Modelli multi-agente per robot LEGO Mindstorms applicati a soccer robot per RoboCup

Pini Alberto

Università degli Studi di Bologna
Seconda Facoltà di Ingegneria, Cesena

Corso di Sistemi Multi-Agente LS

a.a. 2009-2010

Obiettivo

Si vogliono analizzare alcuni modelli di coordinazione ed interazione multi-agente sperimentati su robot LEGO Mindstorms e confrontarli con modelli utilizzati nei robot da competizione della RoboCup, per poter valutare una possibilità di integrazione ed applicazione di tali pattern.

Ci si pone quindi, quindi, l'obiettivo di ipotizzare poi un particolare tipo di scenario di utilizzo di questi sistemi di coordinazione ad agenti collaudati in laboratorio su robot usati in ambito agonistico per disputare la RoboCup.

1. Introduzione

Dal 1997 la RoboCup è una delle competizioni di maggior rilievo in campo robotico; oltre ad avere un fine ludico permette di sperimentare nuove forme e modelli di coordinazione, infatti essa promuove lo sviluppo e la ricerca nel campo dell'intelligenza artificiale e della robotica permettendo di valutare concretamente teorie, algoritmi ed architetture ad agenti. Infatti, affinchè un robot giochi a calcio ragionevolmente bene, deve essere integrata un'ampia gamma di tecnologie e numerosi passi tecnici devono essere portati a termine. Il range di tecnologie spazia dalla ricerca in IA a quella in robotica, come la progettazione di agenti autonomi, la collaborazione multi-agente, la consapevolezza strategica, il ragionamento e la pianificazione in real-time, l'intelligenza robotica, la percezione sensoriale, e così via.

I robot sono quindi il più classico esempio di sistema multi-agente: noi avvertiamo un robot come la realizzazione concreta di un agente software che cerca di destreggiarsi in un ambiente fisico. In realtà ogni robot fisico può essere controllato da uno o più agenti software, inglobati come propria intelligenza interna o che hanno il controllo del robot in remoto. Nel complesso, però, semplificando, un robot è un'entità autonoma definita solamente all'interno di un ambiente e che utilizza sensori per percepire i cambiamenti di quest'ultimo, interagendo poi di conseguenza con esso tramite i suoi attuatori per portare a termine il proprio compito, al pari di un agente software.

In ambito universitario buoni risultati si sono ottenuti tramite la sperimentazione in laboratorio con robot LEGO Mindstorms che offrono un buon compromesso qualità/prezzo e permettono di costruire il robot nella forma necessaria in maniera semplice e veloce.

Grazie ad essi si sono potuti sviluppare modelli di coordinazione e comunicazione ad agenti, in particolare basati sul linguaggio e sul motore ReSpecT.

Tuttavia questi tipi di robot non sono adatti per la RoboCup: hanno una tecnologia troppo limitata, poco personalizzabile (esistono solo pochi tipi di sensori predefiniti) e non sempre precisa ed ottimale, inoltre il punto più critico rimane la fragilità della struttura che è in LEGO e non è neanche adatta ad essere di grosse dimensioni.

I robot che invece vengono in genere impiegati nella RoboCup sono robusti ed abbastanza massicci (giunture, corpo e parti meccaniche in solida plastica o in metallo, anche saldate), dotati di sensori ed attuatori costruiti o scelti ad hoc (spesso sofisticati, come le videocamere), ma il loro comportamento è in generale molto elementare e privilegia l'individualità.

Si vuole valutare la possibilità di adattare i modelli ad agenti sviluppati sui robot LEGO Mindstorms ai robot usati nelle competizioni RoboCup.

In questo articolo innanzitutto verrà fatta una panoramica sulla RoboCup per capire come essa è strutturata, poi si procederà con l'illustrare alcune soluzioni di robot/agenti software adottate per giocare nella RoboCup, distinguendo fra le varie competizioni e soffermandosi soprattutto sulla parte di comportamento ed interazione. Successivamente verranno presentati i robot LEGO Mindstorms come strumento ideale per la sperimentazione in laboratorio; si introdurranno i concetti fondamentali su alcuni mezzi di coordinazione (in particolare lo spazio di tuple) e si descriverà poi l'infrastruttura a livelli per la comunicazione ad agenti, basata su Situated ReSpecT, poi ci si soffermerà sullo sviluppo della parte relativa al comportamento che sta al livello applicativo. Infine, dopo aver mostrato alcuni casi d'uso di tale infrastruttura ad agenti sui robot LEGO Mindstorms, si proverà ad ipotizzare una generica soluzione di adattamento del modello comportamentale sviluppato, ai robot RoboCup da competizione.

1.1. MAS nella RoboCup

La RoboCup coinvolge numerosi aspetti di ricerca:

- Architettura ad agenti in generale
- Combinazione dell'approccio reattivo con quello basato su modelli e pianificazione
- Riconoscimento, pianificazione e ragionamento in tempo reale
- Ragionamento ed azione in un ambiente dinamico
- Fusione delle percezioni sensoriali

- Sistemi multi-agente in generale (ma in particolare la collaborazione)
- Comportamento di apprendimento per compiti complessi
- Acquisizione di strategia
- Modellazione della conoscenza in generale

Ci si rende conto che tutti questi aspetti si possono ricondurre principalmente a 2 macrodiscipline dell'informatica: l'intelligenza artificiale ed i sistemi multi-agente in ambito distribuito, che alla fine si integrano tra loro. Sotto il profilo dei sistemi multi-agente, si può affermare che un gioco di calcio è costituito da uno specifico (e molto attraente) ambiente multi-agente in real-time: ogni robot (reale o simulato), infatti, possiede un'architettura ad agenti e ne ha le caratteristiche principali (è autonomo, proattivo, situated e sociale).

I robot attualmente esistenti sono stati creati per realizzare in linea di massima singoli comportamenti come spingere/avanzare/ruotare con la palla, palleggiare o colpire. Un robot giocatore dovrebbe essere ideato in modo tale da eseguire anche più compiti composti insieme come tirare (che include il fatto di calciare), scartare (che include il fatto di spingere la palla in avanti), passare, colpire di testa e lanciare una palla che spesso specializzano un comportamento elementare. Gli agenti devono avere una conoscenza sia locale che globale; per interagire con l'ambiente è spesso necessario che abbiano un approccio reattivo, poiché più veloce, tuttavia esso non basta per competere in un gioco in cui sono anche fondamentali sia la strategia globale che la tattica locale, le quali però necessitano di un processo di ragionamento e pianificazione.

Le squadre competono per un grande obiettivo comune, cioè vincere la partita, ma gli obiettivi delle 2 squadre, considerati insieme, ovviamente, sono incompatibili e contrastanti: la squadra avversaria può essere vista come un ambiente dinamico ed ostico, che disturba il raggiungimento dell'obiettivo comune della propria squadra. Per raggiungere questo scopo ogni squadra deve portare a termine il più possibile il sottoobiettivo di segnare, ma per fare ciò bisogna che ogni membro della squadra abbia un comportamento veloce, flessibile e cooperativo, considerando sia la situazione locale che globale.

Le sfide più comuni dal punto di vista dei sistemi multi-agente sono:

- L'ambiente di gioco che è estremamente dinamico
- La percezione di ogni giocatore che potrebbe essere limitata localmente
- I giocatori potrebbero avere ruoli differenti
- La comunicazione tra i giocatori è spesso limitata, ed in generale occorre adottare un comportamento principalmente autonomo e flessibile.

2. RoboCup

Prima di analizzare alcuni possibili tipi di robot bisogna soffermarsi su come è strutturata la RoboCup, poiché i robot cambiano notevolmente in base alla modalità di gioco (potrebbero essere anche solo agenti software).

Innanzitutto occorre dire che la *RoboCup* è divisa in 4 diverse discipline:

- ***RoboCup Soccer***
- ***RoboCup Rescue***
- ***RoboCup@Home***
- ***RoboCupJunior*** ***(Soccer Challenge, Dance Challenge, Rescue Challenge, General)***

La ***RoboCup Soccer***, l'unica che tratteremo, a sua volta si suddivide in 5 categorie:

- ***Standard Platform League***

  Tutte le squadre usano lo stesso tipo di robot, questo permette ai vari team di concentrarsi soprattutto sullo sviluppo software piuttosto che sulla meccanica dei robot. I robot operano completamente in maniera autonoma, senza controllo remoto né da parte umana né tramite computer.

- ***Small Size League (aka F180)***

  Due squadre di 5 robot di limite massimo di 18 cm di diametro e 15 cm di altezza giocano a calcio con una pallina da golf arancione. Sono identificati e rintracciati da telecamere poste in alto, connesse con un computer fuori campo. Lo stato dei robot e della palla, inclusi i loro ID e la loro posizione sono poi mandati ai computer dei team. I loro software di intelligenza artificiale mandano poi i comandi ai propri robot basandosi sui dati visivi.

  E' una delle competizioni più vecchie della RoboCup Soccer e si concentra sul problema della cooperazione intelligente tra multi robot/agenti e sul controllo in un ambiente estremamente dinamico con un sistema ibrido (centralizzato/distribuito).

- ***Middle Size League***

  Due squadre di al massimo 6 robot di medie dimensioni (non più di 50 cm di diametro) e con tutti i sensori incorporati giocano a calcio su un campo in scala simile a quelli per umani con un pallone regolamentare FIFA. La comunicazione (diretta o mediata) tra robot, se presente, è basata su comunicazioni wireless. Non è permesso alcun intervento umano esterno, a parte l'inserimento o la rimozione dei robot in/dal campo.

  La ricerca si concentra sulla completa autonomia e cooperazione a livello di pianificazione e percezione.

- ***Simulation League***

  E' caratterizzata da giocatori software mobili indipendenti (agenti) che giocano a calcio in un campo virtuale dentro ad una simulazione al computer. Prevede diversi tipi di simulazione: *2D Soccer League*, *3D Soccer League*, *3D Development*, *Mixed Reality Soccer Simulation*.

  Insieme alla Small Size League, è tra le competizioni più vecchie della RoboCup Soccer. Si concentra sullo sviluppo ed applicazione di tecniche di intelligenza artificiale e sulla strategia di squadra.

- ***Humanoid League***

  Robot autonomi dall'aspetto e dai sensi simili a quelli umani giocano a calcio uno contro l'altro. Diversamente dai robot umanoidi al di fuori della *Humanoid League*, la percezione e la modellazione del mondo non sono semplificati da un sistema e da valori sensoriali che non siano umani.

  I robot sono divisi in 3 classi:

  - *KidSize* (30-60 cm di altezza). Due squadre da 3 robot.
  - *TeenSize* (100-120 cm). Due robot competono tra loro.
  - *AdultSize* (130 cm ed oltre). Due robot si sfidano ai rigori.

  Si concentra sulla fedele riproduzione del comportamento e delle limitazioni sia fisiche che percettive dell'uomo.

L'ambizioso ed utopistico obiettivo finale della RoboCup, ovviamente irraggiungibile (almeno per ora) ma a cui si cerca di tendere idealmente, è realizzare un team di robot che riesca a battere la nazionale di calcio del Brasile (o più modestamente, una squadra che riesca a giocare come veri giocatori umani).

2.1. Soccer Robot

In questa sezione ci preoccuperemo di illustrare alcuni esempi molto interessanti di robot realizzati per disputare la RoboCup Soccer.

Ogni volta che sarà possibile, ci si soffermerà soprattutto sul comportamento che contraddistingue tali robot, ossia sulla struttura degli agenti che lo controllano, e sull'interazione tra essi.

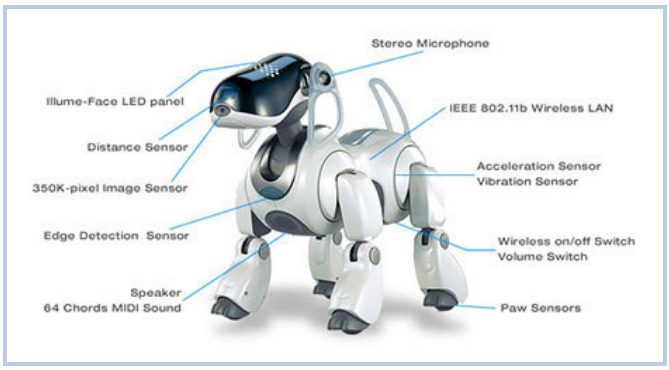
I robot predisposti a giocare a calcio riflettono con molta efficacia in maniera fisica le caratteristiche dei sistemi software in cui sono confinati gli agenti, tuttavia, il tipo di ambiente, il tipo di potenzialità e di funzionalità richieste e la natura degli agenti stessi variano considerevolmente in base al tipo di competizione, pertanto si riporteranno esempi classificandoli a seconda della League che i robot sono destinati a disputare.

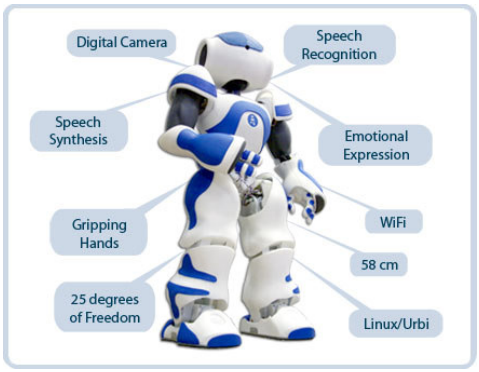
2.1.1. Robot per Standard Platform League

In letteratura, la *Standard Platform League* è forse quella che ha meno esempi pratici concreti e che forse suscita un po' meno interesse, poichè non è molto personalizzabile e dà meno spazio alla creatività.

L'hardware è infatti standard e prestabilito, ossia uguale per tutti; ogni team deve adeguarsi al tipo di robot concesso, non permette di dare sfogo quindi all'immaginazione ed agli aspetti più pratici che caratterizzano la creazione e la messa a punto di un robot.

Questa competizione ha sostituito la *Four-Legged League*, che si disputava con il robot canino *SIBO* della Sony, ora invece il robot di base che viene utilizzato è l'umanoide *Nao* della Aldebaran.





Anche se meno stimolante, a livello di sviluppo comunque è molto importante perché permette di testare gli agenti dal punto di vista puramente software, quindi poiché occorre concentrarsi soprattutto su questo aspetto, si può arrivare a concepire agenti anche abbastanza complessi.

Lo sviluppo è incentrato quindi sul tipo di agenti che faranno parte del sistema, sulla cooperazione che essi hanno tra loro e sugli algoritmi utilizzati, poiché nessuna squadra parte in vantaggio tecnologicamente rispetto alle altre, e pertanto la performance del team dipende esclusivamente dall'intelligenza e dal comportamento che contraddistingue il robot.

2.1.2. Robot per Small Size League

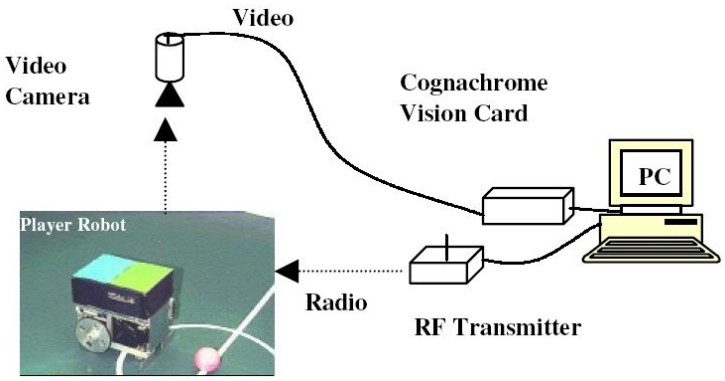
La *Small Robots League* è una piattaforma ideale per collaudare tecniche di intelligenza artificiale distribuita, tra cui ovviamente i sistemi multi-agente, che promuovono l'intelligenza artificiale collettiva incentrandosi sull'autonomia degli agenti e sulla loro intercomunicazione.

La caratteristica fondamentale della RoboCup Soccer è che l'ambiente in cui i robot si trovano è dinamico: l' unico elemento statico nel campo da gioco è il campo stesso; i robot (compagni o avversari), come la palla, possono muoversi e trovarsi in qualsiasi parte del campo per mezzo di un intenzionale posizionamento strategico, di un'azione mancata o di uno spostamento forzato. Questo ha portato a considerare più proficuo orientarsi su un comportamento di tipo reattivo, piuttosto che su un comportamento tradizionale basato su modelli di controllo top-down. Infatti i robot non devono sprecare una grande quantità di tempo e di risorse costruendo mappe e generando percorsi che poi risultano spesso inutili al momento di agire. Si suppone, invece, che i robot debbano reagire agli immediati cambiamenti all'interno dell'ambiente con un semplice meccanismo di risposta agli stimoli.

