CAPITOLO 5

CONTROLLO DI ALTO LIVELLO

In questo capitolo è illustrato l'utilizzo di tecniche di I.A. per lo sviluppo di comportamenti avanzati per il ruolo di portiere. Il sistema di controllo di alto livello adottato è basato su un *Behaviour Engine* a regole fuzzy che elabora i dati in ingresso convertendoli in predicati fuzzy e restituisce delle attuazioni da inviare alla logica di basso livello. L'utilizzo della logica fuzzy è legato alla rappresentazione dell'incertezza di cui è affetto il modello del mondo fornito dai sensori, nel nostro caso il sistema di visione. In ogni momento i comportamenti vengono elaborati dal *Behaviour Engine* che decide quale comportamento attivare e in che misura. Le attivazioni dei comportamenti sono quindi accompagnate da un peso che ne rappresenta la desiderabilità. Le attivazioni vengono successivamente miscelate in funzione dei loro pesi, dando così origine alle attuazioni.

La Logica Fuzzy

Il termine fuzzy è stato introdotto all'interno della comunità scientifica negli ultimi anni sessanta da Lofti Zadeh, un ricercatore proveniente dall'area della teoria dei sistemi. Egli introdusse la formalizzazione dei concetti fuzzy per sorpassare le limitazioni dei tradizionali modelli numerici, i quali sono poco diversi dal modo di ragionare usato comunemente dalle persone. Dopo alcuni anni di sviluppo teorico, le prime applicazioni dei concetti fuzzy apparvero principalmente nel campo dei sistemi di controllo e della rappresentazione della conoscenza. All'inizio degli anni ottanta, si iniziò a considerare interessante l'approccio fuzzy per sviluppare applicazioni industriali ed in pochi anni i controllori fuzzy iniziarono ad apparire anche nelle applicazioni della vita quotidiana. Ora i concetti fuzzy sono ampiamente usati in controlli automatici, sistemi esperti ed in molti altri campi.

Nella logica fuzzy il grado di verità di una proposizione è fuzzy e solitamente assume valori nell'intervallo reale [0,1]. Tutti gli elementi della logica del primo ordine possono essere estesi alla logica fuzzy; in particolare possiamo avere:

- predicati fuzzy che possono essere parzialmente veri (es. alto, veloce, grosso), in contrasto con i predicati crisp che possono essere solo veri o falsi (es. mortale, più grande di);
- modificatori fuzzy che servono per modificare il valore di verità di una formula logica (es. molto, abbastanza, estremamente), l'unico modificatore crisp è il not;
- quantificatori fuzzy che quantificano gli elementi che prendono parte ad una formula (es. pochi, molti, spesso, usualmente);

- operatori fuzzy, per combinare formule e valori logici. Gli operatori fuzzy sono estensioni dei tradizionali operatori logici and, or, not. Ci sono molti differenti approcci per interpretare queste estensioni, tra quelle più comuni: per l'and è prendere il minimo delle funzioni di appartenenza, per l'or è prendere il massimo e per il not è prendere il complemento a uno.

Nella logica polivalente, o fuzzy, non hanno più ragion d'essere i principi della logica bivalente: il principio di non contraddizione e il principio del terzo escluso.

- Il *principio di non contraddizione*, afferma che un elemento (x) non può appartenere contemporaneamente ad un insieme A e alla sua negazione Ā. La loro intersezione è l'insieme vuoto di tutti gli elementi a valore zero.
- Il *principio del terzo escluso* afferma che l'unione di un insieme e del suo complemento produce l'insieme universo X al quale qualsiasi elemento appartiene, cioè: (A U Ā) = X.

I Fuzzy set

Un fuzzy set è un'estensione del concetto matematico d'insieme. Un insieme X è definito da una funzione d'appartenenza booleana μ che, applicata ad un elemento x, ritorna *vero* se x appartiene all'insieme X, *falso* altrimenti.

Nel caso di un insieme fuzzy, la funzione d'appartenenza non è più booleana ma è una funzione continua sull'intervallo [0,1].

Se x appartiene a X allora μ(x)=1, se non appartiene a X allora μ(x)=0. Tuttavia, è anche possibile avere una parziale appartenenza di x a X. In questi casi una funzione rappresenta il grado d'appartenenza di x all'insieme X. Un esempio è mostrato in *Figura 5.1*. L'insieme degli uomini e quello degli ingegneri sono insiemi definiti mentre gli adulti ed i ragazzi hanno bordi sfumati.

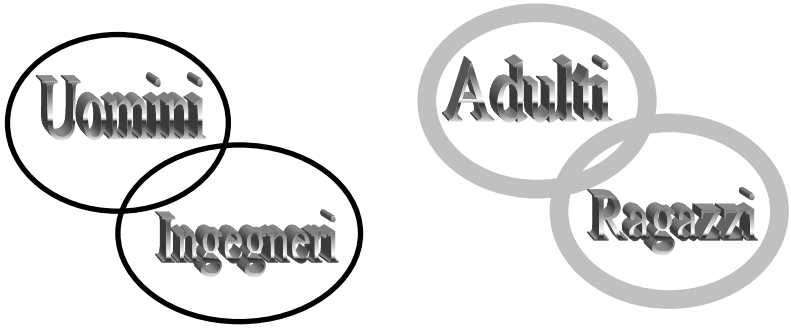


Figura 5.1 - L'insieme degli uomini e quello degli ingegneri è ben definito a differenza di quello degli adulti e dei ragazzi.

In altri termini, i concetti di adulti e di ragazzi che abbiamo in mente sono compatibili con un'appartenenza graduale degli elementi ad entrambi gli insiemi nella zona sfumata.

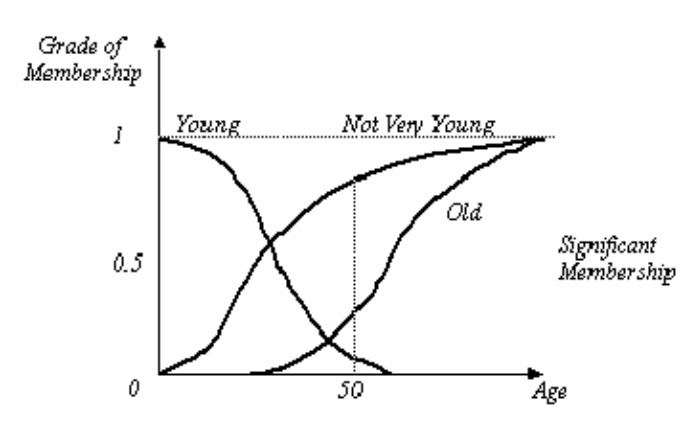
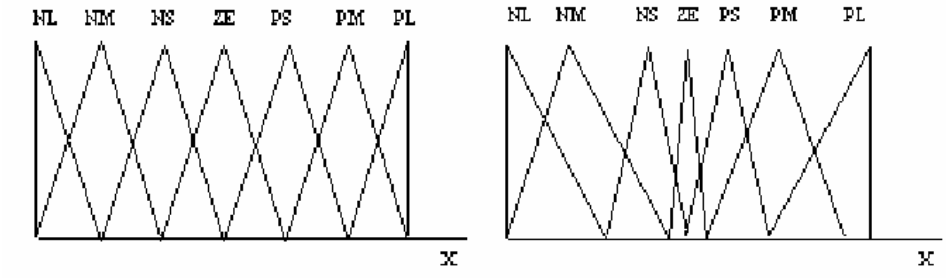


Figura 5.2 - Un esempio di Membership Function

In *Figura 5.2* viene mostrato un esempio di una funzione d'appartenenza che definisce tre insiemi fuzzy: giovani, non giovani ed anziani. Una persona di cinquant'anni può appartenere con diverso grado ai tre insiemi fuzzy. In molte applicazioni le funzioni di appartenenza non sono funzioni generiche, ma, sono definite come triangoli o trapezoidi. Queste forme possono essere sinteticamente rappresentate con tre o quattro numeri, e il calcolare le intersezioni di valori qualunque con gli insiemi fuzzy, non è computazionalmente gravoso. Solitamente i sistemi fuzzy sono robusti rispetto a piccole variazioni di valori di appartenenza così che le eventuali approssimazioni fatte, definendo insiemi fuzzy triangolari o trapezoidali, sono piccole.

In molte applicazioni, soprattutto nel controllo basato su logica fuzzy, l'intervallo di valori definito da una variabile fuzzy è coperto da un numero di insiemi fuzzy (e dalle corrispondenti funzioni di appartenenza). In alcuni casi, per semplificare il processo di progettazione e per ottenere dal sistema le proprietà desiderate, si adotta uno schema standard per disporre le funzioni di appartenenza. Il più comune consiste in uno schema con insiemi fuzzy equamente distribuiti e con funzioni di appartenenza triangolari che si intersecano l'una con l'altra ad un livello di appartenenza pari a 0.5. Questo schema assicura che ogni valore numerico della variabile corrisponde ad un'interpolazione (al massimo due) con gradi la cui somma sia pari ad uno (*Figura 5.3 (a)*).



(a) **(b)**

Figura 5.3 - Due possibili schemi per disegnare le shape

Un altro schema utilizzato consiste in insiemi fuzzy accumulati attorno ad un punto di particolare interesse (*Figura 5.3(b)*). Ciò da la possibilità di avere una risposta più precisa intorno a questo punto ed un ragionamento meno dettagliato dove il dettaglio non è necessario.

Brian

Brian[2] [6] è un motore a regole che ha il compito di elaborare i dati forniti in ingresso di restituire dopo un processo di manipolazione, dei dati in uscita.

E' stato realizzato all'interno del gruppo *Robocup 2000,* con la precisa funzione di costituire il modulo decisionale dei robot utilizzati per questa competizione ma per le sue caratteristiche di modularità e flessibilità è stato utilizzato anche in altri progetti.

La programmazione ad oggetti rende l'architettura di Brian modulare e facilmente espandibile. Ogni componente del programma è un oggetto e la comunicazione tra gli oggetti avviene tramite lo scambio di altri oggetti.

La struttura del sistema è riportata in *Figura 5.4* dove è stato evidenziato il flusso dei dati all'interno di Brian. Nel riquadro grigio sono inserite le parti utilizzate solo nella fase di costruzione dei diversi moduli. Queste vengono attivate una sola volta durante la creazione degli oggetti e rappresentano dei parser che, leggendo opportuni files di configurazione di ogni modulo, costruiscono le strutture dati che servono per compiere le elaborazioni.

