ALMA MATER STUDIORUM

–

UNIVERSITA' DI BOLOGNA

CAMPUS DI CESENA

SCUOLA DI INGEGNERIA E ARCHITETTURA

CORSO DI LAUREA IN

INGEGNERIA ELETTRONICA PER L'ENERGIA E L'INFORMAZIONE

TITOLO DELL'ELABORATO

ANALISI E IMPLEMENTAZIONE DI ALGORITMI PER IL SELF-POSITIONING

Elaborato in

COMUNICAZIONI DIGITALI ED INTERNET

Relatore

Chiar.mo Prof. Davide Dardari

Presentato da

Giacomo Giorgini

Anno Accademico 2017/2018

INDICE

Capitolo 1

- Introduzione

1.1 Problematiche relative al GPS

Nell'ambito della geolocalizzazione ha preso largo uso il GPS (sistema di posizionamento globale) in cui un oggetto, dotato dell'apposito ricevitore, può essere localizzato univocamente sulla superficie terrestre. Questa tecnologia, sviluppata dal Dipartimento della Difesa statunitense a partire dal 1973, sfrutta il metodo della trilaterazione, in cui tramite misure di distanza effettuate da appositi satelliti posizionati in orbita, è possibile ricavare univocamente la posizione dell'oggetto in considerazione sfruttando proprietà geometriche.

Per alcune applicazioni però il GPS risulta inadeguato; questo succede quando si vuole localizzare un oggetto in un ambiente indoor oppure quando si vuole assicurare un errore di stima della posizione relativamente piccolo. Infatti, il GPS riesce a determinare la posizione globale di un terminale con un errore che varia dai 3 ai 30 metri.

Inoltre, negli ambienti indoor, ogni ostacolo (come può essere un muro) che si interpone nella tratta tra satellite e terminale crea un'attenuazione che può ridurre in modo drastico la potenza del segnale in considerazione, superando anche il limite della sensibilità del ricevitore.

Senza considerare inoltre che questi terminali prevedono hardware complessi e costosi in termini di consumo energetico [7], è chiaro allora che per certe applicazioni in cui è necessaria un'accurata stima della posizione di oggetti in un dato ambiente non si può fare affidamento sulla tecnologia relativa al GPS.

1.2 Real-time Locating Systems (RTLS)

Grazie soprattutto allo sviluppo tecnologico che ha permesso la nascita di componenti hardware sempre più piccoli ed efficienti e relativamente poco costosi, per superare il problema del posizionamento in ambienti indoor sono state prese in esame i sistemi di posizionamento in tempo reale terrestri denominati comunemente RTLS, con i quali è possibile localizzare un oggetto, con un certo errore di stima e a patto che esso si trovi all'interno dell'area coperta da tale sistema.

Consideriamo quindi una rete di RTLS; teoricamente, essa è costituita da un certo numero di nodi. Il terminale mobile da localizzare interagisce con i nodi ancora, ad esempio stimando la distanza attraverso misure di potenza ricevuta (received signal strength - RSS) oppure misure di tempo di propagazione del segnale radio (Time-of-arrival - TOA).
Ciascuna misura di distanza (ranging) in condizioni ideali determina una circonferenza centrata sul nodo ancora considerato. Se si combinano almeno 3 misure, la posizione del terminale mobile è univocamente determinata dall'intersezione di 3 circonferenze.

Un altro approccio consiste nel misurare la differenza dei tempi di arrivo dei segnali al terminale mobile (time-difference of arrival - TDoA) emessi da due nodi ancora. In questo caso le misure TDoA definiscono un'iperbole. La posizione del terminale mobile è stimata mediante l'intersezione di più iperboli.

In presenza di errori di misura la stima della posizione richiede tecniche più sofisticate come il metodo least-square (LS), [10].

1.3 Struttura di una RTLS

Consideriamo una rete di dispositivi RTLS; essa può essere interpretata come un grafo in cui ogni nodo rappresenta un dispositivo ed ogni collegamento rappresenta idealmente la possibilità o meno che hanno due dispositivi di comunicare tra di loro. Se due dispositivi riescono a scambiarsi informazioni sono chiamati “nodi vicini”.

Come nel caso dei satelliti del GPS, anche in una rete di RTLS è necessario innanzitutto conoscere le coordinate che costituiscono l’infrastruttura, sulla base delle quali è possibile in un secondo momento determinare la posizione di un oggetto presente all’interno dell’area coperta dalla rete.

Considerando di non poter sfruttare la tecnologia GPS e di non voler stimare la posizione dei nodi della rete manualmente, è necessario introdurre un metodo di autolocalizzazione, grazie al quale i nodi del grafo possono autolocalizzarsi (o in modo assoluto o in modo relativo).

Una delle sfide più importanti e interessanti nel campo della geolocalizzazione è proprio il “self-positioning”, ossia la capacità della rete di autolocalizzare i nodi di cui è costituita.

In questo modo viene automatizzata la prima fase di cui ogni sistema di posizionamento necessita, cioè la scoperta dei cosiddetti “nodi ancora”, le cui coordinate vengono considerate per determinare poi la posizione di ulteriori oggetti all’interno della rete.

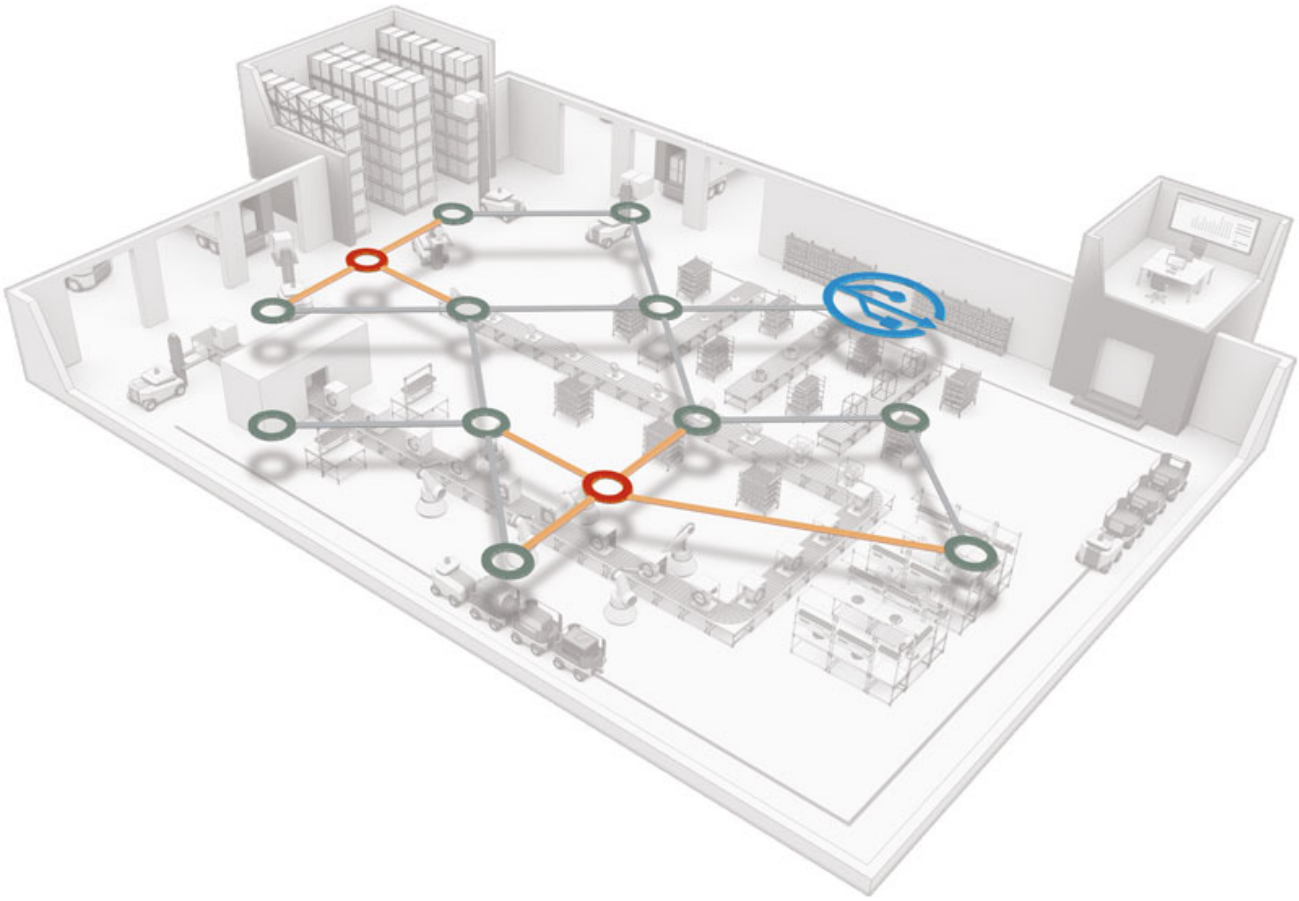


Figura 1 - Rappresentazione di una possibile rete di nodi installata al fine di monitorare l'ambiente circostante

CAPITOLO 2

- Algoritmi di self-positioning

2.1 Caratteristiche degli algoritmi di self-positioning

Il processo di localizzazione è costituito da due fasi: *ranging* e *positioning*. Durante il ranging, ogni nodo scopre i suoi vicini ed effettua misure di distanza al fine di ottenere le informazioni necessarie per determinare, nella successiva fase del positioning, le proprie coordinate all'interno della rete. Gli algoritmi che sono stati elaborati per risolvere il problema del self-positioning possono essere molto diversi tra loro, differenziandosi sia per l'approccio seguito nella fase di ranging, sia per quello seguito nella fase di positioning [3],[6].

Una prima grande distinzione che viene fatta è la seguente:

- algoritmi *centralizzati*, in cui i nodi inviano le informazioni necessarie ad un'entità centrale che, elaborando i dati, crea un immagine topologica della rete;
- algoritmi *distribuiti*, in cui ogni nodo è in grado di calcolare la propria posizione relativa all'interno della rete.

Considerando che ogni algoritmo distribuito può essere implementato in modo centralizzato, nella trattazione che segue si è scelto di considerare ogni algoritmo come se fosse distribuito.

2.2 Tecniche di misurazione principali

Dato che a seconda del tipo di misura disponibile si possono implementare algoritmi di tipo diverso, possiamo subito fare un'altra distinzione, basata sulle tecniche di misurazione più comuni [1].

Tra le tecniche più usate, troviamo:

- *RSSI* (received signal strength indication), in cui viene misurata la potenza tra il segnale inviato e il segnale ricevuto al fine di valutare la distanza che intercorre tra trasmettitore e ricevitore;
- *ToA* (time of arrival), in cui viene misurato il tempo di arrivo del segnale. Conoscendo l'istante di tempo in cui il segnale è stato trasmesso, si converte facilmente l'informazione temporale in una misura di distanza, sapendo che i segnali viaggiano alla velocità della luce;
- *TDoA* (time difference of arrival), in cui un trasmettitore invia un segnale a due diversi ricevitori e ne misura la differenza tra i tempi di arrivo. Anche in questo caso, sulla base della stima temporale è possibile ricavarne un'informazione di distanza;
- *AoA* (angle of arrival), in cui viene misurato l'angolo di arrivo del segnale tra trasmettitore e ricevitore.

Esistono anche algoritmi, chiamati *Range-Free*, basati non sulla misura geometrica tra un nodo e un suo vicino, bensì sul numero di *hop*, ossia il numero di salti che un segnale, inviato da un nodo sorgente, deve compiere prima di arrivare al nodo destinatario (se tali nodi sono vicini, l'hop-count è pari ad uno). In questo tipo di algoritmi, la distanza tra un nodo ed un altro viene stimata ipotizzando un valore di distanza che verrà poi moltiplicato per gli hop-count che separano tali nodi. E' ragionevole pensare quindi che questa classe di algoritmi può essere adottata soltanto nel caso di

topologie dotate di una certa uniformità per quanto riguarda la distruibuzione dei sensori.

Dato che il posizionamento di un nodo è strettamente collegato alla fase di ranging [2], è intuitivo pensare che, in generale, più la misura è precisa, più la posizione stimata è accurata. Il vantaggio di adottare algoritmi di tipo Range-Free può essere dato dalla sempicità implementativa e dal basso costo computazionale, che porta a ottenere risultati fruibili in un tempo relativamente breve.

2.3 Limiti

In realtà, è impossibile stabilire a priori se un algoritmo è meglio di un altro poichè ciascuno di essi viene sviluppato per essere implementato in una determinata applicazione, sulla base di determinate ipotesi. Quello che si può fare è, stabilita una determinata applicazione e definita la tecnologia dei sensori che costituiscono la rete, determinare quale algoritmo è utile implementare al fine di ottenere l'accuratezza desiderata, considerando anche i costi computazionali (energia e tempo). Viceversa, stabilita a priori l'accuratezza che si vuole ottenere e fissati i costi che si è disposti a pagare, è possibile valutare quali dispositivi impiegare nella rete. Va tenuto però in considerazione il fatto che in ogni caso esistono dei limiti, legati sia alla tecnologia sia teorici.

Da un punto di vista tecnologico, si possono elencare i principali limiti contro i quali ogni sistema di localizzazione si deve scontrare:

- *raggio di copertura* di un trasmettitore: infatti, non si può pensare che un nodo possa coprire un'area infinita, bensì a seconda di come è stato costruito è possibile definire un raggio di copertura al di là del quale il segnale risulta troppo debole;
- *sensibilità del ricevitore*: è necessario tenere in considerazione una soglia per quanto riguarda la potenza minima necessaria al ricevitore per funzionare;
- *canale di comunicazione non ideale*: ogni ostacolo presente all'interno della rete può attenuare i segnali in gioco. E' quindi necessario tenere in considerazione il path loss nell'equazione di Friis, la quale stabilisce il legame tra potenza inviata e potenza ricevuta.

Da un punto di vista teorico, la difficoltà di implementazione di algoritmi volti al self-positioning è data soprattutto dal fatto che questi ricadono nella classe dei problemi *NP-hard [7]* (a differenza dei problemi di

localizzazione classica dove la posizione dei nodi ancora è nota e l'unica incognita è la posizione del terminale mobile. Questa informazione ha un notevole impatto nello sviluppo di algoritmi di posizionamento poichè dimostrare che un problema è *NP-hard* non può essere risolto con algoritmi a complessità (e quindi tempo di esecuzione) polinomiale con il numero dei nodi. Infatti, in un problema *NP-hard*, l'algoritmo che vuole ricercare la soluzione ottima (algoritmo esaustivo) prevede di tentare ogni possibile combinazione il cui numero è infinito.

Essendo stato dimostrato in passato che i problemi di self-positioning ricadono proprio nella classe *NP-hard*, sono stati sviluppati diversi algoritmi sub-ottimi grazie ai quali si possono ottenere risultati soddisfacenti, dimenticandosi la ricerca dell'ottimo.

Ovviamente, il risultato ottenuto si può ritenere soddisfacente o meno sulla base dell'applicazione in questione, quindi sull'accuratezza che si vuole ricercare affinchè i dati delle soluzioni ottenute possano essere utilizzati con efficacia.

2.4 Definizione di un problema di self-positioning

Vogliamo ora trovare una possibile soluzione al seguente problema:

"Dato un set di nodi con posizione sconosciuta e un meccanismo tale per cui ogni nodo possa stimare la distanza tra esso e i vicini, determinare le coordinate di posizione di ogni nodo tramite comunicazione locale."

Definendo il problema da un punto di vista matematico e modellando la rete come un grafo, consideriamo *N* nodi distribuiti in una certa regione di spazio. Se è disponibile una comunicazione, che assumiamo essere bilaterale, tra un generico nodo *i* ed un generico nodo *j*, al collegamento tra quest'ultimi viene attribuito un arco del grafo. L'obiettivo è ricavare le coordinate dei nodi del grafo così ottenuto.

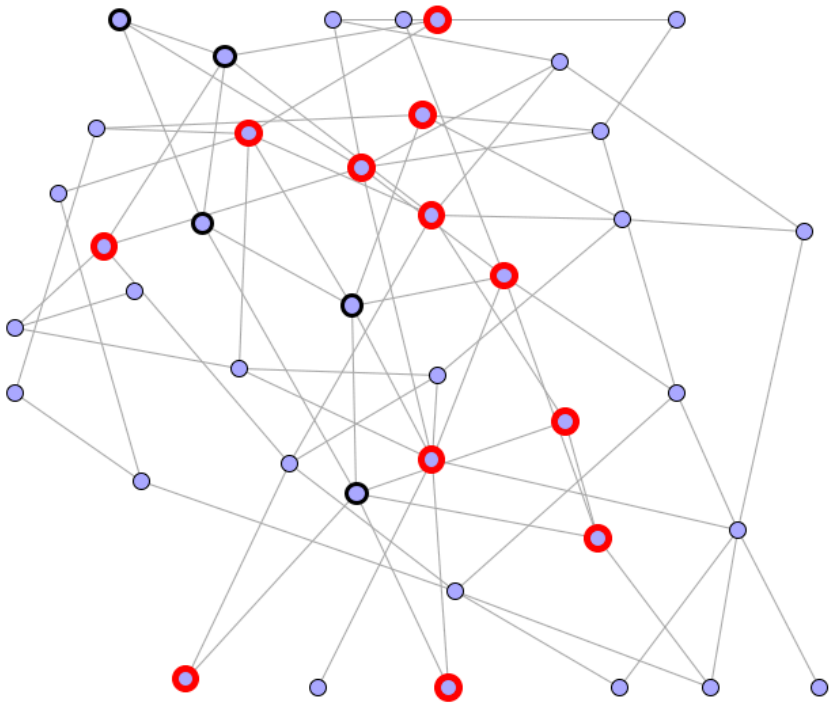


Figura 2 - Rappresentazione teorica mediante grafo di una rete di nodi

2.5 Requisiti geometrici della rete

Avendo definito la rete come un grafo, è allora necessario notare che se si vuole ottenere una soluzione unica per ogni nodo il grafo deve soddisfare il requisito della rigidità globale, condizione più restrittiva della semplice rigidità.

