Università degli studi di Verona
Facoltà di Scienze MM.FF.NN.

Tesi di Laurea Specialistica in

Scienze dell'informatica

Progettazione della politica di trasmissione wireless in un problema di controllo di formazione mediante co-simulazione

Relatore
Dott. Davide Quaglia

Correlatore
Dott. Riccardo Muradore

Candidato
Giacomo Lanza

Anno Accademico 2010/2011

Università degli studi di Verona
Facoltà di Scienze MM.FF.NN.

Tesi di Laurea Specialistica in

Scienze dell'informatica

Progettazione della politica di trasmissione wireless in un problema di controllo di formazione mediante co-simulazione

Relatore
Dott. Davide Quaglia

Correlatore
Dott. Riccardo Muradore

Candidato
Giacomo Lanza

Anno Accademico 2010/2011

Sommario

In questa tesi viene proposto un algoritmo per ottimizzare la potenza trasmissiva in un contesto di controllo di formazione leader-following.

In letteratura il controllo di formazione viene sempre trattato escludendo la parte riguardante la trasmissione dei dati tra i vari dispositivi. C'è da chiedersi come può incidere l'introduzione della rete a livello di controllo di formazione. Siamo qui di fronte ad un particolare caso di networked control system (NCS) in cui controllori e parti controllate sono collegati da reti digitali condivise. La simulazione di NCS gioca un ruolo fondamentale in questo campo. Un simulatore dovrebbe rappresentare sia la parte di controllo/impianto che gli aspetti di networking, ma è difficile che lo stesso strumento riesca a rappresentarli entrambi. Per questo viene adottata la stategia della co-simulazione con la quale si riescono a far lavorare assieme più strumenti in cui parti dell'intero sistema vengono simulate. Nel nostro caso utilizzeremo la co-simulazione Matlab/Simulink - SCNSL. Con il primo si modellerà la parte di controllo mentre con il secondo verrà simulata la parte di comunicazione.

In particolare SCNSL è una libreria di SystemC che permette di simulare gli aspetti di rete. Tramite il lavoro effettuato per questa tesi, tale libreria è stata estesa per permettere di simulare fedelmente una rete wireless. Sono stati implementati due modelli di canale di comunicazione wireless: Shared Channel e Delayed Shared Channel. I due canali si differenziano nel modo in cui modellano il tempo di propagazione del segnale. Nel canale Shared esso è costante e indipendente dalla posizione dei nodi nella rete. Nel canale Delayed Shared esso è proporzionale alla distanza dei nodi e alla velocità di propagazione del segnale fisico.

Lo strumento risultante potrà essere usato per progettare scenari di controllo di formazione in ambiente wireless. Vengono quindi proposti dei testcase per osservare come la potenza trasmissiva in una rete wireless influisca sulle prestazioni del controllo. Tali testcase vengono eseguiti e analizzati grazie a degli specifici tool che permettono di svolgere in automatico un elevato numero di simulazioni calcolando e memorizzando i risultati ottenuti.

Dall'analisi effettuata sui risultati, si nota che si ottengono prestazioni migliori a livello di controllo quando l'interferenza trasmissiva tra i veicoli è minima. Si è sviluppato quindi un algoritmo che ottimizza la potenza

trasmissiva in modo da creare la minor interferenza possibile nella rete. Ogni nodo in base alla distanza dei propri vicini regola la potenza trasmissiva in modo da garantire un margine di raggiungibilità per ciascuno di loro anche in presenza di improvvisi spostamenti.

L'adattamento della potenza trasmissiva come soluzione migliorativa per il controllo di formazione è un esempio di allargamento dello spazio di progetto di networked control system agli aspetti trasmissivi rispetto al tradizionale approccio focalizzato sulla progettazione del solo controllore. L'estensione dello strumento di simulazione e la sua applicazione ad uno scenario di controllo di formazione mostra che tale strumento può essre usato efficacemente per la realizzazione di NCS in cui sia gli aspetti di controllo sia quelli trasmissivi possono essere oggetto dell'esplorazione di progetto.

Indice

Capitolo 1

Introduzione

Negli ultimi dieci anni, il controllo di formazione di più veicoli ha ricevuto molta attenzione da parte delle comunità di controllo a causa delle molteplici applicazioni basate su esso, come sorveglianza, ricerca e soccorso in ambienti pericolosi, esplorazioni, trasporto di grandi oggetti e controllo di satelliti. Tale attenzione deriva dal fatto che ci sono molti vantaggi di tali sistemi rispetto ad un sistema ad entità singola, tra cui una maggiore flessibilità, adattabilità ad ambienti sconosciuti e robustezza.

Numerosi risultati di ricerca sono riportati sulla base di tre metodologie di controllo di formazione principali, come: behavior-based [1], struttura virtuale [2] e approccio leader-following [3]. In particolare, l'approccio leader-following è stato ampiamente utilizzato a causa della semplicità, scalabilità e affidabilità. Questo approccio consiste nel fatto che i veicoli inseguitori devono stare ad una determinata distanza e orientazione rispetto ad un veicolo leader, mentre esso compie una certa traiettoria. In tale approccio, la velocità e la posizione del leader devono essere misurate e comunicate ai suoi follower. I metodi basati su leader-following, quindi, presumono che le velocità angolari e la posizione del leader siano conosciute dai follower per permettere un corretto funzionamento dei loro controllori di formazione locali. Tuttavia, potrebbe essere difficoltoso risolvere tale problema per osservazione diretta. La comunicazione wireless permette di risolvere questo problema fornendo un modo efficiente di condividere informazioni tra i veicoli.

Un insieme di veicoli per i quali è richiesto un controllo di formazione, è un particolare caso di networked control system (NCS) cioè sistemi in cui controllori e parti controllate sono collegati da reti digitali condivise. Negli ultimi anni i networked control system hanno guadagnato una notevole importanza in una vasta serie di applicazioni, tra le quali troviamo l'automazione industriale, la robotica, i veicoli comandati a distanza e l'esplorazione.

Nel controllo di formazione con comunicazione esplicita, come in genera-

le negli NCS, la rete è un aspetto critico. Il canale di comunicazione, infatti, può alterare notevolmente la qualità del controllo. Le prestazioni sono principalmente influenzate da due problemi: tempi di comunicazione e perdita di pacchetti. Il primo problema è ben noto ed è stato affrontato da molti autori che hanno cercato di prevedere il ritardo di una sequenza di pacchetti attraverso un modello matematico del ritardo basato su una connessione Internet. Tuttavia, questo approccio non è adatto per lo sviluppo di modelli di ritardo generali.

La simulazione di NCS gioca un ruolo fondamentale in questo campo. Un simulatore dovrebbe catturare e rappresentare sia la parte di controllo/impianto che gli aspetti della comunicazione. Matlab/Simulink è lo strumento standard per la progettazione e la simulazione di sistemi dinamici. È difficile, purtroppo, simulare anche il comportamento della rete all'interno di questo strumento, pertanto è necessario integrare Matlab/Simulink con qualche altro strumento che consente la simulazione di rete. Questo si può fare tramite SCNSL, una libreria di SystemC, in grado di simulare fedelmente una rete a pacchetto.

Il punto cruciale di questo approccio è il concetto di co-simulazione. La co-simulazione è una strategia per fare lavorare insieme diversi strumenti su cui parti del sistema complessivo vengono simulate. La strategia di co-simulazione Matlab/Simulink - SCNSL permette di simulare allo stesso tempo la parte di controllo e gli aspetti di networking.