Figura 5.4 - Struttura dei moduli di Brian

La peculiarità di un approccio di questo tipo è l'alta configurabilità, in quanto il cambiamento delle impostazioni di qualsiasi modulo avvengono tramite la modifica dei files di configurazione inoltre, per

[2] BRIAN: Brian Reacts by Inferential ActioNs

scrivere qualunque tipo di comportamento per Brian, è necessario solamente conoscere le grammatiche (del resto semplici perché in notazione prefissa LISP like) dei files senza sapere nulla sul codice che esegue il processo di manipolazione simbolica.
La logica fuzzy è stata scelta ed utilizzata in questo motore a regole per rappresentare l'incertezza di cui è affetto, per motivi di inevitabili errori di misura, il modello del mondo generato dalla telecamera, che ricordiamo essere l'unico sensore di cui disponiamo. Approssimazione e incertezza sono due aspetti che caratterizzano l'imperfezione dei modelli e vengono ben assorbiti dalla logica fuzzy.
Inoltre il controllo è una delle applicazioni più diffuse della logica fuzzy per la semplicità di realizzazione di controllori anche complessi. I vantaggi di questo tipo di controllo sono:

- un controllo non lineare, che permettere di risolvere problemi che i controlli lineari tradizionali non possono affrontare in modo efficace;
- è relativamente facile da sviluppare: un esperto dell'ambiente di applicazione può facilmente definire una base di regole che descriva in termini facili da capire, il proprio comportamento come controllore umano, praticamente usando un linguaggio quasi naturale;
- non richiede nessun modello matematico del sistema da controllare, che può essere soggetto a imperfezioni o approssimazioni non trascurabili;
- agisce in maniera dolce ed è robusto rispetto al rumore e ai disturbi, grazie al particolare insieme di Membership Function selezionato per i dati di ingresso;
- può coprire un ampio numero di variabili di ingresso, implementando diversi comportamenti per ciascuna.

Funzionamento

L'esperto di controllo ad alto livello Brian è un agente classificabile come SXR, che ha il compito manipolare i dati in ingresso tramite un motore a regole, per fornire in uscita dei comandi.
L'esperto viene attivato periodicamente dal kernel Ethnos, uno schedulatore real time sviluppato dall'università di Genova, che ha anche la funzione di implementare un sistema di messaggistica tra i vari agenti e processi. Poiché il controllo del robot è di tipo real time, Brian è soggetto a vincoli sul tempo di esecuzione per lasciare il tempo necessario di esecuzione agli altri processi.
L'acquisizione dati avviene attraverso un parser (*Messenger In*), che permette di raccogliere i dati sensoriali dell'ambiente, ottenuti dalla telecamera. Il parser genera una lista di dati crisp nel formato definito dall'interfaccia di Brian.
Successivamente si passa alla parte di elaborazione, che è divisa in tre parti: traduzione fuzzy, gestione predicati e motore a regole, come si può vedere nello schema funzionale mostrato in *Figura 5.5*.
I dati in ingresso vengono tradotti in logica fuzzy e passati alla parte di gestione dei predicati utilizzando una lista di dati fuzzy. I dati fuzzy vengono ripartizionati e uniti attraverso i predicati, per creare informazioni più precise e sintetiche. Vengono poi create le condizioni di possibilità (*cando*) e di

desiderabilità (*want*) basate sui predicati. Le *cando* indicano le abilitazioni ai comportamenti che possono essere attivati a seconda dei dati sensoriali aggiornati ad ogni esecuzione, mentre le *want* rappresentano i pesi e la priorità per i comportamenti che si vorrebbero eseguire nella situazione in cui si trova il robot.

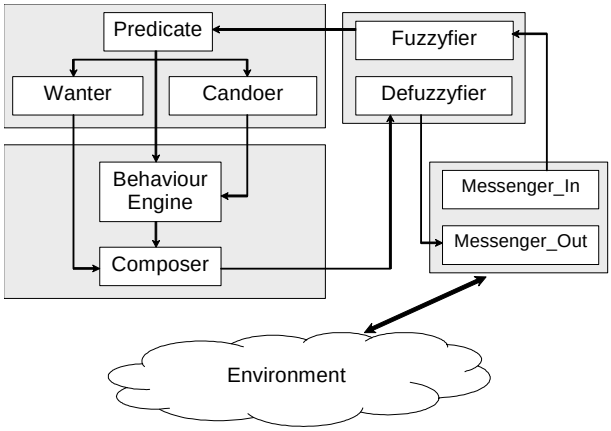


Figura 5.5 - Schema funzionale

Queste condizioni sono passate al vero e proprio motore a regole attraverso le rispettive liste dinamiche. A questo punto vengono attivati i comportamenti abilitati, che generano una lista di azioni da compiere. Queste vengono passate al modulo *composer* che esegue un filtraggio delle azioni da eseguire con le condizioni di desiderabilità e successivamente una fusione delle azioni ridondanti, in quanto diversi comportamenti possono attivare le stesse azioni, fino ad ottenere una lista di azioni da applicare agli attuatori del robot. Al momento questa lista è realizzata ancora secondo la logica fuzzy; viene così passata alla parte che la trasforma in azioni di tipo crisp (*defuzzyfier*), cioè dei veri e propri comandi da mandare ai controllori di basso livello. Quest'ultima parte di trasmissione è gestita dal modulo *Messenger out* che ha il compito di creare messaggi di tipo Ethnos per i vari esperti

Fuzzyficazione – Defuzzyficazione

I moduli di fuzzyfcazione e defuzzyficazione hanno il compito, come mostrato in Figura 5.5, di interfacciare il motore inferenziale vero e proprio dal punto di vista della rappresentazione simbolica del mondo. Infatti questi moduli eseguono le funzioni di:

- prendere i valori crisp ricevuti dal messanger-in e convertirli in valori fuzzy sui quali poi i vari moduli a valle eseguiranno il loro processo di elaborazione simbolica;
- prendere i valori fuzzy uscenti dal processo inferenziale e corrispondenti ai comandi da dare al robot e convertirli in valori crisp utilizzati poi per regolare le velocità dei motori.

Alla base dell'implementazione dei moduli fuzzyficatore e defuzzyficatore c'è l'idea di definire una relazione tra la rappresentazione dei dati nel mondo in crisp (basata sui valori delle misurazioni effettuate dai sensori) ed nel mondo fuzzy (basata sull'appartenenza degli stessi a determinati insiemi).

Fuzzy_sets

Il fuzzy-set è l'oggetto che effettivamente esegue la conversione da dato crisp a dato fuzzy. Esso è rappresentato da un intervallo continuo di valori al quale un qualunque dato crisp o fuzzy può appartenere

in maniera più o meno vera, ovvero con un valore detto membership function appartenente all'intervallo [0,1].

La macro classe dei fuzzy sets è stata specializzata in diverse microclassi, mostrate anche in *Figura 5.6*:

- *triangle*, semplice triangolo che copre un intervallo di valori finito, *divided triangle* triangolo separato in due parti, *trapetium*, classico trapezio generalmente isoscele, *rectangle*, rettangolo e in fine il *singleton* che sarebbe un segmento verticale o impulso. Questi fuzzy sets coprono un intervallo di valori finito.
- *triangle_ol*, triangolo aperto a sinistra, *triangle_or*, triangolo aperto a destra. Questi fuzzy sets coprono un intervallo di valori infinito.

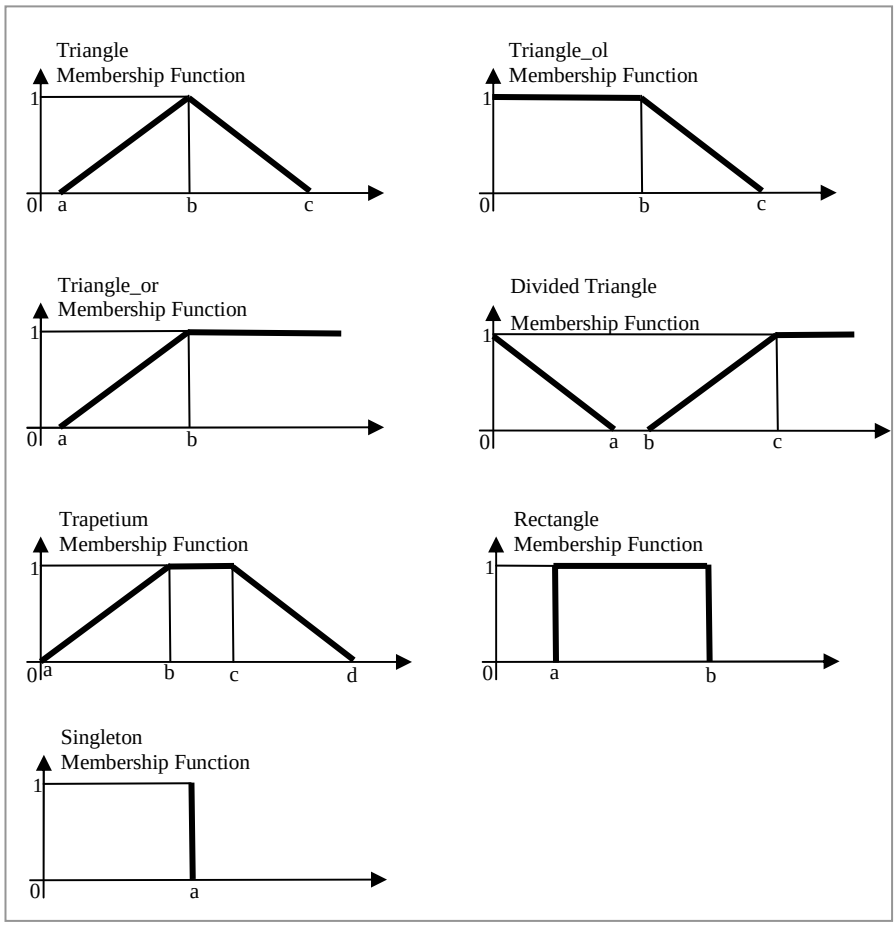


Figura 5.6 - Fuzzy Sets

Shape e Shape list

Le shape rappresentano gli insiemi di fuzzy set utilizzati per coprire tutto l'intervallo di valori che un determinato tipo di dato (crisp o fuzzy) può assumere (*Figura 5.7*).

