UNIVERSITÁ DEGLI STUDI DI CAGLIARI
Dipartimento di Ingegneria Elettrica e Elettronica
Laurea Specialistica in Ingegneria Elettronica

SIMULAZIONE E ANALISI DI ALGORITMI DECENTRALIZZATI APPLICATI IN PARALLELO SU SISTEMI MULTI-AGENTE INTERCONNESSI

TESI DI LAUREA DI:
Daniele Rosa

RELATORE:
Prof. Alessandro Giua

Anno Accademico
2009 - 2010

Ringraziamenti

Ringrazio Prof. Giua e l'Ing. Mauro Franceschelli per avermi guidato e aiutato nel lavoro di questi mesi.

Ringrazio i miei colleghi Marco, Manu e Fede, ottimi compagni di studio e non solo.

Ringrazio le persone con cui ho condiviso questi anni, in particolare i ragazzi di Via Val D'Elsa, i Cavalli e i Tacchellinnu. Senza di voi probabilmente sarei arrivato qui prima, ma sicuramente sarei una persona peggiore.

Grazie a Romina, che ogni giorno con il suo sorriso rende tutto leggero.

Un immenso grazie alla mia famiglia, che mi ha sostenuto e incoraggiato per tutto questo tempo. Questo successo è tutto vostro!

E grazie a Dio anche io ce l'ho fatta...

Indice

Capitolo 1

Introduzione

1.1 Obiettivi della tesi

Lo scopo di questo lavoro è quello di simulare un sistema multi-agente su cui vengono applicati alcuni algoritmi di controllo decentralizzati in parallelo, che hanno come obiettivo quello di far disporre il plotone secondo una serie di formazioni desiderate. Tra le diverse possibili combinazioni di algoritmi che verranno analizzate, verranno valutate quelle maggiormente performanti a seconda delle condizioni applicative. Verrà fatta inizialmente una panoramica sulle problematiche relative ai *sistemi multi-agente*, argomento che nell'ultimo decennio ha suscitato fortemente l'interesse di numerosi ricercatori e studiosi di diverse discipline. Ci si concentrerà sugli aspetti legati all'automatica e verranno proposti alcuni algoritmi che vengono utilizzati per effettuare un controllo distribuito su una rete di agenti.

Il maggior contributo di questo lavoro è dato dallo sviluppo di un simulatore linguaggio MATLAB, che implementa un sistema multi-agente su cui vengono applicati contemporaneamente due algoritmi:

- Un algoritmo randomizzato di tipo *gossip* per la determinazione di un sistema di riferimento comune a tutti gli agenti, descritto in [1].
- Un algoritmo deterministico grazie al quale ciascun agente riesce a raggiungere una *posizione obiettivo*.

Verranno effettuate una serie di simulazioni per valutare gli effetti della sovrapposizione di questi due algoritmi sul plotone di agenti, e si cercherà di ottenere alcune regole che possano essere utilizzate per progettare una corretta implementazione di tale sistema. Verranno inoltre sviluppate una serie di funzioni

MATLAB che permettono di interfacciare il simulatore con un tool di realtà virtuale: il *VIRTUAL REALITY TOOLBOX*. Questa interfaccia permetterà di ottenere una rappresentazione virtuale, in 3D, dell'evoluzione del sistema nello spazio, che permette di visualizzare meglio il comportamento emergente del gruppo di agenti.

Il simulatore è stato sviluppato in maniera modulare, cercando di sfruttare i vantaggi della *programmazione orientata agli oggetti*, in modo tale che il codice sviluppato per questo lavoro possa essere riutilizzato, modificando solo le funzioni che implementano la regola di controllo locale, per la simulazione di altri algoritmi su sistemi multi-agente.

1.2 I sistemi multi-agente

I *sistemi multi-agente* (MultiAgent Systems, MAS) sono una classe di sistemi caratterizzati da un insieme di entità, gli agenti, che interagiscono tra di loro in un ambiente condiviso per raggiungere un obiettivo comune. È da sempre noto all'uomo, e facilmente osservabile in natura, come la collaborazione tra individui porti a risolvere facilmente dei problemi che, per il singolo, risultano impossibili o comunque molto difficili. Proprio per questo motivo negli ultimi anni lo studio dei *MAS* ha catturato l'attenzione dei ricercatori grazie alla molteplicità di aree scientifiche in cui questi possono essere utilizzati. Si parla di *MAS* in ambito economico, sociologico, filosofico e, ovviamente, informatico e elettronico.

Nell'ambito dell'automatica i *MAS* rappresentano un argomento di interesse sempre maggiore. Infatti, grazie allo sviluppo di sempre più potenti tecniche di controllo, di calcolo e di comunicazione, le potenzialità applicative sono in forte crescita. Alcune delle numerose applicazioni attuali sono elencate qui sotto.

- Coordinamento di veicoli autonomi (aerei, sottomarini, satelliti, veicoli spaziali).
- Monitoraggio ambientale (stima e prevenzione dell'inquinamento o degli incendi).
- Operazioni di ricerca e soccorso.
- Coordinamento di robot mobili.

Con il termine *agente* ci si riferisce a un sistema autonomo, con una dinamica propria, in grado di interagire con l'ambiente che lo circonda e di prendere decisioni autonome per raggiungere un determinato obiettivo.

Le applicazioni più importanti dei sistemi multi agente in automatica sono relative ai *sistemi di controllo interconnessi* (networked contol systems, NCS). In un sistema di controllo interconnesso basato su un sistema multi agente, o più semplicemente *sistema multi agente interconnesso* (networked multi agent system), esiste tra gli agenti una vera e propria rete di comunicazione che permette a ciascuno di loro di scambiare informazioni con gli altri, e di modificare il proprio comportamento in base alle informazioni che riceve. Questo scambio di informazioni opera su ogni agente una vera e propria azione di controllo sul proprio stato interno. La presenza di una rete di comunicazione tra gli agenti permette di applicare su di loro degli algoritmi, prevalentemente distribuiti, in grado di far ottenere dei comportamenti di gruppo intelligenti. Alcuni vantaggi dei sistemi di controllo multi agente, rispetto ai classici sistemi isolati, vengono ora elencati.

- I sistemi multi-agente sono in grado di risolvere sotto-problemi in parallelo riducendo il tempo per il completamento del compito dell'intero sistema e riducendo il costo computazionale (sotto-problemi più semplici del problema principale).

- L'assenza di un singolo centro decisionale rende il sistema complesso maggiormente affidabile e robusto ai guasti.

- Utilizzando un sistema multi-agente è possibile progettare agenti diversi e specializzati possibilmente più semplici e meno costosi di un singolo agente complesso.

Ovviamente l'utilizzo di sistemi multi agente pone una serie di nuovi problemi e difficoltà da affrontare.

- Gestire (coordinare) un elevato numero di agenti.

- Gestire in modo uniforme agenti non omogenei.

- Gestire una informazione non centralizzata, ma locale.

- Gestire l'elevata necessità di comunicazione.

1.3 Panoramica sui problemi di consenso

Uno degli argomenti di maggior interesse nello studio dei sistemi multi agente interconnessi è rappresentato attualmente dallo sviluppo di algoritmi decentralizzati in grado di generare movimenti coordinati.

In una rete di agenti, il termine *consenso* indica il raggiungimento di un accordo su una determinata grandezza di interesse che dipende dallo stato interno di tutti gli agenti.

Un *algoritmo di consenso* (o *protocollo*) è una legge di interazione che regola lo scambio di informazioni tra ciascun agente e i suoi vicini. Ogni agente utilizza lo stesso algoritmo condiviso da tutti, e prende le proprie decisioni in base alle informazioni disponibili localmente e a quelle che riceve dagli altri.

Si parla di *algoritmi di consenso sulla media* (*average-consensus algorithm*) quando lo scopo dell'algoritmo è quello di far convergere il valore di una determinata grandezza, che caratterizza lo stato di ogni agente, alla media dei valori iniziali che tale grandezza assume per ciascun agente.

In [2] viene ripercorsa la storia degli algoritmi di consenso, a partire dai primi approcci alla computazione distribuita di DeGroot, passando per le prime applicazioni, risalenti agli anni '80, nel campo dei sistemi di controllo, fino ai giorni nostri. Sempre in [2] vengono proposte una serie di applicazioni e di metodologie che riguardano questo tipo di problemi. In [6] vengono presentati alcuni algoritmi di consenso, lineari e non lineari, per reti di agenti dinamici.

Alcuni dei più comuni problemi di coordinamento sono i seguenti.

- Problemi di *Rendez-vous*.
- Problemi di *Flocking*.
- Problemi di *Deployment*.

1.3.1 Il problema del Rendez-Vous

Il problema del *rendez-vous* consiste nel far convergere un certo numero di agenti, inizialmente sparsi nello spazio. Questo problema può essere rivisto come un caso particolare del riposizionamento degli agenti in predeterminate formazioni geometriche o distribuzioni. Le problematiche relative a questa classe di problemi sono molteplici e sono strettamente legate alle condizioni in cui gli agenti lavorano e alla modalità con cui avviene lo scambio di informazioni. Ad esempio, supponiamo che un plotone di aerei automatizzati debba convergere in un

determinato punto. Se tutti i veicoli hanno un sistema di riferimento comune, e ciascuno di loro è in grado di determinare la propria posizione rispetto a questo sistema di riferimento, è sufficiente che ciascuno di loro riceva le coordinate del punto di convergenza per raggiungere l'obiettivo che ci si è prefissati. Risulta invece molto più complicta la situazione in cui i veicoli non abbiano un sistema di riferimento in comune. Ancora, la situazione in cui ogni agente è in grado di scambiare informazioni con tutti gli altri è ben diversa dalla situazione in cui un agente è in grado di comunicare solo con una parte del gruppo. Alcuni possibili soluzioni del problema del rendez-vous sono presentati in [2, 6, 7, 13]

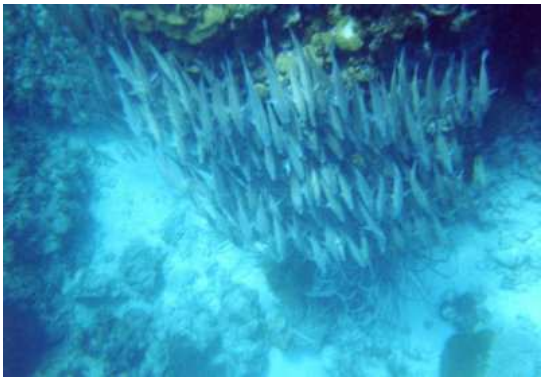
1.3.2 Il problema del Flocking

Anche questo è un problema di coordinamento di un grande numero di agenti che interagiscono per raggiungere un obiettivo di gruppo. In particolare l'obiettivo è di far disporre gli agenti nello spazio secondo formazioni prestabilite, e fare in modo che il plotone si muova mantenendo queste formazioni. In [4] viene messo in evidenza come problemi di questo tipo nascano dall'osservazione del comportamento di alcuni gruppi biologici che si muovono in modo coordinato formando sciami ordinati (api, uccelli, pesci...). Ogni animale decide autonomamente il proprio comportamento in base alla percezione locale della dinamica di gruppo. Il movimento che si percepisce è quindi il risultato di una densa interazione tra i comportamenti relativamente semplici di ogni individuo. La Figura 1.1 rappresenta due esempi di questo fenomeno. Gli algoritmi di *flocking* dettano le regole fondamentali che ogni agente deve rispettare in modo da ottenere dei comportamenti simili per tutti gli individui di un dato insieme. La teoria e l'analisi di alcuni problemi di questo tipo viene discussa in [3].

1.3.3 Il problema del Deployment

L'obiettivo del *deployment* è quello di ottimizzare la comunicazione e il movimento degli agenti, in modo da esplorare un dato spazio. Si cerca, quindi, di massimizzare la distanza tra gli agenti in modo che rimanga comunque attiva la connessione tra di loro. Questo tipo di problemi vengono affrontati in larga scala in campo militare (mappatura e bonifica di zone inesplorate) e nel campo delle esplorazioni spaziali.

(a) Uccelli in formazione



(b) Banco di pesci

Figura 1.1: Esempi di Flocking riscontrabili in natura

1.4 Gli algoritmi di tipo gossip

Gli *algoritmi di tipo gossip* (o semplicemente algoritmi gossip) sono anche definiti *epidemici* in quanto la modalità di trasmissione dell'informazione tra i nodi è simile alla diffusione di una malattia. Questi algoritmi si basano su una modalità di scambio delle informazioni che si può paragonare alla diffusione del 'pettegolezzo' tra la persone, e per questo prendono il nome di 'gossip'. Generalmente basano su un protocollo asincrono e inaffidabile per lo di scambio dell'informazione, ma allo stesso tempo risultano molto semplici e di ampia applicabilità, e per questo motivo stanno diventando una applicazione standard nelle reti di telecomunicazioni. Dal punto di vista dei sistemi multi-agente, le caratteristiche comuni di questi algoritmi sono elencate qui sotto.

- Ciascun agente ha una visione solo parziale del sistema di cui fa parte.
- Ciascun agente può comunicare solo con una parte degli agenti che fanno parte del sistema, e può utilizzare solo le informazioni che riceve da questi.
- Ciascun agente decide volta per volta con quali agenti comunicare, e la comunicazione può anche non essere sincronizzata.
- Gli agenti condividono con gli altri solo una porzione delle informazioni che possiedono.

Ad ogni iterazione avviene uno scambio di informazione solo tra una parte degli agenti che compongono il sistema, e gli agenti coinvolti in questo processo di comunicazione aggiornano localmente il proprio stato. Proprio per queste caratteristiche, gli algoritmi gossip vengono utilizzati per risolvere problemi di consenso in condizioni in cui la possibilità di comunicazione tra gli agenti risulti limitata. Il *pairwise gossip* (*gossip a due a due*) è un particolare tipo di algoritmo in cui lo scambio di informazione avviene a coppie di agenti. Alcuni algoritmi di questo tipo sono analizzati in [8, 9, 10]

1.5 Struttura della tesi

Nel Capitolo 2 verrà presentata la teoria algebrica dei grafi, e verranno proposti alcuni metodi per la risoluzione dei problemi di consenso basati su di essa. Verranno presentate le formulazioni più classiche dei problemi di consenso

per sistemi multi agente interconnessi a tempo continuo e a tempo discreto, tempo-invarianti e tempo-varianti. Verranno inoltre presentati gli algoritmi di consenso randomizzati, e in particolare quelli di tipo gossip.

Nel Capitolo 3 verrà descritto il sistema che si vuole analizzare e verranno presentati gli algoritmi utilizzati per le simulazioni. Verrà analizzato un sistema multi agente interconnesso in cui ogni agente vede l'ambiente che lo circonda secondo un proprio sistema di riferimento locale. L'obbiettivo è quello di fare in modo che gli agenti si accordino su un frame comune e si dispongano nello spazio secondo in modo da ottenere determinate formazioni di gruppo desiderate. Per ottenere questo verranno applicati simultaneamente due algoritmi:

- un algoritmo randomizzato di tipo gossip per il consenso su un frame comune;
- un algoritmo distribuito per spostare ogni agente nella posizione desiderata.

Il contributo più importante di questo lavoro è proprio lo studio degli effetti della sovrapposizione di diversi algoritmi in parallelo sul sistema multi agente.

Nel Capitolo 4 verrà descritto il simulatore, realizzato in MATLAB, che implementa il sistema multi agente descritto nel Capitolo 3. Verrà descritta la classe Agent, che rappresenta il fulcro del simulatore realizzato utilizzando la programmazione ad oggetti. Verrà descritto il funzionamento delle funzioni più importanti, che implementano gli algoritmi di controllo descritti nel Capitolo 3, e verrà presentato presentato il *VIRTUAL REALITY TOOLBOX*, un software per la rappresentazione di mondi virtuali che verrà interfacciato al simulatore.

Nel Capitolo 5 verranno riportati i risultati dei diversi test sul sistema, descritto nel Capitolo 3, che sono stati effettuati col simulatore descritto nel Capitolo 4. I test descritti saranno di diversi tipi:

- test per la verifica del funzionamento del simulatore;
- test per l'analisi del comportamento del sistema sotto l'azione dei due algoritmi applicati separatamente;
- test per l'analisi del comportamento e delle prestazioni del sistema sotto l'azione simultanea dei due algoritmi.

Verranno proposte delle varianti adattative per l'algoritmo che regola lo spostamento degli agenti verso la posizione obiettivo. Verranno simulate tutte le possibili combinazioni che permettono di applicare gli algoritmi in parallelo. Verrà

fatta un'analisi della convergenza del sistema alle posizioni desiderate, e si cercherà di valutare i limiti applicativi degli algoritmi proposti e si confronteranno le prestazioni delle diverse varianti implementative.

Capitolo 2

Metodi matematici per la risoluzione di problemi di consenso

In questo capitolo vengono presentati alcuni metodi per la risoluzione dei problemi di consenso. Esistono in letteratura diverse formulazioni di questo tipo di problemi, e sono state presentate diverse soluzioni. Qui riportiamo solamente le formulazioni più classiche dei problemi di consenso e proponiamo delle soluzioni che si basano sulla teoria algebrica dei grafi, ed in particolare sulla *matrice Laplaciana*.

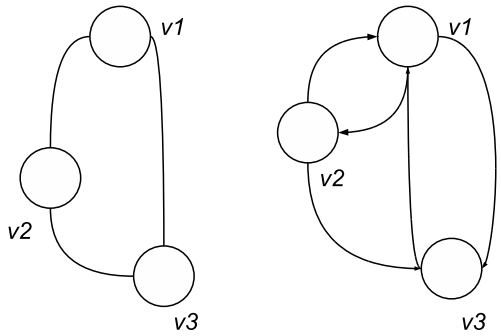
2.1 Teoria algebrica dei grafi

Definizione 2.1.1. *Un* grafo *è una coppia $G = (V, E)$ in cui V rappresenta l'insieme dei* nodi *(o vertici), e $E \in V \times V$ rappresenta l'insieme degli* archi.

Gli archi sono rappresentati, quindi, da coppie di nodi (i, j) con $i, j \in V$.

Definizione 2.1.2. *Un grafo si dice* ***orientato*** *se per gli archi vale $(i, j) \neq (j, i)$ con $i, j \in V$. Un grafo orientato si dice* ***bilanciato*** *se per ogni nodo $v_i \in V$ la somma degli archi entranti è pari alla somma degli archi uscenti.*

Definizione 2.1.3. *Un grafo si dice* ***non orientato*** *se $(i, j) = (j, i)$ con $i, j \in V$. Ne segue che un grafo non orientato è un grafo bilanciato.*



(a) Grafo non orientato (b) Grafo orientato

Figura 2.1: Esempi di grafo a 3 nodi.

I grafi possono essere rappresentati in diversi modi. La rappresentazione più comune è quella grafica (Figura 2.1), in cui i nodi vengono rappresentati con dei cerchi etichettati, e gli archi con delle linee che congiungono i nodi. Se un grafo è *orientato* (Figura 2.1b) queste linee avranno una freccia che ne indica il verso (l'arco (i, j) sarà rappresentato da una freccia che va dal nodo i al nodo j). Se il grafo è *non orientato* le linee non avranno un verso (Figura 2.1a).

Indichiamo con $N = |V|$ il numero di vertici di un grafo G. Due nodi si dicono adiacenti se esiste un arco che li collega.

Definizione 2.1.4. *Un **cammino** di G è una sequenza di archi successivi*

$$\{e_1, e_2, \dots e_k \in E\}$$

in cui ogni arco è formato da una coppia di nodi che contiene un nodo dell'arco precedente. Se il grafo è orientato ogni coppia deve avere come nodo sorgente il nodo destinazione dell'arco precedente. Un nodo $i \in V$ si dice raggiungibile *dal nodo $j \in V$ se esiste in G un cammino che parta da i e arrivi a j.*

Definizione 2.1.5. *Un grafo è **connesso** se esiste sempre un cammino fra qualsiasi coppia di nodi, mentre è **fortemente connesso** se esiste sempre un cammino orientato fra qualsiasi coppia di nodi di V.*

Un grafo non orientato connesso, sarà anche fortemente connesso.

I grafi possono essere descritti per mezzo di opportune matrici, in modo da formalizzare matematicamente i problemi da esso definiti.

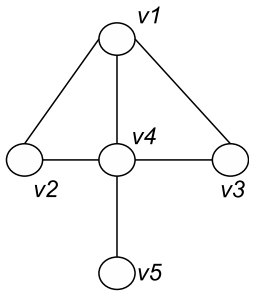


Figura 2.2: Grafo dell' esempio 2.1.1

Definizione 2.1.6. *La **matrice di adiacenza** **A** di un grafo G è la matrice $N \times N$ i cui elementi sono:*

$$a_{ij} = \begin{cases} 1 & se\ (i,j) \in E \\ 0 & se\ (i,j) \notin E \end{cases}$$

Gli elementi della matrice di adiacenza vengono identificati con a_{ij}.

In un grafo non orientato, in cui $(i,j) = (j,i)$ con $i, j \in V$, la matrice di adiacenza è simmetrica.

Definizione 2.1.7. *Per ogni nodo $v_i \in V$ si definisce insieme dei nodi adiacenti N_i (o insieme dei vicini) il sottoinsieme di V definito in questo modo:*

$$N_i = \{j \in V : a_{ij} \neq 0\}; V = \{1, 2, \ldots, n\}$$

Definizione 2.1.8. *Dato un nodo v_i, si definisce **grado** di v_i, e si indica con $d_g(v_i)$, il numero di nodi ad esso adiacenti.*

Ad esempio, nel grafo non orientato di Figura 2.1a il grado del nodo v_1 è $d_g(v_1) = 2$.

Esempio 2.1.1. Il grafo non orientato in figura 2.2 è rappresentato dalla coppia $G = (V, E)$, con $V = \{v_1, v_2, v_3, v_4, v_5\}$ e $E = \{(v_1, v_2), (v_1, v_3), (v_1, v_4), (v_2, v_4), (v_3, v_4), (v_4, v_5)\}$. La sua matrice di adiacenza è:

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Essendo un grafo non orientato tale matrice è simmetrica.

Definizione 2.1.9. *La **matrice laplaciana** L di un grafo G è la matrice $N \times N$ definita da $L = D - A$, dove A è la matrice di adiacenza, e D è una matrice diagonale di ordine N, che ha sulla diagonale principale il grado di ciascun nodo, e che risulta strutturata in questo modo:*

$$D = \begin{pmatrix} d_g(v_1) & 0 & \dots & 0 \\ 0 & d_g(v_2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & d_g(v_n) \end{pmatrix}$$

Nel caso di grafi non orienti la matrice laplaciana è simmetrica.

Esempio 2.1.2. Per il grafo di Figura 2.2, la cui matrice di adiacenza è stata calcolata nell'esempio 2.1.1,si ottiene la matrice laplaciana in questo modo:

$$D = \begin{pmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$L = D - A = \begin{pmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 3 & -1 & -1 & -1 & 0 \\ -1 & 2 & 0 & -1 & 0 \\ -1 & 0 & 2 & -1 & 0 \\ -1 & -1 & -1 & 4 & -1 \\ 0 & 0 & 0 & -1 & -1 \end{pmatrix}$$

Per costruzione la somma degli elementi di una riga della matrice laplaciana L di un grafo è pari a zero. Di conseguenza la matrice laplaciana avrà almeno un autovalore $\lambda = 0$ Indicando con $\mathbf{1}$ il vettore $\mathbf{1} = (1, ..., 1)^T$ si può dunque scrivere $L\mathbf{1} = 0$, vale a dire 1 è *autovettore destro* di L relativo all'*autovalore* zero. Si può dimostrare, mediante il teorema di Gershgorin, che tutti gli autovalori di L sono interni al disco chiuso nel piano complesso centrato in $\Delta + 0j$ e con raggio pari a Δ, dove $\Delta = max_i(d_g(v_i))$, cioè pari al massimo grado del grafo. Di conseguenza tutti gli autovalori della matrice laplaciana hanno parte reale non negativa. Per grafi non orientati, la matrice laplaciana, in quanto simmetrica, ha autovalori reali che possono essere così ordinati:

$$0 = \lambda_1 \leq \lambda_2 \leq \lambda_3 \leq \ldots \leq \lambda_N \leq 2\Delta$$

Se il grafo G è fortemente connesso l'autovalore zero è isolato, e $\lambda_2 > 0$. Il secondo autovalore λ_2 di L, chiamato *connettività algebrica* gioca un ruolo importante nella misura delle prestazioni degli algoritmi di coordinazione e consenso.

Un'altra matrice importante nella trattazione formale dei problemi di consenso è la matrice di *Perron*.

Definizione 2.1.10. *La **matrice di Perron** di parametro ϵ, $P(\epsilon)$, associata ad un grafo G, è definita come*

$$P = I - \epsilon L$$

dove L è la matrice laplaciana e $\epsilon \in (0, 1/\Delta]$ è uno scalare.

La matrice di Perron ha, quindi, tutti gli elementi non negativi.

Una matrice non negativa viene chiamata *stocastica* se la somma degli elementi delle sue righe vale uno, *doppiamente stocastica* se è stocastica e la somma degli elementi delle sue colonne è pari a uno. Una matrice stocastica è *primitiva* (o *ergodica*) se ha un solo autovalore di modulo massimo. La matrice di Perron P di parametro $\epsilon \in (0, 1/\Delta]$ di un grafo non orientato G con massimo grado Δ gode delle seguenti proprietà:

1. P è stocastica con autovalore uno associato all'autovettore $\mathbf{1}$;
2. tutti gli autovalori di P sono interni al cerchio unitario;
3. se G è bilanciato, allora P è doppiamente stocastica;
4. se G è un grafo non orientato, poiché L è simmetrica, allora lo sarà anche P, e per questo P è doppiamente stocastica;
5. se G è fortemente connesso e $0 < \epsilon < 1/\Delta$, allora P ha un solo autovalore pari ad 1 in quanto L ha un solo autovalore nullo. Quindi esiste un solo autovalore di P di modulo massimo e P è primitiva.

Un'importante proprietà delle matrici primitive ci è data dal seguente teorema:

Teorema 2.1.1 (Perron-Frobenius)**.** *Sia P una matrice primitiva con autovettori sinistro e destro w e v, rispettivamente, tali che*

$$Pv = v, w^T P = w^T, v^T w = 1$$

Allora $\lim_{k\to\infty} P^k = vw^T$.

