**Università di Pisa**

FACOLTÀ DI INGEGNERIA
Corso di Laurea Specialistica in Ingegneria dell'Automazione

Tesi di Laurea Specialistica

# Controllo di uno Sciame di Robot utilizzando il Framework delle Funzioni Descrittive

Candidato:
**Andrea Ferrari Braga**

Relatore:
**Mario Innocenti**

Correlatore:
**Lucia Pallottino**

**Anno Accademico 2012–2013**

*«Eccomi adesso qui, povero stolto,*
*e tanto so quanto sapevo prima.*
*Mi chiamano Maestro: anzi Dottore,*
*e son dieci anni che menando vo*
*pel naso i miei scolari,*
*di su di giù, per dritto e per traverso*
*Ma solo per accorgermi*
*che non ci è dato di sapere, al mondo,*
*nulla di nulla.*
*E quasi mi si strugge, ardendo il cuore.»*

Faust, W. Goethe

# INDICE

# ELENCO DELLE FIGURE

# ELENCO DELLE TABELLE

# SOMMARIO

Lo studio dei sistemi *multi-agente* ha da sempre attirato l'interesse della comunità scientifica per le enormi potenzialità applicative e per la quantità di discipline coinvolte. L'unirsi in gruppi infatti, anche per molte forme viventi, è da sempre stato uno strumento fondamentale per la soluzione di tantissimi problemi quotidiani, ed è per questo motivo che anche in robotica si è cercato di riproporre gli stessi espedienti e stratagemmi usati in natura.

Pur considerando l'ampia letteratura al riguardo, lo studio dei sistemi multi-agente non ha ancora visto nascere una metodologia generale utilizzabile per ogni possibile situazione e per qualunque tipologia di aggregazione di robot. In quest'ottica pertanto, nella presente tesi, viene proposto l'utilizzo di un nuovo *framework* con cui è possibile studiare, sotto un unico formalismo matematico, un gran numero di problemi legati al controllo e alla gestione degli *Heterogeneous Swarm* (sciami eterogenei), che vengono così realizzati da squadre miste di veicoli, velivoli, sensori o robot di qualunque tipo. Da questo punto di vista i termini *Sensor-Network*, *Swarm* e *Multi-Robot* diventano sinonimi e risulta così più appropriato parlare di *agenti* quando occorre riferirsi al singolo elemento.

La soluzione adottata in questa tesi si basa sull'utilizzo delle *Funzioni Descrittive* che servono a modellare sia le qualità operative degli agenti, sia le richieste del task che deve essere portato a termine. Considerando una particolare norma della funzione di errore, calcolata a partire dalla differenza fra la *TDF* (funzione descrittiva del task) e la *cTDF* (funzione descrittiva dello swarm), viene generato il moto dello sciame, secondo un algoritmo di tipo discesa del gradiente. Ricordando che un sistema di questo tipo rientra nel campo dei sistemi distribuiti, è anche proposta una soluzione di tipo decentralizzato per implementare gli algoritmi di controllo contenuti nel framework. A questo scopo diventa quindi fondamentale lo scambio delle informazioni all'interno della rete di comunicazione (il *Network*), e vengono così utilizzati algoritmi di *Agreement* o *Consensus*.

# ABSTRACT

The Control of a swarm of agents is a general problem that has attracted attention from a lot of researchers and research organizations in the recent past for the great potentiality and for the several field involved (mobile robotics, operation research, distributed networking, aerospace, etc.). For many natural living species, join groups is obviously a good practices for the solution of many different problems. Researchers, for this matter, want to used the same natural expedients also in robotics.

The basic questions have been addressed and solved in very different ways but a general methodology, which could handle several scenarios and ensemble of heterogeneous agents, is not arisen too. This thesis proposes a new framework with which is possible to study, under the same mathematical approach, a lot of control and management problems that could be realized by team of vehicles, aircraft, sensors or different kind of robots. From this point of view Sensor-Network, Swarm and Multi-Robot are considered synonymous, and is too much appropriate talk about *agent* when the single element is mentioned.

The methodology used in this thesis is based on the definition of *Descriptor Function*, that model the capabilities of the single agent and the task requirement. The DF is not strictly a density function applicable to coverage task only, but, more generally, is a representation of the properties that identifies uni-vocally a single agent in the economy of the problem. The swarm motion is controlled by minimizing a suitable norm of the error between the TDF (task descriptor function) and the cTDF (the sum of agents descriptor functions), in a gradient descent solution. This systems are always realized in distributed form and for this reason is proposed also a decentralized solution for the DF Framework: considering the Network Communication and the State Estimation problems, Agreement and Consensus algorithms are used.

# 1. INTRODUZIONE

Gli argomenti su cui si sviluppa questa tesi appartengono alla branca della ricerca che si occupa dello studio dei sistemi *multi-robot*. Più precisamente vengono trattati argomenti riguardanti la *robotica degli sciami*, in cui vengono studiati il controllo e la gestione di un sistema composto da una moltitudine di veicoli e/o velivoli con caratteristiche spesso differenti. Il termine "*Swarm of Heterogeneous Vehicles*", spesso utilizzato in questi casi, vuole proprio sottolineare la diversità fra i componenti dello sciame sia una della caratteristiche fondamentali. Occorre poi aggiungere che nella concezione più generale possibile uno *swarm* può essere anche composto da elementi non mobili, come dei sensori o delle antenne, e per questo motivo è sempre più opportuno parlare di agenti piuttosto che di veicoli. In questo modo sarà possibile considerare non solo squadre di veicoli/velivoli/robot identici ma anche possibili combinazioni fra agenti con caratteristiche fisiche, dinamiche, qualitative ed operative differenti.

Lo studio del controllo, del comportamento, e della gestione di un insieme di agenti ha fortemente attirato negli ultimi anni l'interesse della comunità scientifica. Il motivo è facilmente comprensibile dato che l'unirsi in gruppi, anche per tantissime forme viventi, è da sempre stato uno strumento fondamentale sia per risolvere problemi quotidiani che per sopravvivere. Per quanto appena detto l'approfondimento di questi argomenti ha interessato e sta coinvolgendo settori di diverse branche scientifiche, dalla sociologia alla biologia, dalla matematica all'ingegneria. Per ciò che riguarda le discipline ingegneristiche, e quindi anche gli argomenti di questa tesi, l'obiettivo principale è quello di modellare e studiare, nei sistemi multi-agente artificiali, le diverse problematiche ad essi inerenti. Alcune di queste possono essere ad esempio la comunicazione fra elementi di un network, il controllo coordinato del moto e la risoluzione ottimale di un task.

Da un punto di vista pratico è inoltre possibile intuire come le *Sensor-Network*, i sistemi Multi-Agente e gli sciami di veicoli possano anche venir impiegati per risolvere compiti non realizzabili con un singolo elemento. A parità di risultati e numero di compiti svolti è inoltre sempre preferibile optare per un sistema composto da più elementi cooperanti di basso costo piuttosto che un unico robot di elevato costo e di difficile progettazione.

Durante questi ultimi anni sono state proposte una moltitudine di soluzioni specifiche che hanno cercato di far fronte a tutta quella serie di problemi che nascono dall'utilizzo contemporaneo di più robot. Alcuni esempi tipici di applicazione sono: lo studio del moto coordinato (in cui è stato preso spunto dalla Biologia per quanto concerne il movimento di stormi, sciami e branchi), la comunicazione e l'apprendimento (vedi Sociologia per lo studio del comportamento delle masse), ma anche l'assegnamento e la suddivisione dei compiti da svolgere (assegnamento e gestione delle risorse).

I primi studi ingegneristici in questo settore sono stati fatti per riproporre dal punto di vista matematico comportamenti e situazioni già esistenti in natura, emulando così quanto fatto da alcuni gruppi animali. In questo ambito è importante ricordare quanto scritto in "*Swarm Intelligence*" di Kennedy e Eberhart [28], "*Swarm Intelligence: from natural to artificial systems*" di Bonebeau, Dorigo e Theraulaz [29], e "*Biomimicry for Optimization, Control, and Automation*" di Passino [30]. Un esempio tipico è quello di coordinare i movimenti dei veicoli/velivoli in maniera da realizzare quello che in natura viene fatto dagli stormi di uccelli, dove il movimento del gruppo verso una certa direzione comporta anche tutto un insieme di sotto-movimenti effettuati dai singoli elementi che lo compongono. Alcune soluzioni a questo problema sono state proposte da Innocenti, Pollini e Niccolini negli articoli [3], [4], [5].

*Fig. 1.1:* Natural vs Robot Swarm

Con il passare del tempo l'interesse della comunità scientifica si è poi esteso anche verso la ricerca della soluzione di problemi differenti. Un esempio può essere quello riguardante la copertura di una determinata area, il *coverage*, dove è possibile citare il lavoro svolto da Bullo e collaboratori (vedi [8]), ma anche da Hussein e Stipanovic (vedi [9], [20] e [34]) che presentano un approccio simile a quello che verrà utilizzato in questa tesi. Un ulteriore

aspetto di fondamentale importanza è quello legato allo studio della comunicazione e del mantenimento della connettività all'interno del network, dove è importante ricordare quanto fatto da Egerstedt e collaboratori (vedi [22] e [35]), e da Beard e Ren (vedi [7], [17], [27] e [33]) che hanno applicato l'analisi matematica riguardante la teoria dei grafi e del consenso sui sistemi di tipo multi-agente.

Considerando tutti questi aspetti è possibile pensare ad un problema prettamente decisionale, dove l'obiettivo diventa quello di definire una legge di controllo in grado di muovere in maniera ottimizzata gli elementi dello sciame. La domanda a cui rispondere diventa la seguente: "*dato un certo numero di compiti da eseguire, è possibile definire un'unica legge, valida per tutti gli agenti, che li porti a coordinarsi, muoversi, cooperare, in modo da risolvere tutti i compiti proposti secondo un certo criterio di ottimo?*". La presente tesi si sviluppa in quest'ultima direzione e fa riferimento a due lavori che costituiscono i presupposti e le basi teoriche da cui partire: la relazione scritta dall'Ing. Mario Innocenti "Shape Control of a Swarm of Heterogeneous Vehicles: Final Report" [1]; la tesi di dottorato presso la scuola Leonardo Da Vinci dell'Università di Pisa "Swarm Abstractions for Distributed Estimation and Control" [2], scritta dall'Ing. Marta Niccolini.

Quello che sostanzialmente emerge da queste due ricerche è un insieme di strumenti matematici utili per il controllo coordinato degli agenti. La mia tesi prende spunto da una delle soluzioni utilizzate e propone l'utilizzo di un nuovo "*framework*" con cui poter rappresentare tutti gli agenti dello sciame attraverso le medesime funzioni matematiche. L'idea base è quella di partire considerando quelle che sono le qualità operative di ciascun agente nell'eseguire un determinato compito, o task, per poi realizzare una opportuna espressione matematica in grado di descriverle in funzione dell'ambiente circostante. Tralasciando l'implementazione pratica di quanto è stato appena detto, che verrà invece ben illustrata nel proseguo della tesi ( vedi capitolo 3 ), è quindi possibile considerare per ogni agente una "funzione descrittiva", o *Descriptor Function*, che lo rappresenterà dal punto di vista operativo.

Ampliando questo concetto all'intero sciame, sarà quindi possibile definire una funzione descrittiva globale indicativa dello stato complessivo dell'intero sistema. Tale funzione indicherà all'interno dello spazio di lavoro la distribuzione delle risorse utili a realizzare i diversi task. In questa ottica sono quindi realizzate sia una funzione descrittiva desiderata, o *Desired Descriptor Function*, sia una *Current Task Descriptor Function* che rappresenterà lo stato di esecuzione di quel determinato task.

Sfruttando quanto definito nel framework, in [1] e [2] viene anche proposta una legge di tipo algoritmo del gradiente utile a far muovere gli agenti nella direzione di minimizzazione di un opportuno funzionale di costo. Partendo

dalle problematiche che questa legge non riesce a risolvere, e che saranno più dettagliatamente esposte nel capitolo 4, si svolge il vero lavoro di questa tesi che consiste nello sviluppare e nel migliorare quanto già presente in questo primo framework delle funzioni descrittive. In questa direzione un passo essenziale verrà compiuto nel progettare un adeguato meccanismo di *obstacle avoidance* che permetterà agli agenti di muoversi nello spazio evitando ogni sorta di collisione.

Ottenuta una nuova legge di controllo e dimostrata la sua efficacia, occorrerà implementare il tutto in un sistema distribuito realizzato attraverso il network di comunicazione rappresentativo dello sciame. Partendo dalla teoria dei grafi (capitolo 2), con cui è possibile modellare matematicamente una qualunque rete di connessioni, verrà proposto un *algoritmo di consenso* (capitolo 5), che servirà a realizzare la legge di controllo all'interno dello swarm, in una visione decentralizzata del problema.

Quello che con più precisione viene approfondito in questa seconda parte della tesi è il cosa e il come gli agenti devono comunicare fra loro per avere tutti una visione esatta e unanime dello stato globale del problema. In altre parole, verranno prima di tutto definiti un certo numero di parametri che gli agenti si devono comunicare per conoscere lo stato complessivo dell'esecuzione di un task, dopo di che, occorrerà anche realizzare un meccanismo di salvaguardia e stima qualora non sia sempre possibile per un singolo agente comunicare con tutti gli altri.

L'efficacia delle soluzioni proposte verrà dimostrata attraverso alcune simulazioni e con un "hardware testbed", in cui verranno utilizzati due veicoli reali di tipo "mini-car" come agenti dello swarm (capitolo 8).

Prima di proseguire viene qui di seguito riportata una breve e schematica panoramica di quella che è la classificazione dei problemi di controllo nell'ambito dei sistemi multi-agente.

## 1.1 Controllo Distribuito

Nell'ambito del controllo multi-robot è difficile stabilire una classificazione finale fra le diverse tipologie di problemi, dato che tutt'ora questo è uno dei tanti argomenti discussi fra i ricercatori. Tuttavia è comunque sempre possibile, partendo dal concetto base di *neighbours*, definire una prima suddivisone fra problema *locale* e *globale*. Dato un elemento dello sciame e definito come *neighbour* un qualunque altro agente con cui è possibile scambiare delle informazioni (vedi capitolo 2), si può fare la seguente distinzione:

- Se il problema di controllo può essere formulato come combinazione dei valori dello stato dei soli agenti appartenenti al *neighbourhood* il problema viene definito *locale*.

- se invece per risolvere il problema di controllo è necessario conoscere il valore dello stato di tutti gli agenti del network il problema viene definito *globale*.

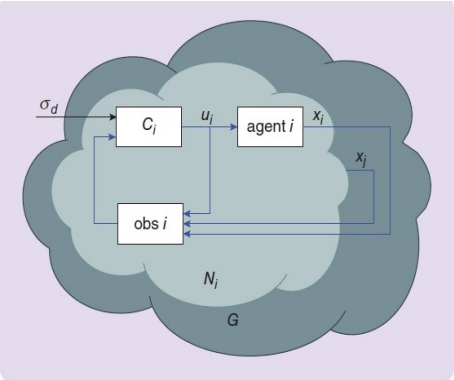


*Fig. 1.2:* Global Problem in a Distributed Architecture

Partendo da questo presupposto e considerando ora l'architettura del sistema di controllo, è invece possibile distinguere fra sistemi distribuiti o decentralizzati, dove vengono scambiate e sono note le sole informazioni locali, e sistemi centralizzati, in cui è sempre disponibile, in ogni istante di tempo l'intero vettore di stato del sistema. In questo modo si vengono a creare le seguenti possibili combinazioni:

- Problema di Controllo Globale risolto con una struttura centralizzata: le informazioni scambiate sono globali ed il controllore può accedere all'intero vettore di stato $x$.

- Problema di Controllo Globale risolto con una struttura Parzialmente Distribuita: gli agenti sono in grado di accedere alle informazioni globali che vengono così elaborate a bordo.

- Problema di Controllo Locale risolto con una struttura Distribuita: gli agenti comunicano con i soli *neighbours* e risolvono un problema noto a solo quel sottogruppo.

- Problema di Controllo Globale risolto con una struttura Distribuita: gli agenti comunicano con i soli *neighbours* per risolvere un problema che invece coinvolge tutto lo swarm.

Il problema che verrà affrontato in questa tesi è riconducibile all'ultimo caso anche se inizialmente (all'interno del cap 4), per semplicità di trattazione e per rendere le dimostrazioni più comprensibili, le leggi e gli algoritmi proposti saranno applicati ipotizzando una architettura di tipo centralizzata.

# 2. GRAFI E ALGORITMI DI CONSENSO

## *2.1 Introduzione*

Lo studio degli swarm è legato strettamente alla teoria dei grafi che svolge un ruolo fondamentale per l'analisi e la formalizzazione delle leggi matematiche che governano questi sistemi. Volendo infatti schematizzare più agenti all'interno di un generico ambiente e pensando di mappare la rete formata dalle loro *connessioni*, è possibile rappresentare quanto appena detto attraverso un grafo.

In una rete generica, due agenti possono quindi essere messi in collegamento in funzione di diversi aspetti che possono andare da quello comunicativo, legato allo scambio di informazioni, a quello di visibilità. In ogni caso per un singolo agente si può parlare di "*neighbors*" riferendosi a tutta quella serie di veicoli e/o agenti che in quel determinato momento sono a lui direttamente collegati. Un esempio, che ricalca una situazione in cui la connettività è ottenuta tramite visibilità, è espressa nella figura 2.1 in cui è facilmente intuibile come gli archi risultino tracciati da un nodo ad un altro solo in presenza di una linea di vista.

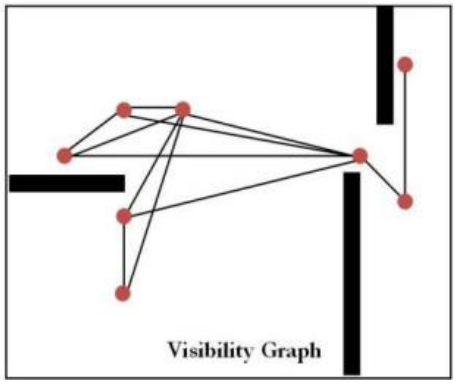


*Fig. 2.1:* Visibility Graph

In ogni caso, da ora in avanti, si farà sempre riferimento al grafo rappresentativo della comunicazione fra gli agenti, poiché la gestione e lo scambio delle informazioni saranno di fondamentale importanza per la realizzazione degli algoritmi di controllo. Ogni agente d'altronde non può conoscere completamente lo stato di esecuzione di un task se non comunicando e scambiandosi informazioni con gli agenti vicini.

Nel caso particolare di agenti in moto all'interno dell'ambiente di lavoro, è possibile intuire come la presenza di una arco possa eliminarsi o generarsi a seconda delle posizioni relative che il movimento induce nei veicoli. Per quanto appena detto è possibile suddividere i network in **statici**, quando gli archi di collegamento fra i nodi rimangono sempre costanti, e in **dinamici**, quando la topologia del grafo annesso alla rete risulta variabile nel tempo.

Mentre nel proseguo della tesi verranno proposte nel dettaglio le soluzioni ai problemi legati ai quesiti "*che tipo di informazioni si devono scambiare gli agenti* " e "*come è possibile gestire lo scambio delle informazioni in presenza di un network dinamico*", in questo capitolo viene invece fatta una breve overview sulle tecniche e gli strumenti teorici solitamente utilizzati in questi casi. Per ciò che riguarda le definizioni, la nomenclatura e le proprietà dei grafi è stata invece realizzata l'appendice A, dove si può trovare un breve elenco da consultare. Si ricorda inoltre che la dinamica dell'agente verrà sempre approssimata con quella di un singolo integratore da cui $\dot{x} = u$.

## 2.2 Protocolli di Consenso

In questa sezione vengono affrontati e presentati tutti quegli argomenti che in letteratura si possono ritrovare sotto il nome di *Consensus Protocol Problem*, che fanno parte del campo del calcolo distribuito o *distributed computation*. Per prima cosa occorre chiarire cosa si intende quando si parla di consenso o, in alternativa, di raggiungimento del consenso. In maniera molto intuitiva e del tutto generale è possibile percepire il consenso come una particolare situazione in cui tutti i soggetti coinvolti si trovano in accordo sul valore di una certa quantità di interesse che è stata presa in esame. Riportando il tutto al caso di un generico network di agenti è quindi possibile considerare la seguente definizione formale: «*il consenso viene raggiunto quando ciascuno degli agenti appartenenti ad uno sciame concorda sul valore di un opportuno set di variabili condivise*». Tali variabili prenderanno quindi il nome di *Variabili di Consenso* o, considerandole come un'unica entità, con il nome di *Information State*, che di qui in avanti verrà indicato con $x$.

L'utilizzo del *Consensus* permette la realizzazione e l'esecuzione di differenti operazioni che vanno dalla definizione di algoritmi di stima distribuita

all'esecuzione di operazioni coordinate. Alcuni esempi (vedi i riferimenti contenuti in [6]) possono essere la ricerca di un punto di *Randezvous*, il controllo di formazione (*Formation Control and Flocking Theory*) e della forma (*Shape Control* [1] [3] [4] ), la stima di un parametro in una rete di sensori, la sincronizzazione di una coppia di oscillatori. Prima di entrare nel merito della spiegazione occorre anche ricordare che la maggior parte delle formule che sono proposte qui di seguito trovano riscontro e possono essere maggiormente approfondite in [6] e [7].

Partendo dall'analisi di un network statico rappresentato con un generico grafo non orientato (*undirected graph*), indicato con $G(V, E)$ (*Vertices and Edge*), è possibile realizzare il più comune algoritmo di consenso (*Average Consensus*) con l'utilizzo della matrice Laplaciano ( vedi appendice A ) per cui si ottiene la seguente equazione dinamica:

$$\dot{x}(t) = -L(t)x(t) \Rightarrow \dot{x}_i(t) = \sum_{j \in N_i}^{n} a_{ij}(t)(x_j(t) - x_i(t)) \tag{2.1}$$

Dove $L(t)$ è il Laplaciano, $x(t)$ è il vettore su cui deve essere ottenuto il consenso, e $a_{ij}(t)$ è l'elemento della matrice di adiacenza che comporta la presenza o meno di comunicazione fra l'agente $i - esimo$ e $j - esimo$. Nello sviluppo dell'equazione si definiscono poi con $x_i(t)$ il valore dell'*Information State* del generico agente e con $N_i$ l'insieme dei vicini o *neighbours* con cui è connesso. Dato che la matrice $L(t)$ è per definizione sempre semi-definita positiva, allora esiste anche una convergenza asintotica della soluzione $x(t)$ non appena è presente al massimo un solo autovalore nullo in $L(t)$. Quest'ultima condizione è verificata non appena il secondo più piccolo autovalore di $L(t)$ ovvero $\lambda_2$ risulta diverso da zero. La condizione appena espressa è del tutto equivalente a dire che il grafo analizzato debba essere connesso (vedi appendice A) e ciò, nel caso di grafo non orientato, si ottiene non appena è presente un albero di copertura o *Spanning Tree*.

**Teorema 1** (Convergenza del Consenso in un Grafo Non Orientato)**.** *Il vettore information state $x(t)$ converge al consenso, quale che sia la condizione iniziale $x(0)$, se e solo se il grafo associato alla rete di agenti contiene un albero di copertura.*

Sotto queste ipotesi l'autovalore $\lambda_2$ definisce inoltre il limite dinamico con cui avviene la convergenza. Il valore asintotico finale può essere espresso

attraverso la seguente formula che altro non é che la media dei valori iniziali contenuti nell'*information state* dei singoli agenti.

$$x_i(\infty) \to \frac{1}{N} \sum_{k=1}^{N} x_k(0) \tag{2.2}$$

Per cui, considerato l'autovettore $1_N$ legato all'autovalore nullo, è possibile riscrivere la formula precedente in questo modo:

$$x_i(\infty) = 1_N \frac{1}{N} \sum_{k=1}^{N} x_k(0) = 1_N \alpha \tag{2.3}$$

Un esempio di quanto appena detto è illustrato nella seguente immagine che rappresenta il grafo di figura A.1 considerato un vettore iniziale $x(0) = [1, 2, 3, 4, 5]^T$. Il riscontro grafico conferma uno stato finale pari a $x(\infty) = 15/5 = 3$.

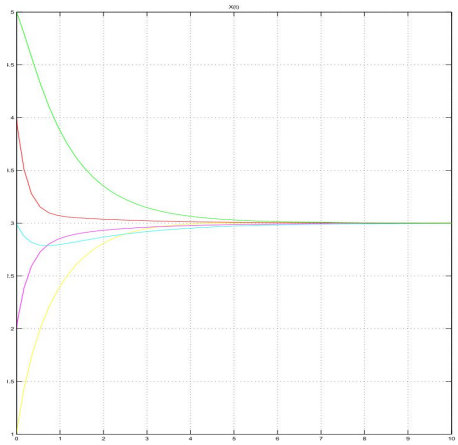

*Fig. 2.2:* Consensus Protocol Convergence with $x(0) = [1, 2, 3, 4, 5]^T$

Analizzando invece il caso di grafo orientato o *digraph* è possibile ottenere la convergenza dell'algoritmo di consenso se e solo se il grafo risulta fortemente connesso. In queste condizioni l'*average consensus* può essere raggiunto una volta soddisfatte anche le condizioni di bilanciamento, ovvero quando $\sum_{i \neq j} a_{ij} = \sum_{i \neq j} a_{ji}$ $\forall i \in V$. Nei grafi orientati la proprietà di connessione

forte implica (e non vale il viceversa) la presenza di un *rooted direct spanning tree* ovvero di un albero ricoprente con radice o *leader*.

Se in un network si volessero considerare diversamente le comunicazioni che avvengono tra due agenti piuttosto che tra altri, è possibile introdurre degli opportuni pesi in modo da valorizzare differentemente ogni arco del grafo non orientato. Il problema appena introdotto prende il nome di *Weighted Consensus Problem* e viene risolto in maniera del tutto simile al caso precedente. La matrice di adiacenza, come del resto quella dei gradi, deve però essere ridefinita considerando gli opportuni pesi. Mentre in un grafo non orientato il peso $m_{ij}$ deve essere identico a $m_{ji}$, ottenendo così sempre un Laplaciano simmetrico, nel caso orientato il Laplaciano risulterà in generale non simmetrico. E' possibile riscrivere la matrice di adiacenza secondo la seguente regola.

$$a_{ij} = \begin{cases} m_{ij}, & \text{se } (j,i) \in E \\ 0, & \text{altrimenti.} \end{cases}$$

Per la matrice $D$ dei gradi si procede calcolando il singolo elemento sulla riga/diagonale $i - esima$ come $d_i = \sum_{j=1}^{N} m_{ij}$ ottenendo infine il nuovo Laplaciano calcolato sempre come $L = D - A$. Il problema di consenso diventa dunque il seguente:

$$\dot{x}_i(t) = \sum_{i=j}^{N} m_{ij}(t)(x_j(t) - x_i(t)) \tag{2.4}$$

La cui soluzione di convergenza $x(\infty)$ è calcolabile come indicato qui di seguito:

$$x(\infty) = 1_N \frac{\sum_{k=1}^{N} m_i x_k(0)}{\sum_{k=1}^{N} m_i} \tag{2.5}$$

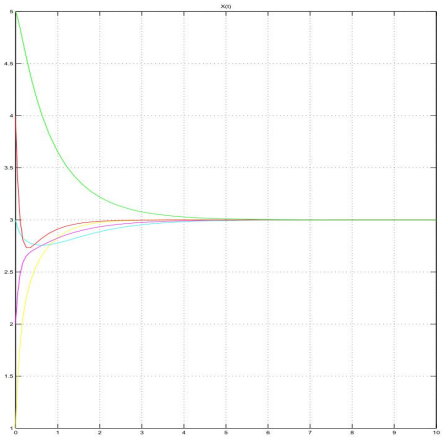

*Fig. 2.3:* Weighted Consensus Protocol Convergence in the same graph of 2.2 with the weigths $(1,4) = 4$, $(4,2) = 3$, $(2,3) = 2$, $(4,5) = 1$

Come è possibile vedere dalla figura 2.3, nel caso di grafo non orientato, si raggiunge lo stesso *information state* finale ottenuto anche nel caso precedente ma con una dinamica che risulta profondamente cambiata. Nel caso di grafo orientato è possibile riproporre ed utilizzare le medesime considerazioni fatte nel caso di grafo non pesato.