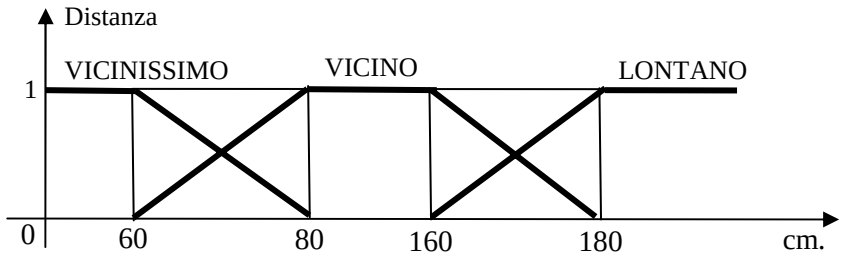


Figura 5.7 - Esempio di shape usata per fuzzyficare una distanza

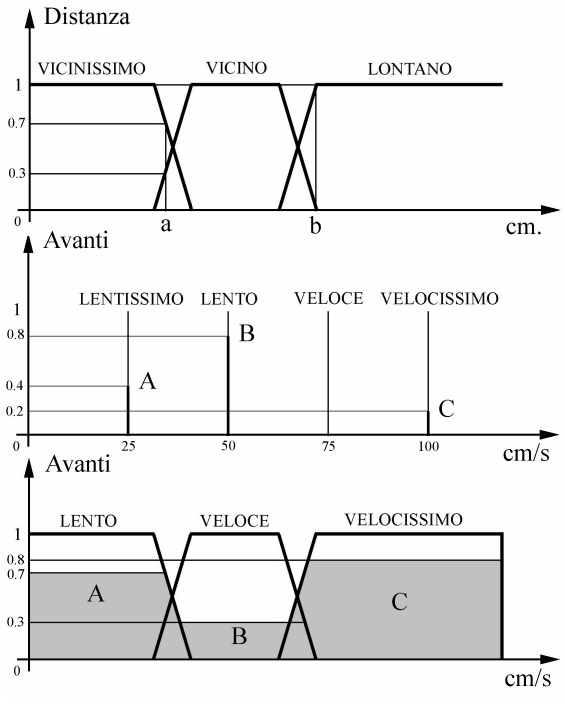
La shape list e la lista contenete tutte le shape utilizzate dal modulo per eseguire le trasformazioni da crisp a fuzzy e viceversa.

Associaton e Association list

L'association è l'oggetto che instaura la corrispondenza tra un dato (fuzzy o crisp) e la relativa shape per la sua fuzzyficazione o defuzzyficazione.
L'association list è la lista che contiene tutte le associazioni tra crisp e fuzzy e viceversa.

Fuzzyficatore

Il fuzzyficatore è l'oggetto che, mettendo insieme tutte le strutture dati precedenti descritte, si occupa, data una lista di dati crisp, di effettuare la conversione di questi ultimi in fuzzy.
Per rendere più semplice la comprensione di questo modulo viene illustrato con un piccolo esempio, in *Figura 5.8*, il concetto di fuzzyficazione e defuzzyficazione.



I dati **a** e **b** appartengono agli insiemi:

	VICINISSIMO	VELOCE	LONTANO
a	0.7	0.3	0
b	0	0	1

$$\text{Avanti} = \frac{0.4\times 25 + 0.8\times 50 + 0\times 75 + 0.2\times 100}{0.4 + 0.8 + 0 + 0.2}$$

Il valore di Avanti è dato dal baricentro dell'area evidenziata.

Figura 5.8 - Fuzzyficatore e Defuzzyficatore con centroide e a baricentro d'area

Il procedimento è il seguente:

- a start-up il fuzzyficatore carica da due file la lista delle associazioni tra dati crisp e shapes e la lista di shapes secondo cui effettuare la conversione.
- si fuzzyficano i dati in ingresso, cioè si trasformano i dati crisp in valori di appartenenza agli insiemi fuzzy corrispondenti. Per ogni dato di ingresso il fuzzyficatore cerca incrociando association_list e shape_list la shape corrispondente al dato in questione dopodiché lo fa passare attraverso tutti i fuzzy-set contenuti nella shape i quali ritornano il valore di appartenenza del dato all'insieme da loro rappresentato (vedi Figura 5.8).
- Il fuzzyficatore costruisce così una lista di dati fuzzy che poi passerà al modulo successivo.

Defuzzyficatore

Il defuzzyficatore è l'oggetto duale del fuzzyficatore. Data una lista di dati fuzzy si occupa di effettuare la conversione di questi ultimi in crisp.

Il procedimento è il seguente:

- come il fuzzyficatore a start-up il defuzzyficatore carica da due file la lista delle associazioni tra dati fuzzy e shapes e la lista di shapes secondo cui effettuare la conversione.
- si defuzzyficano i dati in uscita in modo da ottenere dati crisp, utilizzando due tecniche.
    1. Metodo del singleton o del centroide: una volta trovata la shape di defuzzyficazione con lo stesso metodo del fuzzyficatore il valore del dato crisp corrispondente viene ottenuto come baricentro dei singleton della shape pesati nel loro valore per il corrispondente valore del dato fuzzy (vedi Figura 5.8).
    2. Metodo del baricentro d'area il valore del dato crisp corrispondente viene ottenuto come baricentro della figura risultante dopo aver ridimensionato tutti i fuzzy set con il corrispondente valore del dato fuzzy (vedi Figura 5.8).

Preacher

Questo modulo costruisce e valuta i predicati fuzzy, cioè delle strutture che assumono valori da zero a uno fondendo dati fuzzy differenti con lo scopo di una maggiore sinteticità e minore ridondanza.

Durante la fase di start-up genera un oggetto parser, che gli permette di conoscere la composizione dei predicati e realizza una struttura ad albero al più binario per la sua rappresentazione. Mentre, durante la fase runtime, esegue una valutazione dei predicati, leggendo da una lista di dati fuzzy i valori delle *membership function* e delle loro affidabilità (reliability) e associandoli attraverso gli operatori *and or* e *not* per calcolare il valore del predicato e della sua affidabilità. Ad esempio un predicato può essere:

GoalAvanti = (OR (D AngoloGoal AVANTI1) (D AngoloGoal AVANTI2))

Candoer

Questo modulo costruisce e valuta le condizioni di *cando*, cioè delle strutture che assumono valori da zero a uno fondendo predicati fuzzy differenti con lo scopo abilitare solo determinati comportamenti, che hanno senso nella situazione attuale in cui si trova il robot, ed evitando l'attivazione di comportamenti insensati o inopportuni. Durante la fase di startup, genera un oggetto parser, che gli permette di conoscere la composizione delle condizioni cando e realizza una struttura ad albero al più binario per la sua rappresentazione, simile a quella dei predicati. Mentre durante la fase runtime esegue una valutazione delle condizioni, leggendo da una lista di predicati fuzzy il loro valore e le loro affidabilità (reliability), associandoli attraverso gli operatori *and or* e *not* per calcolare il valore della cando e della sua affidabilità. Ad esempio una condizione cando può essere:

GotoBall = (AND (P Auto) (P VedoPalla))

Behaviour Engine

Rappresenta il motore dei comportamenti. La sua funzione, ricevuti in ingresso i predicati e le cando, è quello di eseguire i comportamenti adeguati e di passare le azioni che propongono alla fase successiva di composizione.

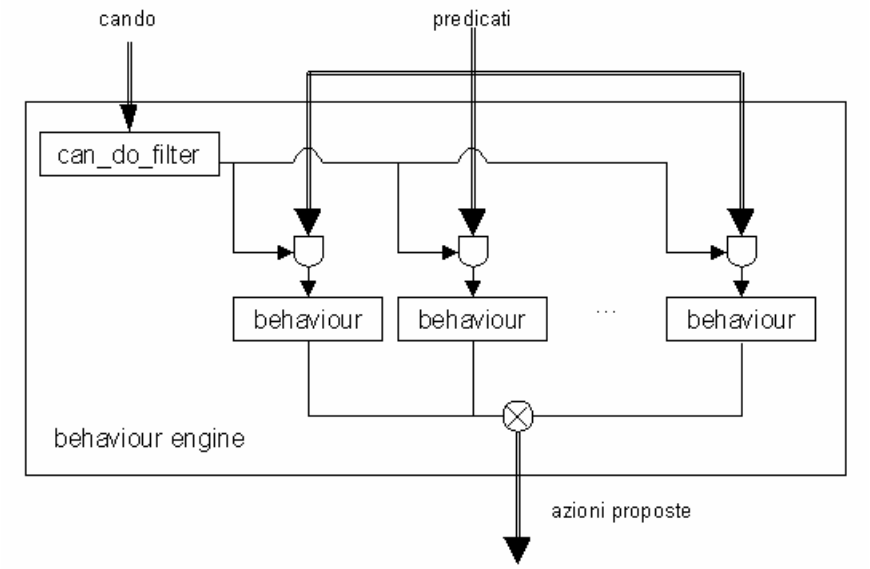


Figura 5.9 - Behaviour Engine

Il funzionamento è brevemente descritto dal disegno in *Figura 5.9*. La prima fase è relativa al filtraggio delle cando, di questo si occupa la classe *cando_filter*.

La classe *behaviour_engine* possiede al suo interno la lista completa, caricata a start-up, dei comportamenti, e la itera, seguendo le indicazioni del *cando_filter*, passando di volta in volta la lista dei predicati, raccogliendo le azioni proposte, e fondendole insieme in uscita.

I comportamenti sono implementati in apposite classi derivate da *behaviour*, astratta, potendo definire il tipo di comportamento secondo i propri bisogni. Per ora è stata implementata solo la classe *rules_behaviour* che descrive comportamenti formati da regole del tipo

Precondizione => Azione

La fase di start-up è affidata alla classe *behaviour_parser*, la quale si preoccupa di leggere da file la lista dei comportamenti e dei relativi file di configurazione, creare gli oggetti adeguati e inserirli nell'elenco che verrà utilizzato da *behaviour_engine.*

Composer

Il modulo si occupa della composizione, tra loro e con i pesi delle *want*, delle azioni proposte dai comportamenti prima di passare alla defuzzyficazione.

In *Figura 5.10* è mostrato come avviene il processo. Si può notare che in ingresso ci possono essere più azioni con lo stesso nome e etichetta, proposte da comportamenti diversi o addirittura uguali. Il primo passo è quello di modularle a seconda delle want relative a chi le ha proposte utilizzando la classe *weight_composer*. Tale modulazione può avvenire in diversi modi, per esempio moltiplicando i valori della want e dell'azione oppure prendendo il minimo o il massimo tra want e azione.