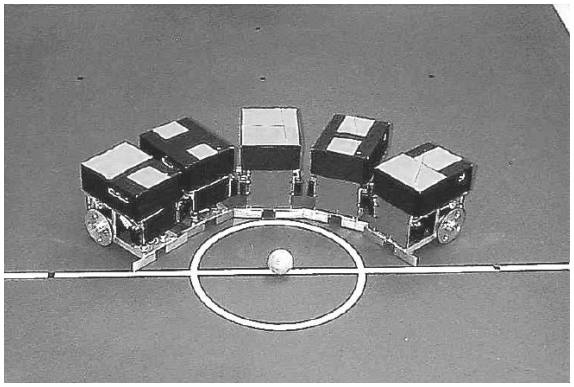
A causa dei vincoli sulla grandezza dei robot imposti da questo tipo di competizione, risulta molto complesso inserire dispositivi per la visione direttamente sui robot. Le telecamere, invece, sono piazzate sopra al campo, ottenendo così la visuale completa di tutta la configurazione di quest'ultimo (le coordinate dei robot, le identità e la posizione della palla) su un unico computer esterno.

Questo tipo di controllo centralizzato ha portato a considerare un approccio ibrido (deliberativo e reattivo).

Prendiamo in considerazione per esempio i robot del *Temasek POlytechnic* Team (*TPOT*), il comportamento reattivo è inserito direttamente nel robot per le azioni urgenti da eseguire in un lasso di tempo critico, come evitare un ostacolo ed eseguire un comando. L'elaborazione dei dati visivi ed il loro filtraggio, così come i ragionamenti di alto livello (per esempio la previsione della posizione della palla) sono eseguite nel computer remoto. Associato ad ogni robot vi è un agente interno incaricato di farlo muovere nel campo di gioco e fargli eseguire i comandi generati dall'agente remoto. Gli agenti remoti selezionano ed implementano i compiti richiesti, che sono basati sui dati visivi.

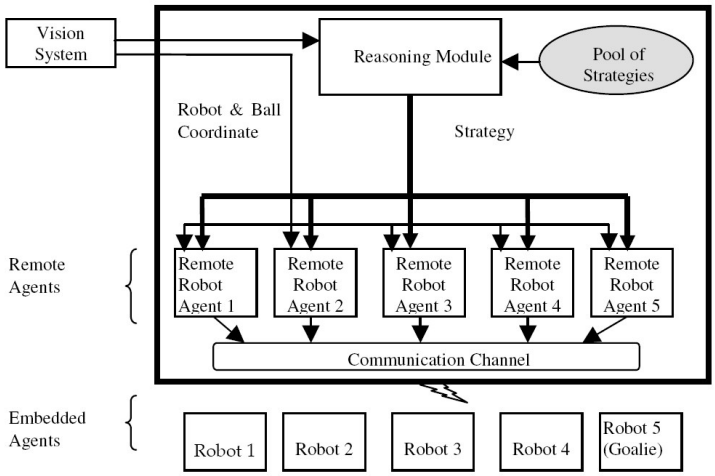


Per poter reagire agli stimoli, possiedono 3 sensori ad infrarossi montati nel fronte e nel retro in modo da rilevare gli ostacoli, mentre per ricevere le informazioni dal computer remoto viene utilizzata la tecnologia ad onde radio. Per essere rilevati e distinti adeguatamente dalla telecamera (posizione, identificazione ed orientamento) i robot possiedono 2 etichette colorate nella parte superiore (differenti per ognuno).

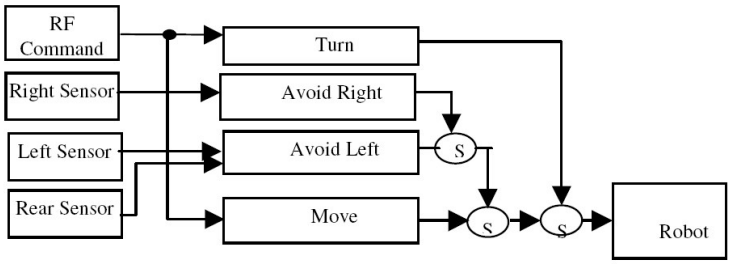


Come detto in precedenza, abbiamo 2 tipi di agente: uno integrato direttamente nel robot, quindi situato in campo, ed uno remoto, attivo nel computer fuoricampo in un modello di campo astratto. Il primo consiste in diversi comportamenti reattivi in competizione tra loro per mezzo di livelli di attivazione (inibizione e soppressione). Il ruolo principale di questo agente è eseguire i comandi del computer remoto e far girare il robot in maniera sicura nel campo, evitando gli altri robot o altri generi di ostacolo. Il secondo tipo, invece, implementa le strategie generate dal modulo di ragionamento; quest'ultimo, basandosi sulle prestazioni della squadra avversaria, le seleziona da un insieme di strategie prestabilite e le passa agli agenti remoti. Questo permette agli agenti di selezionare il compito appropriato per ogni robot (come intercettare la palla, seguire un obiettivo o muoversi in una determinata posizione).

Tali comportamenti sono implementati negli agenti remoti di ogni robot, ad eccezione del portiere: in questo modo si permette ai robot anche di cambiare ruolo dinamicamente (da difensore ad attaccante, e viceversa).



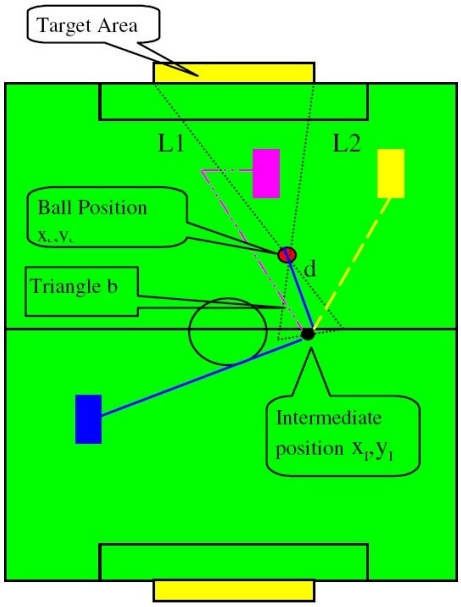
L'agente incorporato nel robot consiste in comportamenti reattivi ideati per realizzare dei task di basso livello per navigare all'interno del campo; questi comportamenti sono semplici moduli che rispondono agli stimoli (un input sensoriale o un comando ricevuto tramite RF). La risposta di un singolo comportamento di priorità più alta prende il controllo del robot ogni volta che lo stimolo a cui è associato si presenta. Evitare gli ostacoli è il principale compito autonomo eseguito dal robot: usando i 3 sensori ad infrarossi, il robot si sposta lontano dagli ostacoli usando i moduli *Avoid Left* e *Avoid Right*, esso poi si muove per una certa distanza fintanto che un percorso in linea retta fino alla posizione finale è libero; l'agente remoto rileva che il robot è fuori dal percorso calcolato precedentemente e lo rielabora, trasmettendo poi il nuovo percorso. I percorsi sono inviati dagli agenti remoti nella forma di angoli di rotazione seguiti da una distanza di viaggio.



Ad un robot che sappia giocare a calcio occorre avere alcune capacità basiche come muoversi verso la posizione della palla, calciarla verso l'area di rigore avversaria, intercettare la palla e passarla ad un membro della propria squadra.

Alcuni comportamenti stabiliti dagli agenti remoti sono quindi:

- *Intercept_ball*: per far muovere il robot dietro alla posizione prevista della palla prima di calciarla verso un'area desiderata, che potrebbe essere l'area di rigore avversaria, un'area libera di fronte ad un membro della propria squadra (passaggio) o semplicemente dalla parte opposta del campo, nel caso di un robot difensore. Questo comportamento può essere eseguito dal robot muovendosi prima di tutto in una posizione intermedia.



- *Follow*: per fare in maniera che il robot segua un oggetto target (la palla, un robot della squadra o un avversario), in modo tale che il robot sia il più vicino possibile alla palla e quindi in una migliore posizione per intercettarla.

- *Homing*: per fare in modo di mettersi in una certa posizione allo scopo di formare, per esempio, un muro difensivo.

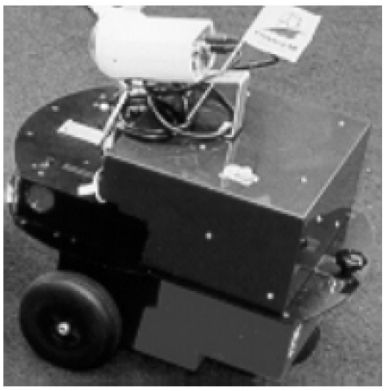
I percorsi vengono generati grazie a dei pianificatori e poi vengono trasmessi, tenendo traccia del robot. Nel caso venissero riscontrate divergenze causate ad esempio da una colluttazione con un robot avversario o da un movimento

intenzionale per evitare un ostacolo, il pianificatore di percorso elabora e trasmette un nuovo itinerario al robot.

2.1.3. Robot per Middle Size League

La *Middle Size League* è una delle competizioni più caratteristiche della RoboCup Soccer: dà ampio spazio alla progettazione sia a livello software sia sotto il profilo hardware. Inoltre una delle sfide più ardue è far avere al robot una cognizione globale della situazione di gioco e non sempre questo è possibile. Comunque la maniera più efficace per realizzare un agente/robot è forse data nel pensare ai comportamenti più semplici possibili.

Un esempio è il *T-Team Tuebingen* realizzato per una delle prime RoboCup (edizione '98), costruito di base con i robot *Pioneer1* e *Pioneer AT* in commercio: il primo provvisto di 3 ruote, usato per realizzare i calciatori, mentre il secondo di 4, usato per il portiere. Poichè la loro velocità di movimento non è sufficiente per imprimere alla palla una velocità adeguata per essere utile come passaggio o insidiosa come tiro, è stato necessario aggiungere un dispositivo "calciante" ad hoc.



Quest'ultimo è stato realizzato in due maniere differenti: la prima consiste in un dispositivo composto da un cilindro pneumatico ad aria compressa (il serbatoio di aria è sufficiente per circa 30 calci), la seconda prevede invece un meccanismo a molla controllato da un potente motore. Il dispositivo si aziona non appena la palla colpisce un micro pulsante situato nel fronte del robot.

Per sensori questi tipi di robot possiedono delle videocamere (per identificare gli oggetti), degli scanner laser (per misurare le distanze) ed un trasduttore ad

ultrasuoni (per evitare gli ostacoli e per aiutare il portiere a mantenere la posizione giusta).

Come detto precedentemente, questo team adotta un comportamento molto semplice, vincolato dal ruolo che ogni giocatore deve assumere in campo, in modo tale da evitare quei problemi che si potrebbero avere a causa di instabilità della comunicazione (diretta o indiretta) tra i robot.

Il ruolo è assegnato ad ogni giocatore staticamente:

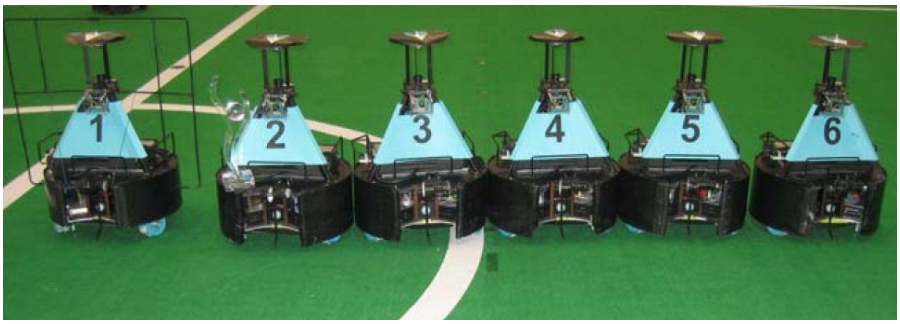
- *Portiere*: poiché possiede un limitato campo visivo, non ha la cognizione dell'intera area che gli sta attorno, pertanto si è cercato di posizionare la videocamera in modo che inquadri bene tutta la parte destra del campo. Il portiere poi tenterà quindi di rimanere sulla traiettoria tra la palla e la porta, mantenendo come posizione di default la parte sinistra di fronte alla porta quando non vede la palla.

- *Difensore*: i 2 difensori rimangono uno nella parte destra e l'altro nella parte sinistra del campo, di fronte alla propria porta. Per evitare che si disturbino l'uno con l'altro, ognuno sarà responsabile della propria metà del campo. Se la palla è più lontana di 1,50 m, l'unico compito del difensore è inseguirla, altrimenti deve muoversi verso di essa e calciarla nella metà campo avversaria. Dopodiché dovrebbe ritornare alla sua posizione.

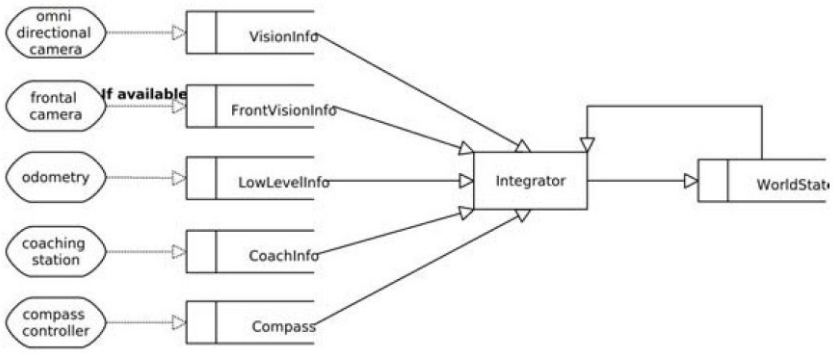
- *Attaccante*: il compito degli attaccanti è spingere o calciare la palla nella porta avversaria. Quando un attaccante rileva la palla, deve andare verso di essa e girarsi con la palla in direzione della porta. Dato che agli attaccanti non viene assegnata una specifica area, per non disturbarsi l'uno con l'altro, solo quello più vicino alla palla gli va incontro, l'altro ne tiene traccia e poi non si muove oltre. Se l'attaccante che ha la palla rileva la porta avversaria, attiva il dispositivo calciante. La volta successiva che la palla toccherà il micro pulsante di fronte al robot, esso la calcerà istantaneamente senza perdere tempo nella direzione della porta avversaria.

Un aspetto di rilevanza più complesso in questa competizione è il rilevamento e riconoscimento di ostacoli: quando si costruisce una rappresentazione dell'ambiente che ha un robot, in un'applicazione multi-agente la fusione della percezione

sensoriale e delle informazioni di diversi elementi dell'ambiente è un compito molto importante. Per costruire in maniera incrementale un modello del mondo sempre più affidabile e completo, uno degli aspetti che bisogna considerare è la trattazione degli ostacoli. Per questa caratteristica prenderemo in considerazione il team *CAMBADA* (*Cooperative Autonomous Mobile roBots with Advanced Distributed Architecture*) dell'Università di Aveiro, che ha partecipato alla RoboCup del 2008. CAMBADA è un robot autonomo che ha delle proprie percezioni sensoriali; può comunicare direttamente con gli altri robot o con un computer esterno (che per regolamento può fungere solo da "coach", ricevendo esclusivamente le informazioni rilevate dai robot in campo, ma che non può avere propri sensori).

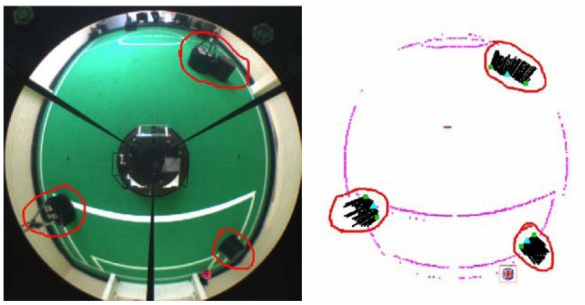


I robot, durante la partita, devono riposizionarsi o avanzare con la palla evitando gli ostacoli in campo, che potrebbero essere sia robot avversari o della propria squadra, sia eventualmente l'arbitro. Una necessità per aumentare le performance è il bisogno di un miglior rilevamento e condivisione delle informazioni sugli ostacoli tra compagni di squadra: questo è importante per avere un'idea globale dell'occupazione del campo. Con una buona copertura degli ostacoli in campo, si possono trovare molto più facilmente traiettorie per passare e corridoi per scartare, permettendo di aumentare la strategia ed il coordinamento della squadra. Quindi un lavoro essenziale è quello di fondere le informazioni sensoriali prese da ognuno dei giocatori; viene condotta un'analisi dell'immagine catturata dal sensore visivo, inoltre possono venire elaborate altre informazioni da altri tipi di sensori ed infine viene eseguita una fusione di tutte queste sorgenti. Tutte le informazioni disponibili dai sensori nel ciclo corrente sono contenute in specifiche strutture dati, che serviranno per eseguire successivamente la fusione e l'integrazione, basate sia sulle informazioni dello stato del mondo corrente, sia su quello precedente.

