit ,lo si seleziona e si clicca il bottone  che si trova nella barra dei strumenti. Tale item è visualizzato nella seguente forma  .

Per trasformare un item in if-unit lo si seleziona e si clicca il bottone  che si trova nella barra dei strumenti. Tale item è visualizzato nella seguente forma  .

Per trasformare un item in if-unit ,lo si seleziona cliccando il bottone  che si trova nella barra dei strumenti. L' item è visualizzato nella seguente forma  .

Per connettere le if-unit ai sensori basta selezionare con il mouse un qualsiasi sensore e poi selezionare un if-unit infine, cliccare il bottone Connect  .

Le connessioni in ingresso ai motor-unit si creano selezionando con il mouse la motor-unit: seleziono l'if-unit o la if-unitmin clicco poi sul bottone Connect . Per connettere la motor-unit con i motori è necessario selezionare il nodo motore e clicco il bottone  .

La soglia delle if-unit e delle if-unitmin è impostata selezionando una delle citate unità, premendo poi il bottone > per le if-unit e il bottone < per le if-unitmin infine, si preme il pulsante "+" o "-" sulla tastiera.

L'interfaccia grafica sviluppata per Evorobot* consente le seguenti opzioni:

- selezionare una, due, o gruppi di unità cliccandoci sopra con il mouse (l'unità selezionata cambia colore).
- Tramite un pulsante è possibile aggiungere un'unità interna aggiuntiva.
- Attraverso il pulsante cancella, se è selezionata un'unità, questa è cancellata, invece, se è selezionata una connessione o un gruppo di connessioni queste possono essere eliminate.
- Sono stati realizzati 4 pulsanti che permettono di cambiare un'unità in if-unit, if-unitmin, motor-unit, motor-block-unit. Azzerando ogni volta i parametri e le connessioni di tali nodi.
- Con il pulsante connetti si ha la possibilità di connettere un nodo o più nodi come input di un altro nodo o più nodi. Non è possibile creare connessioni tra tutti i tipi di nodi (per esempio si possono connettere le motor-unit ai nodi di output, ma non si possono connettere agli input).
- Per cambiare il peso di una connessione bisogna cliccare su tale connessione e in seguito tramite digitazione del pulsante "+" o "-" incrementare o decrescere il valore di suddetta connessione.
- Le connessioni che hanno la possibilità di essere modificate si trovano: tra i nodi motor-unit e i nodi output; tra i nodi input e i nodi if-unit/ifunitmin.

- Per modificare il valore delle if-unit si seleziona il nodo e in seguito si preme il pulsante identificato con ”>” appare il valore del dato nodo e può essere modificato con i tasti “+” e “-“.
- La stessa cosa vale per if-unitmin
- Per impostare il tempo di attivazione di una motor-unit la si deve selezionare poi si preme il pulsante 
- Per visualizzare i valori di tutte le if-unit/if-unitmin basta selezionare con il mouse il pulsante “>” e ”<”.
- Per vedere i pesi delle connessioni in entrata delle if-unit/if-unitmin basta selezionare l’unità, invece, attraverso le motor-unit si possono visualizzare i pesi in uscita.
- Per modificare il label dei nodi, bisogna fare doppio click sul nodo, tramite un widget, si inserisce il nome del nodo.
- Per salvare la reta basta premere il pulsante ***save***, la rete così sarà salvata nel formato ***.net***. Il salvataggio memorizza per ciascun nodo il tipo, la posizione, il valore soglia per le if-unit/if-unitmin, le unità in input e i valori delle connessioni con tali unità, il tempo di attivazione per le motor-unit, i label dei nodi e il numero dei nodi in input.
- Per l’apertura della rete salvata si utilizza il pulsante load.

3.3 Adattamento

Una rete realizzata dall'utente può essere evoluta tramite l'algoritmo di evoluzione presente nel file sorgente evolution.cpp .
Per l'evoluzione è stato necessario definire i parametri che devono essere sottoposti a evoluzione.
I parametri risultati idonei all'addestramento sono:
- i pesi delle connessioni tra i nodi input e i nodi if-unit/if-unitmin,
- i valori dei pesi soglia dei nodi if-unit/if-unitmin,
- i pesi delle connessioni tra i nodi motor-unit e i nodi output.

In caso si voglia addestrare una rete in modo automatico, basta una volta definita la rete, caricarla e selezionare la modalità di addestramento sia in adattamento "adapt", sia in evoluzione "evolve".

L'algoritmo di evoluzione genera stringhe di numeri interi con valore che varia tra 0 e 256. La dimensione di queste stringhe è uguale al numero delle variabili libere da addestrare. Il numero di tali variabili è calcolato con la funzione `compute_nfreep`.
Ogni genotipo è trasformato prima in fenotipo dalla funzione `genotype_to_phenotype,` in seguito è testato sulla rete producendo così una fitness: il risultato è confrontato con i fitness degli altri figli.
Il migliore dei figli sopravvive gli altri, invece, scartati. Tramite il file di configurazione (.cf) si definisce il numero dei figli che sopravvive.

Ciascun elemento di `gi` che è l'array di genotipi dichiarato in evolution.cpp è tradotto nel fenotipo attraverso i seguenti calcoli:

```
value =  (*gi + 1) / 256.0;
nodes[i][k].inputv[kk] = wrange - (value *(wrange * 2.0));
```

Per `wrange` si intende l'intervallo nel quale sono trasformati gli elementi del genotipo `gi`. L'intervallo parte da `-wrange` a `+wrange`.
Gli elementi cosi trasformati detti fenotipi si assegnano ai parametri liberi della rete.

3.4 Funzioni sviluppate

Le difficoltà riscontrate sono state molteplici: in primis l'identificazione delle diverse variabili necessarie a far comunicare la parte del software esistente con le nuove funzioni scritte; la comprensione delle funzioni presenti nel sistema, in modo da poter essere riscritte con la nuova struttura dati .
Il percorso ha previsto il passaggio da una rete valutata a blocchi, in una dove ciascun nodo puo avere un tipo diverso . Per realizzare tali reti ho utilizzato la struttura *node*; la quale definisce il tipo di nodo, i nodi in input, i valori di tali nodi, i label dei nodi, la posizione, i timer e i valori di soglia.

Si è dovuto riscrivere totalmente le seguenti funzioni per quanto riguarda il file ncontroller.cpp:

- `init_network:` inizializza la rete allocando le diverse strutture e chiamando le dovute funzioni .
- `load_net:` legge da file .net lo schema della rete salvata con i corrispettivi valori.