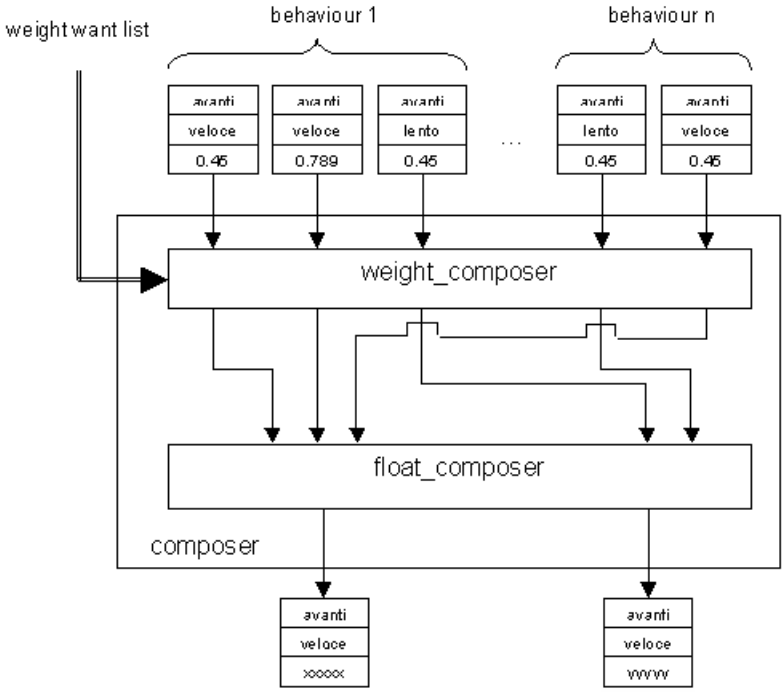


Figura 5.10 - Composer

Eseguita la prima composizione, che modifica solamente i valori delle azioni, si prosegue a raggrupparle, anche se proposte anche da comportamenti diversi, in gruppi con nome ed etichetta identici. I loro valori vengono fusi tramite la classe *float_composer*, per la cui implementazione valgono le stesse considerazioni espresse in precedenza per *weight_composer*.

I comportamenti di Rabbiati

Sono stati sviluppati quattro comportamenti: *Align*, *GoalKeeping*, *Kick* e *GotoHome*. Il comportamento di Align si occupa di mantenere il robot allineato su una sorta di binario immaginario posto ad una distanza predefinita dalla linea di porta. Questo gli consente di muoversi davanti alla porta rimanendo in una sorta di "zona di parata". Il comportamento di GoalKeeping consente al robot di parare seguendo i movimenti della palla. Nel comportamento di Kick sono gestite le funzionalità di calcio della palla. Il comportamento di GotoHome consente al robot di tornare in porta da qualsiasi punto del campo. Nel seguito li vediamo uno per uno in dettaglio.

Align

Il comportamento di Align è fondamentale perché il robot rimanga nella posizione corretta durante il gioco. Esso sfrutta l'autolocalizzazione del robot e si occupa di mantenerlo allineato davanti alla porta in un "binario di parata" lungo il quale il robot si muove. Il comportamento di Align mantiene quindi il robot a una distanza fissa dalla porta rivolto verso la porta avversaria.

Per poter ottenere un comportamento del genere, è stata generata una shape per fuzzyficare la posizione del robot nel campo lungo l'asse delle ascisse del sistema di riferimento iniziale. Come possiamo notare in *Figura 5.11*, l'origine di questo sistema di riferimento è fissata a centro campo con l'asse delle x rivolto verso la porta avversaria.

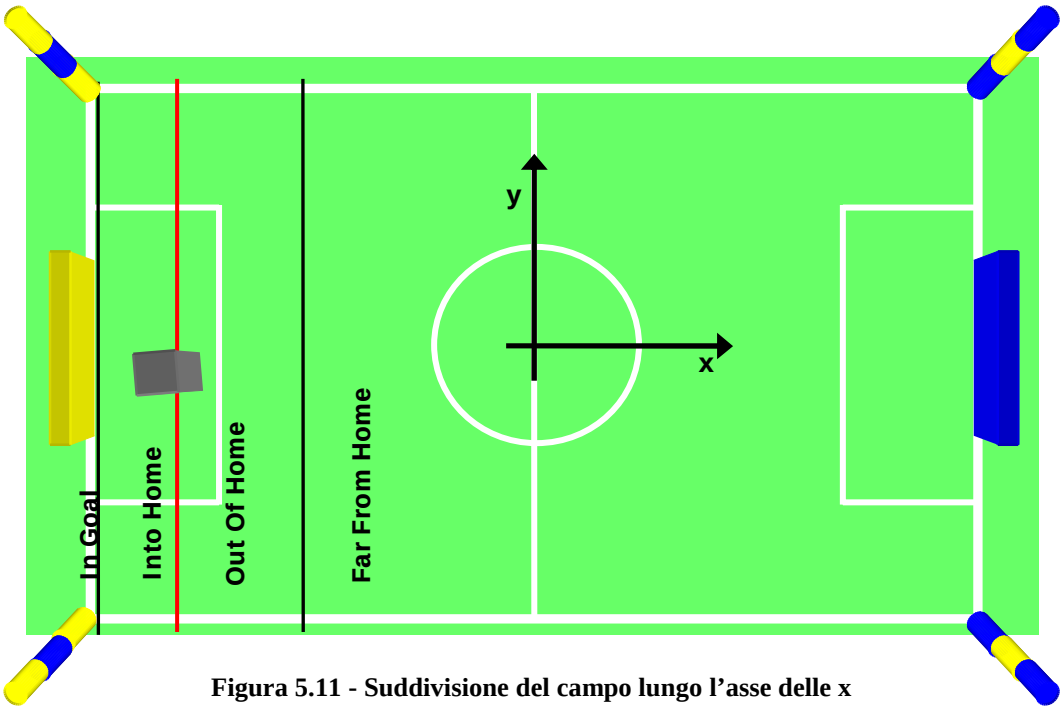


Figura 5.11 - Suddivisione del campo lungo l'asse delle x

Nella Figura 5.11 sono anche evidenziate le zone in cui è stato diviso il campo per poter ottenere il comportamento di Align. Naturalmente i confini delle zone non sono così netti ma sono divisi secondo la shape ROBOT_X_COORD rappresentata in *Figura 5.12.*

Ricordiamo che le shape sviluppate in questo contesto sono compatibili sia con un campo di dimensioni 10 x 5 metri, sia con un campo di dimensioni 10 x 7 metri.

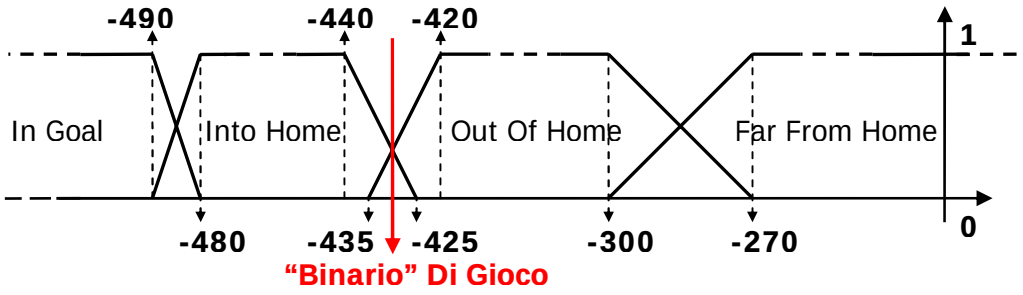


Figura 5.12 - Shape ROBOT_X_COORD

La linea Rossa nelle Figure 5.11 e 5.12 indica il binario lungo il quale il robot si muove. Quando il robot si trova nella zona 'Out Of Home', a seconda che si stia muovendo verso destra o verso sinistra, subirà una rotazione rispettivamente verso destra o verso sinistra, in modo che il suo stesso movimento tenda a riportarlo verso la linea rossa. Quando il robot si trova nella zona 'Into Home' a seconda che si stia muovendo verso destra o verso sinistra, subirà una rotazione rispettivamente verso sinistra o verso destra in modo da ottenere lo stesso risultato. Quando il robot si trova nella zona 'In Goal' significa che la sua posizione è troppo arretrata e rischia di urtare i pali della porta. Le rotazioni impresse al suo movimento hanno la stessa direzione di quelle del caso in cui si trovasse nella zona 'Into Home', ma hanno una maggiore intensità per permettere al robot di uscire più agevolmente da questa zona.

Nella zona 'Far From Home' il robot si trova troppo lontano dalla porta per tentare un qualsiasi tipo di allineamento e quindi in questo comportamento non viene fatto nulla. In seguito verrà mostrato il comportamento del robot in questa situazione.

La rotazione impressa al robot dipende anche dal suo allineamento istantaneo: se è molto inclinato tenderà a raddrizzarsi più velocemente.

Il grado di allineamento del robot è definito dalla shape ROBOT_ORIENTATION rappresentata in *Figura 5.13 (a) e (b).*

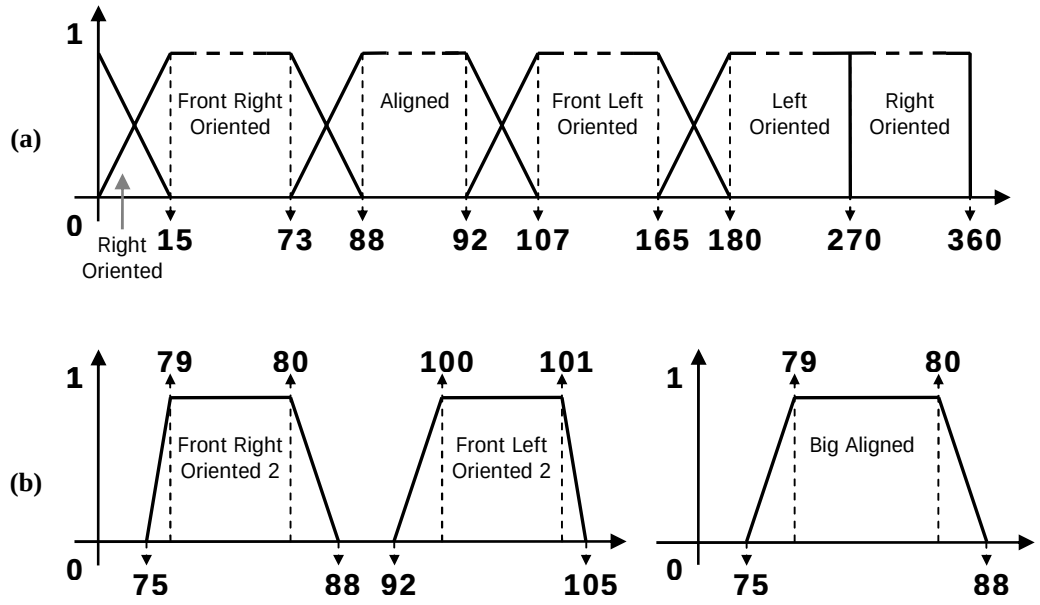


Figura 5.13 - Shape ROBOT_ORIENTATION

Come possiamo vedere sono definiti diversi fuzzy set che vengono presi in considerazione a seconda della situazione. Ricordiamo che il fronte di Rabbiati è diretto verso la sua sinistra, che è il verso positivo di avanzamento del robot. Per questo motivo il robot è allineato alla porta, cioè è orientato in modo da avere la propria porta esattamente alle spalle, quando ha un orientamento di 90° rispetto all'asse delle ascisse del sistema di riferimento iniziale.