Questo risultato sarà utile per dimostrare la convergenza di un algoritmo per il rendez-vous.

2.2 Analisi degli algoritmi di consenso

Il consenso è un problema fondamentale per le reti di agenti mobili che vogliono trovarsi d'accordo su un dato valore che dipende dallo stato di ogni singolo agente: punto del piano in cui incontrarsi, valore di temperatura dell'ambiente, direzione di moto da mantenere, etc. Per questo tipo di problemi è necessario definire un protocollo sulla rete che consenta agli agenti di raggiungere un consenso sul valore in considerazione. Un protocollo definisce le regole di interazione e di scambio di informazione tra un agente e gli agenti vicini. Un sistema multi agente interconnesso può essere rappresentato da un grafo $G = (V, E)$ in cui:

- i nodi rappresentano gli agenti;
- gli archi rappresentano i canali di comunicazione tra gli agenti.

Se il grafo è non orientato, un arco $(i, j) \in E$ rappresenta un canale di comunicazione bidirezionale tra gli agenti i e j, mentre se il grafo è orientato la comunicazione può avvenire da i verso j ma non da j verso i. In questa trattazione si assume che gli agenti siano puntiformi. In questo modo la probabilità di collisione tra due agenti e pressochè nulla, e questo semplifica notevolmente il problema. Vengono ora presentati alcuni tipici problemi di consenso.

2.2.1 Consenso per un *MAS* con rete tempo invariante

Agenti con dinamica a Tempo Continuo

Si considera una rete di agenti autonomi che si muovono nello spazio secondo relazioni lineari e tempo invarianti. Si ipotizza che possano ricevere ed elaborare le informazioni di cui necessitano istantaneamente, quindi l'evoluzione del sistema può venire descritta per mezzo di un modello a tempo continuo. Si ipotizza, inoltre, che la rete di comunicazione tra gli agenti sia simmetrica

e tempo-invariante; vale a dire che se esiste l'arco $(i, j) \in E$, allora esiste anche l'arco $(j, i) \in E$, e se l'arco (i, j) esiste al tempo t_0, allora esso esiste anche $\forall t > t_0$. La posizione nello spazio x_i dell'agente i-esimo varia nel tempo secondo l'equazione:

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

Il protocollo 2.1 può essere espresso in forma matriciale:

$$\dot{x}_i(t) = -\mathbf{L}x_i(t) \tag{2.2}$$

dove L è la matrice laplaciana del grafo che rappresenta la rete di agenti, quindi, come noto, $\mathbf{1}$ è l'autovettore relativo all'autovalore $\lambda = 0$. In caso di grafo connesso il sistema 2.2 è stabile, infatti la matrice $-\mathbf{L}$ ha tutti gli autovalori a parte reale negativa, tranne un'unico autovalore a parte reale nulla. Inoltre, essendo $-\mathbf{L}$ una matrice singolare, il sistema 2.2 ha un numero infinito di stati di equilibrio, cioè, a seconda della posizione iniziale degli agenti nello spazio, il sistema convergerà ad un diverso stato di equilibrio per $t \to \infty$.

Dato un qualsiasi stato iniziale $x(0)$, il sistema all'istante t si trovera nello stato:

$$x(t) = e^{-\mathbf{L}t}x(0)$$

Lo stato iniziale $x(0)$ può essere scomposto in questo modo:

$$x(0) = c\mathbf{1} + \omega$$

Quindi l'evoluzione del sistema può essere scritta come segue:

$$x(t) = e^{-\mathbf{L}t}x(0) = ce^{-\mathbf{L}t}\mathbf{1} + e^{-\mathbf{L}t}\omega$$

Poiché per la matrice $-\mathbf{L}$ l'autovalore relativo all'autovettore $\mathbf{1}$ è $\lambda = 0$, si ottiene che:

$$x(t) = e^{-\mathbf{L}t}x(0) = ce^{-\mathbf{L}t}\mathbf{1} + e^{-\mathbf{L}t}\omega = c\mathbf{1} + e^{-\mathbf{L}t}\omega$$

Quindi:

$$\lim_{t \to \infty} x(t) = c\mathbf{1} \tag{2.3}$$

L'equazione 2.3 mette in evidenza che tutti gli agenti convergono asintoticamente verso un punto, che rappresenta lo stato di equilibrio del sistema a partire dalle condizioni iniziali $x(0)$.

Il grafo che descrive le interazioni tra gli agenti è per ipotesi non orientato, di conseguenza $a_{ij} = a_{ji}$, e perciò la somma degli stati di tutti i nodi è una quantità invariante:

$$\sum_i \dot{x}_i = 0; \sum_i x_i = c\mathbf{1} \tag{2.4}$$

Applicando la 2.4 due volte, al tempo $t = 0$ e $t = 1$ si ottiene:

$$c = \frac{1}{N} \sum_i x_i(t)$$

dove N è il numero degli agenti presenti nel sistema.

In altre parole l'algoritmo 2.2 converge al baricentro del sistema, che rimane invariato nel tempo. L'implementazione dell'algoritmo 2.2 può essere quindi un possibile metodo di risoluzione per problemi di rendez-vous, oppure, mediante opportune modifiche, permette di far raggiungere a un plotone di agenti determinate formazioni desiderate. L'equazione 2.2 ci aiuta a comprendere inoltre il significato del secondo autovalore λ_2 di L, chiamato *connettività algebrica*. Esso è infatti un indice della velocità di convergenza dell'algoritmo: maggiore è questo valore, più velocemente si estingue il transitorio del sistema 2.1 e quindi più velocemente il protocollo 2.2 conduce alla convergenza.

La convergenza dell'algoritmo appena esposto non è limitata ai soli grafi non orientati. In [2] si dimostra che anche per un generico grafo orientato purchè bilanciato (i grafi non orientati sono un caso particolare di questa classe), il protocollo 2.2 garantisce la convergenza asintotica per ogni stato iniziale nel baricentro del sistema.

Agenti con dinamica a tempo discreto

Si può dare anche una versione dell'algoritmo 2.2 per un sistema di agenti con dinamica a tempo discreto:

$$x_i(k+1) = x_i(k) + t_s \sum_{j \in N_i} a_{ij}(x_j(t) - x_i(t)) \tag{2.5}$$

dove t_s è l'intervallo di campionamento che si suppone sufficientemente piccolo. La 2.5 si può esprimere in forma matriciale:

$$x(k+1) = \mathbf{P}x(k) \tag{2.6}$$

dove la matrice $P = I - t_s L$ è la matrice di Perron di parametro t_s del grafo. Per definizione il grafo che rappresenta il sistema è non orientato, quindi bilanciato. Inoltre è anche fortemente connesso. Ne segue che la matrice P, per valori

sufficientemente piccoli dell'intervallo di campionamento t_s, è primitiva e doppiamente stocastica. L'evoluzione di un sistema a tempo discreto, espresso nella forma di 2.6, a partire da uno stato iniziale $x(0)$, è data dall'equazione:

$$x(k) = P^k x(0)$$

e si ha convergenza se $\lim_{t\to\infty} P^t < \infty$. Il teorema di Perron-Frobenius ci dice che il limite esiste per le matrici primitive. Ne segue che si ha:

$$\lim_{t\to\infty} x(k) = v(w^T x(0))$$

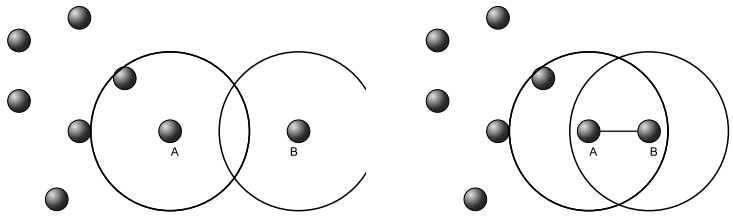
dove v e w sono gli autovettori destro e sinistro di P che soddisfano le ipotesi del teorema di Perron- Frobenius. Dato che P è doppiamente stocastica, ha un autovettore destro pari a $v = \mathbf{1}$ e uno sinistro $w = (1/N)\mathbf{1}$. Di conseguenza si ha la convergenza in:

$$\lim_{t\to\infty} x(k) = \frac{1}{N}\mathbf{1}^T x(0)$$

Vale a dire che anche con l'algoritmo 2.6 gli agenti convergono nel baricentro.

2.2.2 Consenso per un *MAS* con rete tempo variante

In alcuni sistemi multi agente interconnessi può capitare che la topologia della rete cambi continuamente. Si pensi a dei veicoli mobili che montano on-board dei sensori che immagazzinano, elaborano e trasmettono informazioni. È ragionevole supporre che il raggio percettivo dei sensori sia finito, perciò ogni veicolo percepisce solamente quei veicoli che si trovano all'interno di una determinata area di sensing, e riesce a comunicare solamente con loro, come è mostrato in Figura 2.3.



(a) A e B non comunicano (b) A e B comunicano

Figura 2.3: Comunicazione tra agenti in relazione all'area di sensing.

Inoltre, con i veicoli, anche i sensori sono in continuo movimento. Avvicinandosi ed allontanandosi uno dall'altro, i veicoli possono perdere alcune connessioni e formarne di nuove, quindi si ha a che fare con una rete di sensori tempo-variante. Sotto queste nuove condizioni il protocolli sopra esposti non riescono a garantire la convergenza degli agenti in un unico punto. È evidente come sia probabile la perdita di connessione nel grafo che rappresenta la rete di comunicazione, evento che provoca la separazione degli agenti in gruppi e quindi il ricongiungimento di questi in punti separati. Un'utile rappresentazione di un sistema con rete di comunicazione tempo-variante può essere data mediante l'utilizzo di *grafi dinamici*

Definizione 2.2.1. *Un **grafo dinamico** $G(t) = (V, E(t))$ è un grafo in cui l'insieme degli archi $E(t)$ e la matrice di adiacenza $A(t)$ sono tempo-varianti. Di conseguenza anche l'insieme dei vicini di ciascun agente $N_i(t)$ è tempo-variante.*

In base a come gli agenti si muovono nello spazio, avremo quindi, istante per istante, un diverso grafo che rappresenta la rete di comunicazione. Il protocollo 2.2 analizzato precedentemente, per un sistema di agenti con dinamica a tempo continuo, diventerà quindi:

$$\dot{x}(t) = -\mathbf{L}(G_i)x(t)\,; \quad i = s(t)\,,\, G_i \in \Gamma_n \tag{2.7}$$

Γ_n rappresenta l'insieme di tutti i possibili grafi di comunicazione che possono rappresentare la rete in ogni istante. Chiaramente Γ_n è un insieme finito, in quanto tutte le possibili tipologie di rete che possono rappresentare il sistema sono comprese tra il caso in cui non esiste comunicazione tra gli agenti ($E = \emptyset$) e il caso in cui ciascun agente comunica con tutti gli altri (grafo completo).

La funzione $s(t)$ rappresenta invece un segnale di *switch*, che istante per istante associa al sistema un valore di Γ_n che rappresenta la topologia della rete di comunicazione tra gli agenti in quell'istante. In [5] si dimostra che, se l'insieme Γ_n è composto solo da grafi fortemente connessi e bilanciati, per qualsiasi segnale di switch $s(t)$ e per qualsiasi condizione iniziale $x(0)$, il sistema 2.7 converge asintoticamente al baricentro delle posizioni iniziali degli agenti. In altre parole, se il sistema, nella sua evoluzione, rimane sempre fortemente connesso, allora il sistema converge. Si possono fare le stesse considerazioni per il caso di agenti con dinamica a tempo discreto. In questo caso il protocollo 2.6 diventa:

$$x(k+1) = P(G_i)x(k)\,; \quad i = s(k)\,,\, G_i \in \Gamma_n \tag{2.8}$$

2.2.3 Algoritmi di consenso randomizzati

In molti contesti applicativi, non è verosimile immaginare che gli agenti di una rete possano trasmettersi dati simultaneamente: questo a causa, ad esempio, della possibile collisione delle trasmissioni (ad esempio, scambio di informazioni mediante reti wireless) o dei limiti computazionali dei singoli agenti. Si ricorre quindi ad algoritmi dove solo poche trasmissioni alla volta vengono effettivamente effettuate. Ovviamente, per fare in modo che si arrivi al consenso, c'è la necessità di fare in modo che, prima o poi, tutti gli agenti del sistema vengano coinvolti nel processo di comunicazione. Tipicamente i processi random, per la selezione degli agenti che, in un dato istante, comunicano tra di loro, sono quelli che ottimizzano le prestazioni.

Due tipologie di algoritmi random sono stati introdotti nella letteratura: l'algoritmo di tipo *gossip*, dove ogni volta si creano, in modo random, una (o più) coppie di agenti che tra loro scambiano informazioni, e l'algoritmo di tipo *broadcasting*, dove invece, ad ogni istante, un singolo agente si attiva e invia le proprie informazioni a tutti i suoi vicini.

L'analisi teorica di questi algoritmi non è semplice: problemi ancora aperti, relativi a questi algoritmi, riguardano ad esempio,la stima della velocità di convergenza, o lo studio del discostamento del consenso sulla media causato dalla asimmetria di alcuni schemi (capita ad esempio nel 'broadcasting'). Attualmente in letteratura si trovano alcuni parziali risultati sulla convergenza quadratica media in condizioni particolari [8, 9, 12]. Riportiamo un esempio di algoritmo randomizzato, noto come *symmetric gossip model* [11] per un sistema in cui la topologia della rete di comunicazione è tempo-invariante.

Esempio di algoritmo di tipo *gossip*

Si consideri un sistema in cui:

- Esiste un canale di comunicazione per ogni possibile coppia di agenti, cioè tutti possono comunicare con tutti.

- Il processo di comunicazione tra gli agenti non avviene nello stesso tempo per tutti, ma per evitare la collisione di messaggi lo scambio di informazioni avviene in tempi diversi.

- Ad ogni istante di tempo si attiva un unico canale di comunicazione che collega due agenti. Il processo di selezione del canale di comunicazione è randomizzato.
- La probabilità che, in un dato istante, l'agente i comunichi con l'agente j è p_{ij}

Questa situazione può essere descritta da un grafo $G(t) = (V,E)$, che ad ogni istante risulta essere completo e non orientato. Per questo motivo la matrice di adiacenza risulta essere simmetrica. Ad ogni istante di tempo, si attiva un arco $(i,j) \in E$ con probabilita $p_{ij} \neq 0$. Dato che se ne accende uno solo alla volta, $\sum p_{ij} = 1$. Si suppone per semplicità che tutti gli archi abbiano la stessa probabilità di essere selezionati, quindi, poichè il numero degli archi è N^2, $p_{ij} = \frac{1}{N^2}$ Possiamo descrivere il sistema mediante l'equazione:

$$x(k+1) = \mathbf{W}_{ij}(k)x(k) \tag{2.9}$$

in cui $\mathbf{W}_k$, rappresenta la comunicazione tra due agenti all'istante k. Quando si attiva un arco (i,j) i nodi i e j si scambiano l'informazione riguardante il proprio stato, calcolando il nuovo stato come combinazione convessa:

$$\begin{aligned} x_i(k+1) &= (1-q)x_i(k) + qx_j(k), \quad 0 \leq q \leq 1 \\ x_j(k+1) &= (1-q)x_j(k) + qx_i(k), \quad 0 \leq q \leq 1 \end{aligned}$$

mentre tutti gli altri stati non ricevono informazione, e di conseguenza non cambiano il proprio stato:

$$x_m(k+1) = x_m(k); \quad m \neq i, m \neq j$$

$\mathbf{W}_{ij}(k)$ è casuale e dipende da come si accendono gli archi. Le attivazioni sono indipendenti, cioè ciò che succede all'istante $(k+1)$ è indipendente da cio che succede all'istante (k). Non si è certi della convergenza dell'algoritmo, ma in caso positivo questo converge alla media delle condizioni iniziali (average consensus). È facile verificare che il baricentro delle posizioni degli agenti è invariante, e ragionando come nel caso degli algoritmi presentati precedentemente si ottiene che:

$$\lim_{k\to\infty} x(k) = \frac{1}{N}\mathbf{1}^T x(0)$$

Poichè non si ha la certezza della convergenza dell'algoritmo (in quanto esiste la probabilità che alcuni archi non vengano mai attivati), si parla di *convergenza quasi certa*.

Capitolo 3

Descrizione del sistema simulato

Gli algoritmi presentati fino a questo momento sono tutte possibili soluzioni teoriche per problemi di consenso. Ipotizzando una reale implementazione di un sistema multi agente interconnesso su cui viene applicato un algoritmo per il controllo decentralizzato, bisogna fare i conti con una serie di problemi che fino a questo momento sono stati tralasciati. Ad esempio, fino a questo momento si è assunto che ogni agente fosse in grado di determinare la propria posizione, e quella degli altri, basandosi su un sistema di riferimento comune e noto a tutti. In questo modo ogni agente interpreta facilmente in maniera corretta le informazioni spaziali che riceve dagli altri.

Esistono numerosi casi in cui è possibile utilizzare un riferimento comune, sfruttando ad esempio sistemi di posizionamento GPS, oppure, come nel caso di alcuni sistemi satellitari, utilizzando come punti di riferimento determinate stelle fisse. In altri casi però, non è possibile conoscere a priori un sistema di riferimento comune, sia perchè le condizioni dell'ambiente in cui si trovano gli agenti non lo permettono, sia perché non sempre è possibile sostenere il costo delle tecnologie necessarie.

In questo capitolo verrà analizzato un sistema multi agente interconnesso in cui ogni agente vede l'ambiente che lo circonda secondo un proprio sistema di riferimento locale. L'obbiettivo è quello di fare in modo che gli agenti si dispongano nello spazio secondo una serie di formazioni desiderate, e per ottenere questo comportamento è necessario che:

- gli agenti si accordino su un sistema di riferimento comune (*frame* comune);

- ogni agente sia a conoscenza delle coordinate del suo punto di destinazione (posizione obiettivo).

Poiché ciascuno degli agenti ha un proprio sistema di riferimento locale, le coordinate della posizione obiettivo verranno interpretate secondo una stima locale del sistema di riferimento comune. Per questo motivo gli agenti si sposteranno verso la posizione giusta solamente dopo aver raggiunto un consenso sul frame comune. Per raggiungere questo obbiettivo verranno utilizzati due algoritmi:

- Un algoritmo randomizzato di tipo gossip per il consenso su un frame comune.
- Un algoritmo distribuito per spostare ogni agente nella posizione desiderata.

L'algoritmo di tipo gossip è descritto dettagliatamente in [1]. Inizialmente verranno descritti i due algoritmi separatamente, e successivamente si descriverà l'effetto della sovrapposizione di questi sul sistema.

3.1 Assunzioni preliminari

3.1.1 Comportamento degli agenti

Per ogni agente si fanno le assunzioni elencate qui sotto.

- Si muove in uno spazio bidimensionale.
- La sua dimensione si può considerare puntiforme. In questo modo la probabilità che due agenti si scontrino è pressochè nulla.
- Possiede un sistema di riferimento locale definito da una base ortonormale in $\mathbb{R}^2$.
- La comunicazione con gli altri agenti segue una temporizzazione locale, perciò la comunicazione nel sistema è *asincrona*. Quando 'scatta' il clock locale, viene selezionato dall'insieme dei vicini un agente con cui comunicare.
- Può determinare la distanza tra se e gli agenti con cui comunica.
- Può determinare la direzione in cui vede gli agenti con cui comunica secondo il proprio sistema di riferimento locale.

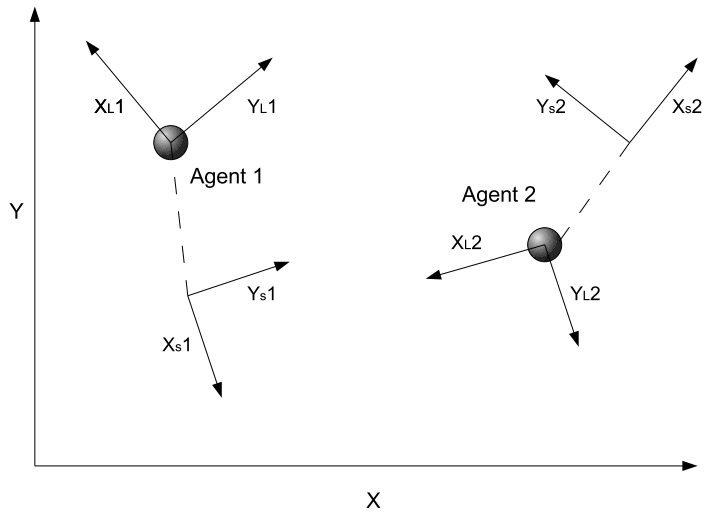


Figura 3.1: Rappresentazione dei diversi sistemi di riferimento

- Mantiene una stima di un sistema di riferimento comune, rispetto al proprio sistema di riferimento locale.

3.1.2 Sistemi di riferimento utilizzati

Per rappresentare il sistema è necessario fare riferimento ad un sistema di coordinate che permetta di descrivere la posizione di tutti gli agenti nello spazio in maniera univoca. Si utilizzeranno i seguenti sistemi di riferimento.

- **Sistema di riferimento globale:** Utilizzato per la rappresentazione del sistema. È unico per tutto il sistema.
- **Sistema di riferimento locale:** È il sistema di riferimento secondo cui ogni agente vede il mondo. Ogni agente ne possiede uno, ed è lui stesso il centro di tale sistema.
- **Sistema di riferimento stimato:** È il sistema di riferimento che ogni agente stima come comune per tutti. Ogni agente ne possiede uno e viene aggiornato continuamente.

In Figura 3.1 viene mostrato in maniera grafica il rapporto che esiste tra i diversi sistemi di riferimento utilizzati. In particolare:

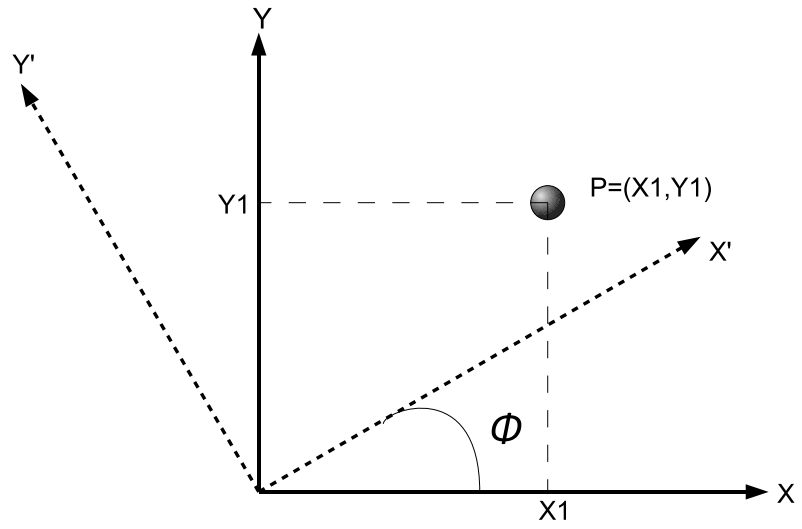


Figura 3.2: Passaggio da un sistema di riferimento ad un altro.

- X-Y è il sistema di riferimento globale.
- X_L1-Y_L1 e X_L2-Y_L2 sono i sistemi di riferimento locali rispettivamente di Agent1 e Agent2
- X_S1-Y_S1 e X_S2-Y_S2 sono i sistemi di riferimento stimati rispettivamente di Agent1 e Agent2

Il passaggio da un sistema di riferimento ad un altro avviene per mezzo delle matrici di rotazione R. Facendo riferimento alla figura 3.2, si supponga di voler esprimere le coordinate del punto P secondo il sistema di riferimento $X'Y'$. Se θ è l'angolo che c'è tra i due assi delle ordinate, la matrice di rotazione per passare dal sistema XY al sistema $X'Y'$ è:

$$R = \begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{pmatrix}$$

Il punto P avrà come nuove coordinate:

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{pmatrix} \begin{bmatrix} X \\ Y \end{bmatrix}$$

Nella trattazione che segue si utilizzeranno le seguenti matrici di rotazione:

- R_{gl} Per passare dal sistema di riferimento globale al sistema di riferimento locale.
- R_{ls} Per passare dal sistema di riferimento locale al sistema di riferimento stimato.

3.2 Algoritmo per il consenso su un frame comune

Il sistema viene descritto mediante un grafo tempo-invariante non orientato $G = \{V, E\}$ dove:

- $V = \{v_i : 1, \ldots, n\}$ è l'insieme dei nodi (agenti).
- $E = \{e_{ij} = (v_i, v_j)\}$ è l'insieme degli archi (connessioni), che rappresentano i canali di comunicazione disponibili tra gli agenti.

Ogni agente mantiene la stima di un punto, s_i, che considera il centro di un sistema di riferimento comune e che, rispetto al sistema di riferimento globale, può essere espressa in questo modo:

$$s_{gi} = R_{gl_i} s_i + p_i$$

dove p_i è la posizione dell'agente rispetto all'origine del sistema di riferimento globale.

L'obiettivo dell'algoritmo è quello di far convergere le stime di tutti gli agenti ad un valore comune. L'algoritmo applicato è di tipo *pairwise gossip*, in cui gli agenti si scambiano informazioni tra di loro a due a due. La comunicazione tra gli agenti è asincrona, e ogni volta che si verifica un *gossip* si attiva in maniera casuale un arco e avviene passaggio di informazione solo nel corrispondente canale di comunicazione.

L'algoritmo può essere definito come una tripla $\{S, IR, \mathbf{e}\}$ dove:

- $S = s_1, \ldots, s_n$ è l'insieme delle stime locali s_i di ogni agenti i del sistema.
- IR (*Interaction Rule*) è una regola di interazione locale che, dato un'arco e_{ij}, calcola per gli agenti i e j una nuova stima a partire dalla stima corrente: $R : (s_i, s_j) \Rightarrow (\hat{s}_i, \hat{s}_j)$.
- **e** è un processo di selezione degli archi che specifica quale arco $e_{ij} \in E$ è selezionato al tempo t.