- `save_net:` scrive sul file .net la rete creata.
- `create_net:` inizializza una rete in base ai dati passati dal file .cf.
- `genotype_to_phenotype:` trasforma il genotipo in fenotipo.
- `phenotype_to_genotype:` trasforma il fenotipo in genotipo.
- `update_ncontroller:` è la funzione principale che aggiorna lo stato dei attuatori in base al tipo di rete.
- `compute_nfreep:` calcola il numero di variabili libere generate per ogni rete.

Riguardo il file di gestione dell'interfaccia utente sono state riscritte le seguenti funzioni:

- `erase_net:` elimina connessioni tra coppie di nodi, gruppi di nodi ed elimina nodi.
- `add_cblock:` crea connessioni tra nodi in base al tipo adeguato.
- `decreasev:` diminuisce il valore di connessioni e il valore di soglia degli if-unit.
- `increasev:` incrementa il valore di connessioni e il valore di soglia degli if-unit.
- `paintEvent:` disegna i nodi e le connessioni.
- `mousePressEvent:` seleziona i nodi e le connessioni .
- `mouseMoveEvent:` permette di spostare i nodi.
- `mouseReleaseEvent:` rilascia il nodo.
- `pseudo_activate_net:` copia i valori delle connessioni e i valori soglia degli if-unit nei corrispettivi nodi.

Aggiunte le seguenti funzioni:

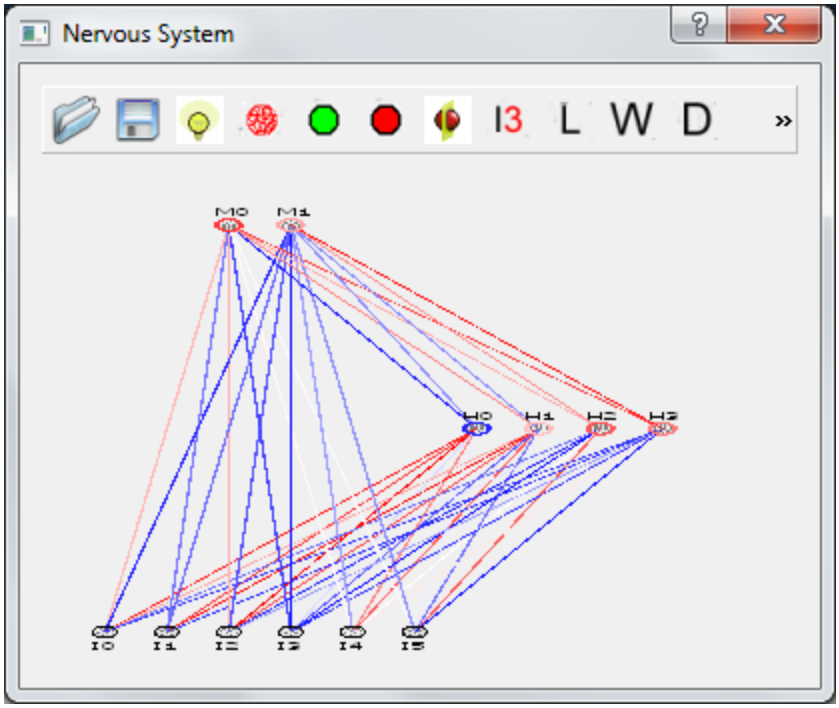
- `add_blockmotorunit:` trasforma un nodo in un block-motor-unit.
- `add_ifunit:` trasforma un nodo in un if-unit.

- `add_ifunitmin:` trasforma un nodo in un if-unitmin.
- `add_motorunit:` trasforma un nodo in un motor-unit.
- `display_if:` visualizza al posto del label il valore di soglia dell'if-unit.
- `display_ifmin:` visualizza al posto del label il valore di soglia dello if-unitmin.
- `add_intenalUnits:` aggiunge un nodo che può essere trasformato in if-unit, motor-unit e block-motor-unit.
- `set_timemotor:` imposta il numero di cicli per i quali un motor-unit deve rimanere attivo.
- `connectedID:` controlla se un nodo è connesso a un altro.
- `mouseDoubleClickEvent:` permette di rinominare il nodo.

4. Esempi di applicazione

Si è osservato per i test effettuati che:
Al crescere della complessità dei comportamenti dei robot, si riscontra una maggiore difficoltà nel realizzare, manualmente una rete capace di attuare comportamenti con delle fitness ottimali.
La rete neurale dimostra al contrario di avere un apprendimento buono, poiché esegue i comportamenti definiti dalla fitness nell'ambiente di test. Qualora tale ambiente (sul quale è stato addestrato il robot) subisse modifiche, la rete neurale deve essere riaddestrata affinché il robot possa eseguire i comportamenti desiderati.
Per quanto riguarda il rapporto tra la rete ***behavior-based*** progettata manualmente e la rete neurale per il test *Obstacle avoidance*, si possono ottenere risultati non uguali ma simili.
Con la fitness *Garbage Collecting*, i risultati dimostrano che la rete progettata a mano è meno abile nel raccogliere gli oggetti rispetto alla rete neurale, poiché con la fitness *Garbage Collecting* si riscontrano maggiori difficoltà nel definire il sistema di controllo e i comportamenti che deve mettere in atto.
Molteplici sono le possibilità che si presentano nel decidere quali e quanti nodi utilizzare, il numero di connessioni da realizzare, considerato che sono diversi i comportamenti che un robot deve assumere.
Per quanto riguarda il funzionamento delle reti neurali, i sensori e i motori si comportano allo stesso modo dei sensori e dei motori descritti per la rete ***behavior-based*** . Le unità di elaborazione sono costituire dai neuroni interni e dai neuroni motori che hanno uno stato di attivazione compreso