Una caratteristica particolare di questo comportamento è che genera solamente attuazioni di rotazione. Il robot non torna sul suo binario immaginario di gioco solo con questo comportamento ma grazie ad esso modifica le attuazioni generate dagli altri comportamenti in modo da ottenere il risultato voluto. In questo modo Rabbiati tende naturalmente a ritornare sulla linea di parata senza la possibilità ci siano attuazioni di spostamento tangenziale contrastanti che potrebbero causare un rallentamento del suo movimento.

Quando il robot è fermo (Tanspeed STEADY) in qualunque posizione nelle zone 'Into Home' o 'Out Of Home' l'Align riallinea comunque il robot anche fuori dal binario di parata. Si è visto, infatti, che al momento di effettuare una parata la posizione migliore di partenza è quella allineata con la porta, anche se in quel momento il robot si dovesse trovare leggermente avanzato o arretrato rispetto al binario di parata. Al primo spostamento comunque esso tenderà a ritornare in posizione.

Goal Keeping

Il comportamento di Goal Keeping si occupa della funzione di parata di Rabbiati. Questo comportamento si basa su diversi parametri: la posizione del robot nel campo proveniente dall'autolocalizzazione, la

distanza della palla, l'angolo della palla rispetto al robot, e la differenza tra l'angolo della porta e l'angolo della palla rispetto al robot. Vediamo ora in dettaglio cosa sono questi parametri e la loro fuzzyficazione.

È stata generata una shape per fuzzyficare la posizione del robot nel campo lungo l'asse delle ordinate del sistema di riferimento iniziale. Nella *Figura 5.14* sono evidenziate le zone in cui è stato diviso il campo per poter ottenere il comportamento di Goal Keeping.

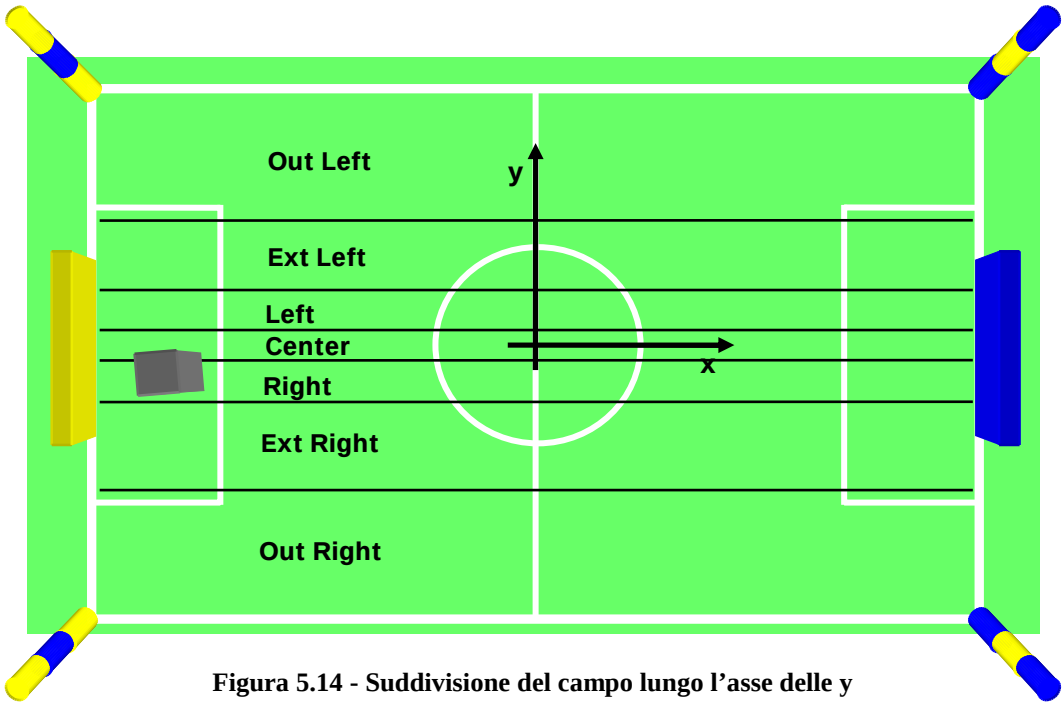


Figura 5.14 - Suddivisione del campo lungo l'asse delle y

Naturalmente i confini delle zone non sono così netti ma sono divisi secondo la shape ROBOT_Y_COORD rappresentata in *Figura 5.15*.

Ricordiamo che le shape sviluppate in questo contesto sono compatibili sia con un campo di dimensioni 10 x 5 metri, sia con un campo di dimensioni 10 x 7 metri.

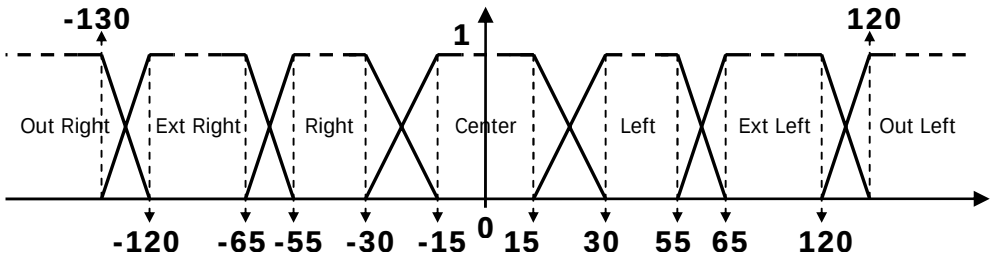


Figura 5.15 - Shape ROBOT_Y_COORD

Grazie a questa shape il robot conosce in ogni istante la sua posizione davanti alla porta. Nel caso in cui, infatti, inseguendo la palla durante una azione di gioco si dovesse venire a trovare nelle zone 'Ext Right' o 'Ext Left', la sua corsa verrebbe interrotta per evitare di uscire dallo specchio della porta. La sua

posizione quindi sarebbe a ridosso del palo: la posizione ideale per coprire la porta quando la palla si trova nei pressi di uno dei corner.

Per poter parare, il robot prende in considerazione anche la distanza della palla. In *Figura 5.16* possiamo vedere la shape utilizzata per fuzzyficare questo dato.

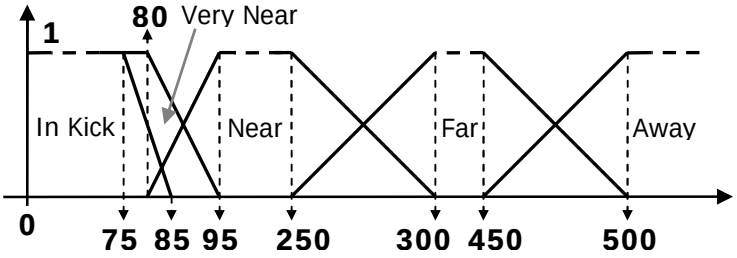


Figura 5.16 - Shape BALL_DISTANCE

La reattività di Rabbiati, infatti, varia a seconda della distanza della palla: quando la palla è molto lontana ('Away') questo comportamento non genera nessuna azione, se la palla è lontana ('Far') il robot si limita a piazzarsi per poter coprire più specchio della porta possibile e i suoi movimenti si fanno più reattivi man mano che la palla si avvicina ('Near') fino a raggiunge la reattività massima andando sulla palla per calciarla quando è vicinissima ('Very Near'). Quando la palla è in Kick si trova alla distanza di calcio.

C'è una significativa differenza nel modo di interporsi tra la palla e la porta tra quando la palla è 'Very Near' e quando si trova più lontana: quando la palla si trova nella zona 'Very Near' il robot utilizza l'angolazione della palla rispetto a se stesso reagendo in modo differente a seconda che questa si trovi in posizione frontale o più o meno laterale rispetto ad esso, in modo da poterla calciare o deviare lateralmente alla porta.

In *Figura 5.17* sono mostrate le possibili posizioni angolari che può assumere la palla rispetto la robot.

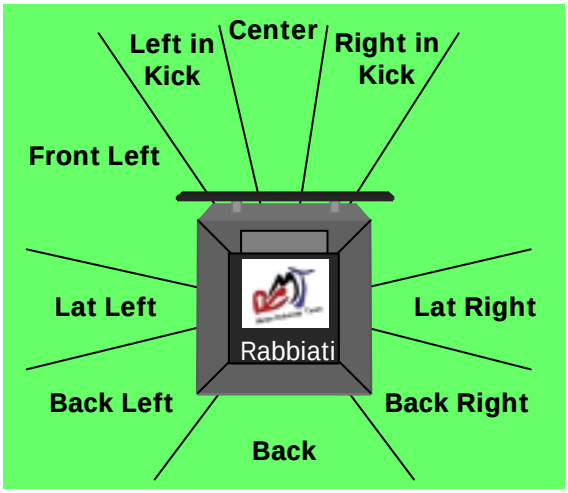


Figura 5.17 - Shape BALL_ANGLE

Quando la palla ritorva nella zona 'Center' il robot sta fermo, pronto a calciare. Quando la palla si trova nelle zone 'Right in Kick' e 'Left in Kick' il robot si muove lateralmente a bassa velocità rispettivamente verso destra e verso sinistra per riallinearsi con la palla. Quando la palla si trova nelle zone 'Front Right' e 'Front Left', questo movimento diventa estremamente rapido per potersi porre davanti alla palla, e successivamente calciare, o per deviarla. Quando la palla viene a trovarsi nelle zone 'Lat Right' e 'Lat Left', l'unica possibilità è spingerla verso l'esterno della porta muovendosi alla massima velocità verso di essa. In effetti questa è la migliore azione possibile nel breve tempo a disposizione. Se si cercasse di

arretrare per riallinearsi frontalmente alla palla, si perderebbe troppo tempo rischiando inoltre un autogol. Quando la palla arriva nelle zone ‘Back Left’ e ‘Back Right’ il robot si muove alla massima velocità verso di essa per tentare un ultimo “colpo di reni” e deviarla in fallo laterale. Nel caso particolare in cui la palla superasse il portiere e si venisse a trovare nella zona ‘Back’ senza essere entrata in porta, il robot si limita a coprirla in modo che nessun attaccante la raggiunga. Ogni suo movimento infatti potrebbe causare un autogol.