In uno scenario in cui sono presenti agenti con caratteristiche differenti per dimensioni, task da eseguire, capacità comunicative e/o operative, è possibile utilizzare il concetto di peso appena visto per realizzare quello che viene chiamato problema del consenso multi-valore o *Multivalued Consensus* ( vedi [3] e [4]). Considerando un grafo non orientato i cui archi sono pesati dagli elementi $a_{ij}$, e i cui nodi sono pesati dagli elementi $m_i$, è possibile ridefinire la dinamica dell'*information state* $i-esima$ come segue:

$$\dot{x}_i(t) = \sum_{i=j}^{N} a_{ij}(t)(m_j x_j(t) - m_i x_i(t)) \tag{2.6}$$

Il Laplaciano, ottenibile come nel caso del *Weighted Consensus*, deve essere successivamente moltiplicato per la matrice dei pesi $M$ secondo il prodotto elemento per elemento di *Hadamard*, in modo da ottenere la forma finale $L^*(t)$ che realizza l'equazione dinamica.

$$L^*(t) = M \circ L \Rightarrow \dot{x}(t) = -L^*(t)x(t)$$

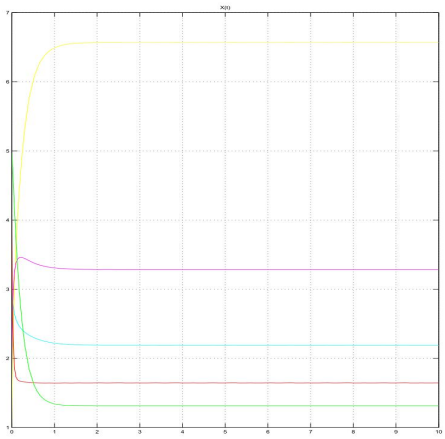

*Fig. 2.4:* Multivalued Consensus Protocol Convergence in the same graph of 2.2 with the weigths $m_1 = 1$, $m_2 = 2$, $m_3 = 3$, $m_4 = 4$, $m_5 = 5$

Per questa tipologia di Network la convergenza del problema è ottenibile se e solo se il grafo non orientato risulta connesso (dimostrazione in [1]) e l'autovettore associato all'autovalore da cui è possibile definire il valore di convergenza è calcolabile come:

$$\bar{v} = \frac{\sum_{k=1}^{N} x_k(0)}{m_1 \sum_{j=1}^{N} \frac{1}{m_j}} \begin{bmatrix} 1 \\ \frac{m_1}{m_2} \\ \frac{m_1}{m_3} \\ \dots \\ \frac{m_1}{m_n} \end{bmatrix} \tag{2.7}$$

Occorre adesso dare alcune precisazioni riguardanti il Laplaciano $L(t)$, che fino ad ora è sempre stato considerato sia appartenente al dominio del tempo continuo $t$, sia rappresentativo di un network di tipo statico. Entrambe queste condizioni ovviamente non sono sempre verificabili nei network reali, ed è necessario quindi ricorrere ad opportune soluzioni di ripiego.

Analizzando il caso di grafo a tempo discreto, che poi rappresenta l'implementazione pratica di molti Network (dato che i protocolli di comunicazione sono solitamente realizzati in digitale), è possibile riscrivere l'equazione di *Average Consensus* esprimendola direttamente in funzione del passo $k$:

$$x(k+1) = Px(k) \Rightarrow x(k+1) = \sum_{j=1}^{N} p_{ij} x_j(k) \tag{2.8}$$

Dove $P = I - \epsilon L$ è la *Matrice di Perron* di un grafo $G$ con parametro $\epsilon$. Questa è funzione di $I$ matrice identità, $L$ il Laplaciano ed $\epsilon$ il passo di campionamento o *step-size*. Un importante lemma (vedi [6]) che definisce le proprietà di convergenza nel caso di grafi a tempo discreto è qui di seguito riportato:

**Lemma 1.** *Sia $G$ un grafo orientato con $n$ nodi e con grado massimo pari a $\delta = \max_i(\sum_{i \neq j} a_{ij})$. Presa la matrice di Perron $P$ con $\epsilon \in (0, 1/\delta)$ questa soddisfa le seguenti proprietà:*

- *$P$ è una matrice di tipo* row stochastic *non negativa con un autovalore semplice unitario.*
- *Tutti gli autovalori di $P$ sono all'interno del cerchio unitario.*
- *Se $G$ è un grafo bilanciato allora $P$ è* doubly stochastic *quindi sia per ricghe che per colonne*
- *Se $G$ è un grafo fortemente connesso e $0 < \epsilon < \frac{1}{\delta}$ allora $P$ è una matrice primitiva ovvero possiede un solo autovalore di modulo massimo.*

Il seguente teorema definisce quindi le proprietà di convergenza che si derivano dalle considerazioni geometriche espresse dal precedente lemma.

**Teorema 2** (Consenso in un Grafo a Tempo Discreto)**.** *Considerando un network di agenti $x_i(k+1) = x_i(k)+u_i(k)$ con topologia $G$ è possibile applicare il seguente algoritmo di consenso distribuito:*

$$x_i(k+1) = x_i(k) + \epsilon \sum_{j \in N_i} a_{ij}(x_j(k) - x_i(k))$$

*dove $0 < \epsilon < \frac{1}{\delta}$ e $G$ è un grafo orientato fortemente connesso. Allora:*

- *Un consenso viene raggiunto per qualunque stato iniziale $x_i(0)$.*
- *Il valore che si ottiene è dato da $\alpha = \sum_i w_i x_i(0)$ con $\sum_i w_i = 1$*
- *Se il grafo orientato è anche bilanciato (o alternativamente $P$ è doubly stochastic) è raggiunto l'average consensus paria a $\alpha = \sum_i x_i(0)/n$.*

In presenza di una topologia dinamica è invece opportuno effettuare un'analisi legata al concetto di *Dwell Time* che rappresenta il lasso di tempo in cui la topologia di un grafo rimane costante. Calcolando il dwell time $\tau_i$ come $\tau_i = t_{i+1} - t_i$ è quindi possibile riscrivere la dinamica del consenso in funzione dell'istante in cui è avvenuto l'ultimo cambiamento nella rete.

$$\dot{x}(t) = -L_D^*(t_i)x(t) \text{ per } t \in [t_i, t_i + \tau_i] \tag{2.9}$$

La variabile di consenso $x(t)$, in relazione alla sua storia, è esprimibile a questo punto in funzione dell'evoluzione temporale dei diversi Laplaciani.

$$x(t) = [e^{-L_D^*(t_k)(t-t_k)} e^{-L_D^*(t_{k-1})(\tau_{k-1})} \dots] x(0) \tag{2.10}$$

Dove, nella formula appena scritta, evidenziata dalle parentesi quadre, è presente una matrice di transizione definita da un prodotto infinito di **SIA** (*Stochastic Indecompensable and Aperiodic Matrices*). Per quanto riguarda il raggiungimento del consenso, questo viene garantito solo se, durante il passaggio da una topologia alla successiva, il grafo mantiene comunque le proprietà necessarie espresse nelle pagine precedenti.

## 2.3 Il Consenso Dinamico

Quando il consenso deve essere raggiunto su una variabile che si modifica nel tempo, gli algoritmi proposti nella sezione 2.2 perdono di efficacia. Per ovviare a questo inconveniente è stata studiata da Murray M., Spanos P., Olfati-Saber R. una prima soluzione (vedi [10]) denominata *Dynamic Consensus* [1]. Partendo dall'equazione 2.1 è possibile aggiungere un termine esogeno $z_i(t)$ che può rappresentare il comportamento di un ingresso esterno o di un disturbo da filtrare (ma non è questo il caso in analisi). Si ottiene:

$$\dot{x}_i = \sum_{j=1}^{n} a_{ij}(x_j - x_i) + z_i \ \forall i \tag{2.11}$$

[1] E' importante ricordare di non confondere l'attributo "dinamico", fino ad ora assegnato alla topologia del network, con quanto verrà scritto nelle pagine seguenti

Questa espressione può essere studiata in frequenza attraverso una trasformazione nel dominio di Laplace. Partendo per semplicità dall'equazione $\dot{x}(t) = -Lx(t)$ si ottiene:

$$\dot{x}(t) = -Lx(t) \longrightarrow sX(s) - X(0) = -LX(s) \tag{2.12}$$

Da cui è possibile scrivere:

$$X(s) = (sI + L)^{-1}X(0) \tag{2.13}$$

Considerando poi $X(0)$ come un generico ingresso esterno di tipo gradino è possibile effettuare i seguenti passaggi dove viene ora utilizzato $Z(s) = X(0)/s$ e da cui è possibile derivare l'espressione di una matrice di trasferimento I/O:

$$X(s) = s(sI + L)^{-1}Z(s) \longrightarrow H_{xz} = s(sI - L)^{-1} \tag{2.14}$$

La matrice di trasferimento $H_{xz}$ suggerisce quindi una possibile nuova soluzione per il raggiungimento del consenso dato un segnale in ingresso generico $z(t)$. La sua anti-trasformata produce infatti l'espressione:

$$\dot{x} = -Lx(t) + \dot{z}(t) \tag{2.15}$$

Dove i singoli termini $z_i$ sono puramente locali e dove vale la seguente proprietà di conservazione per cui $\frac{d}{dt}\sum_i x_i = \frac{d}{dt}\sum_i z_i$. Da quanto appena scritto è possibile concludere che, se il sistema è definito con la matrice di trasferimento 2.14, ogni agente può raggiungere il consenso con errore a regime nullo solo se il segnale di ingresso $Z(s)$ possiede al massimo un polo in zero (ad esempio segnale a gradino).

Volendo generalizzare quanto appena scritto per il caso di segnali in ingresso con più poli nell'origine ( inseguimento alla rampa, parabola, ecc... ) è poi possibile utilizzare un *filtro predittivo* $C(s)$ per cui la matrice di trasferimento del sistema può essere messa nella forma seguente:

$$H_{xz} = C(s)(C(s)I - L)^{-1} \tag{2.16}$$

Una scelta possibile di $C(s)$ per fare in modo che il sistema raggiunga il consenso inseguendo un segnale a rampa può essere la seguente con $\alpha > 0$:

$$C(s) = \frac{s^2}{(s+\alpha)^2} \tag{2.17}$$

Questa soluzione comporta però l'inconveniente pratico per cui occorre conoscere il valore sia di $\dot{z}(t)$ che di $\ddot{z}$. In una applicazione reale ciò vuol dire che, per stimare la posizione di un veicolo che si muove incrementando la sua posizione con una dinamica di tipo rampa, occorre conoscerne sia la velocità che l'accelerazione.

## 2.4 Consensus Tracking

Quando si presentano situazioni in cui è necessario che tutti gli elementi di un network inseguano un medesimo valore di riferimento imposto da uno degli agenti (il *leader*), è possibile sfruttare la teoria del *Consensus Tracking*, che in questa sezione verrà brevemente riproposta negli aspetti più inerenti allo sviluppo della tesi.

Il nodo che rappresenta il leader/root viene considerato come esterno al network, dato che possiede solo archi uscenti. Preso quindi un generico grafo orientato, questo sarà formato da $n$ nodi, che prenderanno il nome di *followers*, e da un leader, che rappresenterà l'elemento $n+1$. Una prima conseguenza di quanto appena scritto si ripercuote nell'espressione della matrice di adiacenza che verrà considerata in una sua forma particolare dove il leader svolge la sola funzione di *root*:

$$A(G) = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & a_{(1,n+1)} \\ a_{21} & a_{22} & \dots & a_{2n} & a_{(2,n+1)} \\ a_{31} & a_{32} & \dots & a_{3n} & a_{(3,n+1)} \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} & a_{(n,n+1)} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Chiamando con $x^r(t)$ il valore imposto dal leader sull'information state ed ipotizzando di avere per questo primo esempio $x(t) = const$, è possibile arrivare alla medesima conclusione del capitolo 2.2, in cui tutti i followers raggiungono il riferimento costante imposto dal leader, non appena è presente

nel grafo un albero ricoprente (*direct spanning tree*) a partire dall'elemento radice. Questa conclusione però perde di valore non appena $x(t)$ diventa variabile nel tempo. Una prima soluzione a questo problema viene fornita dalla seguente equazione dinamica di tracking (vedi [7]):

$$\dot{x}_i = a_{(i,n+1)}\dot{x}^r - a_{(i,n+1)}\alpha_i(x_i - x^r) - \sum_{j=1}^{n} a_{ij}(x_i - x_j) \tag{2.18}$$

Dove $\alpha_i$ è una generica costante scalare scelta arbitrariamente, $a_{ij}$ è il valore dell'elemento della matrice di adiacenza e $a_{(i,n+1)}$ è l'elemento della matrice di adiacenza che vale 1, se il leader è direttamente collegato a quel nodo o 0, altrimenti. Utilizzando quest'ultimo algoritmo di tracking si può concludere che:

**Teorema 3** (Inseguimento del Consenso)**.** *Ipotizzando di lavorare su un grafo orientato di tipo statico, se tutti gli elementi della matrice di adiacenza $a_{(i,n+1)}$ legati alla comunicazione con il leader sono pari ad* 1*, allora il problema di inseguimento del valore di riferimento tempo variabile viene risolto correttamente con l'algoritmo 2.18.*

Quanto precedentemente scritto comporta quindi che il leader sia collegato direttamente con un arco a ciascuno degli altri agenti e deve poter comunicare sia il valore $x^r(t)$ che la sua derivata temporale $\dot{x}^r(t)$. Questo primo algoritmo può anche essere visto come una possibile conseguenza dell'applicazione della teoria del consenso dinamico.

Un secondo algoritmo di tracking può essere ottenuto anche nel caso in cui il valore di $x^r(t)$ e $\dot{x}^r(t)$ è disponibile solo per un sottogruppo di agenti (vedi [7] e [18]):

$$\dot{x}_i = \frac{1}{\eta_i}\sum_{j=1}^{n} a_{ij}[\dot{x}_j + \gamma(x_j - x_i)] + \frac{1}{\eta_i}a_{(i,n+1)}[\dot{x}^r + \gamma(x^r - x_i)] \tag{2.19}$$

Dove $a_{ij}$ è il generico elemento della matrice di adiacenza, $\gamma$ è una costante scelta arbitrariamente e $\eta_i(t) = \sum_{j=1}^{n+1} a_{ij}$. Nell'implementazione pratica è opportuno ricordare che il valore di $\dot{x}_j(t)$ può essere calcolato direttamente per derivazione dal valore di $x_j(t)$ comunicato dai *neighbours*. Vale in questo caso il seguente teorema ipotizzando anche qui la presenza di un grafo orientato di tipo statico:

**Teorema 4** (Inseguimento del Consenso)**.** *Utilizzando l'algoritmo 2.19 il problema di inseguimento del valore di riferimento tempo variabile viene risolto correttamente se e solo se il grafo orientato $G_{n+1}$ contiene un* ***directed rooted spanning tree****.*

Volendo infine concludere con un piccolo confronto fra le due strategie, è importante sottolineare come l'implementazione dell'algoritmo 2.19 richieda condizioni meno stringenti sulla topologia del network poiché non necessita di un leader in costante comunicazione con tutti gli elementi, ad ogni istante di tempo.

# 3. IL FRAMEWORK DELLE FUNZIONI DESCRITTIVE

## *3.1 Introduzione*

In questo capitolo viene presentato un nuovo framework utilizzabile per lo studio e per il controllo di uno sciame eterogeneo. Il singolo agente considerato come una risorsa viene descritto attraverso una particolare funzione che ne identifica le proprietà dinamiche e operative. Tale funzione prende il nome di Descrittore, o Descriptor Function (abbreviazione *DF*), e caratterizza dal punto di vista qualitativo l'agente mettendo a disposizione, a seconda del caso, proprietà quali ad esempio la dimensione, la potenza di rilevamento o la bravura nell'eseguire un determinato compito. Partendo da quanto fatto nella tesi di dottorato di Niccolini [2], in cui gli agenti erano rappresentati da sole funzioni di tipo *Gaussiano*, è stata qui sviluppata una nuova concezione di DF che sfrutta l'utilizzo di funzioni *Sigmoidee*. Sempre in aggiunta al lavoro precedente, tutte le DF sono ora state fornite anche di un ulteriore grado di libertà che ne esprime l'orientazione all'interno dell'ambiente operativo.

Volendo generalizzare il concetto di Funzione Descrittiva, all'interno del capitolo é anche indicato com'è possibile rappresentare sotto lo stesso formalismo matematico non solo le DF degli agenti (*ADF*), ma anche la DF dello swarm (*cTDF*) e la DF del task che deve essere eseguito (*TDF*).

## *3.2 DF Gaussiana*

Una prima scelta della Funzione Descrittiva (vedi [1] e [2]), può essere fatta utilizzando una curva Gaussiana che può esprimere una generica proprietà omnidirezionale dell'agente. Se ad esempio si volesse rappresentare il comportamento di un'antenna, la DF che lo descrive potrebbe essere indicativa della potenza di TX/RX del segnale; alternativamente, se si volesse rappresentare il moto di un veicolo, tale funzione potrebbe fornire in termini probabilistici la sua posizione stimata nello spazio. Questa scelta rimane comunque arbitraria e funzionale al problema assegnato all'agente. Definito un generico ambiente operativo $Q$ nelle sue $N$ dimensioni possibili $q_1$, $q_2$, ...,

$q_N$, è possibile ottenere una diversa realizzazione della DF a seconda della dimensione scelta. Per il caso monodimensionale vale la seguente espressione, considerato l'agente generico $i-esimo$ :

$$DF \longrightarrow D_i(p_i, q) = \begin{cases} D_i(p_i, q_1) = Ae^{-0.5\frac{(q_1-p_i)^2}{\sigma^2}}, & \text{se } q_1 < |p_i \pm n\sigma| \\ D_i(p_i, q_1) = 0, & \text{altrimenti} \end{cases}$$

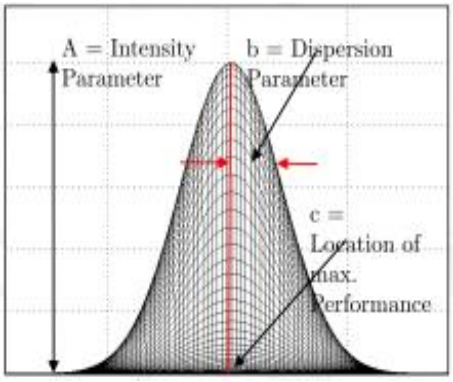


*Fig. 3.1:* Gaussian Function

Dove $p_i$ è la posizione di massima intensità della funzione (in figura 3.1 denominata con $c$), $A$ è il valore massimo che raggiunge la funzione in $p_i$ e $\sigma$ indica il grado di dispersione (in figura 3.1 denominato con $b$). La funzione inoltre è definita a tratti per rendere realistica la sua applicazione: per limitare l'estensione dei rami della gaussiana è stato scelto infatti di effettuare un troncamento ad una certa distanza dal centro $p_i$ definita dal prodotto $n\sigma$ con $n$ parametro arbitrario. Il caso bidimensionale a cui più spesso verrà fatto riferimento in questa tesi è invece espresso dalle seguenti equazioni (trascurando il termine di troncamento per semplicità):

$$D_i(p_i, q) = Ae^{-0.5(\frac{(q_1-p_{ix})^2}{\sigma_x^2}+\frac{(q_2-p_{iy})^2}{\sigma_y^2})} \tag{3.1}$$

Dove $q = [q_1, q_2]^T$, ed i valori di posizione e di dispersione sono considerati legati alle coordinate spaziali nel seguente modo: i valori "$x$" sono riferiti a

$q_1$ mentre i valori "$y$" a $q_2$. Nell'immagine 3.2 sono illustrate alcune DF gaussiane al variare dei parametri costruttivi $\sigma_x$, $\sigma_y$, $A$ e $p_i = [p_{i_x}, p_{i_y}]^T$.

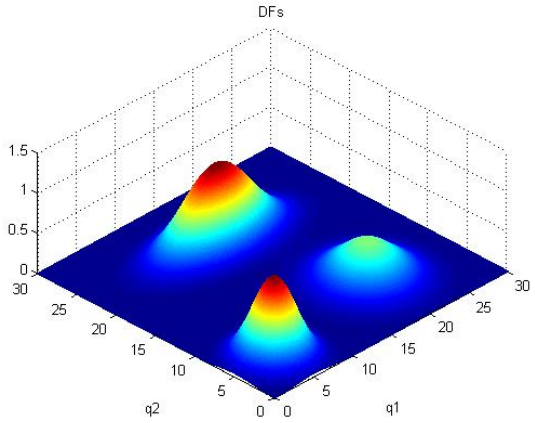


*Fig. 3.2:* Gaussian DFs

## 3.3 DF Sigmoidea

Una forma alternativa alla funzione descrittiva gaussiana si rende necessaria qualora occorra rappresentare un agente secondo una certa caratteristica che possiede un particolare *field of view*. Pensando ad esempio al caso di una videocamera in grado di poter ruotare su sé stessa, o al caso di un ecoscandaglio, è difficile trovare una qualche relazione fisica e grafica che leghi i parametri della DF gaussiana alle caratteristiche del sensore. E' stata così studiata ed elaborata in questa tesi una seconda tipologia di "*curve a campana*" che differisce dalla gaussiana per il fatto di possedere una direzione preferenziale. Queste nuove DF vengono realizzate attraverso la combinazione di funzioni gaussiane con funzioni sigmoidee (*Sigmoids*) o in alternativa dal solo utilizzo di quest'ultime.

La generica funzione Sigmoidea, talvolta chiamata anche "funzione logistica", può essere espressa per il caso monodimensionale nel seguente modo:

$$f(x) = \frac{1}{1 + e^{-K(x-p)}} \tag{3.2}$$

Dove i parametri principali che la caratterizzano sono $K$, che impone la pendenza, e $p$, che definisce invece il centro di simmetria. In figura 3.3 sono indicate alcune funzioni sigmoidee al variare del parametro $K$.

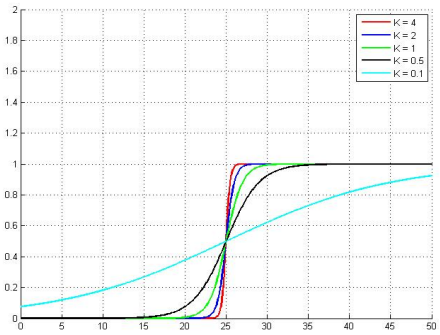


*Fig. 3.3:* Sigmoids with $p = 25$ and $K$ variable

.

Queste funzioni possono essere sfruttate per realizzare un'asimmetria nella DF Gaussiana, realizzando così un comportamento diverso della DF a seconda della direzione. A partire dal caso monodimensionale si può scrivere la DF Sigmoidea del generico agente $i-esimo$ come il prodotto di una funzione Sigmoids per una gaussiana:

$$D_i(p_i, q_1) = \frac{Ae^{-0.5\frac{(q_1-p_i)^2}{\sigma^2}}}{1+e^{-K(q_1-p_i)}} \tag{3.3}$$

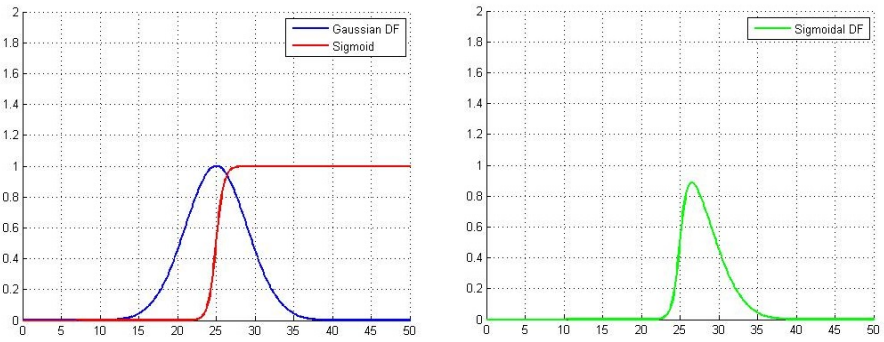


*Fig. 3.4:* Sigmoidal DF with $p = 25$, $K = 2$, $A = 1$, $\sigma = 4$

.

Il vettore di parametri necessario alla realizzazione della funzione può essere espresso come $\xi = [A, p_i, \sigma, K]$. Il risultato finale di questo prodotto è indicato in figura 3.4, in cui è possibile vedere come la nuova funzione descrittiva abbia una maggiore estensione delle sue capacità operative in una direzione piuttosto che nell'altra. Un secondo metodo per ottenere una funzione con queste proprietà é realizzato andando a moltiplicare due sole funzioni sigmoidee caratterizzate però da un andamento opposto: mentre una cresce all'aumentare del valore di $q_1$, l'altra deve diminuire. La DF è quindi ottenuta dall'espressione seguente:

$$D_i(p_i, q_1) = \frac{1}{1 + e^{-a_1(q_1 - p_{i_1})}} \frac{1}{1 + e^{-a_2(q_1 - p_{i_2})}} \tag{3.4}$$

Con:

- $a_1$ costante che impone la pendenza della prima funzione.
- $a_2$ costante che impone la pendenza della seconda funzione.
- $p_{i_1}$ che rappresenta il centro di asimmetria della prima funzione.
- $p_{i_2}$ che rappresenta il centro di asimmetria della seconda funzione.

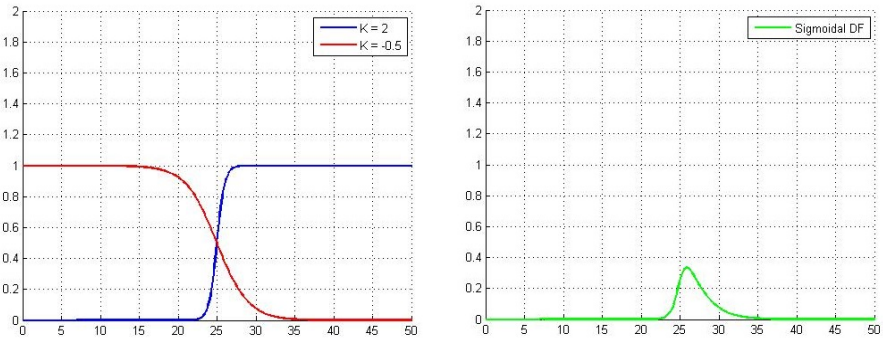


*Fig. 3.5:* The product of two sigmoids: the blue one with $a_1 = 2$, the red one with $a_2 = -0.5$

.

L'esempio illustrato in figura 3.5 rappresenta il risultato di un prodotto fra due funzioni che condividono il medesimo centro di simmetria in $p = 25$. Ovviamente è possibile anche utilizzare un valore di $p$ diverso per ciascuna funzione, ma in questo modo non si otterrebbe nessun tipo di vantaggio, mentre invece si aumenterebbe il numero di parametri necessario a realizzare la DF.

Come per il caso gaussiano, anche per entrambe le tipologie di DF di tipo Sigmoids é possibile aumentare la dimensione del problema. Per ciò che riguarda il caso bidimensionale, è importante introdurre fin da subito il concetto di *field of view* e indicare come questo possa venir realizzato attraverso il prodotto di due funzioni sigmoidee. Definito uno spazio operativo nelle sue due coordinate $q = [q_1, q_2]^T$ è possibile individuarne un semipiano di interesse (*Sigmoidal Plane*) attraverso la seguente espressione:

$$SP_R(\phi, p, q) = \frac{1}{1 + e^{-K\{(q_1 - p_{q1})cos(\phi) + (q_2 - p_{q2})sin(\phi)\}}} \tag{3.5}$$

In questa formula compare l'equazione 3.2 modificata per il caso in esame, più un parametro aggiuntivo $\phi$ che ne indica la rotazione destrogira intorno al punto $p = [p_{q1}, p_{q2}]^T$. Il risultato che si ottiene è espresso nella figura 3.6 in cui è indicato il piano di interesse descritto secondo i parametri $K = 2$, $\phi = 30$ e $p = [30, 30]^T$.