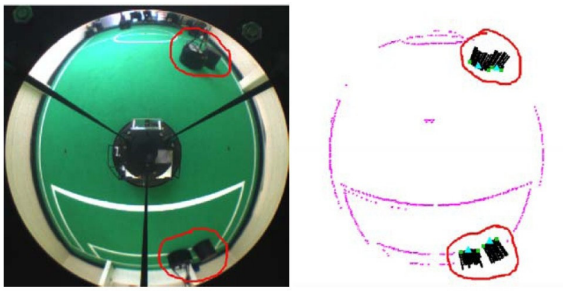


La prima fase per evitare gli ostacoli è il loro rilevamento. I robot raccolgono le informazioni sull'ambiente circostante per mezzo di un sistema di visione robotico: solo la videocamera omnidirezionale raccoglie le informazioni sugli ostacoli, mentre la videocamera frontale per il momento non è usata.

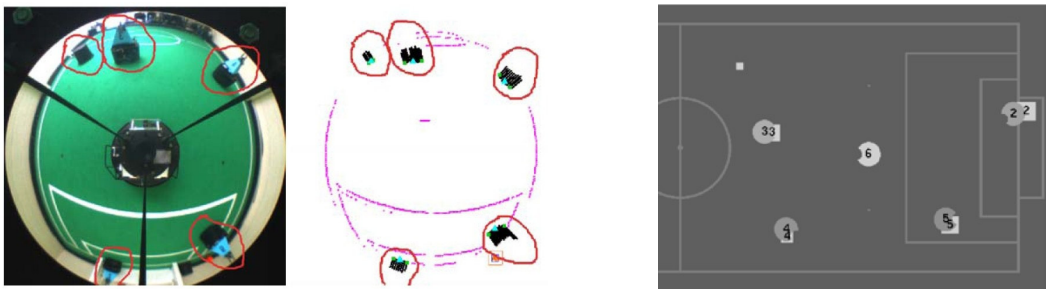
Come stabilito dalle regole ufficiali della RoboCup, i robot devono essere principalmente neri, questo può essere sfruttato a proprio vantaggio facendo in modo che l'algoritmo di visione possa rilevare gli ostacoli associandoli alle macchie nere dentro al campo di gioco; attraverso la mappatura delle posizioni all'interno dell'immagine alle reali posizioni metriche, gli ostacoli sono identificati tramite il loro centro e gli estremi destro e sinistro (con una tecnica di scanlines). Inoltre, poiché la dimensione degli oggetti di un'immagine varia in base alla distanza di questi ultimi dal robot, è possibile ricavare una relazione tra le distanze dell'immagine e quelle del mondo reale tramite algoritmi metrici.



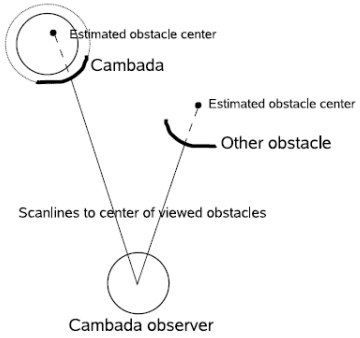
A volte, inoltre, è possibile che un ostacolo venga visto come più macchie separate, principalmente a causa di rumore nel rilevamento o di errori di classificazione di colore: per risolvere questo problema viene considerato un offset minimo nello spazio per separare i vari pezzi rilevati. Per mantenere una precisione coerente, questo offset aumenta con la vicinanza dal robot.



La fase successiva consiste nella selezione ed identificazione degli ostacoli per tentare di capire se essi siano degli avversari o dei compagni di squadra. Poiché a lunghe distanze una identificazione è inaffidabile, essa viene eseguita non a più di una certa vicinanza stabilita; se gli ostacoli rilevati sono fuori dal campo o più piccoli di 10 cm vengono ignorati (non vengono ritenuti dei robot).

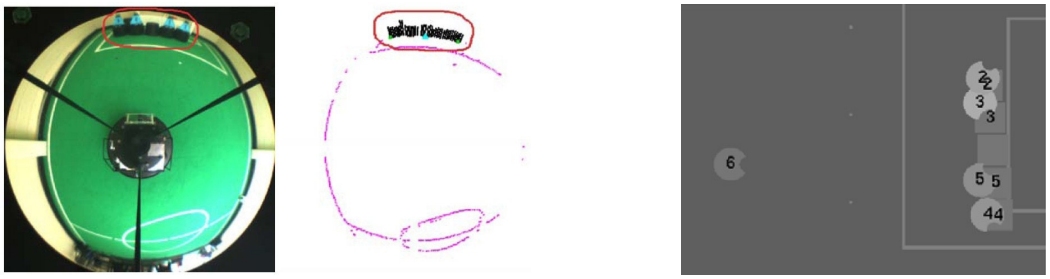


Successivamente, grazie alla fusione e condivisione di informazioni da parte degli altri robot del team (soprattutto per quanto riguarda la loro posizione), e dopo aver eseguito alcuni calcoli metrici, ogni robot può presupporre la squadra di appartenenza degli ostacoli presenti.



In contrapposizione all'effetto visto in precedenza (rilevamento di più macchie per un unico ostacolo), è possibile che si verifichi invece che il robot si imbatta in una macchia troppo grande per essere un solo robot: in questo caso viene eseguita

un'analisi più approfondita e precisa per identificare quanti robot in realtà siano contenuti in quell'agglomerato di robot.



L'ultima operazione che continuamente ogni robot esegue è condividere le proprie informazioni sugli ostacoli rilevati con gli altri robot: infatti è possibile che vengano anche presi in considerazione falsi ostacoli (soprattutto per le condizioni di illuminazione, zone d'ombra o di materiali riflettenti, che possono trarre in inganno il robot). Quindi dopo essersi costruiti una propria idea del mondo circostante, ogni robot interroga uno ad uno gli altri robot e viene eseguito un confronto tra gli ostacoli rilevati individualmente da ognuno. Se un oggetto non è presente nella lista dell'altro robot, viene messo in una lista temporanea; se esso non è presente in nessun altro compagno di squadra, allora viene dimenticato, altrimenti viene confermato. Viceversa, lo stesso vale per gli oggetti non presenti nella propria lista che invece hanno altri robot: in questo caso, se il nuovo oggetto, di cui si ignorava l'esistenza, viene confermato, verrà aggiunto alla lista.

2.1.4. Robot (agenti software) per Simulation League

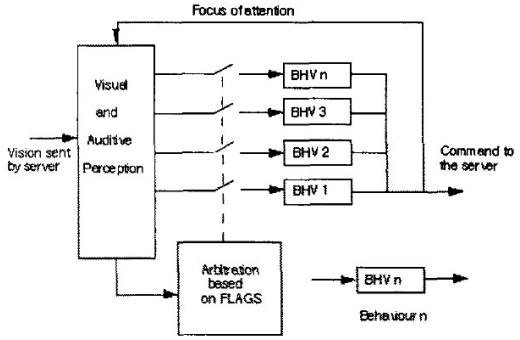
Nella *Simulation League* occorre implementare agenti puramente software, quindi vengono coinvolti esclusivamente l'intelligenza artificiale ed i sistemi multi-agente dal punto di vista informatico, senza avere i problemi e gli imprevisti dovuti al mondo reale.

Infatti un aspetto che la Simulation League consente di sviluppare maggiormente è quello tattico, e questo grazie al fatto che l'ambiente è leggermente più deterministico rispetto alle altre competizioni (le percezioni e gli spostamenti dei robot, le traiettorie della palla sono frutto di calcoli esatti al computer, non intervengono fattori esterni o casuali); nelle competizioni reali, l'hardware e lo stesso ambiente di gioco sono causa di imprecisioni ed imprevedibilità dovute alla

tecnologia (inaffidabilità dei sensori/attuatori, guasti, ecc..) ed a fattori fisici (attrito con il terreno, effetto della palla, spostamenti d'aria, ecc..) non prevedibili. Considerando ciò, ci si può dedicare più facilmente ad aspetti più complessi, mentre in ambienti reali occorre attuare meccanismi più semplici per tenere conto di eventuali e frequenti errori o di elaborazioni di tragitti e spostamenti calcolati in base a posizioni e percezioni sensoriali che in realtà sono sbagliate.

Come mostrato anche in alcuni esempi precedenti, la forma più elementare di comportamento di un robot è quella reattiva; l'esperienza porta a considerare che in un gioco di calcio sia comunque più efficace utilizzare una fase reattiva prima di prendere in esame una fase razionale e pianificata. Per reattivo si intende un comportamento in cui l'agente reagisce immediatamente ed individualmente agli stimoli ed al cambiamento del mondo esterno, comportandosi di conseguenza. Questo non è molto indicato in una situazione in cui un robot debba muoversi in un ambiente fisico ben strutturato e sconosciuto a priori; in questo caso dovrebbe prevedere una fase di pianificazione perché non ha modo di sapere nell'immediato quale azione sia più opportuno eseguire per arrivare al goal prefissato. In un gioco del calcio simulato, invece, in ogni momento del gioco i giocatori sanno come eseguire e portare a termine il loro compito, usando solo le informazioni percettive. Ad ogni istante di gioco, qualsiasi giocatore può facilmente segnare un goal come obiettivo individuale, anche se in realtà dietro vi è uno scopo globale. Questo permette ad ogni singolo agente di provare a segnare direttamente un goal ogni volta gli sia possibile.

Il meccanismo di base è dominato dal concetto di arbitrio che influisce sia a livello spaziale (arbitrio statico; attivazione di un set di primitivi comportamenti riflessivi) che temporale (arbitrio dinamico; decide la sequenza di esecuzione del set dei comportamenti).



Ciascun giocatore, in ogni momento della partita, sarà quindi abilitato con un comportamento che dipende sia dalla sua posizione corrente e dal suo orientamento nel campo di calcio sia dai pattern e dagli oggetti di cui l'agente è a conoscenza. Il mondo circostante sarà la causa delle differenze tra gli agenti, poiché essi hanno un'identica struttura, ma non avranno attivi gli stessi comportamenti.

Un approccio a livello di coordinazione prevede due tipi di cooperazione tra i robot: comunicazione implicita, che avviene indirettamente osservando il comportamento degli altri agenti, ed esplicita, in cui ogni agente volontariamente avvisa gli altri agenti tramite messaggi; si potrebbe usare un mix dei due.

I comportamenti della struttura degli agenti riproducono le principali capacità di basso livello dei reali giocatori di calcio. Nel gioco del calcio sono importanti in primo luogo le abilità individuali del singolo, prima di poter provare una qualsiasi strategia di squadra. Inoltre è un giocatore singolo che porta a termine il sottoobiettivo (se consideriamo come obiettivo finale vincere la partita) di segnare un goal, quindi sebbene sia sbagliato considerare il calcio come gioco individuale, l'acquisizione delle capacità individuali diventa il compito primario.

Il comportamento di base, quindi, sarebbe permettere ai giocatori di imparare abilità di basso livello, come tirare in porta, intercettare la palla, usando le reti neurali, ma al contrario è possibile utilizzare anche un approccio di tipo analitico.

Per quanto riguarda la consapevolezza dello stato globale del sistema, dopo ogni messaggio visivo o sonoro ricevuto, ogni agente, tramite una funzione, si costruisce una struttura dati che rappresenti il mondo così come l'ha percepito l'ultima volta (posizioni, velocità, orientamento di ogni oggetto presente). Occorre che si abbia una struttura dati individuale perché ogni agente ha il proprio punto di vista.

Tramite questa struttura dati ogni agente è in grado di estrarre i flag (che indicano lo stato corrente) usati per l'arbitrio: per esempio è in grado di stabilire se la squadra è in posizione di attacco o di difesa. Un flag potrebbe essere *kickable*, che viene settato quando un compagno di squadra entra in possesso della palla; successivamente lo stesso giocatore manda un comando uditivo a tutti i giocatori che possono ascoltarlo.

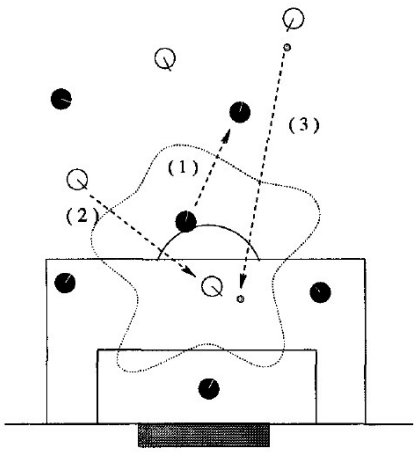
Nella modalità di attacco esistono due tipi di ruoli: chi è in possesso della palla (comportamento *playball*) e chi potenzialmente può ricevere la palla (comportamento *smarcamento*).

In fase difensiva, invece, principalmente le azioni sono contraddistinte da fare pressing (*chasing the ball*) ed ostacolare (*interdict*).

Per realizzare questo tipo di interazione l'agente sceglie un comportamento tra quelli candidati considerando i flag globali (che rappresentano particolari stati dell'intero team) e quelli locali (riferiti al singolo agente).

Il comportamento cooperativo che emerge deriva in realtà da un comportamento determinato dai singoli individui: ogni sorta di azione intelligente come triangolazioni o passaggi non espliciti derivano principalmente dall'interazione tra i singoli giocatori e l'ambiente (nel nostro caso rappresentato dagli avversari).

Ad esempio, quando un giocatore della propria squadra ha la palla (*playball*), presumibilmente l'avversario più vicino in assetto da difesa gli andrà incontro per rubargliela, e si creerà dietro a quest'ultimo un'area vuota, dove un compagno di squadra in stato di *smarcamento*, accorgendosi del buco apertosi, si inserirà; a questo punto, il possessore della palla, vedendo il proprio compagno in posizione ottimale, potrà passargliela. In questo modo si può quindi realizzare in maniera efficace un passaggio filtrante, senza alcun tipo di comunicazione diretta tra i due agenti.



In fase difensiva, invece, un possibile tipo di coordinazione è la negoziazione, per selezionare chi debba andare a fare pressing sul portatore di palla, in modo tale da non lasciare totalmente sguarnita la linea difensiva.

Inoltre, un altro meccanismo di coordinazione consiste nel cambiare dinamicamente la posizione di ogni singolo giocatore della squadra, in base alla posizione della palla, in modo tale da seguire la palla in caso di attacco, o, in caso contrario, tornare indietro gradualmente in posizione di difesa.

Tra gli aspetti più complessi invece possiamo prendere in esame per esempio la formazione con cui una squadra scende in campo (nelle competizioni con robot reali sarebbe poi difficile mantenerla durante il gioco); a tale scopo prendiamo in considerazione un altro modello di squadra, diverso dal precedente.

Innanzitutto occorre fare delle distinzioni delle caratteristiche tra i robot virtuali che fanno parte della squadra: per ognuno di essi occorre designare delle capacità che siano adatte al tipo di ruolo in campo.

Avremo quindi:

- *Forward-type (F-type):* robot che si muovono più veloci e che calciano molto più forte degli altri. Tuttavia essi hanno un campo sensoriale molto ristretto.
- *Defender-type (D-type)*: robot che si muovono più lentamente e che calciano con meno potenza la palla. Tuttavia essi hanno un raggio sensoriale che arriva a lunga distanza.
- *Mid-fielder-type (M-type)*: robot in cui tutte le caratteristiche principali sono medie rispetto agli altri 2 tipi di robot.

Ogni robot, in base alle proprie caratteristiche, esibisce un comportamento appropriato. Tra i comportamenti di tipo offensivo abbiamo per esempio:

- Passare a giocatori che sono in posizione ottimale per tirare.
- Fare movimento per ricevere un passaggio.

Nel primo caso chi effettua un passaggio è ovviamente sempre il possessore della palla, ma il ricevente non è mai fissato durante il gioco. Un passaggio è comunque

considerato come un semplice scambio della palla tra 2 giocatori; risulta complicato ideare tecniche efficaci di triangolazione o sovrapposizione.