Infatti, se il grafo è solamente rigido, nel senso che non può essere flessibile preservando le distanze (come invece succede nel caso di un rettangolo), può comunque essere soggetto a *local flips* [4]. Per esempio, se consideriamo due triangoli che condividono un lato, un triangolo può essere specchiato attraverso quel lato senza che varino le distanze.

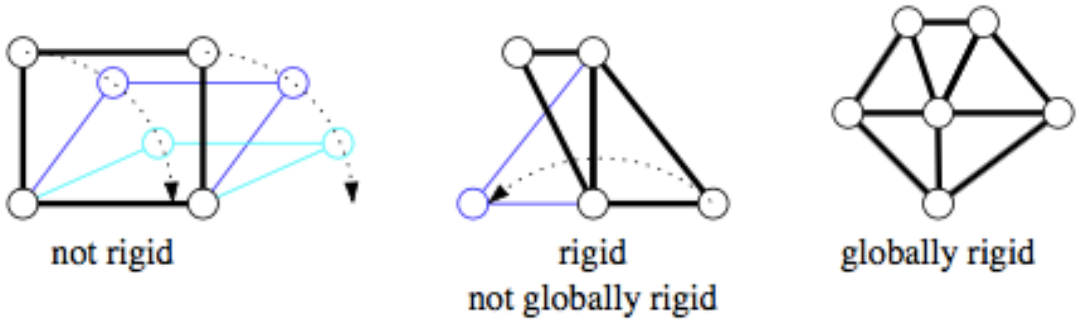


Figura 3 - Esempio esplicativo dei concetti di rigidità e rigidità globale

Generalmente, per garantire la rigidità globale è sufficiente che la connettività all'interno del grafo sia alta; in altri termini, ogni nodo deve avere un numero sufficientemente elevato di vicini (solitamente, un numero maggiore o uguale a 3, nel caso bidimensionale).

Si può quindi affermare che la buona riuscita di un algoritmo di self-positioning sia determinata anche dalla topologia della rete; infatti, per certe configurazioni di nodi, è possibile che le soluzioni cadano in zone di minimo locale della funzione che cerca di minimizzare la distanza tra la posizione reale e quella stimata; queste zone, se molto distanti dal minimo assoluto, portano ad un risultato completamente scorretto.

Sulla base di tutte queste considerazioni, in questo lavoro si è voluto studiare un particolare algoritmo di self-positioning: l'AFL [2].

Capitolo 3

- Anchor-Free Localization

3.1 Descrizione dell'algoritmo

L'AFL (Anchor-Free Localization) è un algoritmo di tipo decentralizzato in cui ogni nodo, tramite misure di distanza, riesce a stimare la propria posizione all'interno della rete.

L'algoritmo consiste in due fasi fondamentali: all'inizio, viene prodotta una prima immagine del grafo "simile" a quello che si vuole ottenere, mentre nella seconda fase si cerca di aggiustare le coordinate ottenute nella prima in modo da minimizzare il più possibile gli errori di distanza tra posizione reale e stima.

Passiamo ora ad analizzare la prima fase per capire in che modo sia possibile ottenere un'immagine simile alla topologia della rete in considerazione, limitandoci al caso bidimensionale. Partendo dal presupposto che ogni sensore abbia la tecnologia adeguata per effettuare misure di distanza (ad esempio sfruttando il RSSI), vengono scelti 5 nodi di riferimento in modo tale da creare un sistema di riferimento locale. A tal proposito, verranno selezionate due coppie di nodi, (n_1 , n_2) e (n_3 , n_4), in modo tale che siano il più possibile perpendicolari tra di loro, ed un nodo centrale in modo che risulti essere circa al centro del grafo. Prima di definire gli step necessari per eleggere i nodi di riferimento, definiamo $h_{i,j}$ come la distanza, in termini di hop-count, tra il nodo i-esimo e il nodo j-esimo.

3.2 Fase iniziale

Vediamo ora il procedimento in cui consiste la prima fase:

- Step 1. Selezionare un nodo n_0 in modo arbitrario, ad esempio scegliendo quello con ID minore. Successivamente, selezionare un ulteriore nodo, n_1, in modo tale che venga massimizzata la distanza, in termini di hop, tra n_0 e n_1.
- Step 2. Selezionare il nodo n_2 in modo da massimizzare gli hop-count da n_1.
- Step 3. Selezionare il nodo di riferimento n_3 per minimizzare $|h_{1,3} - h_{2,3}|$. Spesso capita che più di un nodo soddisfi questo requisito; quindi selezionare quello che massimizza $|h_{1,3} + h_{2,3}|$, tra i possibili contendenti.
- Step 4. Come nel precedente step, selezionare n_4 affinchè sia minima $|h_{1,4} - h_{2,4}|$. Tra i vari contendenti, scegliere quello che massimizza $h_{3,4}$.
- Step 5. Come nel precedente step, selezionare n5 minimizzando $|h_{1,5} - h_{2,5}|$. Tra i possibili contendenti, scegliere quello che minimizza $|h_{3,5} - h_{4,5}|$. Questo step seleziona il nodo centrale del sistema di riferimento.

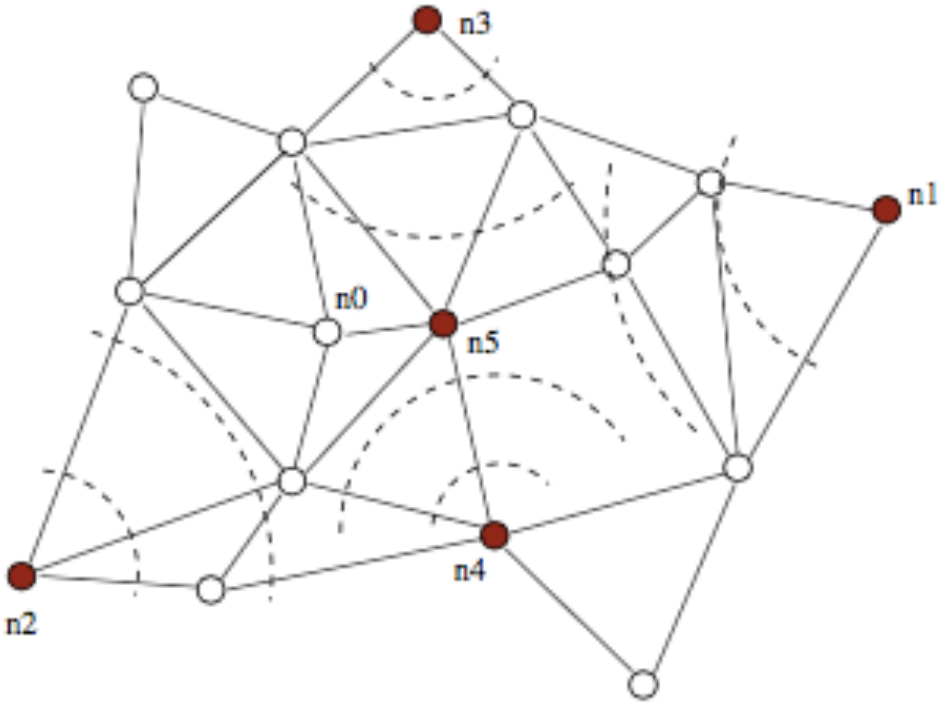


Figura 3.1 - Nodi eletti al termine della prima fase [2]

Per tutti gli altri nodi n_i vengono calcolate le coordinate polari (ρ, θ) con le formule indicate qui sotto, dove R rappresenta il massimo range di comunicazione disponibile per ogni nodo (si ipotizza che ogni sensore abbia lo stesso raggio di copertura).

$$\rho_i = h_{5,i} x R$$

$$\theta_i = tan^{-1}(\frac{h_{1,i} - h_{2,i}}{h_{3,i} - h_{4,i}})$$

Equazione 1 - Coordinate polari del nodo n_i

3.3 Fase di ottimizazzione (mass-spring)

La seconda fase, al fine di aggiustare le coordinate ottenute nella prima in modo molto approssimativo, si basa sul concetto delle molle. Immaginiamo ogni arco del grafo come una molla la cui lunghezza a riposo è pari alla distanza tra le due masse a cui è collegata. Se la stima della distanza tra due nodi è più grande della distanza reale, la molla imprime una forza che li spinge lontano; viceversa, se la distanza stimata è più piccola di quella effettiva, la molla tenderà ad avvicinare i due nodi. Se applicassimo questo concetto per ogni nodo del grafo, si otterrebbe un sistema di molle in cui, a seconda delle misure prese volta per volta, i nodi vengono avvicinati o allontanati tra di loro a seconda della forza che imprimono loro le varie molle [2],[5]. La stima finale delle posizioni dei nodi equivale al punto di riposo del sistema meccanico massa-molle il quale raggiunge l'equilibrio quando l'energia del sistema è minimizzata.

Per applicare questo concetto viene seguito il seguente approccio:

I. In un certo momento, ogni nodo n_i ha una stima della propria posizione all'interno della rete ed invia, periodicamente, le proprie coordinate ai nodi vicini. A questo punto, ogni nodo n_i si calcola la distanza geometrica $d_{i,j}$ verso ogni vicino n_j. Nominando $\hat{v}_{i,j}$ come il versore in direzione da p_i a p_j (con p_i stima del generico nodo i), e ponendo $r_{i,j}$ come la distanza misurata tra i due, si ottiene l'espressione della forza, indicata nell'equazione 2, applicata tra i e j;

$$\vec{\mathbf{F}}_i = \hat{v}_{i,j}(d_{i,j} - r_{i,j})$$

Equazione 2 - Espressione del vettore Forza impresso al nodo i dal nodo j

II. Essendo ogni nodo soggetto a tante forze quanti sono i vicini, si calcola la risultante sul generico nodo n_i:

$$\vec{\mathbf{F}}_\mathbf{i} = \sum_{i,j} \vec{\mathbf{F}}_{i,j}$$

Equazione 3 - Espressione della forza totale applicata sul nodo *i* da tutti i suoi vicini

III. Ogni nodo viene spostato in modo proporzionale alla forza risultante applicata ad esso. In modo empirico si è selezionato lo spostamento indicato nell'equazione successiva, secondo il quale un nodo *i* si dovrebbe muovere grazie ad un vettore di direzione e verso stabiliti dalla forza risultante, ma di modulo pari a quello della forza diviso il numero complessivo di vicini m_i.

$$\frac{|\vec{\mathbf{F}}_\mathbf{i}|}{2m_i}$$

Equazione 4 - Espressione dello spostamento subito da n_i

IV. Il procedimento viene ripetuto in modo tale da correggere la posizione di ogni nodo sulla base della stima della misura tra un nodo e l'altro, ad ogni iterazione.

Teoricamente, affinchè la stima delle coordinate di ogni nodo vada a coincidere con la posizione reale, è necessario che la forza risultante applicata ad ogni nodo sia nulla. In realtà, dato che la forza dipende dalla differenza tra distanza reale e distanza misurata e dato che le misure effettuate dai nodi sono soggette ad imprecisioni, non si può far altro che stabilire un margine di errore tale per cui il risultato ottenuto può essere considerato accettabile oppure

no. Inoltre, ogni volta che le posizioni dei nodi vengono aggiustate in direzione di quelle reali, le forze risultanti ottenute nell'iterazione successiva saranno minori rispetto a quelle precedenti, generando quindi degli spostamenti meno significativi rispetto all'iterazione precedente.

Al fine di ottenere uno spostamento che non sia troppo brusco per non rischiare di portare i vari nodi in posizioni completamente distanti da quelle reali e rischiando quindi di incorrere in minimi locali, è di estrema importanza la scelta del coefficiente di spostamento.

Tale coefficiente, che non deve essere neanche troppo piccolo per non rendere necessarie troppe iterazioni, è stato selezionato in modo empirico pari a *1/2m* (equazione 4), dove *m* è il numero di vicini del nodo in considerazione.

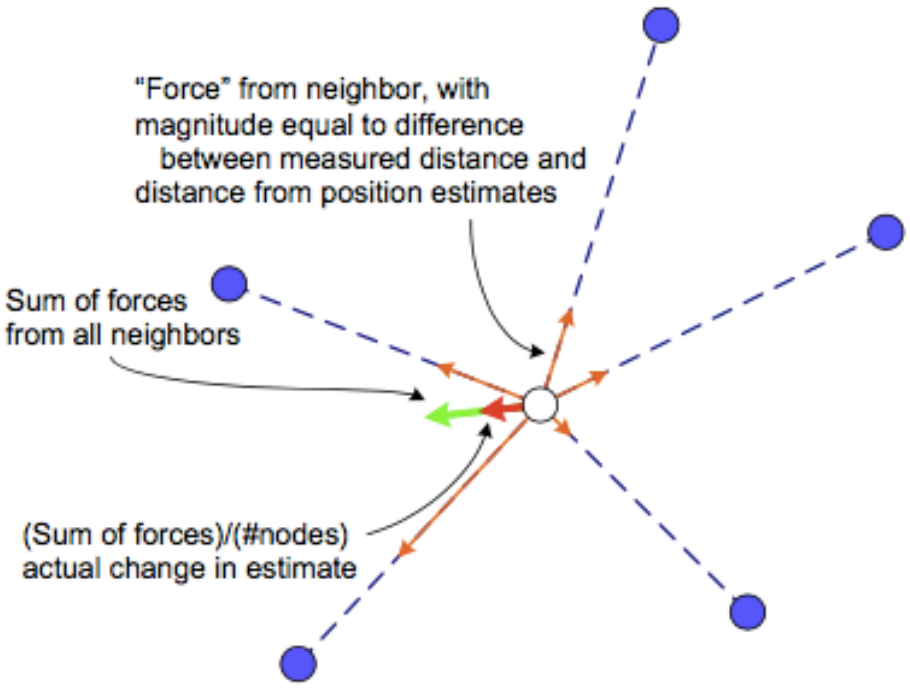


Figura 3.2 - Rappresentazione delle forze che agiscono su un nodo [9]

Fondamentalmente, ci sono due modalità diverse per concludere la mass-spring optimization: selezionando a priori il numero di iterazioni oppure iterando finchè su ogni nodo venga esercitata una forza risultante inferiore ad un certa soglia. Se tale soglia viene presa molto bassa (idealmente zero) allora si assicura una stima finale prossima alla posizione reale, nel caso di convergenza dell'algoritmo. E' però utile notare che se la soglia scelta è effettivamente troppo bassa rispetto alle distanze in gioco, potrebbe essere necessario un numero molto alto di iterazioni della fase di mass-spring in quanto i vettori forza assumono valori sempre più piccoli, determinando spostastamenti via via inferiori.

L'approccio di ottimizzazione qui trattato fa molto affidamento alla stima iniziale. Infatti, il concetto alla base del mass-spring è quello secondo il quale la posizione dei nodi deve solamente essere aggiustata, sulla base della miglior stima in nostro possesso. Quindi, se dalla stima ricavata nella prima fase i nodi risultano essere posizionati da tutt'altra parte rispetto alla posizione reale, non possiamo aspettarci che il mass-spring giunga a convergenza.

Capitolo 4

– Implementazione AFL e analisi simulazioni

4.1 - Descrizione caso reale di localizzazione all'interno di un magazzino con misure di distanza

Analizziamo ora il seguente caso di particolare interesse pratico. Consideriamo un magazzino all'interno del quale si vuole montare un sistema di localizzazione al fine di garantire il tracciamento di alcuni terminali mobili. Si decide quindi di montare dei nodi ancora sul soffitto del magazzino in modo da monitorare la zona. Come prima cosa, il sistema si deve autolocalizzare una volta per tutte per poi riuscire a localizzare i vari terminali.

Entra qui in gioco l'AFL che ha proprio come scopo l'autolocalizzazione dei nodi montati.

I nodi ancora installati devono essere dotati della tecnologia necessaria per effettuare misure di distanza, ad esempio tramite RSSI o misura del tempo di arrivo del segnale.

Nella trattazione è molto importante considerare che le misure effettuate non potranno mai essere precise del tutto; infatti, anche con una tecnologia molto avanzata, è sempre presente almeno il rumore termico che influisce nel processo di misura. Da un punto di vista matematico, ipotizziamo di modellare l'errore con una variabile aleatoria guassiana di valor medio nullo e varianza σ^2_m.