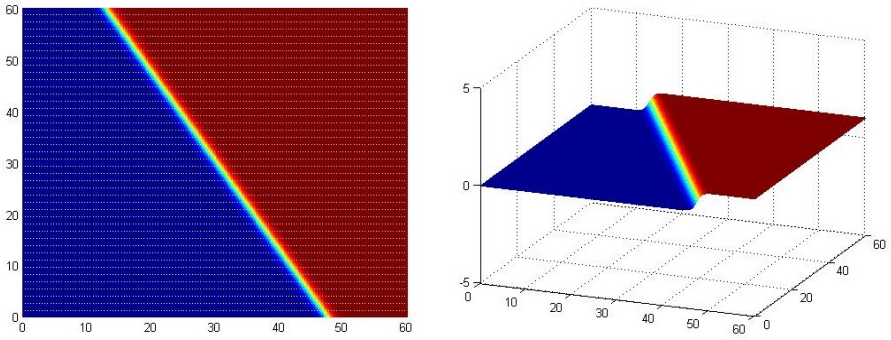


*Fig. 3.6:* The Sigmoidal Plane

Un secondo semipiano può essere ottenuto dall'equazione precedente effettuando una piccola variazione per rendere la sua rotazione intorno al punto $p$ sinistrogira. Si ottiene così:

$$SP_L(\phi, p, q) = \frac{1}{1 + e^{-K\{-(q_1 - p_{q1})cos(\phi) + (q_2 - p_{q2})sin(\phi)\}}} \tag{3.6}$$

L'intersezione di due piani, realizzati così come appena indicato, rende possibile la creazione di un campo angolare, o campo di vista, compreso fra $0 \div 180°$. Il valore della DF risulta quindi in maniera approssimativa diverso da zero solo all'interno di questa zona. La funzione descrittiva del *Field of View* diventa:

$$FoV(\phi, p, q) = SP_R(\phi, p, q) SP_L(\phi, p, q) \tag{3.7}$$

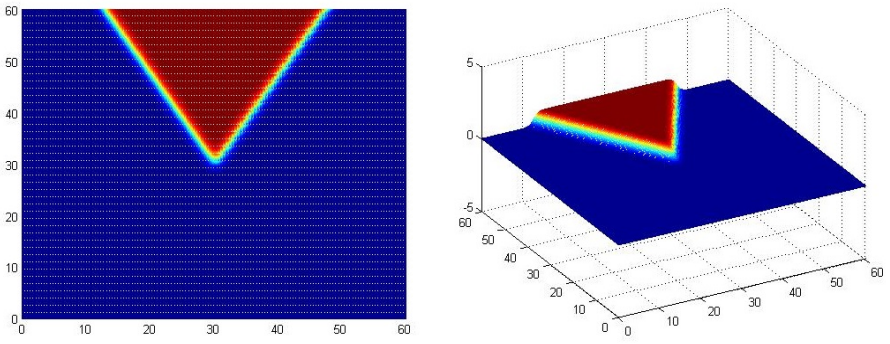


*Fig. 3.7:* The Field of View Function

L'esempio di figura 3.7 mostra l'intersezione fra due piani caratterizzati da $K = 2$ e da $\phi = 30°$; il *fov* é di $60°$ dato che la formula che lega la rotazione dei piani al *field of view* è la seguente:

$$\begin{cases} fov = 2\phi, & \text{se } 0 < \phi \leq \frac{\pi}{2} \\ fov = 2(\pi - \phi), & \text{se } \frac{\pi}{2} < \phi \leq \pi \\ fov = 2(\phi - \pi), & \text{se } \pi < \phi \leq \frac{3\pi}{2}, \text{ nel semipiano inferiore} \\ fov = 2(2\pi - \phi), & \text{se } \frac{3\pi}{2} < \phi \leq 2\pi, \text{ nel semipiano inferiore} \end{cases} \tag{3.8}$$

A partire dalla gaussiana bidimensionale definita nell'equazione 3.1 é ora possibile realizzare una DF con *field of view*. Si consideri infatti la seguente espressione ottenuta dalla moltiplicazione fra l'equazione della gaussiana e

quella appena trovata, che identifica una porzione ben precisa del piano di lavoro:

$$D_i(p_i, q) = FoV(\phi, p, q) * Ae^{-0.5(\frac{(q_1 - p_{ix})^2}{\sigma_x^2} + \frac{(q_2 - p_{iy})^2}{\sigma_y^2})} \tag{3.9}$$

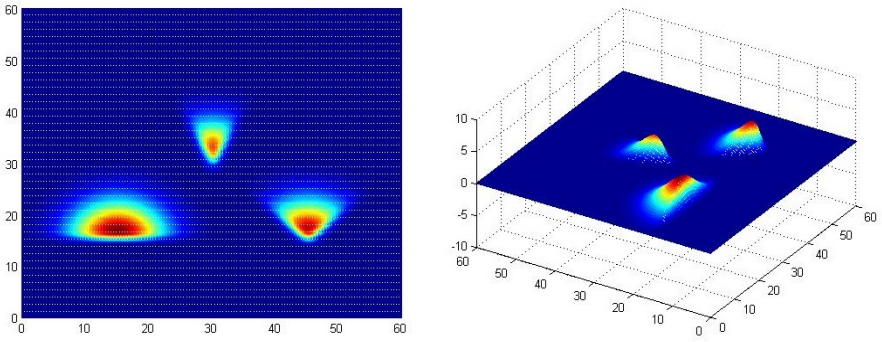

*Fig. 3.8:* The Field of View DFs

Alcune possibili realizzazioni di DF Sigmoidee sono quindi indicate nella figura 3.8. A partire da sinistra si può osservare una DF con $fov = 180°$, una con $fov = 50°$ e l'ultima con $fov = 90°$. In tutti e tre i casi è stato utilizzato $K = 2$. La modifica di quest'ultimo valore incide sulla pendenza dei due semipiani che si intersecano e all'aumentare di K la ripidità risulta più marcata.

Una seconda possibile realizzazione della DF bidimensionale con campo angolare, può essere ottenuta andando a sostituire nell'equazione 3.9 una funzione sigmoidea al posto della gaussiana, avente parametro di pendenza $K_2$ per cui $sign(K2) \neq sign(K)$. L'espressione che si ottiene ha però validità solo nel range $0° < fov < 180°$ e occorre anche tenere conto che le DFs che si creano degenerano graficamente quando $\phi \approx 0°, 90°, 180°$. Considerando queste limitazioni vale la sola parte di definizioni dell'equazione 3.8 compresa fra 0° e 180°. Si ottiene:

$$D_i(p_i, q) = FoV(\phi, p, q) * A\frac{1}{1 + e^{-K_2(q_2 - p_{q2})}} \tag{3.10}$$

Nella figura 3.9 è illustrato un confronto fra questi due modelli bidimensionali di DF. Si può vedere come quella originata da una funzione gaussiana

mantenga un campo circolare all'interno del suo field of view, mentre l'altra è costituita da una DF con linee di livello orizzontali. In entrambi i casi è stato scelto $\phi = 50°$ da cui si ottiene $fov = 100°$. Per i valori degli altri parametri si ha:

$FoV(\phi, p_{q1}, p_{q2}, K) = [50, 30, 30, 2]$
$DF_{Gaussian}(A, p_{q1}, p_{q2}, \sigma_{q1}, \sigma_{q2}) = [5, 30, 30, 6, 6]$
$DF_{Sigmoid}(A, p_{q2}, K_2) = [5, 30, -0.2]$

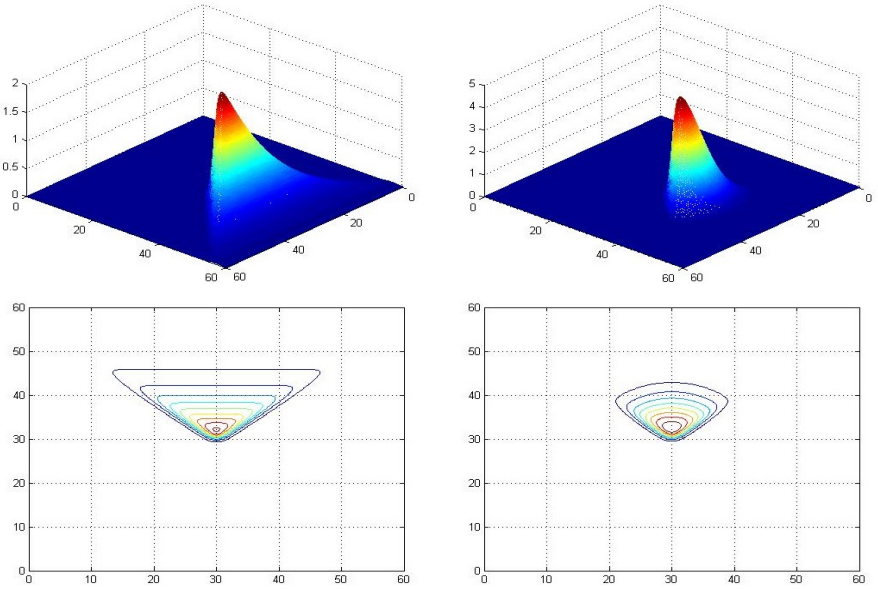

*Fig. 3.9:* The Field of View DFs: Only Sigmoids vs Sigmoid/Gaussian

Per semplificare la notazione le DF bidimensionali fin'ora proposte verranno chiamate nel seguente modo:

- $DF_G(p, q)$ la funzione descrittiva gaussiana ottenuta nel capitolo 3.2 di equazione 3.1.
- $DF_{SG}$ la funzione descrittiva con $fov$ ottenuta dalla moltiplicazione di una gaussiana con la $FoV(\phi, p, q)$.
- $DF_{SS}$ la funzione descrittiva con $fov$ ottenuta dalla moltiplicazione di una sigmoide con la $FoV(\phi, p, q)$.

## 3.4 Orientazione delle Funzioni Descrittive

Tutte le funzioni descrittive fino ad ora proposte ( vedi equazioni 3.1, 3.9 e 3.10) hanno l'inconveniente di non poter essere orientate liberamente nello spazio di lavoro. Se per il caso di DF gaussiana con $\sigma_x = \sigma_y$ ciò non apporterebbe nessun vantaggio (è una funzione omnidirezionale) per tutti gli altri casi questa modifica risulterebbe fondamentale. In questo modo, infatti, si andrebbe ad aggiungere una caratteristica fisica fondamentale agli agenti che li renderebbe molto più vicini a quello che è il comportamento di un veicolo/velivolo reale. Per questo motivo è stato scelto di aggiungere alle DF un ulteriore grado di libertà che potesse fornire il valore di orientazione rispetto al sistema di riferimento $Q = [q_1, q_2]^T$. La DF può quindi essere riscritta con l'aggiunta del parametro $\theta$ che indica quindi l'angolo di rotazione fra il sistema di riferimento "inerziale" $Q = [q_1, q_2]^T$ ( che descrive i punti dell'*environment* ) e il sistema di riferimento "body" posto sopra l'agente. Facendo riferimento alla figura 3.10 si può scrivere la seguente espressione generale che indica la relazione di rotazione fra terne:

$$p_i^{\{Q\}} = [C_B^Q] p_i^{\{B\}}$$

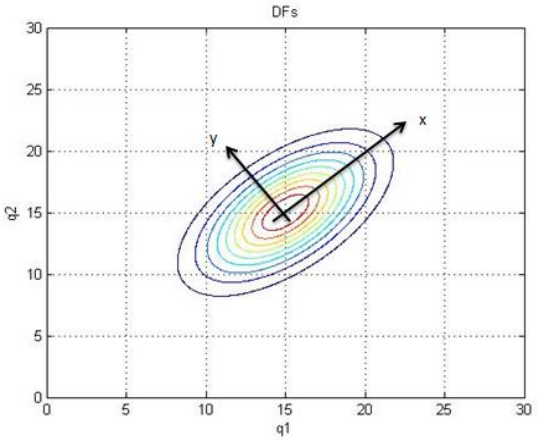


*Fig. 3.10:* DF with rotation parameter

Dove $p_i^{\{Q\}}$ è il punto espresso nella terna inerziale, $p_i^{\{B\}}$ è il punto espresso nella terna body e $C$ è la matrice di rotazione che indica quanto è ruotato il riferimento $\{B\}$ rispetto a $\{Q\}$. Nel caso bidimensionale vale che:

$$C_B^Q = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \longrightarrow C_B^Q = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Per ruotare la DF occorre quindi effettuare il seguente cambio di coordinate dove con $[q_i - p_i]'$ è indicato il valore aggiornato da sostituire all'interno della DF:

$$\begin{bmatrix} [q_1 - p_{q1}]' \\ [q_2 - p_{q2}]' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} [q_1 - p_{q1}] \\ [q_2 - p_{q2}] \end{bmatrix}$$

In altre parole nelle formule 3.1, 3.9 e 3.10 indicate precedentemente, occorrerà sostituire il valore:

$$\begin{cases} (q_1 - p_{q1}) \longrightarrow +(q_1 - p_{q1})cos(\theta) + (q_2 - p_{q2})sin(\theta) \\ (q_2 - p_{q2}) \longrightarrow -(q_1 - p_{q1})sin(\theta) + (q_2 - p_{q2})cos(\theta) \end{cases} \tag{3.11}$$

Per ciascuna funzione descrittiva bidimensionale gaussiana è ora possibile riscrivere la forma finale così come indicato qui di seguito:

$$DF_G(p,q) = Ae^{-0.5(\frac{((q_1 - p_{i_x})\cos\theta + (q_2 - p_{i_y})\sin\theta)^2}{\sigma_x^2} + \frac{(-(q_1 - p_{i_x})\sin\theta + (q_2 - p_{i_y})\cos\theta)^2}{\sigma_y^2})} \tag{3.12}$$

La solita formula in una notazione più compatta può essere riscritta come segue:

$$D_i(p_i,q) = Ae^{-0.5[K_1(q_1 - p_{i_x})^2 + K_2(q_1 - p_{i_x})(q_2 - p_{i_y}) + K_3(q_2 - p_{i_y})^2]} \tag{3.13}$$

$$\begin{cases} K_1 = \frac{\sin\theta^2}{\sigma_x^2} + \frac{\cos\theta^2}{\sigma_y^2} \\ K_2 = \frac{\sin 2\theta}{\sigma_x^2} - \frac{\cos 2\theta}{\sigma_y^2} \\ K_3 = \frac{\cos\theta^2}{\sigma_x^2} + \frac{\sin\theta^2}{\sigma_y^2} \end{cases} \tag{3.14}$$

Passando all'analisi del caso delle DF con FoV occorre prima di tutto ridefinire la funzione descrittiva del FoV in funzione del parametro $\theta$, ricordando poi che, data la simmetria del problema, il valore della gaussiana nella $DF_{SG}(\phi, p, q)$ non viene messo in funzione di tale parametro:

$$DF_{SG}(\phi, p, q, \theta) = FoV(\phi, p, q, \theta) * Ae^{-0.5(\frac{(q_1 - p_{i_x})^2}{\sigma_x^2} + \frac{(q_2 - p_{i_y})^2}{\sigma_y^2})} \tag{3.15}$$

$$DF_{SS}(\phi, p, q, \theta) = \frac{FoV(\phi, p, q, \theta) * A}{1 + e^{-K_2(-(q_1 - p_{q1})sin(\theta) + (q_2 - p_{q2})cos(theta))}} \tag{3.16}$$

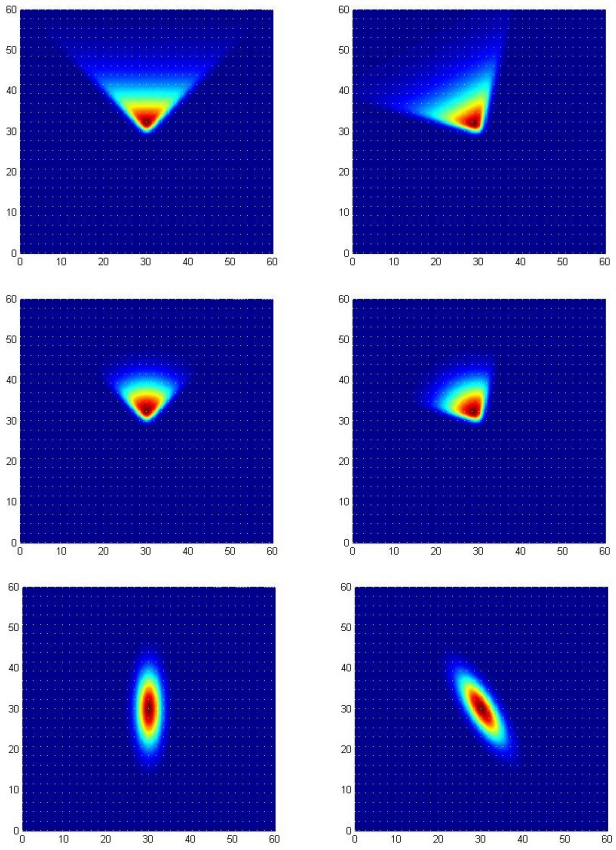

*Fig. 3.11:* DFs rotated by 30°

Nelle immagini di figura 3.11 è illustrato il comportamento delle DF dopo un rotazione di 30°. Le funzioni descrittive sono realizzate attraverso i seguenti parametri, dove si ricorda che $fov = 2\phi$:

$DF_G(A, p_{q1}, p_{q2}, \sigma_{q1}, \sigma_{q2}, \theta) = [4, 30, 30, 2, 6, 2, 30°]$
$DF_{SG}(A, p_{q1}, p_{q2}, \sigma_{q1}, \sigma_{q2}, \theta, \phi, K) = [5, 30, 30, 6, 6, 30°, 45°, 2]$
$DF_{SS}(A, p_{q1}, p_{q2}, \theta, \phi, K, K_2) = [5, 30, 30, 30°, 45°, 2, -0.2]$

I vantaggi operativi che si ottengono nell'utilizzo delle DFs così modificate sono notevoli rispetto allo stato precedente del framework. Un primo esempio pratico (altri saranno fatti nei prossimi capitoli) può essere fatto osservando come l'aggiunta dell'orientazione renda più naturale la risoluzione del problema raffigurato in 3.12.

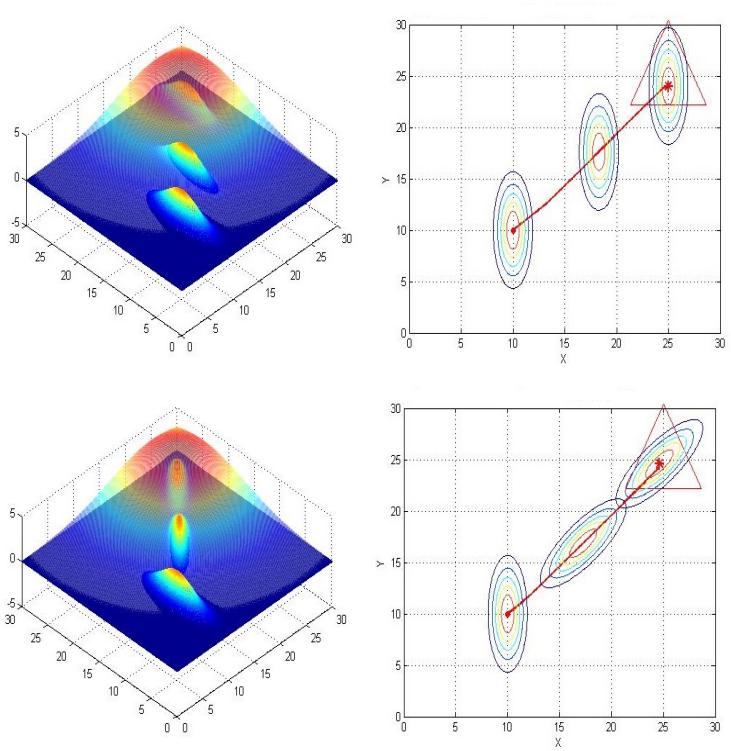

*Fig. 3.12:* DF with and without Attitude Parameter

Un agente, rappresentato da una $DF_G$, si muove verso la zona con maggior richiesta di risorse indicata dalla DF trasparente; pur rimanendo identico

il percorso seguito senza la possibilità di ruotare l'agente si muove traslando lateralmente utilizzando una parte della sua DF che non è fortemente attrattiva (il movimento risulta più lento). Con l'aggiunta del parametro $\theta$ è invece possibile vedere come l'agente prima si orienti e poi punti direttamente alla zona che deve essere raggiunta sfruttando la parte di DF con caratteristiche migliori. A conclusione di queste sezioni è possibile quindi dire che il comportamento di un agente nel risolvere un determinato task può essere rappresentato secondo tre differenti leggi. Quando l'agente è considerato per una sua risorsa di tipo omni-direzionale è possibile usare la DF gaussiana semplice, ovvero la $DF_G$; quando invece occorre specificare un determinato "field of view" è possibile utilizzare una delle due DF sigmoidee, in relazione al tipo di linee di livello che meglio approssimano il comportamento dell'agente. In tutti questi casi è poi sempre possibile specificare un valore esatto di orientazione rispetto al riferimento inerziale dato da $Q$.

## 3.5 Sviluppo del Framework: TDF, cTDF e TEF

Nel portare a termine una missione un qualunque agente dello sciame può trovarsi a dover eseguire più task differenti. Se ciascuna DF descrive le qualità o le risorse possedute dall'agente $i-esimo$ per l'esecuzione di un singolo compito è possibile considerare il seguente vettore che tiene conto delle diverse DF necessarie a portare a termine la missione completa (composta da $m$ task):

$$d_i(p_i, q) = \begin{bmatrix} d_i^1(p_i, q) \\ d_i^2(p_i, q) \\ \vdots \\ d_i^k(p_i, q) \\ \vdots \\ d_i^m(p_i, q) \end{bmatrix} = \begin{bmatrix} f_1(p_x, p_y, A, \theta, \sigma_x, \sigma_y) \\ f_2(p_x, p_y, A, \theta, \sigma_x, \sigma_y, \phi, K) \\ \vdots \\ f_k(p_x, p_y, A, \theta, \phi, K, K_2) \\ \vdots \\ f_m(p_x, p_y, A, \theta, \sigma_x, \sigma_y) \end{bmatrix}$$

La singola $d_i^k(p_i, q)$, che da ora in poi verrà chiamata *Agent Descriptor Function* o *ADF* (relativamente al task considerato), assumerà forma gaussiana o sigmoidea nel caso in cui l'agente sarà in grado di eseguire quel determinato task, altrimenti assumerà valore nullo. Considerando già una prima parametrizzazione della funzione è possibile riscrivere il vettore delle *ADFs* come indicato a destra dell'uguale.

L'intero sciame può poi essere suddiviso in squadre, in maniera tale che ogni *team* risulti composto da tutti gli agenti in grado di portare a termine un determinato task. In questo modo, una missione composta da $M$ task porterebbe al costituirsi di $M$ team in cui un singolo agente può prendere parte a uno o più di questi sottoinsiemi. Definito il set di task che l'agente $i-esimo$ sta eseguendo al tempo $t$ con $T_i(t)$, il Team che sta eseguendo il generico task $k-esimo$ può essere formalizzato come segue, dove con $V_i$ è indicato il generico agente:

$$T^k = \{V_i : k \in T_i(t)\} \tag{3.17}$$

Nel proseguo della tesi (vedi capitolo 7) verrà indicato come sarà possibile sfruttare queste definizioni per risolvere i diversi task che compongono una missione. Ripensando il team come un insieme di $DF$ e non più come ad un semplice gruppo di agenti, è possibile descrivere lo stato attuale dello swarm, in relazione al completamento di un singolo task, come una sommatoria di $DF$. Si viene così a definire la *Current Task Descriptor Function* o *cTDF* esprimibile come segue:

$$D_i(p_i, q) = \sum_{V^i \in T^k} d_i^k(p_i, q) \tag{3.18}$$

La cTDF è quindi una misura dell'insieme delle risorse possedute dallo swarm per eseguire un task ed in termini sensoriali si può ad esempio parlare di *Sensor Intensity Field*.

A completamento di quanto detto finora è possibile dare una formalizzazione in termini di DF anche per quanto riguarda l'espressione del task da eseguire. Intuitivamente per ogni task è possibile pensare ad una DF che rappresenti la richiesta di risorse in un determinato punto dell'*environment* in analogia a quanto fatto sulle *ADFs*.

Questa funzione, che da ora in poi verrà chiamata *Desired Task Descriptor Function*, o *dTDF*, definisce quindi come e dove devono distribuirsi gli agenti per portare a compimento il task assegnato. Definendo l'ambiente operativo bidimensionale $Q = [q_1, q_2]^T$, si può esprimere la *dTDF* con la seguente espressione che indica come tale funzione debba sempre risultare un numero positivo:

$$d_*^k(q, t) := Q \longrightarrow \Re^+, k = 1, 2, \ldots, N_T \tag{3.19}$$

Dove $N_T$ è il numero totale di task con cui è realizzata la missione. Alcuni esempi possibili di task da eseguire, rappresentati con il framework delle funzioni descrittive, sono indicati nel capitolo successivo e riprendono buona parte delle problematiche studiate in letteratura (Uniform Deployment,Static Coverage, Dynamic Coverage, etc ...).

La funzione di errore, o *Task Error Function* ( *TEF* ), serve a misurare la differenza fra la dTDF e la cTDF in ciascun punto dell'ambiente operativo. Tale funzione può essere valutata in alcuni modi differenti pur mantenendo inalterata la proprietà per cui, laddove è presente una forte richiesta di risorse, questa dovrà avere un valore maggiore rispetto alle zone in cui la richiesta è più piccola.

- **TEF Assoluta**. La TEF viene calcolata come la differenza algebrica della dTDF con la cTDF. In questo modo si ottiene una misura della mancanza di risorse quando il valore calcolato è positivo, mentre si ottiene una misura dell'eccesso quando il valore calcolato è negativo. Considerando il generico task $k-esimo$ si ottiene:

$$e_A^k(p,q,t) = d_*^k(q,t) - d^k(q,t)$$

- **TEF Assoluta con** $\max(\bullet)$. Volendo riconsiderare la funzione la TEF assoluta, non penalizzando la presenza di un eccesso di risorse, è possibile considerare la seguente variante, per cui si ottiene:

$$e_A^k(p,q,t) = \max(0, d_*^k(q,t) - d^k(q,t))$$

- **TEF Relativa**. In questa soluzione la TEF viene calcolata come il rapporto fra la dTDF e la cTDF, risultando così un valore sempre semi-definito positivo. In questo modo la TEF non rappresenta più il valore esatto di richiesta di risorse nell'ambiente, ma costituisce comunque una funzione per cui è possibile capire dove, in un determinato istante, c'è una richiesta maggiore o minore:

$$e_R^k(p,q,t) = \frac{d_*^k(q,t)}{d^k(q,t) + c}$$

Ovviamente la scelta di una TEF piuttosto che un'altra comporterà anche l'ottenimento di risultati diversi. E' stato scelto però ti affrontare tali problematiche più avanti, dove si parlerà direttamente del loro impiego nella legge di controllo.

# 4. CONTROLLO CENTRALIZZATO

## *4.1 Introduzione*

In questo capitolo viene affrontato e risolto il problema del controllo dello sciame utilizzando il framework delle funzioni descrittive. A differenza di quanto verrà fatto nel capitolo 5, qui di seguito si ipotizzerà che ogni agente sia sempre in piena conoscenza, in ogni istante di tempo, di tutte le informazioni necessarie per implementare gli algoritmi proposti: con queste ipotesi l'architettura di controllo risulterà quindi centralizzata. Il problema viene quindi risolto attraverso l'ottimizzazione di un opportuno funzionale di costo per cui, utilizzando un algoritmo di tipo discesa del gradiente, verrà realizzato il moto dell'agente all'interno dell'envirnoment. La dinamica del singolo veicolo, così come spesso accade in letteratura, viene semplificata con l'utilizzo di un singolo integratore per cui l'ingresso è un valore di velocità, mentre l'uscita definisce il nuovo valore di posizione.

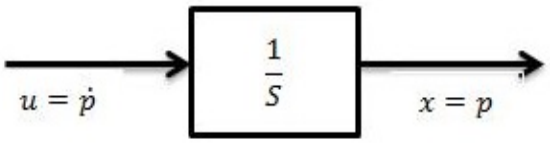


*Fig. 4.1:* Single Integrator Dynamic

Alla legge di controllo verrà poi aggiunto un semplice meccanismo in grado di garantire *Obstacle Avoidance* sia per quello che riguarda le collisioni con ostacoli fissi sia per i possibili scontri agente/agente.