L'idea che ha portato allo sviluppo di questa tesi è quella di creare un ambiente in grado di simulare un controllo di formazione tra veicoli che si scambiano messaggi tramite la rete simulata da SCNSL. Si è pensato dunque di sviluppare un sistema di simulazione che utilizzi la metodica della co-simulazione per simulare in modo combinato e sincronizzato il controllore, utilizzando il linguaggio Matlab/Simulink, e la rete, utilizzando SCNSL.

La libreria SCNSL ha la capacità di simulare diversi scenari di rete con la possibilità di scegliere tra una vasta gamma di canali da utilizzare. Mancano però dei modelli accurati di canale wireless, sia nel caso di trasmissione radio, sia nel caso di trasmissione acustica. In particolare in questo secondo caso la velocità di propagazione incide in modo determinante sul ritardo di trasmissione. Inoltre per rispecchiare in modo accurato le dinamiche di una rete wireless, la libreria necessita di un meccanismo per simulare il movimento dei nodi. Per tenere conto del movimento dei nodi nella rete e del ritardo di trasmissione dovuto alle loro posizioni la libreria SCNSL deve essere estesa.

Lo strumento risultante potrà essere efficacemente utilizzato per progettare degli scenari di controllo di formazione in ambiente wireless. Verranno quindi effettuate simulazioni e studi su questo tipo di problema, modellando la dinamica dei veicoli in Matlab/Simulink e facendoli comunicare attraverso la rete a pacchetto simulata tramite SCNSL.

In particolare, simulando scenari di trasmissione wireless con veicoli in formazione si potrà osservare come la potenza trasmissiva è un aspetto critico del sistema. Infatti se la potenza trasmissiva è troppo elevata si possono creare interferenze tra trasmissioni indipendenti con conseguente impedimento nel trasferimento di informazioni; se al contrario essa non è sufficiente per raggiungere nodi che si stanno allontanando si rischia la loro definitiva perdita. È necessario quindi sviluppare un algoritmo allo scopo di migliorare la comunicazione tra i veicoli basandosi sull'ottimizzazione della potenza trasmissiva. Quest'algoritmo avrà lo scopo di modificare la potenza trasmissiva di ogni veicolo in relazione alla sua posizione nella formazione. L'analisi delle prestazioni del controllo di formazione mediante adattamento della potenza trasmissiva ha anche la funzione di validare l'efficacia delle estensioni apportate all'ambiente di simulazione.

La tesi è strutturata in nove capitoli. Il secondo capitolo è dedicato allo stato dell'arte, nel quale verranno descritti gli argomenti fondamentali per lo sviluppo della tesi. Si inizia con una descrizione della comunicazione wireless e delle sue peculiarità per passare poi ai sistemi NCS, alla co-simulazione e al controllo di formazione. Il terzo capitolo affronta gli obiettivi fissati all'inizio di questo lavoro. Il quarto capitolo descrive l'estensione della libreria SCNSL e la sua validazione tramite l'introduzione di test appositi. Il quinto capitolo è focalizzato sulla descrizione dei tool utilizzati in questo lavoro di tesi per simulare gli scenari di controllo di formazione mentre il sesto capitolo riporta e analizza i risultati della validazione dei tool appena citati. Successivamente, nel settimo capitolo, si tratterà l'algoritmo per l'ottimizzazione della potenza trasmissiva. Il tutto verrà validato mediante l'analisi dei test creati. L'ottavo capitolo analizza alcune istanze di configurazione dell'algoritmo precedente che ne modificano alcuni aspetti con l'obiettivo di migliorarne le performance. Infine, il nono capitolo tratta le conclusioni derivanti dall'analisi dei risultati ottenuti.

Capitolo 2

Stato dell'arte

In questo capitolo viene descritto e analizzato lo stato dell'arte dei principali ambiti trattati in questa tesi. In sezione 2.1 viene trattata la comunicazione wireless compresa una descrizione dei protocolli utilizzati in questo ambiente e una analisi sull'utilizzo dell'adattamento della potenza trasmissiva in reti di sensori. In sezione 2.2 vengono introdotte le caratteristiche principali dei Networked Control Systems. Nella sezione 2.3 verrà descritta la metodica della co-simulazione tra SCNSL e Matlab ed infine in sezione 2.4 verrà illustrato il problema del controllo di formazione, base di partenza per il lavoro svolto in questa tesi.

2.1 Comunicazioni wireless

In informatica e telecomunicazioni il termine wireless (dall'inglese senza fili) indica una comunicazione tra dispositivi elettronici che non fa uso di cavi. Per estensione sono detti wireless i rispettivi sistemi o dispositivi di comunicazione che implementano tale modalità di comunicazione. I sistemi tradizionali basati su connessioni cablate sono invece detti wired. Generalmente il wireless utilizza onde radio a bassa potenza; tuttavia la definizione si estende anche ai dispositivi, meno diffusi, che sfruttano la radiazione infrarossa o il laser. La comunicazione e i sistemi wireless trovano diretta applicazione nelle reti wireless di telecomunicazioni, fisse e mobili e più in generale nelle radiocomunicazioni. Ogni sistema di comunicazione wireless è composto da un trasmettitore, un ricevitore e dagli elementi deputati all'irradiazione elettromagnetica ovvero alla trasduzione elettro-elettromagnetica e viceversa ovvero le antenne, i laser, i fotorivelatori. La comunicazione può essere monodirezionale, bidirezionale half-duplex, bidirezionale full-duplex. Le frequenze elettromagnetiche utilizzate variano a seconda dello standard implementato.

2.1.1 Il progetto IEEE 802

Nel 1962 l'*American Institute of Electrical Engineers (AIEE)* e l'*Institute of Radio Engineers (IRE)* si unirono per formare l'*Institute of Electrical and Electronics Engineers (IEEE)*. La Costituzione dell'IEEE definisce le finalità dell'organizzazione come una continua ricerca orientata verso il progresso della teoria e della pratica soprattutto nei campi elettronico, elettrotecnico, e di molti rami dell'ingegneria, delle arti e delle scienze.

Nel perseguire tali obiettivi la IEEE pubblica articoli per i maggiori giornali e riviste ed è sponsor ufficiale di oltre 300 conferenze internazionali in tutto il mondo. L' IEEE gioca un ruolo fondamentale non solo per l'industria ma anche per il campo universitario in quanto più di 1430 studenti appartengono alla comunità di ricerca dell'IEEE in più di 80 Università. L'autorizzazione alla formazione del progetto IEEE 802 fu richiesta nel 1979 per sviluppare degli standard per le tecnologie emergenti. La IEEE favorì lo sviluppo di apparecchiature per local area networking per fare in modo che diversi fornitori potessero lavorare insieme. In aggiunta, la IEEE LAN standards fornisce un obbiettivo comune ai fornitori per poter raggiungere un mercato relativamente grande ed evitare di sviluppare architetture proprietarie. Questo, a sua volta, comporta un grosso beneficio in termini economici che si esprime in un abbassamento dei costi di sviluppo e produzione su larga scala.

La famiglia di standard IEEE 802 fa riferimento al local area network e al metropolitan area network. I servizi e i protocolli specificati si riferiscono agli ultimi due livelli (Data link e fisico) del modello di rete di riferimento OSI. Infatti, IEEE 802 suddivide il livello Data link in due sotto-livelli chiamati Logical Link Control (LLC) e Media Access Control (MAC). La famiglia di standard IEEE 802 è mantenuto dalla commissione IEEE 802 LAN/MAN Standard Committee (LMSC). I gruppi di sviluppo più conosciuti sono i seguenti:

- IEEE 802.1 Bridging (Networking) e Network Management
- IEEE 802.3: Ethernet
- IEEE 802.11: Wireless LAN (WLAN)
- IEEE 802.15: Wireless Personal Area Network (PAN)
- IEEE 802.16: Broadband Wireless Access (WiMAX)

2.1.2 Lo standard 802.15.4

L'IEEE 802.15.4 è uno standard che definisce i livelli MAC e fisico per la comunicazione wirless low-power e low-rate utilizzata nelle reti di sensori.