Prima di implementare l'algoritmo possiamo fare la ragionevole ipotesi di conoscere gia in modo grossolano la posizione dei nodi. A tal proposito, possiamo sostituire la

prima fase dell'AFL con una stima iniziale approssimativa $\hat{s}$ delle coordinate, modellabile nel seguente modo:

$$\hat{s} = p + e$$

dove e rappresenta una variabile aleatoria gaussiana di valor medio nullo e varianza σ^2_i e p la posizione reale dei nodi.

Avendo quindi una prima stima delle coordinate, si può passare direttamente alla fase di ottimizzazione (in quanto la prima fase dell'AFL consiste nel trovare una stima iniziale da cui partire con il mass-spring).

Data la particolare geometria della rete, si è notato empiricamente che sono sufficienti un numero relativamente basso di iterazioni della fase di mass-spring optimization. Infatti, anche solo con una trentina di iterazioni, l'algoritmo offre ottimi risultati (nel caso in cui converga).

Due sono i parametri decisivi da considerare:

- deviazione standard relativa all'errore di misura (σ_m);
- deviazione standard relativa all'errore di stima iniziale (σ_i).

La deviazione standard σ_m dipende fondamentalmente dal tipo di ambiente in cui viene installata la rete: infatti, la presenza di più o meno ostacoli, la temperatura a cui si trovano a lavorare i nodi, la presenza o meno di altri tipi di disturbi sono tutte cause che influenzano il processo di misura.

La deviazione standard σ_i dipende invece da quanto siamo incerti circa la stima iniziale dei nodi.

Al fine di valutare in maniera efficace l'impatto che provocano queste variabili sul risultato finale, vengono effettuate simulazioni applicando il metodo di Monte Carlo.

4.1.1 Discussione parametri e simulazione

Impostiamo il problema considerando i seguenti parametri:

- soffitto del magazzino di forma rettangolare, con area 80x60 m^2;
- nodi disposti lungo il perimetro e sugli assi del soffitto uno circa ogni 20 metri;
- raggio di copertura di ogni nodo pari a 40 metri;
- deviazione standard relativa all'errore di misura pari a 20 cm;
- deviazione standard relativa all'incertezza sulla stima iniziale pari a 3 metri.

Una volta avviato l'algoritmo, verrà generato un grafo i cui nodi saranno disposti lungo il perimetro di un rettangolo rispettando le specifiche richieste. Come prima stima verrà considerata la posizione reale dei nodi a cui si aggiunge una variabile aleatoria gaussiana di valor medio nullo e di deviazione standard σ_i. Atteso che l'algoritmo giunga a conclusione, possiamo confrontare le figure 4.1 e 4.2, in cui vengono riportate le immagini relative al grafo originato casualmente e al grafo stimato.

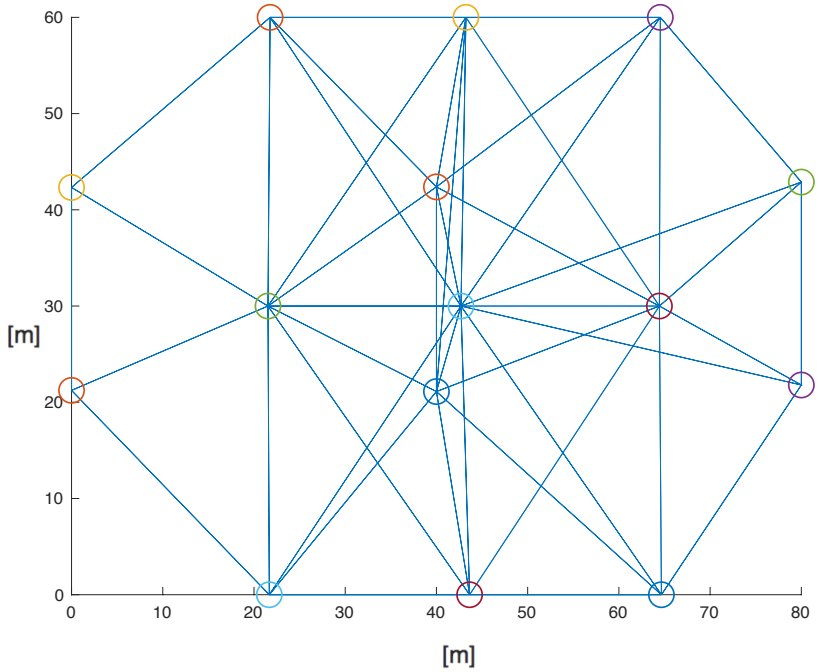


Figura 4.1 - Grafo generato casualmente (con σ_i = 1m)

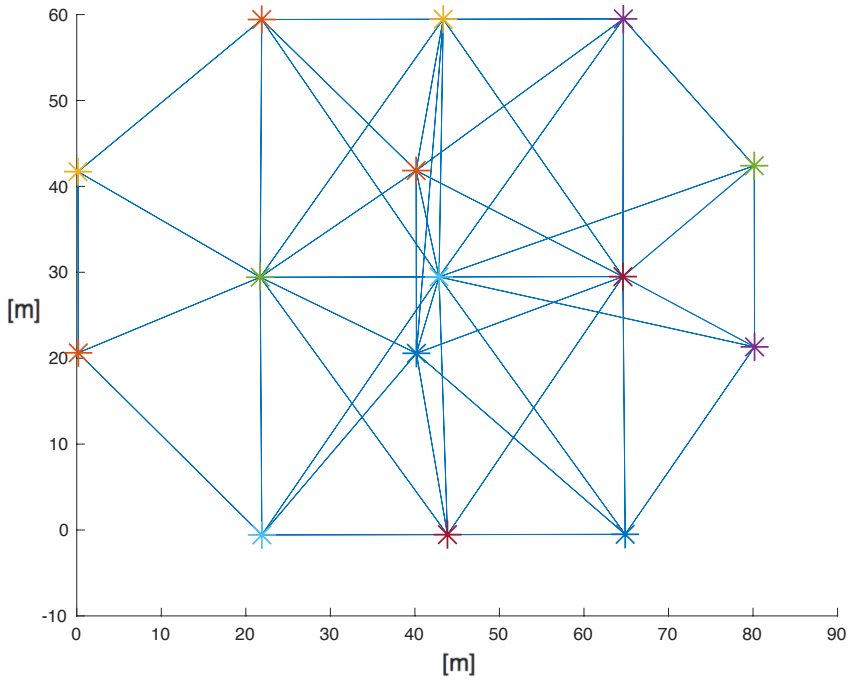


Figura 4.2 - Stima finale delle posizioni ottenuta al termine dell'algoritmo

Come prima informazione, dalle figure 4.1 e 4.2 si può valutare se l'algoritmo è arrivato a convergenza o meno.

Infatti, se la funzione volta a minimizzare la distanza tra posizione reale e posizione stimata dei nodo della rete, cade in zone di minimo locale molto lontane dal minimo assoluto, la forma del grafo ottenuto è completamente diversa da quello di partenza. Viceversa, la topologia del grafo stimato rimane la medesima di quello orginiario.

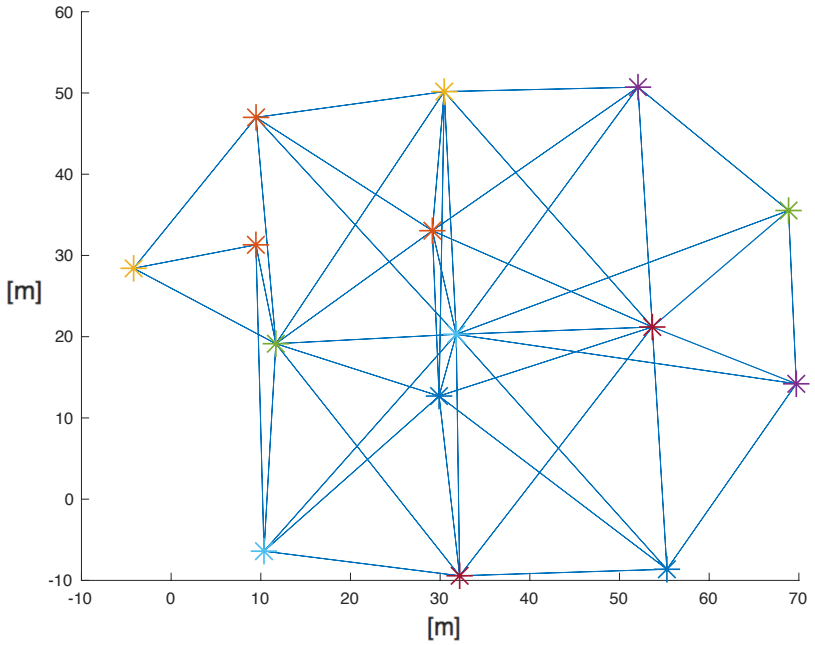


Figura 4.3 - Caso di non convergenza (ottenuto impostando σ_i=15m e σ_m=1m)

Nella figura 4.3 si nota che la non convergenza può essere determinata da local-flips. In questo caso, infatti, è facile immaginare che il problema sia nato da una pessima stima del nodo presente in basso a sinistra, che ha portato l'algoritmo a localizzarlo in una zona di minimo locale lontana dalla zona di minimo assoluto.

4.1.2 Root mean square error

Per poter valutare efficacemente i risultati ottenuti, introduciamo il RMSE (root mean square error) come figura di merito; definito come la media delle differenze tra posizione reale di un nodo e posizione ottenuta, dà un'informazione globale di quanto la stima di ogni nodo sia lontana da quella iniziale.

Per simulare il comportamento dell'algoritmo al variare di σ_i e σ_m, assumendo come figura di merito il RMSE, si è adottato il metodo di Monte Carlo al fine di ottenere risultati il più realistici possibili. Infatti, per ogni iterazione di Monte Carlo, ciascuna variabile aleatoria viene ricalcolata; mediando i risultati ottenuti si arriva a disporre di dati più affidabili.

Bisogna però tenere in mente che non ha senso considerare il RMSE nel caso di non convergenza; infatti, assumendo questo un valore molto elevato, la media globale degli errori di distanza commessi si alzerebbe troppo. A tal proposito, è utile introdurre un meccanismo di riconoscimento dei casi di non convergenza.

4.1.3 Simulazioni Monte Carlo

Consideriamo *m* iterazioni di Monte Carlo, con *m* elevato. Tenendo traccia di ogni RMSE in un vettore di dati, al termine dell'ultima iterazione è possibile farne una media. Se si andasse a verificare ogni valore di questo vettore, ci si accorgerebbe del fatto che a volte si riscontrerebbero valori estremamente elevati rispetto agli altri (outliers), alzando cosi la media complessiva. Dato che l'obiettivo è quello di valutare come i parametri σ_i e σ_m influenzano il risultato nel caso di buona riuscita dell'algoritmo, è utile non considerare i casi di non convergenza (in cui il RMSE assume valori elevati). Quindi, ad ogni simulazione volta a ricercare un legame tra il RMSE e le deviazioni standard (σ_i o σ_m) è necessario affiancare un meccanismo che tenga traccia del numero di casi di non convergenza e che scarti i relativi valori di RMSE.

I valori RMSE che si decide di scartare sono quelli che, fissato uno scostamento *S*, soddisfano il seguente requisito:

$$RMSE > RMSE\_media + S$$

In questo modo si può valutare con efficacia quale RMSE si ottiene mediamente dato un certo valore di σ_i o σ_m, avendo anche l'informazione su quante volte l'algoritmo è caduto in zone di non convergenza.

Di seguito vengono riportati i grafici relativi alle simulazione effettuate, variando σ_m tra 0 (caso ideale di misura perfetta) e 0.8 metri e σ_i tra 1 e 5 metri. Per ogni simulazione i grafici riportano l'andamento del RMSE (valutato solo considerando i casi di convergenza) e l'andamento dell'indice di affidabilità (reliability) che fornisce la percentuale delle volte in cui l'algoritmo ha raggiunto la convergenza.

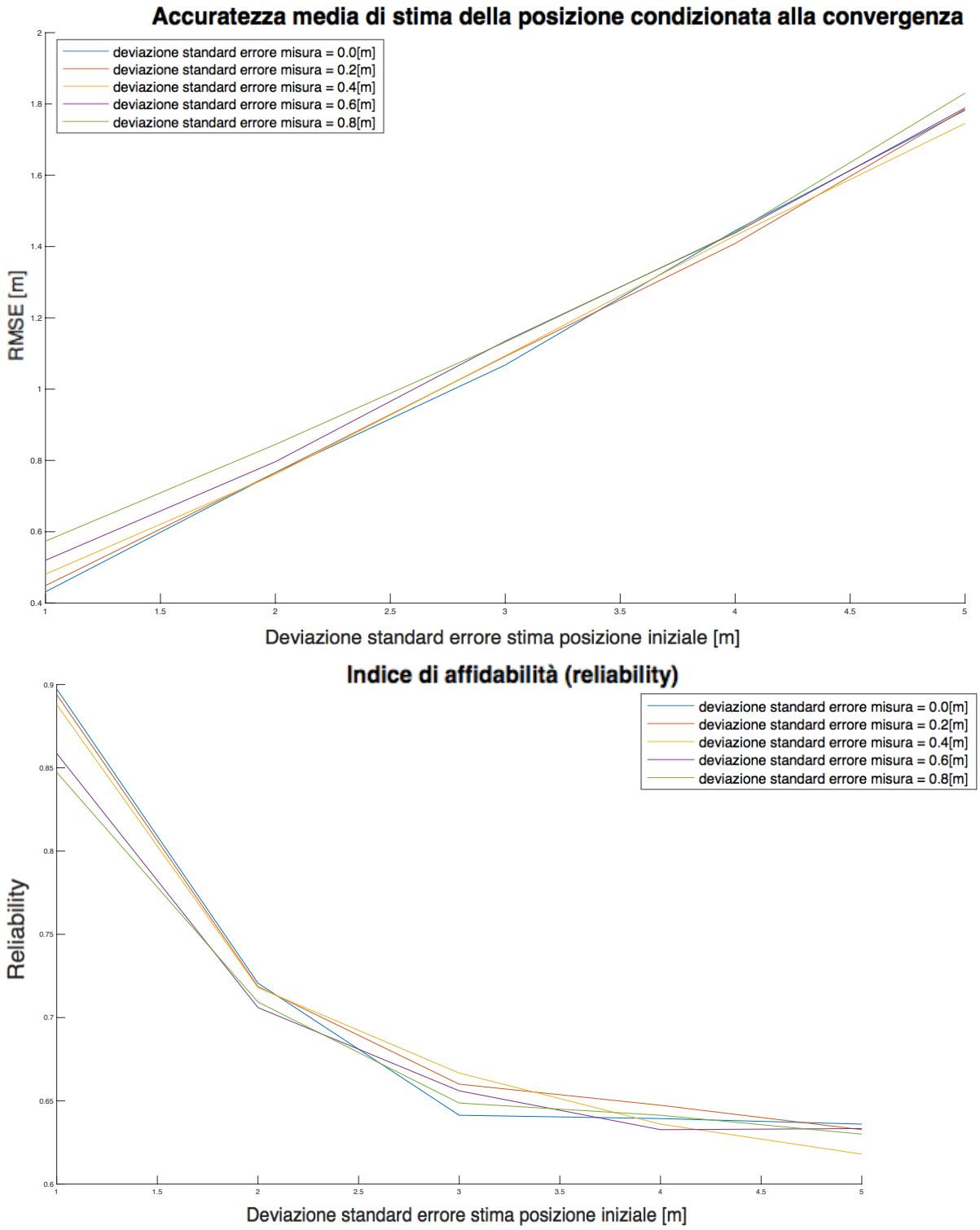


Figura 4.4 - Simulazioni con misure di distanza (m=1500)