I passi dell'algoritmo possono essere descritti in questo modo:

- **START**: $k = 0$, $s_i(0) = s_{i0}$ $\forall i = 1, \ldots, n$
- **STOP**: $s_i(k_{stop})$ $\forall i = 1, \ldots, n$
- **while** not stop_condition **do**

- $k = k + 1$
- **e** seleziona in maniera random $e_{ij} \in E$
- $(s_i(k+1), s_j(k+1)) = R(s_i(k), s_j(k))$

- **end**

La regola R che viene presentata viene applicata localmente da ogni agente per raggiungere con gli altri un consenso sulla stima. Data una coppia di nodi (i, j) per cui esiste un arco $e_{ij} \in E$, si può definire la direzione in cui l'agente i vede l'agente j, rispetto al suo sistema di riferimento locale, in questo modo:

$$\hat{c}_{ij} = R_{gl_i}^T \frac{(p_j - p_i)}{\|p_j - p_i\|}$$

dove $p_j, p_i \in \mathbb{R}^2$ sono le posizioni dei due agenti. Viene indicato con $\hat{c}_{ij}^{\perp}$ il versore perpendicolare a $\hat{c}_{ij}$. Vale la seguente proprietà:

$$R_{gl_i} \hat{c}_{ij} = -R_{gl_j} \hat{c}_{ji}$$

La distanza relativa tra due nodi è:

$$d_{ij} = d_{ji} = \|p_j - p_i\|_2$$

Definizione 3.2.1. *Regola di interazione locale* (R)

Quando l'arco (i, j) *viene selezionato dal processo di attivazione degli archi* **e**, *gli agenti* i *e* j *applicano la seguente regola di interazione locale* R *per aggiornare le proprie stime:*

$$\begin{aligned} s_i(k+1) &= \Delta(k) \cdot \hat{c}_{ij} + \Delta^{\perp}(k) \cdot \hat{c}_{ij}^{\perp} \\ s_j(k+1) &= \Delta(k) \cdot \hat{c}_{ji} + \Delta^{\perp}(k) \cdot \hat{c}_{ji}^{\perp} \end{aligned} \tag{3.1}$$

dove:

$$\begin{aligned} \Delta(k) &= \frac{d_{ij} - s_j(k)^T \hat{c}_{ji} + s_i(k)^T \hat{c}_{ij}}{2} \\ \Delta^{\perp}(k) &= \frac{s_i(k)^T \hat{c}_{ij}^{\perp} - s_j(k)^T \hat{c}_{ji}^{\perp}}{2} \end{aligned} \tag{3.2}$$

Si possono riassumere le caratteristiche e il comportamento dell'algoritmo in questo modo:

- La regola di interazione locale è decentralizzata.
- I parametri coinvolti nel processo di comunicazione sono locali. Si fa riferimento solo alla direzione in cui i due agenti si vedono e a quella ad essa perpendicolare.

- Ognuno dei due agenti selezionati determina la posizione dell'altro rispetto al proprio sistema di riferimento locale, la distanza relativa e il versore che indica la direzione della congiungente ($\hat{c}_{ij}$). Ognuno calcola il valore scalare della proiezione della propria stima nella direzione della congiungente ($s_i(k)^T \hat{c}_{ij}$) e nella direzione della perpendicolare alla congiungente ($s_i(k)^T \hat{c}_{ij}^{\perp}$). I due agenti si scambiano queste informazioni.
- I due agenti aggiornano la propria stima s_i facendo una media tra i valori delle proiezioni calcolati localmente e quelli ricevuti. La nuova stima, per entrambi gli agenti coinvolti, corrisponderà al punto medio del segmento che congiunge le due stime locali.

In figura 3.3 viene mostrato come avviene l'aggiornamento della stima tra due agenti coinvolti in un gossip. In [1] si dimostra che, se il grafo è connesso per tutto il tempo in cui viene applicato l'algoritmo, la stima di tutti gli agenti converge ad un unico valore che è il baricentro delle stime iniziali.

$$\forall i \quad \lim_{k\to\infty} R_{gl_i} s_i(k) + p_i = \frac{1}{n}\sum_{i=1}^{n} R_{gl_i} s_i(0) + p_i$$

Poichè la rete di connessione è, per ipotesi, tempo-invariante, è sufficiente che il grafo sia connesso per $k = 0$.

L'algoritmo 3.1 può essere espresso anche facendo riferimento al sistema di riferimento globale:

$$\mathbf{s}(k+1) = \mathbf{W}(\mathbf{e}(k))\mathbf{s}(k) \tag{3.3}$$

dove:

- $\mathbf{s} = [s_{g1}, s_{g2}, \ldots, s_{gn}]^T$ è il vettore delle stime locali espresse in coordinate globali.
- $\mathbf{W}(\mathbf{e}(k))$ è la matrice che rappresenta l'aggiornamento delle stime del sistema al passo k.

Ad ogni passo computazionale il processo $\mathbf{e}(k)$ seleziona un arco $(i, j) \in E$. A questo arco corrisponde una matrice $\mathbf{W}(e_{ij})$ costruita in questo modo:

$$\mathbf{W}(e_{ij}) = \mathbf{I} - \frac{(\hat{e}_i - \hat{e}_j)(\hat{e}_i - \hat{e}_j)^T}{2} \tag{3.4}$$

dove $\hat{e}_i = [0 \ldots 0 \underbrace{1}_{i} 0 \ldots 0]$ è un vettore $n \times 1$ con tutte le componenti uguali a 0 tranne la posizione i-esima uguale a 1. Esiste dunque una possibile matrice $\mathbf{W}(e_{ij})$ associata ad ogni arco $(i, j) \in E$, e ciascuna di queste matrici è formata in questo modo:

- Ogni riga $r \neq i, j$ è formata dal vettore $\hat{e}_r = [0 \dots 0 \underbrace{1}_{r} 0 \dots 0]$
- Le righe i e j sono formate dal vettore $\hat{e}_i = [0 \dots 0 \underbrace{\frac{1}{2}}_{i} 0 \dots 0 \underbrace{\frac{1}{2}}_{j} 0 \dots 0]$

L'evoluzione del sistema al al tempo k è data da:

$$\mathbf{s}(k) = \prod_{t=1}^{k} \mathbf{W}(e_{ij}(t))\mathbf{s}(0)$$

Ogni matrice $\mathbf{W}(e_{ij})$ ha le seguenti proprietà:

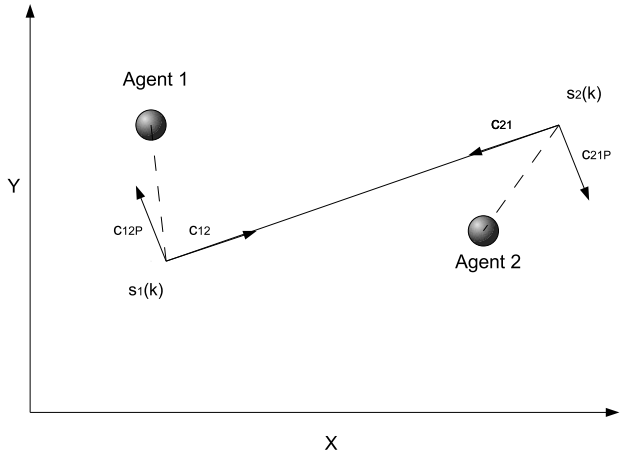
- È doppiamente stocastica.
- Dal prodotto di matrici $\mathbf{W}(e_{ij})$, si ottiene una matrice che è, a sua volta, doppiamente stocastica.

Queste proprietà garantiscono la stabilità dell'algoritmo, ma non ne garantiscono la convergenza, in quanto il processo di attivazione degli archi, puramente casuale, non garantisce che vengano selezionati tutti gli archi. Basta che un solo arco non venga mai selezionato e gli agenti non raggiungeranno mai un consenso sulla stima. Per la convergenza è necessario che il grafo di connessione sia connesso per $k = 0$ poiché la rete è, per ipotesi, tempo-invariante. Questa condizione non è sufficiente per la convergenza, ma se è rispettata e se il sistema converge, il punto di convergenza è il baricentro delle stime. Se la stima iniziale di ogni agente corrisponde alla propria posizione ($\mathbf{s}(0) = \mathbf{x}(0)$) , la stima converge al baricentro delle posizioni.

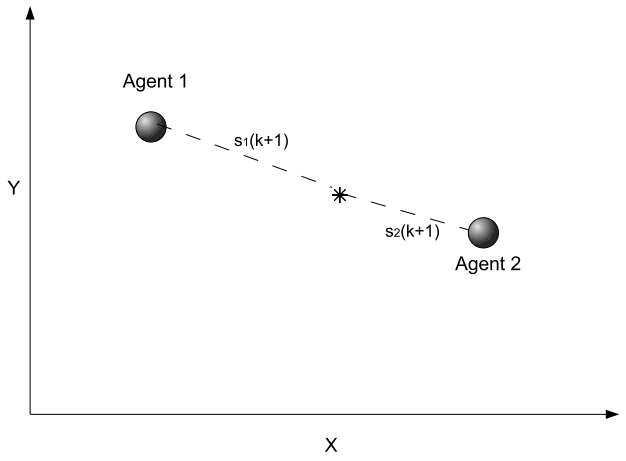
$$\lim_{k \to \infty} \mathbf{s}(k) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i(0)$$

Il problema del tempo di convergenza degli algoritmi randomizzati di tipo gossip è attualmente ancora aperto, e per la sua complessità non verrà trattato in questa tesi. Per determinare un sistema di assi comuni è sufficiente fare consenso su due punti. Un'asse avrà la direzione del segmento che unisce questi due punti, mentre l'altro avrà la direzione perpendicolare a tale segmento. Ad esempio, se gli agenti applicano su due diverse stime contemporaneamente l'algoritmo che è stato descritto, fanno consenso su due diversi punti:

- P_1 che sarà il centro del sistema di assi stimato.
- P_2 che servirà per determinare la direzione di uno degli assi.



(a)



(b)

Figura 3.3: Esempio di gossip.

3.3 Algoritmo per il raggiungimento della posizione obiettivo

Nella descrizione di questo algoritmo si suppone che gli agenti abbiano già raggiunto il consenso su un frame comune. L'obiettivo è quello di far convergere ogni agente ad una diversa posizione desiderata, rispetto al sistema di assi su cui si sono accordati. Ogni agente conosce la distanza che deve percorrere e la direzione verso cui si deve muovere, perciò è sufficiente che si sposti lungo tale direzione. Viene applicata su ogni agente una regola di controllo locale che determina, la posizione che verrà assunta allo step successivo in questo modo:

$$x_i(k+1) = (1-q_i)x_i(k) + (q_i)x_{id} \tag{3.5}$$

dove:

- $x_i(k)$ è la posizione corrente dell'agente i-esimo.
- x_{id} è la posizione obiettivo, espressa in coordinate globali, dell'agente i-esimo.
- $q_i \in [0,1)$ è un parametro per la regolazione della velocità.

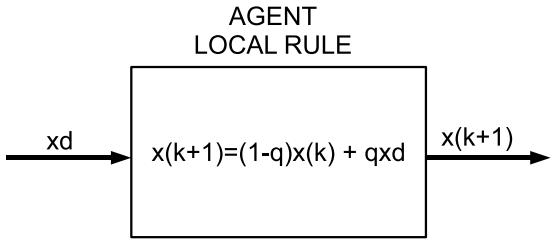


Figura 3.4: Regola locale per lo spostamento

È evidente che ogni agente possiede già tutte le informazioni necessarie per calcolare la nuova posizione, perciò può essere considerato come un sistema indipendente a cui viene applicato un ingresso costante, come mostrato in figura 3.4.

Poichè q_i è minore di 1, la posizione dell'agente converge alla posizione desiderata La dinamica del sistema può essere espressa in questo modo:

$$\mathbf{x}(k+1) = (\mathbf{I} - \mathbf{I}q)\mathbf{x}(k) + (\mathbf{I}q)\mathrm{x_d} \tag{3.6}$$

La dinamica del sistema, a tempo discreto, è espressa mediante una rappresentazione in *variabili di stato*, in cui ogni variabile x_i rappresenta la posizione dell'agente i-esimo. Il sistema è stabile in quanto gli autovalori della matrice $(\mathbf{I} - \mathbf{I}q)$ sono tutti all'interno del cerchio di raggio unitario. Ogni agente converge alla propria posizione obiettivo indipendentemente dagli altri, e la velocità di convergenza del sistema dipende dall'autovalore più grande (in modulo) della matrice $(\mathbf{I} - \mathbf{I}q)$. Infatti, l'autovalore corrispondente all'agente i-esimo è $1 - q_i$, e il modo associato ad esso può considerarsi estinto in un numero di passi computazionali circa pari a:

$$k = -\frac{5}{\lg(1-q)}$$

Questa regola di controllo prevede che tutti gli agenti si muovano con una velocità regolata da un parametro q_i costante. Nelle simulazioni del Capitolo 5 verranno utilizzati altri due algoritmi, derivati da questo, in cui i valori di q_i variano nel tempo secondo delle regole adattative.

3.4 Sovrapposizione dei due algoritmi

L'obiettivo è quello di valutare il comportamento del sistema quando i due algoritmi vengono applicati contemporaneamente sugli agenti. Ogni agente si muove nello spazio per raggiungere quella che per lui è la posizione obiettivo, mentre comunica con gli altri per raggiungere un frame comune. Si ipotizza un funzionamento di questo tipo:

- Il tempo medio che intercorre tra due gossip è t_g.
- Ogni agente aggiorna la sua posizione con una frequenza $f_s = \frac{1}{t_s}$, dove t_s è il tempo di campionamento dell'algoritmo 3.5.

Risulta subito evidente che vengono meno alcune delle assunzioni che sono state fatte per i due algoritmi.

Per quanto riguarda l'algoritmo di tipo gossip descritto dall'equazione 3.4 viene meno l'ipotesi di rete tempo-invariante, infatti gli agenti non sono più fermi, ma si muovono nello spazio, e per questo:

- La rete di comunicazione non può più essere descritta da un grafo tempo-invariante, ma verrà descritta da un grafo dinamico $G(t) = (V, E(t))$, in cui l'insieme degli archi varia nel tempo e dipende dalle posizioni che gli agenti assumono nello spazio.
- Condizione necessaria per la convergenza delle stime è che il grafo $G(t) = (V, E(t))$ risulti connesso per $k = 0$.
- Se il sistema converge, non è detto che il punto di convergenza sia il baricentro delle stime iniziali, in quanto gli agenti sono in movimento, e di conseguenza si sposta anche la stima che è relativa al sistema di riferimento locale, come è mostrato in figura 3.5. Se le stime convergono, il consenso avverrà per un punto in $\mathbf{span}\{\mathbf{1_n}\}$ non determinabile a priori.

Per quanto riguarda l'algoritmo che regola lo spostamento degli agenti verso la posizione obiettivo, descritto dall'equazione 3.6, viene meno l'ipotesi di un frame comune già determinato, e pertanto:

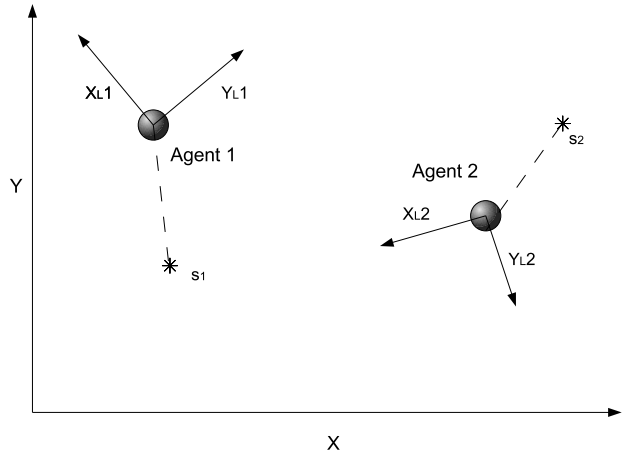
- Ogni agente ha una stima propria del frame comune e non è detto che sia la stessa degli altri.

- Ogni agente si sposta verso una posizione obiettivo riferita alla propria stima del frame comune, e per questo motivo non è detto che il plotone raggiunga la formazione desiderata.
- La stima locale del frame comune è tempo-variante, perciò cambia continuamente anche la posizione obiettivo. Pertanto non si può determinare con certezza il tempo di convergenza, e non si ha nemmeno la certezza che il sistema converga.

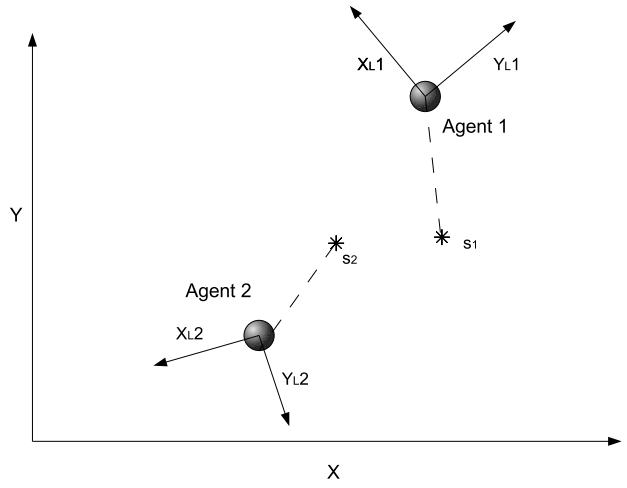
Il comportamento di questo tipo di sistemi è stato studiato grazie all'ausilio di un simulatore sviluppato con MATLAB e descritto nel Capitolo 4, che è stato utilizzato per:

- Verificare le proprietà dei due algoritmi separati.
- Valutare il comportamento del sistema quando i due algoritmi vengono applicati contemporaneamente.
- Valutare eventuali parametri che influenzano il comportamento del sistema.
- Valutare sotto quali condizioni il sistema si comporta nel modo desiderato e, nei casi in cui è possibile, valutarne il tempo di convergenza.

I risultati delle simulazioni sono esposti nel Capitolo 5



(a)



(b)

Figura 3.5: Spostamento della stima locale dovuto allo spostamento degli agenti.

Capitolo 4

Descrizione del simulatore

In questo capitolo viene descritto il simulatore, realizzato in MATLAB, che implementa il sistema multi agente descritto nel Capitolo 3. Si è scelto di realizzare il simulatore utilizzando una *programmazione orientata agli oggetti*, in modo da rendere il codice il più possibile riutilizzabile e adattabile alla simulazione di sistemi multi-agente con caratteristiche diverse. In questa stessa ottica, si è cercato di costruire il simulatore nel modo più modulare possibile. Nella descrizione delle varie parti del simulatore, in relazione ai diversi sistemi di riferimento utilizzati, si parlerà di:

- *coordinate globali*, relative al sistema di assi che vengono utilizzati per la rappresentazione grafica del sistema;
- *coordinate locali*, relative al sistema di riferimento locale di ciascun agente;
- *coordinate stimate*, relative alla stima del riferimento comune che ciascun agente possiede.

4.1 La classe *Agent*

Il centro del simulatore è costituito dalla classe *Agent*, che implementa tutte le funzionalità del singolo agente. Ogni istanza della classe Agent rappresenta un sistema autonomo, con dinamica propria e con la possibilità di interagire con l'ambiente in cui si trova. Un oggetto di tipo Agent è caratterizzato da:

- Una serie di *attributi*, che definiscono lo stato interno di ciascun agente.

- Una serie di *metodi*, che definiscono le modalità di interazione con l'ambiente esterno, e quindi anche le modalità di comunicazione con gli altri agenti.

4.1.1 Attributi della classe Agent

Lo stato interno di ogni agente è definito di seguenti attributi:

- **id**: rappresenta un identificatore numerico dell'agente implementato. In alcune simulazioni è utile identificare ciascun agente con il suo numero.
- **ts**: attributo che tiene in memoria il tempo di campionamento dell'agente istanziato.
- **initstate**: vettore bidimensionale che memorizza la posizione iniziale dell'agente istanziato in coordinate globali.
- **state**: vettore bidimensionale che memorizza la posizione corrente dell'agente istanziato in coordinate globali.
- **evolution**: matrice a 2 colonne che tiene memoria di tutte le posizione occupate dall'agente a partite dall'istante iniziale $t(0)$. La riga i-esima corrisponde all'i-esimo istante di campionamento

Gli attributi relativi alla posizione fisica dell'agente nello spazio sono stati espressi per comodità in coordinate globali, in modo da renderne più veloce la rappresentazione grafica. In realtà ciascun agente è non è a conoscenza della propria posizione rispetto al sistema di riferimento globale.

Altri attributi sono:

- **rotationMatrix**: matrice di rotazione 2×2 per passare dalle coordinate locali alle coordinate globali.
- **currentEstimate**: vettore bidimensionale che rappresenta la stima corrente del centro $P1$ del sistema di riferimento comune, espressa in coordinate locali.
- **currentEstimate2**: vettore bidimensionale che rappresenta la stima corrente di un punto $P2$ situato sull'asse delle ordinate del sistema di riferimento comune, espressa in coordinate locali. Il vettore $\mathbf{P}2 - \mathbf{P}1$ permette di determinare una direzione (che va da $P1$ a $P2$) che rappresenta la stima locale dell'asse delle ordinate del frame comune.

- **localEstimateEvoluion**: matrice a 2 colonne che tiene memoria di tutte le stime del centro del frame comune (punto $P1$) a partite dall'istante iniziale $t(0)$. La riga i-esima corrisponde all'i-esimo istante di campionamento.
- **localEstimateEvoluion2**: matrice a 2 colonne che tiene memoria di tutte le stime del punto $P2$, situato sull'asse delle ordinate del sistema di riferimento comune, a partite dall'istante iniziale $t(0)$. La riga i-esima corrisponde all'i-esimo istante di campionamento.
- **comDir**: Versore stimato dell'asse delle ordinate del sistema di riferimento comune.
- **destination**: vettore bidimensionale che rappresenta la posizione obiettivo per l'agente, espressa in coordinate stimate.
- **q**: parametro che determina la velocità di convergenza dell'agente alla posizione obiettivo. Il valore di q deve essere compreso tra 0 e $qmax$.
- **qmax**: valore massimo raggiungibile dal parametro q. Utilizzata negli algoritmi adattativi.
- **count**: variabile che conta i cicli di aggiornamento della posizione dell'agente. Utilizzata negli algoritmi adattativi.

4.1.2 Metodi della classe Agent

Un oggetto di tipo Agent modifica il suo stato interno utilizzando sia le informazioni di cui è in possesso, sia le informazioni che riceve dall'esterno. I metodi implementano le regole che l'agente utilizza per modificare il proprio stato.

Metodo *Agent*

Nella programmazione a oggetti il *costruttore* è il metodo utilizzato per istanziare oggetti di una determinata classe, e solitamente questo metodo prende lo stesso nome della classe a cui appartiene. Gli ingressi di questa funzione vengono utilizzati per inizializzare gli attributi dell'oggetto. L'interfaccia del metodo *Agent* è:

```
function obj=Agent(id,x1,x2,t,s1,s2,d1,d2,o1,o2,a,q)
```

Gli attributi dell'oggetto *obj* vengono inizializzati in questo modo:

```
obj.id=id;
obj.initstate=[x1,x2];
obj.state=[x1,x2];
obj.ts=t;
obj.evolution=obj.initstate(1,1:2);
obj.currentEstimate=[s1,s2];
obj.localEstimateEvolution=[s1,s2];
obj.currentEstimate2=[d1,d2];
obj.localEstimateEvolution2=[d1,d2];
obj.destination=[o1,o2];
obj.q=q;
```

Il parametro di ingresso a serve per l'inizializzazione della matrice di rotazione memorizzata nell'attributo **rotationMatrix**. Il valore di a è quello di $\cos\phi$, dove ϕ è l'angolo tra l'asse delle ordinate del sistema di riferimento globale e l'asse delle ordinate del sistema di riferimento locale. Dal valore di a si ricava il valore $b = \sin\phi$. I valori di a e b sono sufficienti per determinare la matrice di rotazione.

```
b=sqrt(1-(a^2));
obj.rotationMatrix=[a,-b;b,a];
obj.comDir=([d1,d2]-[s1,s2])/norm([d1,d2]-[s1,s2]);
```

Esempio 4.1.1. *Il comando:*

```
A=Agent(1,3,2,0.05,2.5,1.5,4,5,0,0,0.5,0.001)
```

crea un oggetto di tipo Agent e produce il seguente output:

```
A =
  Agent
  Properties:

                      id: 1
                      ts: 0.0500
               initstate: [3 2]
          rotationMatrix: [0.5000,-0.8660;0.8660,0.5000]
                   state: [3 2]
```

```
                evolution: [3 2]
     localEstimateEvolution: [2.5000 1.5000]
            currentEstimate: [2.5000 1.5000]
    localEstimateEvolution2: [4 5]
           currentEstimate2: [4 5]
                     comDir: [0.3939 0.9191]
                destination: [0 0]
                          q: 0.0010
```

Il metodo *EvolutionStep*

Il metodo *EvolutionStep* implementa il passo computazionale dell'algoritmo per il raggiungimento della posizione obiettivo. Il metodo non ha ingressi, oltre l'oggetto stesso che lo invoca, e utilizza esclusivamente le informazioni locali per determinare la posizione all'istante successivo. L'interfaccia di questo metodo è:

```
function obj=EvolutionStep(obj,q)
```

Il metodo *EstimateStep*

Il metodo *EstimateStep* implementa il passo computazionale dell'algoritmo di gossip per il consenso sul frame comune. L'agente su cui viene invocato il metodo ($Agent_i$) comunica con un unico agente ($Agent_j$) con cui avviene uno scambio di informazioni. Il metodo calcola le stime successive per i punti $P1$ (stima del centro del sistema di riferimento comune) e $P2$ (stima di un punto generico sull'asse delle ordinate del sistema di riferimento comune). L'interfaccia di questo metodo è:

```
function obj=
  EstimateStep(obj,sj1,sj2,dj1,dj2,d,cijx,cijy,cjix,cjiy)
```

I parametri in ingresso sono:

- **s1,s2**: Dati ricevuti dall'agente ($Agent_j$)con cui avviene il gossip. È la stima del punto $P1$ di $Agent_j$ espressa nelle sue coordinate locali.
- **d1,d2**: Dati ricevuti dall'agente ($Agent_j$)con cui avviene il gossip. È la stima del punto $P2$ di $Agent_j$ espressa nelle sue coordinate locali.
- **d**: Distanza tra i due agenti tra cui avviene il gossip.

- **cijx,cijy**: Versore parallelo al segmento che congiunge i due agenti che comunicano, diretto da $Agent_i$ a $Agent_j$, espresso nelle coordinate locali di $Agent_i$.
- **cjix,cjiy**: Dati ricevuti dall'agente ($Agent_j$) con cui avviene il gossip. Versore parallelo al segmento che congiunge i due agenti che comunicano, diretto da $Agent_j$ a $Agent_i$, espresso nelle coordinate locali di $Agent_j$.