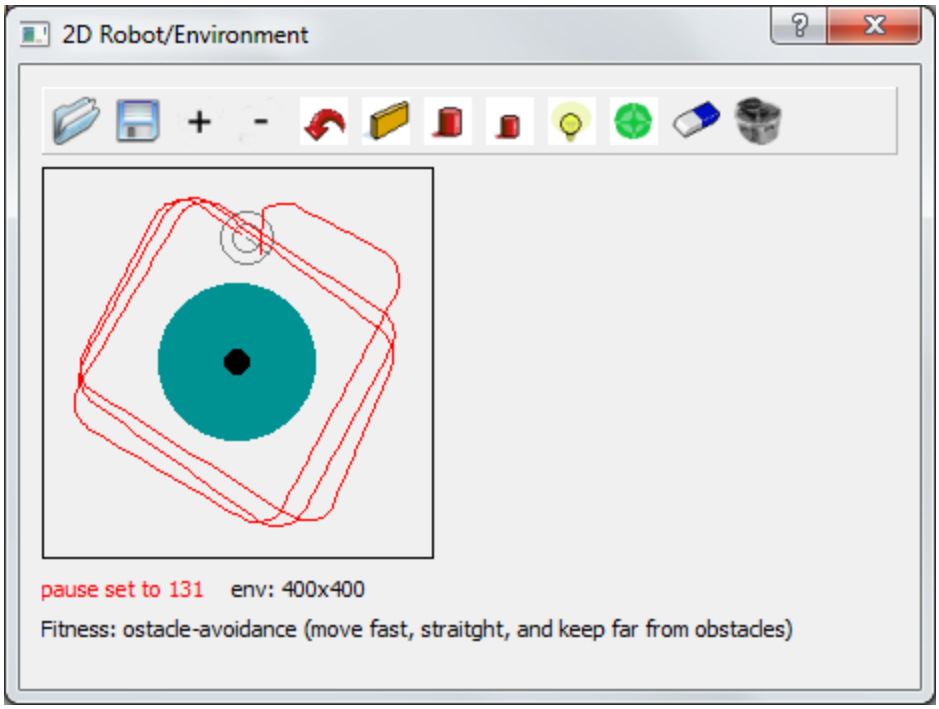
tra 0 e 1, ricevono una serie di connessioni pesate con valori tra -5 e +5. Lo stato di attivazione dei neuroni si ottiene calcolando il netinput, ovvero la somma algebrica dello stato del neuroni, che inviano la connessione moltiplicata per i pesi.



7 Rete neurale addestrata per l'evitamento ostacoli

4.1 Obstacle Avoidance

L'*Obstacle Avoidance* è tradotto in "evitamento ostacoli", tale funzione fitness premia il robot se si muove con velocità senza entrare in collisione con gli oggetti e i muri presenti nell'ambiente di test.
Robot realizzati con le reti neurali nell'ambiente di test hanno una fitness pari a 13.000-15.000 su 40 test di 500 cicli.
Tali robot nell'ambiente evitano gli ostacoli girando in una sola direzione.

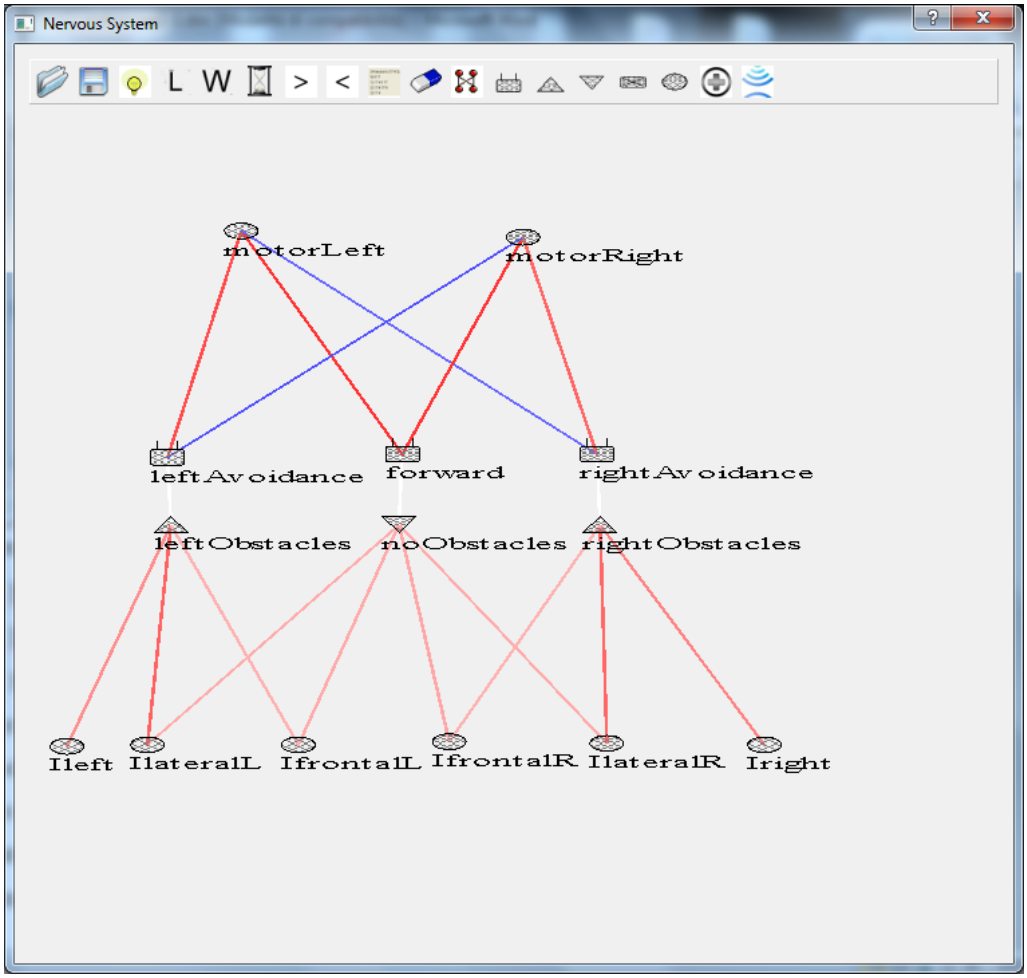


8 Esempio di comportamento di un robot definito dall'utente per quanto riguarda la fitness Obstacle Avoidance

Per ciò che concerne le reti realizzate manualmente, sono stati definiti i seguenti comportamenti:

- Evitamento ostacoli situati a sinistra,
- Evitamento ostacoli situati a destra,
- Proseguire avanti finchè non c'è un ostacolo.

Nei primi due comportamenti se i valori delle motor-unit sono impostati in modo simmetrico, tali comportamenti entreranno in conflitto l'uno con l'altro in una situazione di attivazione simmetrica dei neuroni. Risulta pertanto necessario che le motor-unit siano impostate in modo asimmetrico.



9 Rete realizzata per l'Obstacle Avoidance.

Tale rete realizzata a mano nell'ambiente di test ha una fitness pari a 11.000-13.000 su 40 test di 500 cicli.