Molti dispositivi 802.15.4 sono predisposti per operare attraverso batterie quindi il problema dell'efficienza energetica è il punto chiave per la

definizione del livello MAC 802.15.4. I dispositivi 802.15.4 possono spegnere i loro ricetrasmettitori radio quando non hanno frame da spedire e risvegliarsi per un breve periodo esclusivamente per i beacon frame. Se i beacon frame avvisano il dispositivo per una richiesta di ricezione dei dati viene spedito un Data Request frame e il ricevitore si accende per tutto il tempo necessario alla ricezione dei dati. Nel caso in cui ci sia da ricevere dati da molti dispositivi, in accordo con lo standard 802.15.4, essi provano ad effettuare quasi contemporaneamente la trasmissione dei frame creando così a una maggior probabilità di collisione che porta ad un ulteriore consumo energetico e ad un ulteriore ritardo nella trasmissione.

Esistono due tipi di dispositivi nelle reti WPAN 802.15.4: full-function device (FFD) e il reduce-function device (RFD). L'FFD possiede più capacità computazionali e più energia rispetto all'RFD e quindi ha la responsabilità di essere il coordinatore della rete PAN. Un RFD è da utilizzare per semplici applicazioni come sensore passivo o come uno switch in quanto minimizza le risorse e la capacità di memorizzare dati. Un dispositivo FFD può comunicare con i dispositivi RFD e altri FFD mentre l'RFD può parlare esclusivamente con altri RFD. Un dispositivo FFD viene eletto per essere il coordinatore della rete PAN. Il coordinatore ha i compiti di trasmettere beacon frame, schedulare l'allocazione del canale e associare i nuovi dispositivi con quelli già presenti nella rete.

Livello fisico

I servizi, che il livello fisico offre, intervengono sull'accensione e sullo spegnimento del modulo radio. Questa opzione permette al sistema un risparmio d'energia ed effettua inoltre la scelta del canale di comunicazione migliore, calcolando una stima sull'occupazione del canale e del rapporto segnale/rumore SNR, diminuendo così la probabilità di errori in trasmissione. Infine modula e demodula i segnali in ricezione e in trasmissione. Il livello fisico opera nella Banda ISM utilizzando tre possibili bande libere:

- `868-868.6 Mhz:` banda utilizzata nella maggior parte dei paesi europei con un bitrate di 20kbps disponendo di un solo canale;
- `902-928 Mhz:` banda utilizzata nel continente oceanico e nel continente americano, offrendo un data-rate di 40Kbps e 10 canali disponibili;
- `2400-2483.5 Mhz:` banda utilizzabile in quasi tutto il globo con un data-rate massimo di 250 Kbps e 16 canali a disposizione.

La modulazione del segnale più diffusa è il DSSS utilizzando le tecniche BPSK o QPSK, anche se recentemente sono state introdotte nuove modulazioni come la PSSS. Il datagram a livello fisico è formato da una serie di campi: all'inizio si trova il preambolo, formato da 32 bit, con finalità di sincronizzazione tra i nodi. Successivamente viene utilizzato un ottetto

(11100101) prefissato che funziona da indicatore di inizio pacchetto. Segue un campo physical header che indica la lunghezza del payload (Phisical Service Data Units) che può avere una lunghezza massima di 127 bytes.

Livello MAC

Il secondo livello (MAC) offre servizi quali la possibilità di creare una rete PAN, la trasmissione dei beacon e l'accesso al canale tramite il protocollo CSMA/CA. Questo livello supporta algoritmi di cifratura basata su AES-128 per la sicurezza dei dati, gestisce inoltre l'handshake cioè gli Acknowledge per la ritrasmissione dei dati in caso di mancata o erronea ricezione di questi ultimi. Infine calcola e verifica l'integrità della PDU (Protocol Data Unit, l'unità di informazione scambiata tra due entità).

Inoltre può supportare reti fino ad un massimo di 65536 nodi utilizzando un indirizzamento fino a 16 bit. Il livello MAC prevede una struttura chiamata superframe. Questa frame è costruita dal coordinatore della rete ed è contenuta tra due messaggi chiamati beacon. I beacon contengono informazioni che possono essere utilizzate per la sincronizzazione dei dispositivi, per l'identificazione della rete, per descrivere la struttura della superframe e la periodicità della loro spedizione. La superframe è divisa in 16 slot temporali di uguale grandezza, dove il beacon frame è trasmesso nel primo e nell'ultimo slot di ognuna. Si possono avere due tipi di superframe:

- senza GTS (Guaranteed Time Slot);
- con GTS.

Quando viene inviata una superframe senza GTS, l'accesso al canale è regolarizzato dal protocollo CSMA/CA dove ogni dispositivo deve competere con gli altri per assicurarsi l'accesso ad uno slot. Questo periodo è chiamato CAP (Contention Access Period).

In altri tipi di applicazioni, invece, si necessita di costruire reti tali da garantire a tutti i nodi di poter trasmettere. E' il caso di superframe con GTS, la quale dedica fino ad un massimo di sette slot temporali, detti CFP (Contention-FreePeriod), a determinati nodi. In questo periodo possono comunicare solamente i nodi prestabiliti e l'accesso al canale non è più CSMA/CA .

Protocollo CSMA/CA

In uno scenario wireless si può pensare a due dispositivi che desiderano trasmettere un pacchetto nello stesso istante. Se in quel momento il canale non è in uso e quindi libero, i due dispositivi trasmettono i rispettivi pacchetti. I segnali quindi interferiranno creando la cosiddetta **collisione** la quale non causa alcun danno fisico ai dispositivi ma impedisce la corretta ricezione dei

pacchetti. Per evitare che i dispositivi trasmettino simultaneamente e quindi evitare le collisioni si utilizza il protocollo CSMA/CA (acronimo inglese di `Carrier Sense Multiple Access with Collision Avoidance`, ovvero accesso multiplo tramite rilevamento della portante che evita collisioni).

In telecomunicazioni CSMA/CA è un protocollo utilizzato nelle reti WI-FI come evoluzione del protocollo CSMA con ulteriori accorgimenti per ridurre le collisioni ed alternativo al CSMA/CD. L'algoritmo è comunemente adottato come algoritmo di accesso multiplo, secondo varianti implementative differenti, negli standard IEEE 802.11 e IEEE 802.15.4.

Per evitare le collisioni nel momento in cui una stazione vuole tentare una trasmissione ascolta semplicemente il canale (`Listen-before-Transmit`). Se il canale risulta libero (`idle`) attende per un certo lasso di tempo (`DIFS Distribuited Inter Frame Space`). Se il canale continua ad essere libero la stazione trasmette il pacchetto e successivamente attende un tempo SIFS (`Short Inter Frame Space`) di durata inferiore al DIFS per la ricezione di un ACK di conferma dell'avvenuta ricezione del pacchetto da parte della stazione ricevente. In tutto questo periodo di tempo le altre stazioni trovando il canale occupato non potranno trasmettere evitando così di trasmettere e produrre così collisioni (se l'ACK non può essere rilevato da una stazione nascosta, per evitare il rischio di trasmissione e quindi di collisione sull'ACK da parte di questa è possibile implementare un meccanismo di attesa da parte della stazione per un tempo almeno pari alla somma del SIFS e dell'ACK). Se il canale invece è occupato in trasmissione oppure ci sono state altre prenotazioni da parte di altre stazioni, la stazione attiva un timer di durata casuale (detto tempo di `backoff`) che viene decrementato solo durante i periodi di inattività del canale e congelato (`freezed backoff`) durante i restanti periodi di trasmissione sul canale da parte di tutte le altre stazioni. Quando il timer arriva a zero la stazione fa un altro tentativo di trasmissione. Tale tipologia di accesso è detto Accesso Base. In IEEE 802.11 l'intervallo di tempo casuale CW di back-off estratto è limitato dalla seguente disuguaglianza:

$$0 < CW < 2^N(CW_m + 1) - 1$$

con CW (`Collision Window`) compresa tra un valore minimo (CW_min) ed un valore massimo (CW_max) ed N numero progressivo di tentativo di trasmissione (retry). L'obiettivo di Collision Avoidance si ottiene dunque per via statistica (e non deterministica) proprio grazie all'estrazione del numero casuale compreso in questo intervallo di tempo, data la scarsa probabilità che due o più stazioni estraggano lo stesso numero di back-off, che è tanto minore quanto maggiore è l'intervallo possibile di estrazione, il quale aumenta a sua volta all'aumentare del numero N di tentativi di ritrasmissione.