4.2 Funzioni per l'analisi e il controllo del sistema

Per simulare un *MAS*, in cui ogni agente è rappresentato da un'istanza della classe *Agent*, sono state sviluppate una serie di funzioni esterne che operano su *Array di oggetti di tipo Agent*. Anche in questo caso si è cercato di sviluppare un codice riutilizzabile. In questa sezione vengono riassunte le funzioni più importanti, partendo da quelle utili per l'analisi di un generico sistema multi-agente, fino a quelle specifiche per il sistema che si è studiato in questo progetto.

function d = CalculateDistance(obj1,obj2) Calcola la distanza tra i due oggetti di tipo *Agent* passati in ingresso

function d = DistanceMatrix(Array) Dato in ingresso un Array di N di oggetti di tipo *Agent*, restituisce in uscita una matrice $N \times N$ in cui, ad ogni posizione (i, j) corrisponde la distanza tra l'agente i e l'agente j.

function a=AdiacenceMatrix(array,radius) Dato in ingresso un Array di N di oggetti di tipo *Agent*, e un numero reale positivo (radius), restituisce in uscita la matrice di adiacenza che rappresenta il grafo di comunicazione tra gli agenti. I valori della matrice di adiacenza sono:

$$a_{ij} = \begin{cases} 1 & \text{se CalculateDistance(i,j)} \leq \text{radius} \\ 0 & \text{se CalculateDistance(i,j)} > \text{radius} \end{cases}$$

In altri termini, un agente vede solo quegli agenti che si trovano all'interno di una sfera di sensing di raggio r =*radius*, e può comunicare solamente con questi. In base al valore di *radius* sarà diversa la topologia della rete.

function xy=PositionMatrix(array) Dato in ingresso un Array di N di oggetti di tipo *Agent*, restituisce una matrice a $N \times 2$ in cui, alla riga i-esima corrisponde la posizione corrente dell'agente i-esimo espressa in coordinate globali.

function xy=EstimateMatrix(array) Dato in ingresso un Array di N di oggetti di tipo *Agent*, restituisce una matrice a $N \times 2$ in cui, alla riga i-esima corrisponde la stima del centro del riferimento comune $P1$ dell'agente i-esimo espressa in coordinate globali.

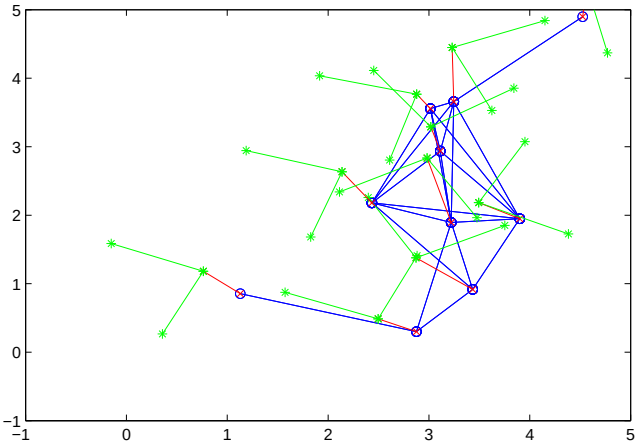
function Array=PlotArray(Array,radius) È una delle funzioni più utili durante le simulazioni perchè gestisce la rappresentazione grafica del sistema. Prende in ingresso un Array di N di oggetti di tipo *Agent*, e un numero reale positivo (radius). L'uscita della funzione è una rappresentazione grafica del sistema, in cui:

- Gli agenti sono rappresentati da dei cerchi blu tagliati da una 'x' rossa.
- Gli archi del grafo di comunicazione sono rappresentati da delle linee blu.
- Per ogni agente è rappresentata con una linea rossa la stima del riferimento comune.
- Per ogni agente vengono rappresentati in verde gli assi del sistema di riferimento comune stimato.

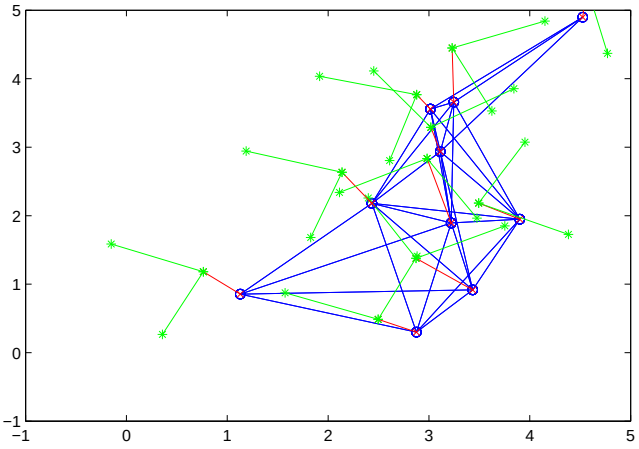
In Figura 4.1 viene riportato l'output della funzione PlotArray applicata ad un array di 10 oggetti di tipo Agent. Si può vedere come la topologia del grafo cambi al variare del raggio di sensing *radius*.

function Array=ArrayEvolutionStep(Array,radius) È la funzione che controlla l'evoluzione globale del sistema gestendo la sovrapposizione dei due algoritmi che vengono applicati. Viene evocata ad ogni passo computazionale per eseguire una serie di operazioni che simulano, agendo dall'esterno, lo scambio di informazioni tra gli agenti e l'aggiornamento dello stato di ognuno di loro,. Ogni volta che si verifica un gossip gestisce lo scambio di informazioni tra i due oggetti di tipo Agent coinvolti, seguendo questa procedura:

- Seleziona in maniera completamente casuale un'arco del grafo di comunicazione. I nodi collegati dall'arco selezionato rappresentano gli agenti coinvolti nel gossip.
- Estrae dagli oggetti selezionati le informazioni che riguardano lo stato dei due agenti (posizione, stima del punto $P1$, stima del unto $P2$).

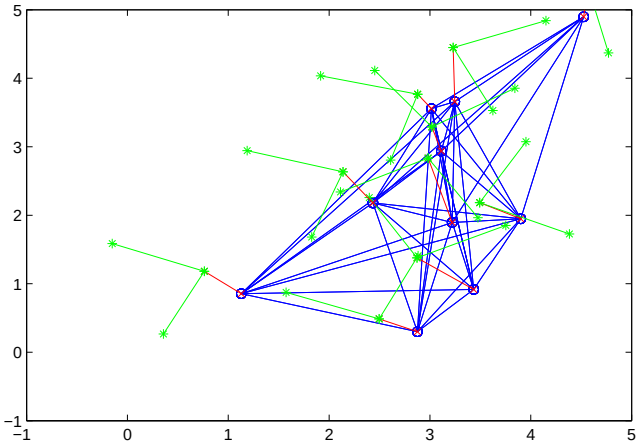


(a) radius=1.85



(b) radius=2.5

Figura 4.1: Applicazione della funzione *PlotArray* ad un Array di 10 oggetti di tipo Agent - 1

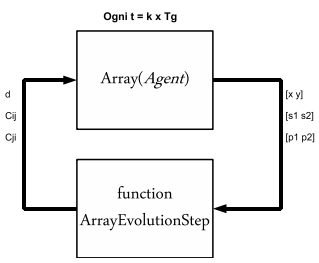


(c) radius=4

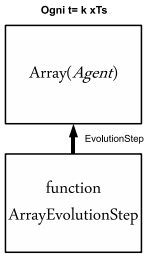
Figura 4.1: Applicazione della funzione *PlotArray* ad un array di 10 oggetti di tipo Agent - 2

- Elabora le informazioni estratte calcolando la distanza tra i due agenti e, per ciascuno di loro, determina i versori paralleli e perpendicolari al segmento che li congiunge.
- Invoca sui due agenti il metodo *EstimateEvolutionStep*, passando in ingresso i giusti parametri.

Inoltre, con una frequenza $1/t_s$, evoca su ogni agente il metodo *EvolutionStep*. In Figura 4.2 è schematizzato il comportamento di questa funzione.



(a) Gestione del gossip



(b) Gestione dello Spostamento

Figura 4.2: Schematizzazione dell'azione di supervisione della funzione *ArrayEvolutionStep*

La funzione *ArrayEvolutionStep* si comporta in maniera diversa a seconda del tipo di situazione che si vuole simulare.

- Se $t_g = t_s$, ogni volta che la funzione viene invocata seleziona una sola coppia di agenti, invoca su di loro la funzione *EstimateEvolutionStep* e successivamente invoca su tutti gli oggetti dell'array di Agent la funzione *EvolutionStep*.

- Se $t_g = N t_s$, ogni volta che la funzione viene invocata seleziona una sola coppia di agenti, invoca su di loro la funzione *EstimateEvolutionStep* e successivamente invoca su tutti gli oggetti dell'Array di Agent, per N volte, la funzione *EvolutionStep*.

- Se $t_g = (1/N)\, t_s$, ogni volta che la funzione viene invocata seleziona per N volte una coppia di agenti, invoca su di loro la funzione *EstimateEvolutionStep* e successivamente invoca su tutti gli oggetti dell'array di Agent la funzione *EvolutionStep*.

4.3 Rappresentazione animate in 3D del sistema

4.3.1 Il VIRTUAL REALITY TOOLBOX

Il *VIRTUAL REALITY TOOLBOX* permette di visualizzare e di interagire con le simulazioni di sistemi dinamici in un ambiente di realtà virtuale a 3 dimensioni. Il toolbox collega MATLAB e Simulink con un ambiente di rappresentazione di realtà virtuale, abilitandoli a controllare la posizione, la rotazione e le dimensioni delle immagini 3-D definite in questo ambiente. Il risultato è una animazione in 3-D, attraverso la quale è possibile visualizzare il comportamento dei sistemi simulati in una modalità che può essere anche abbastanza vicina alla realtà.

Per la creazione e la gestione dei mondi virtuali, il tool utilizza lo standard *VRML*, un linguaggio molto diffuso e utilizzato, che permette di descrivere realtà virtuali e quindi di creare ed esplorare mondi tridimensionali.

Il *VIRTUAL REALITY TOOLBOX* permette di:

- Collegare i segnali MATLAB-Simulink al mondo virtuale per controllarne alcune proprieta, come per esempio il movimento degli oggetti.

- Realizzare mondi virtuali mediante un apposito editor VRML.

- Produrre registrazioni video delle simulazioni virtuali in diversi formati.

- Interagire con Simulink per la visualizzazione di simulazioni in tempo reale.

Il tool è stato progettato principalmente per interfacciarsi con l'ambiente Simulink, ma esistono alcune funzioni che permettono di far interagire gli oggetti virtuali con il Workspace di MATLAB. Una parte del simulatore realizzato consiste proprio in una serie di funzioni 'ad hoc', che permettono di associare lo stato e l'evoluzione degli oggetti di tipo Agent alla posizione e ai movimenti di determinati oggetti nello spazio virtuale.

Il linguaggio VRLM

VRML è l'acronimo di *Virtual Reality Modeling Language*. È un linguaggio di programmazione che consente la simulazione di mondi virtuali tridimensionali. Si tratta, come nel caso dell'HTML, di un linguaggio di markup in formato ASCII. Esso è utilizzabile dai comuni browser, grazie a dei plug-in specifici. I file in VRML, che tipicamente hanno estensione **.wrl**, sono semplici file ASCII che contengono i comandi e le istruzioni necessari a descrivere ambienti in 3D. I comandi inseriti nel file permettono la descrizione di oggetti e strutture in termini di geometria, rendering, illuminazione, colore, inoltre grazie all'utilizzo di textures l'aspetto degli ambienti può essere reso il più realistico possibile, permettendo così visite virtuali in soggettiva grazie ad una interfaccia grafica semplice ma potente. Mediante il VRML è possibile, cioè, descrivere ambienti virtuali contenenti oggetti, sorgenti luminose, immagini, suoni, filmati. Questi mondi, inoltre, possono essere animati e presentare caratteristiche anche complesse di interattività. Un mondo virtuale viene memorizzato in file con estensione **.wrl**. Una volta creato il mondo, con tutti gli oggetti all'interno, mediante dei *Players VRML* è possibile spostarsi virtualmente all'interno di questo mondo e visualizzarlo da diversi punti di vista.

Negli anni il VRML è diventato un vero e proprio standard per la descrizione di scene 3D, al punto che, attualmente, un file VRML pubblicato su Internet è accessibile da qualsiasi macchina indipendentemente dalla piattaforma, nello stesso modo in cui i documenti HTML possono essere visualizzati sostanzialmente allo stesso modo su macchine Windows, Mac, Unix, Linux, ecc.

Gli oggetti, nel modo virtuale, sono descritti da un blocchi di testo che ne specificano forma, posizione, rotazione, colore e aspetto. Ad esempio, un possibile blocco per istanziare una semplice sfera, che no definisce esclusivamente colore e dimensione, è il seguente:

```
#VRML V2.0 utf8
# sfera VRML
Shape {
appearance Appearance
material Material { emissiveColor 1 0 0 }
}
geometry Sphere { radius 1 }
}
```

Il *VIRTUAL REALITY TOOLBOX* mette a disposizione un editor di file .wrl, che permette di costruire un mondo virtuale mediante l'ausilio di un'interfaccia grafica, che genera il codice VRML corrispondente agli oggetti che l'utente inserisce nello spazio di lavoro.

4.3.2 Funzioni di interfaccia con il *VITRUAL REALITY TOOLBOX*

Il simulatore è stato progettato per studiare una serie di sistemi con un numero di agenti non costante. Per questo motivo è necessario che ci siano delle funzioni che creino, per ogni sistema simulato, un mondo virtuale che rappresenti la situazione che si sta simulando. Inoltre è necessaria una funzione che, per ogni simulazione, ne salvi la rappresentazione virtuale in un file video.

function world=CreateFileWrl(N) Prende in ingresso il numero di agenti del sistema N, e crea un mondo virtuale, in linguaggio VRML, con:

- Una sfera colorata per ogni agente del sistema. Ogni sfera avrà un colore diverso per distinguersi dalle altre.
- Un parallelepipedo di colore verde che rappresenta il piano nel quale si muovono gli agenti.
- Un *background* di colore nero per esaltare il contrasto con gli altri oggetti istanziati.
- Un *viewpoint* che rappresenta il punto da cui l'osservatore visulizzerà la scena.

Il mondo virtuale viene memorizzato nel file '*newWorld.wrl*'. La Figura 4.3 mostra esempio di come viene visualizzato il file '*newWorld.wrl*' corrispondente ad un sistema con 8 agenti, creato dalla funzione *CreateFileWrl*. Si osservi che

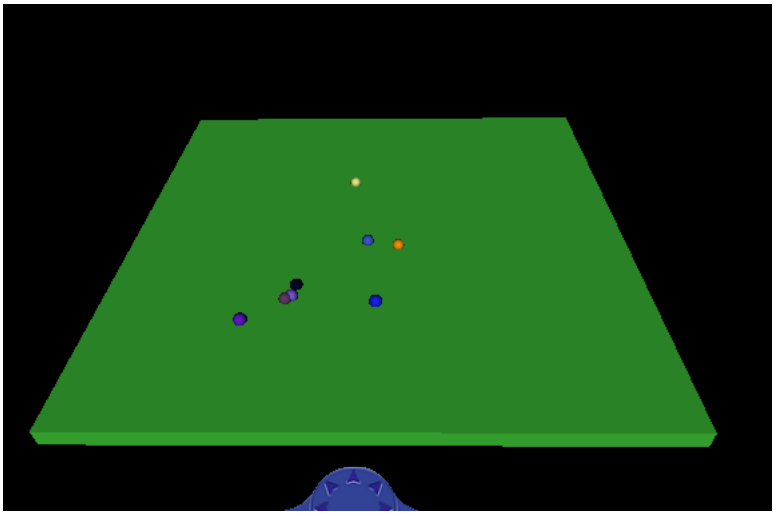


Figura 4.3: Rappresentazione virtuale di un MAS con 8 agenti

nella trattazione dei capitoli precedenti abbiamo ipotizzato che gli agenti fossero puntiformi, mentre in questo caso vengono rappresentati come sfere con un determinato raggio. Questa differenza fa si che, in alcuni casi, si verifichino sovrapposizioni tra sfere che rappresentano agenti che si trovano molto vicini.

function tab = CreateEvolutionTab(Array) Crea una matrice di valori con $3N+1$ righe, dove vengono memorizzate le evoluzioni spaziali di tutti gli agenti. Per ogni agente viene memorizzata una posizione ogni 0.05 secondi, in modo che l'evoluzione temporale sia descritta con una frequenza di 20 pos/sec. La matrice è formata in questo modo:

- Nella prima riga viene memorizzato il valore della variabile tempo.
- Nelle righe 2,3 e 4 viene memorizzata la successione delle posizioni occupate dall'agente identificato con l'id 1. Ogni posizione è espressa nelle coordinate globali X (riga 2), Y (riga 3) e Z (riga 4). Poichè sono stati simulati spostamenti bidimensionali, ogni agente avrà sempre coordinata $Z = 0$.
- Nelle successive righe, a gruppi di 3, è memorizzata l'evoluzione degli altri agenti che fanno parte del sistema. La matrice viene memorizzata nella variabile *tab*.

VirtualEvolution e NewVirtualEvolution Sono le due funzioni che creano la connessione tra la variabile *tab*, che memorizza l'evoluzione temporale degli agenti, e il mondo virtuale. Il loro funzionamento può essere sintetizzato in questo modo:

- viene lanciato il player che permette di visualizzare il mondo virtuale descritto nel file '*newWorld.wrl*';
- viene agganciato ogni oggetto presente nel mondo virtuale ai corrispondenti valori della variabile *tab*;
- per ogni colonna della variabile *tab* viene creata un'immagine nel mondo virtuale. Il flusso delle immagini viene memorizzato in un file video, in formato *.avi* con una frequenza di immagine di 20 frame al secondo;
- dopo che viene creato il file video, si può modificare dal player il punto da cui la scena viene visualizzata. Invocando la funzione *NewVirtualEvolution* viene creato un nuovo video che permette di visualizzare l'evoluzione del sistema dal nuovo punto di vista;

Capitolo 5

Risultati dei test

In questo capitolo vengono riportati i risultati dei diversi test sul sistema, descritto nel Capitolo 3, che sono stati effettuati col simulatore descritto nel Capitolo 4 . Gli algoritmi utilizzati nelle simulazioni sono:

$$\mathbf{x}(k+1) = (\mathbf{I} - q\mathbf{I})\mathbf{x}(k) + (q\mathbf{I})\mathrm{x_d} \tag{5.1}$$

$$\mathbf{s}(k+1) = \mathbf{W}(k)\mathbf{s}(k) \tag{5.2}$$

L'equazione 5.1 rappresenta l'algoritmo di convergenza di ciascun agente alla relativa posizione obiettivo. Tale equazione rappresenta la versione base dell'algoritmo, da cui sono state ricavate altre due varianti adattative. L'equazione 5.2 rappresenta l'algoritmo di tipo gossip che viene utilizzato per la stima di due punti, $P1$ e $P2$, che vengono utilizzati per la determinazione di un frame comune per tutti gli agenti. Sono stati effettuati tre tipi di test:

- test per la verifica del funzionamento del simulatore.
- test per l'analisi del comportamento del sistema sotto l'azione dei due algoritmi applicati separatamente.
- test per l'analisi del comportamento e delle prestazioni del sistema sotto l'azione simultanea dei due algoritmi.

Nella trattazione che segue si confronteranno i risultati delle simulazioni con quelli teorici, si valuteranno in maniera qualitativa i parametri che influenzano i tempi di convergenza dei due algoritmi e si cercherà di valutare eventuali limiti applicativi e regole di implementazione che, se rispettate, garantiscano

una probabilità di convergenza accettabile. Si ipotizzeranno, alla fine, alcune possibili procedure per poter implementare sistemi di questo tipo in modo che manifestino, con una certa approssimazione, il comportamento desiderato.

5.1 Test per la verifica del corretto funzionamento del simulatore

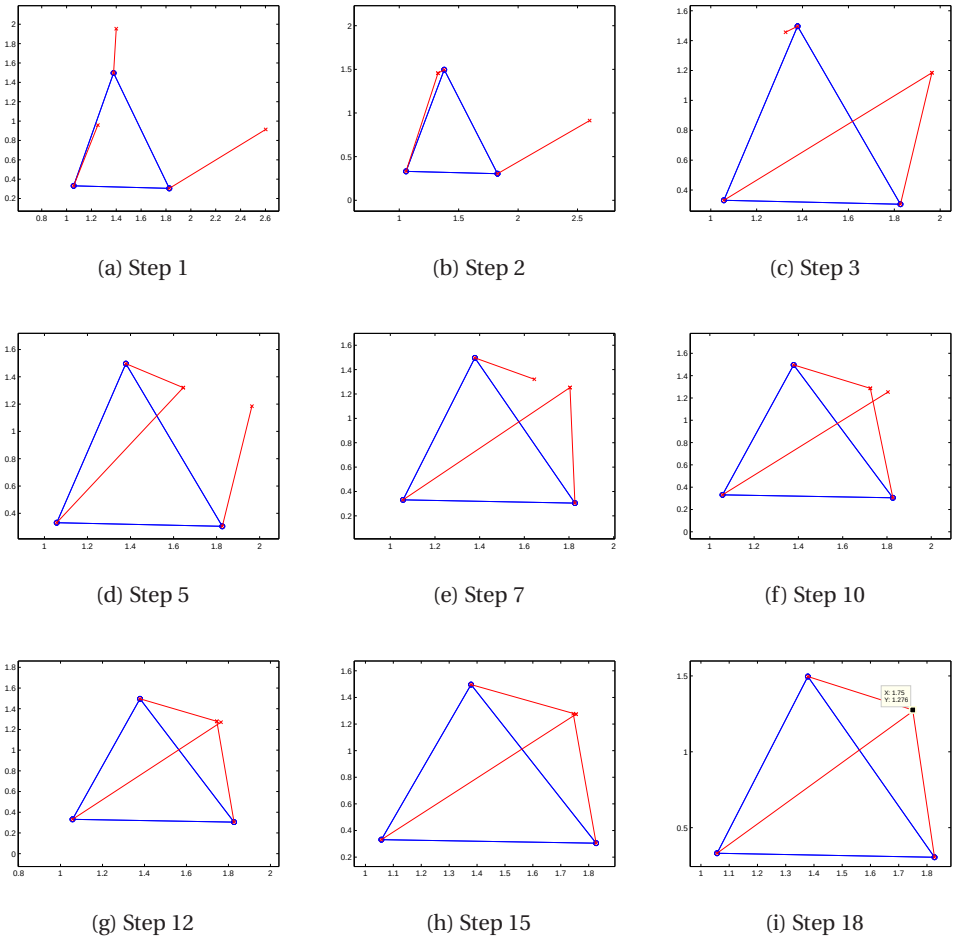
5.1.1 Test 1: Convergenza delle stime locali ad un punto comune

Per valutare la corretta implementazione dell'algoritmo per la convergenza ad un frame comune, bisogna prima verificare che le stime locali convergano asintoticamente ad un punto comune. È stato applicato l'algoritmo 5.2 ad un sistema con $N = 3$ agenti con grafo di comunicazione completo. Ci si aspetta che la stima converga al baricentro delle stime iniziali. In Tabella 5.1 sono riportati i dati relativi alle stime iniziali.

Agent	X	Y
Agent 1	1.2519	0.9587
Agent 2	1.4000	1.9541
Agent 3	2.6020	0.9136
Media	1.7513	1.2754

Tabella 5.1: Dati relativi al Test sulla Convergenza ad un Punto Comune

La Figura 5.1 mostra come le stime del punto comune convergano asintoticamente, e il punto di convergenza è esattamente il baricentro delle stime iniziali. Successive prove su sistemi con un maggior numero di agenti hanno confermato il corretto funzionamento dell'algoritmo.



(a) Step 1 (b) Step 2 (c) Step 3

(d) Step 5 (e) Step 7 (f) Step 10

(g) Step 12 (h) Step 15 (i) Step 18

Figura 5.1: Test1, Tre Agenti si accordano su un Punto Comune

5.1.2 Test 2: Convergenza delle stime locali ad un frame comune

Il test è simile a quello precedente, ma ora gli agenti si accordano due punti in comune ($P1$ e $P2$), e da questi determinano la loro stima del frame comune. Ci si aspetta che le stime convergano ad un sistema di assi centrato nel baricentro delle stime iniziale del punto $P1$. In Tabella 5.2 sono riportati i dati relativi alle stime iniziali.

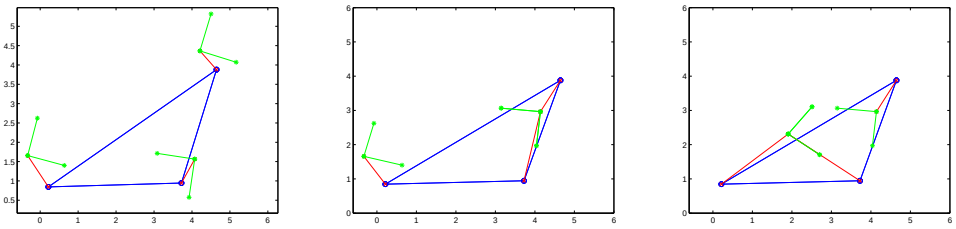
Agent	X	Y
Agent 1	-0.3305	1.6568
Agent 2	4.0724	1.5641
Agent 3	4.2100	4.3644
Media	2.6506	2.5284

Tabella 5.2: Dati relativi al test sulla convergenza ad un frame comune

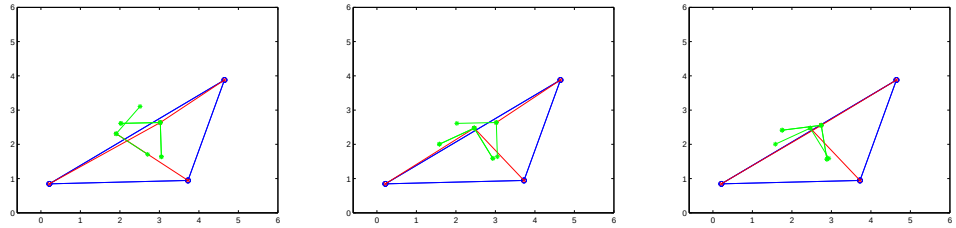
La Figura 5.2 mostra come gli agenti raggiungano effettivamente un accordo su un frame comune, e il centro degli assi è esattamente il baricentro delle stime iniziali del punto $P1$. Il corretto funzionamento del simulatore è stato confermato da una serie di simulazioni successive.