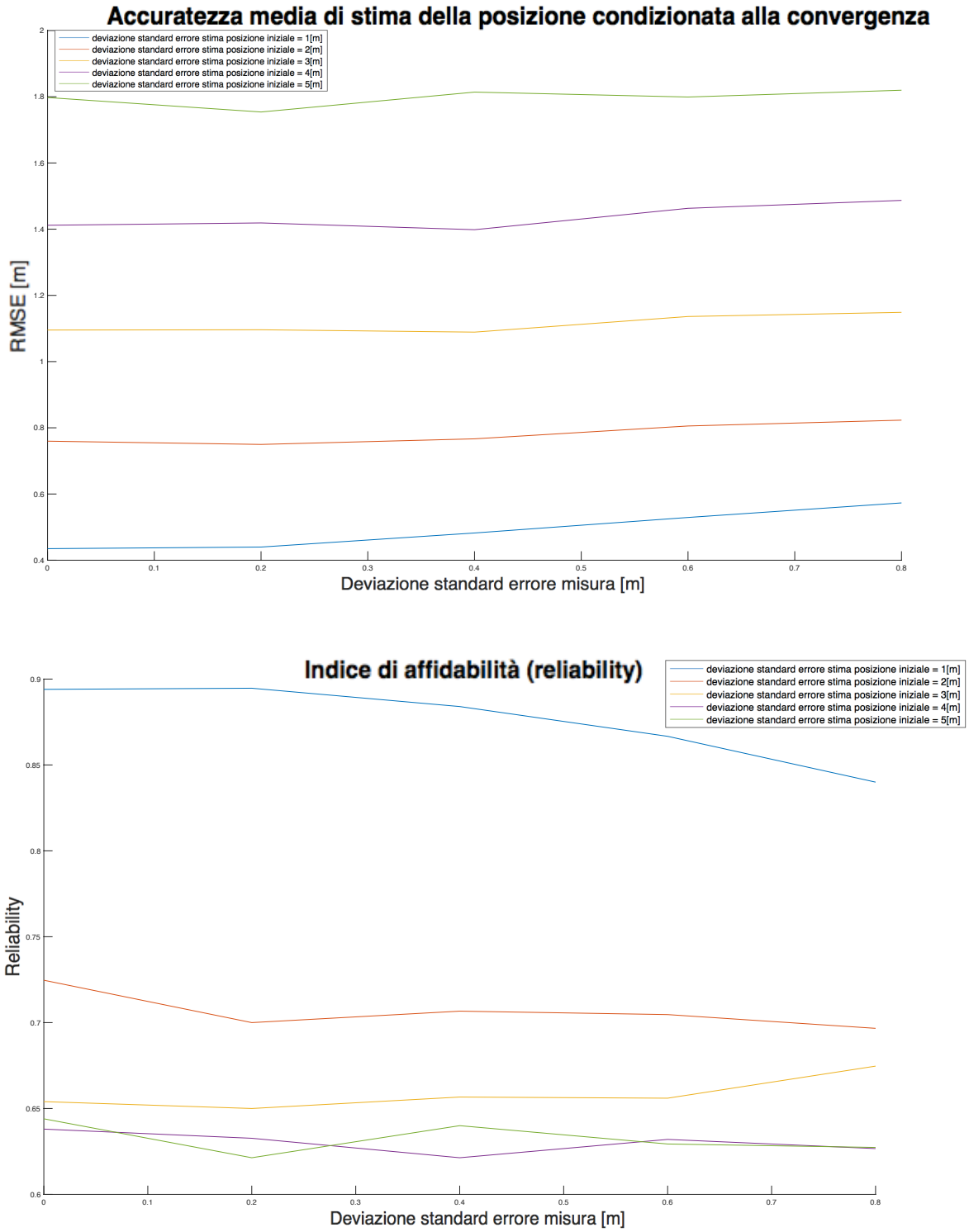


Figura 4.5 - Simulazioni con misure di distanza (m=1500)

Le due simulazioni riportano gli stessi risultati ma in forma diversa. Osservando i grafici si può evincere che, fissato un errore di misura, valgono le seguenti affermazioni:

1. Il RMSE aumenta all'aumentare dell'incertezza sulla posizione iniziale dei nodi;
2. L'affidabilità diminuisce all'aumentare dell'incertezza sulla posizione iniziale dei nodi.

L'andamento lineare del RMSE in funzione di σ_i dà la possibilità di fissare una soglia per stabilire l'errore che si è disposti a tollerare sulla stima iniziale della posizione dei nodi ancora. In ogni caso, si nota che è sempre conveniente lavorare nelle condizioni di σ_i il più basso possibile.

Questo risultato evidenzia come l'approccio mass-spring sia estremamente influenzato dalla prima stima del grafo; infatti, più la posizione dei nodi viene inizialmente stimata lontano da quella di origine, più è probabile che si incorra in minimi locali lontani da quello assoluto. Viceversa, più σ_m è bassa, più i minimi locali saranno vicini al minimo assoluto.

Si rivela anche che se σ_i è piccola, o nulla, il RMSE è dominato dagli errori di stima di misura; viceversa, se σ_i è alta, gli errori di misura incidono meno e a dominare gli effetti del risultato sono gli errori di stima iniziale.

4.2 Caso reale di localizzazione all'interno di un magazzino con sensori dotati di misure TDoA

In questo lavoro si è affrontato anche il caso in cui i sensori non dispongano della tecnologia per effettuare misure di distanza diretta. Ipotizzando di avere a disposizioni misure basate sul *TDoA*, si è cercato di adattare l'algoritmo affinchè possa essere comunque applicata la fase di mass-spring.

4.2.1 Misure TDoA

Consideriamo un generico nodo *i* che emette un segnale broadcast. Le misure di TDoA prevedono due diversi ricevitori; in particolare, il TDoA viene calcolato sulla base della differenza di tempo tra il momento in cui il segnale viene captato prima da un sensore poi dall'altro. Conoscendo la velocità di propagazione del segnale, l'informazione temporale viene convertita in misure di differenze di distanze.

A livello fisico, i nodi che adottano la tecnica TDoA necessitano di un hardware più complesso in quanto richiedono una sicronizzazione accurata dei nodi.

Consideriamo tre nodi *A*,*B*,*C*. Ipotizziamo che *A* emetta un segnale, captato da *B* e da *C* in determinati istanti temporali. Posto $r_{i,j}$ la distanza tra due generici punti, possiamo definire la misura ottenuta tramite TDoA come

$$r_{A,B} - r_{A,C}$$

Avendo anche una stima della posizione dei tre nodi, è possibile calcolare la stessa misura, definita come $d_{A,B} - d_{A,C}$, posto $d_{i,j}$ come la distanza tra due punti sulla base delle loro coordinate. Confrontando $r_{A,B} - r_{A,C}$ con $d_{A,B} - d_{A,C}$ si può valutare in che direzione e verso un nodo dovrebbe essere spostato per cercare di minimizzare il divario tra misura reale e distanza stimata.

Per applicazioni che necessitano di particolare precisione, è utile tenere in considerazione che le misure di

TDoA definiscono come luogo geometrico delle iperboli (essendo l'iperbole il luogo geometrico dei punti del piano tali per cui resta costante la differenza delle distanze da due punti fissi detto fuochi) [10].

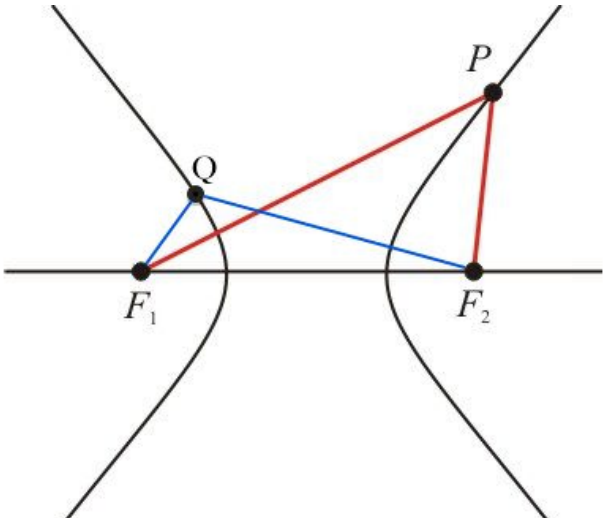


Figura 4.6 - Rappresentazione distanza tra due punti generici e i due fuochi dell'iperbole

4.2.2 Simulazioni con misure TDoA

Con gli stessi parametri adottati per il caso di misure di distanza, effettuando le simulazioni si sono ottenute le seguenti curve:

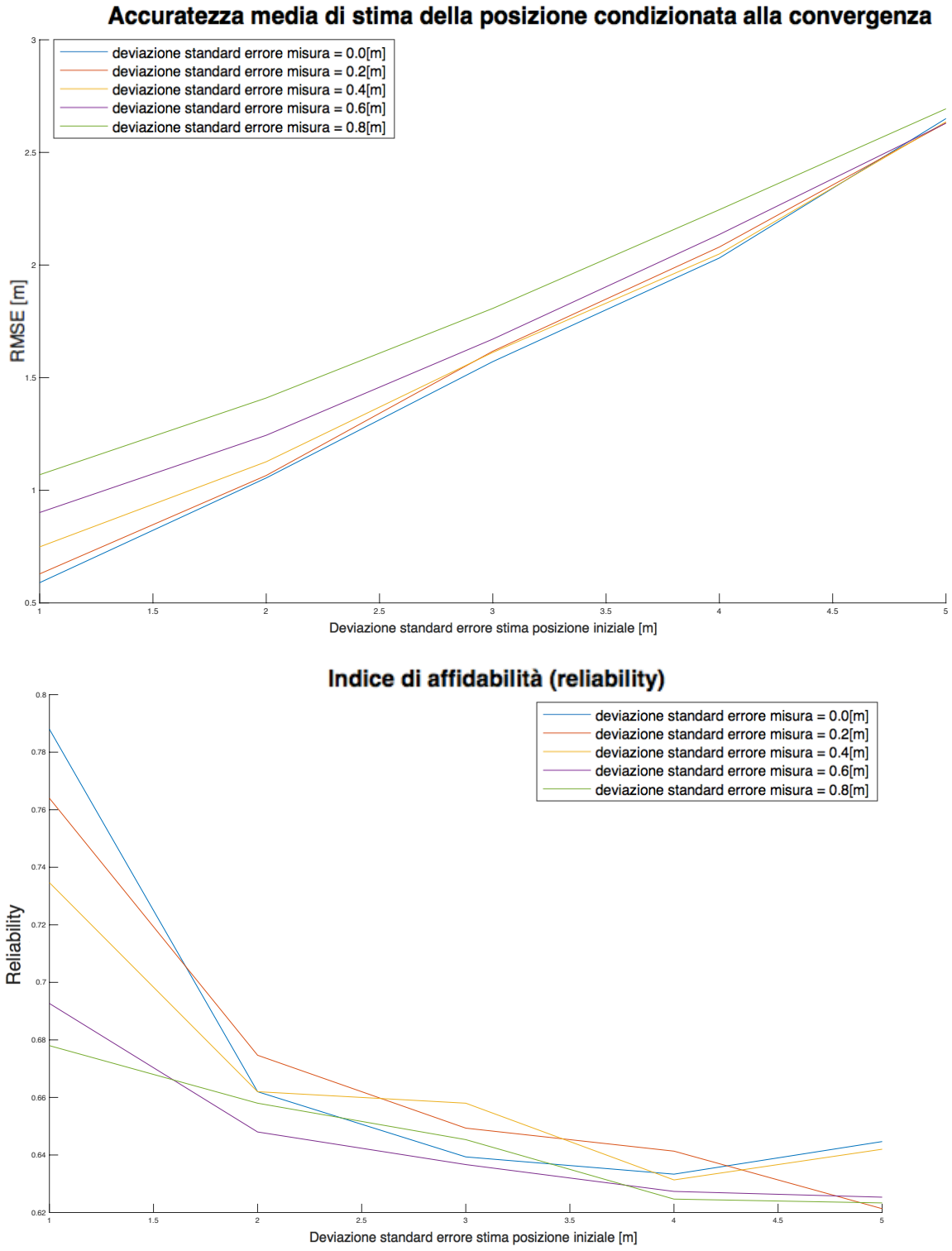


Figura 4.7 - Simulazioni con misure TDoA (m=1500)

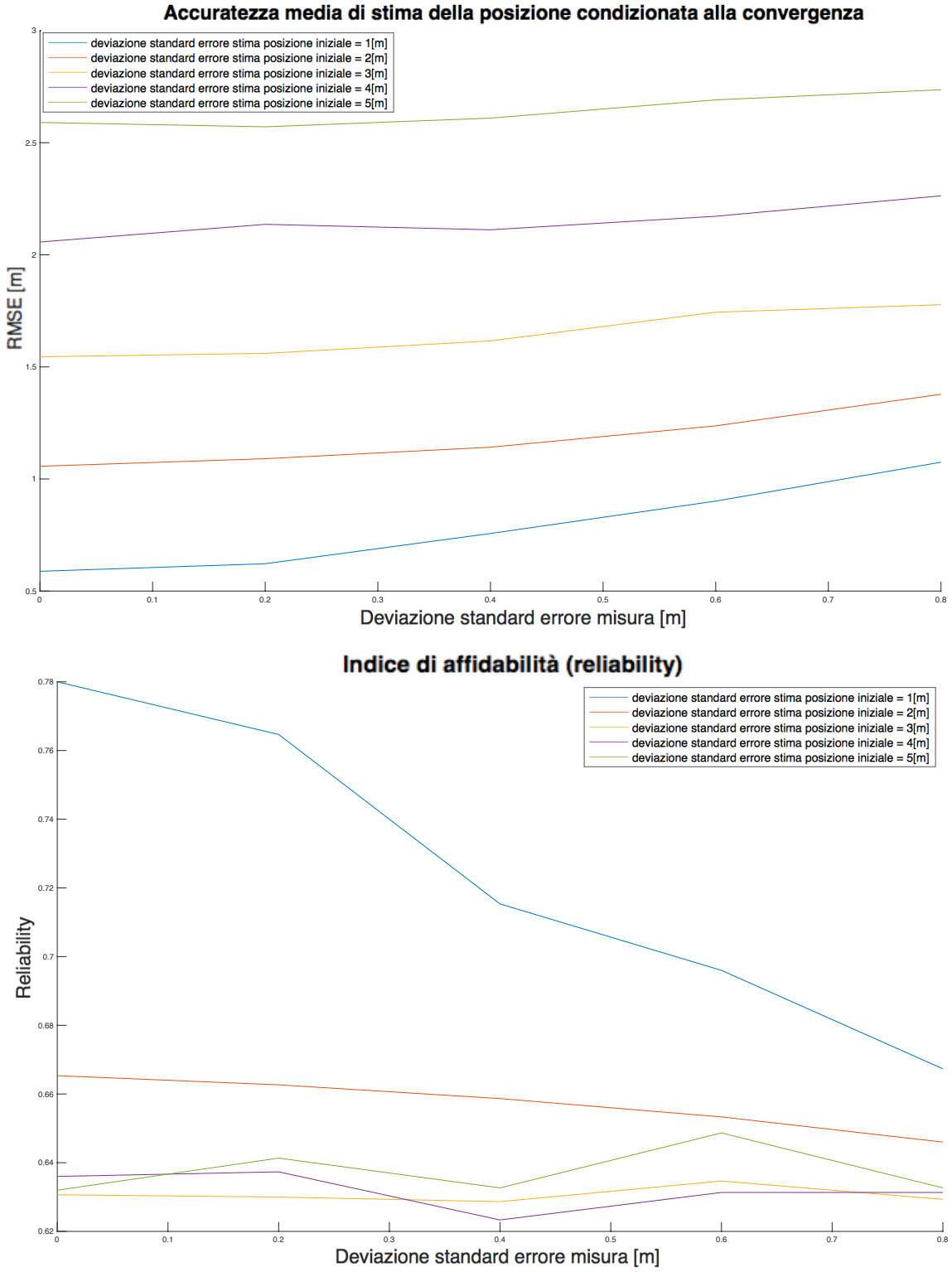


Figura 4.8 - Simulazioni con misure TDoA (m=1500)

Dalle simulazioni si può notare che l'andamento è lo stesso del caso di misure di distanza ma a parità di parametri il mass-spring con misure TDoA fornisce risultati meno precisi. Anche l'affidabilità è minore e dai grafici si riscontra lo stesso principio ricavato dal caso precedente: per valori di σ_m relativamente bassi, a dominare gli effetti del risultato è σ_i.

Il fatto che la fase di mass-spring fornisca risultati meno precisi rispetto al caso precedente si spiega tenendo conto della natura dell'algoritmo. La fase di ottimizzazione dell'AFL è infatti basata sul concetto delle molle: la differenza tra la misura vera e la stima della distanza tra un i-esimo e j-esimo nodo dà un'informazione diretta sull'entità del modulo, sulla direzione e sul verso della forza che andrebbe applicata per aggiustare adeguatamente la posizione. Dato che le misure di TDoA sono legate non alla distanza tra il nodo *i* e il nodo *j* ma alla differenza di distanze tra *i* e *j* e tra *i* e un terzo nodo *k*, è intuitivo pensare che per questa tipologia di algoritmo siano più adeguate misure di distanza diretta.

4.3 Nuvola di droni con distribuzione uniforme

4.3.1 Introduzione scenario

Consideriamo ora uno scenario differente. Consideriamo una nuvola di droni sparsa in un territorio al fine di monitorare la zona circostante. A differenza del caso di nodi installati in un magazzino, è necessario fare affidamento sulla prima fase dell'AFL per poter avere una stima iniziale della posizione. A tal proposito bisogna fare alcune precisazioni al fine di applicare correttamente il metodo proposto dall'AFL.

La prima fase dell'AFL presuppone che:

- il grafo sia uniformemente distribuito;
- la connettività globale sia sufficientemente elevata (requisito soddisfatto se ogni nodo è connesso ad almeno altri 3 nodi);
- il raggio di copertura non sia troppo elevato al fine di evitare che anche i nodi più distanti tra di loro possano comunicare direttamente.

L'uniformità della distribuzione dei nodi è richiesta dalla natura stessa della prima fase dell'AFL; se infatti supponiamo che il grafo possa assumere qualunque topologia non c'è alcuna garanzia sul fatto che i nodi scelti come riferimento formino un sistema di assi cartesiani adeguato. Le equazioni delle coordinate polari (equazione 1) utilizzate per il calcolo della prima stima dei nodi richiedono l'esistenza di un sistema di riferimento in cui n_5 sia al centro del grafo e n_1, n_2, n_3, n_4 nei punti più esterni possibile.