La rete se evoluta automaticamente assume un comportamento simile alla rete neurale infatti, in caso di ostacoli gira in una sola direzione;

nell'ambiente di test ha una fitness pari a 10.000-11.000 su 40 test di 500 cicli.

Per quanto riguarda l'adattamento da rete manuale si ha un buon risultato tale da riportare una fitness pari a14.000.

Test svolti per l'evitamento ostacoli con 40 prove da 500 cicli

Tipo rete	Tipo di addestramento	Fitness	Comportamento
Neurale	Evoluzione	15.000	Evita in una sola direzione

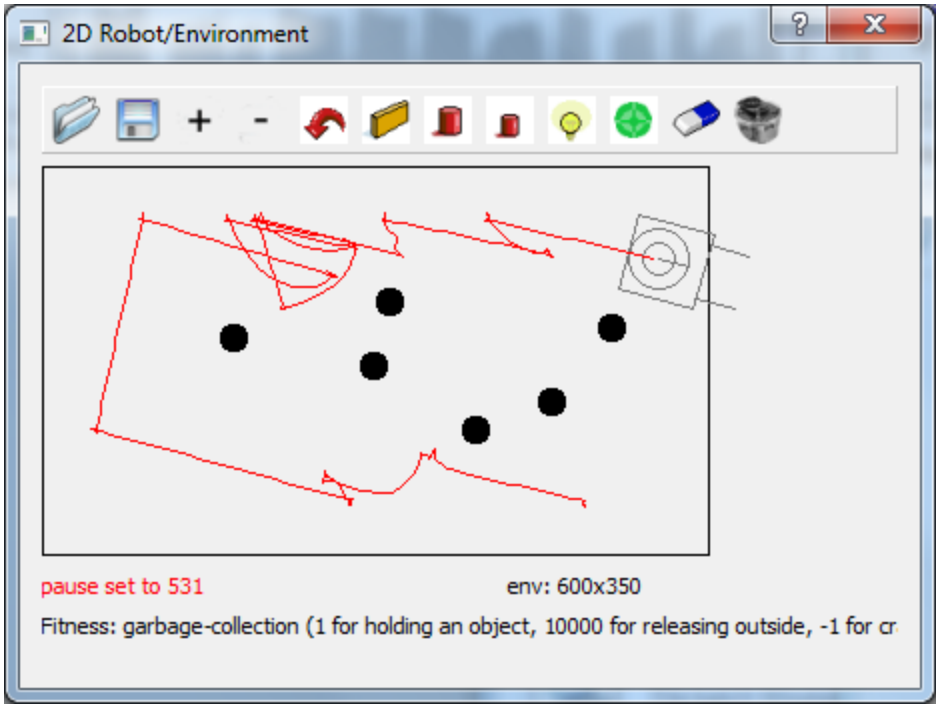
Behavior-based	manuale	13.000	Evita in entrambe le direzioni
Behavior-based	Evoluzione	11.000	Evita in una sola direzione
Behavior-based	Adattamento da rete manuale	14.000	Evita in una sola direzione

4.2 Garbage Collecting

La funzione *Garbage Collecting* tradotta letteralmente come "raccolta di rifiuti" richiede al robot la raccolta di oggetti presenti nell'ambiente di simulazione, per poi portarli fuori dall'ambiente di test.

La fitness premia il robot di 1 punto per ogni oggetto afferrato e di 10.000 punti per ogni oggetto rilasciato all'esterno dell'ambiente.

I robot addestrati attraverso le reti neurali, hanno la caratteristica di raccogliere e di evitare gli oggetti sempre dalla stessa direzione; la fitness prodotta da questi robot è pari a 2.000.000-2.200.000 punti su 40 test lunghi 500 cicli.



10 Esempio di comportamento di un robot realizzato con rete neurale per quanto riguarda la fitness Garbage collecting

Riguardo le reti realizzate manualmente, all' inizio del lavoro si era pensato di testare il modulo sviluppato per *Obstacle Avoidance*, nella parte della locomozione di *Garbage Collecting*.

Tale supposizione si è dimostrata negli step successivi (ovvero durante l'esecuzione dei test) inadeguata, considerato che il *Garbage Collecting* presuppone una ricerca di oggetti e non solo l'evitamento di ostacoli.

Per il *Garbage Collecting* definiamo tre comportamenti richiesti a livello generale che si suddividono in altri sotto comportamenti:

- **Evitamento ostacoli**:
  - Evitamento ostacoli situati a sinistra
  - Evitamento ostacoli situati a destra
  - Proseguire avanti finchè non c'è un ostacolo
- **Gestire la pinza**
  - Prelevare il cilindro se si trova davanti
    - Se il robot non si trova davanti al muro

        - Se sono attivi solo i sensori frontali
        - Se il sensore di cilindro preso, non è attivo
    - Posare il cilindro se i sensori frontali sono tutti attivi anche al 60% per valori di connessioni pari 0,6-0,8 con una soglia del if unit pari a 1,0-1,2.