5.1.3 Test 3: Convergenza alle posizioni desiderate

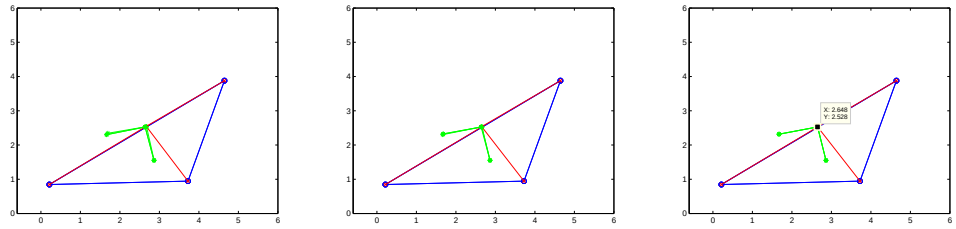
Per testare il corretto funzionamento dell'algoritmo 5.1 è stato simulato un rendezvous di un sistema con $N = 8$ agenti nell'origine del sistema di riferimento globale. È bastato mettere come posizione obbiettivo di ogni agente il punto di coordinate globali $(0,0)$. Gli agenti convergono nel punto desiderato, come si può vedere in Figura 5.3



(a) Step 1 (b) Step 2 (c) Step 3



(d) Step 5 (e) Step 7 (f) Step 10

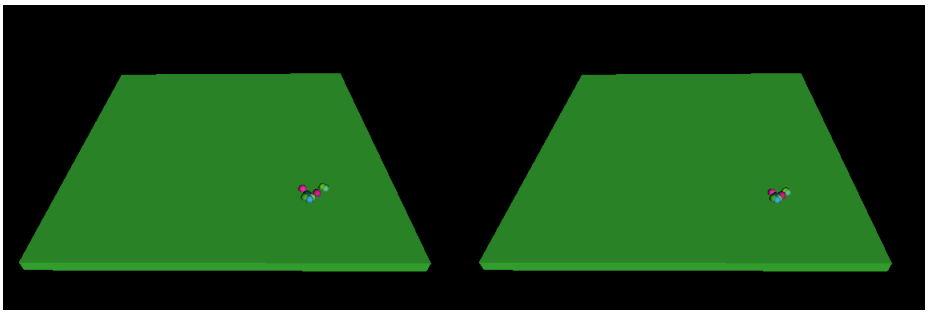


(g) Step 12 (h) Step 15 (i) Step 18

Figura 5.2: Test1, Tre agenti si accordano su un punto comune

(a) Step 1 (b) Step 2

(c) Step 3 (d) Step 4



(e) Step 5 (f) Step 6

Figura 5.3: Test3, rendez-vous di 8 agenti nell'origine

5.2 Test sulla convergenza degli algoritmi

Per valutare il tempo di convergenza degli algoritmi 5.1 e 5.2, sono stati effettuati alcuni test facendo variare di alcune caratteristiche del sistema, per valutare quali sono i parametri che influenzano questa grandezza. Si indicherà con t_s il tempo di campionamento dell'algoritmo 5.1, mentre con t_g il *tempo di gossip*, cioè il tempo medio tra un gossip e l'altro, che per semplicità verrà considerato costante.

Nei test che sono stati effettuati, per *tempo di convergenza medio relativo* t_{ave} si intende il numero di passi computazionali necessari a portare il valore dello stato di tutti gli agenti entro i seguenti limiti:

- $x_i(k) \in \{x_d \pm 0.001 x_d\}$ per le posizioni obiettivo.
- $s_i(k) \in \{\frac{1}{N}\sum s_i(0) \pm 0.1\%\}$ per la stima dei punti comuni.

Per *tempo di convergenza medio assoluto* t_{abs} si intende il tempo necessario, espresso in secondi, per portare il valore dello stato di tutti gli agenti entro i limiti che sono stati definiti. Inizialmente gli algoritmi sono stati testati separatamente.

Nel caso dell'algoritmo 5.1 sono stati effettuati i seguenti test, supponendo un valore di q uguale per tutti gli agenti:

- variazione del tempo di convergenza relativo t_{ave} al variare del parametro q per la limitazione della velocità;
- variazione del tempo di convergenza relativo t_{ave} al variare del numero di agenti N;
- variazione del tempo di convergenza assoluto t_{abs} al variare del parametro q.

Poiché il tempo di campionamento t_s non influenza la velocità di convergenza del sistema, si assume che i risultati ottenuti dalle simulazioni effettuate siano validi per qualsiasi valore di t_s. Ovviamente t_s inciderà sul tempo di convergenza assoluto t_{abs}: a parità di tempo di convergenza relativo t_{ave}, il tempo di convergenza assoluto crescerà linearmente al crescere di t_s. La topologia della rete non influenza il tempo di convergenza del sistema, in quanto ogni agente si muove verso la posizione obiettivo indipendentemente dagli altri.

$$t_{abs} = t_s \cdot t_{ave}$$

Nel caso dell'algoritmo 5.2 sono stati effettuati i seguenti test:

- variazione del tempo di convergenza relativo t_{ave} al variare del tempo di gossip t_g;
- variazione del tempo di convergenza relativo t_{ave} al variare del numero di agenti N;
- variazione del tempo di convergenza relativo t_{ave} al variare della topologia della rete;
- variazione del tempo di convergenza assoluto t_{abs} al variare del tempo di gossip t_g.

5.2.1 Convergenza degli agenti alla posizione obiettivo

Test 1: Variazione di t_{ave} al variare dei q.

I dati relativi all'esperimento sono riportati in Tabella 5.3. Sono state effettuate 201 simulazioni diverse per 201 diversi valori di q, e per ogni simulazione sono state effettuate 10 ripetizioni. Per ogni valore di q è stata fatta la media dei tempi di convergenza relativi ottenuti nelle 10 ripetizioni effettuate. In Figura 5.4 viene rappresentata in ordinata la variazione del tempo di convergenza relativo medio al variare del parametro q.

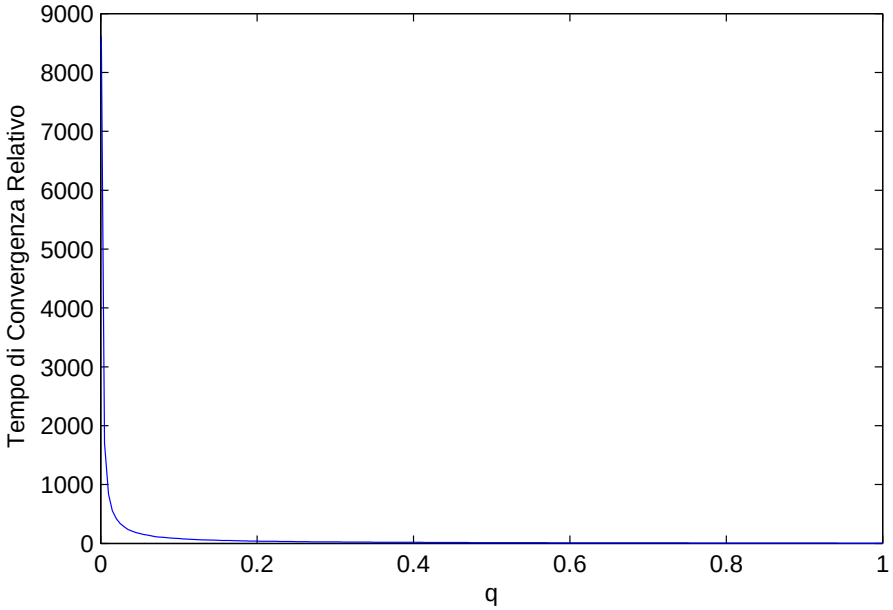
Numero di Agenti (N)	10
Tempo di Campionamento t_s	0.05 sec
Valore Minimo di q	0.001
Valore Massimo di q	1
Numero di simulazioni	201
Numero di ripetizioni per simulazione	10

Tabella 5.3: Variazione del tempo di convergenza relativo al variare di q.

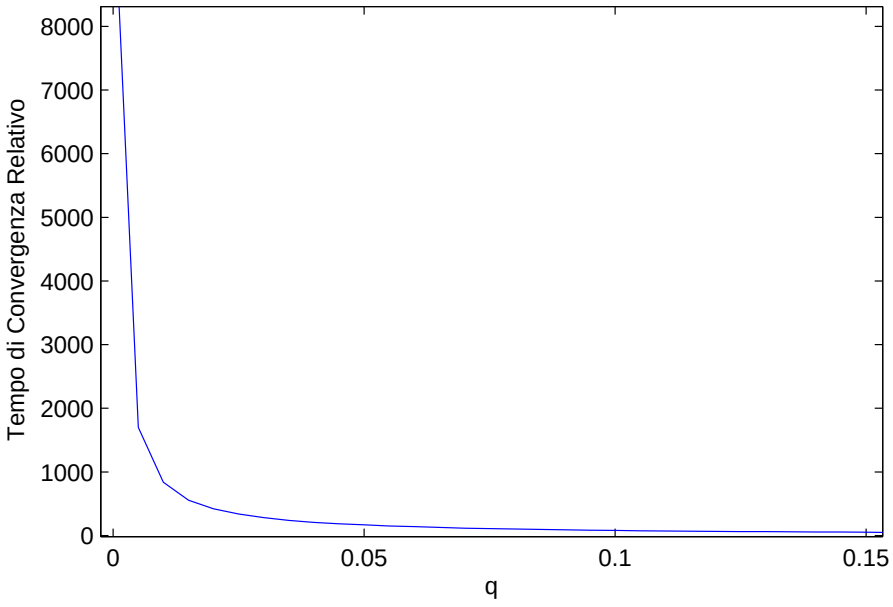
Si può osservare come il tempo di convergenza relativo aumenti al diminuire di q. In base ai risultati ottenuti si ricava che:

$$-\frac{7}{\ln(1-q)} \leq t_{ave} \leq -\frac{10}{\ln(1-q)}$$

Questa regola, ricavata dalle simulazioni, permette di prevedere, per come è stato definito, il tempo di convergenza medio relativo al variare di q.



(a)



(b)

Figura 5.4: Variazione di t_{ave} al variare di q.

Test 2: Variazione di t_{ave} al variare del numero di agenti N

I dati relativi all'esperimento sono riportati in Tabella 5.4. Per ogni valore di N è stata fatta la media dei tempi di convergenza relativi ottenuti nelle 10 ripetizioni effettuate. In Figura 5.5 viene rappresentata in ordinata la variazione del tempo di convergenza relativo medio al variare del numero degli agenti N. Si può osservare che il tempo di convergenza relativo non dipende da N.

Numero di Agenti Minimo	3
Numero di Agenti Massimo	30
Tempo di Campionamento t_s	0.05 sec
Valore di q	0.05
Numero di simulazioni	28
Numero di ripetizioni per simulazione	10

Tabella 5.4: Variazione di t_{ave} al variare di N.

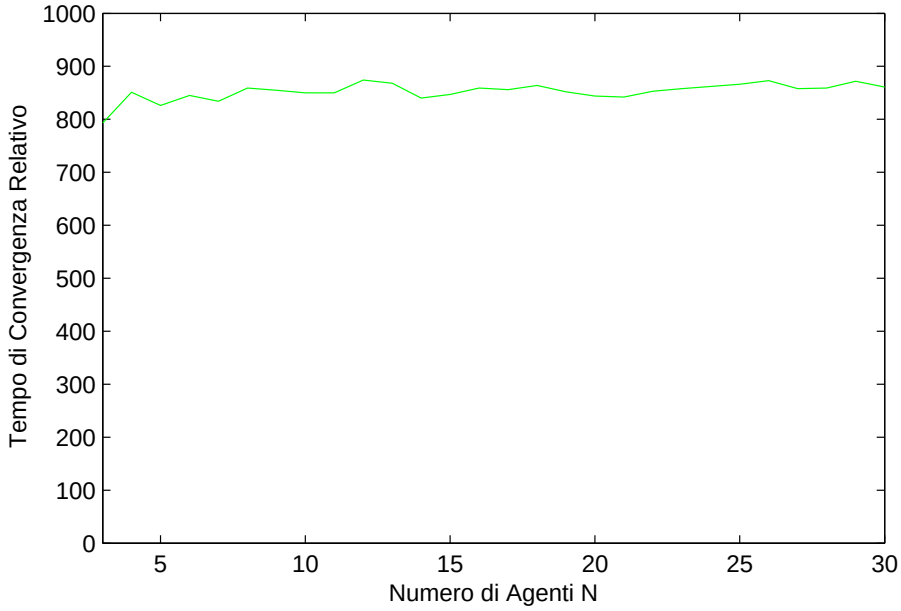


Figura 5.5: Variazione di t_{ave} al variare di N.

Test 3: Variazione di t_{abs} al variare del parametro q

Il tempo di convergenza medio relativo t_{ave} e il tempo di convergenza assoluto t_{abs} sono legati dalla relazione

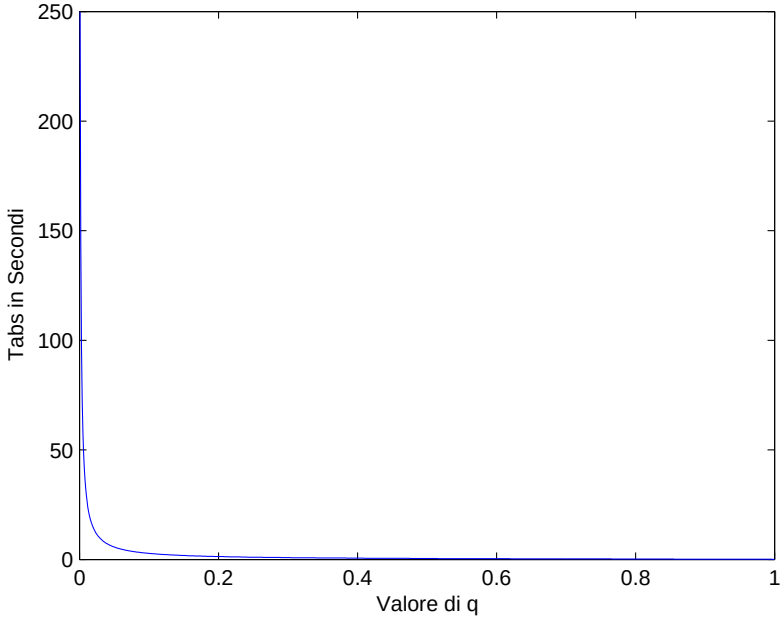
$$t_{abs} = t_s \cdot t_{ave}$$

Per tale motivo sono stati utilizzati gli stessi dati ottenuti dal test sulla variazione di t_{ave} al variare di q, i cui dati sono riportati in Tabella 5.3 I risultati sono presentati in Figura 5.6.

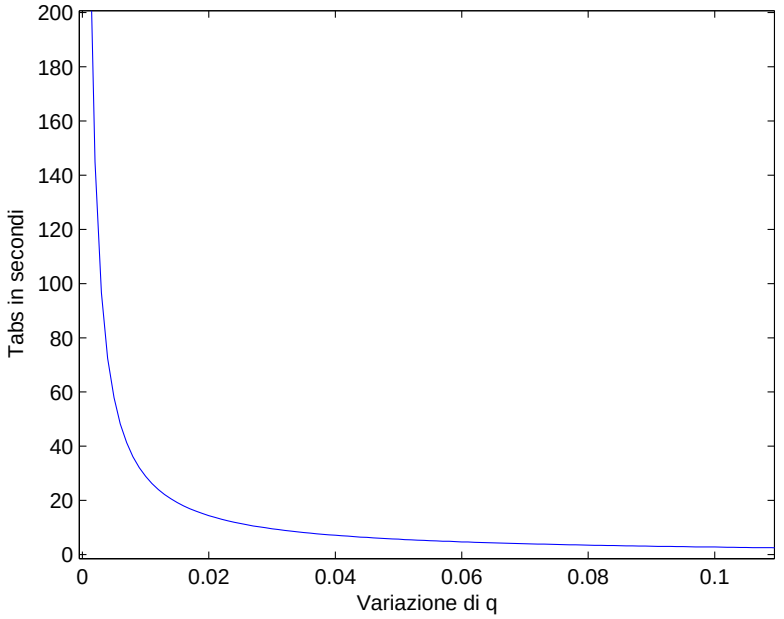
Fissato t_s, la procedura per far convergere gli agenti alla posizione obbiettivo nel tempo desiderato t_d è la seguente:

- Si calcola il tempo di convergenza relativo desiderato: $t_{ave_d} = \lfloor \frac{t_d}{t_s} \rfloor$
- Poichè $-\frac{7}{\ln(1-q)} \leq t_{ave} \leq -\frac{10}{\ln(1-q)}$, invertendo questa relazione si ottiene il valore di q.

$$1 - e^{-\frac{7}{t_{ave_d}}} \leq q \leq 1 - e^{-\frac{10}{t_{ave_d}}}$$



(a)



(b)

Figura 5.6: Variazione di t_{abs} al variare di q.

5.2.2 Convergenza della stima sul frame comune

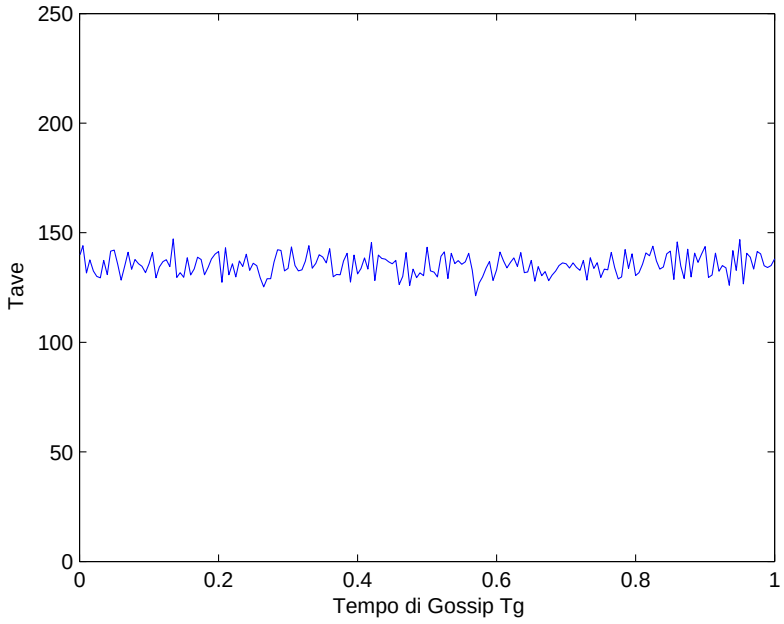
Test 1: Variazione di t_{ave} al variare di t_g.

I dati relativi all'esperimento sono riportati in Tabella 5.5. Sono state effettuate 201 simulazioni diverse per 201 diversi valori di t_g, e per ogni simulazione sono state effettuate 10 ripetizioni. Per ogni valore di t_g è stata fatta la media dei tempi di convergenza relativi ottenuti nelle 10 ripetizioni effettuate. In Figura 5.7 viene rappresentata in ordinata la variazione del tempo di convergenza medio relativo al variare tempo di gossip t_g.

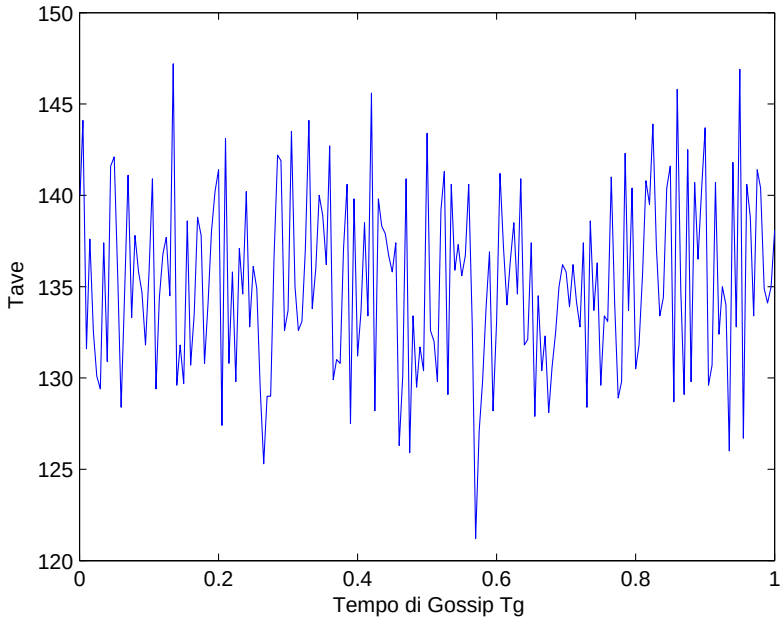
Topologia del Grafo	Grafo Completo
Numero di Agenti (N)	10
Tempo di Campionamento Min $t_{g_{min}}$	0.001 sec
Tempo di Campionamento Min $t_{g_{max}}$	0.001
Numero di simulazioni	201
Numero di ripetizioni per simulazione	10

Tabella 5.5: Gossip: Variazione di t_{ave} al variare di t_s.

Si può osservare che il tempo di convergenza medio relativo t_{ave} è indipendente dal tempo di gossip t_g. Le variazioni evidenziate dalla curva in Figura 5.7 dipendono dalla natura casuale del processo di selezione degli archi.



(a)



(b)

Figura 5.7: Test 1: Variazione di t_{ave} al variare di t_g.

Test 2: Variazione di t_{ave} al variare di N.

I dati relativi al test sono riportati in Tabella 5.6. Per ogni valore di N è stata fatta la media dei tempi di convergenza relativi ottenuti nelle 10 ripetizioni effettuate. In Figura 5.8 viene rappresentata in ordinata la variazione del tempo di convergenza relativo medio al variare del numero degli agenti N. Si può osservare che il t_{ave} cresce al crescere di N.

Topologia del Grafo	Grafo Completo
Numero di Agenti Minimo	3
Numero di Agenti Massimo	30
Tempo di Gossip t_g	0.01 sec
Numero di simulazioni	28
Numero di ripetizioni per simulazione	10

Tabella 5.6: Variazione di t_{ave} al variare di N.

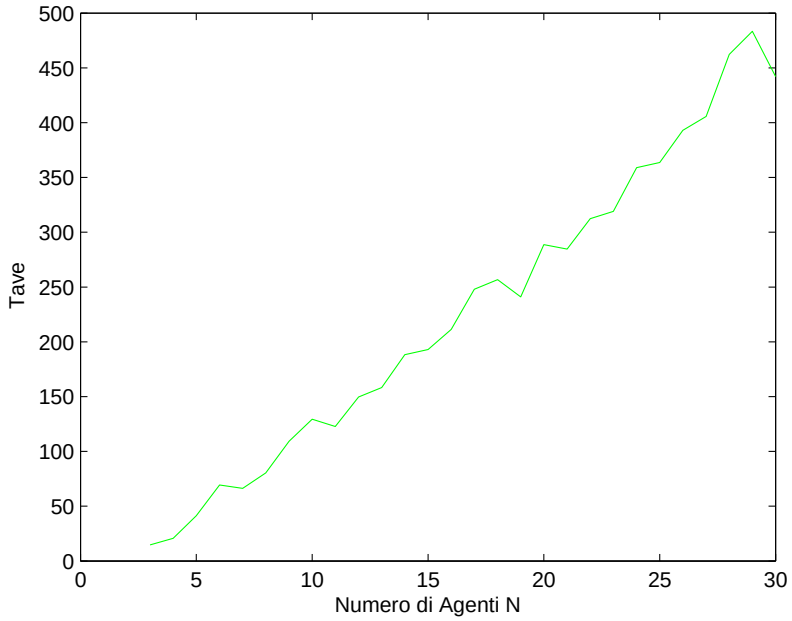


Figura 5.8: Test 2: Variazione di t_{ave} al variare di N.

Test 3: Variazione di t_{ave} al variare della topologia della rete.

È stato ripetuto il Test 1 su tre diverse tipologie di grafo tempo-invariante:

- Grafo Completo
- Grafo con $N-1$ archi
- Grafo con un numero di archi $(N-1) \leq N_{archi} \leq N^2$

I risultati del test sono rappresentati in Figura 5.9. Si può osservare che all'aumentare del numero degli archi, e quindi dei canali di comunicazione tra gli agenti, diminuisce il tempo di convergenza relativo t_{ave}, e questo avviene perchè aumenta la probabilità di comunicazione tra gli agenti.

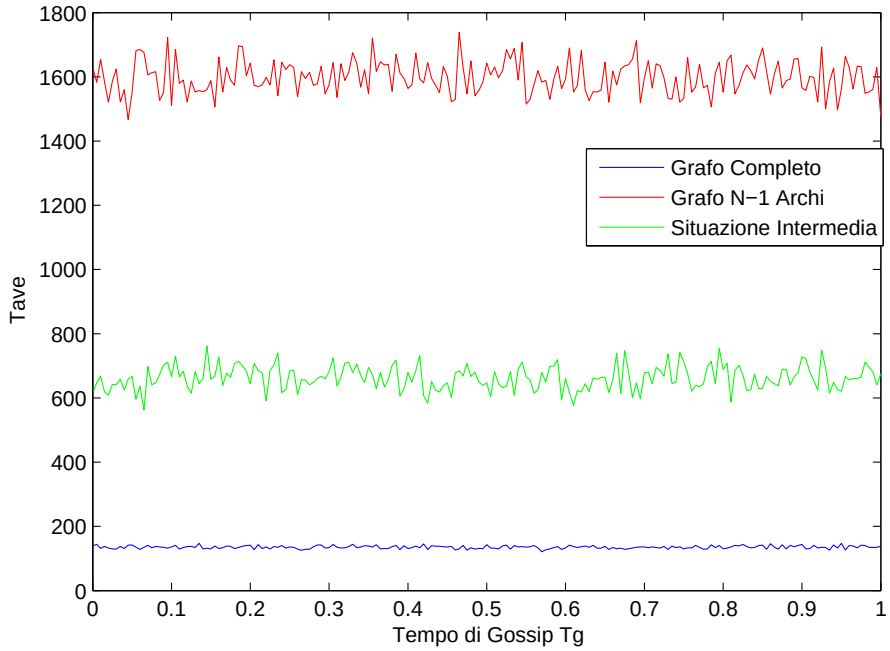


Figura 5.9: Test 3: Variazione di t_{ave} al variare della Topologia.