Se un giocatore ha la palla può decidere se avanzare con essa o passarla, e questo è influenzato soprattutto dalle capacità individuali del robot e dagli avversari che avverte entro la sua area sensoriale.

Nel caso decida di passarla può scegliere se:

- Provare a passarla al giocatore che è più vicino alla porta avversaria (e quindi è in posizione ottimale).
- Nel caso in cui le distanze siano le stesse, provare a passarla al robot che è più bravo a tirare (F-type).

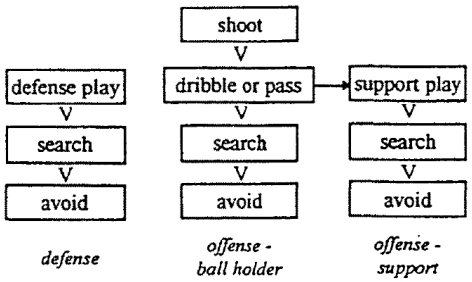
Se il giocatore invece non ha la palla, può semplicemente muoversi fino ad una certa distanza dalla porta avversaria.

Tra i comportamenti di tipo difensivo, invece, si ha:

- Ostruire un tiro posizionandosi nella traiettoria tra l'avversario e la propria porta.
- Ostruire un passaggio posizionandosi nella traiettoria tra l'avversario con la palla ed un suo compagno di squadra.

Un robot in fase difensiva, quando non ha la palla, tende semplicemente a rimanere ad una certa distanza dalla propria porta e poi a cercare un avversario da disturbare.

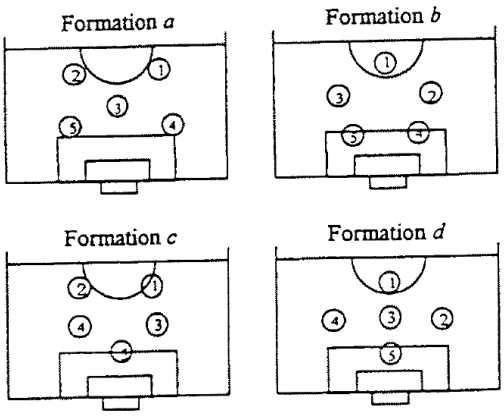
La scelta del comportamento è comunque molto elementare.



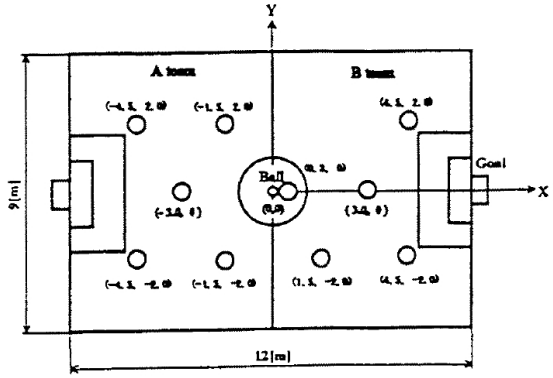
Dopo aver fatto tutte queste considerazioni sulle caratteristiche offensive e difensive, possiamo valutare il tipo più idoneo di formazione da adottare.

Mettiamo a confronto 4 moduli (come caratteristiche individuali di riferimento si considerano i robot di tipo D):

- *A*: 2-1-2
- *B*: 2-2-1
- *C*: 1-2-2
- *D*: 1-3-1



La situazione più critica è il calcio d'inizio, infatti la squadra che lo batte in genere poi perde. Questo è dovuto al fatto che il giocatore incaricato di calciarlo è anche quello che in tale momento è il più vicino alla porta avversaria, pertanto non passerà la palla a nessun altro compagno di squadra (nel raggiungimento dell'obiettivo di segnare, è contro natura effettuare un passaggio indietro), questo lo porterà prima o poi a perderla.



I robot di tipo D tendono più che altro a passare la palla, anziché scartare. Se i robot fossero di tipo F, il possessore della palla probabilmente proverebbe a scattare spesso verso la porta e la palla gli sarebbe frequentemente rubata dagli avversari: in generale quindi con un robot di tipo D si ha un comportamento più cooperativo.

Facendo giocare 2 squadre e utilizzando tutte le combinazioni possibili derivanti dai 4 moduli, risulta che con la formazione *C* è più facile che la squadra perda se deve battere il calcio d'inizio, nel caso contrario, invece, grazie al tipo di schieramento che porta a circondare l'avversario, si riesce quasi sempre a riconquistare la palla: questo significa che la formazione *C* è ideale per difendere. La formazione *D*, invece, raggiunge la miglior prestazione in entrambi i casi, questo perché la linea centrale è quella più forte e difficile da superare, adatta quindi per difendere. Questa attualmente sembrerebbe quindi la formazione migliore.

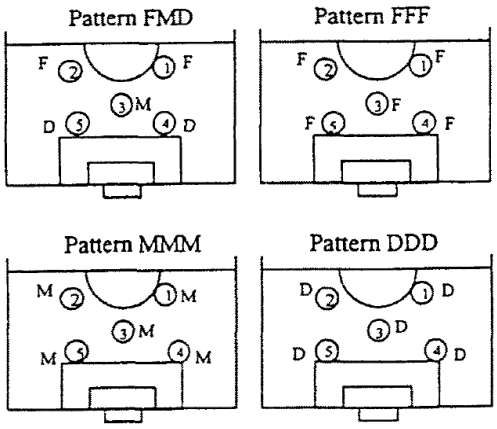
	team A		team B	(kick off)	time	winner
	Formation	total points	Formation	total points	(sec)	
(1)	b	107	a	54	703	B
(2)	c	91	a	19	504	A
(3)	d	77	a	91	358	A
(4)	a	207	b	57	1238	A
(5)	c	130	b	176	1401	A
(6)	d	69	b	16	264	A
(7)	a	128	c	52	821	A
(8)	b	151	c	20	631	A
(9)	d	85	c	36	547	A
(10)	a	334	d	316	2453	B
(11)	b	35	d	130	557	B
(12)	c	19	d	83	324	A

Nel dettaglio, si può dire che con la formazione *D*, se si batte il calcio d'inizio, tutti i giocatori contribuiscono positivamente a favore della squadra (ossia acquistano molti punti di valutazione, utili per indicarci la performance ottenuta), mentre nel caso opposto solo alcuni robot cooperano favorevolmente; questo dipende dalle posizioni relative dei robot, e talvolta può accadere che 2 giocatori si passino la palla inutilmente.

Ora individuiamo il pattern ideale per la combinazione delle caratteristiche dei robot (prendendo di base la classica formazione *A*):

- D-M-F
- F-F-F

- M-M-M
- D-D-D



Nel caso in cui nella formazione siano presenti tutti i tipi di robot (*FMD*) e nella squadra avversaria la linea più avanzata non sia composta da robot difensivi, accade che quasi sempre il robot, subito dopo aver battuto il calcio di inizio o dopo aver conquistato palla (in caso l'avesse battuto l'avversario) tende sempre a scartare per poi tirare, mostrando un comportamento altamente individuale. Prendendo come riferimento la formazione *A*, che non è molto robusta difensivamente perché ha il centro molto scoperto e debole, solo le combinazioni con robot dello stesso tipo (*FFF* o *DDD*), riescono a fare in modo che gli altri robot laterali possano lo stesso coprire bene uniformemente l'area centrale grazie alla percezione sensoriale che è uguale per tutti.

	team A		team B (kick off)		time (sec)	winner
	pattern	total points	pattern	total points		
(1)	FMD	0	FMD	90	111	B
(2)	FFF	105	FMD	27	155	A
(3)	MMM	0	FMD	90	111	B
(4)	DDD	128	FMD	30	383	A
(5)	FMD	0	FFF	90	111	B
(6)	FMD	134	MMM	10	203	A
(7)	FMD	12	DDD	89	503	B

Al contrario di quanto si possa immaginare, quindi, le combinazioni migliori sono quelle omogenee, cioè *FFF* e *DDD*, forse anche perché il numero dei robot non è sufficiente per adottare una squadra con caratteristiche miste.

2.1.5. Robot per Humanoid League

Per poter partecipare alla *Humanoid League* i robot devono avere un corpo simile a quello umano: un tronco, 2 braccia, 2 gambe ed una testa, inoltre il solo modo per spostarsi deve essere per mezzo di una camminata a 2 piedi, ma soprattutto i robot devono essere completamente autonomi (non sono permessi sorgenti di energia esterna o controllo remoto).

Esistono tipi particolari di competizioni come quella ai rigori (con robot di grandi dimensioni) ed il *Freestyle* (ogni anno c'è una sfida diversa, ad esempio un percorso da eseguire, che un robot deve superare entro 5 minuti sotto la visione di una giuria), ma rimanendo sul tradizionale, prenderemo in considerazione la sfida a calcio che si svolge a squadre di 3 elementi.

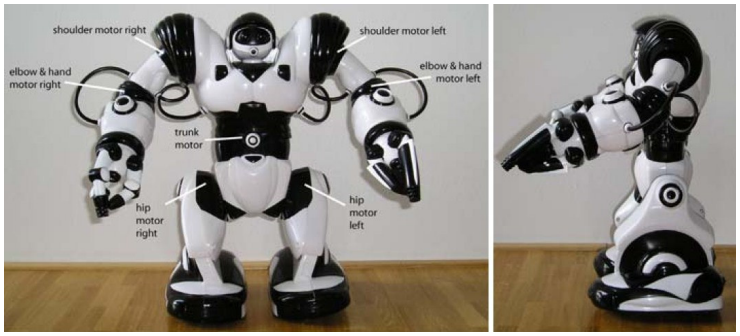
Nel 2004 la sfida a calcio è stata vinta dal robot *VisiON* del Team Osaka: come un reale portiere, esso poteva buttarsi a terra per ostacolare un tiro, per poi rialzarsi da solo senza problemi.

Il problema più evidente per questi tipi di robot rimane comunque il bilanciamento: è difficile infatti fare in modo che essi camminino e calcino senza perdere l'equilibrio.

A causa della limitata disponibilità di robot umanoidi e gli alti costi coinvolti, gli esperimenti multi-agente con questi robot si sono resi molto difficili da realizzare. Essi sono però molto utili perché bisogna considerare che i robot umanoidi non sono utilizzati solo a scopo ludico (nel nostro caso per giocare a calcio): un robot con un corpo umano porta numerosi vantaggi in ogni ambiente adatto agli umani, dove essi in determinate circostanze, devono sostituirsi all'uomo stesso.

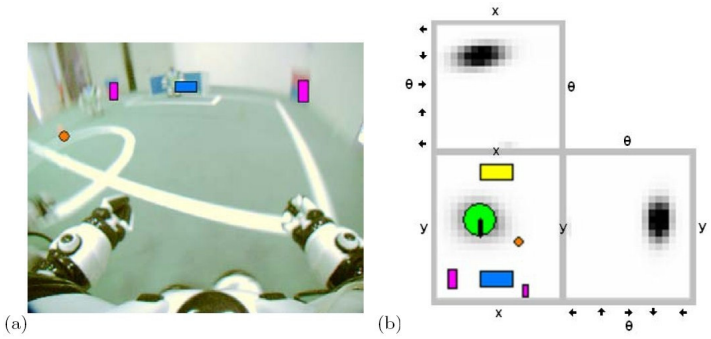
Una soluzione potrebbe essere utilizzare un robot già in commercio ed adattarlo per la RoboCup, senza dovere costruirsene uno ad hoc.

Consideriamo per esempio il *RoboSapien*, un robot umanoide in commercio a costo relativamente basso; questo tipo di robot inoltre è molto stabile poiché ha il centro di massa basso. Possiede 7 motori (3 usati per muoversi), e braccia con 3 dita prensili l'una; il RoboSapien originale viene mosso tramite controllo remoto (è possibile eseguire 67 tipi diversi di movimento).



Per renderlo autonomo, e quindi regolamentare per la Humanoid League, gli si è rimossa la testa per sostituirla con un Pocket PC (che lo controlla) ed una micro camera (con lenti modificate per avere un raggio visivo più ampio).

Siccome nella RoboCup Soccer gli oggetti chiave, come la palla e le porte, sono contraddistinti da particolari colori, è possibile sfruttare tali caratteristiche informative a proprio vantaggio. Infatti l'unica cognizione del mondo esterno che il robot ha, deriva dall'immagine catturata dalla videocamera, ed in essa vengono individuati gli oggetti chiave caratterizzati da determinati colori prestabiliti. Vengono stimate le coordinate dei vari oggetti in un frame egocentrico (distanza ed angolo di orientamento), in base alla loro posizione ed alla loro grandezza; con questo il robot è in grado per esempio di prendere consapevolezza della posizione rispetto alla palla. Per implementare invece i comportamenti globali del team come il calcio d'inizio, occorrono le coordinate in un frame allocentrico (posizione sul campo ed orientamento).



Nel caso la palla o la porta non fossero nella visuale del robot, si usano metodi di localizzazione per elaborarne le coordinate. Tali metodi di localizzazione cercano inoltre di integrare le visuali locali dei vari robot, per stimare una visuale globale della squadra.

Per quanto riguarda il comportamento, occorre focalizzarsi sulle questioni di alto livello, come tenere la palla, posizionarsi in campo e giocare in squadra.

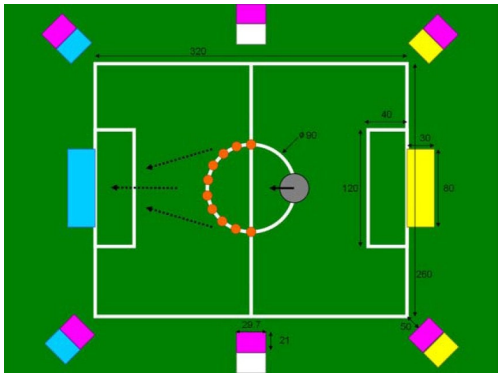
Il robot può eseguire diverse azioni fra quelle presenti in un set di funzioni di movimento parametrizzate, tra cui camminare dritto per una certa distanza o ruotare di un certo angolo. Un'altra possibilità di movimento potrebbe essere iniziarlo e continuare ad eseguirlo fintanto che non cambia un certo stato desiderato, riportato dal software di visione.

Si è cercato di far muovere un robot in modo che gli oggetti chiave (palla, porte e punti di riferimento) rimanessero nel suo campo visivo, ma questo non è sempre possibile: infatti se il robot vuole girare attorno alla palla, prima deve passarla. Il robot può solo girarsi verso la palla ancora quando essa è dietro al robot, fuori dal suo campo visivo. Un altro esempio è quando il robot si trova davanti alla porta e vuole muoversi lateralmente alla palla per allinearla con la porta e direzionare quindi il tiro: in questo caso il robot deve girarsi (perdendo la vista della palla), camminare verso la traiettoria palla-porta e voltarsi indietro (per questa situazione è stato studiata una macro che prevede un concatenamento di 4 movimenti).

La velocità e la qualità delle decisioni dipendono comunque dal frame rate della videocamera e dalla velocità con cui il RoboSapien accetta nuovi comandi.

Un altro comportamento molto elementare è che se i robot non vedono la palla, iniziano a vagare per il campo per cercala. Inoltre, possono avvicinare la palla abbastanza da essere di fronte alla porta; possono portare la palla verso la porta e se la palla è tra un robot e la propria porta possono muoversi attorno alla palla.

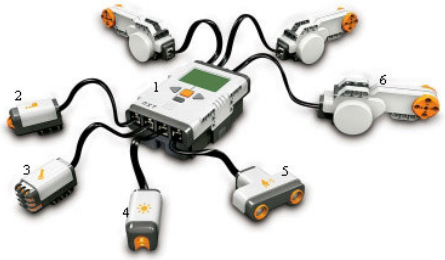
Affinchè questi comportamenti siano realizzabili ci si deve però affidare a dei riferimenti che facciano da supporto al riconoscimento. Innanzitutto quelli essenziali sono la palla (arancione), il campo (viene usato un tappeto verde, delimitato da linee bianche) e le porte. Sono stati poi aggiunti attorno al campo dei punti di riferimento di diverso colore per aiutare nella localizzazione.



Questo tipo di robot è stato impiegato a titolo dimostrativo al RoboCup German Open a Paderborn nel 2005. E' stato poi utilizzato dal team *Hiro* alla RoboCup dello stesso anno nella Humanoid League.