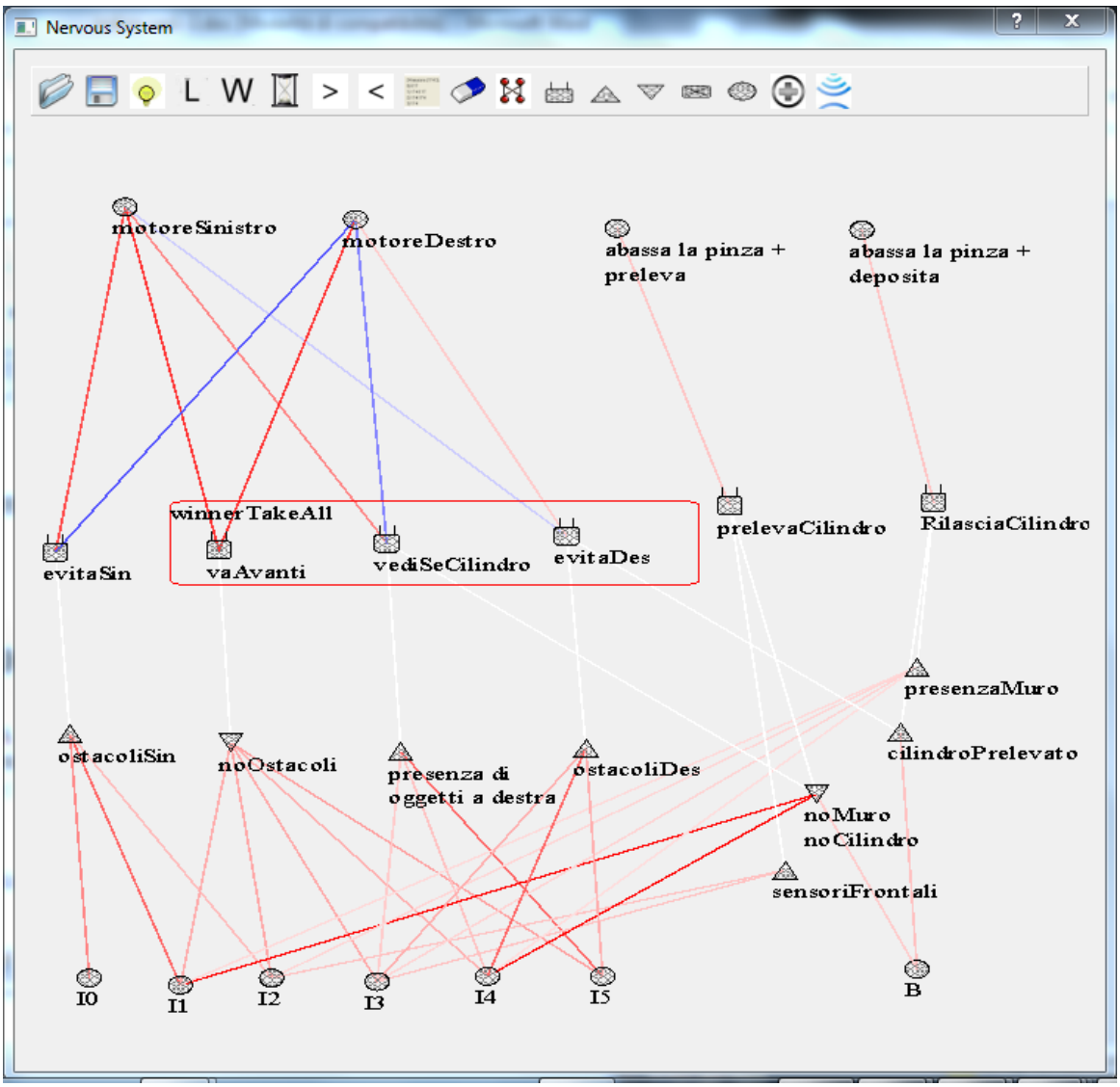
      Tale impostazione sarebbe stata pensata al fine di non posare l'oggetto in presenza di cilindri ma solo di muri,
- **Cercare i cilindri e trasportarli fuori dall' ambiente** (fase che interferisce con l'evitamento ostacoli, poiché il robot ha reazioni contrarie sugli stessi input)
    - Non possiedo l'oggetto così quando si attivano i sensori sulla destra controllo che cosa è:
        - se è un muro lo evito
        - se è un cilindro lo prelevo

          Tale comportamento è difficilmente attuabile poiché i sensori a infrarossi non hanno una percezione della forma, ma solo la presenza di un oggetto.

          A una data distanza con 6 sensori a infrarossi non si distinguono muri o cilindri facilmente, soprattutto con il solo utilizzo di sensori laterali: questo richiede al robot di ruotare su se stesso per controllare se ha di fronte a se un cilindro oppure un muro. Nel caso il cilindro è presente, il robot preleva l' oggetto altrimenti gira.
    - Possiedo l'oggetto voglio così evitare i cilindri, ma non i muri
        - se è un muro rilascio il cilindro

- se è un cilindro lo evito

I comportamenti elencati sono stati realizzati con la rete manuale, presente nella figura che segue.



11 Rete realizzata per la gestione del Garbage Collecting

Il conflitto tra il comportamento "evita a destra" e "controlla se il cilindro è a destra": è stato risolto con la connessione dei motor-unit corrispettivi e con il modulo motor-block-unit .

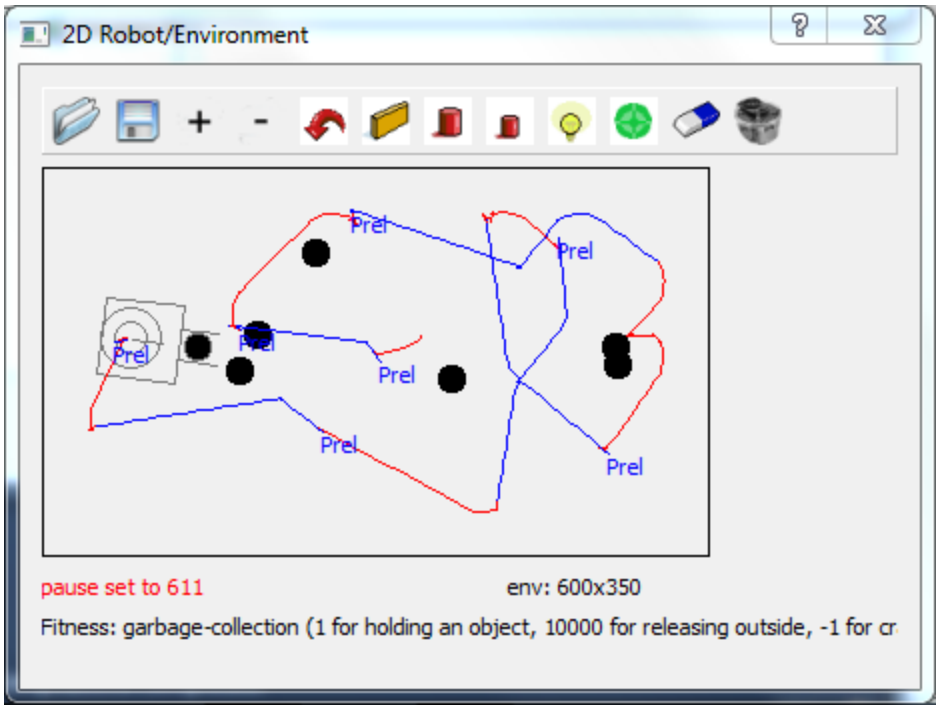
La motor-block-unit da precedenza all'unità più attiva: l' unità "Controlla se cilindro a destra" è attiva al 50% solo se l'if-unitmin è attiva finchè nessun cilindro è prelevato e il restante 50% percento è

attivo se l'if-unit connessa ai sensori di destra percepisce un oggetto, quindi se entrambe le if sono attive, la motor-unit è attiva al 100% girando a destra.