Se il grafo potesse assumere qualunque topologia è molto improbabile che i nodi di riferimento vengano selezionati in modo adeguato, come sopra specificato.

4.3.2 Disposizione droni

Fatte queste considerazioni si è scelto di ipotizzare la seguente topologia di rete, ipotizzando che i droni non si muovano in modo completamente indipendete l'uno dall'altro ma esplorino il territorio come un'unica entità.

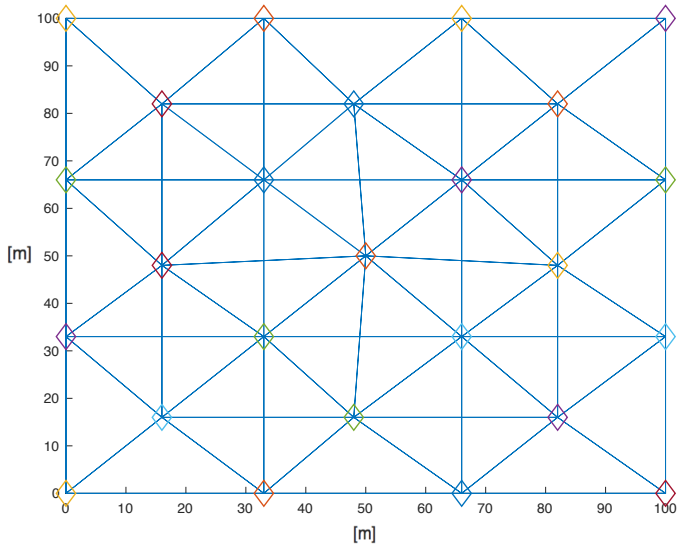


Figura 4.9 - Distribuzione uniforme di 25 droni in un'area di circa 100x100 m^2

Ovviamente non si vuole pretendere che la nuvola, durante il movimento, rimanga istante per istante formata da droni posizionati esattamente nella configurazione sopra menzionata, ma che comunque ogni drone sia condizionato nel movimento. Se infatti i droni si spostassero in modo indipendente l'uno dall'altro l'algoritmo si troverebbe a dover stimare i nodi di riferimento sulla base di una topologia non uniforme, e fallirebbe a cause delle motivazioni sopra citate (a meno che il numero di droni non sia cosi elevato da formare comunque un grafo uniformemente distribuito). Quindi, anche in questo caso, è utile modellare lo scostamento di ogni drone dalla posizione ideale da noi ipotizzata con una variabile aleatoria

di tipo gaussiano a valor medio nullo e deviazione standard σ_i.

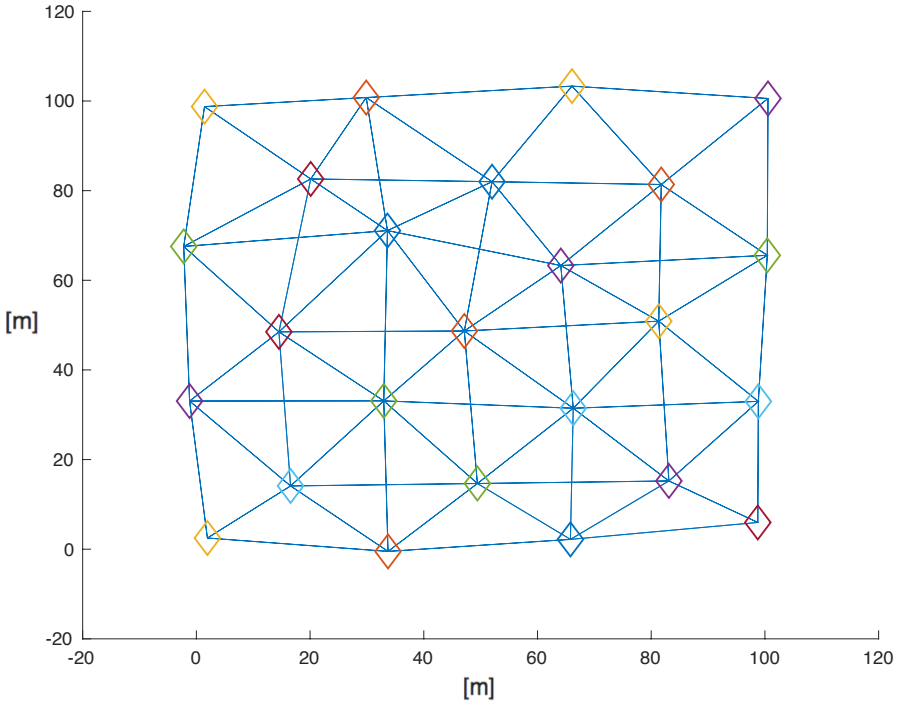


Figura 4.10 - Modello realistico di distribuzione uniforme di droni (σ_i=2m)

E' utile notare che, a differenza del caso del magazzino, σ_i non è volta a modellare l'imprecisione della stima iniziale in nostro possesso, in quanto supponiamo non avere nessuna informazione circa la posizione dei droni; σ_i viene infatti introdotta per modellare l'impossibilità dei droni di formare istante per istante, durante il movimento, una rete identica a quella di partenza.

Questa imprecisione, modellata dalla deviazione standard σ_i, deve assumere però un valore non troppo elevato al fine di garantire l'uniformità della rete (non è quindi ammesso il caso in cui i droni si muovano completamente senza vincoli nel territo).

4.3.3 Discussione parametri e simulazione

Empiricamente si è verificato che sono necessarie molte più iterazioni della fase di mass-spring rispetto al caso del magazzino. Questo perchè, come si può notare dalla figura 4.11 sotto riportata, non possiamo considerare la prima fase dell'AFL efficace come avere una stima (seppur grossolana) dell'intero grafo; infatti, nel caso di topologia random, non si riesce fin da subito ad avere una stima iniziale dei nodi che sia già vicina alla zona di minimo assoluto. Per questo motivo, le simulazioni sono state effettuate con un numero di iterazioni della fase di mass-spring pari a 1000 (contro le 30 sufficienti nel caso trattato sopra). Anche a causa delle eccessive iterazioni, non si è potuto simulare il comportamento complessivo del sistema come nei casi sopra trattati; si è scelto di restringere l'intervallo di simulazione solo per quei valori di σ_m e di σ_i veramente significativi (ad esempio, si è voluto escludere i casi di σ_i troppo elevato in quanto si suppone che nei casi reali i droni non sbaglino movimento in modo esagerato). Per quanto riguarda la deviazione standard relativa agli errori di misura, si è scelto di analizzare il sistema con σ_m pari a 0 e 0.2 metri, in modo tale da considerare sia il caso ideale sia uno dei casi più comuni di errori di misura.

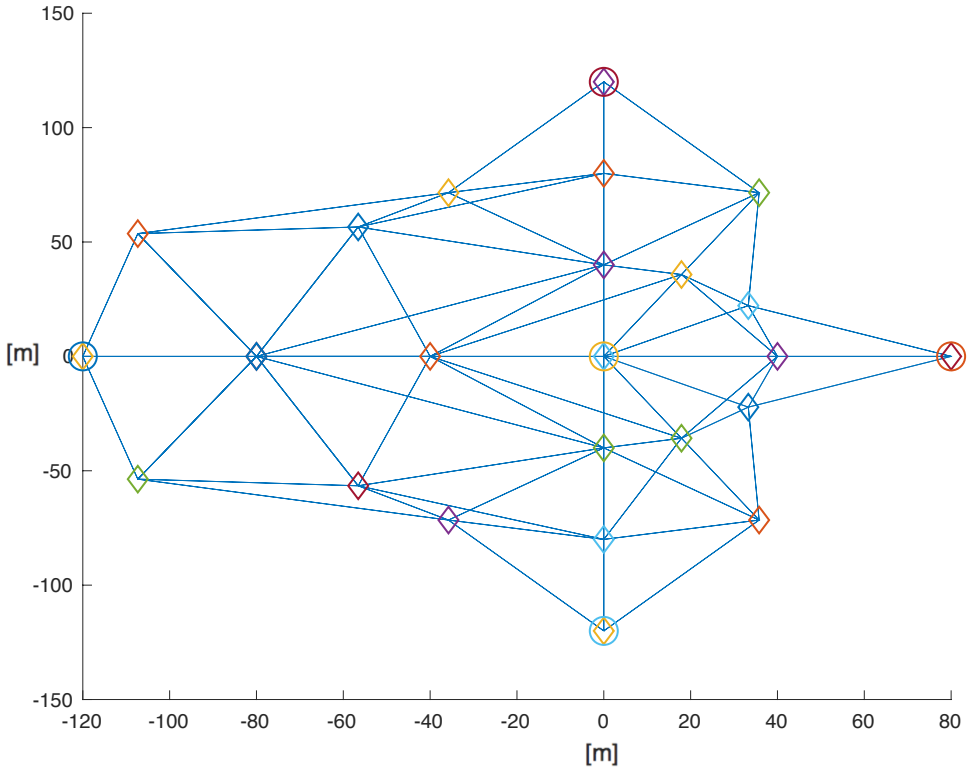


Figure 4.11 - Stima della posizione iniziale dei nodi ottenuta tramite la prima fase dell'AFL (i nodi cerchiati sono i 5 nodi scelti come riferimento)

Dalla figura 4.11 si può notare come i nodi di riferimento formino un sistema di assi cartesiani e come le posizioni stimate siano ben lontane dalle posizioni reali.

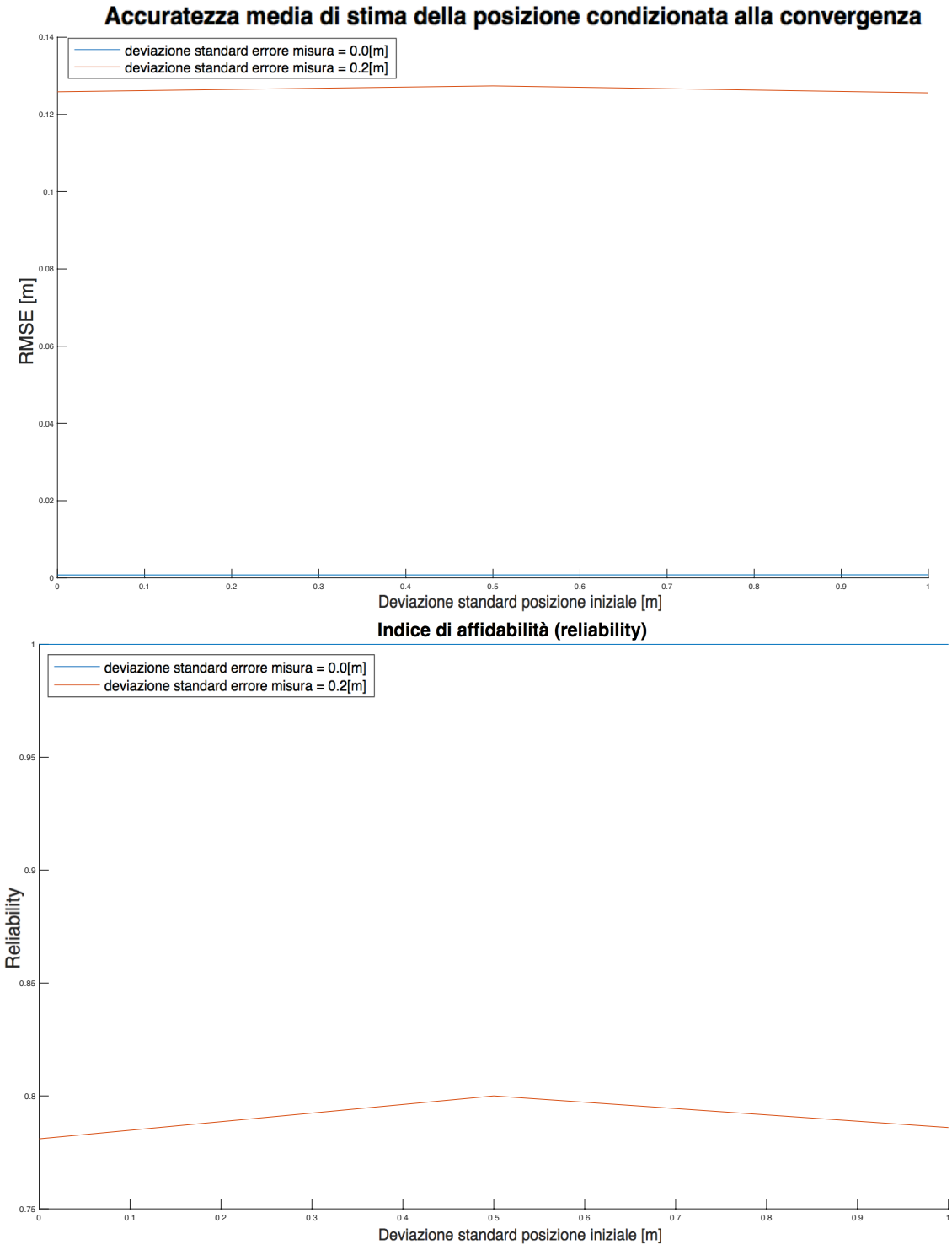
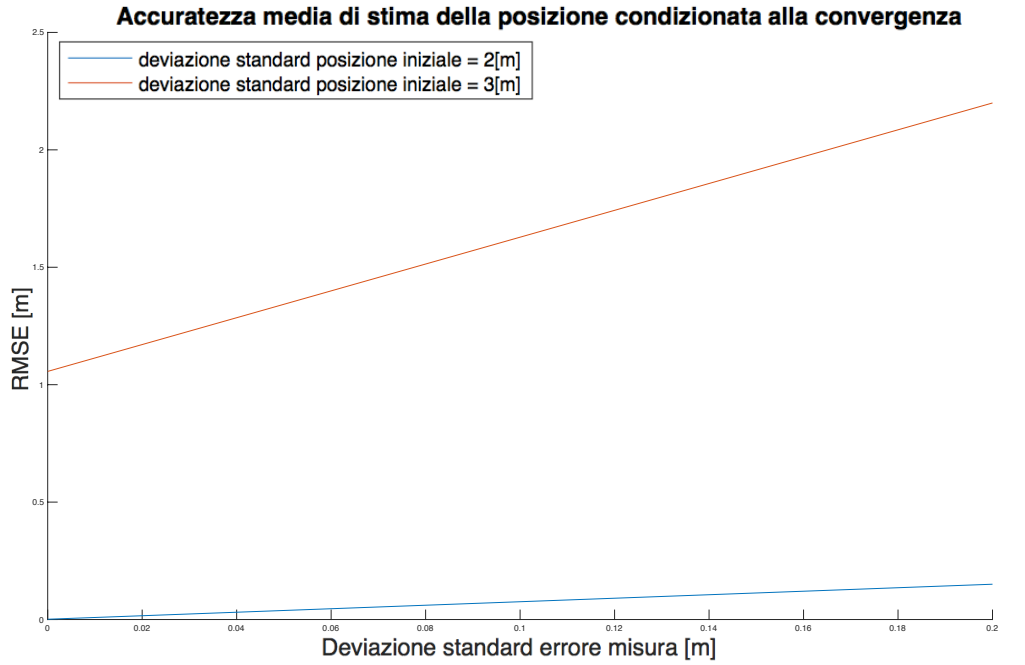


Figura 4.12 - Simulazione con stima iniziale ottenuta tramite AFL (m=1000)



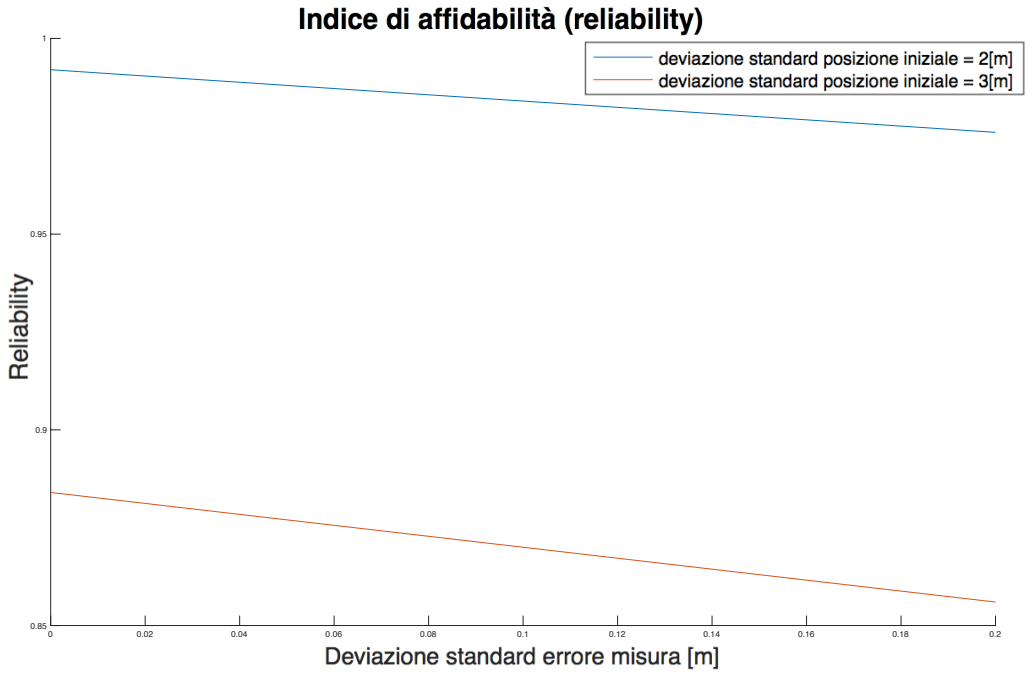


Figura 4.13 - Simulazionioni con stima iniziale ottenuta tramite AFL (m=1000)

Prendiamo in considerazione la figura 4.12. Possiamo notare immediatamente che, a differenza della figura 4.4, fissato un σ_m, all'aumentare di σ_i non aumenta il RMSE.
In realtà bisogna notare che le simulazioni effettuate nel caso della figura 4.12 sono state ristrette ad un intervallo di σ_i i cui valori sono molto piccoli: in particolare compreso tra 0 e 1 metro (mentre nel caso del magazzino la simulazione è stata fatta considerando un intervallo di σ_i compreso tra 1 e 5 metri).