Spesso il protocollo CSMA/CA è usato congiuntamente alla tecnica RTS-/CTS (prenotazione del canale) per affrontare il problema del cosiddetto

`hidden node`, ovvero il fatto che una stazione A che trasmette verso una stazione B può non essere in grado (ad esempio a causa della distanza) di rivelare una stazione C anch'essa impegnata in una comunicazione con B, generando così collisioni in ricezione su B.

Modalità di trasferimento

La trasmissione dei dati può avvenire in tre modi differenti a seconda di chi spedisce i dati. La prima modalità si riferisce al trasferimento dati dal nodo al coordinatore, la seconda da coordinatore a nodo e la terza tra due nodi. Le prime due modalità possono essere utilizzate in tipologia di reti a stella, mentre per le reti mesh possono essere utilizzate tutte e tre le modalità.

E' possibile utilizzare due meccanismi differenti per la trasmissione dei dati:

- trasmissioni basate sui beacon: Beacon-Enable;
- trasmissione senza l'utilizzo di beacon: Beacon-Disable.

Con una rete beacon-enabled, quando un nodo deve trasferire i dati ad un coordinatore, ascolta il beacon e si sincronizza alla superframe. Il nodo trasmette il relativo pacchetto al coordinatore il quale, opzionalmente dopo aver ricevuto i dati, trasmette un pacchetto di conferma dell'avvenuta ricezione al dispositivo (ACK). In una rete beacon-disable, quando un dispositivo desidera trasferire dei dati al coordinatore, trasmette semplicemente utilizzando il protocollo di accesso al canale CSMA/CA. Anche in questo caso il coordinatore dopo la ricezione dei dati trasmette un ACK opzionale.

La comunicazione diventa più articolata quando è il coordinatore che necessita di comunicare con un nodo. Se la rete è beacon-enabled, il coordinatore indica nel beacon la necessità di trasferire dati ad un dispositivo, il quale risponde attraverso una PDU chiamata MAC command. La MAC command ha il significato di conferma all'invio dei dati. Il coordinatore, dopo aver ricevuto la MAC command, spedisce in successione una PDU ACK di conferma e successivamente i dati. Il nodo a trasferimento completato invia un ACK. Se la rete è beacon-disabled, il trasferimento dati da coordinatore a nodo avviene allo stesso modo con la sola differenza che il coordinatore memorizza i dati finché non è il nodo stesso a stabilire la connessione. In questa modalità si ricorda che ogni accesso è fatto attraverso il CSMA/CA.

Tipi di frame a livello MAC

I frame definiti dall'IEEE 802.15.4 sono di quattro tipi:

- `Beacon Frame:` sono utilizzati per delimitare una superframe e per coordinare la sincronizzazione tra nodo e coordinatore.

Figura 2.1: Esempio di WSN.

- `Data Frame:` questo frame ha un payload massimo di 104 byte. Il suo compito principale è la trasmissione dei dati ma fornisce altri servizi come assicurarsi di una tracciatura dei pacchetti e di eseguire controlli sull'integrità della PDU.

- `ACK Frame:` il compito dell'ACK frame è quello di fornire un feedback attivo dal ricevitore della corretta ricezione di un pacchetto.

- `MAC Command Frame:` essa consente meccanismi per il controllo e la configurazione dei vari nodi come visto per le comunicazioni tra coordinatore e nodo.

2.1.3 Adattamento della potenza trasmissiva

L'adattamento della potenza trasmissiva è stato studiato nel contesto delle reti wireless di sensori (WSN) che possono essere usate per monitorare aree inaccessibili o rischiose. Ogni WSN è composta di molti modi, ognuno dei quali ha una propria alimentazione, per esempio la batteria. Lavorando in aree ad accesso difficile o pericoloso è fondamentale minimizzare la potenza trasmissiva per prolungare il più a lungo la vita della WSN (meno potenza uso, meno batteria consumo). Bisogna tenere a mente anche che l'affidabilità dei dati trasmessi rappresenta un requisito fondamentale. L'ottimizzazione della potenza trasmissiva e l'affidabilità quindi sono concetti fondamentali quando si opera con sistemi moderni basati sulle WNS. L'idea è quella di trovare un trade-off tra l'efficenza e l'energia consumata in ambienti di questo tipo.

In [8] [9], viene proposta una tecnica chiamata TPSO (Transmission Power Self-Optimization) per risolvere il problema appena descritto. Questa tecnica consiste di un algortimo in grado di garantire una equa qualità del servizio (QoS) concentrandosi sull'efficenza (Ef) delle WSN mentre si ottimizza la potenza trasmissiva necessaria per la comunicazione dati.

La tecnica TPSO si occupa di trovare un trade-off tra l'efficienza (Ef) di una WSN e il consumo di energia. Il modello di comunicazione usato prevede la presenza di un Master Node (MN), chiamato anche stazione base e n Slave Node (SN) come si vede in Figura 2.1. Si adotta un concetto di

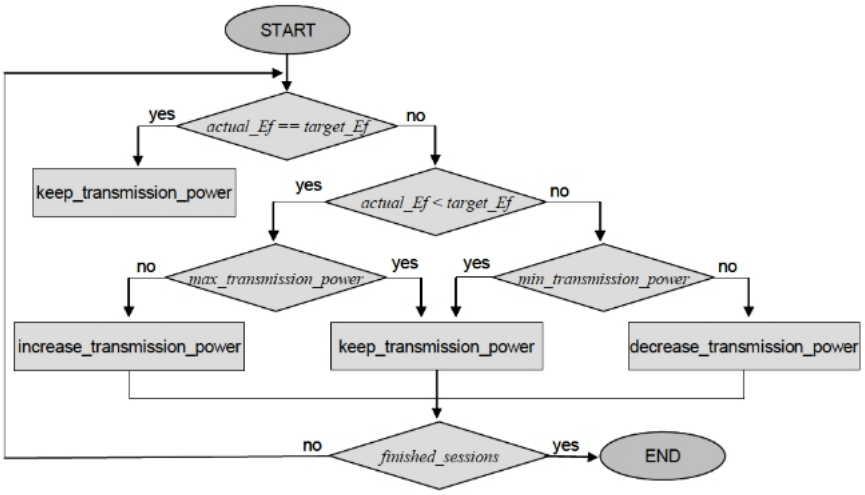


Figura 2.2: Struttura dell'algoritmo TPSO.

sessione S, composta di t sessioni di tempo (ST) di lunghezza R. Quindi S sarà composta da ST_0,ST_1,...,ST_{t-1}. Il MN effettua una fusione dei dati che arrivano dagli SN entro la stessa sessione S. La metrica di efficienza considerata è data dalla formula:

$$Ef = \frac{\sum_{i=1}^{N} Mr}{E_{Ms}}$$

dove N è il numero dei round, Mr è il numero dei messaggi ricevuti e E_{Ms} è la stima dei messaggi inviati dai SN. Altra metrica importante è la QoF, cioè il numero medio di messaggi ricevuti dal MN durante le ST [10].