3. Robot LEGO Mindstorms

Le caratteristiche dei robot *LEGO Mindstorms* fanno in modo che essi si prestino favorevolmente alla sperimentazione. Grazie al loro costo contenuto, alla possibilità di utilizzare vari linguaggi di programmazione, all'opportunità di personalizzare il robot nella forma (e crearne di diversi tipi) a partire dagli stessi pezzi ed alla disponibilità di avere sensori ed attuatori in grado di effettuare la maggior parte dei compiti attribuibili ad un robot (esplorazione, ricerca, trasporto, riconoscimento di oggetti, ecc…), sono prediletti per testare concretamente nel mondo reale modelli sviluppati solo in un ambiente virtuale.



In particolare, per avere conoscenza del mondo esterno, i Mindstorms possono disporre dei sensori di:

- *Tatto.* Rileva la pressione di un tasto.
- *Distanza.* Rileva i movimenti e la distanza da oggetti grazie alla tecnologia ad ultrasuoni.
- *Suono.* Rileva l'intensità di rumore presente nell'ambiente.
- *Luce.* Rileva l'intensità luminosa presente nell'ambiente, riconoscendo gradazioni di grigio.
- *Colore.* Distingue il colore di un oggetto.

Ed in risposta ad essi, per eseguire azioni, si possono usare come attuatori:

- *Servomotori.* Regolabili in velocità, distanza e gradi per far muovere il robot.
- *LED.* Un LED rosso è presente nel sensore di luce. Viene utilizzato soprattutto per farne assimilare meglio l'intensità luminosa, ma può essere anche utilizzato individualmente come output segnaletico.

Il tutto è gestito da un piccolo computer principale denominato *Brick* a cui vengono collegati i sensori (massimo 4) e gli attuatori (massimo 3); la comunicazione con altri Brick (ossia con altri robot) avviene tramite tecnologia bluetooth.

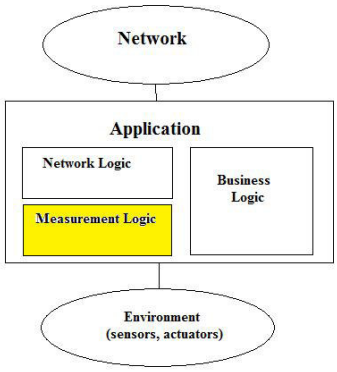
Questi tipi di sensori/attuatori e di caratteristiche a cui si fa riferimento appartengono alla linea di prodotti *LEGO Mindstorms NXT*, evoluzione dell'*RCX*.

La tecnologia di tali robot però non sempre offre affidabilità e precisione; i risultati dei testing dipendono anche dalla complessità dell'applicazione.

Il linguaggio nativo per i LEGO Mindstorms NXT è l'*NXC* (di tipo C-like), tuttavia supporta numerosi altri linguaggi.

3.1. Architettura logica dei sistemi pervasivi

La struttura concettuale di base di un sistema pervasivo (ossia distribuito attorno a noi e che utilizza l'ambiente come mezzo di interazione) prevede all'interno dell'applicazione, una *Network Logic*, che si occupa della logica di comunicazione di rete, una *Business Logic* che invece svolge le funzionalità caratteristiche, ed una *Measurement Logic* che permette la comunicazione tra sistema ed ambiente.



Il sistema deve essere flessibile, scalare e robusto, pertanto centralizzare il tutto in un'unica applicazione non è consigliabile (specialmente a fronte di un numero di entità elevato ed a priori non ben definito da gestire), quindi si lasceranno queste parti ben distinte tra loro.

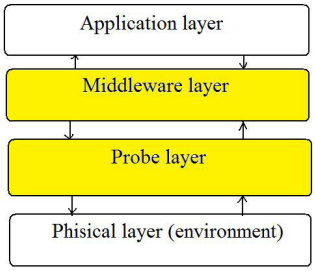
La *Measurement Logic*, dato che ha funzionalità molto comuni, è probabile che sia la stessa anche per applicazioni che invece hanno diversa *Business Logic* poiché quest'ultima è la parte più specifica, pertanto si possono implementare le due parti separatamente. La *sonda* è l'applicazione che ha lo scopo di interagire con l'ambiente e trasmettere le rilevazioni; lo sviluppo di questo componente è completamente indipendente da quello delle applicazioni che lo utilizzano.

Prendendo in considerazione un modello layer-oriented, la *Measurement Logic* verrà inserita nel livello intermedio *Probe Layer*, che si occuperà solo dell'elaborazione delle rilevazioni e delle trasmissioni, senza però interpretarle.

Il *Probe Layer* si occupa quindi delle funzionalità di basso livello e deve essere in grado di astrarre a livello software dallo specifico tipo di hardware (sensori ed attuatori) che deve essere gestito.

In questo modo si raggiungono gli obiettivi di interoperabilità, scalabilità e sviluppo indipendente, inglobando però anche i problemi dovuti alla coordinazione, inoltre avendo una struttura distribuita, avremo anche problemi dovuti alla comunicazione delle varie parti.

La *Business Logic*, invece, rimane a livello applicativo ed è composta da entità software di alto livello, cioè dagli agenti. Questi ultimi hanno il compito di mettere in atto le specifiche politiche per la gestione del sistema e devono essere in grado di interagire con il *Probe Layer* per entrare in contatto con l'ambiente fisico. Per quest'ultimo scopo viene aggiunto il *Middleware Layer* che fa tramite e si occupa di fornire una comunicazione orizzontale tra le diverse applicazioni, interoperabilità tra livello applicativo e sonde, trasparenza alla distribuzione e sviluppo indipendente.



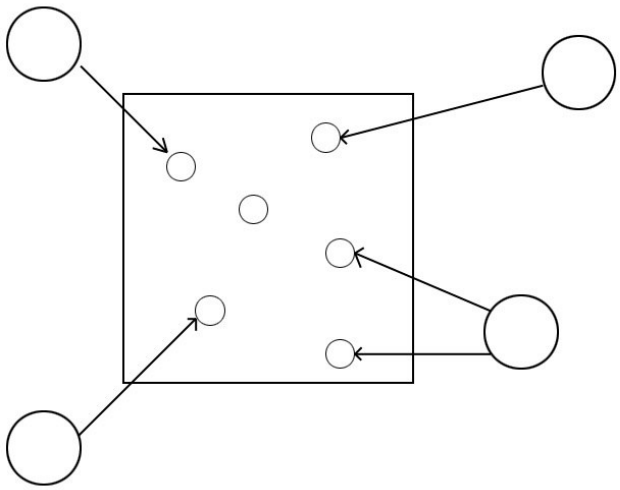
3.2. Modelli di coordinazione space-based

I sistemi distribuiti sono composti da entità attive che interagiscono tra loro in maniera concorrente. L'interazione e la cooperazione di queste entità (*coordinables*) avviene tramite un mezzo di coordinamento, detto *medium di coordinazione*, mentre la parte di elaborazione viene mantenuta separata poiché gestita in maniera indipendente ed autonoma dalle stesse entità, che nel nostro caso saranno agenti.

Il medium, che costituisce il collante del sistema, deve svolgere le attività di creazione/rimozione, comunicazione e distribuzione degli agenti all'interno del sistema e sincronizzazione/distribuzione delle loro azioni nel tempo.

Il sistema di coordinamento è quindi composto da entità, dal medium e da regole di coordinazione (che definiscono il comportamento del medium oltre a definire la struttura delle informazioni e degli agenti). In genere questi modelli di coordinazione vengono costruiti partendo dal concetto di spazio condiviso che può mantenere strutture con cui possono interagire gli agenti.

Lo spazio di tuple è un meta-modello di coordinazione *data-driven*, ossia l'interazione e la comunicazione degli agenti avviene esclusivamente tramite le informazioni ospitate nel medium di coordinazione. Questo meta-modello risulta essere molto utile in applicazioni in cui il numero di agenti che ne fanno parte (o che ne entreranno a fare parte dinamicamente) non è conosciuto a priori e in situazioni in cui le informazioni devono essere accessibili a tutte le entità.



Le tuple, le strutture dati utilizzate in questo sistema di coordinamento, sono collezioni ordinate di informazioni in genere eterogenee tra loro, e costituiscono il mezzo con cui gli agenti comunicano tra loro. Hanno vita propria all'interno dello spazio, sono accessibili da tutti gli agenti (tramite associazione, e non tramite indirizzamento) ma non mantengono alcun riferimento con essi.

Il linguaggio di coordinazione di riferimento per questo modello è *Linda*: gli elementi che quindi lo contraddistinguono sono le tuple, i template (che identificano una

classe di tuple) ed un meccanismo di matching (tramite cui avviene l'associazione tra una tupla ed un suo template; devono coincidere il numero dei campi, il tipo, i valori e la lunghezza delle stringhe).

Le primitive di base che permettono l'interazione sono *in(T)*, con cui si rimuove dallo spazio una tupla e successivamente la si legge; *out(T)* che inserisce una tupla nello spazio e *rd(T)*, che permette di leggere una tupla senza però rimuoverla dallo spazio.

Il vantaggio principale dello spazio di tuple è il disaccoppiamento temporale (le tuple hanno vita propria e persistono nello spazio fintanto che questo esiste o fino al momento della loro rimozione da parte di un agente che può essere entrato a far parte del sistema in un qualsiasi momento) e spaziale (le tuple rimangono sempre in quel determinato spazio ed ogni agente aggiuntosi successivamente o proveniente da un altro sistema, può sempre accedervi).

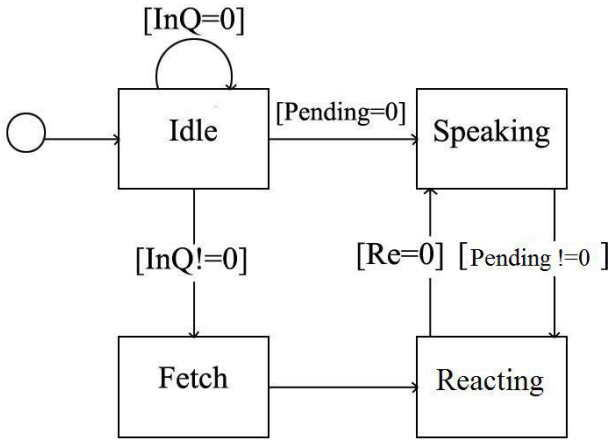
3.2.1. ReSpecT e Situated ReSpecT

Un meta-modello come lo spazio di tuple, per i motivi descritti precedentemente, agevola molti aspetti di comunicazione in genere problematici, tuttavia sarebbe ancora più adatto un modello orientato agli eventi, piuttosto che alle informazioni.

Introduciamo quindi il concetto di centro di tuple come estensione dello spazio di tuple, in cui il coordinamento non è di tipo *data-driven* ma segue un pattern di tipo *control-driven*, in particolare *event-driven*, dove ad ogni operazione interna (in entrata o in uscita) viene generato un evento che poi sarà catturato dall'infrastruttura.

Nei sistemi control-driven abbiamo una quasi netta separazione tra la componente di coordinamento e quella computazionale: i valori attuali dei dati manipolati dai processi non sono quasi mai coinvolti. Le parti computazionali (cioè gli agenti) sono trattate come delle scatole nere con delle ben definite interfacce di input/output; mentre i sistemi data-driven si occupano della coordinazione dei dati, i control-driven tendono a coordinare le entità.

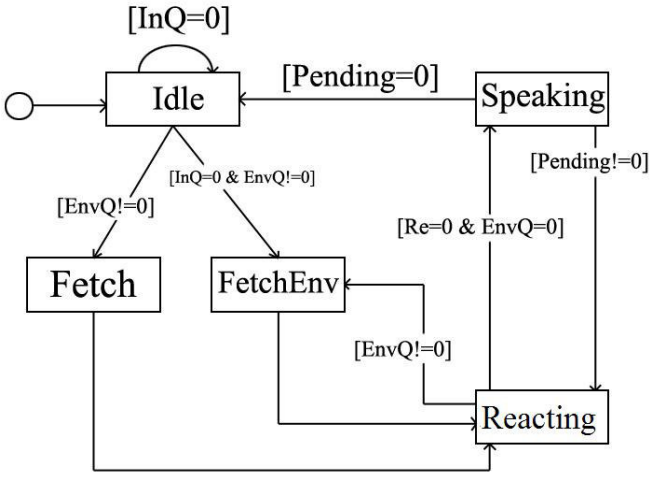
Per realizzare in concreto ciò, si è aggiunto uno strato logico sopra a Linda composto da un automa a stati infiniti, che effettua elaborazioni event-driven.



ReSpecT (*Reaction Specification Tuples*) è il linguaggio per questo nuovo modello di coordinamento, già più orientato verso i sistemi multi-agente. Esso dà la possibilità di specificare reazioni da eseguire in base al verificarsi di particolari eventi.

Le reazioni di ReSpecT hanno una struttura del tipo: *reaction(E,G,R)*, dove *E* è il template con cui fare matching, *G* è la guardia (la condizione che deve essere verificata dopo che l'evento *E* è stato catturato) e *R* è la lista delle reazioni che devono essere eseguite.

Situated ReSpecT è un'ulteriore estensione di ReSpecT, che dà anche la possibilità di specificare predicati di coordinamento relativi ad eventi riguardanti risorse esterne, ossia reazioni ambientali, raggiungendo così un livello di semantica nel linguaggio adeguato ai sistemi multi-agente, in cui è fondamentale il concetto di *situatedness*.

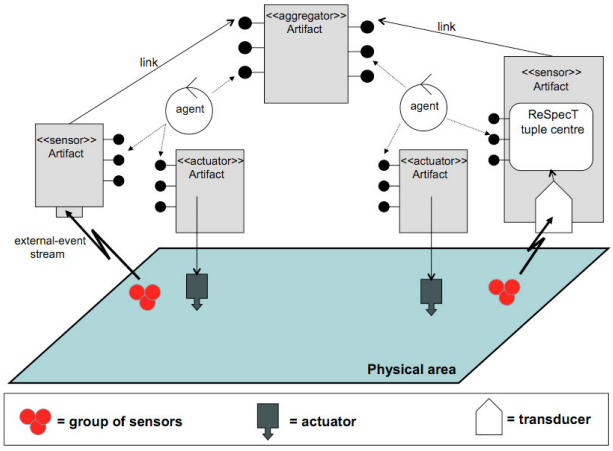


Il linguaggio che prenderemo in considerazione sarà quindi Situated ReSpecT.

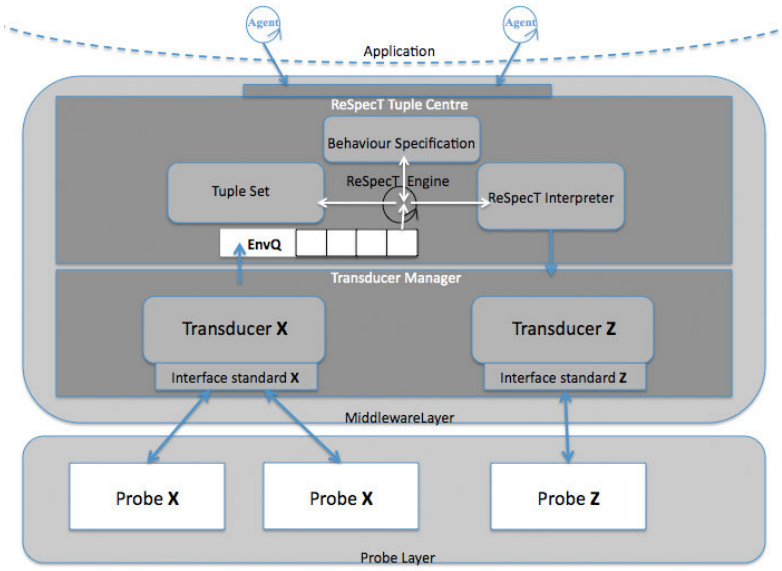
3.3. Infrastruttura software

Dopo aver discusso a livello generale la struttura logica dell'infrastruttura, vediamo ora nel concreto la sua realizzazione per poter poi integrare la tecnologia LEGO Mindstorms con Situated ReSpecT.

Occorre introdurre innanzitutto dei nuovi componenti, detti *Transducer*, ossia delle entità software che nascondano i dettagli tecnici della comunicazione fra sensori/attuatori ed il centro di tuple e che si svincolino quindi dal particolare tipo di tecnologia utilizzata.



Il *Middleware* sarà quindi logicamente composto da 2 componenti: il *Transducer Manager* (che si incarica di gestire i *Transducer*) e l'effettivo *ReSpecT Tuple Centre*.