Quando la palla, contrariamente al caso precedente, non è prossima la robot (cioè si trova nelle zone ‘Near’ o ‘Far’ della shape BALL_DISTANCE), esso prende in considerazione per posizionarsi anche la posizione della porta. Ricordiamo che la posizione della porta è da intendersi come la posizione del *centro della porta*. Il parametro necessario per questo scopo è il *Ball-Home Angle.* Questo dato è calcolato come la differenza tra la posizione angolare della palla e la posizione angolare della porta rispetto al robot in direzione antioraria, come mostra la *Figura 5.18*.
Utilizzando questo parametro Rabbiati tende continuamente a posizionarsi sulla retta che congiunge la palla e il centro della porta, questa retta è rappresentata in Figura 5.18 dalla linea rossa tratteggiata.

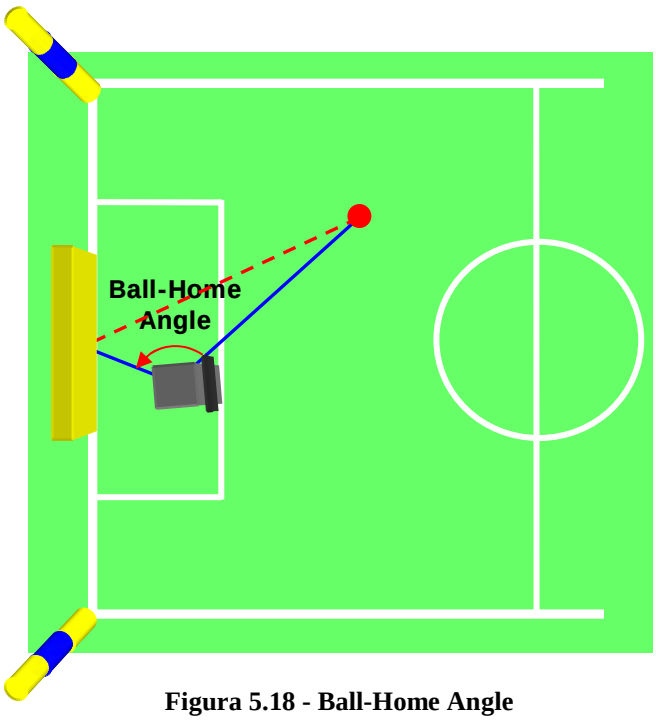


Figura 5.18 - Ball-Home Angle

Come si può notare in *Figura 5.19* questo posizionamento è quello che consente la maggior copertura dello specchio della porta. Quando il robot è allineato tra la palla e la porta il Ball-Home Angle è di 180°.

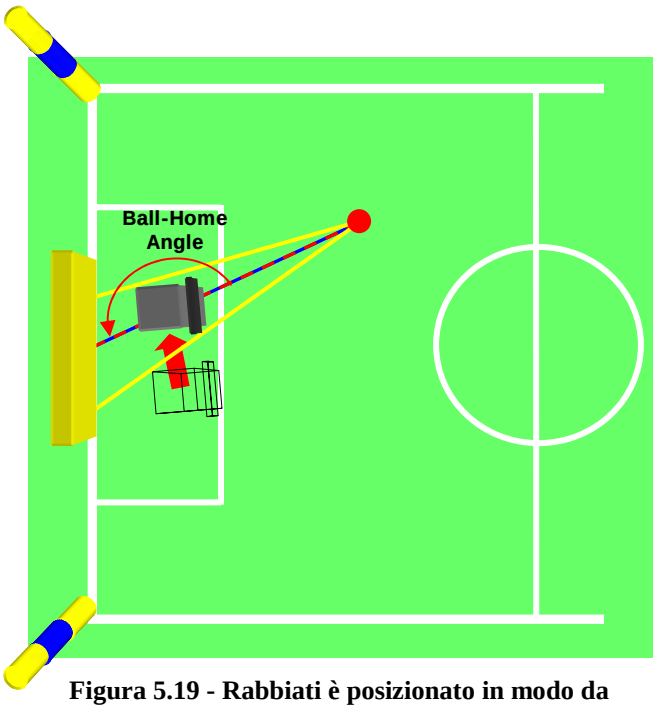


Figura 5.19 - Rabbiati è posizionato in modo da ottenere la massima copertura dello specchio della porta

Il Ball-Home Angle viene fuzzyficato tenendo in considerazione la distanza tra il robot e la retta che interseca la palla e il centro della porta. Come abbiamo detto se il robot è vicino alla congiungente palla-porta il Ball-Home Angle sarà prossimo a 180° mentre, se il robot si trova più lontano da essa, l'angolo risulterà molto maggiore di 180°, se Rabbiati si trova alla sinistra della congiungente palla-porta, o molto minore di 180°, se si trova alla destra di essa. Di seguito è riportata la dichiarazione della shape ANGLE_DIFFERENCE riguardante il Ball-Home Angle:

```
(ANGLE_DIFFERENCE
(TRA (BIG_LEFT 20 30 150 155))

(TRA (LEFT_NEAR 150 155 167 172))
(TRA (OPPOSITE_NEAR  167 172 188 193))
(TRA (RIGHT_NEAR 188 193 205 210))

(TRA (LEFT_FAR 150 155 170 175))
(TRA (OPPOSITE_FAR  170 175 185 190))
(TRA (RIGHT_FAR 185 190 205 210))

(TRA (BIG_RIGHT  205 210 330 340))

(TRA (ALIGNBACK 0 0 35 45))
(TRA (ALIGNBACK 315 325 360 360))
)
```

La velocità con cui il robot si riallinea tra la palla e la porta, è funzione della distanza che lo divide dalla congiungente palla-porta. Maggiore è questa distanza, maggiore sarà la sua velocità. Un altro dato in

grado di determinare la velocità degli spostamenti è la distanza della palla: se la palla si trova nella zona 'Near', il robot si muoverà più velocemente mentre se si trova nella zona 'Far', si muoverà più lentamente. Visto la distanza della palla in questo ultimo caso, infatti, uno spostamento minore è sufficiente a garantire la copertura della porta.

Kick

Questo comportamento si occupa delle funzionalità di calcio (*Kick*) del robot. Rabbiati monta un sistema di calcio pneumatico ad aria compressa formato da due pistoni, due elettrovalvole e due bombole. Il circuito pneumatico unisce fra di loro le bombole in modo che la pressione resti identica in entrambe. Due elettrovalvole comandano indipendentemente i due pistoni e sono attivate dalla scheda di controllo di basso livello (che vedremo nei capitoli successivi).

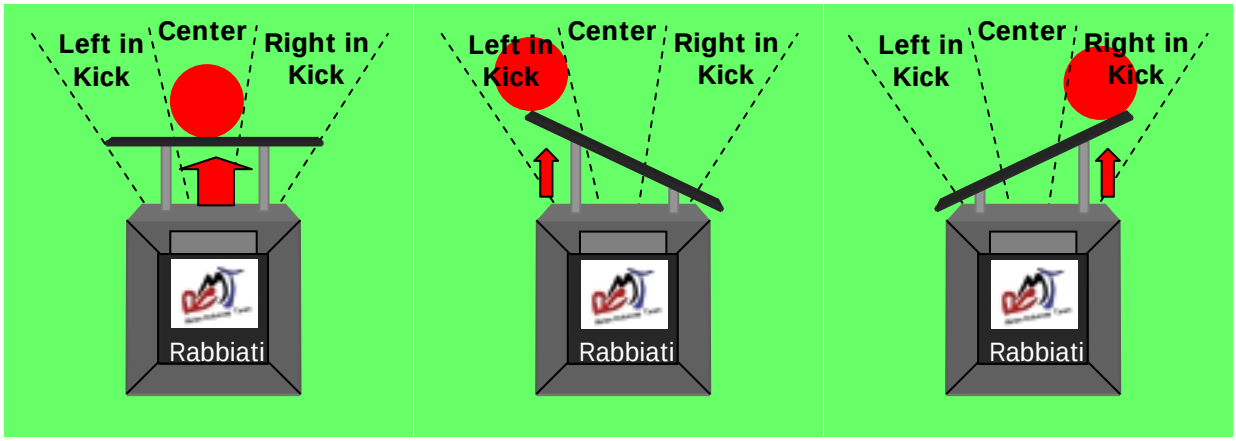


Figura 5.20 - Kick

Come possiamo vedere nella *Figura 5.20*, Rabbiati è quindi in grado di calciare con una solo o entrambi i pistoni di cui è dotato. Sfruttando la shape BALL_ANGLE già vista in precedenza, il robot è in grado di determinare in che posizione si trova la palla al momento del calcio. Quando la palla è centrale la "kickkata" avviene utilizzando entrambi i pistoni, usufruendo quindi di una maggiore potenza di calcio. Quando la palla si trova angolata nelle due aree denominate 'Right in Kick' e 'Left in Kick', il calcio avviene con un solo pistone. In questo modo la palla viene respinta in una direzione diversa da quella da cui è arrivata, evitando così di ripassare la palla all'avversario che l'ha calciata.

La "kickkata" è attivata anche in funzione della distanza della palla. Utilizzando la shape BALL_DISTANCE, gia vista in precedenza, si attiva il calcio quando la palla arriva nella zona 'In Kick' (vedi Figura 5.16). Questa distanza è leggermente sovradimensionata rispetto alla massima estensione del kicker per compensare il ritardo tra percezione della palla e attuazione del comando di kick, in modo che, quando la palla si muove verso il robot, essa possa essere colpita al momento giusto.