L'unità "Evita a destra" si attiva al 50% una volta che il cilindro è prelevato, mentre l'altro 50% si attiva se a destra si ritrova un ostacolo, si attiva il 100% se entrambe le condizioni sono soddisfatte.

L'unità "va Avanti" è stata connessa al motor-block-unit dato che, almeno una delle altre due è attiva al 50% , "va Avanti" vince finché non si attivano i sensori che identificano la presenza di oggetti.

Per migliore tale test è stato necessario introdurre il timer, poiché consente al robot di effettuare un'azione per un certo numero di cicli definito dall'utente.



12 Esempio di comportamento di un robot definito dall'utente per quanto riguarda la fitness Garbage Collecting (la traiettoria segnata con il colore blu indica il robot con un cilindro prelevato, quella segnata con il rosso indica l'assenza di cilindro prelevato).

Il timer è impostato per: "Controlla se cilindro a destra" dai 5 ai 15 cicli in

modo che qualora si disattivassero i sensori laterali, considerato che l'oggetto passa nel campo visivo frontale del robot, quest'ultimo può continua a voltarsi fino a che l'oggetto non è veramente di fronte.
Tra le varie reti realizzate a mano per tale ambiente di test la migliore ha una fitness pari a 400.000-500.000 su 50 test lunghi 500 cicli; si tratta in media di 1 cilindro per ogni test, rispetto ai 4 cilindri nei test compiuti sul robot, basato su reti neurali.

L'efficienza del robot realizzato con una rete a mano varia di test in test, si sono riscontrati casi in cui il robot ha rilasciato al di fuori dell'area 3-5 oggetti altre volte ha rilasciato 0-1 oggetti.
L'apprendimento automatico di questa rete è risultato, totalmente inefficiente. Questo sarà dovuto al fatto che ciascun modulo è stato definito per compiere un determinato comportamento con specifici pesi.
E' raro che l'algoritmo di apprendimento automatico comprenda tutte le variabili in modo adeguato.
Si è visto con altri tipi di reti realizzate a mano, nelle quali c'erano 53 nodi tra le if-unit, if-unitmin e motor-unit, che si può ottenere un buon risultato nell'apprendimento automatico.
In tali reti si è riuscito a ottenere una fitness pari a 1.500.000-1.700.000.
Per quanto riguarda l'adattamento automatico si riscontra per la rete in figura 11 un buon risultato, con una fitness pari a 1.000.000.

Test svolti per il Garbage Collecting con 50 prove da 500 cicli

Tipo rete	Tipo di addestramento	Nodi interni	Fitness best	Comportamento
Neurale	Evoluzione	6	2.300.000	Preleva e evita in una sola direzione
Behavior-based	Manuale	15	500.000	Preleva in una sola direzione + Evita in entrambe le direzioni
Behavior-based	Evoluzione	15	10.000	Niente di rilevante

Behavior - based	Evoluzione	53	1.700.000	Preleva e evita in una sola direzione
Behavior - based	Adattamento da rete manuale	15	1.100.000	Preleva e evita in una sola direzione

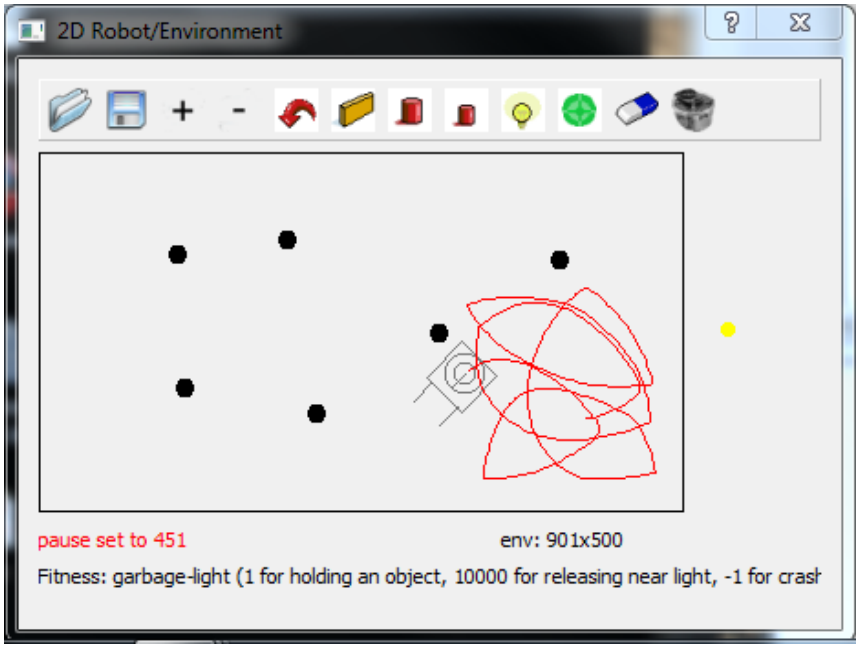
4.3 Navigazione

Questo test è un'estensione del *Garbage Collecting*. Rispetto al *Garbage Collecting* con il test di *Navigazione*, il robot non può depositare il cilindro al di fuori di una qualsiasi parete, ma solo sulla quella parete contraddistinta da un punto luminoso.

In tale test il robot necessita di sensori di luminosità che gli indicano la parete giusta dove depositare l'oggetto. I sensori di luminosità del robot hanno una percezione limitata a metà dell'area di test.

La fitness premia il robot di 1 punto per ogni oggetto afferrato e di 10.000 punti per ogni oggetto rilasciato al di fuori della parete contrassegnata dalla luce.

I robot addestrati attraverso le reti neurali, hanno la caratteristica di raccogliere e di evitare gli oggetti sempre dalla stessa direzione; la fitness prodotta da questi robot è pari a 500.000-1.000.000 punti su 50 test di 500 cicli.



13 Esempio del comportamento dei robot addestrati con reti neurali.

L'estensione della rete realizzata per *Garbage Collecting* non è stata problematica, è bastato aggiungere due motor-unit e tre if-unit. Queste componenti servono, una volta prelevato l'oggetto a dirigere il robot sulla sorgente luminosa.