Analizziamo il risultato ottenuto nella figura 4.12. Per quanto riguarda il caso di σ_m=0[m] il RMSE si assesta su 0[m]: tale risultato non ci sorprende se si pensa che, in un'area di 100x100 m^2, all'interno della quale ogni drone dista da un altro circa 25-30 metri, è possibile considerare ogni σ_i compreso nell'intervallo [0,1] come un valore piccolo, tale per cui la topologia uniforme è sempre garantita.

Lo stesso concetto vale per σ_m=0.2[m], in cui però il RMSE non vale 0 ma poco più di 12 cm (in quanto l'errore di misura determina un errore di stima delle forze in gioco).

Il fatto che l'affidabilità sia alta per entrambi i casi di σ_m è spiegata dal fatto che le deviazioni standard relative agli errori di misura considerate sono molto piccole.

Dato che le figure 4.12 e 4.13, come nel caso del magazzino (figure 4.4 e 4.5), riportano il medesimo risultato relativo alla stessa simulazione ma in grafici diversi (uno in cui l'asse delle ordinate rappresenta i valori di σ_m, l'altro in cui rappresenta i valori di σ_i), nel grafico rappresentato nella figura 4.13 è stato scelto di considerare un differente intervallo di valori relativi a σ_i per studiare anche cosa succede per valori più elevati considerando gli stessi σ_m. In particolare, è stato scelto l'intervallo [2,3] metri per gli errori di stima iniziale.

Considerando sia la figura 4.12 che la figura 4.13 si nota che, superato un certo valore di σ_i tale per cui è garantita l'uniformità del grafo (come può essere σ_i=1m), all'aumentare di σ_m il RMSE aumenta linearmente.

In particolare, più σ_i è elevato più la pendenza della curva che lega σ_m al RMSE è ripida.

Questo perchè, nel caso droni un errore di misura nella prima fase è molto più incisivo dello stesso errore in fase di ottimizzazione e se aumenta il rischio di uscire dalla topologia uniforme (quindi σ_i elevato) aumenta il RMSE. Come precedentemente affermato, la prima fase dell'AFL è molto sensibile alla topologia: avere misure errate (seppur di poco) in condizioni di σ_i elevato può portare ad ottenere un'immagine che rispecchia in maniera meno fedele la topologia reale della rete.

Anche l'affidabilità diminuisce più rapidamente all'aumentare di σ_m.

Capitolo 5

- Conclusioni

Al termine di tutte le analisi condotte in questo lavoro, si può affermare che l'AFL sia un algoritmo adatto principalmente per applicazioni in cui i nodi ancora siano fissi.

Ricapitolando, se di questi nodi si conosce a priori una stima, seppur grossolana, della posizione allora l'algoritmo ha ottime probabilità di convergere.

Se si deve fare affidamento su una topologia casuale, è però necessario che i nodi siano distribuiti in modo il più uniforme possibile, affinchè la prima stima non fallisca del tutto.

Se si volesse applicare questo algoritmo ad esempio al caso dei droni, è necessario valutare i tempi di calcolo con i tempi richiesti dai droni stessi. Infatti, essendo il caso dei droni un problema di tracking, in quanto i nodi della rete si muovono nel tempo, se i tempi di calcolo dell'algoritmo sono molto inferiori rispetto al tempo tale per cui il drone deve decidere lo spostamento da effettuare, allora si può affermare di essere in presenza di condizioni di stazionarietà.

Sulla base dell'AFL si potrebbe lavorare per migliorare la stima della prima fase, ad esempio dotando i nodi della rete di sensori per effettuare misure di tipo AoA. Avendo anche la conoscenza degli angoli tra i vari nodi, si potrebbe pensare un nuovo algoritmo per la determinazione dei cinque nodi di riferimento in modo tale da adattare l'AFL ai casi di topologia random, senza richiedere una distribuzione uniforme.

RINGRAZIAMENTI

Il lavoro svolto è stato possibile grazie all'aiuto del professor Davide Dardari, che ringrazio sentitamente per la sua grande disponibilità e pazienza.
Desidero inoltre ringraziare ogni amico per l'aiuto reciproco dato in questi tre anni di formazione.
Infine, ringrazio di cuore la mia famiglia con cui cammino fianco a fianco ogni giorno della mia vita.

Bibliografia

[1] D. DARDARI, E. FALLETTI, and M. LUISE “Satellite and Terrestrial Radio Positioning Techniques - A signal processing perspective” Elsevier Ltd, London, 2011.

[2] NISSANKA, B., PRIYANTHA, HARI BALAKRISHNAN, ERIK DEMAINE and SETH TELLER “Anchor-Free Distributed Localisation in Sensor Networks” Tech Report 892, MIT Laboratory for Computer Science April 15, 2003.

[3] J.WANG, R.K.GHOSH and S.K.DAS “A survey on sensor localization” South China University of Technology and Academy of Mathematics and Systems Science, CAS and Springer-Verlag Berlin Heidelberg, 2010.

[4] Oh-Heum Kwon, Ha-Joo Song and Sangjoon Park “Anchor-free Localization though Flip Error Resistant Map Stitching in Wireless Sensor Network” Surveillance & Reconnaissance Sensor Network Research Team RFID/USN Research Division IT Convergence Technology Research Laboratory, ETRI, 2010.

[5] Juergen Eckert, Felix Villanueva, Reinhard German and Falko Dressler “Distributed Mass-Spring-Relaxation for Anchor-free Self-localization in Sensor and Actor Networks”, Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN), 2011.

[6] U. Nazir, M. A. Arshad, N. Shahid, S. H. Raza “Classification of Localization Algorithms for Wireless Sensor Network: A Survey“, International Conference on Open Source Systems and Technologies, 2012.

[7] James Aspnes, David Goldenberg and Yang Richard Yang “On the computational complexity of sensor network localization”. Algorithmic Aspects of Wireless Sensor Networks: First International Workshop, ALGOSENSORS 2004, Turku, Finland, July 16, 2004. Proceedings. Lecture Notes in Computer Science 3121, Springer-Verlag, 2004, pp. 32—44. Available as YALEU/DCS/TR-1282, April 2004.

[8] Joseph Thomas “A Novel Trilateration Algorithm for Localization of a Transmitter/Receiver Station in a 2D Plane Using Analytical Geometry”,2014

[9] Gustav J. Jordt, Rusty O. Baldwin, John F. Raquet and Barry E. Mullins “Energy cost and error performance of range-aware, anchor-free localization algorithms”, Ad Hoc Networks 6 (2008) 539—559, 2007

[10] Bonan Jin, Xiaosu XuD and Tao Zhang “Robust Time-Difference-of-Arrival (TDOA) Localization Using Weighted Least Squares with Cone Tangent Plane Constraint”, 2018

Indice_ALLEGATO

Codice Matlab AFL - Caso del magazzino - misure RSSI

```
% Giacomo Giorgini, 2018, Tesi di Laurea Triennale (Cesena)
% Implementazione AFL - misure RSSI
% 1) Codice per simulare la creazione di nodi installati sul
% soffitto di un magazzino di dimensione 80x60 m^2.
% 2) Codice per la simulazione della stima iniziale
% 3) Codice per l'ottimizzazione (mass-spring) - sfruttando misure RSSI
% 4) Grafici

clear all
close all
clc

% lista paramentri
R = 40; % raggio di copertura dei nodi
r = 20; % un nodo ogni circa 20 metri
n_iterazioni = 30; % iterazioni fase di mass-spring
scostamento = 0.5;
monte_carlo = 10; % 1500 per simulazioni

indice_out = 1;
for sigma_misura = 0:0.2:0.8

    indice_rmse = 1;
    for sigma_pos_iniziale = 1:5

        for m = 1:monte_carlo

                % le coordinate dei nodi sono contenute in una struttura
                coordinate_iniziali = struct('x',0,'y',0);

                sogliax = 80;
                sogliay = 60;
                % creazione lato sinistro del magazzino (80x60 m^2)
                soglia = 0;
                i = 1;
                while ( soglia < sogliay )
                    e1 = 1;
                    e2 = 2;
                    incertezza = e1 + (e2-e1)*rand; % possibile errore di
posizionamento manuale, [1,2] metri
                    if ( r+incertezza+soglia < sogliay )
```

```
        coordinate_iniziali(i).x = 0;
        coordinate_iniziali(i).y = r+incertezza + soglia;
        i = i+1;
     end
     soglia = r+incertezza + soglia;
end

% creazione lato destro
soglia = 0;
while ( soglia < sogliay )
    incertezza = e1 + (e2-e1)*rand;
    if ( r+incertezza+soglia < sogliay )
        coordinate_iniziali(i).x = sogliax;
        coordinate_iniziali(i).y = r+incertezza + soglia;
        i = i+1;
    end
    soglia = r+incertezza + soglia;
end

% creazione lato inferiore
soglia = 0;
while ( soglia < sogliax )
    incertezza = e1 + (e2-e1)*rand;
    if ( r+incertezza+soglia < sogliax )
        coordinate_iniziali(i).y = 0;
        coordinate_iniziali(i).x = r+incertezza + soglia;
        i = i+1;
    end
    soglia = r+incertezza + soglia;
end

% creazione lato superiore
soglia = 0;
while ( soglia < sogliax )
    incertezza = e1 + (e2-e1)*rand;
    if ( r+incertezza+soglia < sogliax )
        coordinate_iniziali(i).y = sogliay;
        coordinate_iniziali(i).x = r+incertezza + soglia;
        i = i+1;
    end
    soglia = r+incertezza + soglia;
end

% creazione asse orizzontale
```

```
                soglia = 0;
                while ( soglia < sogliax )
                    incertezza = e1 + (e2-e1)*rand;
                    if ( r+incertezza+soglia < sogliax )
                        coordinate_iniziali(i).y = sogliay/2;
                        coordinate_iniziali(i).x = r+incertezza + soglia;
                        i = i+1;
                    end
                    soglia = r+incertezza + soglia;
                end

                % creazione asse verticale
                soglia = 0;
                while ( soglia < sogliay )
                    incertezza = e1 + (e2-e1)*rand;
                    if ( r+incertezza+soglia < sogliay )
                        coordinate_iniziali(i).x = sogliax/2;
                        coordinate_iniziali(i).y = r+incertezza + soglia;
                        i = i+1;
                    end
                    soglia = r+incertezza + soglia;
                end
                i = i-1;

                n_nodi = i;

                matrice_coordinate_iniziali = zeros(n_nodi, 2);
                % creazione matrice (nx2) con le coordinate dei nodi
                for colonna1 = 1:n_nodi
                    matrice_coordinate_iniziali(colonna1,1) =
coordinate_iniziali(colonna1).x;
                end
                for colonna2 = 1:n_nodi
                    matrice_coordinate_iniziali(colonna2,2) =
coordinate_iniziali(colonna2).y;
                end

                matrice_adiacenze = zeros(n_nodi, n_nodi);
                % creazione matrice (nxn) delle adiacenze
                for i = 1:n_nodi
                    for j = 1:n_nodi
                        matrice_adiacenze(i,j) =
confronto(coordinate_iniziali(i), coordinate_iniziali(j), R);
                    end
```

```
                end

                DG = sparse(matrice_adiacenze);

                matrice_stima_iniziale = zeros(n_nodi, 2);
                % prima stima = posizione reale + errore gaussiano
                for i = 1:n_nodi
                    for j = 1:2
                        matrice_stima_iniziale(i,j) =
matrice_coordinate_iniziali(i,j) + (sigma_pos_iniziale * randn(1));
                    end
                end

                % la struttura stima verrà aggiornata ad ogni iterazione
                stima = struct('x',0,'y',0);
                for i = 1:n_nodi
                    stima(i).x = matrice_stima_iniziale(i, 1);
                    stima(i).y = matrice_stima_iniziale(i, 2);
                end
                % salvo le coordinate della stima iniziale
                for i = 1:n_nodi
                    for j = 1:2
                        prima_stima(i,j) = matrice_stima_iniziale(i,j);
                    end
                end

                % mass-spring optimization
                for iterazione = 1:n_iterazioni
                    for i = 1:n_nodi
                        contatore_vicini = 1;
                        for j = 1:n_nodi
                            if ((i ~= j) && (matrice_adiacenze(i,j) == 1)) %
cerco un nodo vicino al nodo i
                                distanza = sqrt( (stima(i).x - stima(j).x)^2 +
(stima(i).y - stima(j).y)^2 );
                                misura = (sqrt( (coordinate_iniziali(i).x -
coordinate_iniziali(j).x)^2 + (coordinate_iniziali(i).y -
coordinate_iniziali(j).y)^2 )) + (sigma_misura * randn(1));
                                differenza = distanza - misura;
                                x = stima(j).x - stima(i).x;
                                y = stima(j).y - stima(i).y;
                                if (distanza ~= 0)
                                    vx = x / distanza; % componente x del versore
                                    vy = y / distanza; % componente y del versore
```

```
                                forza11(contatore_vicini) = differenza * vx;
                                forza22(contatore_vicini) = differenza * vy;
                                contatore_vicini = contatore_vicini + 1;
                            end
                        end
                    end

                    forza_x = 0;
                    forza_y = 0;
                    % calcolo la forza da applicare al nodo i
                    for a = 1:(contatore_vicini-1)
                        forza_x = forza11(a) + forza_x;
                        forza_y = forza22(a) + forza_y;
                    end
                    % muovo il nodo i
                    stima(i).x =  stima(i).x + (forza_x/
(2*(contatore_vicini-1)));
                    stima(i).y =  stima(i).y + (forza_y/
(2*(contatore_vicini-1)));

                end

            end % fine della fase di ottimizzazione

            somma = 0;
            % colcolo il MSE di ogni nodo
            for i = 1:n_nodi
                calcolo(i) = (stima(i).x - coordinate_iniziali(i).x)^2 +
(stima(i).y - coordinate_iniziali(i).y)^2;
                somma = somma + calcolo(i);
            end
            MSE(m) = somma/n_nodi; % vettore lungo (m = iterazioni Monte
Carlo)

        end % fine Monte Carlo

        MSE_somma = 0;
        % calcolo MSE medio
        for i = 1:m
            MSE_somma = MSE_somma + MSE(i);
        end
        MSE_media = MSE_somma / m;

        % ho calcolato la media; ora voglio controllare i valori del vettore MSE
```

```
        % quelli sopra lo scostamento massimo li tolgo.
        indice_mse = 1;
        for i = 1:m
            if ( (MSE(i) - MSE_media) < (scostamento)^2 )
                MSE_new(indice_mse) = MSE(i);
                indice_mse = indice_mse +1;
            end
        end
        % ricalcolo la media degli MSE_new
        o = indice_mse-1; % numero di elementi del vettore MSE_new
        MSE_new_somma = 0;
        for i = 1:o
            MSE_new_somma = MSE_new_somma + MSE_new(i);
        end
        MSE_new_media = MSE_new_somma / o;

        % dopo che ho calcolato la media, calcolo il RMSE
        RMSE(indice_rmse) = sqrt(MSE_new_media);

        % calcolo affidabilità
        out = m - o; % numero di outlier
        out_percentuale(indice_out, indice_rmse) = out/m;
        reliability(indice_out, indice_rmse) = 1 - (out/m);
        indice_rmse = indice_rmse +1;

    end
    indice_rmse = indice_rmse -1;
    indice_out = indice_out +1;

    %creo matrice_coordinate (nx2) con le coordinate degli n nodi
    for colonna1 = 1:n_nodi
        matrice_coordinate_stimate(colonna1,1) = stima(colonna1).x;
    end
    for colonna2 = 1:n_nodi
        matrice_coordinate_stimate(colonna2,2) = stima(colonna2).y;
    end

% sigma_pos_iniziale assume 5 diversi valori (da 1 a 5 con passo unitario)
vettore_sigma(1) = 1;
vettore_sigma(2) = 2;
vettore_sigma(3) = 3;
vettore_sigma(4) = 4;
vettore_sigma(5) = 5;
```

```
% grafico RMSE (in ascissa ho sigma_pos_iniziale)
hold on
plot(vettore_sigma, RMSE);
hold off

end % fine calcoli
indice_out = indice_out -1;

title('Accuratezza media di stima della posizione condizionata alla
convergenza','Fontsize',16);
xlabel('Deviazione standard errore stima posizione iniziale [m]','fontsize',14);
ylabel('RMSE [m]','fontsize',14);
legend('deviazione standard errore misura = 0.0[m]','deviazione standard errore
misura = 0.2[m]','deviazione standard errore misura = 0.4[m]','deviazione
standard errore misura = 0.6[m]','deviazione standard errore misura = 0.8[m]');
figure;

% grafico affidabilità
for i = 1:5 % sigma_misura assume 5 valori
    hold on
    plot(vettore_sigma, reliability(i,:));
    hold off
end
title('Indice di affidabilità (reliability)','Fontsize', 16);
xlabel('Deviazione standard errore stima posizione iniziale [m]','Fontsize',14);
ylabel('Reliability','Fontsize',14);
legend('deviazione standard errore misura = 0.0[m]','deviazione standard errore
misura = 0.2[m]','deviazione standard errore misura = 0.4[m]','deviazione
standard errore misura = 0.6[m]','deviazione standard errore misura = 0.8[m]');
figure;

% GRAFICI

% ULTIMO GRAFICO DELLA SIMULAZIONE
% rete stimata + nodi in posizione originaria + prima stima
hold on
[X,Y] = gplot(DG, matrice_coordinate_stimate);
plot(X,Y);
for i = 1:n_nodi
plot(stima(i).x, stima(i).y, '*');
end
for a = 1:n_nodi
    plot(coordinate_iniziali(a).x, coordinate_iniziali(a).y, 'o', 'Markersize',
5);
```

```
end
for i = 1:n_nodi
    plot(prima_stima(i,1), prima_stima(i,2), 'd');
end
hold off
title ("Stima ottenuta - ultimo caso della simulazione");
xlabel('[m]');
ylabel('[m]');

%% FUNZIONE "CONFRONTO" PER STABILIRE CONNESSIONE
function ritorno = confronto (Z1, Z2, T)
d = sqrt( (Z1.x - Z2.x)^2 + (Z1.y - Z2.y)^2 );
if (d <= T)
    ritorno = 1;
else
    ritorno = 0;
end
end
```