L’architettura finale sarà in definitiva composta da:

- *Probe layer*

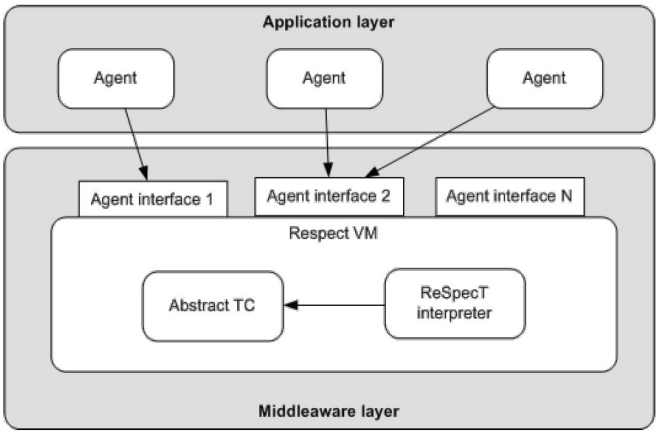
  Fa uso delle API software del costruttore, ossia le funzioni messe a disposizione dal fornitore dell’hardware che gestiscono le risorse fisiche a basso livello. Esso quindi crea uno strato software uniforme ed indipendente dalle specifiche hardware. Le unità software che si occupano della gestione a basso livello delle risorse fisiche all’interno di questo livello sono le *probe*.

- *Middleware layer*

  E’ formato da 2 strati:

  - *ReSpecT Tuple Center*

    che si compone di un *ReSpecT Interpreter* (il componente che si occupa di interpretare il codice ReSpecT, seguendo la particolare sintassi e semantica delle sue primitive, per poi eseguirlo nel centro di tuple) e di un *Abstract TupleCentre* (che realizza le funzionalità principali di un centro di tuple, astraendo dal linguaggio). Il macro-sistema che realizza l’infrastruttura principale del *Middleware* e che gestisce la creazione degli eventi e la loro notifica è il *ReSpecT VM*.



  - *Transducer Manager*

    Il *Middleware* utilizza le probe per ricevere informazioni che trasforma in eventi ambientali, occorrono quindi specifici componenti per ogni tipologia di probe con cui si vuole comunicare.

I *Transducer* incapsulano le logiche di protocollo per la comunicazione con le probe dello stesso tipo. In pratica, essi ricevono i messaggi da queste ultime (o viceversa le trasmettono), ne estraggono il contenuto e lo preparano (ne modificano la rappresentazione) per poterlo utilizzare all'interno del centro di tuple, poi lo passano al *Transducer Manager*, riportando però l'informazione così com'è senza interpretarla. Per ogni centro di tuple possono esserci *Transducer* di vario genere, in base ai tipi di probe da gestire.

Il *Transducer Manager* gestisce i *Transducer* ed incapsula le informazioni ricevute da questi ultimi in nuovi eventi che vengono poi notificati all'*Abstract TupleCenter*, e, viceversa, si occupa di indirizzare un evento in uscita dal centro di tuple al giusto *Transducer* (in quest'ultimo caso l'evento di output viene creato dal *ReSpecT Interpreter*). Quando vi è una richiesta di attuazione o di sensing da parte del centro di tuple verso un probe, il *Transducer*, gestito dal *Tranducer Manager*, cercherà di rimanere in attesa per una risposta, poiché la semantica di ReSpecT prevede la possibilità di specificare una sorgente ed una destinazione. Il *Transducer Manager* è incaricato inoltre di rimuovere o aggiungere dinamicamente i *Transducer* all'occorrenza.

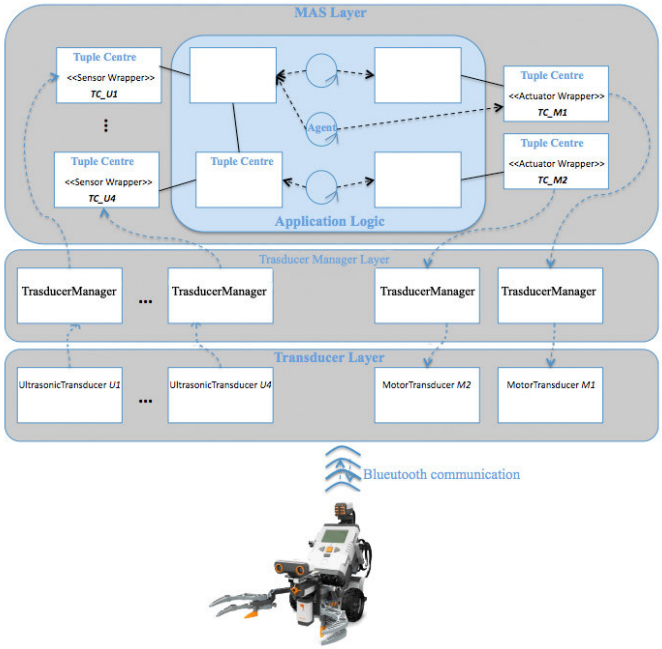
L'aggiunta del *Transducer Manager* non è fondamentale, ma la sua presenza alleggerisce i componenti e li rende meno complessi, sia funzionalmente, sia a livello implementativo; inoltre facilita la mappatura delle risorse e rende il sistema più robusto perché evita la centralizzazione.

Per il caso dei robot LEGO Mindstorms, il *Probe Layer* è rappresentato dalle API *iCommand*: quest'ultimo è un framework che permette di controllare i LEGO Mindstorms utilizzando il linguaggio java ed una connessione bluetooth (utilizza il firmware standard dei Mindstorms per ricevere comandi dal codice java presente su un computer remoto).

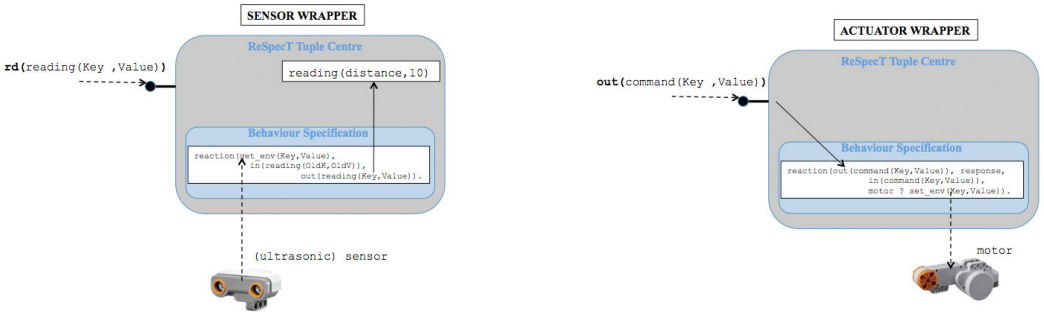
Le interfacce per gestire il robot sono state modellate seguendo le funzionalità sensoriali ed antropomorfe del robot: *IRobot*, *IHearing*, *IMotion*, *ITouch* e *IVision*.

Sono stati implementati poi i vari Transducer, uno per ogni tipo di sensore/attuatore fisico da gestire: il *LightTransducer*, il *TouchTransducer*, l' *UltrasonicTransducer*, il *SoundTransducer*, il *ColorTransducer* ed il *MotorTransducer*. Ognuno di essi è gestito come un thread indipendente e ha una struttura molto semplice.

Il *Transducer Manager*, invece, ha come supporto un HashMap che gli permette di tenere in memoria i *Transducer* da gestire, tuttavia per non sovraccaricare troppo di lavoro un unico *Transducer Manager*, è meglio usare un *Transducer Manager* per ogni *Transducer*. I *Transducer Manager* sono realizzati in modo tale da riceve informazioni dai *Transducer* per poi tradurle in eventi ReSpecT da inserire nel centro di tuple, inoltre essi ricevono notifiche dagli *Actuator Wrapper* con informazioni degli attuatori da passare ai *Transducer*: i *Wrapper*, sono stati introdotti come entità ausiliarie in grado di mappare informazioni del mondo esterno per renderle disponibili all'interno del MAS. Sono realizzati anch'essi con dei centri di tuple e ne viene creato uno per ogni specifica risorsa ambientale, al momento dell'istanziazione di un nuovo *Transducer*.



I wrapper si dividono in *Sensor Wrapper* (il centro di tuple che si occupa di raccogliere le informazioni provenienti da un sensore e renderle disponibili al MAS) ed *Actuator Wrapper* (il centro di tuple che raccoglie le informazioni provenienti dall'*Application Logic* per renderle disponibili agli attuatori esterni).

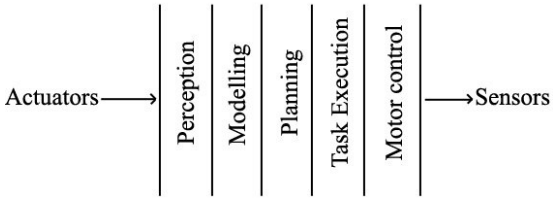


Nell'*Application Logic*, che è il cuore dell'applicazione, invece, viene implementata la parte più specifica del sistema, ossia il sistema di controllo (che come si vedrà più avanti, sarà a livelli), rappresentato dagli agenti.

3.4. Modello di comportamento

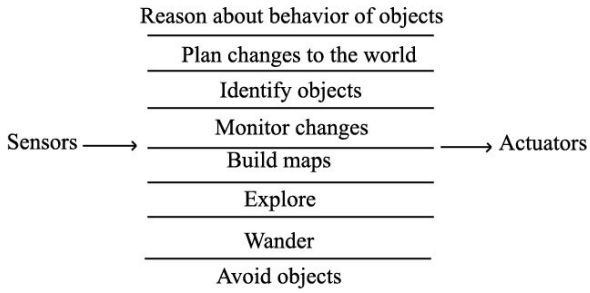
I robot devono processare numerose operazioni in real time, in un ambiente che cambia dinamicamente e che spesso è difficile da delineare in maniera precisa.

L'approccio più usato per interagire con l'ambiente è quello orizzontale, in cui a seguito del cambiamento dell'ambiente dovuto ad una propria azione, vengono elaborate in successione una serie di operazioni (unità funzionali), che portano dalla nuova percezione del mondo, ad un conseguente controllo degli attuatori in maniera adeguata alla situazione corrente (si ha una situazione di feedback).



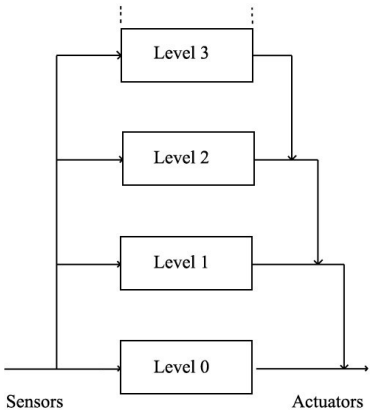
Un modello più efficace, invece, è vedere il tutto dal punto dei vista dei comportamenti richiesti al robot, si ha in questo caso una struttura verticale. Il robot deve agire in conseguenza agli stimoli del mondo ricevuti tramite i sensori, partendo dai comportamenti più elementari (come evitare di andare a sbattere) a quelli più

complessi (per esempio cercare di prevedere come si comporteranno gli altri oggetti presenti nell'ambiente).



In questa analisi bisogna tenere conto che il sistema può avere obiettivi multipli (vari compiti e con priorità differenti) e sensori multipli (occorre rilevare numerose misure, che nella maggior parte dei casi sono soggette ad errori ed inconsistenze), ma comunque deve essere in ogni caso robusto (bisogna considerare anche la spiacevole eventualità di guasti ad attuatori e sensori, situazione in cui il robot deve continuare ugualmente a svolgere il proprio compito) e deve permettere l'additività di nuovi componenti (deve consentire, in caso di necessità, l'aggiunta dinamica di sensori ed attuatori senza aver alcun tipo di problema).

Il modello più adeguato, quindi, è un'architettura verticale, costituita da vari livelli e strati, proposto in letteratura da Brooks. Grazie a questa architettura, infatti si potranno rappresentare classi di comportamento a prescindere dall'ambiente in cui il robot si trova; i livelli diventano sempre più specializzati e complessi andando verso l'alto.



I principali livelli di competenza individuati (a partire da quello più basico) che deve manifestare un robot, sono:

0 Evitare collisioni con oggetti

1 Vagare nell'ambiente

2 Esplorare ed osservare il mondo

3 Costruire una mappatura dell'ambiente e pianificare i tragitti

4 Rilevare i cambiamenti nell'ambiente

5 Riconoscere gli oggetti dell'ambiente ed eseguire operazioni specifiche su essi

6 Pianificare ed eseguire operazioni che portano l'ambiente a modifiche desiderate

7 Ragionare sul comportamento degli oggetti e reagire di conseguenza, modificando eventualmente il proprio comportamento

Su ognuno di questi livelli di competenza possiamo realizzare strati di controllo, con la possibilità di rimuovere o aggiungere nuovi strati all'occorrenza.

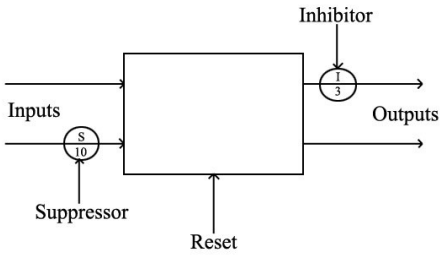
Ogni livello superiore si appoggia su quello sottostante, realizzando un'architettura ad inclusione, inoltre, come si può notare, si parte dai comportamenti più elementari per arrivare a quelli più complessi.

Questa architettura soddisfa i requisiti richiesti precedentemente: è possibile avere obiettivi multipli (ogni livello è indipendente dall'altro, e quindi è possibile perseguire più obiettivi parallelamente, seguendo particolari priorità), sensori multipli (il robot può eseguire più rivelazioni allo stesso tempo e metterle a confronto, riducendo il margine di errore), robustezza (anche se un nuovo livello, magari aggiunto recentemente, non riuscisse a raggiungere il proprio compito a causa di instabilità o errori, quelli sottostanti continueranno a lavorare correttamente, fornendo le funzionalità di base) ed additività (ogni livello è indipendente dall'altro, quindi se ne possono aggiungere tranquillamente altri).

Ogni strato è stato realizzato partendo da un piccolo set di moduli che comunicano l'uno con l'altro; ogni modulo è una macchina a stati finiti e realizza parte del

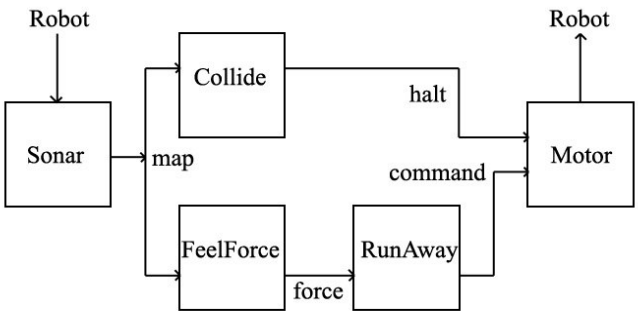
comportamento del livello di appartenenza, lavorando in maniera asincrona senza condividere memoria con gli altri moduli.

Ciascun modulo, se lo ritiene opportuno e se gli è concesso, può alterare o comunque influire sul comportamento di un altro modulo, sopprimendo o inibendo ingressi ed uscite, questo grazie ad una rete logica di comunicazione che nel primo caso agisce sugli input dell'altro, mentre nel secondo caso sugli output.



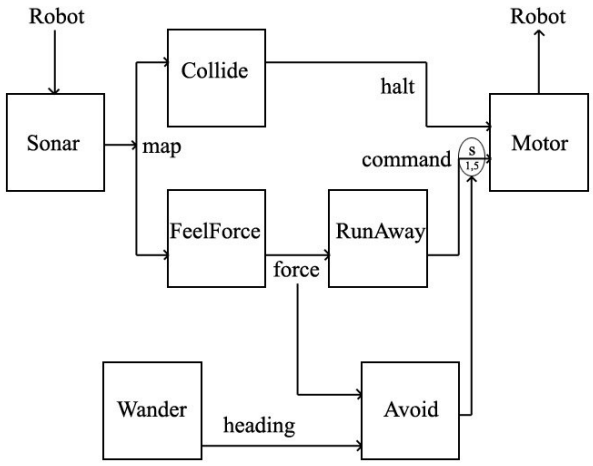
I moduli che compongono la struttura per realizzare, per esempio, il comportamento del *livello 0* (evitare gli ostacoli) potrebbero essere:

- *Motor*. Gestisce gli attuatori (li ferma o li attiva).
- *Sonar*. Rileva le distanze grazie al sensore ad ultrasuoni e crea una mappatura dell'ambiente.
- *Collide*. Grazie alle informazioni fornite dal modulo Sonar interviene (avverte il modulo Motor) se vi è un oggetto troppo in prossimità, per poterlo evitare.
- *FeelForce*. Dopo aver ricevuto informazioni dal modulo Sonar elabora i valori di velocità ed angolazione che il robot deve mantenere.
- *RunAway*. Rielabora le informazioni del modulo FeelForce, e se le ritiene significative, le passa al modulo Motor per poterle attuare.