Test 4: Variazione di t_{abs} al variare di t_g.

La relazione che lega t_{abs} e t_g è la seguente:

$$t_{abs} = t_g \cdot t_{ave}$$

Poichè t_{ave} rimane costante al variare di t_g, ci si aspetta che t_{abs} cresca linearmente al variare di t_g. Questo andamento è osservabile in figura 5.10, nonostante le variazioni dovute alla natura random dal processo di selezione degli archi.

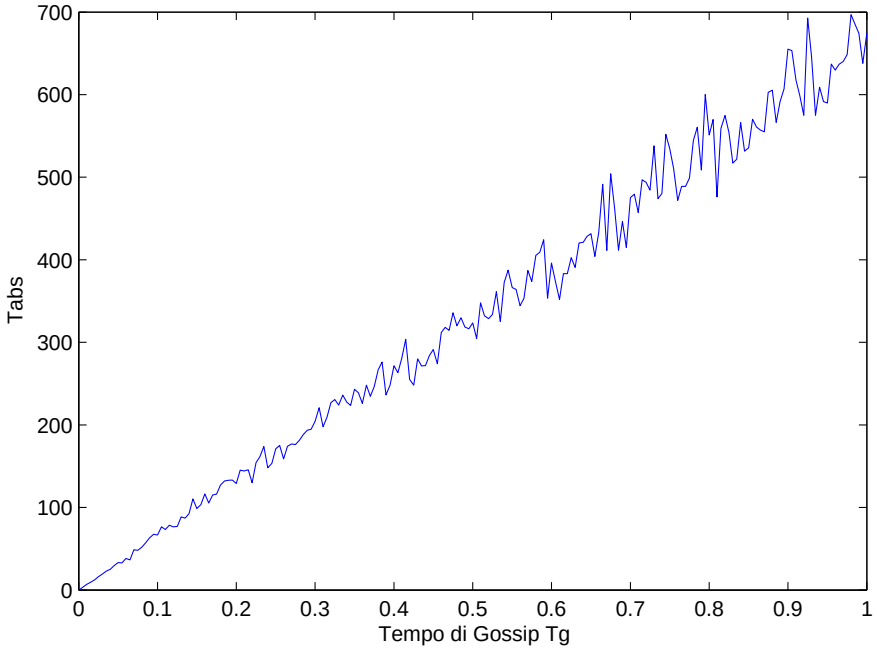


Figura 5.10: Test 4: Variazione di t_{abs} al variare di t_g.

5.3 Applicazione degli algoritmi in successione

Il caso che rappresenta l'implementazione più semplice del sistema è quello in cui gli agenti raggiungono la formazione obiettivo applicando gli algoritmi in due momenti distinti dove:

- ciascun agente comunica con gli altri per raggiungere un accordo su un frame comune;
- una volta raggiunto l'accordo sul frame comune, e aver determinato l'esatta posizione obiettivo, ciascun agente si sposta verso di essa alla massima velocità possibile.

Risulta evidente che il tempo di convergenza totale è dato idealmente dalla somma dei tempi di convergenza dei due algoritmi. Supponendo che ogni agente si sposti ad una velocità massima $q_{max} = 0.25$, sulla base dei risultati ottenuti nei precedenti test, in Figura 5.11 viene mostrato il tempo di convergenza t_{ave} al variare del parametro q.

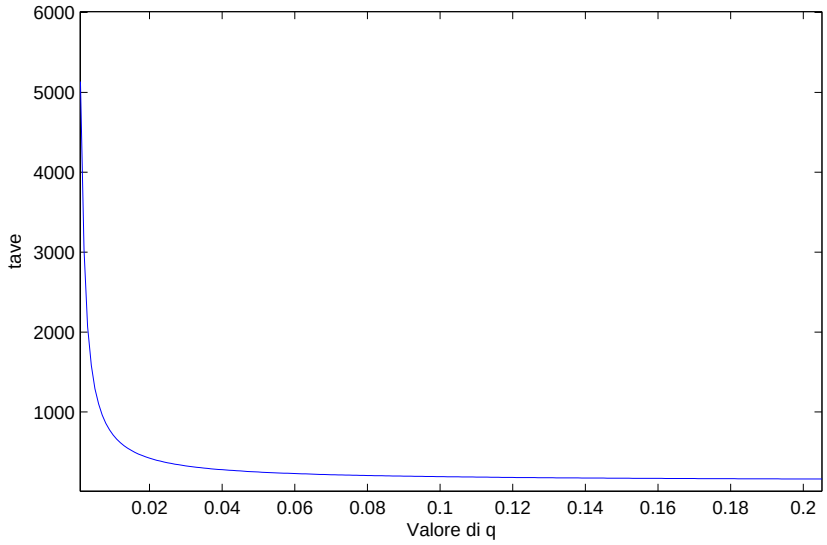


Figura 5.11: Variazione di t_{ave} al variare di q.

5.4 Applicazione parallela degli algoritmi.

I test più interessanti sono quelli relativi al comportamento del sistema quando i due algoritmi presentati vengono applicati contemporaneamente. Il comportamento è quello descritto nel Capitolo 3, cioè:

- gli agenti comunicano tra di loro per raggiungere un accordo su un frame comune, secondo l'algoritmo 5.2. Il tempo medio che intercorre tra due gossip è t_g;
- ogni agente aggiorna la sua posizione con una frequenza $f_s = \frac{1}{t_s}$, dove t_s è il tempo di campionamento dell'algoritmo 3.5.

La natura random dell'algoritmo 5.2 rende molto complicata un'analisi rigorosa relativa al comportamento del sistema, perciò nei test sono state fatte una serie di semplificazioni al fine di rendere più agevoli le simulazioni:

- il tempo di gossip è uguale al tempo di aggiornamento delle posizioni, $t_g = t_s$
- la topologia della rete non cambia nel tempo (grafo tempo-invariante)

L'obiettivo delle simulazioni è quello di assegnare agli agenti delle posizioni da raggiungere rispetto al centro del frame comune, in modo che il plotone si disponga secondo formazioni di gruppo prestabilite. Lo spostamento degli agenti è controllato da tre diverse varianti dell'algoritmo 5.1 in cui:

- il parametro q è lo stesso per tutti gli agenti, quindi si muovono tutti con la stessa velocità (algoritmo semplice);
- il parametro q varia nel tempo per ciascun agente a seconda di come varia la stima locale del frame comune (algoritmo adattativo 1);
- il parametro q varia nel tempo per ciascun agente a seconda di come varia la stima locale del frame comune e a seconda di quanti cicli di aggiornamento sono stati effettuati (algoritmo adattativo 2).

Principalmente si è cercato di valutare il tempo di convergenza relativo medio del sistema alle posizioni desiderate t_{ave}, e i campi di applicabilità delle diverse varianti degli algoritmi. Il primo test è stato effettuato su un sistema di 10 agenti, con un grafo di comunicazione completo, per ottenere informazioni sul comportamento del sistema al variare del parametro q. I successivi test sono stati per valutare il comportamento del sistema in questi casi:

- Variazione della topologia della rete
- Variazione del numero di agenti

I risultati ottenuti forniscono indicazioni interessanti su come varia t_{ave} al variare dei parametri caratteristici del sistema, ma da questi non si è stati in grado di ottenere della leggi valide che permettano di determinarlo a priori, eccetto in alcuni casi particolari.

Già dalle prime simulazioni si sono potuti osservare due effetti della sovrapposizione degli algoritmi:

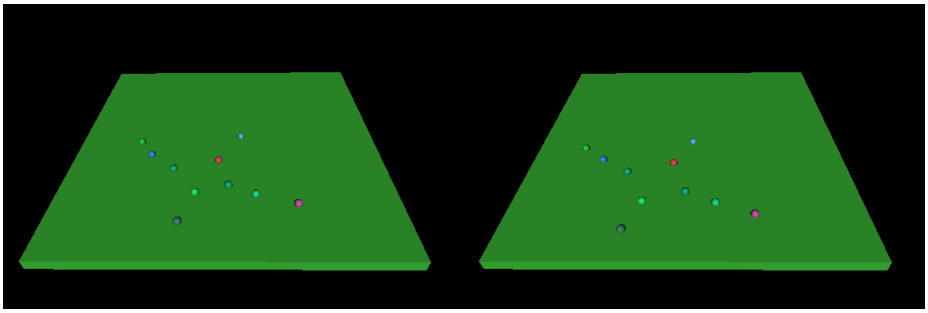
- Se il baricentro delle stime iniziali del punto $P1$ (centro stimato del frame comune) è diverso dal baricentro delle posizioni iniziali degli agenti, cioè se $\frac{1}{n}\sum_i s_i(0) \neq \frac{1}{n}\sum_i p_i(0)$, allora il frame comune, una volta raggiunto, non rimane fermo nella stessa posizione ma trasla. Di conseguenza tutto il plotone trasla seguendo il centro del frame comune.
- Se la somma delle posizioni obiettivo è diversa da 0, cioè $\sum_i x_{di} \neq 0$, allora il plotone trasla.

Se si vuole che gli agenti si dispongano in una determinata formazione senza traslare, si deve inizializzare il sistema in modo che:

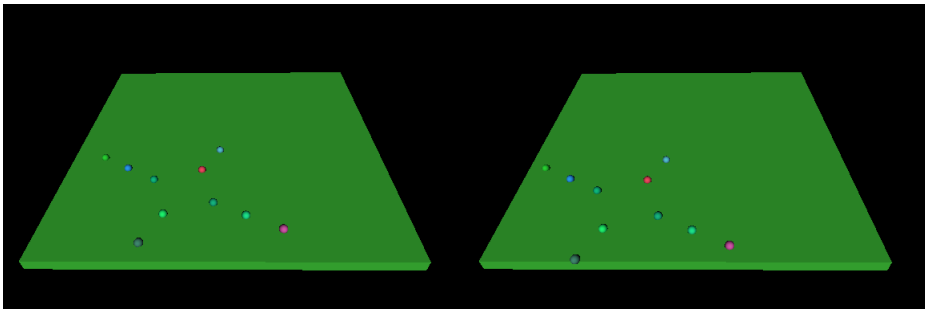
- Per ogni agente la stima iniziale del punto $P1$ coincida con la propria posizione iniziale.
- La somma delle posizioni desiderate sia pari a 0, $\sum_i x_{di} = 0$

In Figura 5.12 viene mostrato un esempio di traslazione del sistema in formazione a causa del fatto che $\frac{1}{n}\sum_i s_i(0) \neq \frac{1}{n}\sum_i p_i(0)$.

(a) (b)



(c) (d)



(e) (f)

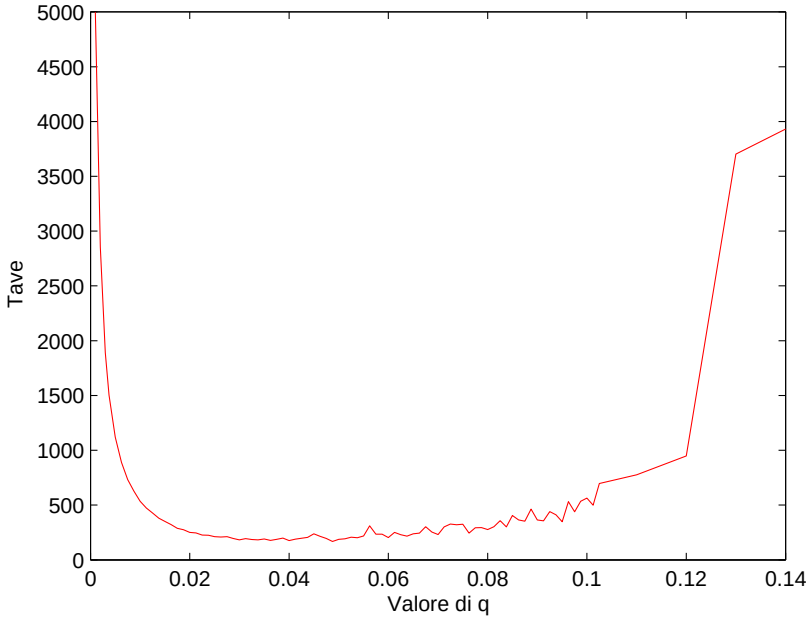
Figura 5.12: Esempio di traslazione del plotone

5.4.1 Test 1: Variazione di t_{ave} al variare di q

I dati relativi all'esperimento sono riportati in Tabella 5.7. Sono state effettuate 100 simulazioni diverse per 100 diversi valori di q, e per ogni simulazione sono state effettuate 10 ripetizioni. Per ogni valore di q è stata fatta la media dei tempi di convergenza relativi ottenuti nelle 10 ripetizioni effettuate. In Figura 5.13 viene rappresentata in ordinata la variazione del tempo di convergenza relativo medio al variare del parametro q.

Numero di Agenti (N)	10
Tempo di Campionamento t_s	0.05 sec
Valore Minimo di q	0.001
Valore Massimo di q	0.14
Numero di simulazioni	100
Numero di ripetizioni per simulazione	10

Tabella 5.7: Variazione di t_{ave} al variare di q.



(a)

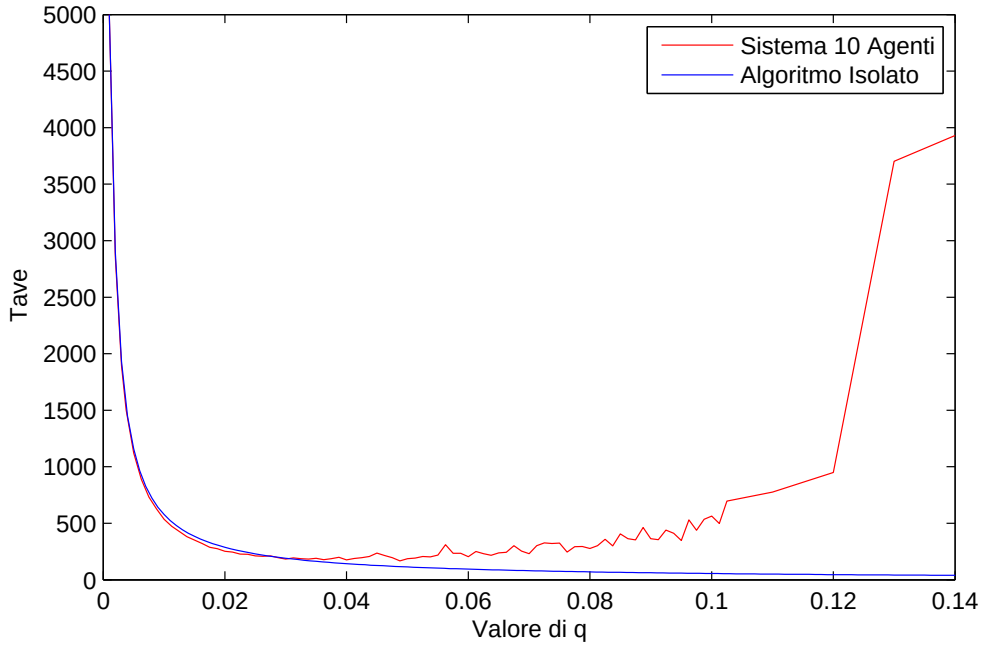
Figura 5.13: Variazione di t_{ave} al variare di q.

In figura 5.14 viene confrontata la curva ottenuta con la curva relativa alla variazione del solo algoritmo per lo spostamento degli agenti al variare di q. Si può osservare che la curva è divisibile in 3 diverse regioni:

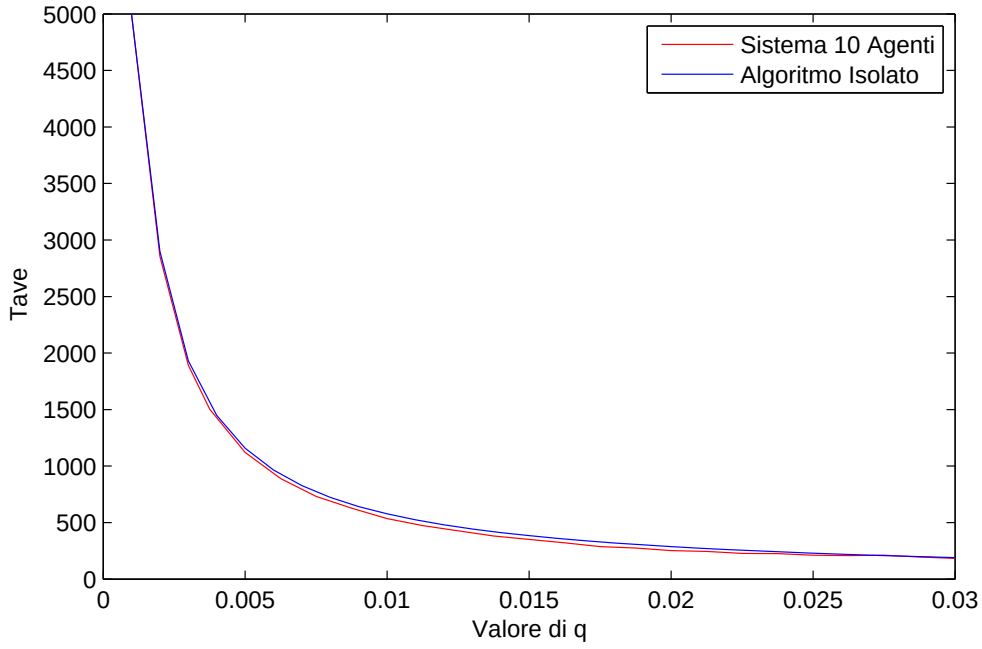
- **Regione1 -** ($0 < q \leq 0.03$): Per valori molto bassi di q il tempo di convergenza t_{ave} del sistema tende a seguire quello dell'algoritmo per lo spostamento degli agenti descritto dall'equazione 5.1. In questa regione l'algoritmo per lo spostamento risulta convergere molto più lentamente rispetto all'algoritmo di tipo gossip 5.2. Il sistema tende a comportarsi come se gli agenti abbiano raggiunto, già dall'inizio, un consenso sul frame comune. Conoscendo il limite q_{L1} al di sotto del quale il sistema si trova in questa regione, si può prevedere il tempo di convergenza t_{ave} utilizzando le relazioni:
$$1 - e^{-\frac{10}{t_{ave}}} \leq q \leq 1 - e^{-\frac{7}{t_{ave}}}$$
$$-\frac{7}{\ln(1-q)} \leq t_{ave} \leq -\frac{10}{\ln(1-q)}$$
Per questo motivo si è deciso di definire questa regione di funzionamento *quasi controllabile*. Un sistema che funziona sotto queste condizioni è presentato nell'esempio 5.4.1.

- **Regione2 -** ($0.03 < q \leq 0.08$): Per un certo range intermedio di valori di q, t_{ave} tende a rimanere costante, o comunque molto vicino ad un valore minimo. I due algoritmi hanno più o meno la stessa velocità di convergenza. I limiti di questa regione sono ($q_{L1} < q \leq q_{L2}$)

- **Regione3 -** ($q < 0.08$): Per valori alti di q, il tempo di convergenza t_{ave} tende a crescere notevolmente. Lo spostamento degli agenti e tanto veloce da non permettere al sistema di raggiungere un accordo sul frame comune. La convergenza non è 'stabile', infatti può capitare che gli agenti riescano a raggiungere la formazione desiderata ma che non riescano a mantenerla. Si è deciso di definire questa regione *non controllabile*. Un sistema che funziona sotto queste condizioni è presentato nell'esempio 5.4.2



(a)



(b)

Figura 5.14: Confronto tra curve.

Esempio 5.4.1. I dati del sistema sono presentati in Tabella 5.8 Il sistema converge alla formazione desiderata e la mantiene stabilmente. Il tempo di convergenza t_{ave} è in accordo con il grafico di Figura 5.14. In Figura 5.15 si possono osservare alcuni passi dell'evoluzione del sistema.

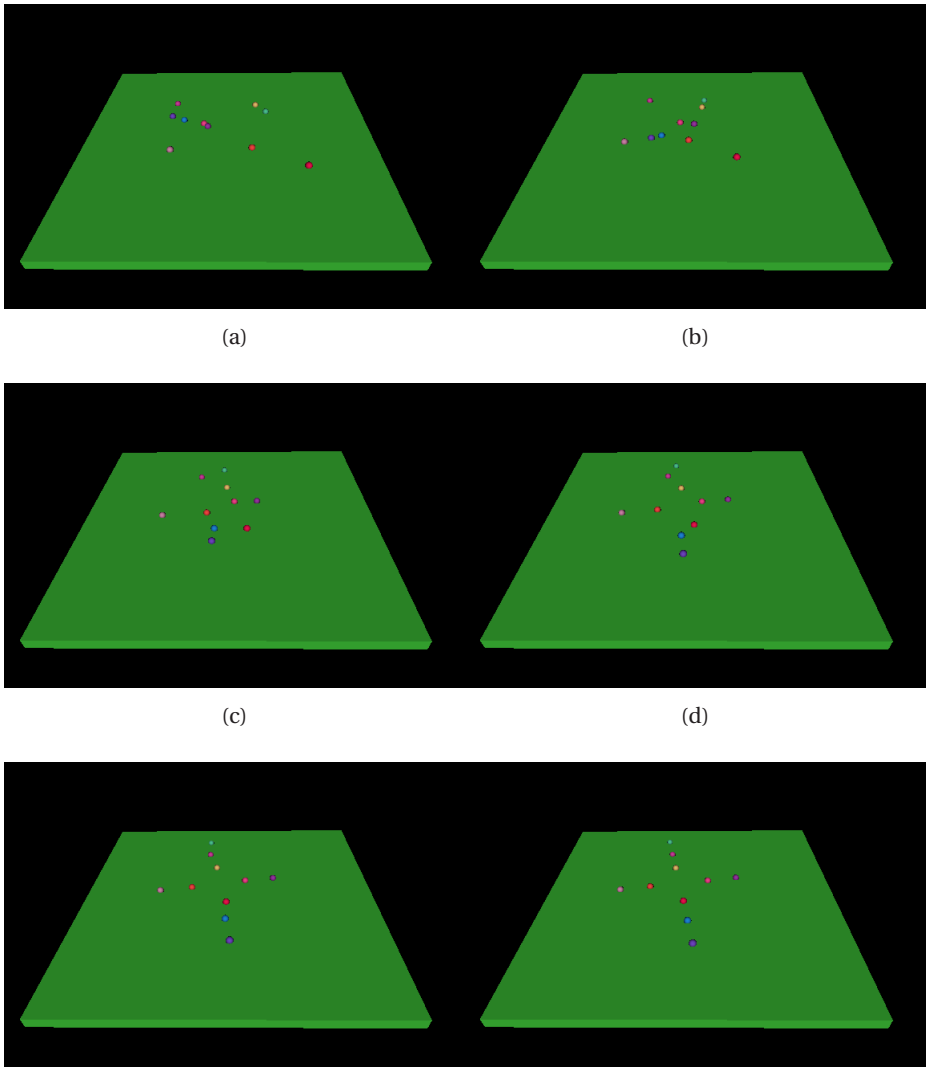
Numero di Agenti (N)	10
Tempo di Campionamento t_s	0.05 sec
Tempo di Gossip t_g	0.05 sec
Valore di q	0.008
Formazione desiderata	Stella
Tempo di convergenza t_{ave}	683 cicli

Tabella 5.8: Sistema di 10 agenti con q in regione quasi controllabile.

Esempio 5.4.2. I dati del sistema sono presentati in Tabella 5.9 Il sistema tende a muoversi in maniera caotica nello spazio, convergendo alla formazione desiderata in un tempo molto lungo. La formazione, una volta raggiunta, non viene mantenuta stabilmente. In Figura 5.16 si possono osservare alcuni passi dell'evoluzione del sistema.

Numero di Agenti (N)	10
Tempo di Campionamento t_s	0.05 sec
Tempo di Gossip t_g	0.05 sec
Valore di q	0.008
Formazione desiderata	Stella

Tabella 5.9: Sistema di 10 agenti con q in *regione non controllabile*.

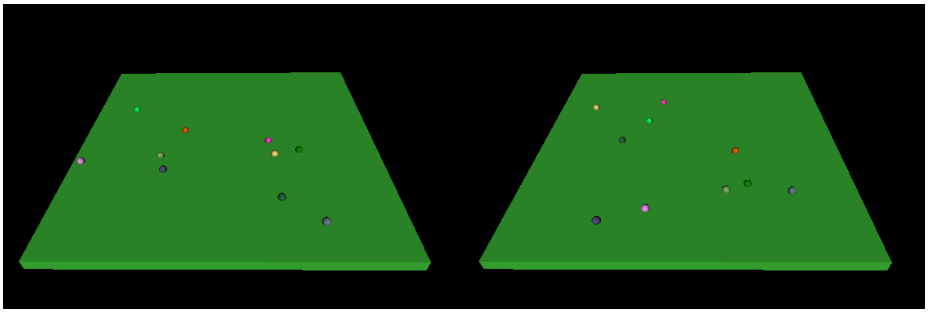


(a) (b)

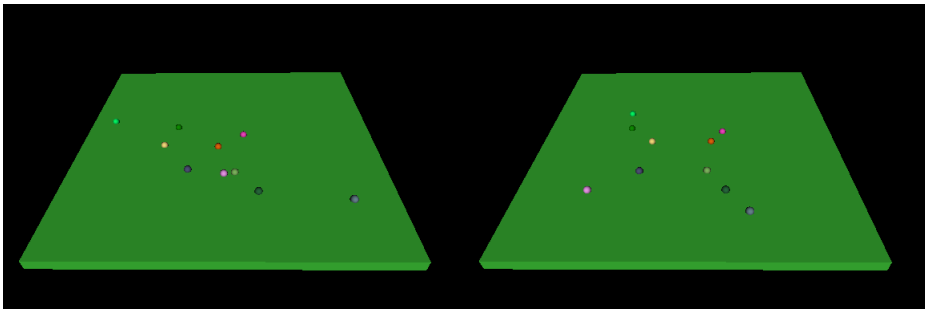
(c) (d)

(e) (f)

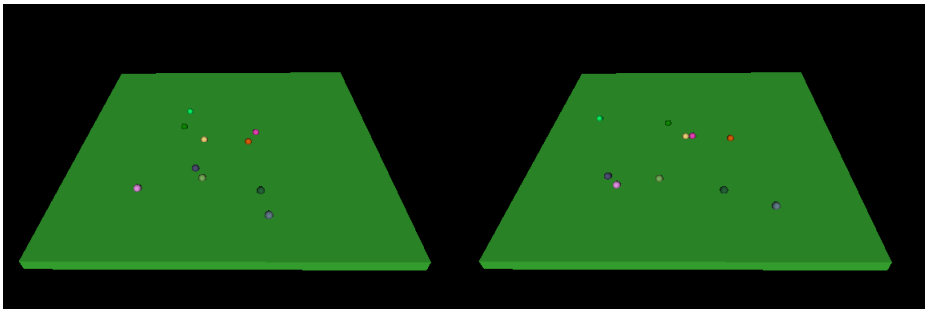
Figura 5.15: Sistema di 10 agenti con q in *regione quasi controllabile*.