## *4.2 La legge di controllo*

Tenendo conto di quanto descritto nella sezione 3.5, è ora possibile pensare ad una soluzione del problema di controllo facendo muovere gli agenti in modo

da minimizzare una delle espressioni di errore proposte. Questa idea viene realizzata nel seguente modo in cui il valore della TEF è inserito all'interno di un opportuno funzionale di costo:

$$J^{T_k} = \int_Q f(e^{T_k}(p,q)\sigma(q)) \tag{4.1}$$

Dove:

- $f(x)$ è una funzione limitata, continua e semi-definita positiva su tutto $Q$ e $\forall x \in \Re$.
- $\sigma(q)$ è una funzione peso che serve per concentrare l'attenzione su alcune aree di $Q$.
- $p(t)$ è il vettore che indica la posizione del veicolo e la rotazione $\theta$ della DF; nel caso bidimensionale con $Q = [q_1, q_2]^T$ si ha per il singolo veicolo $i$ $p_i(t) = [p_{q_1}(t), p_{q_2}(t), \theta(t)]^T$ o alternativamente $p_i(t) = [p_x(t), p_y(t), \theta(t)]^T$

La soluzione del problema viene quindi effettuata in relazione al vettore $p(t)$ che è il macro-vettore contenente tutti i $p_i(t)$ appartenenti ai veicoli. Nel caso di uno sciame composto da $N$ agenti si ha:

$$p(t) = \begin{bmatrix} p_1(t) \\ p_2(t) \\ \vdots \\ p_N(t) \end{bmatrix} = \begin{bmatrix} p_{1_x}(t) \\ p_{1_y}(t) \\ \theta_1(t) \\ \vdots \\ p_{N_x}(t) \\ p_{N_y}(t) \\ \theta_N(t) \end{bmatrix}$$

Utilizzando un approccio di tipo "algoritmo del gradiente", che viene solitamente usato per la ricerca di minimi su funzioni non-lineari, è possibile valutare la "posizione" ottima $\bar{p}$ che l'agente deve occupare:

$$\bar{p} = \arg\min_{p \in T_k} J^{T_k}(p) \tag{4.2}$$

Il funzionale $J(\cdot)$ ha quindi il compito di mappare la funzione di errore con un numero reale positivo che indica quanto è lontana la copertura desiderata. In questo modo l'azzeramento del funzionale è raggiunto solo quando sussiste una perfetta coincidenza fra le due coperture (reale e desiderata). Considerando un controllore che sfrutta l'algoritmo del gradiente è possibile arrivare alla seguente legge ( per il generico agente $i$ ), dove con $\beta$ si indica il guadagno della legge di controllo:

$$u_i = \dot{p}_i = -\beta \frac{\partial J(p)}{\partial p_i} = -\beta \int_Q \frac{\partial f(e^{T_k}(p,q))}{\partial p_i} \sigma(q) dq \tag{4.3}$$

Una prima proprietà di questa legge è la convergenza verso un punto critico, facilmente dimostrabile dalla seguente applicazione della *chain-rule*, la cui dimostrazione può essere provata attraverso la teoria di Lyapunov usando la $J^{T_K}(p)$ come funzione candidata.

$$\dot{J}(p,q) = \frac{\partial J(p,q))}{\partial t} = \frac{\partial J(p,q))}{\partial p} \frac{\partial p}{\partial t} = \frac{\partial J(p,q))}{\partial p} \dot{p} = -\beta (\frac{\partial J(p,q))}{\partial p})^2 \leq 0$$

Per quanto riguarda invece la stabilità della soluzione $\bar{p}$, è possibile dire che tutti i punti critici sono allo stesso modo raggiungibili: a seconda delle condizioni iniziali sono quindi raggiungibili minimi locali, globali e anche punti di sella. Si può facilmente riscontrare che fra questi solo i minimi sono equilibri stabili e robusti alle perturbazioni ( basta osservare la legge di controllo ). Per assicurarsi quindi di aver raggiunto un minimo è sempre opportuno andare a sollecitare l'agente che si è fermato con una qualche perturbazione esterna. In ogni caso il limite inferiore potrà essere raggiunto solo per una corretta definizione della dTDF, della cTDF e solo quando non si entra in dei minimi locali da cui non si può uscire. Per quanto riguarda la scelta della $f(\cdot)$, questa deve essere fatta considerando funzioni almeno semi-definite positive. Tenendo conto di quanto scritto in 3.5 in questa tesi è stato scelto di utilizzare la legge di controllo realizzata a partire da un errore di tipo assoluto con $max(\cdot)$. Questa scelta è anche motivata da quanto dimostrato nella tesi di dottorato [2] per ciò che riguarda la volontà di non penalizzare l'eccesso di risorse, e per il fatto che gli effetti di una legge di controllo di questo tipo possono essere visti come il risultato di una forza attrattiva esercitata sull'agente.

$$J(p) = \int_Q max(0, e_A(p,q))^2 \sigma(q) dq \tag{4.4}$$

Con questo accorgimento vengono così eliminati un certo numero di minimi locali dovuti all'equilibrio fra forze repulsive e forze attrattive, il tutto a discapito di un rallentamento nella convergenza dell'algoritmo, causato da tutte quelle zone di $Q$ in cui ora il funzionale di costo risulta nullo (vedi [2]). La legge di controllo si modifica nel seguente modo:

$$u_i(t) = \dot{p} = -\beta \frac{\partial J(p,q)}{\partial p} = 2\beta \int_Q max(0, e_A(p,q)) \frac{\partial d_i(p,q)}{\partial p_i} \sigma(q) dq \tag{4.5}$$

Una legge di controllo che deriva dall'utilizzo dell'errore assoluto può anche essere disaccoppiata in due contributi. Preso per semplicità il caso con $\sigma(q) = 1$, quando la funzione $max(\cdot) > 0$ si ottiene:

$$u_i = 2\beta \int_Q d_*(q) \frac{\partial d_i(p,q)}{\partial p_i} dq - 2\beta \int_Q \sum_{i=1}^{N} d_i(p,q) \frac{\partial d_i(p,q)}{\partial p_i} dq \tag{4.6}$$

Per cui è possibile suddividere l'ingresso $u_i$ in:

$$u_i = 2\beta(u_i^{(1)} + u_i^{(2)}) \tag{4.7}$$

$$u_i^{(1)} = \int_Q d_*(q) \frac{\partial d_i(p,q)}{\partial p_i} dq$$

$$u_i^{(2)} = -\int_Q \sum_{i=1}^{N} d_i(p,q) \frac{\partial d_i(p,q)}{\partial p_i} dq$$

Dove i due contributi al moto possono essere interpretati come segue:

- $u_i^{(1)}$ è il termine che spinge l'agente a muoversi in direzione del contributo massimo che può dare al task. In altre parole l'agente si muove in modo da massimizzare il prodotto fra la sua DF e la TDF.

- $u_i^{(2)}$ è il termine che spinge l'agente ad allontanarsi da posizioni dell'environment già ricoperte da altri. Il movimento viene fatto in modo da minimizzare il prodotto fra la propria DF e la DF di un qualunque altro agente. Tale contributo in ogni caso non garantisce un meccanismo anti-collisione soddisfacente. Nel caso in cui $u_i^{(1)}$ è nella stessa direzione di $u_i^{(2)}$ e vale $|u_i^{(1)}| > |u_i^{(2)}|$, la collisione non potrà mai essere evitata.

## 4.3 Obstacle Avoidance

Per risolvere il problema di *Obstacle Avoidance* è stato scelto di utilizzare un meccanismo derivato dalla teoria dei *Potential Field* di cui in letteratura si possono trovare diversi riferimenti (vedi ad esempio [12] e [13]). Partendo da quanto affermato nel capitolo 4.2, dove il moto dell'agente può essere visto come il risultato dell'effetto di un campo attrattivo, allo stesso modo si è pensato di creare un campo repulsivo radiale per ogni elemento dell'environment.

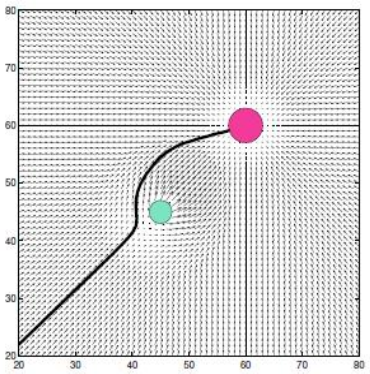


*Fig. 4.2:* Agent behaviour under the effect of an attractive and a repulsive Potential Field

Si consideri da principio il problema cinematico e si ipotizzi nota la posizione di tutti gli ostacoli fissi e di tutti gli altri agenti (ostacoli mobili). Per prima cosa occorre definire un raggio che rappresenti la distanza di sicurezza oltre cui bisogna muoversi per evitare la collisione. In questo modo il meccanismo proposto verrà utilizzato solo ed esclusivamente al superarsi della distanza minima $r$. Mantenendo un approccio di tipo conservativo ogni

agente vedrà gli altri elementi presenti nello scenario come illustrato in figura 4.3.

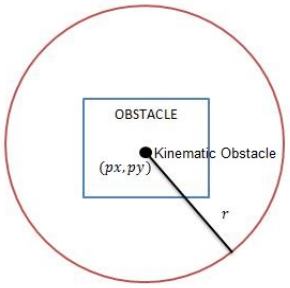


*Fig. 4.3:* Obstacle radius

La realizzazione di un campo di tipo repulsivo radiale passa attraverso l'utilizzo dei seguenti parametri:

- $d$ distanza fra agente e ostacolo.
- $\gamma$ angolo compreso fra l'agente e l'ostacolo.
- $r$ raggio di sicurezza.

Dove $r > 0$ è deciso «*a-priori*», mentre $\gamma$ e $d$ sono calcolati in funzione della posizione attuale dell'agente $p_A = [p_{x_A}, p_{y_A}]^T$ e dell'ostacolo $p_O = [p_{x_O}, p_{y_O}]^T$:

- $d = \sqrt{(p_{x_O} - p_{x_A})^2 + (p_{y_O} - p_{y_A})^2}$
- $\gamma = atan2(\frac{(p_{y_O} - p_{y_A})}{(p_{x_O} - p_{x_A})})$

Occorre ricordare che la parte legata alla orientazione della funzione descrittiva $\theta$ non influisce sul problema, dato che non modifica il valore di posizione dell'agente, che è invece l'unico dato di interesse quando si vuole fare *obstacle avoidance* usando i Potential Fields.

La legge di controllo $u_i$, che da ora in poi verrà indicata con il vettore $\bar{u}_{ott}$ (ad indicare il risultato del problema di ottimizzazione), viene quindi modificata secondo un termine additivo $\bar{u}_{obs} = [u_{obs_x}, u_{obs_y}]^T$ calcolato nel seguente modo:

$$u_{obs_x} = \begin{cases} 0, & \text{if } d > r \\ -K(r-d)\cos\gamma, & \text{if } d \leq r \end{cases}$$

$$u_{obs_y} = \begin{cases} 0, & \text{if } d > r \\ -K(r-d)\sin\gamma, & \text{if } d \leq r \end{cases}$$

$$\text{if } d \leq r : |\bar{u}_{obs}| = \sqrt{(-K(r-d)\cos\gamma)^2 + (-K(r-d)\sin\gamma)^2} = K(r-d)$$

Dove $\bar{u}_{obs}$ è diretta come la $\bar{u}_{ott}$, ma con verso opposto, e $K$ è un guadagno utile a definire l'intensità della forza repulsiva che dovrà avere un ordine di grandezza tale da potersi comparare con la $\bar{u}_{ott}$ . Una possibile scelta di $K$ può essere fatta considerando una distanza minima $r_{min} < r$ dove si vuole ottenere $|\bar{u}_{ott}| = |\bar{u}_{obs}|$:

$$K = \frac{|\bar{u}_{ott}|}{(r - r_{min})} \tag{4.8}$$

E per i singoli contributi vale:

$$\text{if } d \leq r \begin{cases} u_{obs_x} = -\frac{|\bar{u}_{ott}|}{(r-r_{min})}(r-d)\cos\gamma \\ u_{obs_y} = -\frac{|\bar{u}_{ott}|}{(r-r_{min})}(r-d)\sin\gamma \end{cases} \tag{4.9}$$

La legge di controllo finale sarà a questo punto definita nel seguente modo considerando tutti gli ostacoli $N_O$ di cui è stata superata la barriera di sicurezza $r$:

$$\bar{u}_{TOT} = \bar{u}_{ott} + \bar{u}_{obs} \longrightarrow \begin{bmatrix} u_{TOT_x} \\ u_{TOT_y} \end{bmatrix} = \begin{bmatrix} u_{ott_x} \\ u_{ott_y} \end{bmatrix} + \sum_{i=1}^{N_O} \begin{bmatrix} u_{obs_x} \\ u_{obs_y} \end{bmatrix}_i \tag{4.10}$$

Una soluzione di questo tipo comporta però la presenza di casi grigi in cui si possono presentare o delle situazioni di arresto inaspettate, dovute all'aggiunta del nuovo termine $\bar{u}_{obs}$ (minimi locali), o situazioni in cui il problema di obstacle avoidance viene ignorato.

- **Arresto per Simmetria del Problema**. E' la situazione che si viene a creare quando si presenta una qualunque combinazione dei vettori $\bar{u}_{ott} \neq 0$ e $\sum_{i=1}^{N_O} \bar{u}_{obs_i} \neq 0$ che dà come risultato finale $\bar{u}_{TOT} = 0$.
- **Problema dell'Annullamento del termine** $\bar{u}_{obs}$. E' la situazione che si viene a creare quando, in presenza di più ostacoli che interagiscono con l'agente, si ottiene $\sum_{i=1}^{N_O} [u_{obs_x}, u_{obs_y}]_i^T = 0$.

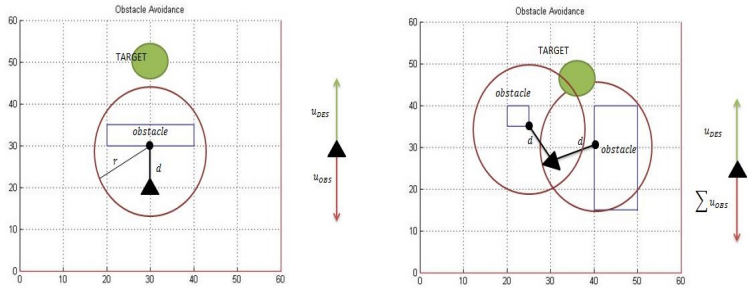


*Fig. 4.4:* Stop due to simmetry $\bar{u}_{ott} = -\bar{u}_{obs}$

La soluzione al secondo problema può essere facilmente trovata considerando di volta in volta il solo ostacolo più vicino, ovvero quello con $d$ più piccola. In questo modo si elimina il termine di sommatoria e si ottiene:

$$\bar{u}_{TOT} = \bar{u}_{ott} + \bar{u}_{obs} \longrightarrow \begin{bmatrix} u_{TOT_x} \\ u_{TOT_y} \end{bmatrix} = \begin{bmatrix} u_{ott_x} \\ u_{ott_y} \end{bmatrix} + \begin{bmatrix} u_{obs_x} \\ u_{obs_y} \end{bmatrix} \tag{4.11}$$

Nella suddetta equazione è però ancora presente il problema legato all'arresto per simmetria, che può essere evitato andando ad aggiungere un termine in grado di modificare il risultato nel caso in cui $u_{TOT_x} = 0$ in presenza di $\bar{u}_{ott} = -\bar{u}_{obs}$. Quello che si vuole quindi ottenere è che $u_{TOT_x}$ si annulli solo al presentarsi della condizione $\bar{u}_{ott} = \bar{u}_{obs} = 0$. Una scelta plausibile è la seguente:

$$\bar{u}_{TOT} = R(\alpha)\bar{u}_{ott} + \bar{u}_{obs} \tag{4.12}$$

$$\bar{u}_{TOT} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix} \begin{bmatrix} u_{ott_x} \\ u_{ott_y} \end{bmatrix} + \begin{bmatrix} u_{obs_x} \\ u_{obs_y} \end{bmatrix} \tag{4.13}$$

Dove l'angolo di rotazione $\alpha$ deve essere preso in modo da risultare diverso da zero all'avvicinarsi della condizione $\bar{u}_{ott} = -\bar{u}_{obs}$. Una scelta di tipo "*on-off*", che realizza quanto voluto, può prevedere:

$$\alpha = \begin{cases} \bar{\alpha}, & \text{if } \bar{u}_{ott} = -\bar{u}_{obs} \\ 0, & \text{otherwise} \end{cases}$$

Con quest'ultima modifica alla legge di controllo l'arresto dell'agente viene limitato a quelle situazioni in cui $\bar{u}_{ott} = 0$, che si presentano solo in corrispondenza di un minimo locale o globale ottenuto dal problema di minimizzazione di $J(p, q)$. Occorre poi ricordare che, con l'aggiunta del nuovo contributo $\bar{u}_{obs}$, non è più garantito, in ogni istante di tempo, il movimento dell'agente verso le direzioni indicate dall'algoritmo del gradiente. In ogni caso però, una volta risolto il problema incombente di Obstacle Avoidance, l'agente può tornare a muoversi verso la direzione corretta per l'esecuzione del task.

L'ultimo problema da affrontare, prima di rendere il meccanismo completamente realizzabile in pratica, è legato alla rappresentazione scenica dell'ostacolo. Se un agente può infatti essere banalmente considerato dal punto di vista cinematico come un punto materiale di posizione $p_A = [p_{x_A}, p_{y_A}]^T$ e raggio di sicurezza $r$, tale espediente non sempre garantisce una rappresentazione corretta per un qualunque ostacolo ambientale. Guadando ad esempio la figura 4.3 si nota come, prendendo $p_A = [p_{x_O}, p_{y_O}]^T$ baricentrici ed un raggio $r$, la distanza reale fra l'ostacolo e il cerchio di sicurezza non è sempre costante e ci sono punti in cui a parità di repulsione l'ostacolo può risultare più vicino. Una prima soluzione a questo problema può essere effettuata rendendo il problema "fortemente conservativo" ed utilizzando quindi un primo cerchio, definito da $r_{min}$, che circonda completamente l'ostacolo e un secondo, di raggio $r$, che definisce la distanza da cui occorre iniziare a considerare $\bar{u}_{obs}$. Una seconda soluzione, che prende invece in considerazione la forma reale dell'ostacolo (ottenuta attraverso un poligono convesso di cui si conoscono tutti gli $N$ vertici $V_i$), si sviluppa come indicato qui di seguito. Per prima cosa si calcola il baricentro $(x_C, y_C)$ come segue:

$$x_{C_O} = \frac{x_{V_1} + x_{V_2} + \cdots + x_{V_N}}{N}$$

$$y_{C_O} = \frac{y_{V_1} + y_{V_2} + \cdots + y_{V_N}}{N}$$

Dopodiché si considera l'intero perimetro dell'ostacolo e si controlla quando l'agente si avvicina ad una distanza minore di $r$ da uno dei punti che vi appartengono ( per effettuare un controllo di questo tipo si può ad esempio utilizzare l'algoritmo espresso dalla funzione Matlab "$p\_poly\_dist.m$", realizzata da Michael Yoshpe e revisionata da Eric Schmitz nel 2008). A questo punto si può proseguire come già visto calcolando però l'angolo $\gamma$ in relazione all'orientazione con il baricentro $(x_C, y_C)$, e l'intensità della forza in relazione alla distanza $d$ con il punto del più vicino del perimetro.

- $d = \sqrt{(p_{x_{per}} - p_{x_A})^2 + (p_{y_{per}} - p_{y_A})^2}$
- $\gamma = atan2(\frac{(y_{C_O} - p_{y_A})}{(x_{C_O} - p_{x_A})})$

Quest'ultima soluzione permette l'aggiramento degli ostacoli statici e degli altri agenti come illustrato nelle figure 4.5 dove è evidente il mantenimento della stessa distanza dai bordi dell'ostacolo fisso.

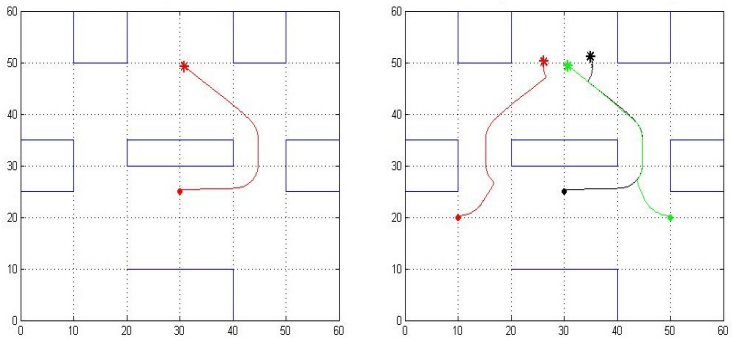


*Fig. 4.5:* Obstacle avoidance trajectory with $\bar{\alpha} = 45°$

Dal punto di vista teorico occorre infine ricordare che la soluzione adottata di tipo $\bar{u}_{TOT} = \bar{u}_{ott} + \bar{u}_{obs}$ può essere vista come la diretta applicazione del metodo di Lyapunov, per cui è data una funzione $V \geq 0$ da cui si vuole trarre un controllo $u$ che definisce $\dot{V} \leq 0$. Nel caso in esame si può scrivere quanto segue (vedi [20] ):

$$V = J(p, q) = J(p, q)_{ott} + J(p, q)_{obs}$$

$$\dot{V} = \dot{J}(p,q)_{ott} + \dot{J}(p,q)_{obs} = \frac{\partial J(p,q)_{ott}}{\partial p}\dot{p} + \frac{\partial J(p,q)_{obs}}{\partial p}\dot{p} = \frac{\partial J(p,q)_{ott}}{\partial p}u_{TOT} + \frac{\partial J(p,q)_{obs}}{\partial p}u_{TOT} \quad (4.14)$$

$$\dot{V} = u_{TOT}\left(\frac{\partial J(p,q)_{ott}}{\partial p} + \frac{\partial J(p,q)_{obs}}{\partial p}\right) \quad (4.15)$$

Per ottenere $\dot{V} \leq Q$ basta operare la seguente scelta su $u_{TOT}$:

$$u_{TOT} = -\left(\frac{\partial J(p,q)_{ott}}{\partial p} + \frac{\partial J(p,q)_{obs}}{\partial p}\right) \quad (4.16)$$

Che dà come risultato finale $\dot{V} = -\left(\frac{\partial J(p,q)_{ott}}{\partial p} + \frac{\partial J(p,q)_{obs}}{\partial p}\right)^2 \leq 0$. Mentre il valore di $J(p,q)_{ott}$ è già stato definito, il valore di $J(p,q)_{obs}$ può essere calcolato per integrazione sapendo che $u_{obs} = \frac{\partial J(p,q)_{obs}}{\partial p}$.

## 4.4 Esempi

In questa sezione sono elencati una prima serie di esempi che vogliono illustrare come, nelle applicazioni pratiche di maggiore interesse, l'utilizzo delle nuove DFs e del meccanismo di Obstacle Avoidance vadano a modificare, ed in certi casi a migliorare, la risoluzione del problema.

### 4.4.1 Uniform Deployment

Il problema della distribuzione uniforme viene spesso utilizzato nelle reti di sensori perché è strettamente legato all'ottenere una certa densità di informazioni all'interno di una opportuna area. Gli agenti, partendo da una configurazione iniziale molto compatta, si allontanano l'uno dall'altro realizzando una espansione dello sciame (fase di *spread out*) che massimizza la densità di copertura dell'area. L'obiettivo dell'auto-distribuzione viene espresso secondo la seguente formula rappresentata dalla dTDF:

$$d_*(q) = \begin{cases} \bar{d}, & \text{se } q \in p \\ 0, & \text{altrimenti.} \end{cases} \tag{4.17}$$

Dove $P$ rappresenta l'area di un poligono all'interno di $Q$ in cui deve avvenire la copertura. Un approccio simile a questo per risolvere il problema di Deployment può essere ritrovato in [17]. L'esempio di figura 4.6 mette a confronto una operazione di questo tipo con e senza il meccanismo di collision avoidance (CA).

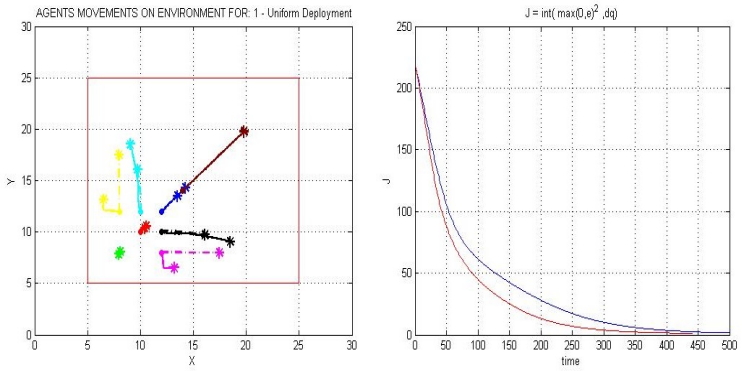


*Fig. 4.6:* Uniform Deployment Task realized by a swarm of 8 gaussian DFs: on the left the CA mechanism is printed by a solid line, on the right with the red line

Nella figura di destra è possibile vedere come il funzionale di costo raggiunge per $t \longrightarrow \infty$ il medesimo valore finale nullo. Con CA il problema viene risolto più velocemente e questo si spiega perché, se gli agenti partono da una configurazione iniziale in cui sono molto vicini, nella fase di *spread out* alla forza di attrazione della TDF si somma anche il contributo di CA, che aiuta gli agenti ad allontanarsi gli uni dagli altri.

Dato che la soluzione di Uniform Deployment è legata alla distribuzione degli agenti nello spazio, non ha molto senso utilizzare le DF con *fov* (le proprietà sensoriali degli agenti sono di poco conto ai fini del problema). In ogni caso una possibile realizzazione di *Uniform Deployment* con $DF_{SS}$ è illustrata in figura 4.7, dove all'interno di un quadrato si distribuiscono le ADFs sigmoidee dei quattro agenti che partecipano al task.

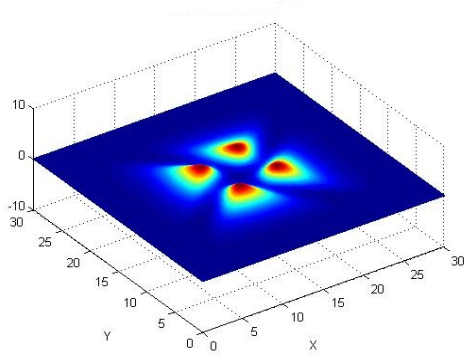


*Fig. 4.7:* Uniform Deployment Task realized by a swarm of 4 sigmoids DFs

### *4.4.2 Static Coverage*

La copertura statica di un'area è un task molto simile a quello di distribuzione uniforme, se non per il fatto che si utilizza il primo quando l'ambiente in cui gli agenti si muovono possiede aree di maggiore interesse rispetto ad altre.

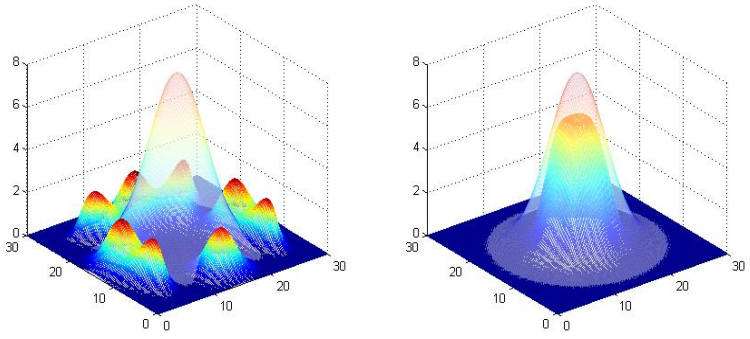


*Fig. 4.8:* Static Coverage Task realized by a swarm of 8 DFs: Initial State on the left and Final State on the right.

Il problema, già trattato in [8], con l'utilizzo delle funzioni descrittive viene migliorato dal punto di vista computazionale, risultando meno dispendioso (vedi [2]). La *dTDF*, trattando aree con importanza differente, dovrà

essere scritta in modo da evidenziare questo aspetto e sarà espressa nella seguente maniera:

$$d_*(q) = \bar{d}_*(q) \tag{4.18}$$

In cui il valore costante $\bar{d}_*(q)$ può anche essere ottenuto da una qualsiasi funzione continua $f(q)$ purché invariante nel tempo. Occorre ricordare che tale formulazione riprende il concetto principale secondo cui gli agenti devono portare una quantità maggiore di informazioni laddove é maggiore il valore della dTDF. Nell'esempio di figura 4.8 è illustrato come, in presenza di uno swarm di DF miste, il problema viene risolto così come accadeva nella versione precedente del framework. Nel caso di figura 4.8 sono state utilizzate le seguenti funzioni descrittive:

$GaussianTDF \longrightarrow d_*(q) = DF_G(A, p_{q1}, p_{q2}, \sigma_{q1}, \sigma_{q2}, \theta) = [8, 15, 15, 4, 4, 0°]$
$DF_G(A, p_{q1}, p_{q2}, \sigma_{q1}, \sigma_{q2}, \theta) = [2, 15, 05, 2, 2, 0°]$
$DF_G(A, p_{q1}, p_{q2}, \sigma_{q1}, \sigma_{q2}, \theta) = [2, 25, 15, 2, 2, 0°]$
$DF_G(A, p_{q1}, p_{q2}, \sigma_{q1}, \sigma_{q2}, \theta) = [2, 15, 25, 2, 2, 0°]$
$DF_G(A, p_{q1}, p_{q2}, \sigma_{q1}, \sigma_{q2}, \theta) = [2, 05, 15, 2, 2, 0°]$
$DF_{SG}(A, p_{q1}, p_{q2}, \sigma_{q1}, \sigma_{q2}, \theta, \phi, K) = [5, 05, 05, 4, 4, 0°, 60°, 2]$
$DF_{SG}(A, p_{q1}, p_{q2}, \sigma_{q1}, \sigma_{q2}, \theta, \phi, K) = [5, 25, 05, 4, 4, 0°, 60°, 2]$
$DF_{SG}(A, p_{q1}, p_{q2}, \sigma_{q1}, \sigma_{q2}, \theta, \phi, K) = [5, 25, 25, 4, 4, 0°, 60°, 2]$
$DF_{SG}(A, p_{q1}, p_{q2}, \sigma_{q1}, \sigma_{q2}, \theta, \phi, K) = [5, 05, 25, 4, 4, 0°, 60°, 2]$

Anche l'effetto di Obstacle Avoidance si fa sentire nel problema di Static Coverage. In figura 4.9 si può osservare come, presa la medesima TDF del caso precedente, senza CA i due agenti dello swarm andrebbero a collidere nella zona di maggior attrazione.