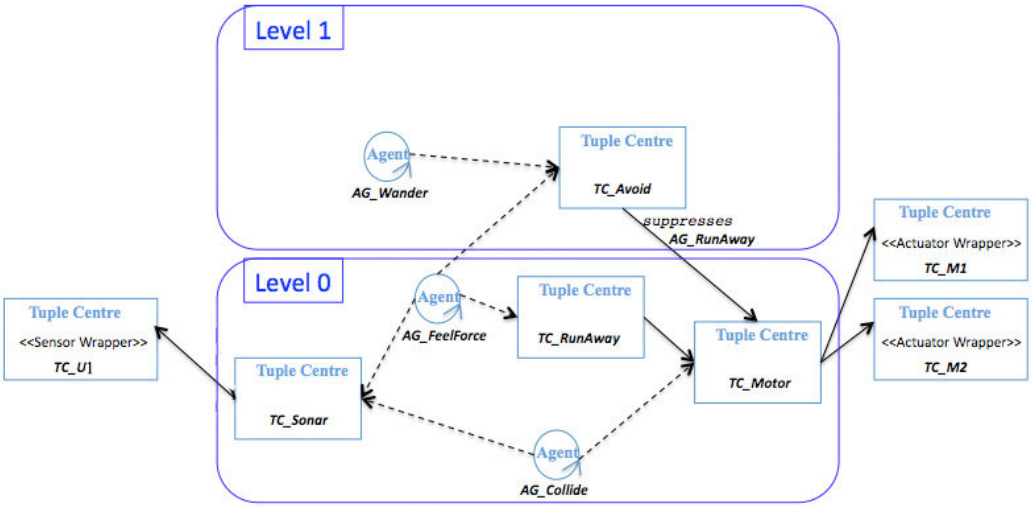


Per il *livello 1* (vagare nell'ambiente), invece si avrà:

- *Wander*. Genera valori di velocità ed angolazione casuali a brevi intervalli di tempo regolari.
- *Avoid*. Prende le informazioni dal modulo FeelForce del livello 0 e le combina con quelle generate dal modulo Wander, per ottenere un mix ottimale. Quando entra in funzione, sopprime l'uscita del modulo RunAway, sostituendosi ad esso per la comunicazione con il modulo Motor del livello 0.



Il robot, quindi ora si muoverà seguendo un itinerario ottimale, evitando gli ostacoli.



Tutta questa parte riferita al comportamento (compresi le strutture dati, gli spazi condivisi e gli agenti che attuano il controllo), verrà inclusa al top level, ossia nell'*Application Logic*.

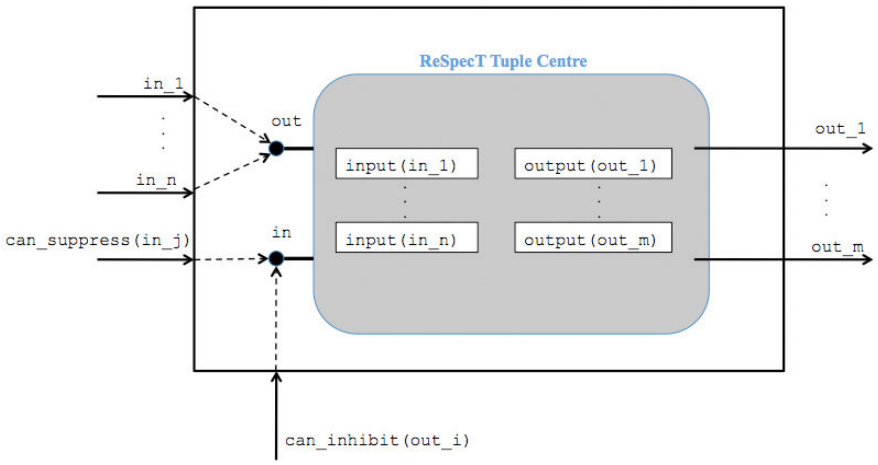
3.5. Architettura del MAS

Situated Respect permetterà di realizzare un controllo software per un insieme di componenti hardware soggetti a cambiamenti dinamici dell'ambiente esterno (occorre che il sistema reagisca opportunamente).

I segnali di input di un modulo sono rappresentati da operazioni di *out* eseguite da altri centri di tuple o agenti, mentre i segnali di output di un modulo continuano ad essere rappresentati da operazioni di *out* eseguite su centri di tuple esterni.

Nel caso in cui l'output sia destinato ad un agente, esso sarà in grado di catturare tali informazioni in maniera attiva, eseguendo un'operazione di *in/inp* sul centro di tuple in questione.

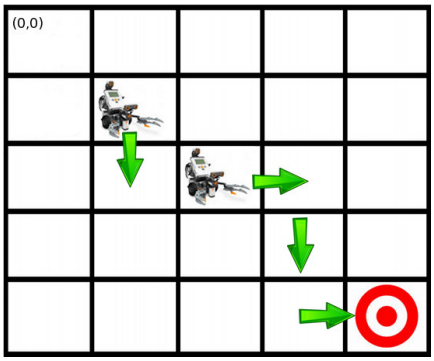
Ogni modulo memorizza al suo interno una tupla *input(Id)* per ogni agente o centro di tuple connesso alle proprie linee di ingresso (l'*Id* identifica la risorsa). La tupla *output(Id)*, invece, specifica quale centro di tuple o agente può ricevere dati dal modulo. Inoltre si possono usare delle tuple per sopprimere un input o per inibire un output, rispettivamente *can_suppress(In1,Tcs,Time)* e *can_inhibit(Out,Tci,Time)*.



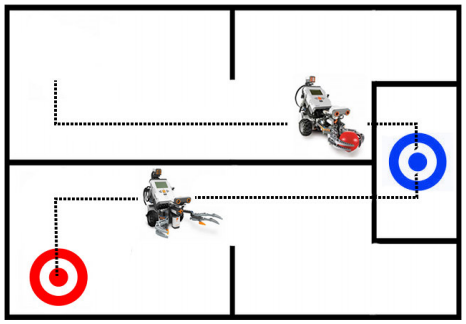
4. Casi d'uso

Per valutare l'efficacia dell'infrastruttura ad agenti descritta precedentemente si sono eseguite semplici prove, affidando ai robot degli obiettivi da portare a termine in cooperazione.

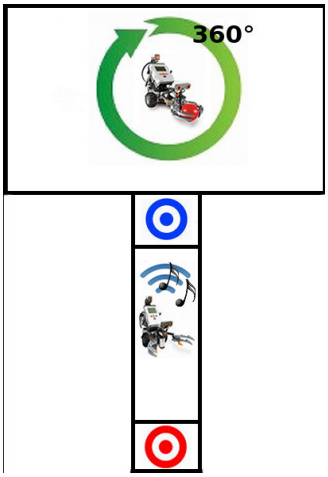
Come this way: Un Leader si sposta casualmente di casella in casella muovendosi in uno spazio diviso in griglia, fino ad arrivare nell'ultima casella target. Un Follower seguirà i movimenti del Leader, spostandosi di volta in volta nella precedente posizione di quest'ultimo. *Esito*: nessun problema riscontrato.



Help me: L'ambiente è diviso in due zone differenti, ognuna delle quali di competenza di un robot ed inaccessibile per l'altro; in più vi è una zona franca in cui invece può entrare chiunque. Un robot, possessore di una palla, si muove dentro la sua zona seguendo una linea guida fino alla zona franca. Qui deposita la palla e poi ritorna nella sua zona; successivamente l'altro robot, avvisato dal primo, entra nella zona franca, raccoglie la palla per portarla poi nella sua zona e lasciarla infine in un punto target. *Esito*: alcuni problemi con il sensore di luce nel rilevare la linea guida.



Listen to me: Un robot cerca un oggetto in un'arena, mentre un secondo attende fuori. Una volta raccolto l'oggetto, il secondo robot emette un segnale sonoro; il primo robot trasporta l'oggetto verso una zona franca seguendo la provenienza e l'intensità rumorosa del segnale acustico, poi rilascia l'oggetto. In seguito il secondo robot entra nella zona franca, raccoglie l'oggetto e lo porta nella zona esterna in una posizione target. *Esito*: grossi problemi dovuti soprattutto al sensore di suono.



Questi casi di studio hanno evidenziato come l'infrastruttura basata su Situated ReSpecT sia molto efficace come strumento per la coordinazione.

Tuttavia già a partire dal secondo caso (sono stati presentati in complessità crescente), emergono difficoltà che spesso fanno fallire l'esperimento (specialmente nel terzo caso in cui la qualità percettiva del sensore sonoro non è ottimale), ma ciò è dovuto soprattutto all'hardware dei LEGO Mindstorms che è limitato, non allo sviluppo dell'infrastruttura soprastante.

L'unica pecca software che accompagna un po' tutti gli esperimenti, è nello strato più basso, cioè il Probe Layer, che a volte risulta inadatto ed implementa alcuni aspetti di comunicazione in maniera non molto efficiente (ha soprattutto la predisposizione per controllare un unico robot), ma questo dipende da iCommand, e non ci si può porre riparo (volendo, al suo posto, si potrebbe usare anche un altro framework).

5. Infrastruttura ReSpecT in soccer robot

Ora che abbiamo analizzato sia alcune soluzioni concrete adottate per robot reali che hanno fatto parte di team di passate edizioni della RoboCup sia, invece, soluzioni ad agenti per il coordinamento di robot LEGO Mindstorms utilizzati esclusivamente in laboratorio, possiamo fare un confronto.

Sotto il profilo hardware, come visto in precedenza, i robot LEGO Mindstorms non hanno raggiunto un livello tecnologico tale da poter sperare di essere competitivi nella RoboCup (i sensori non sono affidabili più di tanto, inoltre come osservato per la maggior parte dei soccer robot, anche i modelli più obsoleti privilegiano senza dubbio spesso una percezione completamente visiva (tramite videocamera), che risulta la più efficace dal punto di vista informativo, ma che i Mindstorms non possono avere. Inoltre, in fatto di resistenza agli urti, i soccer robot sono abbastanza pesanti e costituiti da parti non facilmente smontabili; i sistemi calcianti sono generalmente realizzati con sistemi idraulici, ad aria compressa o a molla per avere più potenza.

Tutto questo porta ad escludere l'uso dei Mindstorms in ambito agonistico, tuttavia grazie ad essi, come già visto, si sono potuti sperimentare in tempi relativamente brevi, modelli di coordinazione ad agenti. Il passo successivo è cercare di immettere i modelli software dei robot Mindstorms in robot simili a quelli presentati inizialmente.

Infatti dal punto di vista dell'infrastruttura software, il motore Situated ReSpecT, come mostrato, prescinde dal tipo di hardware, quindi basterà solo cambiare la parte relativa al livello più basso, ossia il Probe Layer che dipenderà dal tipo di robot in uso e dal suo particolare linguaggio di programmazione, quindi dalle sue specifiche API per interagire con esso.

Inoltre occorrerà implementare i nuovi Transducer per quello specifico robot, poiché essi dipendono sia dai sensori/attuatori montati sul robot, sia dalla forma con cui le informazioni vengono passate dal Probe Layer, quindi questa parte andrà realizzata ad hoc. Tutto il resto rimane immune dal particolare tipo di tecnologia.

5.1. Modello di comportamento per soccer robot

A questo punto non rimane altro che occuparci della parte specifica, ossia del comportamento, adattando il modello generale visto per i Mindstorms ad un modello consono per giocare a calcio. Quindi si può dire che, dopo l'analisi condotta sui vari tipi di prototipi di robot da competizione, sotto il profilo del comportamento (e quindi a livello di Application Logic), ci si è accorti che la soluzione più ricorrente e che comunque sembra la più logica ed efficace, è dare priorità ai comportamenti reattivi, cioè dettati dagli immediati stimoli ricevuti dai sensori, per permettere al robot di eseguire azioni basilari critiche nel minor tempo possibile, ma occorre conferirgli anche la possibilità di elaborare piani e strategie complesse quando la situazione ed i tempi di esecuzione lo consentano, poichè questi gli permettono di variare il gioco e creare più opportunità di segnare.

A questo scopo si è visto come la struttura a livelli si adatta bene a tal fine: si può infatti costruire un tipo di comportamento in livelli con priorità decrescente. Si avranno quindi quelle attività critiche e più importanti ai livelli più bassi; mano a mano che si sale di livello le azioni passeranno dal tipo reattivo a quello pianificatore/strategico. Il comportamento sarà quindi simile a quello umano: parte istintivo (più efficace nell'immediato, ma elementare) e parte razionale (più efficace nel complesso, ma più lento da mettere in atto).

Ovviamente non si può fare un'analisi specifica (specialmente dal punto di vista tattico) perché i tipi di percezioni e di possibilità dipendono dai sensori che si hanno a disposizione, non avendo come riferimento alcun particolare di robot, quindi ci si proverà a focalizzare su aspetti generali.

Riprendendo i livelli di competenza (già elencati in precedenza) che un generico robot deve eseguire, proviamo a ricondurli genericamente al gioco del calcio:

0 Evitare collisioni con oggetti

E' un compito di primaria importanza che è sempre valido, in ogni situazione. L'unica collisione consentita è quella con la palla, uno scontro con un avversario o comunque con un altro giocatore potrebbe invece danneggiare il robot, fargli perdere l'orientamento o l'equilibrio, è quindi fondamentale cercare di evitare qualsiasi tipo di scontro accidentale.

1 *Vagare nell'ambiente*

Solo se è riferito al sapersi muovere nell'ambiente, perché altrimenti non è utile vagare a caso in un campo da calcio (è più opportuno mantenere la posizione e agire solo se ritenuto proficuo).

2 *Esplorare ed osservare il mondo*

L'esplorazione nel gioco da calcio non è utile, potrebbe però essere utile ricavare delle informazioni sull'osservazione di eventuali ostacoli o altri oggetti presenti in campo.

3 *Costruire una mappatura dell'ambiente e pianificare i tragitti*

Utile per avere una visione globale del campo e quindi abbozzare le prime tattiche di gioco elementari (movimenti senza palla in caso di attacco o sul portatore di palla nel caso di difesa). E' utile anche per mantenere la posizione.

4 *Rilevare i cambiamenti nell'ambiente*

Occorre che un giocatore si attivi nella maniera adeguata, ogni volta che un altro giocatore (avversario o amico) o la palla cambiano posizione.

5 *Riconoscere gli oggetti dell'ambiente ed eseguire operazioni specifiche su essi*

E' fondamentale che vengano riconosciuti gli oggetti chiave (palla, porte e giocatori). Il riconoscimento più difficile forse risulta quello della squadra di appartenenza di un giocatore.

6 *Pianificare ed eseguire operazioni che portano l'ambiente a modifiche desiderate*

Fondamentale per attuare strategie, tattiche o passaggi particolari che porterebbero a segnare un goal più facilmente, o a non prenderlo in caso di attacco subito.

7 *Ragionare sul comportamento degli oggetti e reagire di conseguenza, modificando eventualmente il proprio comportamento*

Questa caratteristica è importante soprattutto in ambito difensivo: occorre infatti prevedere la traiettoria della palla per potere intercettare un tiro o un passaggio di un avversario. Inoltre si possono sfruttare movimenti avventati degli avversari per potere trarre vantaggio e spostare i propri giocatori nella posizione giusta. Occorre anche valutare i movimenti dei propri compagni di squadra, potrebbe essere infatti necessario cambiare assetto (difesa o attacco) o variare ruolo ad un giocatore.

Questa suddivisione in livelli, come detto, è generica; occorrerebbe specializzare questo modello per permettere ad un robot di giocare a calcio; alcuni livelli risultano superflui o non applicabili, mentre è possibile che sia necessario aggiungerne degli altri più particolari.

Si proverà, quindi, a dare una linea guida su ciò che differisce o che viene specializzato rispetto alle considerazioni appena effettuate sui vari livelli, in base al ruolo che un giocatore occupa in campo, perché differenti ruoli implicano anche compiti, obiettivi e competenze individuali diversi, che occorre ottimizzare.