(a) (b)



(c) (d)



(e) (f)

Figura 5.16: Sistema di 10 Agenti con q in Regione 3.

È interessante fare il confronto tra la curva che è stata ottenuta applicando gli algoritmi in parallelo e quella relativa all'applicazione degli algoritmi in successione. Dalla Figura 5.17 si può osservare come l'applicazione parallela degli algoritmi abbia delle prestazioni paragonabili a quelle degli algoritmi applicati in successione. Il valore minimo di t_{ave} è lo stesso per entrambe le modalità applicative. Tuttavia l'applicazione parallela degli algoritmi presenta una serie di vantaggi:

- non bisogna conoscere il tempo ottimale per passare da un'algoritmo ad un'altro;
- gli agenti convergono alle posizioni obiettivo con un tempo di convergenza ottimale muovandosi più lentamente;
- in caso di applicazione degli algoritmi in successione è necessario che gli agenti siano in grado di svolgere operazioni più complesse rispetto al caso di algoritmi applicati in parallelo. Infatti, per determinare l'effettivo raggiungimento di un accordo su un frame comune, è necessario che gli agenti siano in grado di utilizzare una serie di tecniche per l'interpretazione dei dati ricevuti dalla rete, che non risultano banali da un punto di vista computazionale.

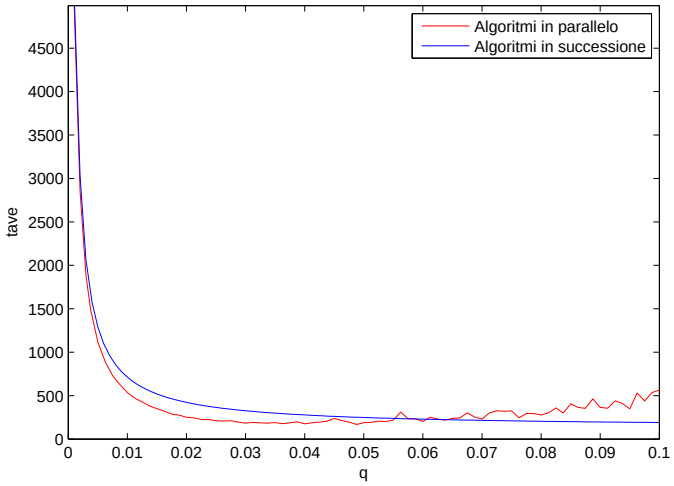


Figura 5.17: Confronto tra applicazione in successione e applicazione parallela degli algoritmi.

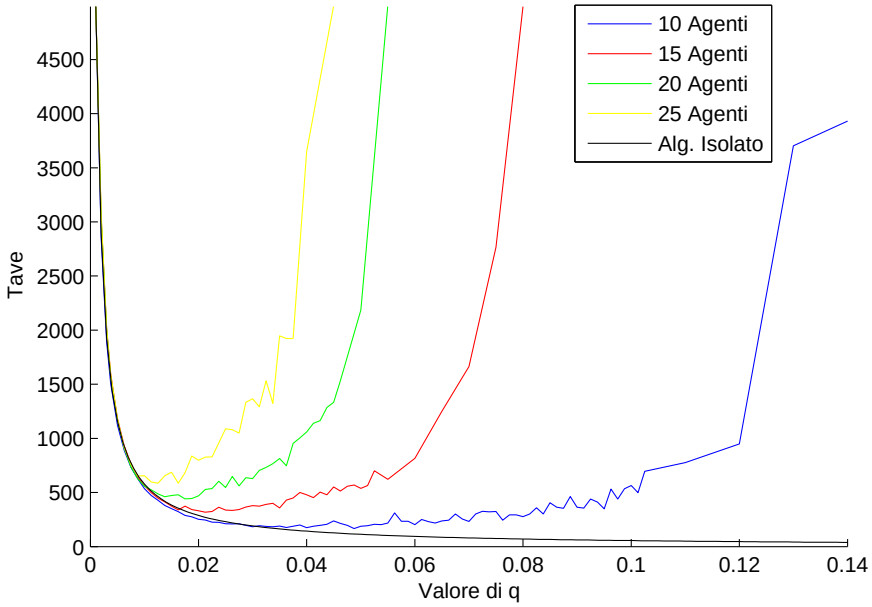
5.4.2 Test 2: Variazione di t_{ave} al variare di N

I dati relativi al test sono riportati in Tabella 5.10. È stato valutato il tempo di convergenza relativo t_{ave} al variare di q su diversi sistemi con un numero diverso di agenti N. Per ogni valore di N è stata fatta la media dei tempi di convergenza relativi ottenuti nelle 10 ripetizioni effettuate. In Figura 5.18 viene rappresentata in ordinata la variazione del tempo di convergenza t_{ave} al variare di q. Si può osservare che la curva rimane la stessa del test precedente, cioè si possono sempre individuare tre regioni di funzionamento, ma cambia la loro larghezza e il valore minimo di t_{ave}. In particolare:

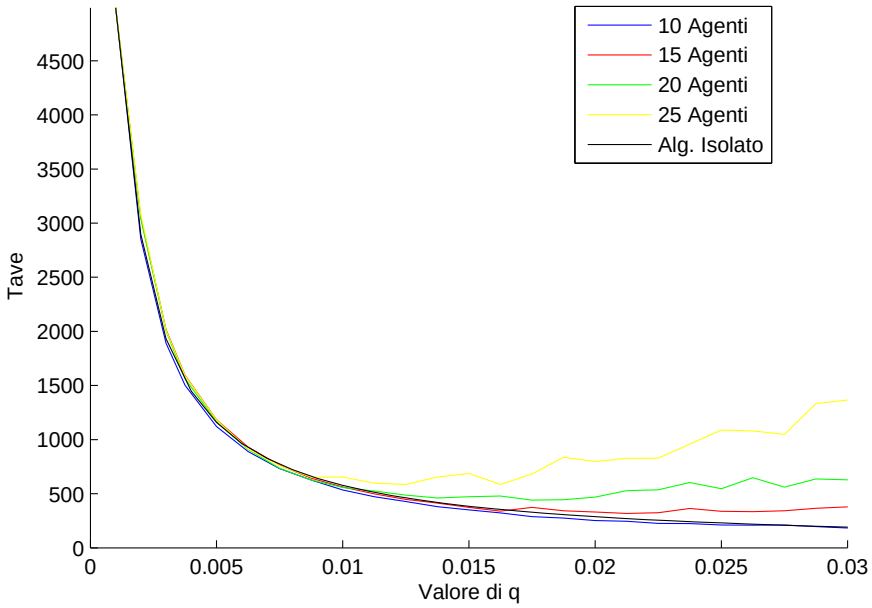
- Il valore minimo di t_{ave} aumenta all'aumentare di N.
- La larghezza della regione quasi controllabile diminuisce (q_{L1} diminuisce) all'aumentare del numero di agenti N.

Topologia del Grafo	Grafo Completo
Numero di Agenti Minimo	3
Numero di Agenti Massimo	30
Tempo di Campionamento t_g	0.05 sec
Tempo di Gossip t_g	0.05 sec
Numero di simulazioni	28
Numero di ripetizioni per simulazione	10

Tabella 5.10: Variazione di t_{ave} al variare di N.



(a)



(b)

Figura 5.18: Variazione di t_{ave} al variare di N

5.4.3 Test 3: Variazione di t_{ave} al variare della topologia del rete

È stato ripetuto il test 1 su tre diverse tipologie di grafo tempo-invariante:

- Grafo Completo
- Grafo con $N-1$ archi
- Grafo con un numero di archi $(N-1) \leq N_{archi} \leq N^2$

I risultati del test sono rappresentati in Figura 5.19. Si può osservare che all'aumentare del numero degli archi, e quindi dei canali di comunicazione tra gli agenti, diminuisce il tempo di convergenza relativo t_{ave} a parità di q, e questo avviene perchè aumenta la probabilità di comunicazione tra gli agenti.

Per qualunque topologia di grafo si presenta sempre lo stesso tipo di curva suddivisibile in 3 regioni. Al diminuire del numero di archi si ha un restringimento della regione quasi controllabile (q_{L1} diminuisce).

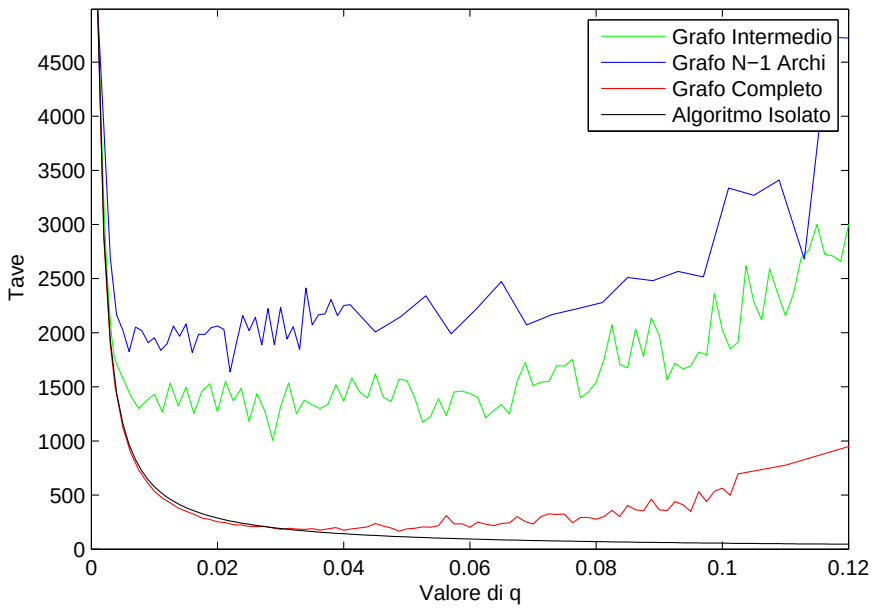


Figura 5.19: Variazione di t_{ave} al variare della topologia della rete

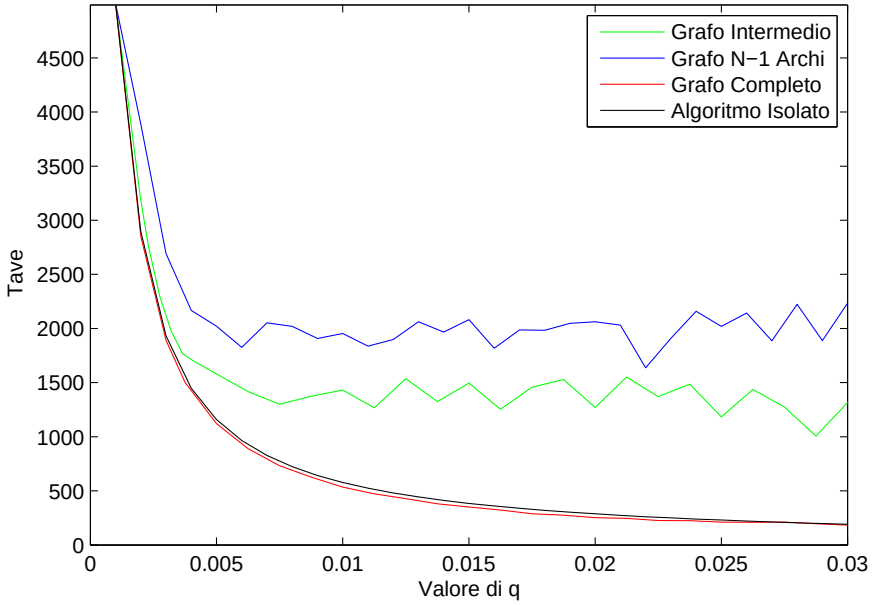


Figura 5.19: Variazione di t_{ave} al variare della topologia della rete, Zoom

5.5 Regole di controllo alternative

L'implementazione che è stata ipotizzata prevede che tutti gli agenti si muovano con la stessa velocità, regolata dal parametro q. Si è pensato di dare la possibilità a ciascun agente di poter modificare la propria velocità in maniera locale utilizzando delle regole adattative che mirano a ottimizzare q in base alle informazioni che si ricevono dall'esterno. Un esempio di controllo decentralizzato della velocità è dato dall'algoritmo adattativo 5.5.1, in cui è stata introdotta una velocità massima q_{max}. Ogni agente modifica la propria velocità a seconda di come varia la stima locale del frame comune tra due gossip successivi in cui è coinvolto. Se la variazione è maggiore di un valore ε_{max} allora l'agente rallenta diminuendo il valore di q. In particolare, nel caso del sistema simulato, il valore di q viene dimezzato. Questo scelta è stata presa utilizzando la tecnica del *trial and error*, e non è stata applicata nessuna tecnica di ottimizzazione. Se la variazione è minore di un valore ε_{min} l'agente suppone che il consenso sul frame comune sia stato raggiunto e accelera, moltiplicando la sua velocità per un fattore 1.5, fino a quando non viene raggiunta la velocità massima consentita q_{max}. La scelta del fattore moltiplicativo è stata presa utilizzando, ancora una

volta, la tecnica del trial and error.

Dalle varie simulazioni effettuate, si è osservato che un limite dell'implementazione che è stata ipotizzata è che, per prevedere il comportamento del sistema, bisogna conoscere a priori il numero N di agenti che ne fanno parte. Questo non è sempre possibile nella realtà, e a seconda delle applicazioni il numero di agenti può essere altamente variabile. Per questo motivo si è cercato di trovare una soluzione che garantisse la convergenza (quasi certa) del sistema indipendentemente dal numero di agenti presenti. Una possibile soluzione è data dall'algoritmo 5.5.2, in cui è stato introdotto un ulteriore controllo sulla velocità massima degli agenti, grazie all'introduzione della variabile $count$ che conta gli aggiornamenti della posizione dell'agente. Ogni volta che il contatore raggiunge un determinato valore viene diminuita la velocità massima q_{max} che un agente può raggiungere.

5.5.1 Algoritmo adattativo a q_{max} costante

- **START** $q_{max} = 0.5$; $q = \frac{1}{10} q_{max}$;
- **IF** ***Gossip***
  - **IF** $|NewEstimate - OldEstimate| < \varepsilon_{min}$
    - **IF** $q < \frac{q_{max}}{2}$
      - $q = 1.5 \cdot q$
    - **ELSE**
      - $q = q_{max}$
  - **ELSE IF** $|NewEstimate - OldEstimate| > \varepsilon_{max}$
    - $q = \frac{1}{2} q$

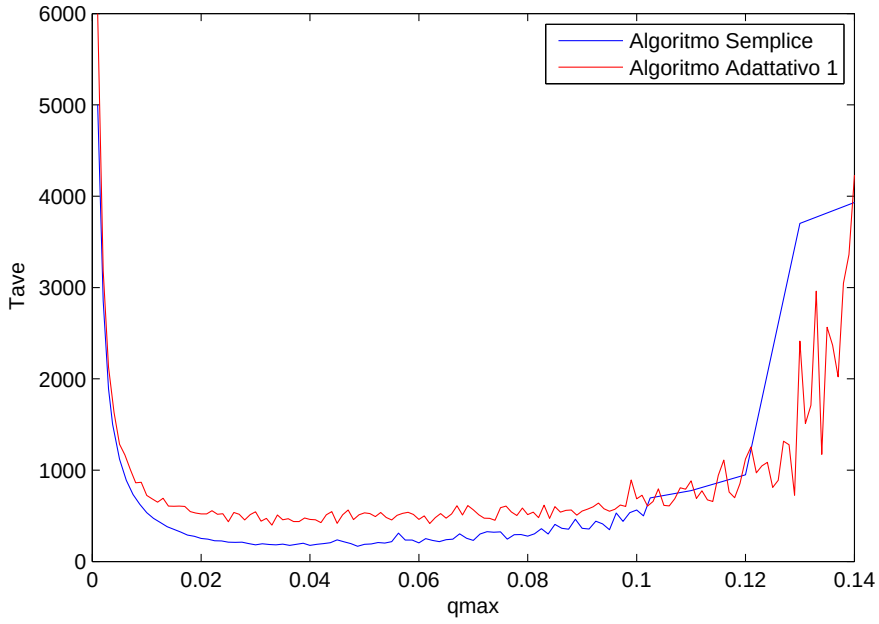
Dove:

- *OldEstimate* è la stima locale del centro del frame comune prima dell'aggiornamento.
- *NewEstimate* è la stima locale del centro del frame comune dopo l'aggiornamento.
- ε_{min} è la massima differenza, tra due aggiornamenti della stima, che da luogo ad un aumento di q.

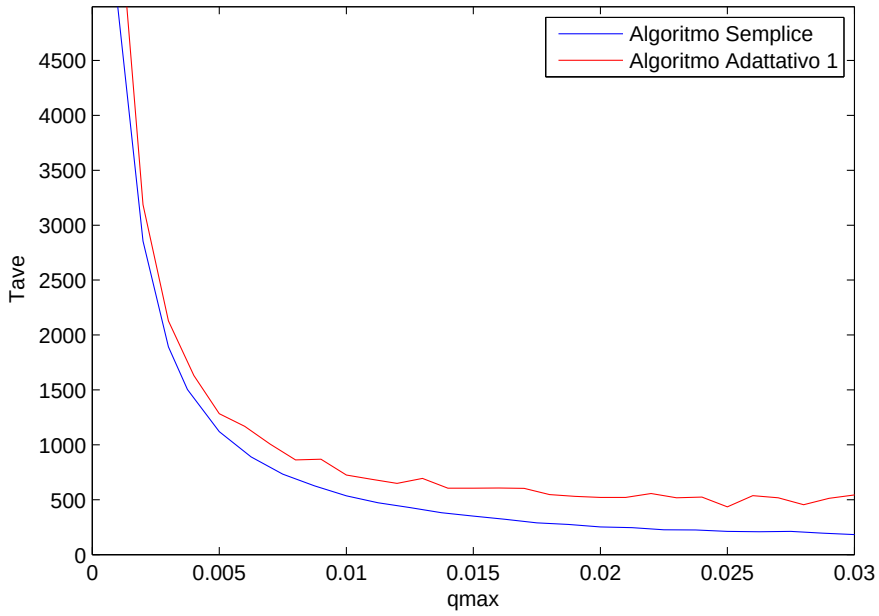
- ε_{max} è la minima differenza, tra due aggiornamenti della stima, che da luogo ad una diminuzione di q.

Il sistema su cui viene applicato questo algoritmo si comporta in questo modo:

- Ogni agente si muove con una dinamica indipendente dagli altri; ciascuno effettua un'azione di controllo sul parametro locale q a seconda di come varia la stima locale del frame comune tra due gossip successivi.
- La velocità massima q_{max}, che garantisce la convergenza (quasi certa) del sistema, diminuisce all'aumentare di N. Per impostare q_{max} bisogna conoscere a priori il numero degli agenti che fanno parte del sistema.
- Il valore minimo di t_{ave} è maggiore rispetto all'algoritmo con q fisso.
- La curva $q_{max} - t_{ave}$ ha lo stesso andamento dell'algoritmo a q fisso, come si può osservare in Figura 5.20. In questo caso i valori di q_{L1} e q_{L2}, che delimitano le diverse regioni di funzionamento del sistema, sono riferiti a q_{max}. In particolare si ha che:
  - Per $q_{max} < q_{L1}$ il tempo di convergenza t_{ave} del sistema tende a seguire quello dell'algoritmo per lo spostamento degli agenti descritto dall'equazione 5.1.
  - Per $q_{L1} \leq q_{max} < q_{L2}$ il tempo di convergenza tende a rimanere costante.
  - Per $q_{max} > q_{L2}$ il sistema converge lentamente e gli agenti non mantengono stabilmente le proprie posizioni.



(a)



(b)

Figura 5.20: Variazione di t_{abs} al variare di q_{max}.

5.5.2 Algoritmo adattativo a q_{max} variabile

- **START** $q_{max} = 0.5$; $q = \frac{1}{10} q_{max}$; $count = 0$;
- **IF** ***Gossip***
  - **IF** $|NewEstimate - OldEstimate| < \varepsilon_{min}$
    - **IF** $q < \frac{q_{max}}{2}$
      - $q = 1.5 \cdot q$
    - **ELSE**
      - $q = q_{max}$
  - **ELSE IF** $|NewEstimate - OldEstimate| > \varepsilon_{max}$
    - $q = \frac{1}{2} q$
- **IF** ***Evolution Step***
  - **IF** $count = count_{max}$
    - $q_{max} = 0.9 \cdot q_{max}$
    - $count = 0$
  - **ELSE**: $count = count + 1$
- *OldEstimate* è la stima locale del centro del frame comune prima dell'aggiornamento.
- *NewEstimate* è la stima locale del centro del frame comune dopo l'aggiornamento.
- ε_{min} è la massima differenza, tra due aggiornamenti della stima, che da luogo ad un aumento di q.
- ε_{max} è la minima differenza, tra due aggiornamenti della stima, che da luogo ad una diminuzione di q.
- $count$ è la variabile che conta il numero di aggiornamenti della posizione dell'agente, e controlla la velocità massima q_{max}.
- $count_{max}$ è il numero di aggiornamenti che determinano una diminuzione di q_{max}. Quando $count = count_{max}$ il contatore viene azzerato.

Il sistema su cui viene applicato questo algoritmo si comporta in questo modo:

- Ogni agente si muove con una dinamica indipendente dagli altri; ciascuno effettua un'azione di controllo sul parametro locale q a seconda di come varia la stima locale del frame comune tra due gossip successivi e a seconda del numero di aggiornamenti effettuati.
- La velocità massima $q_{max}(k)$ diminuisce all'aumentare di k.
- A parità di $q_{max}(0)$, il valore minimo di t_{ave} è maggiore rispetto ai due algoritmi precedenti.
- La curva $q_{max}(0) - t_{ave}$ converge quasi certamente per qualsiasi valore di $q_{max}(0)$, come si può osservare in Figura 5.22.
- Il sistema converge quasi certamente per ogni valore N, come si può osservare in figura 5.21.

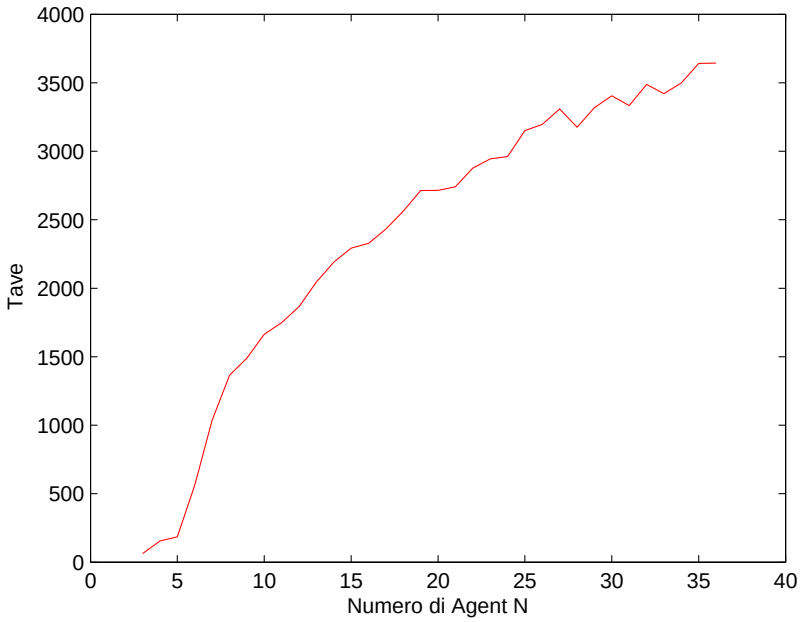
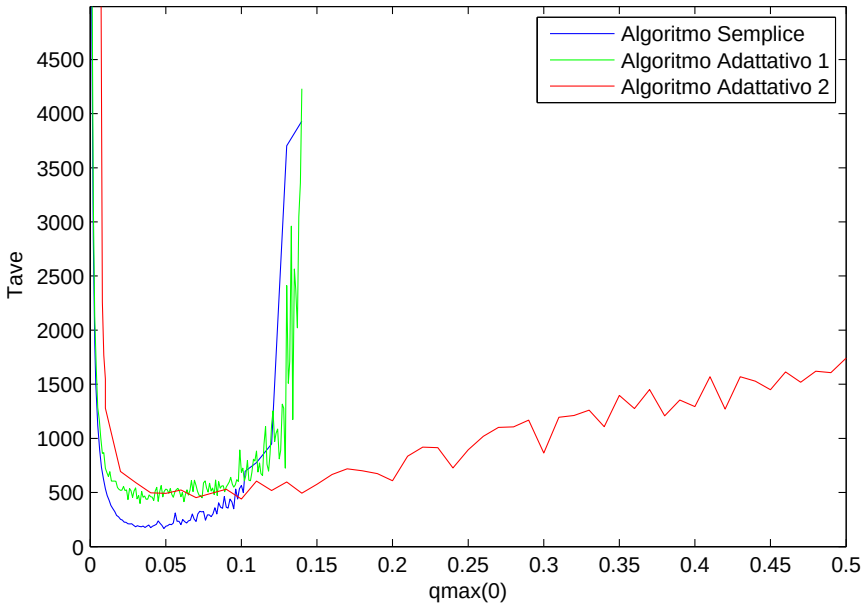
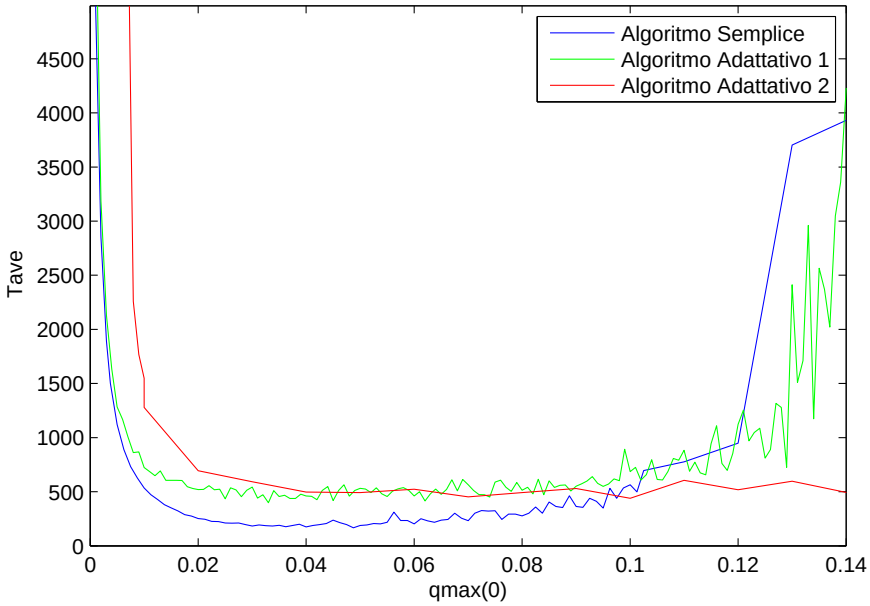


Figura 5.21: Variazione di t_{abs} al variare di N.