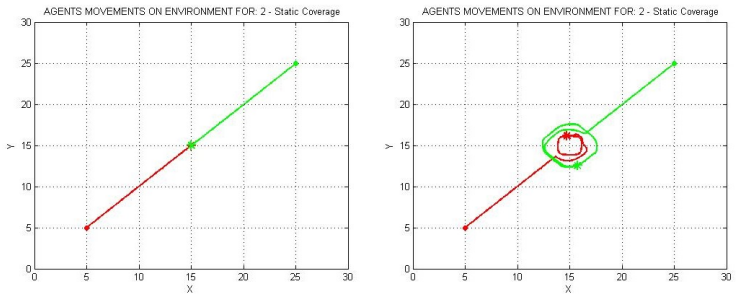


*Fig. 4.9:* Static Coverage with(right) and without(left) Collision Avoidance

### 4.4.3 Effective Coverage

Questo problema è stato presentato per la prima volta da Hussein e Stipanovic [9], e può essere interpretato come la ricerca esaustiva dentro una data area. La dTDF deve dunque definire un livello per cui è possibile ritenere soddisfacente la ricerca, e per far ciò deve tener conto della storia passata, ovvero della copertura che gli agenti hanno già effettuato fino a quell'istante. Dato che la singola ADF rappresenta la quantità istantanea di informazioni che l'agente porta nell'ambiente operativo, è possibile pensare alla ricerca complessiva come all'integrale temporale della cTDF:

$$d_e(q,t) = \int_0^t d(p(\tau),q)d\tau = \int_0^t \sum_{i=1}^{N} d_i(p(\tau),q)d\tau \tag{4.19}$$

Rappresentando con $\bar{C}$ il livello di ricerca desiderato all'interno di un'area si può pensare di rappresentare la dinamica della dTDF con la seguente funzione in cui il valore cala in funzione del moto dello sciame:

$$d_*(q,t) = \max(0, \bar{C} - d_e(q,t)) = \max(0, \bar{C} - \int_0^t \sum_{i=1}^{N} d_i(p(\tau),q)d\tau) \tag{4.20}$$

Il task risulta quindi eseguito quando il valore di $d_*(q,t) = 0$. Con l'utilizzo delle ADF con $fov$ è ora possibile simulare e risolvere tutto un insieme di problemi pratici che prima non potevano essere rappresentati. Si pensi ad esempio al caso in cui si voglia ottenere un certo numero di informazioni (ispezione) su un oggetto/target (relitto subacqueo, palazzo, etc...) e si abbia a disposizione un certo numero di agenti muniti di sensori con un determinato $fov$ (fotocamere,sonar,etc...). Si consideri inoltre il fatto che per ottenere una informazione di una certa qualità devono essere scattate immagini da una distanza ravvicinata in tutti i punti del target. Il problema può ora essere formulato nel seguente modo:

- Il target rappresenta sia un ostacolo, a cui non é possibile avvicinarsi, sia il poligono che delimita l'area in cui è definita la TDF.
- La TDF è definita solo vicino al perimetro del target. I punti al suo interno non son di interesse (informazioni solo perimetrali) e non possono essere neppure raggiunti.

- Solo quando la differenza fra la DF di un agente e la TDF è minore di un certo valore (nell'esempio pari a 1) si utilizza l'equazione dinamica 4.20. Questo serve per far diminuire le richieste di risorse da una zona solo se l'agente si è avvicinato con una parte della sua DF che possiede una elevata capacità (richiesta di informazioni di elevata qualità).

In figura 4.10 sono indicati i risultati ottenuti dall'esecuzione di un task di questo tipo. Gli agenti si muovono circondando il target sempre rivolti verso il perimetro (è la parte su cui è definita la TEF). Alla fine della simulazione ogni elemento dell'oggetto da ispezionare è stato analizzato con la cura necessaria definita dal problema, così come è possibile vedere dall'andamento del funzionale di costo.

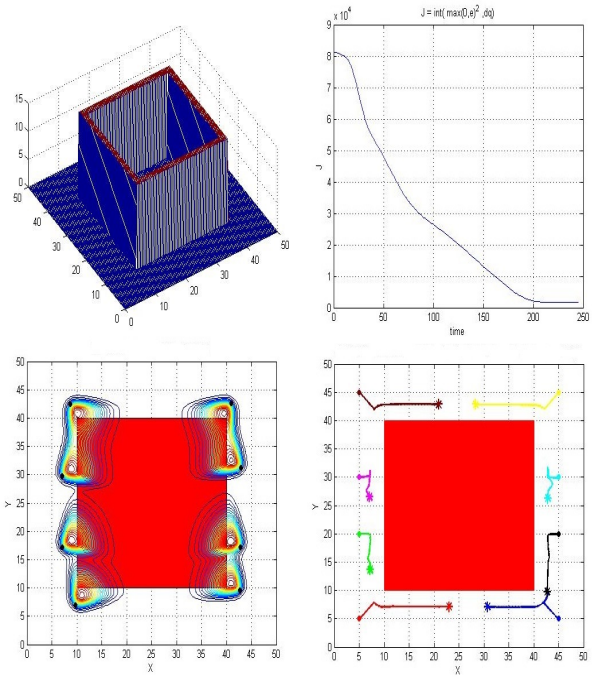


*Fig. 4.10:* Inspection Task, from top left to bottom right: TDF, Cost Functional, cTDF during execution and agents path.

Nell'esempio di figura sono state utilizzate tutte funzioni descrittive identiche di tipo $DF_{SG}$ con la seguente parametrizzazione:

$$DF_{SG_i}(A, p_{q1}, p_{q2}, \sigma_{q1}, \sigma_{q2}, \theta, fov, K) = [3, p_{q1_i}, p_{q2_i}, 4, 4, \theta_i, 90°, 2]$$

### 4.4.4 Dynamic Coverage

L'obiettivo della copertura dinamica consiste nel muovere frequentemente gli agenti all'interno dello spazio operativo in modo da mantenere elevato in tutti i punti il valore di copertura. Per ottenere questo risultato occorre quindi tenere memoria del passaggio di un agente almeno per un certo periodo di tempo: se di fatto un agente passa sopra un'area e successivamente si allontana, la dTDF in quella zona dovrà richiedere nuovamente il passaggio dell'agente solo dopo un certo periodo di tempo. Un metodo per realizzare questa TDF passa attraverso il concetto di "*Information Decay Process*", descritto dalla seguente equazione dinamica:

$$\dot{I}(q,t) = \delta I(q,t) + D(p,q) \tag{4.21}$$

Con $\delta \leq 0$ ad indicare il decadimento del processo rappresentato da $I(q,t)$. In questo contesto $D(p,q)$ rappresenta il guadagno di informazione definito dalla cTDF. La TDF può essere complessivamente rappresentata nel seguente modo:

$$d_*(q,t) = \bar{C} - I(q,t) \tag{4.22}$$

$\bar{C}$ indica il livello minimo di copertura effettiva che si vuole raggiungere per ciascun punto. In termini del tutto generali la dinamica può essere descritta come segue:

$$d_*(q,t) = \bar{C} + H(q,t) - D(p,q,t) \tag{4.23}$$

Il termine $H(q,t)$ rappresenta un processo dinamico che tende ad aumentare nel tempo, mentre $D(p,q,t)$ è la cTDF che cerca di contrastare questo incremento. In quest'ultima concezione del problema, come del resto anche nella precedente, il task si considera risolto quando il valore di $d_*(q,t) \leq \bar{C}$. In figura 4.11 è illustrata una possibile applicazione di questo tipo per un task di sorveglianza. In questo esempio viene messo a confronto il comportamento di uno swarm realizzato con le nuove $DF_{SG}$ e uno con le $DF_G$. Pur non avendo la stessa forma, per realizzare il confronto sono state scelte DF il cui integrale nello spazio fosse quantomeno approssimabile (stessa quantità di risorse ma distribuita diversamente). Quanto appena scritto è confermato dal valore iniziale del funzionale di costo, rappresentato in figura 4.12.

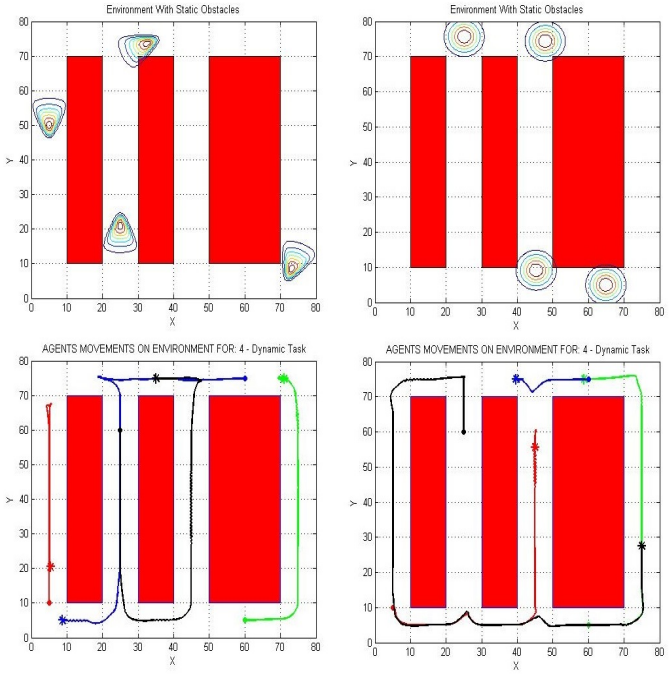


*Fig. 4.11:* Dynamic Coverage: $DF_{SG}$ VS $DF_G$

Come si può vedere le DF con *fov* reagiscono meglio ad un task di questo tipo (vedi andamento del funzionale di costo in figura 4.12) perché sfruttano il fatto di possedere una direzione di preferenza. Gli agenti con DF omnidirezionale, per spostarsi, devono infatti aspettare che ci sia un'asimmetria nella forze di attrazione che ogni punto dell'environment esercita su di loro (cosa che invece è già presente per costruzione nelle DF con campo angolare).

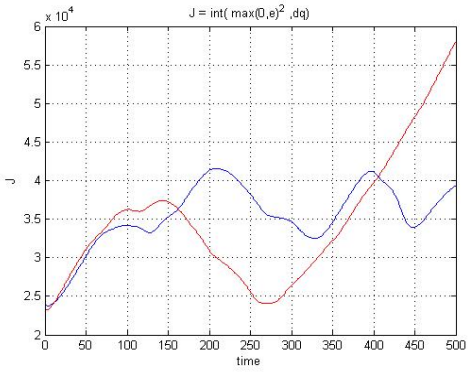


*Fig. 4.12:* Dynamic Coverage: $DF_{SG}$(blue) VS $DF_G$(red) Cost Functional

Queste ultime affermazioni sulla prontezza di una DF con *fov*, rispetto a quelle proposte in [1] e [2], hanno valenza del tutto generale e può essere provato anche in altre tipologie di task come ad esempio Effective Coverage. Nella sezione 4.4.6 verrà fatto qualche esempio specifico.

### 4.4.5 Target Assignment

Considerando uno scenario in cui sono presenti $N_w$ target statici e $N$ agenti, si vuole risolvere il problema di andare a spostare ogni veicolo nella posizione contrassegnata da un determinato target. Ovviamente quando $N \geq N_w$ il problema è completamente risolubile, altrimenti, nel caso in cui $N < N_w$ occorrerà ridimensionare il problema considerando solo $N$ target. Utilizzando il framework delle funzioni descrittive è stato scelto di utilizzare anche per il task la stessa espressione matematica utilizzata per modellare gli agenti:

$$d_*(q,t) = \sum_{j=1}^{N_w} d_j(p_t, q) \tag{4.24}$$

In questo task l'utilizzo delle DF con *fov* non porta modifiche sostanziali al problema, se non per il fatto che può solo peggiorare le cose, dato che gli agenti non sentono l'attrazione dei target posizionati alle loro spalle. In ogni caso è qui di seguito illustrato un esempio di *target assignment* effettuato con le DF gaussiane ma con l'aggiunta di ostacoli nell'environment. In questo caso specifico il task è portato completamente a termine.

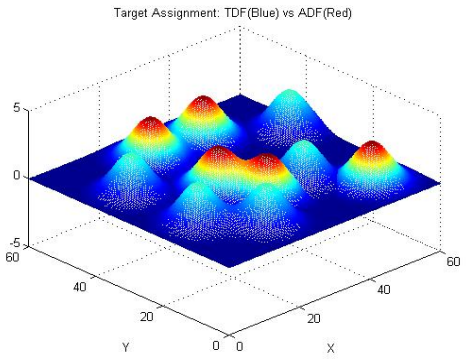


*Fig. 4.13:* Target Assignment: TDF(blue) VS cTDF(red)

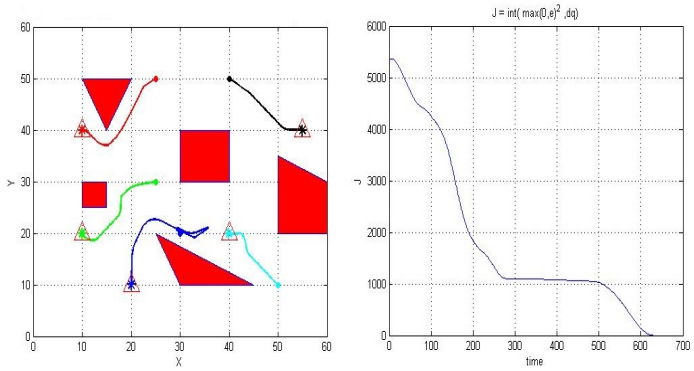


*Fig. 4.14:* Target Assignment using the DF Framework

### 4.4.6 Confronto fra DFs

Così come già accennato nell'esempio 4.4.4, in questa sezione vengono messi a confronto i comportamenti di tre DF diverse nella risoluzione del medesimo task. Per il caso in esame è stato scelto di utilizzare un task di Effective Coverage per cui in tutto l'environment è presente una costante richiesta di risorse che deve essere soddisfatta ($d_*(q, t = 0) = 5$ ). Sono state quindi scelte le seguenti DF:

$DF_G(A, p_{q1}, p_{q2}, \sigma_{q1}, \sigma_{q2}, \theta) = [3, 10, 10, 3, 3, 0°]$
$DF_{SG}(A, p_{q1}, p_{q2}, \sigma_{q1}, \sigma_{q2}, \theta, \phi, K) = [5, 10, 10, 5, 5, 0°, 60°, 2]$
$DF_{SS}(A, p_{q1}, p_{q2}, K, K2, \theta, \phi) = [5, 10, 10, 2, -0.2, 0°, 60°]$

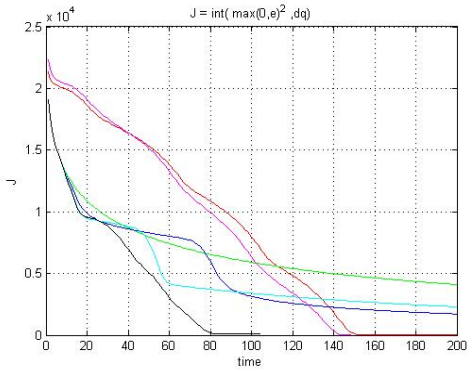


*Fig. 4.15:* Cost Functional: $DF_{SG}$(red), $DF_{SS}$(pink), $DF_G$(green), $DF_G$ with perturbation (others)

La quantità di risorse messe a disposizione da ciascuna delle DF scelte è simile, anche se la funzione di tipo $DF_G$ è leggermente più grande. In entrambi i casi in cui è presente un FoV il task viene portato a termine con successo, come è possibile vedere dal grafico del funzionale di costo (linea rossa e rosa). Per quanto riguarda il caso $DF_G$, se l'agente è posizionato al centro dell'environment, quindi in una posizione di simmetria, questo non si muove (asterisco verde al centro in figura 4.16). Solo perturbando l'ingresso $u$ è possibile farlo uscire da questa situazione di stallo dovuta ad un minimo locale. In ogni caso, però, se la perturbazione è realizzata in modo "simmetrico" (ovvero se $\tilde{u}_{q1} = \tilde{u}_{q2}$ o quando $\tilde{u}_{qi} \neq 0 \wedge \tilde{u}_{qj} = 0$), allora l'agente si muove in linea retta, dato che sentirà due forza attrattive identiche che si annullano nella direzione ortogonale al moto (vedi gli andamenti di colore blu e celeste nei grafici di figura 4.16 e 4.15 ). Anche in questo caso il task non viene completato con successo.

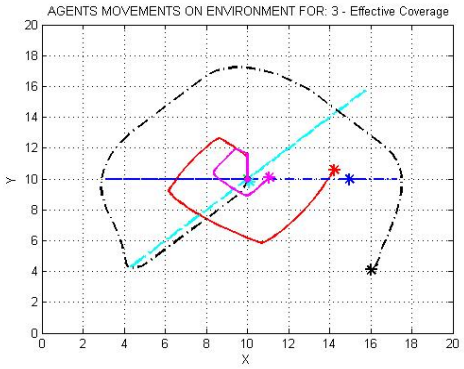


*Fig. 4.16:* Agents Path: $DF_{SG}$(red), $DF_{SS}$(pink), $DF_G$(green), perturbated $DF_G$ (others)

Per far si che anche l'agente con $DF_G$ possa portare a compimento il task, è opportuno andare a perturbare entrambi gli ingressi in modo differente. In questo caso l'agente riesce ad uscire dalla situazione di simmetria e, sfruttando la maggior capacità, porta il task a zero in un tempo minore rispetto a $DF_{SG}$ e $DF_{SS}$ (vedi andamento di colore nero). Se per un semplice task, come quello proposto ora, la simmetria si presenta solo all'avvio, in un task più complesso queste situazioni si possono presentare più volte. Oltre a questo difetto, la capacità di sentire richieste che provengono da ogni direzione, che in certi casi può essere un vantaggio, talvolta comporta un rallentamento dell'agente, così come già dimostrato nella sezione 4.4.4.

# 5. CONTROLLO DECENTRALIZZATO

## *5.1 Introduzione*

Per utilizzare le leggi di controllo proposte nel precedente capitolo in uno scenario reale, è necessario implementare un meccanismo di comunicazione fra gli agenti che renda possibile stimare lo stato dello swarm. Un agente infatti, per potersi muovere all'interno dell'ambiente operativo in maniera corretta, deve essere in grado di conoscere in ogni istante di tempo sia il valore della TDF (nel caso statico possono anche essere note *a priori*) sia le DF degli altri agenti. Ovviamente tali informazioni non sono sempre disponibili in maniera diretta e, a seconda della topologia con cui è possibile rappresentare il network, l'agente deve poterle ricavare comunicando con i soli "*neighbors*"(problema di ottimizzazione globale con architettura del controllo distribuita). Per quanto appena asserito servirà quindi stimare la cTDF e la TDF per ricavare la $u_{ott}$, e le posizioni di tutti gli ostacoli statici (e non) per calcolare la $u_{obs}$. In maniera schematica è possibile elencare i parametri necessari per ottenere questo risultato:

- $p_{V_1}, p_{V_1}, \ldots, p_{C_1}, p_{C_2}, \ldots$. Posizione dei vertici e del baricentro che servono a realizzare il meccanismo anti-collisione presentato nel capitolo 4.3. Queste variabili possono anche essere note a priori da ciascun agente e si possono limitare alle posizioni dei baricentri qualora si volesse utilizzare la soluzione definita "fortemente conservativa".

- $\xi = [p_x, p_y, A, \theta, \sigma_x, \sigma_y, DFtype, task, K, K_2, fov]^T$. Sono il numero massimo di parametri necessari a definire la DF di ogni agente. Noti questi è possibile ricavare il valore della cTDF. Mentre la variabile $df\_type$ indica se la DF è di tipo gaussiana o sigmoidea, $task$ indica qual è il task che sta eseguendo in quel momento l'agente. Tutto questo set di variabili deve essere calcolato e stimato in tempo reale, dato che la posizione di ogni agente è necessaria pure a realizzare il meccanismo anti-collisione presentato nel precedente capitolo.

- $N$. Il numero degli agenti che partecipano al task o più in generale alla missione. E' una informazione necessaria per sapere il numero di

parametri che deve essere stimato, e generalmente $N$ viene assegnato all'inizio della missione; tuttavia per quelle situazioni in cui si vuole modificare la dimensione dello swarm durante l'esecuzione di una missione, può essere possibile realizzare una stima real-time così come proposto in [15].

Nella prossima sezione verrà proposto un algoritmo di "consensus tracking" che sarà utilizzato per la stima dell'*information state*, definito come:

$$x(t) = \begin{bmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_N \end{bmatrix}$$

Dove il generico elemento $\xi_i$ è definito dai parametri che definiscono la DF di ogni agente:

$$\xi_i = [p_{x_i}(t), p_{y_i}(t), A_i, \theta_i, \sigma_{x_i}, \sigma_{y_i}, DFtype_i, task_i, K_i, K2_i, fov_i]^T$$

## 5.2 Stima dei Parametri

Una soluzione al problema di stima può essere fornita attraverso l'utilizzo dell'equazione 2.19, che definisce l'algoritmo di consenso/tracking dinamico utile nel caso in cui l'*information state* sia disponibile ad un solo sottogruppo di agenti.

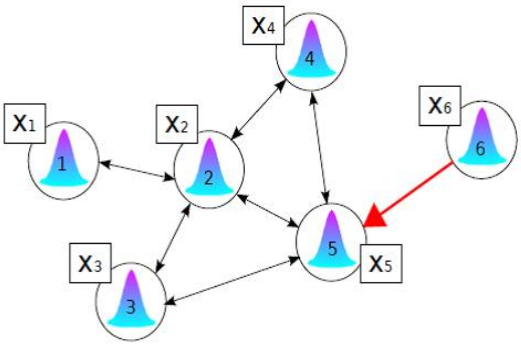


*Fig. 5.1:* Consensus Tracking Topology

L'algoritmo di consenso utilizzato nella presente tesi è stato per la prima volta proposto da Wei Ren ( vedi [7], [18] e [27] )con lo scopo di definire un metodo che funzionasse sia per il caso di "cambiamento dinamico del leader", sia in presenza di una qualunque "topologia dinamica orientata" ( quando non orientata viene studiata come un caso particolare). Anche se il problema viene risolto e dimostrato considerando un solo *leader* (che dà il riferimento) e tutti gli altri *followers* (che eseguono il tracking), tale situazione è facilmente riportabile al caso in esame.

La stima dell'intero vettore $x(t)$ può infatti essere realizzata andando a suddividere il tracking per ciascun elemento $\xi_i$ risolvendo così $i$ problemi diversi di *Consensus Tracking*. Ogni agente sarà così leader del "sub-network" relativo alla comunicazione dei propri parametri, ma sarà a sua volta follower in tutti gli altri (la situazione illustrata in figura 5.1 si ripete quindi per ogni agente o se si vuole per ogni $\xi_i$ ). Se ad esempio è presente uno sciame di tre agenti, il primo di questi dovrà comunicare per intero $[\xi_1, \xi_2, \xi_3]$, ma dovrà occuparsi di stimare i soli valori $\xi_2$ e $\xi_3$, dato che $\xi_1$ è il suo vettore di stato.

Studiando quindi il problema per il singolo $\xi_i$ le conclusioni tratte dovranno poi essere riportate a tutti gli altri sub-network per poter considerare il problema completamente risolto. La prima conseguenza di questa suddivisione vede l'applicazione del teorema 4 che porta alla conclusione per cui il consenso su $\xi$ è raggiunto con l'algoritmo 2.19 se e solo se all'interno del grafo di comunicazione è presente un albero di copertura orientato con radice. Considerando quindi il singolo vettore $\xi_i$ si può scrivere:

$$\dot{\xi}_i = \frac{1}{\eta_i} \sum_{j=1}^{N} a_{ij}[\dot{\xi}_j + \gamma(\xi_j - \xi_i)] + \frac{1}{\eta_i} a_{(i,n+1)}[\dot{\xi}^r + \gamma(\xi^r - \xi_i)] \qquad (5.1)$$

Dove la formula può essere semplificata considerando il leader $r$ facente parte del *neighbourhood* $N_j$:

$$\dot{\xi}_i = \frac{1}{\eta_i} \sum_{j=1}^{N} a_{ij}[\dot{\xi}_j + \gamma(\xi_j - \xi_i)]$$

Presi i valori della matrice di adiacenza come in appendice A, la formula precedente può quindi essere riscritta nella forma semplificata utilizzata in questa tesi:

$$\dot{\xi}_i = \frac{1}{N_i} \sum_{N_i} [\dot{\xi}_j + \gamma(\xi_j - \xi_i)] \tag{5.2}$$

Dove $N_i$ è la dimensione del *neighbourhood* mentre $\dot{\xi}_j$ è la derivata temporale della variabile che vuole essere stimata. La comunicazione di quest'ultimo parametro può essere anche evitata, dato che è possibile stimarne il valore a partire da quello di $\xi_j$ con una equazione alle differenze di tipo "Eulero":

$$\dot{\xi}_i \approx \frac{\xi_i(K+1) - \xi_i(K)}{\Delta T}$$

Il teorema 4 che definisce le condizioni di convergenza dell'algoritmo generale 5.1, e quindi anche della formula 5.2 utilizzata in questa tesi, viene qui di seguito riportato per intero, in modo da considerare anche le condizioni legate alla validità in presenza di "cambiamento dinamico del leader" e di "topologia dinamica orientata" (vedi anche [18]).

**Teorema 5** (Consensus Tracking)**.** *Dato il sistema dinamico $\dot{\xi}_i = u_i$ con ingresso $u_i$ definito dall'equazione 5.1, sia $\delta_i = \sum_{j=1}^{N} a_{ij}(\hat{\xi}_j - \xi_j)$ e $t_0$ l'istante di inizio osservazione. Si ipotizzi la matrice di adiacenza $A_{n+1}(t)$ continua a tratti e scelta in un set compatto di costanti positive $[\bar{a}, \underline{a}]$. Si ipotizzi inoltre che nel grafo rappresentativo di $A_{n+1}(t)$ sia contenuto un albero di copertura durante tutti gli istanti di cambiamento della topologia definiti come $t_j, tj+1, \ldots$. Il sistema in ciclo chiuso definito dallo stato $\xi_i - \xi^r$ e dall'ingresso $\delta_i$ è stabile ISS (Input to State Stability). Quando $\delta_i(t) \to 0$ allora $\xi_i \to \xi^r$*

*Dimostrazione.* Si consideri per prima cosa che $\frac{1}{\eta_{ij}}$ è un valore sempre definito dato che è stato supposto per $A_{n+1}(t)$ un albero di copertura per ogni istante $t$. L'equazione 5.1 può essere riscritta come segue:

$$[W(t) \otimes I_m]u + [b(t) \otimes I_m]\dot{\xi^r} = -\gamma([W(t) \otimes I_m]\xi + [b(t) \otimes I_m]\xi^r) + \delta$$

Dove:

- $u = [u_1^T, \ldots, u_n^T, \dot{\xi^r}^T]^T$.

- $L_{n\times(n+1)}$ è il Laplaciano considerato un grafo "rettangolare" dovuto all'aggiunta del leader vhe è l'elemento $n+1-$esimo (vedi capitolo 2).
- $L_{(n+1)\times(n+1)} = \begin{bmatrix} L_{n\times(n+1)} \\ 0_{1\times(n+1)} \end{bmatrix}$ il Laplaciano complessivo di rango $n$ per la presenza dell'albero di copertura (vedi capitolo 2 e appendice A).
- $L_{n\times(n+1)} = [W(t)|b(t)]$, dove $W$ rappresenta la matrice di adiacenza del grafo con i soli followers di dimensione $n \times n$, e $b$ è il vettore di dimensione $n \times 1$ che indica con quali agenti è in diretta comunicazione il leader.

Se, come ipotizzato, è presente un albero di copertura durante ogni istante $t$, allora la matrice $W(t)$ è sempre invertibile quindi ogni elemento di $W(t)^{-1}$ risulta limitato in modulo (ipotesi derivata da $[\bar{a}, \underline{a}]$). Considerando che per questo motivo le topologie possibili di un grafo sono limitate, ciò implica che anche le possibili strutture della matrice $W(t)$ lo siano. Questo implica che anche $W(t)^-1$ è definito su un numero di strutture limitate e ciò è equivalente a dire che $det(W(t)) \neq 0$ durante ciascun intervallo di transizione $[t_j, t_{j+1})$. Quanto detto porta a scrivere che:

$$b(t) = -W(t)1_n \longrightarrow -W(t)^{-1}b(t) = 1_n$$

La dinamica del sistema, da cui si ricavano direttamente le conclusioni del teorema, può essere quindi riscritta come segue:

$$\dot{\xi} - W(t)1_n\dot{\xi^r} = -\gamma(\xi - 1_n \otimes \xi^r) + [W(t)^{-1} \otimes I_m]\delta$$

□

Da quanto fin'ora asserito si capisce anche l'importanza della scelta del parametro $\gamma$ al fine di ottenere una velocità di convergenza desiderata nell'inseguimento al riferimento. In figura 5.2 sono illustrati tre test di tracking ottenuti con $\gamma = 1$. Nel primo viene inseguito una rampa con il risultato finale di errore a regime nullo; nel secondo viene inseguito un segnale di ingresso composto dalla somma di due sinusoidi; mentre nel terzo il tracking viene realizzato su segnale reale di posizione preso durante un test.

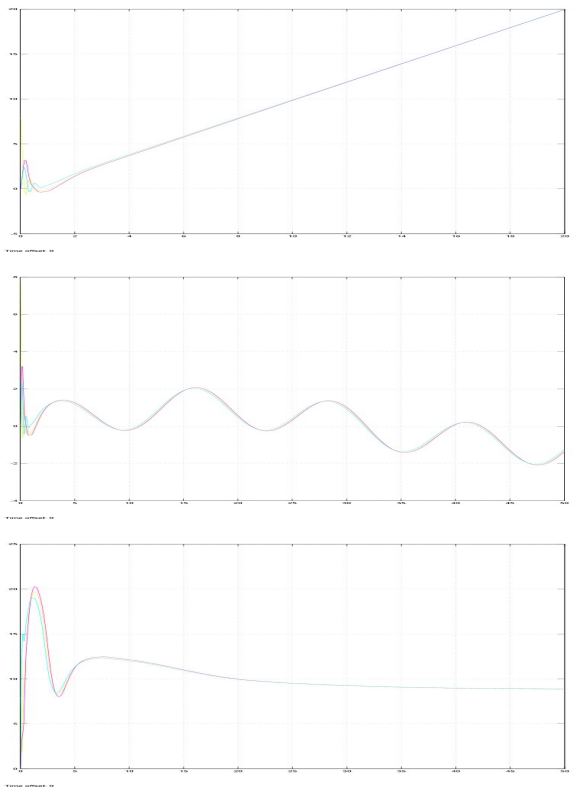

*Fig. 5.2:* Tracking examples

Un confronto fra due soluzioni ottenute prendendo due coefficienti $\gamma$ diversi è raffigurata in figura 5.3 dove, partendo da uno stato iniziale arbitrario diverso da zero, tre agenti inseguono il riferimento prima con $\gamma = 1$ e successivamente con $\gamma = 10$.