L'idea di base della tecnica TPSO è quella di adattare la potenza trasmissiva tenendo conto della *Ef* dell'intera rete. E' importante sottolineare che la tecnica TPSO usa la *Ef* associata ad ogni SN per calcolare la *Ef* totale. Il MN è responsabile di calcolare la *Ef* dell'intera rete e deve inviare la *Ef* specifica ad ogni SN. Infine i SN sono incaricati di regolare la loro potenza trasmissiva basandosi sulla loro *Ef* seguendo l'algoritmo TPSO visibile in Figura 2.2. Bisogna comunque sottolineare il fatto che i sensori hanno una minima e massima potenza trasmissiva predefinita, che non può essere cambiata dall'algoritmo. La potenza trasmissiva sarà quindi incrementata o decrementata passo dopo passo tra tutti i livelli di potenza possibili del sensore.

2.2 Networked Control Systems

Un Networked Control System (NCS) è un sistema distribuito e controllato in cui il controllore è collegato al sistema da controllare (detto anche im-

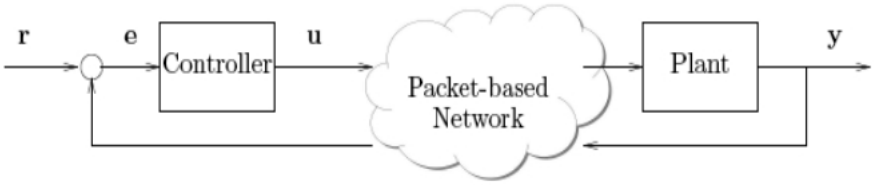


Figura 2.3: Schema a blocchi di un Networked Control System

pianto) mediante una rete digitale a pacchetto che può essere cablata (ad esempio field-bus, ethernet) oppure senza fili (ad esempio WiFi, UMTS, onde sonore). NCS sono descritti in molti articoli e le possibili soluzioni di controllo sono dovute principalmente alle diverse assunzioni sul canale di comunicazione [13] [14] [15] [16].

La teoria moderna sul controllo è largamente basata sull'astrazione che le informazioni sono trasmesse su canali di comunicazione perfetti e le computazioni possono essere istantanee o periodiche. Questa astrazione ha servito tale campo per 50 anni e ha portato a molti successi in un vasta varietà di applicazioni. Le future applicazioni di controllo però saranno molto più ricche di informazioni rispetto al passato e dovranno coinvolgere comunicazioni di rete, computazione distribuita e alti livelli di logica e processi decisionali. La nuova teoria, i nuovi algoritmi e le dimostrazioni dovranno essere sviluppate sul fatto che i segnali di input e output sono pacchetti di dati che arrivano con tempi variabili, non necessariamente in ordine e a volte anche in parti. Le reti tra sensori, attuatori e le loro computazioni dovranno essere prese in considerazione e gli algoritmi dovranno essere un tradeoff tra accuratezza e tempo di computazione.

Lo schema a blocchi tipico di un NCS è quello mostrato in Figura 2.3 in cui `u` è il flusso dei valori di comando da dare all'impianto, `y` è la grandezza che si vuole controllare in uscita dall'impianto, `r` è l'andamento desiderato per `y`, ed `e` è l'errore tra andamento desiderato e andamento reale. I valori campionati di `u` (comandi) e `y` (misure) sono inseriti in pacchetti e trasferiti sulla rete.

I vantaggi di questo approccio sono l'utilizzo di una rete pre-esistente per collegare controllore e impianto e l'eventuale condivisione di tale rete con altre applicazioni (ad esempio, altre coppie controllore-impianto, supervisore dell'impianto mediante trasmissione audio-video).

Gli svantaggi derivano dalla condivisione del canale di comunicazione e sono principalmente:

- ritardi non noti a priori e variabili nel tempo;
- perdite di pacchetti di entità non nota a priori e variabile nel tempo;

- nel caso di trasmissioni senza fili, errori sui bit di entità non nota a priori e variabile nel tempo.

La stabilità e le performance di questi tipi di sistemi sono fortemente influenzati da questi problemi. Essi influenzano sia la trasmissione dei comandi da controllore a impianto sia la trasmissione delle misure da impianto a controllore. Scopo quindi della ricerca in questo campo è il raggiungimento di certe prestazioni di controllabilità dell'impianto nonostante la presenza delle suddette problematiche di trasmissione. Questo scopo finora è stato raggiunto considerando la rete nel caso peggiore e agendo unicamente sul progetto del controllore. Solitamente chi progetta i sitemi di controllo non si occupa della parte di comunicazione che considera come data. Analogamente il progettista di reti non si interessa delle particolari applicazioni che utilizzeranno la rete ma cercherà di progettare un canale ragionevolmente buono per tutti i tipi di processi. Sfortunatamente due "ottimi locali" (lato controllo e lato rete) non sempre raggiungono l'ottimo globale. Per questo motivo è utile che i due tipi di progetto siano integrati e portati avanti contemporaneamente.

Le applicazioni che fanno parte dei NCS sono svariate e ne elenchiamo qui alcuni esempi:

- `Rover:` i rover sono dei veicoli per l'esplorazione della Luna e di Marte. Sono dotati di una certa autonomia decisionale (e.g. collision avoidance) ma per alcuni compiti richiedono comandi inviati dalla Terra (e.g. strumenti di analisi, percorsi).

- `Robocup:` lo scopo di Robocup è lo sviluppo di multi-robot cooperativi e sistemi multi-agente operanti in ambiente avverso (la squadra avversaria).

- `Unmanned Aircraft:` veicoli aerei comandati a distanza.

- `Italian UAV display team:` formazione di veicoli che devono comunicare tra di loro per mantenere una certa formazione seguendo un aereo leader (come gli stormi).

- `Autonomous underwater vehicle auv:` veicoli comandati a distanza in cui il mezzo di trasmissione ha una velocità relativamente bassa rispetto al caso wireless tradizionale.

- `Teleoperation:` un sistema di teleoperazione (bilatera) consiste di un haptic device (master robot), di una rete di trasmissione e di un robot slave. L'operatore agisce sul master e lo slave deve rispondere in maniera appropriata. Particolare attenzione va data al caso in cui lo slave interagisce con l'ambiente.

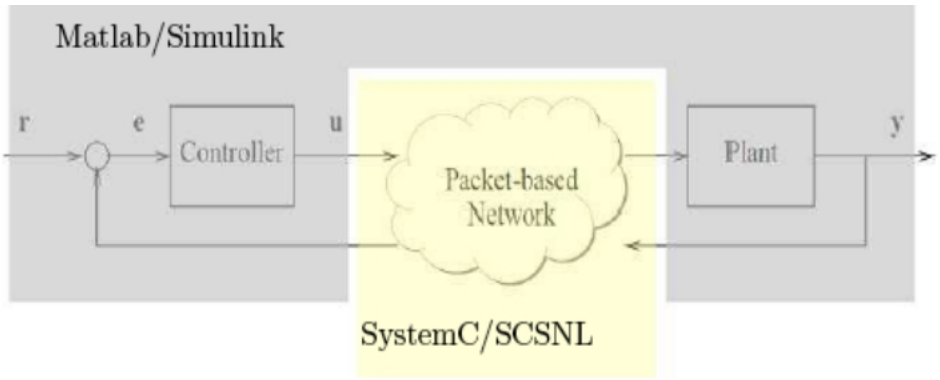


Figura 2.4: Utilizzo combinato di diversi simulatori per un NCS.