(a)



(b)

Figura 5.22: Variazione di t_{abs} al variare di $q_{max}(0)$.

5.6 Analisi dei risultati

Dai test che sono stati effettuati emergono chiaramente tutte le difficoltà di implementazione di un sistema come quello descritto nel Capitolo 3. La natura casuale del processo di selezione degli archi, nell'algoritmo di tipo gossip per il consenso su un frame comune, non da garanzie sul raggiungimento di un comportamento desiderato da parte degli agenti. Supponendo che siano verificate tutte le assunzioni preliminari che sono state introdotte, non si è stati in grado di determinare delle relazioni generali che permettano di prevedere, entro limiti accettabili, il tempo di convergenza degli agenti alle posizioni obiettivo.

Se lo spostamento è regolato dall'algoritmo a q costante, o dall'algoritmo adattativo a q_{max} costante descritto nella sezione 5.5.1, si può ricavare una procedura che permetta di prevedere t_{ave} solamente nel caso in cui il sistema operi nella regione che è stata definita *quasi controllabile*. Questa procedura è applicabile solo nel caso in cui:

- Si conosca a priori il numero di agenti N.
- Si conoscano i limiti delle regioni di funzionamento q_{L1} e q_{L2}.

Se queste ipotesi sono verificate allora fissato t_s, la procedura per far convergere gli agenti alla posizione obbiettivo nel tempo desiderato è la seguente:

- Si sceglie un tempo di convergenza desiderato in secondi t_d, a cui corrisponde un tempo di convergenza relativo desiderato: $t_{ave_d} = \lfloor \frac{t_d}{t_s} \rfloor$, in modo che il sistema funzioni in regione *quasi controllabile*:

$$t_{ave_d} \geq -\frac{10}{\ln(1-q_{L1})}$$

- Poichè $1 - e^{-\frac{7}{t_{ave_d}}} \leq q \leq 1 - e^{-\frac{10}{t_{ave_d}}}$, scegliamo:

$$q = 1 - e^{-\frac{7}{t_{ave_d}}}$$

  Deve risultare $q < q_{L1}$.

- Il sistema converge alle posizioni obbiettivo in maniera quasi certa nel tempo:

$$\left\{ -\frac{7}{\ln(1-q)} \leq t_{ave} \leq -\frac{10}{\ln(1-q)} \right\} \simeq t_{ave_d}$$

Capitolo 6

Conclusioni

In questa tesi sono state effettuate una serie di simulazioni con MATLAB, per valutare il comportamento di un sistema multi agente su cui vengono applicati degli algoritmi distribuiti, con l'obbiettivo di portare gli agenti a disporsi nello spazio secondo delle formazioni prestabilite. Poiché ogni agente interpreta ciò che lo circonda secondo un sistema di coordinate locali, è necessario che gli agenti si accordino su un sistema di riferimento comune prima di spostarsi verso la posizione che ciascuno di loro deve occupare nella formazione desiderata.

La prima modalita di implementazione a cui si pensa è quella in cui gli agenti svolgono in momenti diversi le operazioni di consenso sul frame comune e di spostamento. Ognuna di queste operazioni viene svolta mediante l'applicazione di un algoritmo, e gli effetti di una applicazione parallela di questi algoritmi non sono facilmente ipotizzabili e calcolabili.

Per ottenere il consenso su un frame comune è stato utilizzato un algoritmo randomizzato di tipo gossip, mentre lo spostamento degli agenti verso le posizioni obiettivo è stato controllato da un algoritmo deterministico caratterizzato da un parametro q che regola la velocità di convergenza.

I risultati delle simulazioni evidenziano che è possibile ottenere il comportamento desiderato dagli agenti sia applicando i due algoritmi in momenti diversi, sia applicandoli simultaneamente. Risulta molto interessante osservare come, dai risultati dei test, emerge che non c'è molta differenza di prestazioni, in termini di tempo di convergenza, tra la modalità di applicazione degli algoritmi in successione e la modalità di applicazione degli algoritmi in parallelo.

Il grosso vantaggio dell'applicazione parallela rispetto a quella in successione sta nella complessità di implementazione. Infatti, in caso di applicazione parallela degli algoritmi, è sufficiente implementare agenti molto semplici e in

grado di eseguire molte meno operazioni rispetto a quelle che dovrebbe eseguire un agente nel caso di applicazione degli algoritmi in successione.

Sono state testate tre diverse varianti dell'algoritmo per lo spostamento:

- algoritmo a q costante;
- algoritmo a q_{max} costante;
- algoritmo a q_{max} variabile.

Risulta interessante osservare i vantaggi e gli svantaggi che ogni algoritmo offre a seconda delle condizioni implementative. Dal confronto tra i risultati dei diversi test si possono fare le seguenti considerazioni:

- per applicazioni in cui si conosce a priori il numero N di agenti che fanno parte del sistema, le prestazioni migliori, in termini di $t_{ave_{min}}$ si ottengono dall'algoritmo a q costante. A parità di q gli algoritmi adattativi convergono più lentamente.
- per applicazioni in cui non è possibile stabilire a priori il numero di agenti N che fanno parte del sistema, le prestazioni migliori sono ottenute dall'algoritmo adattativo a q_{max} variabile descritto nella sezione 5.5.2, che è l'unico degli algoritmi presentati che garantisce la convergenza degli agenti alle posizioni desiderate per qualsiasi valore di N.

Negli algoritmi adattativi che sono stati simulati, i parametri di controllo ε_{min}, ε_{max} e $count_{max}$ sono stati scelti utilizzando la tecnica del trial and error. L'ottimizzazione di questi valori permetterebbe sicuramente di ottenere un miglioramento delle prestazioni in termini di t_{ave}, e la risoluzione di questo problema rimane aperto per eventuali sviluppi futuri.

Rimane, inoltre, ancora aperto il problema della determinazione di relazioni che permettano di calcolare numericamente il tempo di convergenza medio. Questo problema, a causa della natura random del processo di selezione degli archi nell'algoritmo di tipo gossip per la determinazione del frame comune, risulta particolarmente complicato da affrontare da un punto di vista matematico. Tuttavia le simulazioni effettuate risultano molto utili per comprendere come funziona il sistema che è stato ipotizzato, e mettono in evidenza in maniera qualitativa la dipendenza che esiste tra il tempo di convergenza t_{ave} e i vari parametri caratteristici del sistema.

A causa della potenza limitata degli elaboratori utilizzati si sono potuti simulare solo sistemi con un numero di agenti inferiore al centinaio. Pertanto non è possibile affermare con certezza che i risultati ottenuti siano validi anche per sistemi con un numero di elevato agenti.

Bibliografia

[1] M. Franceschelli, A. Gasparri. *On Agreement Problems With Gossip Algorithm in absence of Common Frame*. IEEE International Conference on Robotics and Automation, ICRA 2010, (Anchorage, Alaska, USA, 3-7 May 2010), pp. 4481 to 4486 [1, 24, 30]

[2] R. Olfati-Saber, J. Alex Fax,R. M. Murray. *Consensus and Cooperation in Networked Multi-Agent Systems* Proceedings of the IEEE, vol. 95, no. 1, pp. 215-233. (Jan. 2007). [4, 5, 18]

[3] R. Olfati-Saber. *Flocking for Multi-Agent Dynamic Systems: Algorithms and Theory*, IEEE Trans. on Automatic Control, vol. 51(3), pp. 401-420. (Mar. 2006). [5]

[4] S. Martinez, J. Cortez, F. Bullo. *Motion Coordination with Distributed Information* IEEE Control Systems Magazine, 27(4):75-88. (2007). [5]

[5] R. Olfati-Saber, R. M. Murray. *Consensus Problems in Networks of Agents with Switching Topology and Time-Delays* IEEE Trans. Autom. Control, vol. 49, no. 9, pp. 1520 to 1533. (Sep. 2004). [20]

[6] R. Olfati-Saber, R. M. Murray. *Consensus protocols for networks of dynamic agents* In Proc. Am. Control Conf., pp. 951 to 956, (2003). [4, 5]

[7] J. Cortes, S. Martinez, and F. Bullo. *Robust Rendezvous for Mobile Autonomous Agents via Proximity Graphs in Arbitrary Dimensions*. IEEE Transactions on Automatic Control, 51(8):1289-1298. (2006). [5]

[8] S.Boyd, A.Ghosh, B.Prabhakar, D.Shah. *Randomized Gossip Algorithms* IEEE Transactions on Information Theory, Special issue of IEEE Transactions on Information Theory and IEEE ACM Transactions on Networking, 52(6):2508-2530. (June 2006). [7, 21]

[9] S.Boyd, A.Ghosh, B.Prabhakar, D.Shah. *Gossip Algorithms: Design, Analysis and Applications* Proceedings IEEE Infocomm, 3:1653-1664, Miami. (March 2005). [7, 21]

[10] F. Fagnani, S. Zampieri. *Asymmetric Randomized Gossip Algorithms for Consensus*. In Proceedings of IFAC 2008 17th World Congress The International Federation of Automatic Control , pp.9052-9056. (Seoul 6-11 July, 2008). [7]

[11] F. Fagnani, S. Zampieri . *Randomized consensus algorithms over large scale networks* IEEE J. on Selected Areas of Communications, 26 pp. 634 to 649. (2008). [21]

[12] A. G. Dimakis, S. Kar, J. M. F. Moura, M. G. Rabbat, A. Scaglione *Gossip Algorithms for Distributed Signal Processing* Proceedings of the IEEE (2010) [21]

[13] M. Franceschelli, M. Egerstedt, A. Giua, and C. Mahulea. *Constrained Invariant Motions for Networked Multi-Agent Systems*. American Control Conference. (St. Louis, MO, USA, June 2009). [5]

Appendice A

Codice MATLAB

La classe Agent

```
%
%
%AUTHOR: Daniele Rosa
%Università degli studi di Cagliari
%2010
%
%

classdef Agent
%Un oggetto di Agent rappresenta un agente con una
%dinamica a tempo discreto.
properties(SetAccess = public)
id; % Identificatore Agente
ts; % Tempo di campionamento
initstate; % Posizione iniziale dell'agente
rotationMatrix; % Matrice di rotazione per
%passare dalle coordinate locali alle coordinate
%della rappresentazzione
state ; % Attributo che memorizza la posizione
%corrente dell'agente
evolution; % Evoluzione della posizione dell'Agente
localEstimateEvolution; % Evoluzione della stima del
%centro del frame comune in coordinate locali
```

```
currentEstimate; % Stima corrente del centro del
%frame comune espressa in coordinate locali
localEstimateEvolution2; % Evoluzione della stima
% del punto P2 espressa in coordinate locali
currentEstimate2; % Stima corrente di un punto P2
%nell'asse X del frame comune, espressa in coordinate locali.
comDir; % Versore stimato dell'asse X del frame comune
destination; % Punto di destinazione dell'agente, espresso in
%coordinate globali(in base al frame stimato)
qmax;%Parametro di controllo della velocità max.
q;%Parametro di controllo della velocità
count;%contatore dei cicli di clock

end

methods
%Costruttore
function obj=Agent(id,x1,x2,t,s1,s2,d1,d2,o1,o2,a,q)
obj.id=id;
obj.initstate=[x1,x2];
obj.state=[x1,x2];
obj.ts=t;
obj.evolution=obj.initstate(1,1:2);
obj.currentEstimate=[s1,s2];
obj.localEstimateEvolution=[s1,s2];
obj.currentEstimate2=[d1,d2];
obj.localEstimateEvolution2=[d1,d2];
b=sqrt(1-(a^2));
obj.rotationMatrix=[a,-b;b,a];
obj.comDir=([d1,d2]-[s1,s2])/norm([d1,d2]-[s1,s2]);
obj.destination=[o1,o2];
obj.qmax=q;
obj.q=(q/10);
obj.count=0;
end
```

```
 function plotdinamica(obj)
 plot(obj.evolution(:,1),obj.evolution(:,2));
 end

function obj=EvolutionStep(obj)
 % Metodo che determina la posizione all'istante di
%campionamento
% successivo secondo la formula
%P(k+1)=P(k)+ts*[(Rg*destination)+currentEstimate]
step=(obj.q)*(([obj.comDir(1,1),-obj.comDir(1,2);
obj.comDir(1,2),obj.comDir(1,1)]*(obj.destination'))+
(obj.currentEstimate'));
newPosition=obj.state+((obj.rotationMatrix*step)');
obj.state=newPosition;
obj.evolution=[obj.evolution;newPosition];
if obj.count == 100
obj.qmax = obj.qmax*0.9;
obj.count=0;
else obj.count=obj.count+1;
end
end
function obj=EvolutionStep2(obj)
     % Metodo che determina la posizione all'istante di
     %campionamento successivo secondo la formula
     %P(k+1)=P(k)+ts*[(Rg*destination)+currentEstimate]
     step=(obj.q)*obj.state;
     newPosition=obj.state-step;
     obj.state=newPosition;
     obj.evolution=[obj.evolution;newPosition];
end

function obj=EstimateStep(obj,sj1,sj2,dj1,dj2,d,cijx,cijy,cjix,cjiy)
% Metodo che, quando viene invocato, calcola la stima successive
% dei punti P1 e P2 in base alla distanza dall'agente con cui
%sta comunicando
% e alle informazioni che riceve da esso.
```

```
cpijx=-cijy;
cpijy=cijx;
cpjix=-cjiy;
cpjiy=cjix;
cij=[cijx;cijy];
cji=[cjix;cjiy];
cpij=[cpijx;cpijy];
cpji=[cpjix;cpjiy];
del=(d-([sj1,sj2]*cji)+(obj.currentEstimate*cij))/2;
delp=((-([sj1,sj2]*cpji))+(obj.currentEstimate*cpij))/2;
del2=(d-([dj1,dj2]*cji)+(obj.currentEstimate2*cij))/2;
delp2=((-([dj1,dj2]*cpji))+(obj.currentEstimate2*cpij))/2;
sx=((del*cijx)+(delp*cpijx));
sy=((del*cijy)+(delp*cpijy));
dx=((del2*cijx)+(delp2*cpijx));
dy=((del2*cijy)+(delp2*cpijy));
newEstimate=[sx sy];
if (max(abs(newEstimate-obj.currentEstimate))) < 0.05
if obj.q < (obj.qmax/2)
obj.q=1.5*obj.q;
else obj.q=obj.qmax;
end
elseif (max(abs(newEstimate-obj.currentEstimate))) > 0.1
obj.q=obj.q/2;
end
obj.currentEstimate=[sx sy];
obj.localEstimateEvolution=[obj.localEstimateEvolution;
obj.currentEstimate];
obj.currentEstimate2=[dx dy];
obj.localEstimateEvolution2=[obj.localEstimateEvolution2;
obj.currentEstimate2];
obj.comDir=([dx,dy]-[sx,sy])/norm([dx,dy]-[sx,sy]);
end

end
end
```

ArrayEvolutionStep.m

```
function Array=ArrayEvolutionStep(Array,radius)
[i j]=SelectCouple(Array,radius);
P=PositionMatrix(Array);
d=CalculateDistance(Array(1,i),Array(1,j));
Pi=P(i,:);
Pj=P(j,:);
cij=((Array(1,i).rotationMatrix)')*(((Pj-Pi)')/d);
cji=((Array(1,j).rotationMatrix)')*(((Pi-Pj)')/d);
cijx=cij(1,1);
cijy=cij(2,1);
cjix=cji(1,1);
cjiy=cji(2,1);
si1=Array(1,i).currentEstimate(1,1);
si2=Array(1,i).currentEstimate(1,2);
sj1=Array(1,j).currentEstimate(1,1);
sj2=Array(1,j).currentEstimate(1,2);
di1=Array(1,i).currentEstimate2(1,1);
di2=Array(1,i).currentEstimate2(1,2);
dj1=Array(1,j).currentEstimate2(1,1);
dj2=Array(1,j).currentEstimate2(1,2);

for n=1:size(Array,2)

if n==i
Array(1,n)=EstimateStep(Array(1,n),sj1,sj2,dj1,dj2,d,cijx,cijy,cjix,cjiy);
Array(1,n)=EvolutionStep(Array(1,n));
elseif n==j
Array(1,n)=EstimateStep(Array(1,n),si1,si2,di1,di2,d,cjix,cjiy,cijx,cijy);
Array(1,n)=EvolutionStep(Array(1,n));
else
Array(1,n)=EvolutionStep(Array(1,n));
end
end
end
```

CalculateDistance

```
function d = CalculateDistance(obj1,obj2)
     p1=ExtractPosition(obj1);
     p2=ExtractPosition(obj2);
     x1=p1(1,1);
     y1=p1(1,2);
     x2=p2(1,1);
     y2=p2(1,2);
     a = (x1-x2)^2;
     b = (y1-y2)^2;
     d = sqrt(a+b);
end
```

DistanceMatrix.m

```
function d = DistanceMatrix(Array)
for i=1:size(Array,2)
    for j = 1:size(Array,2)
        d(i,j)=CalculateDistance(Array(1,i),Array(1,j));

    end
end
end
```

InizializzazioneRandom.m

```
function Array=InizializzazioneRandom(n,t)
Array=[];
DestTab1=[0,2;0,-2;2,0;-2,0;0,1;1,0;-1,0;0,-1];
DestTab2=1.5*DestTab1;
DestTab=[DestTab1;DestTab2];
DestTab3=1.2*DestTab;
DestTab=[DestTab;DestTab3];
for i=1:n
    x1=5*rand(1);
    x2=5*rand(1);
    s1=rand(1);
```

```
    s2=rand(1);
    d1=rand(1);
    d2=rand(1);
    a=rand(1)-rand(1);
    o1=DestTab(i,1);
    o2=DestTab(i,2);
    temp=Agent(i,x1,x2,t,0,0,d1,d2,o1,o2,a,0.05);
    Array=[Array,temp];
end
end
```

PlotArray.m

```
function Array=PlotArray(Array,radius)
M=PositionEstimationMatrix(Array);
N=PositionEstimationMatrix2(Array);
gplot(AdiacenceMatrix(Array,radius),PositionMatrix(Array),'-o');
axis ([-1 5 -1 5]);
axis manual;
hold;
pause(0.05)
for i=1:size(Array,2)
    n=2*i;
    m=n-1;
    plot(M(m:n,1),M(m:n,2),'-xr');
    %plot(N(m:n,1),N(m:n,2),'-xg');
    p1=M(n,:);
    pi=p1+((Array(1,i).rotationMatrix)*((Array(1,i).comDir)'))';
    p=[p1;pi];
    plot(p(:,1),p(:,2),'-*g');
    dir=(Array(1,i).comDir);
    dirp=[-dir(1,2);dir(1,1)];
    pip=p1+((Array(1,i).rotationMatrix)*(dirp))';
    pp=[p1;pip];
    plot(pp(:,1),pp(:,2),'-*g');
end
pause(0.05)
```

```
hold off;

end
```

CreateFileWrl.m

```
function world=CreateFileWrl(N)
%Crea un file WRL che rappresenta un mondo virtuale con un numero di
%palline pari al numero degli Agenti.

if exist('newWorld.WRL')==2
    delete('newWorld.WRL');
    world=fopen('newWorld.WRL','wt');
else
    world=fopen('newWorld.WRL','wt');
end

%Definizioni iniziali
fprintf(world,'%s\n','#VRML V2.0 utf8');
fprintf(world,'%s\n','');
fprintf(world,'%s\n','#Created with V-Realm Builder v2.0');
fprintf(world,'%s\n','#Integrated Data Systems Inc.');
fprintf(world,'%s\n','#www.ids-net.com');
fprintf(world,'%s\n','');
fprintf(world,'%s\n','');


%Definizione del ViewPoint
fprintf(world,'%s\n','Viewpoint {');
fprintf(world,'%s\n','fieldOfView 0.8');
fprintf(world,'%s\n','orientation 0.400807 -0.38601 -0.830873 1.74926');
fprintf(world,'%s\n','position -6.75686 2.39796 7.05314');
fprintf(world,'%s\n','description ""');
fprintf(world,'%s\n','}');

%Definizione del BackGround
```

```
fprintf(world,'%s\n','Background {');
fprintf(world,'%s\n','groundAngle [ 0.9, 1.5, 1.57 ]');
fprintf(world,'%s\n','groundColor [ 0 0 0,');
fprintf(world,'%s\n','  0 0 0,');
fprintf(world,'%s\n','  0 0 0,');
fprintf(world,'%s\n','  0 0 0 ]');
fprintf(world,'%s\n','skyAngle [ 0.1, 1.2, 1.57 ]');
fprintf(world,'%s\n','skyColor [ 0 0 0,');
fprintf(world,'%s\n','  0 0 0,');
fprintf(world,'%s\n','  0 0 0,');
fprintf(world,'%s\n','  0 0 0 ]');
fprintf(world,'%s\n','}');

%Navigation Info
fprintf(world,'%s\n','NavigationInfo {');
fprintf(world,'%s\n','visibilityLimit 0');
fprintf(world,'%s\n','}');

%Definizione Piano
fprintf(world,'%s\n','DEF Piano Transform {');
fprintf(world,'%s\n','bboxCenter 3 3 0');
fprintf(world,'%s\n','translation 0 0 0');
fprintf(world,'%s\n','scale 0.629075 0.629075 0.629074');
fprintf(world,'%s\n','center 7.5 7.5 -0.5');
fprintf(world,'%s\n','children Shape {');
fprintf(world,'%s\n','appearance Appearance {');
fprintf(world,'%s\n','material Material {');
fprintf(world,'%s\n','ambientIntensity 0.2');
fprintf(world,'%s\n','diffuseColor 0.256075 0.8 0.23044');
fprintf(world,'%s\n','shininess 0.2');
fprintf(world,'%s\n','transparency 0');
fprintf(world,'%s\n','}');
fprintf(world,'%s\n','');
fprintf(world,'%s\n','}');
fprintf(world,'%s\n','');
fprintf(world,'%s\n','geometry Box {');
```

```
fprintf(world,'%s\n','size 100 100 0.5');
fprintf(world,'%s\n','}');
fprintf(world,'%s\n','');
fprintf(world,'%s\n','}');
fprintf(world,'%s\n','}');



%Definzione degli agenti
for i=1:N
    x=rand(1);
    y=rand(1);
    z=rand(1);
    fprintf(world,'%s','DEF Sfera');
    fprintf(world,'%i',i);
    fprintf(world,'%s\n',' Transform {');
    fprintf(world,'%s\n','translation 0 0 0');
    fprintf(world,'%s\n','rotation -0.0041578 -0.994639 0.103323  0.0808658');
    fprintf(world,'%s\n','scale 0.0459137 0.0459138 0.0459138');
    fprintf(world,'%s\n','children Shape {');
    fprintf(world,'%s\n','appearance Appearance {');
    fprintf(world,'%s\n','material Material {');
    fprintf(world,'%s','diffuseColor ');
    fprintf(world,'%i',x);
    fprintf(world,'%s',' ');
    fprintf(world,'%i',y);
    fprintf(world,'%s',' ');
    fprintf(world,'%i\n',z);
    fprintf(world,'%s\n','}');
    fprintf(world,'%s\n','');
    fprintf(world,'%s\n','}');
    fprintf(world,'%s\n','');
    fprintf(world,'%s\n','geometry Sphere {');
    fprintf(world,'%s\n','radius 2.5');
    fprintf(world,'%s\n','}');
    fprintf(world,'%s\n','');
```

```
    fprintf(world,'%s\n','}');
    fprintf(world,'%s\n','}');
end

world=fclose('all');
end
```

CreateEvolutionTab.m

```
function tab = CreateEvolutionTab(Array)
%Dato in ingresso un array di Agent e crea una matrice con tutte le
%evoluzioni degli agenti.
tf=0.05;
ts=Array(1,1).ts;
r=tf/ts;
temp2=[];
n=size(Array,2);
m=size(Array(1,1).evolution,1);
for i=1:n
    temp=Array(1,i).evolution';
    temp=[temp;zeros(1,m)];
    temp2=[temp2;temp];
end
tab1 = [0:ts:(ts*(size(temp2,2)-1))];
tab1=[tab1;temp2];
col=fix(size(tab1,2)/r)-1;
tab=tab1(:,1);
for j=1:col
    k=1+(r*j);
    tempcol=tab1(:,k);
    tab=[tab,tempcol];
end
```

VirtualEvolution.m

```
t=0.05;
view(world)
```

```
f=get(world,'Figures');
set(f,'Record2DFPS',1/t);
pause(1);
set(world,'RecordMode','manual');
set(f,'Record2D','on');
set(f,'NavPanel','none');
set(world,'Recording','on');
Tev=0;
for i=1:size(tab,2)
    for j=1:(size(nodes,1)-1)
        nodes(j).translation=[tab((3*j)-1,i),tab((3*j),i),tab((3*j)+1,i)];
    end
    Tev=Tev+t;
    set(world,'Time',Tev);
    pause(t);
end

set(world,'Recording','off');
set(f,'NavPanel','halfbar');
clear f;
```