Per il test di *Navigazione* si sono utilizzati i tre comportamenti definiti, precedentemente, per il *Garbage Collecting*:

- **Evitamento ostacoli.**
- **Gestire la pinza.**
- **Cercare i cilindri e trasportarli fuori dall'ambiente.**

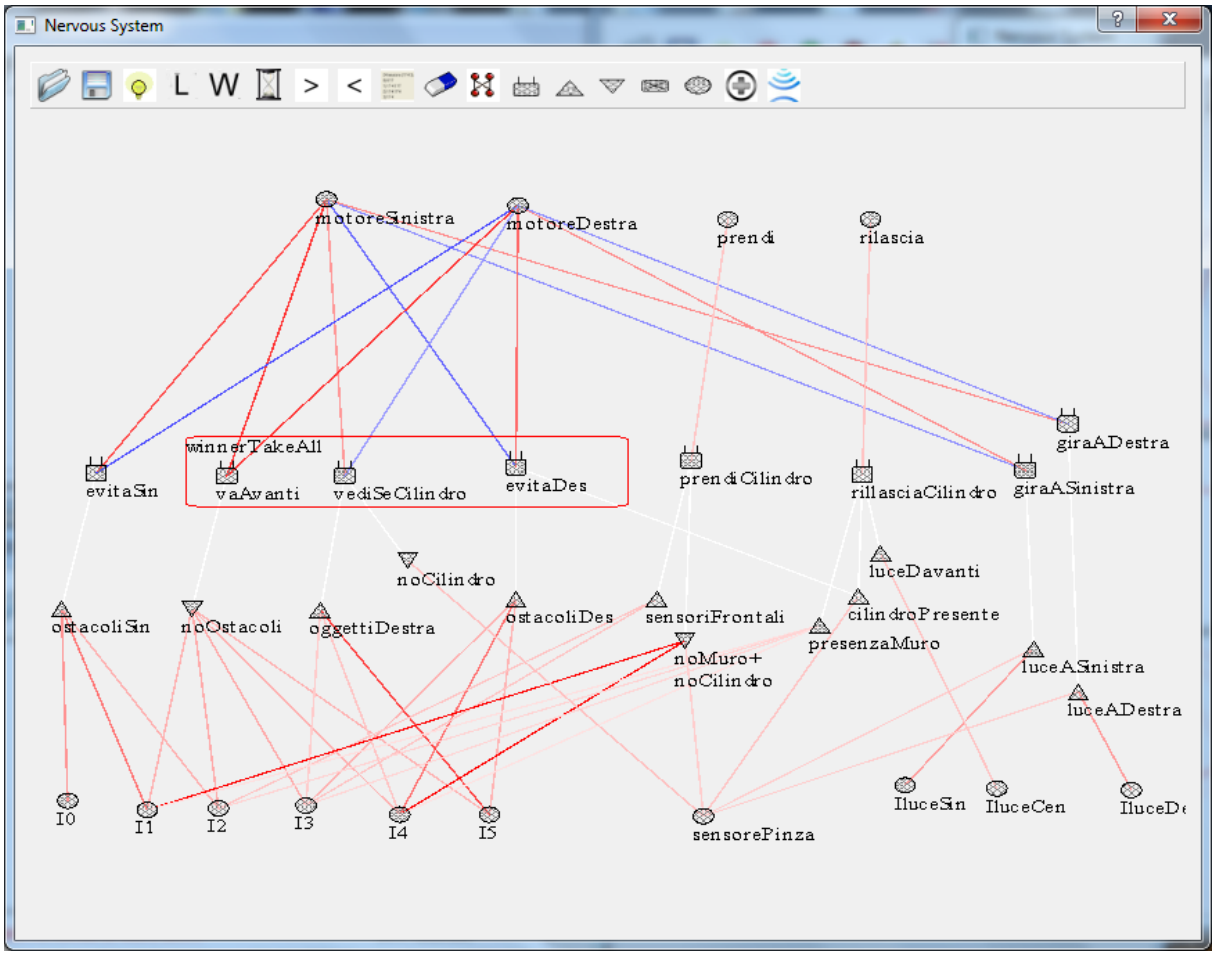
In aggiunta sono stati definiti i seguenti comportamenti:

- Se il cilindro è prelevato e la sorgente luminosa si trova a sinistra: il robot gira a sinistra,
- Se il cilindro è prelevato e la sorgente luminosa si trova a destra: allora il robot gira a destra,
- Se il cilindro non è prelevato, il robot non cerca la sorgente

luminosa,

- Rilascia l'oggetto se si attivano i sensori frontali, il sensore cilindro presente e se si attiva il sensore luce frontale

I comportamenti elencati sono stati realizzati con la rete manuale, presente nella figura che segue.

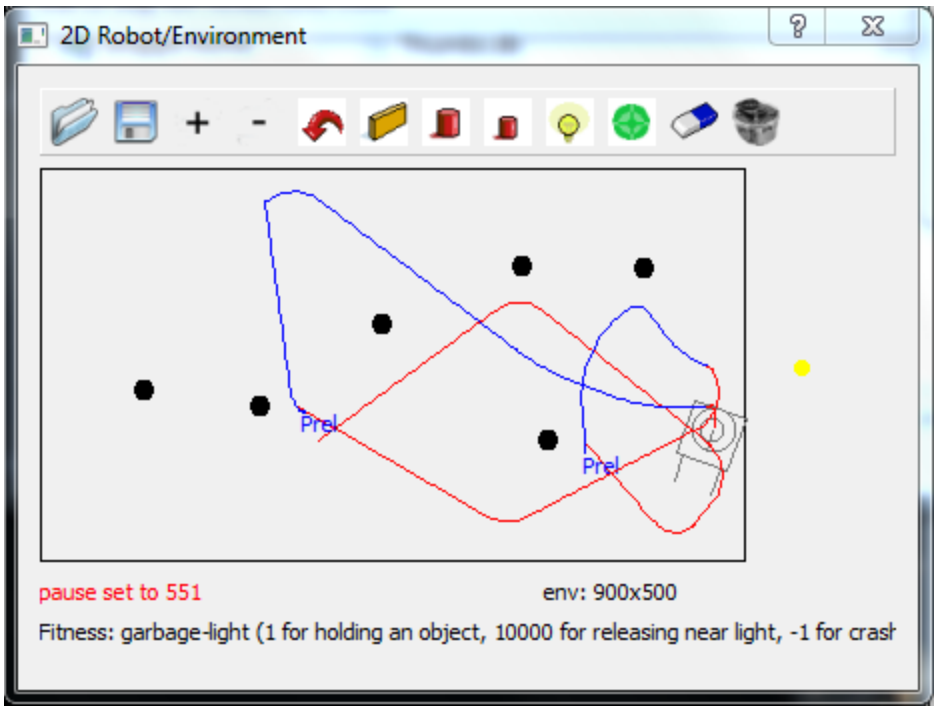


Tra le varie reti realizzate manualmente per tale ambiente di test la migliore ha una fitness pari a 300.000-400.000 su 50 test lunghi 500 cicli. Le reti realizzate dall'utente possono essere migliorate, ma necessita una attenta osservazione delle risposte fornite dal robot a ogni modifica.