2.3 Co-simulazione

La simulazione è una metodica consolidata per verificare il comportamento di un sistema in cui è stato introdotto un controllore. Uno degli strumenti più utilizzati per la simulazione di sistemi è Matlab/Simulink, [17], che è diventato il *de facto tool* (almeno nel mondo accademico) per effettuare questo tipo di analisi e sintesi. Un problema cruciale nella simulazione di NCS è la necessità di riprodurre il funzionamento anche della rete a pacchetto. Matlab/Simulink non dispone di elementi che permettano la modellazione fedele di una rete a pacchetto in cui diverse sorgenti di traffico condividono la stessa capacità trasmissiva. Esistono simulatori di rete dedicati a questo scopo che tuttavia non consentono un'agevole simulazione dei blocchi controllore e impianto e la generazione di grafici. Questo limite ha portato allo studio di una tecnica innovativa per utilizzare in modo combinato e sincronizzato

- Matlab/Simulink per la simulazione di controllore e impianto e la generazione di vari grafici risultanti;

- un'estensione del SystemC [11](chiamata SCNSL [12]) per la simulazione della rete in cui vari flussi trasmissivi competono per l'utilizzo del canale.

Questa tecnica si chiama co-simulazione ed è una strategia che permette di utilizzare differenti tools per simulare aspetti diversi di un unico sistema [18]. Lo scopo è di utilizzare il miglior strumento per ogni aspetto del problema. Siccome gli strumenti devono lavorare in maniera sincrona per avere risultati affidabili, il meccanismo di scambio dei dati è l'aspetto più importante. La Figura 2.4 mostra come sono utilizzati tali simulatori rispetto allo schema a blocchi di un NCS nel quale Matlab/Simulink e SystemC lavorano insieme per simulare allo stesso tempo aspetti di controllo e aspetti di rete.

Ogni simulatore consiste di un kernel e un set di moduli simulati. Il kernel controlla il tempo di simulazione e invoca i moduli simulati secondo una determinata politica. Il kernel Matlab è un simulatore time-driven e chiama i moduli periodicamente, mentre il kernel SystemC è un simulatore event-driven e gestisce una lista di eventi time-sorted.

Quando un modulo simulato deve comunicare con un altro modulo appartenente all'altro simulatore vi sono due soluzioni, i.e., *module-based communication* and *kernel-based communication*. Nel primo caso ogni coppia di moduli appartenenti a simulatori diversi devono gestire un canale tra i processi (e.g., socket). In questo caso una esplicita API a livello utente per lo scambio di messaggi deve essere posta in entrambi i moduli per limitare il loro riutilizzo. Inoltre, ogni coppia di moduli di interazione necessita di un canale differente che porta a problemi di scalabilità. Infine, una sincronizzazione consistente non è facile da realizzare poichè il tempo di simulazione è gestito all'interno di ogni kernel di simulazione. Nelle comunicazione kernel-based, i kernel di simulazione si scambiano messaggi direttamente per trasferire dati tra i modelli e per mantenere la sincronizzazione. Rispetto al primo approccio, la sincronizzazione è più facile da mantenere e l'overhead di comunicazione è più basso quando più coppie sono coinvolte nell'interazione. Come si vede in Figura 2.5 si usa una comunicazione kernel-based lato SystemC mentre si usa una comunicazione module-based a lato Matlab. A questo scopo, è stato creato un speciale blocco Simulink chiamato *Matlab wrapper*, sviluppato come Level-2 m-file s-function.

Tutti i moduli che interagiscono con SystemC sono connessi con questo wrapper che contiene tutti gli strumenti di comunicazione per preservare la scalabilità. La nozione di tempo di simulazione può essere recuperata dal wrapper poichè la simulazione Matlab è di tipo time-driven e gli eventi accadono a velocità costante.

Il wrapper Matlab usa una libreria socket per la connessione al kernel SystemC. L'uso di socket, invece di meccanismi di memoria condivisa, permette la simulazione distribuita non solo tra diverse CPU, ma anche tra diversi host per permettere il bilanciamento del carico.

Le nuove entità lato Matlab/Simulink e lato SystemC sono basate sul concetto di *porte* condivise tra i moduli dei due tool che interagiscono. Ogni porta ha un identificativo unico e una capacità di archiviazione che permette a questa co-simulazione Matlab/SystemC di scambiarsi dati di tipo double.

Nel modello Matlab il ruolo di connessione con gli altri tools di simulazione è svolto dal wrapper Matlab. Questo modello può essere connesso ad un numero scalare o un vettore di porte (sia di input che di output) definito dall'utente. Ogni porta ha un identificatore unico e viene aggiornata con una certa frequenza. Matlab esegue questo blocco al massimo valore della frequenza e per ogni porta di input/output che deve essere aggiornata chiama il corrispondente *Packer/Unpacker*. Il Packer raccoglie i dati dalla porta e li mette nel corpo di un messaggio da inviare al kernel SystemC. Il

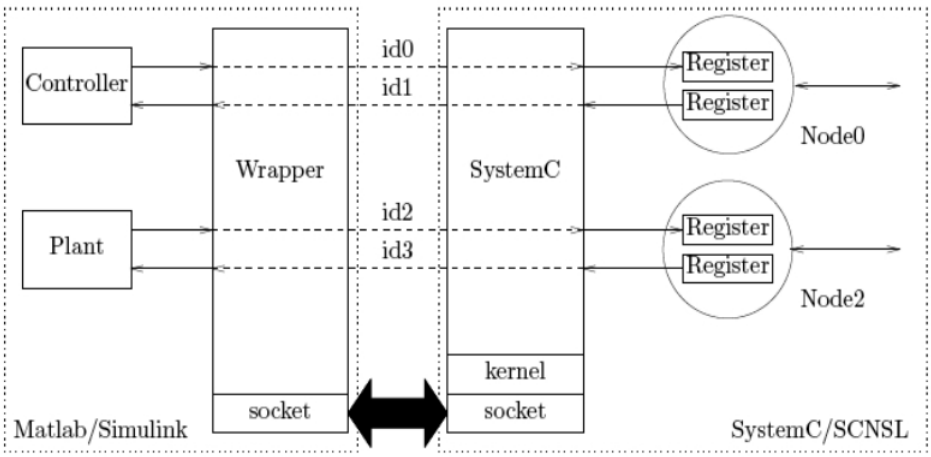


Figura 2.5: Relazione tra le nuove entità in Matlab e SystemC.

messaggio contiene anche il numero identificativo della porta e il tempo di simulazione. Il Unpacker scrive i dati nella data porta quando il wrapper riceve e elabora messaggi provenienti da SystemC con l'identificativo della destinazione corrispondente. Tutti i messaggi dalle/alle porte del wrapper sono multiplexati nei socket connessi al kernel SystemC.

La libreria SystemC è stata estesa per fornire nuove porte di input/output per inviare e ricevere dati da altri tool di simulazione. Esse sono derivate dalle classi template sc_in e sc_out e sono gestite dai metodi write() e read(). Ogni istanza di queste porte ha un identificativo unico che è usato per associarle alla corrispondente porta del wrapper Matlab. Ogniqualvolta il kernel SystemC riceve un messaggio da Matlab, esso genera un evento per la porta destinataria di input con il relativo timestamp specificato nel messaggio. Il kernel risveglia la funzione associata a questo evento e questa funzione può recuperare i dati tramite la funzione read(). Quando il processo SystemC chiama la funzione write() per inviare dati su una porta di output, il kernel di simulazione costruisce un messaggio per Matlab.