Codice Matlab AFL - Caso del magazzino - misure TDoA

```
% Giacomo Giorgini, 2018, Tesi di Laurea Triennale (Cesena)
% Implementazione AFL - misure TDoA
% 1) Codice per simulare la creazione di nodi installati sul
% soffitto di un magazzino di dimensione 80x60 m^2.
% 2) Codice per la simulazione della stima iniziale
% 3) Codice per l'ottimizzazione (mass-spring) - sfruttando misure TDoA
% 4) Grafici

clear all
close all
clc

% lista paramentri
R = 40; % raggio di copertura dei nodi
r = 20; % un nodo ogni circa 20 metri
n_iterazioni = 30; % iterazioni fase di mass-spring
scostamento = 0.5;
monte_carlo = 5; % 1500 per le simulazioni

indice_out = 1;
for sigma_misura = 0:0.2:0.8

    indice_rmse = 1;
    for sigma_pos_iniziale = 0.5

        for m = 1:monte_carlo

                % le coordinate dei nodi sono contenute in una struttura
                coordinate_iniziali = struct('x',0,'y',0);

                sogliax = 80;
                sogliay = 60;
                % creazione lato sinistro del magazzino (80x60 m^2)
                soglia = 0;
                i = 1;
                while ( soglia < sogliay )
                    e1 = 1;
                    e2 = 2;
                    incertezza = e1 + (e2-e1)*rand; % possibile errore di
posizionamento manuale, [1,2] metri
                    if ( r+incertezza+soglia < sogliay )
```

```
        coordinate_iniziali(i).x = 0;
        coordinate_iniziali(i).y = r+incertezza + soglia;
        i = i+1;
     end
     soglia = r+incertezza + soglia;
end

% creazione lato destro
soglia = 0;
while ( soglia < sogliay )
    incertezza = e1 + (e2-e1)*rand;
    if ( r+incertezza+soglia < sogliay )
        coordinate_iniziali(i).x = sogliax;
        coordinate_iniziali(i).y = r+incertezza + soglia;
        i = i+1;
    end
    soglia = r+incertezza + soglia;
end

% creazione lato inferiore
soglia = 0;
while ( soglia < sogliax )
    incertezza = e1 + (e2-e1)*rand;
    if ( r+incertezza+soglia < sogliax )
        coordinate_iniziali(i).y = 0;
        coordinate_iniziali(i).x = r+incertezza + soglia;
        i = i+1;
    end
    soglia = r+incertezza + soglia;
end

% creazione lato superiore
soglia = 0;
while ( soglia < sogliax )
    incertezza = e1 + (e2-e1)*rand;
    if ( r+incertezza+soglia < sogliax )
        coordinate_iniziali(i).y = sogliay;
        coordinate_iniziali(i).x = r+incertezza + soglia;
        i = i+1;
    end
    soglia = r+incertezza + soglia;
end

% creazione asse orizzontale
```

```
                soglia = 0;
                while ( soglia < sogliax )
                    incertezza = e1 + (e2-e1)*rand;
                    if ( r+incertezza+soglia < sogliax )
                        coordinate_iniziali(i).y = sogliay/2;
                        coordinate_iniziali(i).x = r+incertezza + soglia;
                        i = i+1;
                    end
                    soglia = r+incertezza + soglia;
                end

                % creazione asse verticale
                soglia = 0;
                while ( soglia < sogliay )
                    incertezza = e1 + (e2-e1)*rand;
                    if ( r+incertezza+soglia < sogliay )
                        coordinate_iniziali(i).x = sogliax/2;
                        coordinate_iniziali(i).y = r+incertezza + soglia;
                        i = i+1;
                    end
                    soglia = r+incertezza + soglia;
                end
                i = i-1;

                n_nodi = i;

                matrice_coordinate_iniziali = zeros(n_nodi, 2);
                % creazione matrice (nx2) con le coordinate dei nodi
                for colonna1 = 1:n_nodi
                    matrice_coordinate_iniziali(colonna1,1) =
coordinate_iniziali(colonna1).x;
                end
                for colonna2 = 1:n_nodi
                    matrice_coordinate_iniziali(colonna2,2) =
coordinate_iniziali(colonna2).y;
                end

                matrice_adiacenze = zeros(n_nodi, n_nodi);
                % creazione matrice (nxn) delle adiacenze
                for i = 1:n_nodi
                    for j = 1:n_nodi
                        matrice_adiacenze(i,j) =
confronto(coordinate_iniziali(i), coordinate_iniziali(j), R);
                    end
```

```
                end

                DG = sparse(matrice_adiacenze);

                matrice_stima_iniziale = zeros(n_nodi, 2);
                % prima stima = posizione reale + errore gaussiano
                for i = 1:n_nodi
                    for j = 1:2
                        matrice_stima_iniziale(i,j) =
matrice_coordinate_iniziali(i,j) + (sigma_pos_iniziale * randn(1));
                    end
                end

                % la struttura stima verrà aggiornata ad ogni iterazione
                stima = struct('x',0,'y',0);
                for i = 1:n_nodi
                    stima(i).x = matrice_stima_iniziale(i, 1);
                    stima(i).y = matrice_stima_iniziale(i, 2);
                end
                % salvo le coordinate della stima iniziale
                for i = 1:n_nodi
                    for j = 1:2
                        prima_stima(i,j) = matrice_stima_iniziale(i,j);
                    end
                end

                % mass-spring optimization
                for h = 1:n_iterazioni
                    for i = 1:n_nodi
                        for j = 1:n_nodi
                            contatore_vicini = 1;
                            if ((i ~= j) && (matrice_adiacenze(i,j) == 1))
                                for t = 1:n_nodi
                                    if ((t ~= j) && (matrice_adiacenze(t,j) ==
1))
                                        if ((t ~= i) && (matrice_adiacenze(t,i)
== 1))

                                            d_ti = sqrt( (stima(t).x -
stima(i).x)^2 + (stima(t).y - stima(i).y)^2 );
                                            d_tj = sqrt( (stima(t).x -
stima(j).x)^2 + (stima(t).y - stima(j).y)^2 );
                                            tdoa_d = d_ti - d_tj;
```

```
                                        r_ti =
(sqrt( (coordinate_iniziali(t).x - coordinate_iniziali(i).x)^2 +
(coordinate_iniziali(t).y - coordinate_iniziali(i).y)^2 ));
                                        r_tj =
(sqrt( (coordinate_iniziali(t).x - coordinate_iniziali(j).x)^2 +
(coordinate_iniziali(t).y - coordinate_iniziali(j).y)^2 ));
                                        tdoa_r = (r_ti - r_tj) +
(sigma_misura * randn(1));

                                        if (tdoa_r > tdoa_d)

                                            x = stima(i).x - stima(t).x;
                                            y = stima(i).y - stima(t).y;

                                            vx = x / d_ti; % componente x del
versore
                                            vy = y / d_ti; % componente y del
versore

                                        else

                                            x = stima(t).x - stima(i).x;
                                            y = stima(t).y - stima(i).y;

                                            vx = x / d_ti;
                                            vy = y / d_ti;

                                        end
                                        diff = abs(tdoa_r - tdoa_d);
                                        forza11(contatore_vicini) = diff *
vx;
                                        forza22(contatore_vicini) = diff *
vy;
                                        contatore_vicini = contatore_vicini +
1;
                                    end
                                end
                            end

                            forza1 = 0;
                            forza2 = 0;
                            for a = 1:(contatore_vicini-1)
                                forza1 = forza11(a) + forza1;
                                forza2 = forza22(a) + forza2;
```

```
                                end
                                % spostamento del nodo
                                stima(i).x =  stima(i).x + (forza1/
(2*(contatore_vicini-1)));
                                stima(i).y =  stima(i).y + (forza2/
(2*(contatore_vicini-1)));

                            end
                        end
                    end
                end % fine ottimizzazione

                somma = 0;
                % colcolo il MSE di ogni nodo
                for i = 1:n_nodi
                    calcolo(i) = (stima(i).x - coordinate_iniziali(i).x)^2 +
(stima(i).y - coordinate_iniziali(i).y)^2;
                    somma = somma + calcolo(i);
                end
                MSE(m) = somma/n_nodi; % vettore lungo (m = iterazioni Monte
Carlo)

        end % fine Monte Carlo

        MSE_somma = 0;
        % calcolo MSE medio
        for i = 1:m
            MSE_somma = MSE_somma + MSE(i);
        end
        MSE_media = MSE_somma / m;

        % ho calcolato la media; ora voglio controllare i valori del vettore MSE
        % quelli sopra lo scostamento massimo li tolgo.
        indice_mse = 1;
        for i = 1:m
            if ( (MSE(i) - MSE_media) < (scostamento)^2 )
                MSE_new(indice_mse) = MSE(i);
                indice_mse = indice_mse +1;
            end
        end
        % ricalcolo la media degli MSE_new
        o = indice_mse-1; % numero di elementi del vettore MSE_new
        MSE_new_somma = 0;
        for i = 1:o
```

```
            MSE_new_somma = MSE_new_somma + MSE_new(i);
        end
        MSE_new_media = MSE_new_somma / o;

        % dopo che ho calcolato la media, calcolo il RMSE
        RMSE(indice_rmse) = sqrt(MSE_new_media);

        % calcolo affidabilità
        out = m - o; % numero di outlier
        out_percentuale(indice_out, indice_rmse) = out/m;
        reliability(indice_out, indice_rmse) = 1 - (out/m);
        indice_rmse = indice_rmse +1;

    end
    indice_rmse = indice_rmse -1;
    indice_out = indice_out +1;

    %creo matrice_coordinate (nx2) con le coordinate degli n nodi
    for colonna1 = 1:n_nodi
        matrice_coordinate_stimate(colonna1,1) = stima(colonna1).x;
    end
    for colonna2 = 1:n_nodi
        matrice_coordinate_stimate(colonna2,2) = stima(colonna2).y;
    end

% sigma_pos_iniziale assume 5 diversi valori (da 1 a 5 con passo unitario)
vettore_sigma(1) = 1;
vettore_sigma(2) = 2;
vettore_sigma(3) = 3;
vettore_sigma(4) = 4;
vettore_sigma(5) = 5;

% grafico RMSE (in ascissa ho sigma_pos_iniziale)
hold on
plot(vettore_sigma, RMSE);
hold off

end % fine calcoli
indice_out = indice_out -1;

title('Accuratezza media di stima della posizione condizionata alla
convergenza','Fontsize',16);
xlabel('Deviazione standard errore stima posizione iniziale [m]','fontsize',14);
ylabel('RMSE [m]','fontsize',14);
```

```
legend('deviazione standard errore misura = 0.0[m]','deviazione standard errore
misura = 0.2[m]','deviazione standard errore misura = 0.4[m]','deviazione
standard errore misura = 0.6[m]','deviazione standard errore misura = 0.8[m]');
figure;

% grafico affidabilità
for i = 1:5 % sigma_misura assume 5 valori
    hold on
    plot(vettore_sigma, reliability(i,:));
    hold off
end
title('Indice di affidabilità (reliability)','Fontsize', 16);
xlabel('Deviazione standard errore stima posizione iniziale [m]','Fontsize',14);
ylabel('Reliability','Fontsize',14);
legend('deviazione standard errore misura = 0.0[m]','deviazione standard errore
misura = 0.2[m]','deviazione standard errore misura = 0.4[m]','deviazione
standard errore misura = 0.6[m]','deviazione standard errore misura = 0.8[m]');
figure;

% GRAFICI

% rete stimata + nodi in posizione originaria + prima stima
hold on
[X,Y] = gplot(DG, matrice_coordinate_stimate);
plot(X,Y);
for i = 1:n_nodi
plot(stima(i).x, stima(i).y, '*');
end
for a = 1:n_nodi
    plot(coordinate_iniziali(a).x, coordinate_iniziali(a).y, 'o', 'Markersize',
5);
end
for i = 1:n_nodi
    plot(prima_stima(i,1), prima_stima(i,2), 'd');
end
hold off
title ("Stima ottenuta - ultimo caso della simulazione");
xlabel('[m]');
ylabel('[m]');

%% FUNZIONE "CONFRONTO" PER STABILIRE CONNESSIONE
function ritorno = confronto (Z1, Z2, T)
d = sqrt( (Z1.x - Z2.x)^2 + (Z1.y - Z2.y)^2 );
if (d <= T)
```

```
    ritorno = 1;
else
    ritorno = 0;
end
end
```