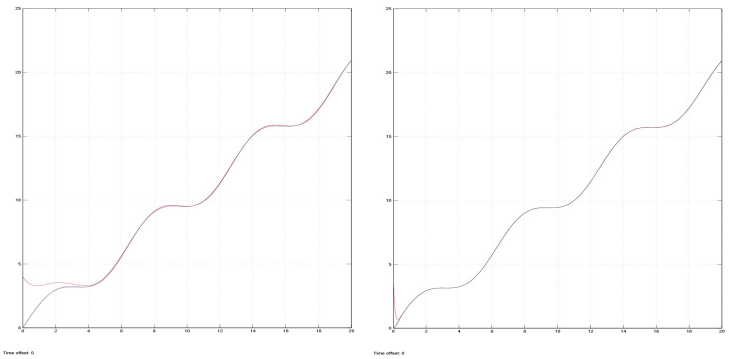

*Fig. 5.3:* Tracking examples with $\gamma = 1$ and $\gamma = 10$

Una realizzazione pratica del *Consensus Tracking* qui proposto, può essere fatta definendo inizialmente un numero di filtri pari a $N-1$ (dove $N$ è il numero degli agenti dello sciame). Poiché a seconda della topologia della rete il vettore $\xi$ sarà ricevuto solo dai *neighbours*, l'agente dovrà poi considerare valide le sole dinamiche attinenti a quel gruppo di agenti e dovrà così invalidare tutte le altre. Occorre porre molta attenzione a questo procedimento poiché l'invalidazione non si realizza ponendo nulli gli ingressi ( per cui verrebbe calcolata ugualmente la dinamica considerando dei segnali di riferimento nulli), bensì annullando le uscite delle singole dinamiche. Nel capitolo 8 verrà illustrato con maggior precisione e con l'utilizzo di Simulink quanto appena detto.

## 5.3 Esempi

Nei test che seguono vengono riprese delle tipologie di task proposte anche nel capitolo 4.4, ora però interpretate considerando il problema decentralizzato. Tutte le simulazioni seguenti sono effettuate utilizzando:

1. Una realizzazione Simulink del sistema funzionante a tempo discreto.
2. La simulazione di un Network reale con possibili modifiche delle comunicazioni (purché sia sempre garantita la presenza di un albero ricoprente con radice) e con un ritardo di $0.1sec$.
3. Il filtro di consenso proposto in questo capitolo con $\gamma = 5$.
4. Il meccanismo di Obstacle Avoidance proposto nel capitolo 4.3 ottenuto con $r = 5$ e $r_{min} = 1$.
5. Una parametrizzazione delle DF per cui si utilizza il seguente vettore per indicarle $[p_x, p_y, A, \theta, \sigma_x, \sigma_y, K, K_2, fov, DFtype, task]^T$.

### 5.3.1 Uniform Deployment

E' stato deciso di analizzare come primo caso un task di Uniform Deployment, realizzato con uno swarm formato da quattro agenti inizialmente già posizionati all'interno del poligono/quadrato *poly* in cui è anche definito il task. Il quadrato ha vertici $\{(0,0),(0,20),(20,20),(20,0)\}$ e la TDF e le ADFs sono così definite:

$$d_*(q) = \begin{cases} 1, & \text{se } q \in poly \\ 0, & \text{altrimenti.} \end{cases}$$

$ADF_1 = [2, 2, 2, 0, 4, 4, Gauss, UniDep]$
$ADF_2 = [5, 5, 2, 0, 4, 4, Gauss, UniDep]$
$ADF_3 = [2, 5, 2, 0, 4, 4, Gauss, UniDep]$
$ADF_4 = [5, 2, 3, 0, 3, 3, Gauss, UniDep]$

La situazione iniziale è quella espressa nella prima immagine di figura 5.4 in cui si nota che la cTDF è già completamente contenuta all'interno di *poly*.

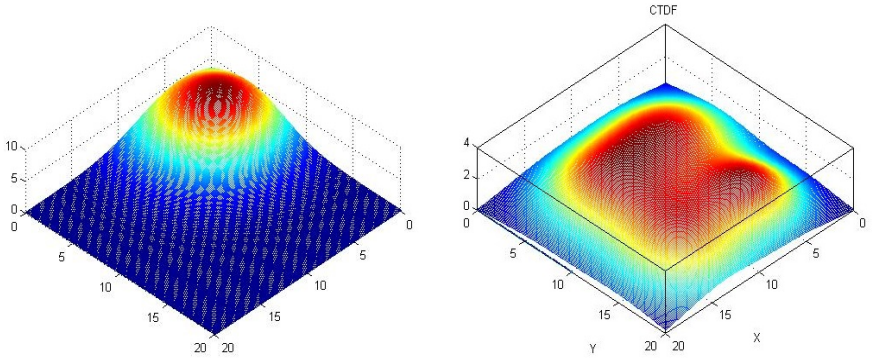


*Fig. 5.4:* Uniform Deployment: Task Complete

Al termine della simulazione (vedi seconda immagine di figura 5.4) gli agenti si sono distribuiti all'interno dello spazio delimitato dal poligono secondo un equilibrio definito dalle forze di attrazione e repulsione. Nell'immagine di figura 5.5 viene mostrata la stima dell'andamento del funzionale di costo fatta da ciascun agente attraverso la legge di consenso 5.1. Mentre le oscillazioni sul valore finale sono dovute alla vicinanza di un minimo locale che viene raggiunto con un passo di discesa troppo elevato, il fatto che dopo una brusca discesa vi sia una leggera risalita è invece dovuto alle forze repulsive che gli agenti si scambiano per evitare le collisioni. Dopo un breve transitorio iniziale, i valori stimati convergono, ad indicare che gli agenti hanno una visione omogenea dello stato di risoluzione del task. La stima del funzionale di costo è però solo la conclusione di un processo di stima che

vede come variabili di stato i parametri indicativi delle DFs. Vale infatti la seguente relazione:

$$\tilde{J}(\tilde{\xi}, q) = \int_Q max(0, d_*(q,t) - \sum_{i=1}^{N} \tilde{d}_i(\tilde{\xi}))^2 dq \tag{5.3}$$

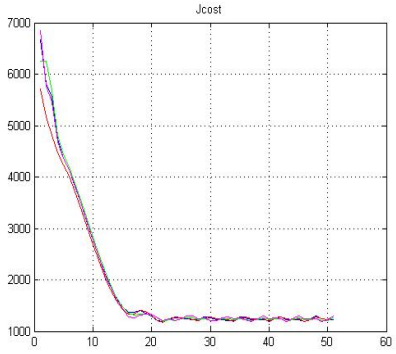


*Fig. 5.5:* Uniform Deployment: Cost Functional

### 5.3.2 Effective Coverage

In un problema di Effective Coverage, dove rispetto al caso di Uniform Deployment gli agenti tendono a muoversi di più, i filtri di stima garantiscono comunque il tracking dei parametri. Quello che qui si vuole dimostrare è infatti l'efficacia del filtro all'inseguimento dei valori di posizione e rotazione che hanno una dinamica molto più veloce che nell'esempio precedente. Per questo motivo agli agenti sono state assegnate delle DFs con $fov$ che comportano una continua variazione anche del parametro $\theta$.

$\xi = [p_x, p_y, A, \theta, \sigma_x, \sigma_y, DFtype, task, K, K_2, fov]$ :

$ADF_1 = [5, 5, 3, 0, 4, 4, DF_G, EffCov, \dots]$
$ADF_2 = [10, 10, 4, 0, 3, 3, DF_{SG}, EffCov, 3, -0.2, \pi/2]$
$ADF_3 = [5, 2, 5, 0, 6, 6, DF_{SG}, EffCov, 3, -0.2, \pi/4]$
$ADF_4 = [2, 5, 5, 0, 3, 3, DF_{SG}, EffCov, 2, -0.2, \pi/1.5]$

$$d_*(q) = \begin{cases} 5, & \text{se } q \in poly \\ 0, & \text{altrimenti.} \end{cases}$$

Dove l'area di definizione del task è contenuta nel poligono *poly* i cui vertici sono: $\{(0,0),(0,50),(50,50),(50,0)\}$. Nelle immagini che seguono sono indicate le stime fatte dagli agenti in relazione ai parametri $p_{q1}$, $p_{q2}$ e $\theta$ del terzo agente.

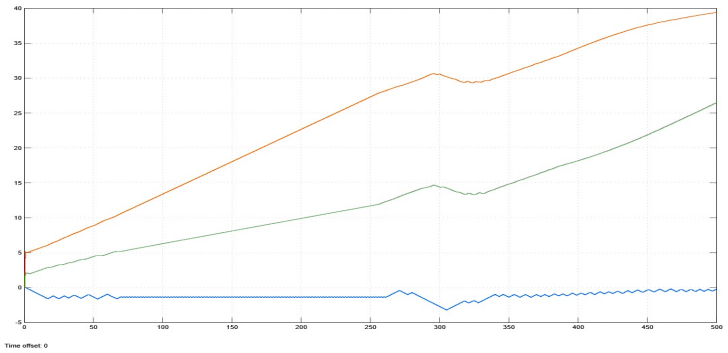

*Fig. 5.6:* Agent parameters estimation: $p_x$ (red), $p_y$(green) and $\theta$ (blue).

Gli ingrandimenti mettono a risalto il processo di stima a tempo discreto. Nell'immagine 5.7 a sinistra viene raffigurato il transitorio iniziale che porta all'inseguimento dei parametri imposti dal *root*; nel riquadro di destra è raffigurata invece una fase di passaggio nella stima del valore di $p_{q1}$.

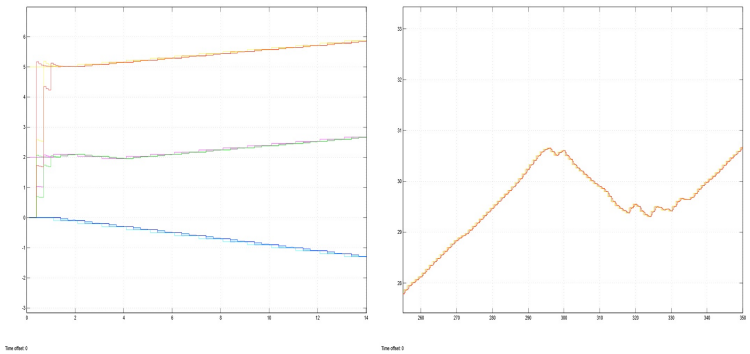

*Fig. 5.7:* Agent parameters estimation: initial phase zoom(left), and a general phase of the $p_{q1}$ estimation tracking process (right).

Anche la stima del funzionale di costo viene quindi eseguita correttamente (dato che dipende direttamente da quella dei singoli parametri) e in figura 5.9 è possibile verificare quanto appena detto. Dopo un breve transitorio iniziale tutti gli agenti convergono sullo stato globale dello swarm e ciò, come già detto per l'esempio precedente, garantisce la corretta esecuzione del task.

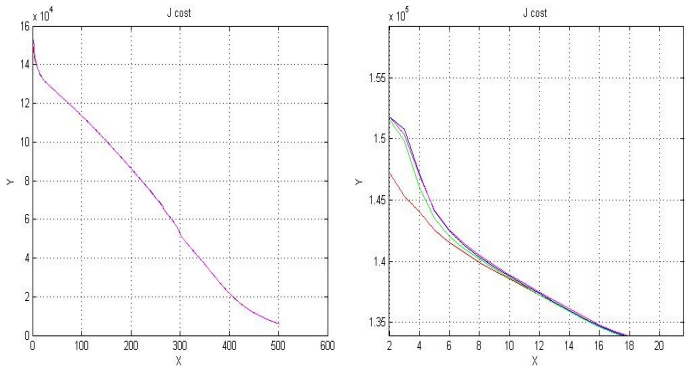


*Fig. 5.8:* Effective Coverage: Cost Functional with a zoom on the initial phase of the estimation process

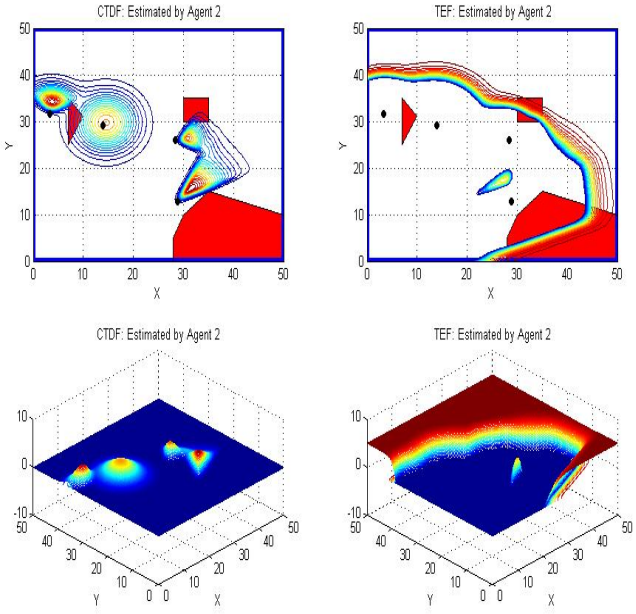


*Fig. 5.9:* Effective Coverage: an estimation of cTDF and TEF realized by agent $n°2$.

# 6. CONNETTIVITÀ

L'utilizzo dell'algoritmo di *Consensus Tracking* proposto nel capitolo precedente comporta che all'interno di ogni sub-network sia presente un albero di copertura con radice. Lo studio di questo problema può essere ricondotto alla teoria della *Connectivity Maintenance*, che si occupa di garantire il mantenimento di alcune proprietà del grafo durante l'esecuzione del task, ovvero durante il moto degli agenti nello spazio. Il problema di connettività può essere studiato direttamente dall'osservazione del Laplaciano, andando a valutare l'autovalore $\lambda_2$. Come scritto anche nell'appendice A, è possibile mettere in relazione la condizione $\lambda_2 > 0$ con la presenza di un albero di copertura nel network che è la proprietà che si vuole mantenere.

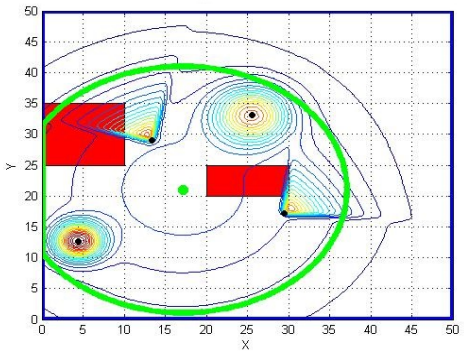


*Fig. 6.1:* Connectivity Range

In questa tesi il problema non viene studiato dal punto di vista analitico, ma viene proposta invece una soluzione pratica (con un approccio di tipo conservativo), realizzabile nuovamente con le funzioni descrittive. Considerando la distanza massima a cui gli agenti possono comunicare è possibile definire una circonferenza di diametro pari a quel valore. All'interno di questa area le comunicazioni fra gli agenti sono ovviamente garantite ed è per ciò presente anche un albero di copertura (di tipo *rooted* se il grafo è orientato).

## 6.1 L'Agente Virtuale

Sfruttando le proprietà di simmetria e omni-direzionalità, è possibile utilizzare una $DF_G$ per definire un'area di interesse all'interno della quale gli agenti devono muoversi per mantenere la connettività. Questa funzione, che prenderà il nome di *Connectivity Descriptor Function*, o $C(p_V, q)$, può essere definita come segue:

$$CDF = C(p_V, q) = Ae^{-0.5(\frac{(q_1 - p_{V_x})^2}{\sigma_x^2} + \frac{(q_2 - p_{V_y})^2}{\sigma_y^2})} \tag{6.1}$$

I parametri $\sigma_x$ e $\sigma_y$ possono essere scelti arbitrariamente in funzione del range di comunicazione (ovviamente $\sigma_x = \sigma_y$), mentre la posizione $p_V$ è legata al moto di un agente virtuale che si muove utilizzando tale DF. Questo agente, che può essere simulato da una *Ground Station* o da uno degli altri veicoli/velivoli (ma i cui parametri devono essere stimati da tutti), si muove in direzione della massimizzazione fra la $CDF$ e la $TDF$, per cui vale:

$$(\dot{p}_{V_{q1}}, \dot{p}_{V_{q2}}) = u_V(t) = K \int_Q d_*(p, q) \frac{\partial C(p_V, q)}{\partial p_V} dq \tag{6.2}$$

Dove $K$ è il guadagno dell'algoritmo trovato come nell'equazione 4.5, dove veniva invece scritto nella forma $2\beta$. La CDF, che è solo virtuale, non comporta nessuna modifica al problema di minimizzazione del funzionale di costo, ma serve solo a muovere lo sciame in maniera compatta. Gli agenti reali, invece, si muovono con la seguente legge di controllo in cui è inserito il problema di connettività:

$$u_i(t) = 2\beta \int_Q \{max(0, C(p_V, q)e_A(p, q) + C(p_V, q))\} \frac{\partial d_i(p, q)}{\partial p_i} dq \tag{6.3}$$

Il doppio utilizzo della $C(p_V, q)$ serve per due motivi ben precisi: il primo è legato a mantenere valida la funzione di errore $e_A(p, q)$ solo all'interno dell'area dove è garantita la connettività (può essere interpretato come una operazione di filtraggio); il secondo è legato a far sì che, quando $e_A(p, q) = 0$, rimanga comunque presente un contributo sempre attivo che tiene gli agenti

vicini e comunicanti (può essere invece interpretato come un task di Static Coverage sempre presente). La CDF è quindi utilizzata come una lente su cui focalizzare l'attenzione degli agenti.

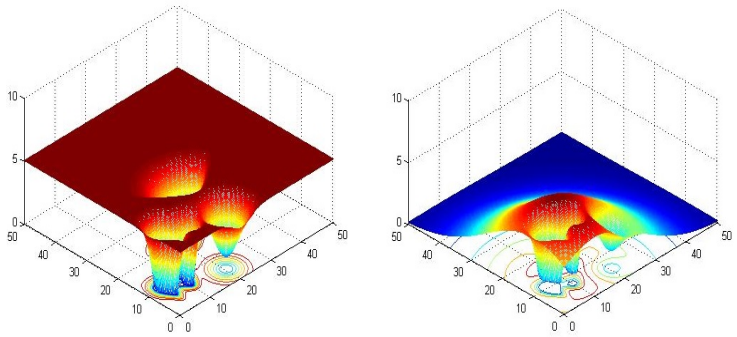

*Fig. 6.2:* $TEF$ vs $max(0, C(p_V, q)e_A(p, q) + C(p_V, q))$

E' inoltre opportuno ricordare che, in presenza di task la cui dinamica è variabile in funzione della posizione degli agenti (ad esempio Effective Coverage e Dynamic Coverage), non viene tenuto conto del valore della CDF, che non prende quindi parte al calcolo della $cTDF$, come è logico aspettarsi.

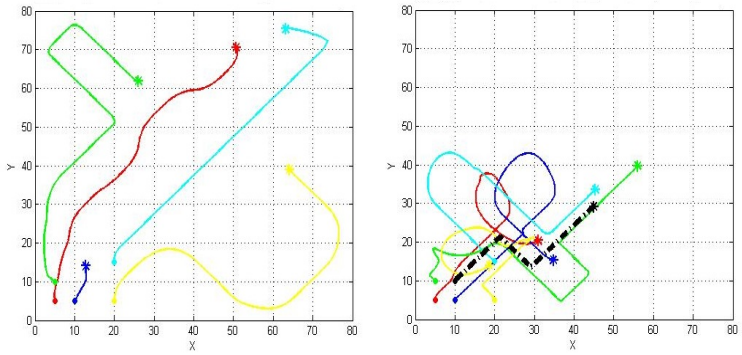


*Fig. 6.3:* Connectivity Maintenance

Nella figura 6.3 è messa a confronto l'esecuzione di un task di Effective Coverage senza preoccuparsi del problema di connettività (a sinistra), e considerando la soluzione con CDF (a destra). Lo swarm è costituito da DFs tutte con *fov* ed il range di comunicazione è di $40m$. La $CDF$ è invece parametrizzata nel seguente modo:

$$C(A, p_{q1}, p_{q2}, \sigma_{q1}, \sigma_{q2}) = [3, p_{Vq1}, p_{Vq2}, 12, 12]$$

Mentre nel primo caso la limitazione sul range non viene rispetta, dato che non è neppure una variabile considerata nel problema, nel secondo caso la connettività viene mantenuta a discapito di una velocità di movimento degli agenti minore. La linea spezzata in colore nero indica invece il moto della CDF. In figura 6.4 il problema viene complicato, ma comunque risolto, con l'aggiunta di ostacoli scenici.

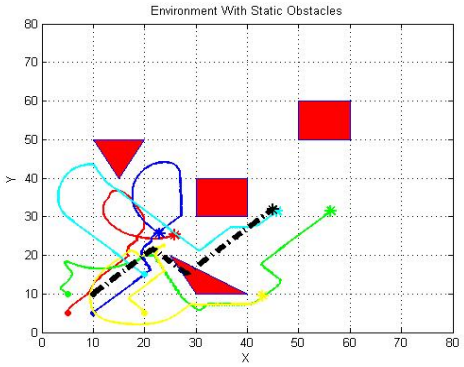


*Fig. 6.4:* Connectivity Maintenance with Obstacles

Una soluzione alternativa a quella appena trattata può essere realizzata assegnando dinamicamente il ruolo di agente virtuale ad un elemento reale dello swarm. Considerando di volta in volta l'agente che sta *svolgendo meglio degli altri il task*, e che quindi si presume abbia nelle sue vicinanze la maggior richiesta di risorse, è stato pensato di far coincidere la posizione di massima intensità della CDF con quella di questo agente. Il criterio di assegnamento scelto in questa tesi è legato definizione del seguente parametro chiamato *Actual Contribute*:

$$AC_i^k = \int_Q d_i^k(p, q) d_*^k(q, t) dq \tag{6.4}$$

Dove con $i$ si considera l'agente generico e con $k$ il task in esecuzione. Ciascun agente, utilizzando un filtro di Consensus Tracking, dovrà stimare anche i valori di $AC_j^k$ degli altri elementi dello swarm. Per far si che tutti gli agenti possano avvicinarsi alla posizione del Leader (sia esso virtuale o reale)

è inoltre opportuno rallentarne la velocità. In figura 6.5 è utilizzata questa tecnica per risolvere il medesimo task proposto in figura 6.3.

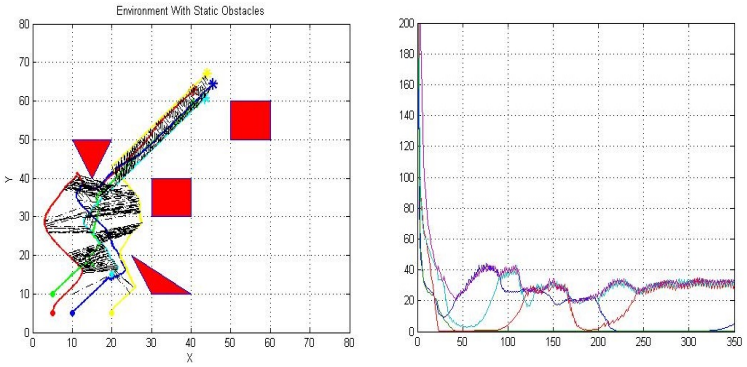


*Fig. 6.5:* Connectivity Maintenance with Leader Election and the Agents ACs Dynamics

Per evitare che ci sia una frequenza troppo elevata di assegnamenti è possibile ricorrere a soluzioni pratiche specifiche, come nel caso illustrato di figura 6.6, in cui viene garantito che un agente, una volta diventato leader, mantiene tale proprietà per un almeno un certo lasso di tempo. Occorre infine ricordare che un numero elevato di switching comporta un rallentamento del moto dello swarm che però dà la possibilità agli agenti di avvicinarsi.

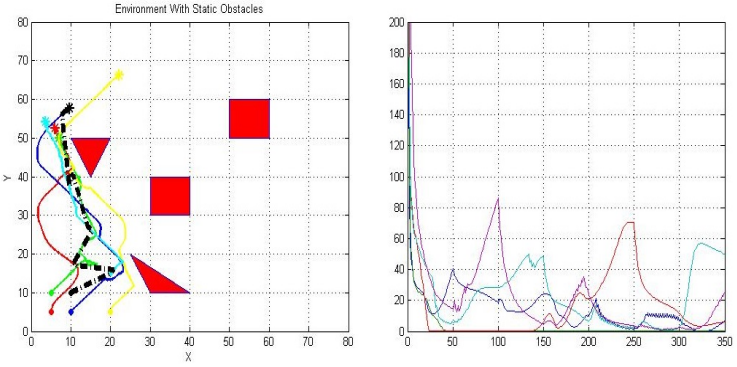


*Fig. 6.6:* Connectivity Maintenance with Leader Election (Low Freq. Switching) and the Agents ACs Dynamics

# 7. ASSEGNAMENTO DELLE RISORSE

## *7.1 Introduzione*

Come già accennato nel capitolo 4.4.5, l'assegnamento delle risorse può essere affrontato nel framework delle funzione descrittive come un problema di minimizzazione del funzionale di costo. La soluzione che ne deriva può però essere affetta da problematiche legate alla presenza di minimi locali e non è quindi garantito il raggiungimento del miglior assegnamento possibile.

Si propone pertanto l'utilizzo di un algoritmo di assegnamento deterministico (in contrasto con quanto fatto ad esempio in [23]), dove il problema viene studiato nei seguenti termini: «*Dato un insieme di TDF* $\{DF_1, DF_2, \dots\}$*, ed un insieme di ADF* $\{ADF_1, ADF_2, \dots\}$*, qual è la legge di assegnamento agente-task migliore?*».

## *7.2 L'algoritmo Ungherese*

Il "Problema di Assegnamento" può essere studiato su un grafo bipartito $G = (N, A)$ dove:

- $N$ è l'insieme dei nodi definito su due sottoinsiemi separati per cui $N = N_1 \bigcup N_2$.
- $A$ è l'insieme degli archi definito secondo il prodotto cardinale $A = N_1 \times N_2$.

Nel framework delle funzioni descrittive $N_1$ rappresenterà l'insieme delle TDFs mentre $N_2$ rappresenterà quello delle ADFs. Occorrerà così trovare, dopo aver definito un peso/costo per ogni arco, l'assegnamento di costo massimo costituito dall'insieme $B \subseteq A$ tale che:

$$B = \arg\max\{C(A') : A' \subset A, \forall i \in N_1 : \exists j \in N_2 : (i,j) \in A' \forall j \in N_2 : \exists i \in N_1 : (i,j) \in A'\} \quad (7.1)$$

Dove $C(A') = \sum (i,j) \in A' c_{i,j}$.