La Figura 2.5 mostra un possibile uso di queste porte in un scenario modellato secondo la Figura 2.4. Il blocco Simulink chiamato `Controller` scambia dati nella rete attraverso Node0; per questo motivo il flusso d'uscita del `Controller` è associato a una porta di input SystemC per Node0 e il flusso in entrata è associato ad una porta di output SystemC per Node0. Allo stesso modo, Node2 riceve dati dalla rete e li invia al blocco Simulink chiamato `Plant`. Node0 e Node2 sono connessi ad una rete simulata da SCNSL che non è riportata in Figura 2.5.

La sincronizzazione tra i due strumenti di simulazione dovrebbe tener conto di approcci differenti, per esempio, un approccio time-driven e uno event-driven. Matlab segue un approccio time-driven nel quale:

- output e stati interni sono valutati a step di tempo costante,
- la lunghezza del tempo di step può cambiare durante la simulazione per descrivere meglio l'evoluzione del sistema.

SystemC segue invece un approccio di tipo event-driven in cui:

- gli eventi non accadono a tempo costante,
- ogni evento ha un tempo di "fire" e questi sono tenuti in una lista ordinata in accordo con il "fire time" di ogni evento.

Ogni tool, contiene una parte dell'intero modello di simulazione ed effettua simulazioni su di esso; i tool vengono eseguiti concorrentemente dal sistema operativo. Inoltre un meccanismo di sincronizzazione è necessario per assicurare che entrambi i tool mantengano lo stesso tempo di simulazione. Per esempio, se un simulatore legge/scrive dati da/per un altro simulatore in cui il tempo di simulazione è differente, l'effetto di tali dati in uno dei simulatori potrebbe non essere corretto. Gli approcci si sincronizzazione si basano sullo scambio di messaggi tramite socket; ogni messaggio contiene l'identificatore della porta, il comando (read o write), il tempo di simulazione del tool che lo ha generato e i dati (nel caso di una write). La sincronizzazione avviene mediante un'operazione read/write bloccante nel socket. In realtà, una operazione bloccante sul socket può interrompere la simulazione in entrambi i simulatori. L'approccio esatto è basato nell'algoritmo tradizionale di accesso a memoria condivisa che assicura un perfetto allineamento del tempo di simulazione ma richiede che solo un tool sia in esecuzione a quel tempo. Ci sono altri approcci che consentono l'esecuzione concorrente al costo però di un algoritmo più complesso.

2.4 Controllo di formazione

Uno dei più importanti e fondamentali problemi nel controllo di veicoli mobili multipli è il *controllo di formazione*. Negli ultimi anni, il controllo di formazione ha ricevuto molta attenzione dalle comunità di controllo. Infatti, mentre molte attività possono essere dimostrate con veicoli multipli, che si muovono liberamente senza vincoli stretti sulle loro posizioni, vi sono altrettante importanti attività che richiedono il coordinamento dei veicoli.

Per esempio, aerei disposti in formazione a V, potrebbero minimizzare il consumo di carburante traendo vantaggi dall'aerodinamica in modo simile agli uccelli che usano tale formazione per aumentare la loro efficenza (vedi Figura 2.6). Inoltre se il controllo di formazione fosse automatizzato, aerei multipli in formazione potrebbero volare grazie ad un solo pilota alla volta, riducendo la fatica il voli di lunga distanza (i piloti di ogni aereo controllano a turno la formazione). In alternativa, si potrebbe avere un unico aereo pilotato in grado di controllare una grossa formazione di velivoli senza pilota.

Figura 2.6: Stormo di uccelli in formazione.

Altre applicazioni importanti che richiedono il controllo di formazione sono la coordinazione di robot multipli e di veicoli marini senza equipaggio, oppure il controllo di satelliti e veicoli spaziali che possono essere usati come array di sensori di posizione in modo da creare delle forme geometriche molto precise.

In letteratura, sono descritti sostanzialmente tre metodi per affrontare il problema del controllo di formazione di veicoli mobili multipli: approccio leader-following, struttura virtuale e behavior based. Ogni metodo ha i suoi vantaggi e i suoi svantaggi, ma l'approccio leader-following è ampiamente utilizzato a causa della sua semplicità, scalabilità e affidabilità.

Approccio Leader-Following

In questo tipo di controllo di formazione, alcuni veicoli sono considerati come leader, mentre il resto del gruppo agisce da follower [5] [6]. Ogni leader compie una traiettoria predefinita mentre i follower seguono il leader grazie a degli schemi predefiniti. Ogni veicolo quindi prende un veicolo vicino come riferimento per calcolare il suo moto. Vi sono molte coppie di leader e follower e si possono raggiungere formazioni complesse attraverso il controllo delle posizioni assunte da queste coppie. Uno dei vantaggi di questo approccio è la facilità di implementazione. Altra caratteristica importante è che la formazione può essere mantenuta nonostante il leader riceva delle perturbazioni. Uno svantaggio però, è dato dal fatto che non c'è un feedback alla formazione nel caso di perturbazioni ad uno o più follower. Se uno di essi viene perturbato, la formazione non può essere mantenuta.

In [3] si può vedere un nuovo framework per studiare problemi della formazione di controllo su un team di robot multipli basato sull'approccio leader-following. Utilizzando controllo distribuito e protocolli di comunicazione, il livello globale di formazione viene trasformato in un problema di tracking locale in cui i follower tengono traccia delle posizioni e delle distanze da tenere dai leader. Gli algoritmi di controllo utilizzano solo informazioni relative e non è necessario un sistema di posizionamento globale. Sulla base

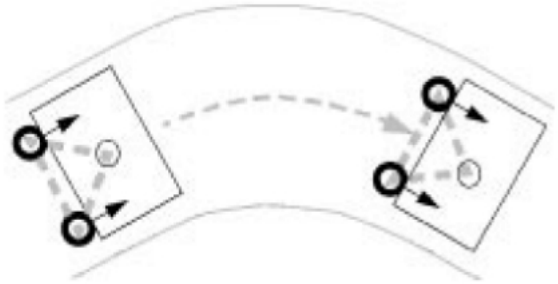


Figura 2.7: Robot mobili muovono il box lungo una certa direzione.

della formazione di un sistema a due entità vengono studiate configurazioni più complesse (n entità), basandosi sulla teoria dei grafi ad alberi. Comparando i risultati con gli altri lavori svolti su questo problema si hanno due contributi. Si è sviluppata una struttura di controllo gerarchica top-down, che unisce i vantaggi dell'approccio leader-following, del controllo distribuito e della teoria su grafi. Il secondo contributo è associato ad un miglior design del controllore.

Approccio Struttura Virtuale

Un problema studiato a fondo dalle cooperative robotiche è quello del "Box pushing". In questo problema robot mobili devono spingere un oggetto con una certa traiettoria e per riuscire a fare questo essi devono mantenere dei vincoli geometrici molto stretti con l'oggetto e tra di loro. Inoltre molti autori sottolineano che è preferibile avere più robot identici piuttosto che robot di dimensioni variabili. Come illustrato in Figura 2.7 muovere il box lungo una traiettoria richiede che i robot mantengano una relazione geometrica prefissata poiché una scarsa aderenza ai vincoli geometrici provocherebbe una non corretta distribuzione della spinta dei robot lungo il lato del box e la perdita del contatto con esso.

Un modo per affrontare questo problema è quello di pensare che ogni singolo robot sia parte di una struttura [7]. I vincoli normalmente imposti in una struttura reale sono qui imposti da una strategia di controllo. Dal momento che i robot vengono visti come parti di una struttura reale, possiamo chiamare questo schema Virtual Structure (VS).