La rete creata con le unità logiche ha un atteggiamento di esplorazione maggiore rispetto a quella realizzata con la rete neurale, che come si può vedere dalla figura sopra riportata, tende a ruotare sempre vicino alla

parete luminosa.

Questo atteggiamento molto probabilmente sarà una conseguenza del fatto che, i cilindri nell'ambiente si trovano equamente distribuiti, per cui la rete neurale non necessita di cercare i cilindri lontano dalla parete con la sorgente luminosa.



14 Esempio di comportamento di un robot definito dall'utente per quanto riguarda la fitness navigazione (la traiettoria segnata con il colore blu indica il robot con un cilindro prelevato, quella segnata con il rosso indica l'assenza di cilindro prelevato).

Come per il Garbage Collecting, l'addestramento automatico di questa rete è risultato inefficiente. Infatti, rispetto al precedente test si hanno solo 7 connessioni e 3 if in più, da sottoporre ad addestramento automatico.

Si riconferma che un addestramento generalizzato per reti definite in base ai comportamenti specifici, difficilmente riesce a trovare i giusti pesi delle connessioni e dei nodi per ottenere un robot con buoni risultati.

Per quanto riguarda l'adattamento automatico della rete realizzata dallo utente si ottengono delle fitness pari a 500.000-800.000, risultati simili alle reti neurali .

Test svolti per la Navigazione con 50 prove da 500 cicli

Tipo rete	Tipo di addestramento	Nodi interni	Fitness best	Comportamento
Neurale	Evoluzione	6	1.000.000	Preleva e evita in una sola direzione
Behavior-based	Manuale	21	400.000	Preleva in una sola direzione + Evita in entrambe le direzioni
Behavior-based	Evoluzione	21	10.000	Niente di rilevante
Behavior-based	Adattamento da rete manuale	21	800.000	Preleva e evita in una sola direzione

5. Conclusioni

Problematiche

Durante l'attività di tirocinio si sono dovute affrontare alcune problematiche che vanno dai problemi prettamente tecnici all'inevitabile mancanza d'esperienza.
Il problema iniziale con cui ci si è dovuti confrontare è stato far funzionare le librerie di QT con Visual Studio express.
Difficoltà maggiori si sono riscontrate nella comprensione delle sorgenti di Evorobot*, poiché la loro quantità era notevole. Tuttavia quando si è preso atto, che il lavoro di tesi si doveva fondamentalmente focalizzare sulle sorgenti: ncontroller.cpp e rend_controller.cpp per i quali è stata necessaria un'analisi approfondita e una riscrittura in più fasi.
Tutto questo perché la definizione iniziale del progetto non chiariva gli obiettivi e i requisiti del sistema definitivo; inizialmente si pensava che la rete neurale dovesse essere conservata, in seguito però il lavoro si è concentrato sulla programmazione visuale eliminando così tutte le variabili, le strutture tradotte dal sistema a reti neurali.
Una volta definito la forma dell'interfaccia grafica e delle componenti che dovevano essere implementate, il lavoro ha acquisito una forma più nitida.
Riguardo la sperimentazione della programmazione visuale numerose difficoltà si sono manifestate. Sono state pertanto necessarie apportare differenti variazioni al sistema di aggiornamento dei nodi e all'utilizzo dell'interfaccia grafica.

Eventuali sviluppi futuri

L'interfaccia di programmazione grafica affinché diventi matura dovrebbe essere adeguata all'esigenze dell'utente tramite diversificati test.
Altri elementi logico-grafici potrebbero essere aggiunti, si potrebbe anche realizzare una connessione a più livelli tra costrutti dello stesso tipo in modo da poter creare condizioni di programmazione più complesse.
Potrebbe, inoltre essere aggiunto, dal mio punto di vista, un apprendimento automatico più specifico e con la possibilità di definire intervalli di adattamento per ogni componente.
Grazie all'operatività e alla notevole competenza del Dott. Nolfi , il lavoro ha dato i suoi risultati.

Ringraziamenti

Ringrazio per il supporto in ogni fase di realizzazione della tesi, il Dottor Stefano Nolfi e il Professor Sterbini, che mi hanno dispensato di utilissimi consigli.

Chiara per l'appoggio morale e una tenacia infinita.

Jan e Zofia per la loro pazienza e sostegno.

Un grazie infine ai amici per i momenti di distrazione.

Bibliografia

D. Parisi, *Mente: i nuovi modelli della vita artificiale*, Il Mulino, Bologna, 1999.

M. Mirolli, S. Nolfi, *Evolution of Communication and Language in Embodied Agents*, Berlin Heidelberg Springer, 2010.

S. Nolfi, *Che cos'è la robotica autonoma*, Carrocci, Roma, 2010.

R.A. Brooks- Massachusetts Institute of Technology. Artificial Intelligence Laboratory, *Intelligence without reason*, Defense Technical Information Center, 1991.

http://ni.com/labview/whatis/graphical-programming/i/

http://it.wikipedia.org/wiki/LabVIEW

http://.mackacademy.com/ICA/blogs/MackAcademy.nsf/dx/labview-for-lego-mindstorms-nxt

http://.gostai.com/support/faq/

http://.sharprobotica.com/2010/07/qa-of-urbi-a-robot-os/

http://en.wikipedia.org/wiki/URBI

http://.federica.unina.it/corsi/sistemi-governo-robot/#cattedra

http://.absoluteastronomy.com/topics/Robotic_paradigms

http://.delucagiovanni.com/files/Navigazione%20di%20robot_file/frame.htm

http://users.dimi.uniud.it/~antonio.dangelo/robolab/refs/sr/sr.html

http://.storns.net/mediawiki/index.php?title=Evolvere_robot_stigmergici_in_Evorobot*_-_Cosa_può_fare_una_formica_con_pochissimi_neuroni

http://laral.istc.cnr.it/nolfi/garbage_collecting/

http://en.wikipedia.org/wiki/Evolutionary_robotics

http://.evolutionaryrobotics.org

http://drdobbs.com/database/231400148