Per quanto riguarda il valore del costo $c(i,j)$ dell'arco che unisce i nodi $(i,j)$, è stato scelto di definirlo come la quantità di risorse che l'agente può apportare al task ovvero l'integrale della $ADF_i$ per l'esecuzione del task $j$.

$$c(i,j) = \int_Q d_i^j(p,q)dq \tag{7.2}$$

Dove la posizione $p$ in cui deve essere calcolata la ADF può essere scelta in maniera arbitraria purché si faccia allo stesso modo anche per gli altri agenti (in questo modo i valori calcolati possono essere correttamente paragonati). In aggiunta alla formula precedente è anche possibile definire il costo c(i,j) in termini percentuali, considerando il contributo risolutivo che ha un agente nell'eseguire un task. In altre parole si può definire un "degree of completeness":

$$DOC = c(i,j) = \frac{\int_Q d_i^j(p,q)dq}{\int_Q d_*^j(q,t)dq} \tag{7.3}$$

In questo modo se un agente fornisce un numero di risorse maggiore o uguale a quelle richieste dal task si otterrà un valore $DOC \geq 1$ ad indicare che è in grado di completare da solo e per intero il task; in caso contrario si otterrà banalmente $DOC \leq 1$. La situazione che si viene a creare, a prescindere dal tipo di valore $c(i,j)$ scelto, è quella espressa nella seguente tabella, dove con $N$ ed $M$ sono indicati rispettivamente il numero degli agenti ed il numero dei task che devono essere eseguiti.

| | $ADF_1$ | $ADF_2$ | ... | $ADF_N$ |
|---|---|---|---|---|
| $TDF_1$ | c(1,1) | c(1,2) | ... | c(N,1) |
| $TDF_2$ | c(2,1) | c(2,2) | ... | c(N,2) |
| ... | ... | ... | ... | ... |
| $TDF_M$ | c(M,1) | c(M,2) | ... | c(N,M) |

*Tab. 7.1:* Task Assignment Table

Nulla vieta inoltre di assegnare ad un valore di $c(i,j) = 0$ ad indicare che quell'agente non è assolutamente in grado di risolvere quel task o alternativamente che quell'agente non possiede risorse utili a fronteggiare quella determinata richiesta. Nel formalismo matematico proprio della programmazione lineare booleana è possibile scrivere il problema di assegnamento come segue, nelle ipotesi standard in cui gli elementi $c(i,j)$ sono dei costi che devono essere minimizzati e vale $N = M$:

**MINIMUM WEIGHTED PERFECT TASK ASSIGNMENT PROBLEM IN A BIPARTITE GRAPH**

$$\min \sum_{(i,j)\in A} c_{(i,j)} x_{(i,j)} \tag{7.4}$$

Con:

- $\sum_{(i,j)\in A} x_{(i,j)} = 1 \forall i \in N_1$
- $\sum_{(i,j)\in A} x_{(i,j)} = 1 \forall j \in N_2$
- $x_{(i,j)} = 0, 1 \forall (i,j) \in A$

Per la risoluzione del problema è quindi possibile applicare l'*algoritmo di Munkres* chiamato anche *metodo ungherese*, che realizza l'accoppiamento di costo minimo nelle ipotesi $N = M$. Anche se per questo algoritmo sono state elaborate delle soluzioni nel caso in cui $M \neq N$, come ad esempio quelle studiate da Bourgeois e Lassalle (vedi [14]), è comunque preferibile modificare di volta in volta il problema in modo da poter sempre gestire una tabella quadrata. Questa scelta viene fatta perché solo così è possibile includere nella soluzione finale tutti gli agenti presenti nello swarm. Gli accoppiamenti proposti in[14] (caso $N > M$) fornirebbero infatti al massimo $M$ assegnamenti e lascerebbero $N - M$ agenti non operativi. Il problema in cui $M \neq N$ può essere riportato allo studio di un problema "quadrato" $N = M$ tenendo conto di possibili raggruppamenti fra gli agenti. In questo modo $N$ sarà indice del numero di sottogruppi con cui è possibile suddividere lo swarm:

- Nelle ipotesi iniziali $N > M$ ogni agente definisce un insieme di sottotabelle quadrate in modo da coprire tutti i casi possibili di raggruppamento fra agenti; il numero di raggruppamenti per tabella sarà pari al numero di Starling di seconda specie.

- Per ciascuna di queste possibili sotto-tabelle viene applicato l'algoritmo di accoppiamento di Munkres.
- Il risultato dell'algoritmo definirà quindi per ciascuna sotto-tabella il *matching* sottogruppo/task ed anche il valore percentuale totale di risoluzione dei task che fornisce tale accoppiamento.
- Ogni agente eseguirà quel task definito dalla sotto-tabella il cui matching garantirà il massimo valore percentuale di completamento totale.
- Nei casi particolari in cui vale $M > N$, o quando il problema presentato nei punti precedenti non garantisce la risoluzione di tutti i task, vengono via via scartati i task meno incombenti fino ad ottenere nuovamente la condizione $N = M$.

Per comprendere quanto affermato, è stato fatto il seguente esempio dove sono presenti due task $\{DF_1, DF_2\}$ e tre agenti $\{ADF_1, ADF_2, ADF_3\}$. Ogni agente, noti i vari c(i,j), deve costruire un numero di tabelle pari al *numero di Stirling di seconda specie* che viene calcolato come segue:

$$Num.\ Tabelle = \binom{n}{k} = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n \tag{7.5}$$

Dove con $n$ sono indicati gli elementi dell'insieme e con $k$ è indicato il numero di gruppi con cui deve essere effettuata la suddivisione. Nel caso in esame vale $n = 3$ e $k = 2$ per cui il numero di tabelle è pari a 3. Utilizzando il parametro costo di tipo percentuale si ottengono le seguenti tabelle:

| **TAB. 1** | $ADF_1 + ADF_2$ | $ADF_3$ |
|---|---|---|
| $TDF_1$ | c(1,1)+c(1,2) | c(1,3) |
| $TDF_2$ | c(2,1)+c(2,2) | c(2,3) |
| **TAB. 2** | $ADF_1 + ADF_3$ | $ADF_2$ |
| $TDF_1$ | c(1,1)+c(1,3) | c(1,3) |
| $TDF_2$ | c(2,1)+c(2,3) | c(2,3) |
| **TAB. 3** | $ADF_2 + ADF_3$ | $ADF_1$ |
| $TDF_1$ | c(1,2)+c(1,3) | c(1,1) |
| $TDF_2$ | c(2,2)+c(2,3) | c(2,1) |

*Tab. 7.2:* Task Assignment Example

Con i valori da sostituire pari a:
$c(1,1) = 0.8$, $c(1,3) = 0.0$, $c(1,3) = 1.2$
$c(1,2) = 0.4$, $c(1,3) = 0.8$, $c(1,3) = 1.0$
$c(1,3) = 1.5$, $c(1,3) = 0.2$, $c(1,3) = 0.4$

Per utilizzare l'algoritmo di Munkres su ciascuna delle tre tabelle è di fatto necessario trasformare il problema di massimo in un problema di minimo, dato che, per come sono stati scritti i pesi $c(i,j)$, questi più che rappresentare dei costi rappresentano l'efficienza. Riportare un problema di massimo ad uno di minimo è comunque una cosa abbastanza banale e si può seguire la procedura qui indicata:

1. Si sottrae ad ogni elemento della tabella il valore 1 ad indicare che ogni task presente deve essere eseguito in maniera completa, cioè le richieste devono essere per lo meno esaudite al 100%.

2. Si modificano i valori dei coefficienti nel seguente modo:

$$\begin{cases} c(i,j)' = Inf, & \text{if } c(i,j) < 0 \\ c(i,j)' = c(i,j)^{-1}, & \text{if } c(i,j) > 0 \\ c(i,j)' = \max c(i,j)' + 1, & \text{if } c(i,j) = 0 \end{cases}$$

L'algoritmo di Munkres può così essere applicato seguendo questa procedura:

1. Si Sottrae ad ogni riga l'elemento minimo della stessa.

2. Si cerca un possibile accoppiamento task/agenti in considerazione alle caselle che contengono un valore nullo, in modo da ottenere così il costo relativo minimo.

3. Se non è possibile trovare un accoppiamento si sottrae ad ogni colonna l'elemento minimo della stessa in modo da ottenere degli zeri anche su ogni colonna della tabella.

4. Si cerca un accoppiamento possibile task/agenti dal costo relativo nullo e, se non è possibile trovarlo, si riparte dal punto iniziale in maniera iterativa.

I risultati di assegnamento sono quindi i seguenti, dove con il valore 1 è definito l'assegnamento fra il task e il gruppo di agenti:

| **TAB. 1** | $ADF_1 + ADF_2$ | $ADF_3$ |
|---|---|---|
| $TDF_1$ | 0 | 1 |
| $TDF_2$ | 1 | 0 |
| **TAB. 2** | $ADF_1 + ADF_3$ | $ADF_2$ |
| $TDF_1$ | 1 | 0 |
| $TDF_2$ | 0 | 0 |
| **TAB. 3** | $ADF_2 + ADF_3$ | $ADF_1$ |
| $TDF_1$ | 1 | 0 |
| $TDF_2$ | 0 | 0 |

*Tab. 7.3:* Task matching with munkres algorithm

L'unico assegnamento possibile è dunque quello espresso nella prima tabella, per cui la richiesta di risorse del primo task sarà soddisfatta al 150%, mentre quella del secondo task al 120%. E' comunque importante notare che, anche se il totale di percentuale di completamento è maggiore scegliendo di suddividere gli agenti come indicato nella tabella due ( 2.3+0.8 ), il fatto che il secondo task non venga portato a compimento ( 80% ) discrimina questa scelta. Qualora poi non fosse possibile arrivare ad una soluzione del problema, perché, ad esempio, non è presente alcuna tabella che assegna ciascun task ad almeno un agente, è sempre possibile scartare il task a priorità più bassa e ripetere l'algoritmo così come sopra indicato. I compiti precedentemente scartati potranno essere eseguiti successivamente, dopo aver portato a compimento quelli a priorità più elevata. Per ogni task è quindi indispensabile considerare un parametro che ne identifica la *priorità*: questo valore può o essere definito a priori o ad esempio essere calcolato in funzione della TDF.

Una ulteriore modifica alla soluzione del problema di *Task Assignment* può essere fatta in relazione alla percentuale di completamento con cui si vuole portare a termine un task. Una volta definite le tabelle, al posto di sottrarre il valore 1, ad indicare il completamento totale del task, è possibile sottrarre un valore diverso compreso nel range $0 \div 1$, ad indicare che quel compito deve essere eseguito perlomeno a quel valore percentuale. Se ad esempio viene sottratto un valore 0.8, ciò comporta che la richiesta di risorse espressa dal task deve essere coperta almeno all'80%. In questo modo si può rendere più vario e dinamico il problema, nel senso che in istanti di tempo diversi è

possibile gestire la risoluzione di un insieme di task andando ad aumentare o a diminuire il valore minimo percentuale di completamento. Ovviamente ciò comporta la realizzazione e la gestione di un vettore di dimensione pari al numero dei task che contiene tali valori:

$$\text{Minimum Degree of Completeness} = mDOC \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_M \end{bmatrix} \tag{7.6}$$

L'implementazione di questo algoritmo comporterà quindi l'aggiunta di alcuni parametri da dover stimare:

- Gli agenti dovranno comunicare i valori di $c(i, j)$.
- Un agente "particolare" di tipo leader, implementato anche da una *ground-station*, può comunicare le variazioni del vettore $mDOC$.

La soluzione proposta ha però un difetto riscontrabile nell'elevato numero di tabelle che un agente deve gestire in alcuni casi. Il numero di sottogruppi possibili aumenta infatti in funzione del numero di agenti coinvolti ed in funzione del numero dei task, il tutto secondo il numero di seconda specie di Stirling. Qui di seguito è proposta la tabella che indica, a seconda dei valori $N$ e $M$, il numero di raggruppamenti:

| **N/M** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | | | | | | | | |
| 2 | 1 | 1 | | | | | | | |
| 3 | 1 | 3 | 1 | | | | | | |
| 4 | 1 | 7 | 6 | 1 | | | | | |
| 5 | 1 | 15 | 25 | 10 | 1 | | | | |
| 6 | 1 | 31 | 90 | 65 | 15 | 1 | | | |
| 7 | 1 | 63 | 301 | 350 | 140 | 21 | 1 | | |
| 8 | 1 | 127 | 966 | 1701 | 1050 | 266 | 28 | 1 | |
| 9 | 1 | 255 | 3025 | 7770 | 6951 | 2646 | 462 | 36 | 1 |

*Tab. 7.4:* Starling Number of the Second Kind

## 7.3 Esempio

Un primo test per verificare il funzionamento dell'algoritmo di assegnamento delle risorse può essere fatto considerando nuovamente il problema di *Terget Assignment*. Questa volta però si hanno a disposizione tre diverse TDF con delle corrispettive DFs per gli agenti. Quello che si vuole ottenere è fare in modo che a ciascuna gaussiana che costituisce la TDF venga assegnato un agente con identiche proprietà.

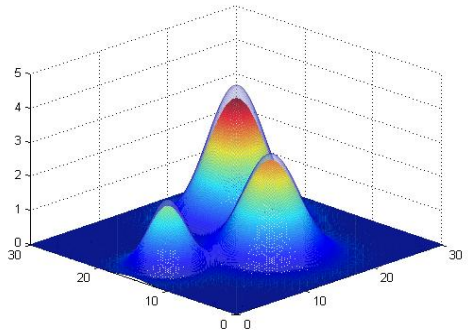


*Fig. 7.1:* Optimal Target Assignment

L'applicazione dell'algoritmo può quindi essere utilizzata prima dell'applicazione della legge di controllo proposta nel capitolo 4, in modo da scartare per ogni agente i target che non devono essere raggiunti. Per l'esempio in questione sono state scelte le seguenti DF:

$$TDF_1 = DF_G(A, p_{q1}, p_{q2}, \sigma_{q1}, \sigma_{q2}) = [2, 05, 15, 2, 2]$$
$$TDF_2 = DF_G(A, p_{q1}, p_{q2}, \sigma_{q1}, \sigma_{q2}) = [3, 15, 10, 3, 3]$$
$$TDF_3 = DF_G(A, p_{q1}, p_{q2}, \sigma_{q1}, \sigma_{q2}) = [4, 20, 20, 3, 3]$$
$$ADF_1 = DF_G(A, p_{q1}, p_{q2}, \sigma_{q1}, \sigma_{q2}) = [2, 10, 10, 2, 2]$$
$$ADF_2 = DF_G(A, p_{q1}, p_{q2}, \sigma_{q1}, \sigma_{q2}) = [3, 05, 10, 3, 3]$$
$$ADF_3 = DF_G(A, p_{q1}, p_{q2}, \sigma_{q1}, \sigma_{q2}) = [4, 10, 05, 3, 3]$$

IL risultato che si ottiene è quello aspettato così come si può vedere in figura 7.1 dove sotto ogni "campana" ve ne è un'altra di uguali dimensioni. Qualora si fosse risolto questo assegnamento utilizzando unicamente le legge di controllo proposta nel capitolo 4, il risultato ottenuto sarebbe stato quello di figura 7.2, in cui con le linee tratteggiate sono indicati i percorsi seguiti dagli agenti senza aver fatto task assignment.

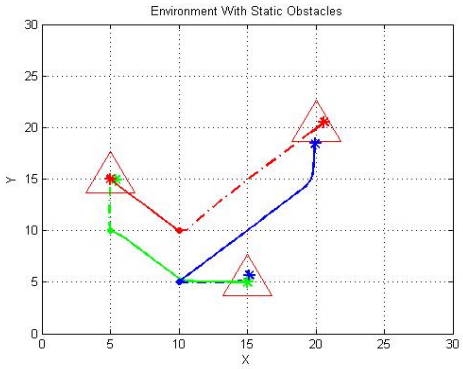


*Fig. 7.2:* Target Assignment with Resource Allocation (solid line) VS Simple Target Assignment (dot line)

# 8. HARDWARE TESTBED

## *8.1 Introduzione*

Verificate all'interno dei capitoli precedenti le capacità risolutive del framework, è stato deciso di effettuare un test anche in una applicazione reale. Per questa prova sono state utilizzate due "mini-car" 4WD radio-controllate di proprietà dell'Università di Pisa che andranno a costituire i due *agenti reali* dello swarm. Dato che entrambi questi veicoli non posseggono un hardware sufficiente a far girare in real-time gli algoritmi proposti nella tesi, è stata adottata una soluzione di ripiego che sfrutta l'utilizzo di *agenti virtuali* simulati su PC. Questi ultimi genereranno i Way-Point che verranno poi raggiunti dalle macchinine con una guida di tipo LOS Steering.

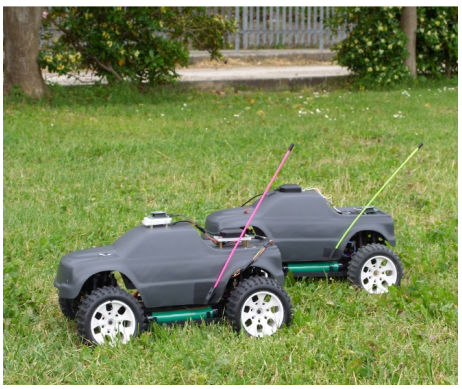

*Fig. 8.1:* Mini-Car Testbed

## *8.2 Mini-Car*

Le Mini-Car sono delle *HSP 1/10 Scale Brontosaurus Electric 4WD Monster Truck* modificate con l'aggiunta dell'elettronica necessaria per rendere

possibile lo sviluppo di compiti cooperanti di base (vedi ad esempio in [24] e [25] l'applicazione *pursuer/thief* ). Le caratteristiche principali possono essere così schematizzate:

- Alimentazione con batteria da modellismo da $7.2V$.
- Controllo Elettronico della velocità *ESC*.
- Quattro ruote motrici.
- Larghezza: $310mm$, Lunghezza: $400mm$, Altezza: $185mm$.
- Peso: $3Kg$. Carico massimo: $1Kg$.


*Fig. 8.2:* Mini-Car

Per migliorare le performances di comunicazione, navigazione e controllo sono stati aggiunti i seguenti componenti elettronici:

- **Ublox LEA-4H**: è il modulo Gps utilizzato come unico strumento di navigazione; il suo utilizzo è necessario non solo per calcolare la posizione e la velocità ma anche per effettuare la sincronizzazione degli agenti tramite il segnale periodico PPS. La frequenza di aggiornamento dei dati è pari a $4Hz$.
- **XBee-Pro**: Dispositivo utilizzato per la trasmissione wireless dei dati fra gli agenti.
- Circuito di alimentazione con in ingresso una tensione di circa $8V$ formato da due regolatori lineari monolitici: **7805CT** (per regolare i $5V$ necessari al $\mu C$) e **LM1086** (per regolare i $3.3V$ necessari al ricevitore GPS e al modulo XBee).

- **Pic 30F6014A**: Processore su cui è eseguito il firmware.

Il motore di trazione e di sterzo è quindi controllato usando un $\mu$C Pic 30F6014A che converte i comandi che arrivano da seriale RS-232 in segnali PWM. Un importante problema che viene risolto con l'utilizzo del GPS è invece quello della sincronizzazione fra gli agenti che è necessaria per evitare la collisione dei pacchetti durante la comunicazione.

L'utilizzo di XBee-Pro permette infatti due modalità di funzionamento: la prima in *Unicast Mode* che implementa un meccanismo di "*acknowledgment and retransmission*", e la seconda di tipo *Broadcast Mode*, dove non vengono però gestite le perdite di pacchetti a causa delle collisioni (vedi [24] e [25] per approfondimenti). Alla luce di quanto detto sopra è stato deciso di implementare un protocollo di tipo TDMA (*Time Division Multiple Access*), dove il tempo viene suddiviso in intervalli di durata fissa di $150ms$, a loro volta ripartiti in un numero fisso di $N$ slot (porzioni) temporali, con $N$ pari al numero degli agenti. Questo protocollo permette di utilizzare efficacemente una comunicazione di tipo half-duplex ($1Mbps$) e Broadcast Mode coniugando l'alta velocità di trasmissione alla collision avoidance sulla comunicazione. In questo modo è anche possibile utilizzare efficientemente tutto il canale. Il problema fondamentale che rende complessa l'implementazione di un algoritmo di questo tipo è la necessità di una entità centralizzata che sincronizzi i clock di tutti gli agenti indipendenti. Nell'applicazione in esame questo ruolo é affidato al ricevitore GPS attraverso il segnale PPS (Pulse per Second).

All'interno del firmware implementato sul $\mu C$ viene anche messo a disposizione un meccanismo di selezione del destinatario che però non viene sfruttato nella comunicazione fra veicoli. Ogni agente è infatti caratterizzato da un numero identificativo, e prima di utilizzare le informazioni ricevute da *XBee* controlla se nel campo "*destinatario*" di quel pacchetto è specificato il suo indirizzo o quello di broadcast. Se così non è, il pacchetto viene scartato. Questa funzionalità aggiuntiva viene invece sfruttata dalla stazione di terra che può così specificare per ciascun agente un diverso comando di controllo qualora ce ne sia bisogno.

Allo stato attuale del firmware è infine opportuno ricordare che uno slot ha dimensione $15ms$ (predisposizione per uno swarm con un numero di agenti pari a $N = 10$). Anche se nel test reale gli agenti saranno solo due, ed uno slot è utilizzato dalla stazione di terra, è stato deciso di mantenere invariato il numero di suddivisioni, lasciando così inutilizzati i sette slot rimanenti (i $15ms$ sono sufficienti per la comunicazione dei dati). In queste condizioni è possibile concludere che le comunicazioni avvengono ad una frequenza di $6.67Hz$ dato che ogni agente comunica una sola volta all'interno dei $150ms$.

### 8.2.1 Cinematica

Dal punto di vista del modello cinematico una mini-car può essere approssimata ad un triciclo, assumendo le condizioni di sterzata studiate da Ackermann (Ackermann's Steering).

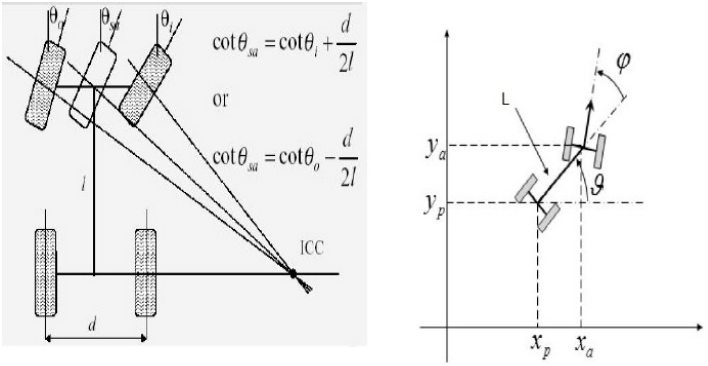


*Fig. 8.3:* Mini-Car: Kinematic

Le equazioni cinematiche che governano il moto sono definite in funzione delle tre variabili di stato $\{x, y, \theta\}$, e delle due variabili $\{v, \varphi\}$ che definiscono gli ingressi:

$$\begin{cases} \dot{x} = \cos(\theta)v \\ \dot{y} = \sin(\theta)v \\ \dot{\theta} = \frac{1}{L}\tan(\varphi)v \end{cases} \tag{8.1}$$

Mentre il vettore $[x, y]$ rappresenta la posizione del punto materiale rispetto al sistema di riferimento inerziale, $\theta$ indica l'orientazione del veicolo ovvero la direzione del vettore velocità rispetto alla terna inerziale. I due ingressi $[v, \varphi]$ rappresentano rispettivamente il modulo della velocità del veicolo e l'angolo di sterzo relativo rispetto a $\theta$. Per rendere il modello cinematico ancora più simile al veicolo reale è stato deciso di aggiungere al sistema di equazione 8.1 una discontinuità di tipo saturazione, dovuta al massimo grado di sterzo attuabile:

$$|\varphi| \leq \frac{\pi}{12} \tag{8.2}$$

### 8.2.2 Controllo del Veicolo

Il sistema di controllo del veicolo è stato realizzato e migliorato in diversi lavori di tesi di laurea triennale fra i quali è opportuno ricordare [24] e [25]. All'interno del firmware il controllore viene realizzato da una funzione che aggiorna i suoi output alla stessa frequenza di quella che si occupa della comunicazione (ovvero $6.67Hz$). Questa scelta, oltre che la più semplice da realizzare, è anche motivata dal fatto che il GPS, che è l'unico sensore di navigazione, fornisce nuovi dati ogni $4Hz$ e sarebbe quindi inutile aumentare la frequenza di esecuzione. E' stato così realizzato il seguente schema, rappresentativo di una architettura di controllo su più livelli:

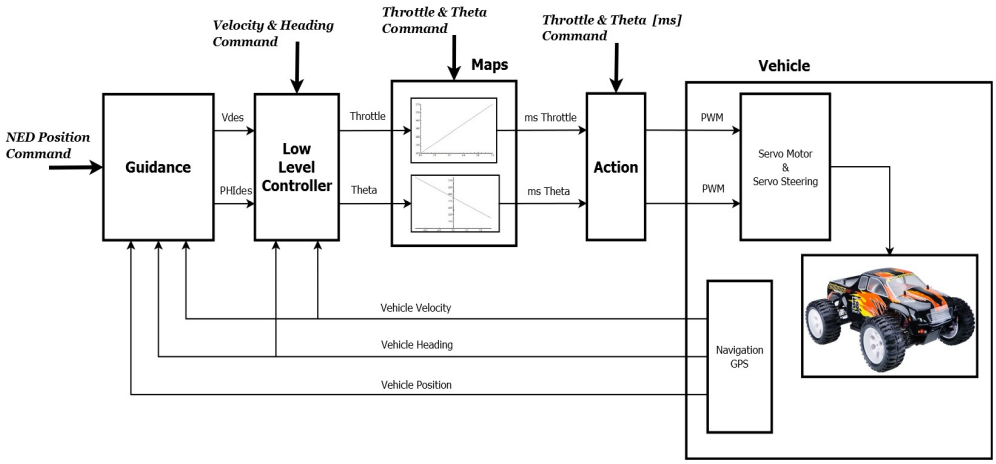


*Fig. 8.4:* MiniCar Control Scheme

Il livello più basso è realizzato intorno al funzionamento del blocchetto **Action** che definisce, in funzione del valore in $ms$ che ha in ingresso, il corretto *duty cycle* dei segnali PWM con cui sono comandati i due attuatori di trazione (throttle) e sterzo (theta/steering). I due valori $ms_{throttle}$ e $ms_{theta}$ possono o essere generati secondo delle opportune mappe caratteristiche definite all'interno del blocchetto precedente, oppure essere spediti direttamente da terra via *XBee* con un apposito comando. In quest'ultimo modo è possibile realizzare il pilotaggio della macchinina via joystick (a seconda del movimento della cloche verranno generati i due segnali). Questo primo livello di controllo è quindi funzionante in anello aperto dato che non c'è retroazione del valore effettivo di sterzata o di trazione. Occorre inoltre

ricordare che da terra è possibile spedire anche i due valori *throttle* e *theta* da cui sono calcolati i rispettivi valori in $ms$.

Il controllo di secondo livello, che in figura 8.4 viene definito dal blocchetto **Low Level Control**, si occupa di regolare la velocità e l'heading del veicolo (in terna NED) rispetto ai valori di riferimento imposti o da terra o dal sistema di guida. Questo processo è realizzato in anello chiuso e sfrutta la retroazione delle informazioni provenienti dal sensore di navigazione GPS. Sia per il segnale di velocità che per quello di heading il controllore utilizzato è di tipo *Proporzionale Integrativo* **PI** (vedi figura 8.5 e la relazione [24] per avere i dati effettivi), in cui la componente integrativa é limitata da una saturazione che evita un sovraccarico nelle situazioni in cui l'agente ha problemi a muoversi (ad esempio quando l'attrito dell'erba è elevato).

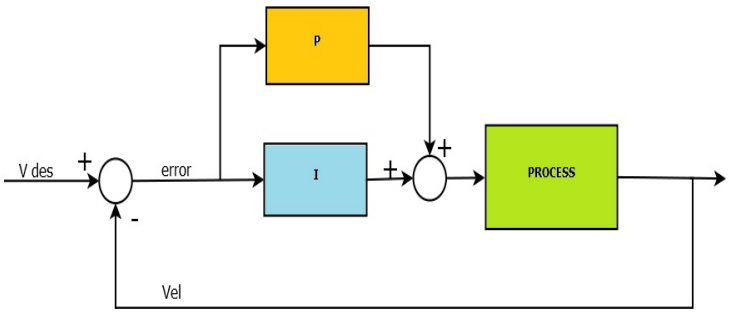


*Fig. 8.5:* Speed PI Controller

Il terzo ed ultimo livello è quello che comprende il sistema di guida che fornisce i riferimenti di velocità $V_{des}$ ed heading $\phi_{des}$ in relazione ad una legge di tipo *LOS Steering* (vedi Fossen [26]). Il sistema prende come ingresso un valore di posizione che viene spedito da terra in coordinate NED (un *Way-Point*). Sulla macchinina questo valore viene utilizzato per elaborare i riferimenti da fornire al *Low Level Control* che sono quindi in funzione della distanza $d$ che incorre fra il veicolo reale ed il Way-Point:

$$V_{des} = \begin{cases} 3m/s, & \text{if } d \geq 10m \\ 1.5m/s, & \text{if } 5m < d < 10m \\ 1m/s, & \text{if } 3m < d \leq 5m \end{cases} \tag{8.3}$$

$$\Phi_{des} = atan2(d_y, d_x); \tag{8.4}$$

### 8.2.3 Agenti Reali e Virtuali

Pur avendo a disposizione un $\mu C$ su ogni MiniCar, questo non è sufficiente a far girare in *real-time* gli algoritmi di controllo e consenso proposti in questa tesi. Per far fronte a questo inconveniente è stato quindi deciso di far girare tutto su PC, che gestisce quindi uno swarm di $N$ agenti che verranno chiamati *virtuali*. Ogni secondo (quindi con frequenza di $1Hz$), il PC si occupa di mandare la posizione raggiunta dai veicoli virtuali ai corrispettivi veicoli *reali* (definendo un destinatario nella comunicazione). Il sistema di guida riconoscerà così l'arrivo del nuovo Way-Point e definirà i riferimenti utili per i controllori di velocità e heading.

Considerando poi che le macchinine sono solamente due è stato deciso di realizzare su Simulink dei "*simulatori dinamici*" di MiniCar in modo che per ogni agente virtuale ci potesse comunque essere una "macchinina" in grado di inseguirlo. Questo espediente è servito per due motivi: il primo, testare la fattibilità della soluzione prima di implementarla direttamente sulle MiniCar, il secondo, osservare se anche con più veicoli reali vengono mantenute le condizioni di collision avoidance (i veicoli reali/simulati hanno bisogno di uno spazio di manovra al contrario di quelli virtuali). In questa concezione esisteranno quindi tre tipi di veicoli:

- **Virtuali**: realizzati con Simulink su PC, sono comandati secondo gli algoritmi di controllo proposti nel capitolo 4 e 5. La loro dinamica è definita dal singolo integratore. Con frequenza $1Hz$ i valori di posizione raggiunti diventano i Way-Point per i veicoli di tipo reale e simulato.