Goto Home

Il comportamento di Goto Home si occupa di riportare Rabbiati in porta da qualunque zona del campo si trovi. I motivi per cui il robot possa venirsi a trovare fuori dalla sua posizione di gioco sono diversi e molti di essi ancora ignoti e imprevedibili. Un qualche genere di malfunzionamento è di solito la causa di questo fenomeno, ad esempio: un momentaneo malfunzionamento del sistema di autolocalizzazione dovuto ad una improvvisa variazione della luminosità ambientale, che renderebbe il robot momentaneamente cieco, potrebbe ingannare il robot convincendolo di essere in una posizione diversa da quella reale. In questo caso Rabbiati comincerebbe ad agire convinto di essere da un'altra parte e molto probabilmente questo lo porterebbe fuori dalla sua posizione di gioco. Questo genere di problema si è manifestato spesso durante il Festival della Scienza svoltosi a Genova nell'ottobre-novembre 2003 al quale il robot ha partecipato. La manifestazione si è tenuta all'aperto esponendo così il campo a continue e casuali variazioni di illuminazione dovute allo spostamento del sole durante le ore del giorno e alla nuvolosità variabile. Questa situazione ambientale non è prevista nelle normali condizioni di gioco.
Questo comportamento è invece volutamente sfruttato per eseguire l'entrata in campo e il posizionamento autonomo del robot. Da una posizione laterale al campo il robot viene attivato e autonomamente si posiziona nella sua porta. Questo comportamento viene premiato in una competizione ufficiale di RoboCup. Il comportamento di Goto Home utilizza come parametri in ingresso la posizione del robot fornita dall'autolocalizzazione, la distanza dalla propria porta e la sua angolazione rispetto al robot.
Per quanto riguarda l'autolocalizzazione è stata necessaria una fuzzyficazione aggiuntiva delle coordinate x,y del robot: sono stati aggiunti due nuovi fuzzy set alla shape ROBOT_Y_COORD e un nuovo fuzzy set alla shape ROBOT_X_COORD in modo da definire tre nuove zone del campo: Left Field, Right Field e In Area. Queste Nuove zone sono mostrate in *Figura 5.21*.

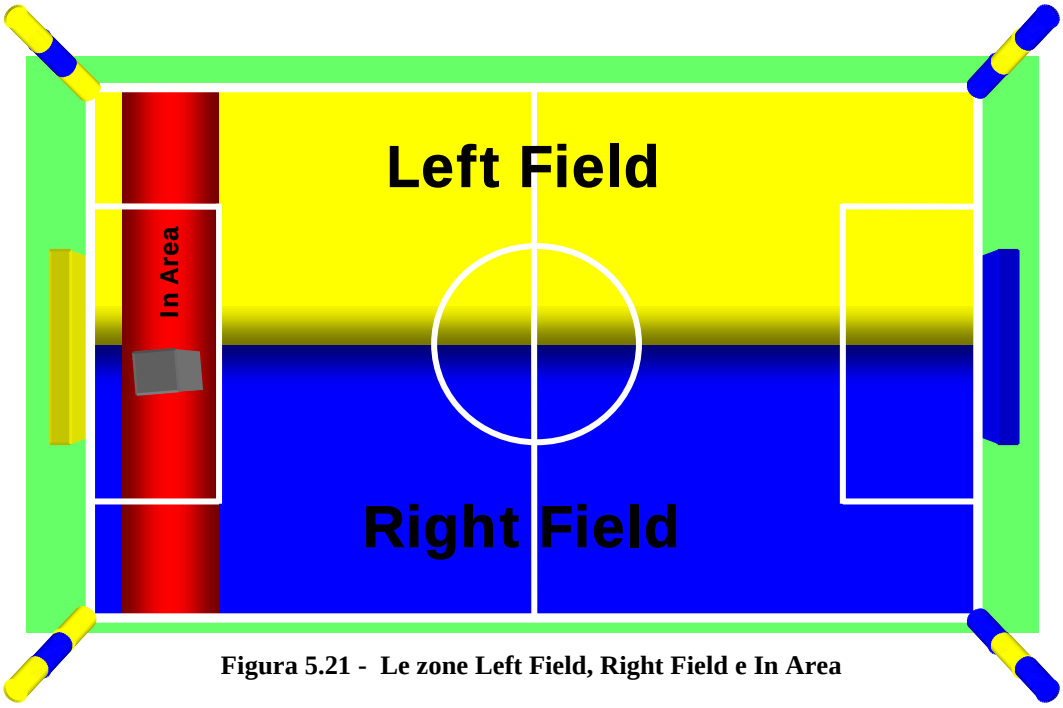


Figura 5.21 - Le zone Left Field, Right Field e In Area

In *Figura 5.22 (a) e (b)* sono mostrati i due nuovi Fuzzy Set.

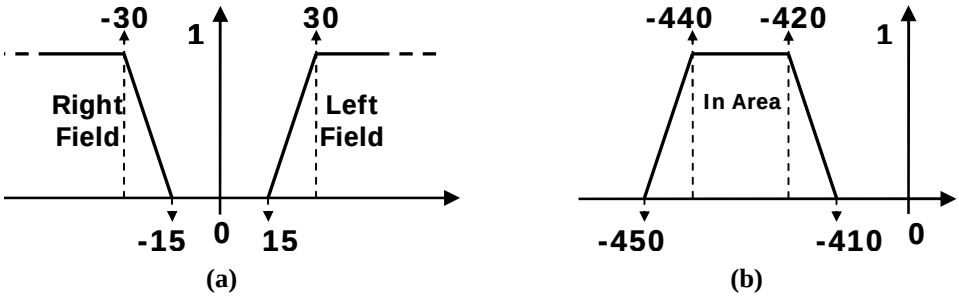


Figura 5. 22 – (a) Fuzzy Set aggiunto alla shape ROBOT_Y_COORD; (b) Fuzzy Set aggiunto alla shape ROBOT_X_COORD

Un altro parametro utilizzato è la posizione della porta rispetto al robot definito come distanza della porta e angolazione della porta. Per distanza dalla porta si intende la distanza dal suo centro. Trattandosi di una distanza da un punto, la sua fuzzyficazione appare come un insieme di zone semi-circolari concentriche che racchiudono dei tratti di campo posti a distanza costante dal centro della porta, come mostrato in *Figura 5.23*

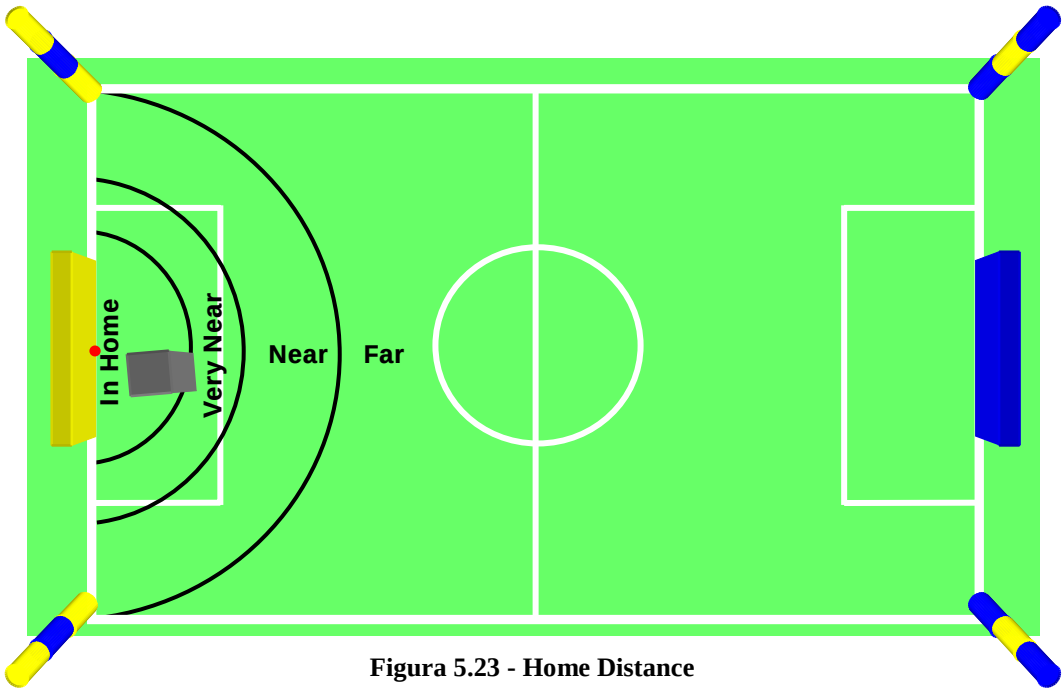


Figura 5.23 - Home Distance

Naturalmente, come nei casi precedenti, la divisione di queste zone non è netta, ma avviene secondo la shape HOME_DISTANCE mostrata in *Figura 5.24.*

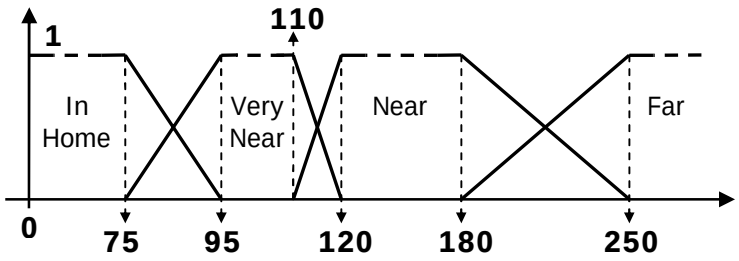


Figura 5.24 - Shape HOME_DISTANCE

L'ultimo parametro utilizzato da questo comportamento è l'angolazione del robot rispetto alla porta. La sua shape è illustrata in *Figura 5.25*.

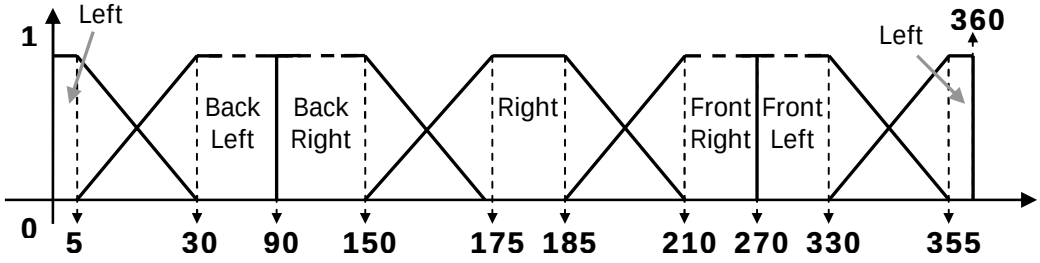


Figura 5.25 - Shape HOME_ANGLE

A seconda della zona del campo in cui si trova, 'Left Field', 'Right Field' o 'Center' ed alla propria posizione angolare, il robot ruota a in modo da allineare la porta alla propria destra o alla propria sinistra secondo la shape HOME_ANGLE. Ricordiamo che la particolare cinematica "da portiere" di Rabbiati gli consente solamente spostamenti laterali a destra o a sinistra.