Codice Matlab AFL - Caso nuvola di droni - misure RSSI

```
% Giacomo Giorgini, 2018, Tesi di Laurea Triennale (Cesena)
% Implementazione AFL - misure RSSI
% 1) Codice per simulare la creazione di una
% nuvola di droni (25 droni, 100x100 m^2, topologia UNIFORME)
% 2) Codice per la prima fase (stima iniziale)
% 3) Codice per l'ottimizzazione (mass-spring) - sfruttando misure RSSI
% 4) Grafici

clear all
close all
clc

% lista parametri
n_nodi = 25; % numero droni
R = 40; % raggio di copertura di ogni nodo
n_iterazioni = 1000; % iterazioni fase di ottimizzazione
scostamento = 0.1;
monte_carlo = 2; % simulazioni effettuate con monte_carlo = 1000

centro_area_x = 50;
centro_area_y = 50;

indice_out = 1;
for sigma_misura = 0:0.2:0.2

    indice_rmse = 1;
    for sigma_iniziale = 0:1:1
        for m = 1:monte_carlo

            % creazione coordinate droni
            coordinate_iniziali(n_nodi) = struct('x',0,'y',0);

            coordinate_iniziali(1).x = 50 + (sigma_iniziale *randn(1));
            coordinate_iniziali(1).y = 50 + (sigma_iniziale *randn(1));

            coordinate_iniziali(2).x = 0 + (sigma_iniziale *randn(1));
            coordinate_iniziali(2).y = 100 + (sigma_iniziale *randn(1));

            coordinate_iniziali(3).x = 66 + (sigma_iniziale *randn(1));
            coordinate_iniziali(3).y = 66 + (sigma_iniziale *randn(1));

            coordinate_iniziali(4).x = 33 + (sigma_iniziale *randn(1));
```

```
coordinate_iniziali(4).y = 33 + (sigma_iniziale *randn(1));

coordinate_iniziali(5).x = 66 + (sigma_iniziale *randn(1));
coordinate_iniziali(5).y = 33 + (sigma_iniziale *randn(1));

coordinate_iniziali(6).x = 16 + (sigma_iniziale *randn(1));
coordinate_iniziali(6).y = 82 + (sigma_iniziale *randn(1));

coordinate_iniziali(7).x = 48 + (sigma_iniziale *randn(1));
coordinate_iniziali(7).y = 82 + (sigma_iniziale *randn(1));

coordinate_iniziali(8).x = 82 + (sigma_iniziale *randn(1));
coordinate_iniziali(8).y = 82 + (sigma_iniziale *randn(1));

coordinate_iniziali(9).x = 82 + (sigma_iniziale *randn(1));
coordinate_iniziali(9).y = 48 + (sigma_iniziale *randn(1));

coordinate_iniziali(10).x = 82 + (sigma_iniziale *randn(1));
coordinate_iniziali(10).y = 16 + (sigma_iniziale *randn(1));

coordinate_iniziali(11).x = 48 + (sigma_iniziale *randn(1));
coordinate_iniziali(11).y = 16 + (sigma_iniziale *randn(1));

coordinate_iniziali(12).x = 16 + (sigma_iniziale *randn(1));
coordinate_iniziali(12).y = 16 + (sigma_iniziale *randn(1));

coordinate_iniziali(13).x = 16 + (sigma_iniziale *randn(1));
coordinate_iniziali(13).y = 48 + (sigma_iniziale *randn(1));

coordinate_iniziali(14).x = 33 + (sigma_iniziale *randn(1));
coordinate_iniziali(14).y = 66 + (sigma_iniziale *randn(1));

coordinate_iniziali(15).x = 33 + (sigma_iniziale *randn(1));
coordinate_iniziali(15).y = 100 + (sigma_iniziale *randn(1));

coordinate_iniziali(16).x = 66 + (sigma_iniziale *randn(1));
coordinate_iniziali(16).y = 100 + (sigma_iniziale *randn(1));

coordinate_iniziali(17).x = 100 + (sigma_iniziale *randn(1));
coordinate_iniziali(17).y = 100 + (sigma_iniziale *randn(1));

coordinate_iniziali(18).x = 100 + (sigma_iniziale *randn(1));
coordinate_iniziali(18).y = 66 + (sigma_iniziale *randn(1));
```

```
            coordinate_iniziali(19).x = 100 + (sigma_iniziale *randn(1));
            coordinate_iniziali(19).y = 33 + (sigma_iniziale *randn(1));

            coordinate_iniziali(20).x = 100 + (sigma_iniziale *randn(1));
            coordinate_iniziali(20).y = 0 + (sigma_iniziale *randn(1));

            coordinate_iniziali(21).x = 66 + (sigma_iniziale *randn(1));
            coordinate_iniziali(21).y = 0 + (sigma_iniziale *randn(1));

            coordinate_iniziali(22).x = 33 + (sigma_iniziale *randn(1));
            coordinate_iniziali(22).y = 0 + (sigma_iniziale *randn(1));

            coordinate_iniziali(23).x = 0 + (sigma_iniziale *randn(1));
            coordinate_iniziali(23).y = 0 + (sigma_iniziale *randn(1));

            coordinate_iniziali(24).x = 0 + (sigma_iniziale *randn(1));
            coordinate_iniziali(24).y = 33 + (sigma_iniziale *randn(1));

            coordinate_iniziali(25).x = 0 + (sigma_iniziale *randn(1));
            coordinate_iniziali(25).y = 66 + (sigma_iniziale *randn(1));

            matrice_coordinate_iniziali = zeros(n_nodi, 2);
            %creo matrice_coordinate (nx2) con le coordinate degli n nodi
            for colonna1 = 1:n_nodi
                matrice_coordinate_iniziali(colonna1,1) =
coordinate_iniziali(colonna1).x;
            end
            for colonna2 = 1:n_nodi
                matrice_coordinate_iniziali(colonna2,2) =
coordinate_iniziali(colonna2).y;
            end

            matrice_adiacenze = zeros(n_nodi, n_nodi);
            %creo matrice (nxn) delle adiacenze
            for i = 1:n_nodi
                for j = 1:n_nodi
                    matrice_adiacenze(i,j) = confronto(coordinate_iniziali(i),
coordinate_iniziali(j), R);
                end
            end

            DG = sparse(matrice_adiacenze);

            % CALCOLO DEI 5 NODI DI RIFERIMENTO
```

```
% si è ipotizzato come nodo n_0 il nodo 23
[dist_0, path_0, pred_0] = graphshortestpath(DG, 23);

% ricerca n_1
check = 0;
n_1 = 0;
for i = 1:n_nodi
    if (dist_0(i) > check)
        check = dist_0(i);
        n_1 = i;
    end
end

[dist_1, path_1, pred_1] = graphshortestpath(DG, n_1);

% ricerca n_2
check = 0;
n_2 = 0;
for i = 1:n_nodi
    if (dist_1(i) > check)
        check = dist_1(i);
        n_2 = i;
    end
end

[dist_2, path_2, pred_2] = graphshortestpath(DG, n_2);

% ricerca n_3
j = 1;
check = 1000;
n_3 = 0;
for i = 1:n_nodi
    diff = abs(dist_1(i) - dist_2(i));
    if (diff < check)
        n_3 = i;
        check = diff;
        for q = 1:j
            possibili_nodi(q) = 0;
        end
    j = 1;
    else if ((diff == check))
        possibili_nodi(j) = i;
        j = j+1;
```

```
        end
    end
end

if (j > 1)
    j = j-1;
    soglia = 0;
    for i = 1:j
        elemento = possibili_nodi(i);
        somma = dist_1(elemento) + dist_2(elemento);
        if (somma > soglia)
            soglia = somma;
            n_3 = elemento;
        end
    end
end

[dist_3, path_3, pred_3] = graphshortestpath(DG, n_3);

% ricerca n_4
j = 1;
check = 1000;
n_4 = 0;
for i = 1:n_nodi
    diff = abs(dist_1(i) - dist_2(i));
    if (i ~= n_3)
        if (diff < check)
            n_4 = i;
            check = diff;
            for q = 1:j
                possibili_nodi(q) = 0;
            end
        j = 1;
        else if ((diff == check))
            possibili_nodi(j) = i;
            j = j+1;
            end
        end
    end
end

if (j > 1)
    j = j-1;
    soglia = 0;
```

```
    max_3_4 = 0;
    for i = 1:j
        elemento = possibili_nodi(i);
        max_3_4 = dist_3(elemento);
        if (n_4 ~= n_1 && n_4 ~= n_2 && n_4 ~= n_3)
            if (max_3_4 > soglia)
                soglia = max_3_4;
                n_4 = elemento;
            end
        end
    end
end

[dist_4, path_4, pred_4] = graphshortestpath(DG, n_4);

% ricerca n_5
j = 1;
check = 1000;
n_5 = 0;
for i = 1:n_nodi
    diff = abs(dist_1(i) - dist_2(i));
    if (i ~= n_4 && i ~= n_3)
        if (diff < check)
            n_5 = i;
            check = diff;
            for q = 1:j
                possibili_nodi(q) = 0;
            end
            j = 1;
        else if ((diff == check))
                possibili_nodi(j) = i;
                j = j+1;
            end
        end
    end
end

if (j > 1)
    j = j-1;
    soglia = 1000;
    for i = 1:j
        elemento = possibili_nodi(i);
        diff = abs(dist_3(elemento) - dist_4(elemento));
        if (diff < soglia)
```

```
                    soglia = diff;
                    n_5 = elemento;
                end
            end
        end

        [dist_5, path_5, pred_5] = graphshortestpath(DG, n_5);

        % riassegno i nodi di riferimento
        nodi(1) = n_1;
        nodi(2) = n_2;
        nodi(3) = n_3;
        nodi(4) = n_4;

        for i = 1:4
            if ( coordinate_iniziali(nodi(i)).x > (centro_area_x) &&
coordinate_iniziali(nodi(i)).y > (centro_area_y) )
                n_2 = nodi(i);
            end
            if ( coordinate_iniziali(nodi(i)).x < (centro_area_x) &&
coordinate_iniziali(nodi(i)).y < (centro_area_y) )
                n_1 = nodi(i);
            end
            if ( coordinate_iniziali(nodi(i)).x < (centro_area_x) &&
coordinate_iniziali(nodi(i)).y > (centro_area_y) )
                n_3 = nodi(i);
            end
            if ( coordinate_iniziali(nodi(i)).x > (centro_area_x) &&
coordinate_iniziali(nodi(i)).y < (centro_area_y) )
                n_4 = nodi(i);
            end
        end

        [dist_1, path_1, pred_1] = graphshortestpath(DG, n_1);
        [dist_2, path_2, pred_2] = graphshortestpath(DG, n_2);
        [dist_3, path_3, pred_3] = graphshortestpath(DG, n_3);
        [dist_4, path_4, pred_4] = graphshortestpath(DG, n_4);

        % FASE 1 _ AFL

        % la struttura stima verrà aggiornata ad ogni iterazione
        stima(n_nodi) = struct('x',0,'y',0);

        stima(n_5).x = 0;
```

```
            stima(n_5).y = 0;

            % calcolo coordinate polari
            for i = 1:n_nodi
                if (i ~= n_5)
                    y = dist_1(i) - dist_2(i);
                    x = dist_3(i) - dist_4(i);
                    ro(i) = dist_5(i) * R; % ro
                    angle(i) = atan2(y,x); % teta
                    stima(i).x = ro(i)*cos(angle(i));
                    stima(i).y = ro(i)*sin(angle(i));
                end
            end

            % creo matrice con le coordinate relative alla prima stima
            for colonna1 = 1:n_nodi
                matrice_prima_stima(colonna1,1) = stima(colonna1).x;
            end
            for colonna2 = 1:n_nodi
                matrice_prima_stima(colonna2,2) = stima(colonna2).y;
            end

                % mass-spring optimization
                for iterazione = 1:n_iterazioni
                    for i = 1:n_nodi
                        contatore_vicini = 1;
                        for j = 1:n_nodi
                            if ((i ~= j) && (matrice_adiacenze(i,j) == 1)) %
cerco un nodo vicino al nodo i
                                distanza = sqrt( (stima(i).x - stima(j).x)^2 +
(stima(i).y - stima(j).y)^2 );
                                misura = (sqrt( (coordinate_iniziali(i).x -
coordinate_iniziali(j).x)^2 + (coordinate_iniziali(i).y -
coordinate_iniziali(j).y)^2 )) + (sigma_misura * randn(1));
                                differenza = distanza - misura;
                                x = stima(j).x - stima(i).x;
                                y = stima(j).y - stima(i).y;
                                if (distanza ~= 0)
                                    vx = x / distanza; % componente x del versore
                                    vy = y / distanza; % componente y del versore
                                    forza11(contatore_vicini) = differenza * vx;
                                    forza22(contatore_vicini) = differenza * vy;
                                    contatore_vicini = contatore_vicini + 1;
                                end
```

```
                        end
                    end

                    forza_x = 0;
                    forza_y = 0;
                    % calcolo la forza da applicare al nodo i
                    for a = 1:(contatore_vicini-1)
                        forza_x = forza11(a) + forza_x;
                        forza_y = forza22(a) + forza_y;
                    end
                    % muovo il nodo i
                    stima(i).x =  stima(i).x + (forza_x/
(2*contatore_vicini));
                    stima(i).y =  stima(i).y + (forza_y/
(2*contatore_vicini));

                end

            end % fine della fase di ottimizzazione

            % creo matrice con le coordinate stimate
            for colonna1 = 1:n_nodi
                matrice_stima(colonna1,1) = stima(colonna1).x;
            end
            for colonna2 = 1:n_nodi
                matrice_stima(colonna2,2) = stima(colonna2).y;
            end

            % al fine di valutare il RMSE bisogna calcolare le
            % coordinate globali

            % rotazione del sistema
            x1 = coordinate_iniziali(n_5).x;
            y1 = coordinate_iniziali(n_5).y;
            x2 = coordinate_iniziali(n_4).x;
            y2 = coordinate_iniziali(n_4).y;
            alfa1 = atan2((y2-y1),(x2-x1));

            x11 = stima(n_5).x;
            y11 = stima(n_5).y;
            x22 = stima(n_4).x;
            y22 = stima(n_4).y;
            alfa2 = atan2((y22-y11),(x22-x11));
```

```
                alfa = alfa1 - alfa2;

                stima_r(n_nodi) = struct('x',0,'y',0);
                for i = 1:n_nodi
                    stima_r(i).x = stima(i).x * cos(alfa) - stima(i).y *
sin(alfa);
                    stima_r(i).y = stima(i).x * sin(alfa) + stima(i).y *
cos(alfa);
                end

                stima_globale(n_nodi) = struct('x',0,'y',0);

                for i = 1:n_nodi
                    stima_globale(i).x = stima_r(i).x +
coordinate_iniziali(n_5).x;
                    stima_globale(i).y = stima_r(i).y +
coordinate_iniziali(n_5).y;
                end

                % creo matrice con le coordinate globali
                for colonna1 = 1:n_nodi
                    matrice_stima_globale(colonna1,1) =
stima_globale(colonna1).x;
                end
                for colonna2 = 1:n_nodi
                    matrice_stima_globale(colonna2,2) =
stima_globale(colonna2).y;
                end

                somma = 0;
                for i = 1:n_nodi
                    calcolo(i) = (stima_globale(i).x -
coordinate_iniziali(i).x)^2 + (stima_globale(i).y - coordinate_iniziali(i).y)^2;
                    somma = somma + calcolo(i);
                end
                MSE(m) = somma/n_nodi;

            end %fine montecarlo

            MSE_somma = 0;
            for i = 1:m
                MSE_somma = MSE_somma + MSE(i);
            end
            MSE_media = MSE_somma / m;
```

```
            % ho calcolato la media; ora voglio controllare i valori del vettore
            % MSE: quelli sopra lo scostamento desiderato li tolgo.
            indice_mse = 1;
            for i = 1:m
                if ( (MSE(i) - MSE_media) < (scostamento)^2 )
                    MSE_new(indice_mse) = MSE(i);
                    indice_mse = indice_mse +1;
                end
            end
            % ricalcolo la media
            o = indice_mse-1;
            MSE_new_somma = 0;
            for i = 1:o
                MSE_new_somma = MSE_new_somma + MSE_new(i);
            end
            MSE_new_media = MSE_new_somma / o;

            % dopo che ho calcolato la media, calcolo il RMSE
            RMSE(indice_rmse) = sqrt(MSE_new_media);

            % calcolo numero di outlayer
            out = m - o; % numero di outlier
            out_percentuale(indice_out, indice_rmse) = out/m;
            reliability(indice_out, indice_rmse) = 1 - (out/m);
            indice_rmse = indice_rmse +1;
    end
    indice_rmse = indice_rmse -1;
    indice_out = indice_out +1;



vettore_sigma(1) = 0;
vettore_sigma(2) = 1;
% grafico RMSE
hold on
plot(vettore_sigma, RMSE);
hold off

end % fine calcoli
indice_out = indice_out -1;

title('Accuratezza media di stima della posizione condizionata alla
convergenza','Fontsize',16);
```

```
xlabel('Deviazione standard posizione iniziale [m]','fontsize',14);
ylabel('RMSE [m]','fontsize',14);
legend('deviazione standard errore misura = 0.0[m]','deviazione standard errore
misura = 0.2[m]');
figure;

% grafico affidabilità
for i = 1:2 % sigma_iniziale assume 2 valori
    hold on
    plot(vettore_sigma, reliability(i,:));
    hold off
end
title('Indice di affidabilità (reliability)','Fontsize', 16);
xlabel('Deviazione standard posizione iniziale [m]','Fontsize',14);
ylabel('Reliability','Fontsize',14);
legend('deviazione standard errore misura = 0.0[m]','deviazione standard errore
misura = 0.2[m]');
figure;

% GRAFICI RELATIVI ALL'ULTIMA SIMULAZIONE EFFETTUATA

% grafo iniziale
hold on
[X,Y] = gplot(DG, matrice_coordinate_iniziali);
plot(X,Y);
for i = 1:n_nodi
    plot(coordinate_iniziali(i).x, coordinate_iniziali(i).y, '*');
end
plot(coordinate_iniziali(n_1).x,coordinate_iniziali(n_1).y, 'o', 'Markersize',
14);
plot(coordinate_iniziali(n_2).x,coordinate_iniziali(n_2).y, 'o', 'Markersize',
14);
plot(coordinate_iniziali(n_3).x,coordinate_iniziali(n_3).y, 'o', 'Markersize',
14);
plot(coordinate_iniziali(n_4).x,coordinate_iniziali(n_4).y, 'o', 'Markersize',
14);
plot(coordinate_iniziali(n_5).x,coordinate_iniziali(n_5).y, 'o', 'Markersize',
14);
hold off
title ('Grafo iniziale - ultima simulazione effettuata');
xlabel('[m]');
ylabel('[m]');
figure;
```

```
% stima globale
hold on
[X,Y] = gplot(DG, matrice_stima_globale);
plot(X,Y);
for i = 1:n_nodi
    plot(stima_globale(i).x, stima_globale(i).y, '*');
end
hold off
title ('Stima globale - ultima simulazione effettuata');
xlabel('[m]');
ylabel('[m]');



%% FUNZIONE "CONFRONTO" PER STABILIRE CONNESSIONE
function ritorno = confronto (Z1, Z2, T)
d = sqrt( (Z1.x - Z2.x)^2 + (Z1.y - Z2.y)^2 );
if (d <= T)
    ritorno = 1;
else
    ritorno = 0;
end
end
```