Alcuni livelli, specialmente per il portiere, che è molto più limitato, dovranno essere eliminati, mentre per tutti, in generale, occorre aggiungere un livello di comportamento reattivo di base da esibire nei casi più critici (quelli che richiedono velocità di esecuzione) e che varia da ruolo a ruolo.

5.1.1. Portiere

Il portiere ha il compito fondamentale di non fare segnare l'avversario. E' un giocatore critico, perché è quello che generalmente fa la differenza: con un portiere progettato male, difficilmente si può vincere una partita. Le sue caratteristiche particolari sono:

- Non necessita di evitare le collisioni con oggetti, poiché non deve vagare all'interno del campo. Potrebbe però capitare che un giocatore (avversario o compagno) si avvicini troppo alla porta tanto da interferire con esso.

  In questo caso il portiere potrebbe prevedere un meccanismo di semplice arresto per non andargli addosso, ma è da escludersi che debba eseguire un

movimento alternativo che lo faccia spostare lungo un percorso per evitare l'ostacolo poiché potrebbe portarlo fuori dalla porta, lasciandola sguarnita.

- E' da escludersi ovviamente un livello che permetta al robot di vagare per il campo. Il portiere di base può spostarsi ma solo tramite movimenti laterali davanti alla porta.

- Dal momento che non può girare libero per il campo, viene di conseguenza che non deve essere abilitato all'esplorazione, tuttavia l'osservazione è fondamentale, per disporsi nella posizione giusta.

- Un livello specifico da aggiungere di bassa computazione, ossia dal comportamento di tipo reattivo, è quello dedicato a *parare la palla*: nel caso sopraggiunga la palla, il portiere deve immediatamente buttarla via il più lontano possibile, senza ragionarci sopra.

- Il riconoscimento della palla è fondamentale, e la pianificazione dei movimenti dipende da esso: il portiere deve rimanere nella traiettoria tra la porta e la palla scegliendo la posizione migliore, ossia privilegiando la traiettoria dritto per dritto (è la situazione di tiro più probabile da parte degli avversari).

- Deve rimanere sempre vigile, per notare i cambiamenti dell'ambiente: nel caso un giocatore gli copra la visuale della palla, deve spostarsi fino a vederla di nuovo. Quest'ultimo comportamento dovrebbe essere eseguito continuamente, anche se non la dovesse vedere perché troppo distante. Inoltre, poiché di base ha un comportamento di tipo reattivo (butta via la palla senza pensarci), deve stare attento a non avere davanti a se un giocatore (avversario o compagno) affinchè in caso di rinvio non si verifichi uno spiacevole effetto "boomerang", ossia che lo colpisca involontariamente e la palla ritorni indietro rischiando di prendere un rocambolesco goal.

- Al livello più alto, ossia ragionare sul comportamento di oggetti e reagire di conseguenza, si può dire che l'azione più particolare che un portiere può eseguire a sua discrezione in certe circostanze è l'*uscita*: nel caso la palla sia più vicina (in possesso dell'avversario o vacante) dei difensori, il portiere deve dirigersi immediatamente verso di essa per rinviarla su (o per chiudere lo

specchio della porta all'avversario), per poi ritornare subito in posizione, davanti alla propria porta.

5.1.2. Difensore

Il difensore ha come obiettivo principale non fare avanzare l'avversario con la palla. In genere deve essere il giocatore più "intelligente", perché deve prevedere sia la mossa futura dell'avversario, sia la possibile traiettoria della palla.

- Ovviamente il task più elementare è evitare le collisioni, mentre per quanto riguarda vagare nell'ambiente si può dire che non è molto utile girare in campo a caso: è più efficace mantenere una posizione, in modo tale da coprire una certa area del campo.

- Poiché è più importante mantenere la posizione, un difensore non deve andare in esplorazione, ma deve ovviamente osservare il mondo, per rilevare il sopraggiungere di avversari o della palla.

- Un comportamento reattivo di base, che deve essere eseguito subito a seguito dell'osservazione del mondo, è *liberarsi della palla* se necessario, ossia spazzare l'area. Questo comportamento di tipo reattivo deve scattare nel caso il robot avverta la presenza di un robot avversario e della palla (può averne anche il possesso) a distanza molto ravvicinata. In questo caso il difensore spazza via la palla a caso verso la metà campo avversaria, liberando la propria area per non correre il rischio che ne entrino in possesso gli avversari.

- Dall'osservazione del mondo occorre cercare di costruirsi un punto di vista locale e se possibile globale (ma nella maggior parte dei casi richiederebbe la collaborazione degli altri robot). In caso l'avversario abbia la palla, cercare di pianificare la posizione migliore per andargli incontro per fare pressing.

- In base ai cambiamenti dell'ambiente deve poi *riprendere la posizione*: se si riconquista palla, e si passa alla fase di attacco, i difensori devono riprendere la propria posizione arretrando, se si sono portati troppo in avanti nell'eseguire pressing.

- Tra le operazioni più elementari da pianificare che portano l'ambiente a modifiche desiderate c'è il *passaggio*: nel caso il difensore entri in possesso della palla e non sia pressato da nessun avversario, può ragionare su come giocarla. Quindi può tentare di individuare un compagno di squadra che gioca in attacco per passargliela.
- Se consideriamo le azioni strategiche frutto del ragionamento al top level, si possono pensare a tattiche (anche cooperative) caratteristiche del difensore:
    - *Difesa a zona*: Nel caso un difensore vada sul portatore di palla avversario, l'altro può avanzare leggermente, disponendosi sulla sua diagonale, come richiede la marcatura a zona. In questo modo è più facile intercettare la palla, chiudere i buchi e cambiare marcatura nel caso l'avversario passasse la palla.
    - *Pressing ed intercettazione*: Il difensore che va sul portatore di palla, invece, potrebbe calcolare l'angolazione di rotazione dell'avversario, per mettersi nella giusta posizione per chiudergli la visuale di tiro/passaggio, in modo da prevedere così la direzione della palla.
    - *Contropiede*: Nel caso il difensore fosse in possesso della palla ed avesse la via completamente libera verso la porta avversaria potrebbe avanzare per cercare di andare a segnare. In questo caso comunicherà all'attaccante della sua stessa fascia di scambiarsi di ruolo, in modo tale che l'attaccante vada subito a posizionarsi come difensore per non rimanere scoperti in caso la palla venisse persa durante l'azione di attacco.

5.1.3. Attaccante

L'attaccante ha come obiettivo primario segnare. Da esso dipende il risultato della partita. Le sue caratteristiche principali sono:

- Anch'esso deve in primo luogo saper evitare gli ostacoli.
- Come gli altri, non deve vagare senza criterio nell'ambiente, perché non è di utilità. Il movimento più elementare che deve cercare sempre di eseguire è

liberarsi in area avversaria: in caso la palla sia nella propria metà di campo, si può fare in modo che gli attaccanti facciano in continuazione lenti movimenti orizzontali, per eseguire uno smarcamento elementare automatico e non premeditato.

- Non occorre esplorazione ma è comunque fondamentale l'osservazione, utile per rilevare avversari e per distinguere la porta, in modo da rimanere girato verso di essa.

- Il comportamento di base reattivo dell'attaccante è *tirare di prima*: nel caso la palla arrivi in qualche modo all'attaccante e nessun avversario sia troppo vicino, il giocatore deve tirare senza pensarci, in modo da sorprendere il portiere (imprimendo così anche più forza e velocità alla palla se questa era già in movimento) ed avere in questa maniera una buona probabilità di segnare. Occorre notare che in conseguenza al livello precedente (se eseguito correttamente) l'attaccante dovrebbe già essere girato verso la porta avversaria.

- Come in difesa, anche in attacco occorre effettuare pressing sugli avversari per cercare di impossessarsi della palla. Occorre quindi elaborare il tragitto per raggiungere una posizione ottimale per farlo. Inoltre occorre anche pianificare la posizione ottimale quando si raggiunge la palla per tirare.

- In caso di cambiamento dell'assetto dell'ambiente, l'attaccante deve essere pronto. Ad esempio, deve effettuare *il recupero della palla ed il riposizionamento*: se la palla viene persa, ossia viene tirata dietro all'attaccante dall'avversario, esso dovrebbe cercare di inseguirla almeno fino alla metà campo per riconquistarla, dopodiché, se fallisce nell'intento, deve riprendere la sua posizione di default.

- Come per il difensore, tra le operazioni di pianificazione che portano l'ambiente a modificarsi a proprio vantaggio, c'è il *passaggio*: se l'attaccante rileva un giocatore avversario a distanza ravvicinata, può cercare di passarla ad un altro attaccante che si trova in una posizione migliore per tirare.

- Tra le operazioni di livello più alto, basate su ragionamento, si può considerare lo *smarcamento*: nel caso l'attaccante non avesse la palla, ma

quest'ultima fosse comunque in possesso della propria squadra, allora l'attaccante può smarcarsi, cioè trovare una posizione ideale in cui nella traiettoria fra esso ed il giocatore con la palla non vi sia alcun ostacolo; questo permette di poter ricevere potenzialmente un passaggio.

5.2. Osservazioni

Questi comportamenti sono una possibile soluzione, utilizzando un'architettura dell'Application Logic a livelli, ricavati sia dai più significativi comportamenti di robot da competizione esistenti e sia da tattiche reali di calcio. Era possibile considerare anche uno scenario in cui tutti i robot fossero alla pari (non ci fossero divisioni di ruolo), e che il comportamento fosse scandito invece dal modulo e dalle posizioni di particolari formazioni, come già mostrato per alcuni tipi di agenti. Bisogna aggiungere che essendo aspetti generali non è inoltre stata fatta alcuna distinzione in base al tipo di competizione, al contrario di come invece erano stati presentati inizialmente i robot delle passate edizioni della RoboCup.

Comunque sia, il comportamento del portiere deve essere sviluppato a parte, perché è un ruolo critico e deve avere caratteristiche totalmente diverse e soprattutto specializzate rispetto agli altri. Per quanto riguarda invece difensori ed attaccanti, si poteva anche generalmente utilizzare un unico robot che avesse entrambe le caratteristiche a seconda della circostanza, solamente che si presuppone che questo implichi il fatto di avere un'architettura comportamentale troppo complessa e che deve fare continuamente ricorso all'interazione fra robot per deliberare (per decidere chi deve rimanere in difesa per non lasciarla sguarnita, chi ha la miglior possibilità di segnare, chi deve andare sulla palla). Tutto ciò potrebbe portare a dover considerare per ogni situazione troppi compiti ed a volte potrebbe capitare che l'agente scelga quello sbagliato, o che l'algoritmo comportamentale non sia sufficientemente complesso per poterlo attuare adeguatamente. Il punto chiave della suddivisione in ruoli è la disposizione in campo ed il mantenimento della posizione che rende efficiente il comportamento del robot.

Tutte queste però sono supposizioni e si fermano ad un livello teorico: andrebbero sperimentate concretamente.

6. Conclusioni

La RoboCup offre l'occasione di sperimentare e sviluppare concretamente tecniche di intelligenza artificiale all'interno di sistemi multi-agente distribuiti. I robot variano notevolmente in base alla competizione che devono disputare, tuttavia si sono identificati alcuni aspetti generali che la maggior parte dei robot ha adottato, e che poi si sono presi come riferimento per adattarla in seguito a seconda del caso. La maggior caratteristica riscontrata è quella di presentare comportamenti di tipo reattivo quando possibile, perché sono quelli più veloci, più efficaci da eseguire e spesso sono gli unici che possono venire eseguiti in un tempo molto breve (come richiedono i tempi di gioco), ma nel contempo un robot deve elaborare modelli e piani per eseguire operazioni di livello più complesso, i quali richiedono un'opera di ragionamento. Per questo scopo è molto importante avere una visione sia locale che globale della situazione (per quella globale è richiesta inoltre un'operazione di fusione delle conoscenze di tutti gli agenti, che è molto complessa e non sempre è attuabile).

La sperimentazione sui robot LEGO Mindstorms ha portato allo sviluppo di un'infrastruttura flessibile, scalabile e robusta, basata su un linguaggio adatto alla coordinazione ed interazione tra agenti. Unendo le caratteristiche dei due tipi di robot si può ipotizzare di utilizzare di base l'infrastruttura Situated ReSpecT sperimentata sui robot Mindstorms, implementando allo strato superiore (quello applicativo) una serie di comportamenti basati su livelli di competenza crescenti, partendo dal livello più semplice, fondato su reattività agli stimoli fino ad un livello più sofisticato basato su ragionamento e pianificazione. Si pensa che questo tipo di schema generale sia applicabile ad un numero considerevole di robot, qualsiasi sia l'hardware (se fisico) di cui sia composto e che sia una linea guida per tutti i robot, indipendentemente dalla competizione RoboCup. Ovviamente non si può scendere più nello specifico se non si prende in considerazione un particolare robot di riferimento.

A questo punto rimane solo da realizzare concretamente tali comportamenti e sperimentarli su un robot da competizione, appoggiandosi sull'infrastruttura di Situated ReSpecT.

Bibliografia

[01] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa, Hitoshi Matsubara
Sony Computer Science Laboratory
Higashi-Gotanda, Shinagawa, Tokyo, Japan
Dept. of Mechanical Engineering, Osaka University
Suita, Osaka, Japan
Electrotechnical laboratory
Umezono, Tsukuba, Japan
RoboCup: A Challenge Problem for AI and Robotics

[02] Nadir Ould Khessal
Temasek Engineering School. Temasek Polytechnic, Singapore
Towards a Distributed Multi-agent System for a Robotic Soccer Team

[03] Akihiro Matsumoto, Hiromasa Nagai
Dept. of Mechanical Engineering, Toyo university, Kawagoe, Saitama, Japan
Decision Making by Characteristics and the Interaction in Multi-agent Robotics Soccer

[04] Joao Silva, Nuno Lao, Antònio J.R. Neves, Joao Rodrigues, Josè Luìs Azevedo
IEETA/Department of Electronics, Telecommunications and Informatics,
University of Aveiro, Portugal
Obstacle Detection, Identification and Sharing on a Robotic Soccer Team

[05] Michael Plagge, Boris Diebold, Richard Gunther, Jorn Ihlenburg, Dirk Jung, Keyan Zahedi, Andreas Zell
W.-Schickard-Institute for Computer Science, Dept. of Computer Architecture, Tubingen, Germany
Design and Evaluation of the T-Team of the University of Tuebingen for RoboCup '98

[06] E.Pagello, F.Montesello, A.D'Angelo, C.Ferrari
Dept. of Electronics and Informatics, Padua University, Italy
Inst. LADSEB of CNR, Padua, Italy
Dept. of Mathematics and Informatics, Udine University, Italy
A Reactive Architecture for RoboCup Competition

[07] Sven Behnke, Jurgen Muller, Michael Schreiber
Albert-Ludwigs-University of Freiburg, Computer Science Institute, Freiburg, Germany
Playing Soccer with RoboSapien

[08] Andrea Omicini
Università degli Studi di Bologna - Seconda Facoltà di Ingegneria, Cesena, Italy
Coordination Models & Languages

[09] Andrea Omicini, Matteo Casadei, Elena Nardini, Alessandro Ricci
Università degli Studi di Bologna - Seconda Facoltà di Ingegneria, Cesena, Italy
Tuple-based Coordination: From Linda to ReSpecT & Tucson

[10] Matteo Mosca
Università degli Studi di Bologna - Seconda Facoltà di Ingegneria, Cesena, Italy
Coordinazione Sociale di Robot Mindstorm in ReSpecT

[11] Marco Alessi
Università degli Studi di Bologna - Seconda Facoltà di Ingegneria, Cesena, Italy
Sviluppo di un Framework Concettuale e Tecnologico per l'Integrazione di Lego Mindstorm e ReSpecT

[12] Matteo Amaducci
Università degli Studi di Bologna - Seconda Facoltà di Ingegneria, Cesena, Italy
Esperienza in Robotica con Lego Mindstorm e ReSpecT

[13] Maicol Urbinati
Università degli Studi di Bologna - Seconda Facoltà di Ingegneria, Cesena, Italy
Tecnologie di Coordinazione per la Situatedness in Ambito Robotico

[14] Alberto Pini
Università degli Studi di Bologna - Seconda Facoltà di Ingegneria, Cesena, Italy
Studio e Sperimentazione di Modelli di Interazione e Coordinazione Software di robot LEGO Mindstorms

[15] *http://www.robocup.org/*

[16] *http://en.wikipedia.org/wiki/RoboCup*