Una volta allineato alla porta si dirige verso di essa ad una velocità che dipende dalla distanza dalla porta secondo la shape HOME_DISTANCE. La velocità è massima se la distanza della porta è 'Far' e va via via diminuendo man mano che il robot si avvicina ad essa.

Quando il robot si trova lontano dalla porta, la sua funzionalità di parata è disabilitata.

Tornando in porta Rabbiati potrebbe colpire la palla accidentalmente e causare un autogol. Per evitare questo sono state previste in questo comportamento un insieme di regole atte ad aggirare la palla nel caso questa si trovi sulla via di casa. In questo modo il robot supera la palla quasi ignorandola per rimettersi a parare nel momento in cui si è riavvicinato alla porta. Per ottenere questo comportamento sono state utilizzate le shape BALL_ANGLE e BALL DISTANCE.

Il robot quindi rientra in porta ignorando completamente la palla (al più la evita) quando si trova nella posizione 'Far From Home' della shape ROBOT_X_COORD (vedi Figura 5.12), mentre, quando arriva nella zona 'Out Of Home' del campo, continua la sua corsa verso casa solo se la palla è lontana o se non riesce a vederla. In caso contrario comincia la parata.

La sua “corsa verso casa” si interrompe quando arriva nella zona ‘In Home’ secondo la shape HOME_DISTANCE.

Quando, dopo un azione di gioco, la palla si allontana al punto di non essere più pericolosa (tipicamente quando supera la metà campo, cioè intorno ai 5 metri), Rabbiati ha la necessità di riposizionarsi al centro della porta per essere pronto alla prossima azione offensiva degli avversari. Anche questa funzionalità è fornita dal comportamento di Goto Home: in questo caso viene sfruttato il nuovo fuzzy set denominato ‘In Area’ e la posizione del robot lungo l’asse delle ordinate. Si suppone che il robot sia nella sua tipica fascia di gioco, ma se così non fosse le altre regole della Goto Home provvederebbero in tal senso. Rabbiati, quindi, quando si trova nella zona ‘In Area’ (il comportamento di Align assicura il suo corretto orientamento) e la palla è ‘Away’, secondo la shape BALL_DISTANCE (vedi Figura 5.16), si riposiziona al centro della porta pronto per una nuova parata.

Defuzzyficazione

Per poter attuare i movimenti decisi, i dati fuzzy prodotti dal Behavior Engine devono essere defuzzyficati in modo che possano essere trasmessi al sistema di controllo di basso livello che ne assicura l’attuazione. I dati prodotti e inviati al basso livello sono una velocità tangenziale, detta Tanspeed, una velocità di rotazione, detta RotSpeed e un comando di Kick. Per poter passare da dati fuzzy a dati *crisp* si utilizzano delle shape che hanno una funzione inversa da quelle incontrate finora.
Di seguito riportiamo le shape per la defuzzyficazione della Tanspeed, della RotSpeed e del comando di Kick:

```
(TANSPEED
(SNG (VERY_FAST_FORWARD 150))
(SNG (FAST_FORWARD 120))
(SNG (FORWARD 80))
(SNG (SLOW_FORWARD 40))
(SNG (VERY_SLOW_FORWARD 30))
(SNG (STEADY 0))
(SNG (VERY_SLOW_BACKWARD -30))
(SNG (SLOW_BACKWARD -40))
(SNG (BACKWARD -80))
(SNG (FAST_BACKWARD -120))
(SNG (VERY_FAST_BACKWARD -150))
)

(KICK
(SNG (RIGHT 1))
(SNG (LEFT -1))
(SNG (CENTER 2))
)
```

```
(ROTSPEED
(SNG (VERY_FAST_RIGHT -70))
(SNG (FAST_RIGHT -45))
(SNG (MORE_RIGHT -31))
(SNG (RIGHT -25))
(SNG (QUITE_RIGHT -18))
(SNG (SLOW_RIGHT -14))
(SNG (VERY_SLOW_RIGHT -8))
(SNG (AHEAD 0))
(SNG (VERY_SLOW_LEFT 8))
(SNG (SLOW_LEFT 14))
(SNG (QUITE_LEFT 18))
(SNG (LEFT 25))
(SNG (MORE_LEFT 31))
(SNG (FAST_LEFT 45))
(SNG (VERY_FAST_LEFT 70))
)
```

Le velocità tangenziale e di rotazione sono espresse in centimetri al secondo, la velocità del robot può variare quindi tra +1.5m/s e -1.5m/s tangenzialmente e tra +0.7m/s e -0.7m/s in rotazione, i tre comandi di Kick sono per la "kickkata" destra, sinistra o centrale.

Attivazione dei Comportamenti e Prove Sperimentali

Durante il gioco, tutti i comportamenti sono pressoché attivi contemporaneamente. L'unica eccezione è fatta per il comportamento di Goal Keeping che viene inibito per mezzo delle condizioni di *cando* nel momento in cui il robot si trova fuori dalla zona di parata o si trova in essa ma fortemente disallineato dal suo binario immaginario di parata. Questa scelta è stata effettuata nel primo caso, per evitare che durante il rientro in porta i robot cominci a parare in una posizione troppo avanzata e vulnerabile all'offensiva avversaria (ad es. a metà campo) solo per il fatto che in quel momento sta passando vicino alla palla, e, nel secondo caso, per evitare che il robot, in quel momento fortemente disallineato, muovendosi lateralmente vada ad urtare violentemente la porta o esca completamente da essa.

Questi comportamenti sono stati testati singolarmente e contemporaneamente, ed hanno subito una continua messa a punto soprattutto durante le varie manifestazioni a cui il robot ha partecipato.
Sono dell'idea che in questo modo Rabbiati abbia raggiunto un livello soddisfacente di gioco ed una buona affidabilità. Nella Figura 5.26 possiamo vedere una sequenza di fotogrammi che ritraggono Rabbiati in parata contro uno dei *Golem* della squadra degli "Artisti Veneti" di Padova durante SMAU 2003.

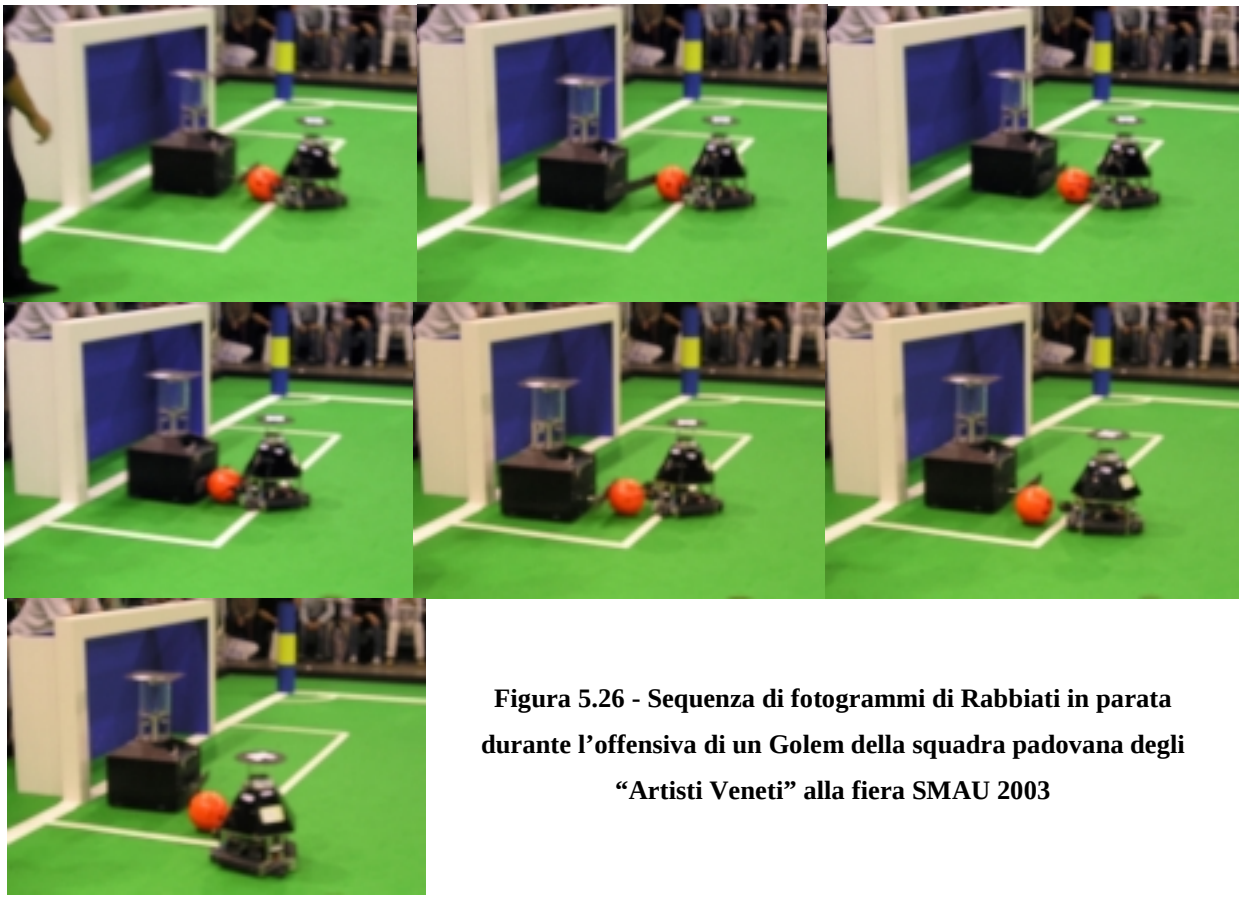


Figura 5.26 - Sequenza di fotogrammi di Rabbiati in parata durante l'offensiva di un Golem della squadra padovana degli "Artisti Veneti" alla fiera SMAU 2003