- **Reali**: sono le due MiniCar e sono guidati con una legge di tipo LOS verso i Way-Point generati dai veicoli virtuali. Per eludere gli errori di posizione che provengono dall'utilizzo GPS, il way-point viene considerato raggiunto dal veicolo una volta che questo si trova entro una distanza di $3m$.

- **Simulati**: funzionano come i veicoli reali ma sono però realizzati con Simulink. Vengono utilizzati quando il numero di veicoli virtuali è maggiore di due. La dinamica di questi veicoli è stata realizzata in maniera molto semplice, considerando il modello cinematico più un sistema del primo ordine per approssimare l'attuazione.

Un test realizzato su Simulink, fatto quindi con la sola presenza di agenti virtuali e simulati, ha dimostrato l'efficacia di questa soluzione. Per l'occasione è stato scelto di svolgere una operazione di Dynamic Coverage con uno swarm composto da 4 agenti:

$$TDF = d_*(q, k+1) = max(0, 0.1 + d_*(q, k) - D(p, q, k)) \forall q \in Q$$

$$ADF \longrightarrow \xi = [p_x, p_y, A, \theta, \sigma_x, \sigma_y, DFtype, task, K, K_2, fov] :$$
$$ADF_1 = [05, 05, 4, 0, 3, 3, DF_{SG}, DynCov, 3, -0.2, \pi/2]$$
$$ADF_2 = [10, 25, 4, 0, 3, 3, DF_{SG}, DynCov, 3, -0.2, \pi/2]$$
$$ADF_3 = [15, 05, 5, 0, 6, 6, DF_{SG}, DynCov, 3, -0.2, \pi/4]$$
$$ADF_4 = [25, 35, 5, 0, 3, 3, DF_{SG}, DynCov, 2, -0.2, \pi/1.5]$$

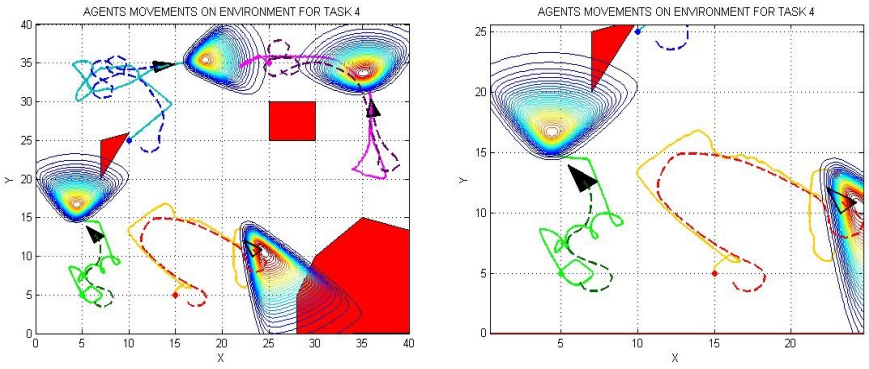


*Fig. 8.6:* Dynamic Coverage Trajectories: Virtual Car [solid lines] VS Simulated Car [dashed lines]

Dalla figura 8.6 è possibile vedere come le traiettorie sono quasi sempre coincidenti ad indicare che l'inseguimento da parte dei veicoli simulati viene eseguito correttamente. Il motivo è anche da riscontrare nel fatto che per gli agenti virtuali è stata scelta una velocità massima molto bassa di $0.1 m/s$.

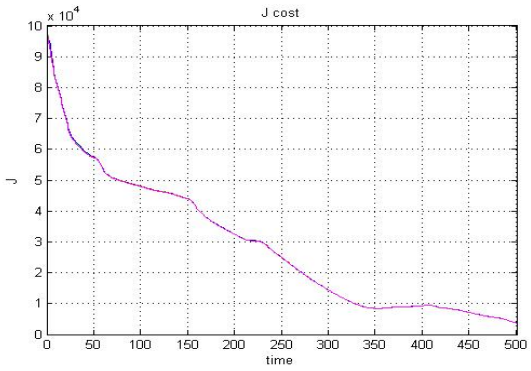


*Fig. 8.7:* Cost Functional

Questo espediente serve sia per agevolare i veicoli a raggiungere i Way-Point, ma è anche la condizione utile per ottenere un passo migliore nella discesa del gradiente (più il passo è piccolo e più si evitano oscillazioni intorno ad un minimo). L'andamento del funzionale di costo indica inoltre che il task viene correttamente eseguito dato che questo decresce e rimane così sotto una certa soglia di confidenza.

## 8.3 Il Test

L'hardware testbed è stato effettuato al campo sportivo CONI di Pisa utilizzando la pedana del salto in alto come spazio operativo. Pur avendo a disposizione l'intero prato è stato scelto di lavorare in questa zona sia perché è più asciutta (l'elettronica sulle macchinine non è coperta), sia perché l'erba può rallentare ed intralciare il cammino dei veicoli. Questa scelta ha però comportato lo svantaggio di dover lavorare in un ambiente ristretto in cui le incertezze che affliggono i dati provenienti dal GPS (errore di misura più il bias) possono compromettere la corretta esecuzione del task.

Durante la giornata di test sono state eseguite diverse prove, ognuna di queste caratterizzata dall'esecuzione di un particolare tipo di task. In questa sezione però, per brevità della trattazione, verranno riportati solo i risultati delle operazioni di Static Coverage e di Effective Coverage. In ogni caso, in tutti i task, è stato utilizzato uno swarm composto da tre agenti dove ai primi due di questi viene assegnata una Mini-Car reale, mentre al terzo viene fatto corrispondere un veicolo simulato.


*Fig. 8.8:* Hardware Testbed

Per far coincidere all'avvio di ciascun test la posizione dei veicoli virtuali con quella dei veicoli reali, (posizionati arbitrariamente nell'environment), è

stata realizzata una breve fase di set-up, sufficiente ad inizializzare la posizione dell'agente virtuale con il dato proveniente dal GPS della Mini-Car corrispondente. L'origine degli assi di riferimento viene centrato in un punto prefissato di cui sono note le coordinate ECEF. Per i casi qui analizzati si farà riferimento al sistema disegnato in figura 8.8. Un task di Static Coverage è stato eseguito utilizzando le seguenti funzione descrittive:

$$\xi = [p_x, p_y, A, \theta, \sigma_x, \sigma_y, DFtype, task, K, K_2, fov] :$$

$$TDF = d_*(q) = [25, -5, 4, 0, 8, 8, DF_G, StatCov, \dots] \forall q \in polyP$$

$$ADF_1 = [9.4, 1.5, 1, 0, 2, 2, DF_G, StatCov, \dots]$$
$$ADF_2 = [21, 1.5, 2, 0, 2, 2, DF_G, StatCov, \dots]$$
$$ADF_3 = [16.1, -4.2, 3, 0, 2, 2, DF_G, StatCov, \dots]$$

Il poligono $P$ all'interno del quale è definito il task altro non è che il perimetro della pedana utilizzata come spazio operativo. I vertici di tale poligono sono: $\{(0,0), (10,10), (30,-5), (25,-15), (0,0)\}$. La massima velocità consentita per un agente virtuale è $0.3m/s$.

I risultati del test sono riassunti nelle immagini seguenti dove vengono illustrate le traiettorie seguite per soddisfare la richiesta di risorse concentrata in $(25,-5)$. Come si può vedere dall'immagine di sinistra le Mini-Car e i veicoli simulati seguono gli agenti virtuali, arrestandosi ogni qual volta si trovano ad una distanza minore di $3m$.

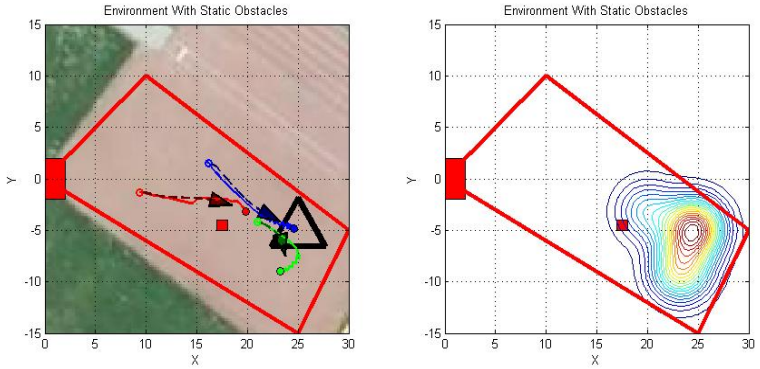


*Fig. 8.9:* Mini-Cars Trajectories in a Static Coverage Operation: Virtual Agents (solid line) VS Real/Simulated Cars (dashed line)

Le traiettorie delle Mini-Car illustrate nelle immagini sono ottenute dopo un filtraggio atto ad eliminare le incertezze provenienti dal GPS. Prima vengono eliminati gli outliers, dopodiché si sfrutta un filtro passa-basso del primo ordine per smussare i percorsi.

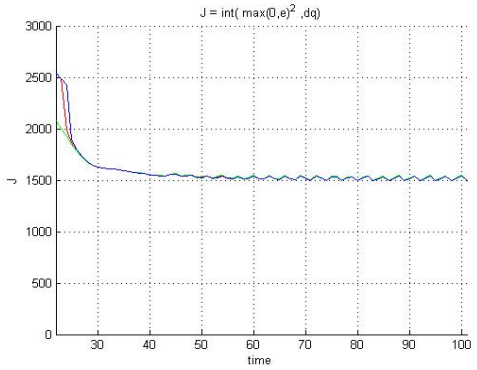


*Fig. 8.10:* Static Coverage: Cost Functional

L'andamento del funzionale di costo in figura 8.10 indica che gli agenti si sono mossi verso la risoluzione del task ma le loro qualità operative non sono sufficienti a portarlo a termine. Le oscillazioni intorno al valore finale sono invece dovute al passo troppo elevato nella discesa del gradiente causato dalla velocità massima con cui si muovono gli agenti virtuali pari a $0.3m/s$ (si potrebbero rallentare ma il test risulterebbe troppo lungo).

Un secondo test è stato fatto eseguendo un task di Effective Coverage all'interno del medesimo spazio operativo. Le DFs degli agenti sono state prese nel seguente modo:

$\xi = [p_x, p_y, A, \theta, \sigma_x, \sigma_y, DFtype, task, K, K_2, fov]$ :

$ADF_1 = [15, -1.5, 5, 0, 2, 2, DF_{SS}, EffCov, 2, -0.5, \pi/2]$
$ADF_2 = [23.9, -9.7, 1, 0, 2, 2, DF_G, EffCov, \dots]$
$ADF_3 = [16.1, 1.5, 2, 0, 1, 1, DF_G, EffCov, \dots]$

Nell'immagine di figura 8.11 sono indicate le traiettorie percorse dai veicoli. Anche qui l'inseguimento viene correttamente eseguito sia da parte del veicolo simulato che dalle Mini-Car. E' però importante osservare come il secondo veicolo reale, di colore verde tratteggiato passi molto vicino all'ostacolo e questo inconveniente è dovuto alla legge di guida scelta. Utilizzando

altre guide è sicuramente possibile migliorare questa situazione realizzando l'inseguimento con più precisione e con maggiori margini di sicurezza.

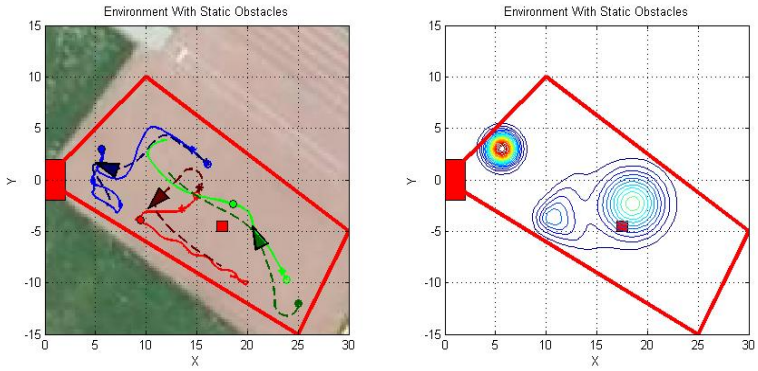


*Fig. 8.11:* Mini-Cars Trajectories in a Effective Coverage Operation: Virtual Agents (solid line) VS Real/Simulated Cars (dashed line)

In figura 8.12 è indicato l'andamento del funzionale di costo stimato dai tre agenti. Rispetto al caso precedente non si presentano oscillazioni e ciò è dovuto al fatto che i veicoli sono ancora in movimento quando viene interrotta la simulazione. Quanto appena detto conferma il fatto che le oscillazioni, causate da una velocità troppo elevata, si riscontrano solo in vicinanza di minimi locali ovvero quando l'agente è in procinto di arrestarsi.

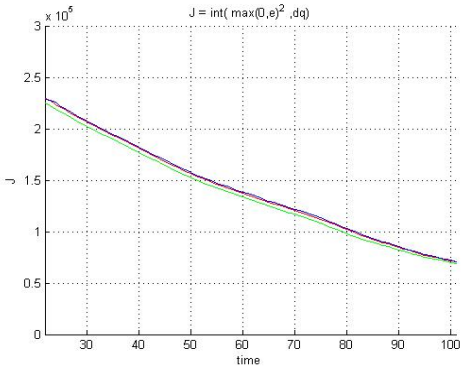


*Fig. 8.12:* Effective Coverage: Cost Functional

# 9. CONCLUSIONI

In quest'ultimo capitolo sono riportate le conclusioni finali sul lavoro svolto nella presente tesi. Sono così elencati sia gli obiettivi raggiunti, a conferma della validità delle procedure e delle soluzioni utilizzate, sia le problematiche ancora aperte, che potrebbero invece costituire gli spunti per ulteriori approfondimenti e ricerche.

- A partire dal framework delle funzioni descrittive, così come proposto in [1] e [2], in questa tesi sono state definite delle nuove tipologie di DF che aggiungono maggiore versatilità e adattabilità a quanto già esistente. Nello specifico vengono realizzate delle nuove DF con *field of view* e viene aggiunta la capacità di ruotare all'interno dell'ambiente operativo. Con delle simulazioni e dei test è stata poi provata l'efficacia delle nuove soluzioni in relazione a quanto presente nel vecchio framework.

- L'utilizzo di un algoritmo di tipo *discesa del gradiente*, come proposto in [2], è servito alla risoluzione del problema di ottimo senza però garantire la globalità della soluzione. Poiché il gradiente per sua natura conduce esclusivamente a punti critici locali, una scelta *ad hoc* di $J(p, q)$ potrebbe invece portare all'esistenza di un unico minimo globale. In ogni caso non è stato ancora trovata una soluzione a questo problema che rimane aperto a possibili sviluppi futuri.

- E' stato inserito nel framework un meccanismo di *Obstacle Avoidance* utilizzando la teoria dei "campi repulsivi". Ne è stata dimostrata l'efficacia sia dal punto di vista pratico che da quello teorico, utilizzando in quest'ultimo caso il metodo di Lyapunov.

- E' stato proposto un controllo di tipo decentralizzato basato sull'utilizzo degli algoritmi di consenso. Gli agenti, comunicando con i soli *neighbours*, riescono a portare a termine un problema di controllo globale risolvendolo attraverso una struttura distribuita. Nella risoluzione di questo problema è stato utilizzato l'algoritmo di consenso di tipo *Reference Tracking* proposto da Ren in [18].

- E' stato risolto, sempre con l'utilizzo del Framework delle Funzioni Descrittive, anche il problema di connettività in modo da garantire le ipotesi necessarie alla convergenza dell'algoritmo di Consensus Tracking. Per il caso specifico sono state proposte due soluzioni: la prima, in cui viene simulata la presenza di un agente virtuale a cui occorre stare vicini, e la seconda, in cui questo compito viene delegato agli agenti reali.

- E' stato proposto un possibile algoritmo di assegnamento deterministico delle risorse fra gli agenti utilizzando l'algoritmo di Munkres. Questo risolve un problema di tipo "minimum weighted perfect task assignment problem in a bipartite graph".

- E' stato effettuato un "*hardware testbed*" servendosi di due *minicar* di proprietà dell'Università di Pisa. Mancando l'hardware sufficiente per far girare in real-time gli algoritmi sui $\mu C$ presenti nelle macchinine, è stato scelto di optare per la soluzione descritta nel capitolo 8. Un possibile sviluppo potrebbe quindi essere quello di potenziare le macchinine sia con dei $\mu C$ più preformanti sia aggiungendo altri sensori di navigazione (IMU, optical flow sensors, ultrasonic sensor, ...).

*Appendice A*

# TEORIA DEI GRAFI

Nelle pagine che seguono, sono elencate una serie di definizioni utili riguardanti la teoria dei grafi. Per una migliore comprensione è stato scelto di affiancarle, là dove necessario, ad esempi pratici tutti facenti riferimento al grafo disegnato nella figura A.1. In ogni caso un generico grafo è rappresentabile attraverso la seguente espressione, dove $V$ rappresenta l'insieme dei nodi o *vertices* ed $E$ l'insieme degli archi o *edge*:

$$G(V; E)$$

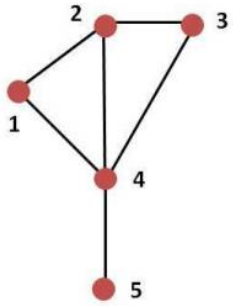


*Fig. A.1:* Graph Example

- Un Grafo si dice *Non Orientato* o *Undirected* quando ogni arco non possiede una direzione di preferenza. Alternativamente quando la connessione $i \longrightarrow j$ ha lo stesso valore della connessione $j \longrightarrow i$.
- Un grafo si definisce *Orientato* o anche *Directed Graph* se l'insieme degli archi è formato da coppie ordinate di nodi: in altre parole l'arco che collega i vertici $(v_i, v_j)$ può essere disegnato con una freccia che

va dal nodo $v_i$ al nodo $v_j$ . Un esempio di insieme di archi orientati può essere formalizzato attraverso una espressione del tipo $E = \{(1,2),(2,3),(2,4),(4,1),(3,4),(5,4)\}$.

- Un Grafo si dice *Connesso* se, presa una qualunque coppia di nodi, esiste sempre un cammino che ha quei vertici come iniziale e finale. Un grafo non orientato che non contiene cicli viene chiamato *Albero*. Pertanto, il grafo di riferimento A.1 risulta quindi connesso ma non è un albero.

- Dato un Grafo Connesso $G(V,E)$ si definisce *Spanning Tree* o *Albero Ricoprente*, un sotto-grafo di $G$ dove sono presenti tutti i nodi contenuti in $V$, ma in cui gli elementi di $E$ sono scelti in modo da non formare cicli. Un esempio può essere quello riportato nella figura A.2.

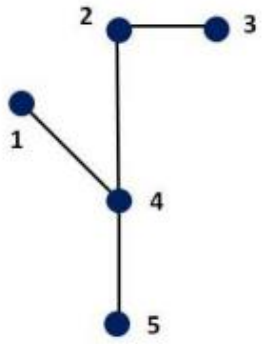


*Fig. A.2:* Spanning Tree Graph

- Un albero si dice *Pesato* se esiste una funzione che associa un valore numerico (cioè un peso) ad ogni arco.

- Un grafo orientato è *Fortemente Connesso* o *Strongly Connected* se per ogni coppia di nodi esiste un cammino orientato che li connette.

- Un grafo è semplicemente *Connesso* o *Weakly Connected* se esiste un cammino non orientato che connette ciascuna coppia di nodi arbitraria.

- Le due precedenti definizioni, nel caso di grafo non orientato, sono coincidenti.

- Si definisce *grado* di un nodo il numero di nodi adiacenti a quest'ultimo.

- Si definisce *Matrice dei Gradi* la matrice diagonale, semi-definita positiva, di dimensione $n \times n$ (con $n$ numero dei nodi), in cui l'elemento posto sulla riga $i - esima$ è calcolato con il valore del grado del nodo $i - esimo$.

$$\Delta(G) = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

- La *Matrice di Adiacenza* di un generico grafo rappresenta lo stato di connessione di ogni singolo nodo e definisce se esiste o meno un arco che collega il nodo $i - esimo$ al nodo $j - esimo$ ( nel caso di grafo orientato la matrice sarà simmetrica ). Il valore dell'elemento generico $a_{ij}$ può avere uno dei due seguenti valori:

$$a_{ij} = \begin{cases} 1, & \text{se } \exists E : v_j \longrightarrow v_i \text{ ovvero } E(j,i), \\ 0, & \text{altrimenti.} \end{cases}$$

$$A(G) = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

- La matrice di adiacenza $A(G)$ di un grafo non orientato è sempre una matrice simmetrica.

- Qualora il grafo sia orientato, è possibile invece descrivere anche la *Matrice di Incidenza* $I(G)$ che tiene conto dell'orientazione dell'arco fra due nodi. Preso un grafo in cui sono presenti $n$ nodi ed $m$ archi si ottiene una matrice di dimensione $n \times m$, in cui il singolo elemento vale:

$$i_{ij} = \begin{cases} +1, & \text{se l'arco } E(i,j) \text{ è entrante} \\ -1, & \text{se l'arco } E(i,j) \text{ è uscente} \\ 0, & \text{altrimenti.} \end{cases}$$

- Si definisce operatore *Laplaciano* di un grafo la matrice risultante dalla differenza fra la matrice dei gradi e quella di adiacenza.

$$L(G) = \Delta(G) - A(G) = \begin{bmatrix} 2 & -1 & 0 & -1 & 0 \\ -1 & 3 & -1 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ -1 & -1 & -1 & 4 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$

- Il Laplaciano è di notevole importanza poiché fornisce informazioni importanti sullo studio della rete. Studiando gli autovalori di $L(G)$ si può infatti notare che:

  1. Gli autovalori rispettano la seguente disequazione $\lambda_1 \leq \lambda_2 \leq \cdots \leq (2 * GradoMax)$.
  2. L'autovalore $\lambda_1$ risulterà sempre nullo ovvero $\lambda_1 = 0$.
  3. Il grafo non orientato risulta connesso se e solo se $\lambda_2 > 0$.

- Anche in presenza di grafi orientati è possibile parlare di alberi ricoprenti. Con il termine *Rooted Directed Tree* viene infatti sottinteso il grafo orientato nel quale è presente un nodo in cui tutti gli archi sono uscenti. Un grafo di questo tipo è anche *Spanning*, ovvero *ricoprente*, quando mette in connessione tutti i vertici. Mentre nel caso dei grafi non orientati la proprietà di *Rooted Directed Spanning Tree* è presente non appena il grafo è connesso, nel caso orientato è importante sottolineare che tale proprietà si verifica solo in presenza di un grafo *fortemente connesso* ma non vale il viceversa.

- Un albero con radice, o *Rooted Tree*, è un albero con un nodo particolare chiamato radice. Da questo nodo tutti gli archi sono uscenti.

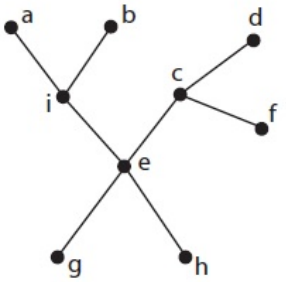


*Fig. A.3:* Rooted Tree

- Un albero orientato, o *Directed Tree*, è un albero i cui nodi sono collegati con archi orientati.

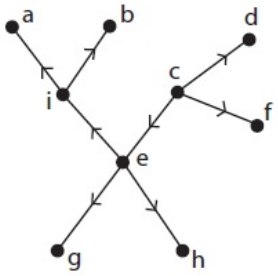


*Fig. A.4:* Directed Rooted Tree on node C

# BIBLIOGRAFIA


[1] Innocenti M., *Control of a Swarm of Heterogeneous Vehicles: Final Report*, 2009.

[2] Niccolini M., *Swarm Abstractions for Distributed Estimation and Control*, PhD Thesis, Università di Pisa, 2009.

[3] Innocenti M., Niccolini M., Pollini L., *Multivalued Consensus for a Swarm of Heterogeneous Agents I*, Icra 2010, September 2009.

[4] Innocenti M., Niccolini M., Pollini L., *Multivalued Consensus for a Swarm of Heterogeneous Agents II*, Icra 2010, September 2009.

[5] Niccolini M., Pollini L., Innocenti M., *Decentralized Control of Swarms with Collision Avoidance Implication.*

[6] Olfati Saber R., Fax A., Murray M., *Consensus and Coordination in Networked Multi-Agent Systems*, Proceeding of the IEEE, Jenuary 2007.

[7] Ren W., Beard R., *Distributed Consensus in Multi-Vehicle Cooperative Control* , Springer, 2008.

[8] Bullo F.,*Distributed Control of Robotics Networks – A Mathematical Approach to Motion Coordination Algorithms*, March 22, 2009.

[9] Hussein I., Stipanovic M., *Effective Coverage Control for Mobile Sensor*, Conference of Decision and Control IEEE, 2006.

[10] Murray M., Spanos P., Olfati Saber R., *Dynamic Consensus for Mobile Networks*, IFAC, 2005.

[11] Olfati-Saber R., Murray M., *Consensus Protocols for Networks of Dynamic Agents*, Proceedings of the American Control Conference, Denver, Colorado, June 4-6, 2003.

[12] Murphy Robert R, *Introducion to AI Robotics*, Ronald C. Arkin Editor, The MIT Press, 2000.

[13] Michael A. Goodrich, *Potential Fields Tutorial*, Computer Science Department TMCB, Brigham Young University.

[14] F. Bourgeois, J.C. Lassalle, *An Extension of the Munkres Algorithm for the Assignment Problem to Rectangular Matrices*, CERN , Geneve, Switzerland, 1971.

[15] Schenato L., *Distributed Estimation and Consensus* , Università di Padova, 7 July 2009, Siena.

[16] Antonelli G., *Interconnected Dynamic System* , IEEE Control System Magazine, February 2013.

[17] Howard A., Mataric M. and Sukhatme G., *Mobile sensor network deployment using potential fields: A distributed scalable solution to the area coverage problem*, 2002.

[18] Ren W., *Consensus Tracking under Directed Interaction Topologies Algorithms and Experiments* American Control Conference, Seattle, Washington, USA 2008.

[19] Giulietti F., Pollini L., Innocenti M., *Autonomous Formation Flight* IEEE Control Systems Magazine, December, 2000.

[20] Hussein I., Stipanovic M., *Effective Coverage Control for Mobile Sensor Networks With Guaranteed Collision Avoidance* IEEE Transaction on Control Systems Technology, VOl. 15, NO. 4, July 2007.

[21] Olfati-Saber R., *Flocking for Multi-Agent Dynamic Systems*, Technical Report CIT-CDS 2004-2005.

[22] Mesbahi M., Egerstedt M., *Graph Theoretic Method in Multiagent Network*, Princeton University Press.

[23] Niccolini M., Innocenti M., Pollini L.,*Multiple UAV Task Assignment using Descriptor Function.*

[24] Della Santina C., Aringhieri A.,*Sviluppo di un Testbed per l'analisi di algoritmi di cooperazione tra veicoli.*

[25] Tranzatto M.,*Sviluppo di un protocollo di comunicazione per veicoli cooperanti.*

[26] Fossen T. I.,*Lecture Notes on Guidance and Control of Vehicles* TTK 4190.

[27] Ren W., *Consenus Tracking under Directed Interaction Topologies: Algoritms and Experiments*, AACC American Control Conference, 2008.

[28] Kennedy J., Eberhart R. C., *Swarm Intelligence*, Morgan Kaufmann Publisher, 2001.

[29] Passino K. M., *Biomimicry for Optimization, Control, and Automation*, Springer, 2005.

[30] Bonebeau E., Dorigo M., Theraulaz G., *Swarm Intelligence: from natural to artificial systems*, Oxford University Press 1999.

[31] Niccolini M., Innocenti M., Pollini L., *Near optimal Swarm Deployment using Decsriptor Functions*, IEEE International Conferenceon Robotics and Automation, Vol.1, Anchorage, AK, USA, May, 2010.

[32] Bracci A., Innocenti M., Pollini L., *Cooperative Task Assignment Using Dynamic Ranking*, 17th IFAC World Congress, Seul, South Korea, July 2008.

[33] Ren W., Cao Y., *Distributed Coordination of Multi-agent Networks*, Springer, 2011.

[34] Hussein L. I., Stipanovich D., Wang Y., *Reliable Coverage Control Using Heterogenous Vehicles*, 46 IEEE Conference on Decision and Control, New Orleans, LA, USA, December 2007.

[35] Martionli M., Mondada F., Mermoud G., Correl N., Egerstedt M., Hsieh M., Parker L., Stoy K., *Distributed Autonomous Robotic Systems*, Springer, January 2013.

[36] Pallottino L., *Sistemi Robotici Distribuiti*, Laurea Specialistica in Ingegneria dell'Automazione/Laurea Magistrale in Automazione e Robotica, Centro E.Piaggio, Università di Pisa, Facoltà di Ingegneria.