Una soluzione al problema del movimento in formazione deve generare dei movimenti che soddisfino allo stesso tempo due condizioni:

- avanzare in una data direzione;
- mantenere un rapporto geometrico rigido con gli altri robot.

E' facile notare che la seconda condizione è lo stesso problema del formare la struttura virtuale accennata precedentemente. La prima condizione

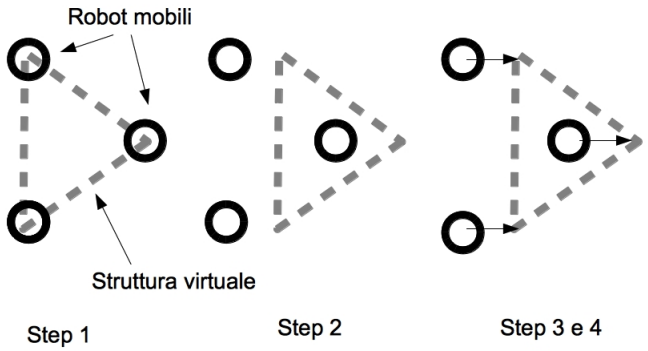


Figura 2.8: Step nell'algoritmo per il controllo di formazione.

è invece quella di muovere la struttura virtuale in una determinata direzione. Come un corpo rigido si muove sotto l'azione di una forza (i.e.,forza di gravità), la struttura virtuale può essere mossa grazie ad una forza virtuale generata grazie all'interazione con un operatore umano o con mezzi automatizzati.

Il problema è quello di progettare un algoritmo di controllo in grado di controllare i robot in modo da farli muovere in formazione in una direzione prestabilita. La struttura virtuale può essere usata per risolvere il problema del movimento in formazione. La soluzione è basata sull'idea che quando una forza virtuale è inserita nella VS ogni singolo robot dovrà muoversi nella direzione della forza. La posizione e l'orientazione della struttura virtuale dovrà poi essere determinata dalle posizioni dei singoli robot. Quindi i robot devono muoversi per soddisfare la struttura virtuale e la struttura virtuale si muove per soddisfare le attuali posizioni dei robot. Questa descrizione ricorsiva dimostra l'essenza della soluzione: avere un controllo bidirezionale. I robot mobili possono essere controllati applicando una forza virtuale alla VS, ma la posizione della struttura virtuale è determinata in ultima analisi dalle posizioni dei robot mobili. Questo comportamento può essere simulato iterando il seguente algoritmo, illustrato in Figura 2.8:

1. allineare la struttura virtuale alle posizioni correnti dei robot;
2. muovere la struttura virtuale di Δx e di $\Delta\theta$;
3. calcolare le traiettori individuali per muovere ogni robot nel punto desiderato;
4. regolare le velocità per seguire la traiettoria desiderata;
5. torna allo step 1.

Nello step 1 la VS è allineata con i robot. Una forza esterna spinge la struttura virtuale in una desiderata direzione (step 2). Nello step 3 e 4 i

robot, nel tentativo di mantenere la struttura virtuale, aggiornano le loro traiettorie in modo che il loro movimento sia come il movimento di punti di una struttura rigida.

Questa è una soluzione generale per il problema del movimento in formazione. E' un algoritmo generale che può essere adattato a differenti tipi di VS e differenti tipi di robot. In particolare gli step riguardanti il movimento della struttura virtuale e il movimento dei robot devono essere adattati per ogni setup.

Approccio behavior based

In questo tipo di approccio, descritto in [1] [4], per ogni veicolo si ha una lista di comportamenti come aggiramento di ostacoli, prevenzione alle collisioni con altri veicoli, raggiungimento dell'obiettivo. Il controllo di formazione è calcolato mediante una ponderazione dell'importanza di ogni comportamento. Questo approccio è naturale per ricavare strategie di controllo quando si hanno più obiettivi contrastanti, inoltre ha come vantaggio la presenza di un feedback esplicito nella comunicazione con i vicini di formazione. Per contro, con questo approccio non è possibile definire in modo esplicito il comportamento del gruppo ed è difficile garantire la stabilità del gruppo vista la difficoltà di analisi matematica.

Capitolo 3

Obiettivi

In questo capitolo verranno descritti nel dettaglio gli obiettivi che hanno portato allo svolgimento di questa tesi:

1. creare un nuovo modello di canale di comunicazione wireless, atrraverso una estensione del simulatore SCNSL, che preveda un tempo di propagazione dei pacchetti variabile, dipendente dalla distanza tra i nodi nella rete e dalla velocità di propagazione del segnale fisico. Inoltre si desidera aggiungere alla libreria la possibilità di simulare l'effetto del movimento dei nodi sulle comunicazione wireless (ad es. interferenza, mancata raggiungibilità e variabilità nel tempo di propagazione). Per la gestione della mobilità dei nodi si dovrà modificare la gestione degli eventi in SCNSL. SCNSL si appoggia sul kernel di simulazione di SystemC che è di tipo event-driven, basato cioè sulla gestione di una coda di eventi ordinati temporalmente. La modellazione del movimento dei nodi e della propagazione del segnale fisico richiede di poter modificare tale coda di eventi in funzione del cambiamento di posizione dei nodi;

2. progettare un modello Matlab sul controllo di formazione di tipo leader-following in cui sia presente, per ogni follower, un controllore in grado di raggiungere/mantenere la formazione ideale. Si dovranno quindi creare degli scenari allo scopo di effettuare la co-simulazione Matlab/-Simulink - SCNSL per introdurre una rete simulata che crei interferenze nella comunicazione;

3. creare Tool di simulazione che permettano di simulare ambienti di co-simulazione e che siano di facile utilizzo per l'utente finale. La creazione di questi Tool deve basarsi su un lavoro di scripting in modo tale da automatizzare completamente il lavoro e dare la possibilità di svolgere un numero considerevole di simulazioni.

4. implementare un algoritmo per la regolazione della potenza trasmissiva in uno schema di controllo di formazione leader-following. Il controllo

della potenza trasmissiva ha l'obiettivo di migliorare le prestazioni del controllo in presenza di una rete wireless. Tale controllo deve trovare dinamicamente il giusto valore di potenza per evitare da un lato le interferenze e dall'altro di mancare il raggiungimento dei destinatari;

5. dimostrare che le estensioni apportate al simulatore di rete consentono di valutare in maniera più efficace le prestazioni del contollo di formazione con adattamento della potenza trasmissiva.

6. modificare alcuni aspetti dell'algoritmo con l'obiettivo di migliorare le performance e ottenere cosi un miglior controllo dei veicoli nella formazione.

Capitolo 4

Estensione del simulatore e validazione

SCNSL fornisce alcune classi che descrivono i canali fisici veri e propri da poter usare durante la simulazione. All'interno della libreria SCNSL sono definiti diversi tipi di canale. I canali provvedono a trasferire i pacchetti da un nodo all'altro gestendone anche l'eventuale collisione. Tra i canali di rete wired troviamo:

- HalfDuplex: canale di rete bidirezionale ma nel quale non viene consentita la trasmissione simultanea in entrambe le direzioni.
- FullDuplex: canale di rete bidirezionale con possibile trasmissione simultanea in entrambe le direzioni.
- Unidirectional: canale con unica direzione di trasmissione.

mentre tra i canali di rete wireless:

- Shared: canale in cui viene considerato costante il tempo di propagazione.
- DelayedShared: canale in cui viene calcolato il tempo di propagazione in funzione della posizione dei nodi.

La caratteristica fondamentale che differenzia i due canali wireless quindi è il tempo di propagazione dei pacchetti. Verrà descritto in seguito tutto ciò che caratterizza i due canali wireless, spiegando nel dettaglio implementazione e funzionamento dei due.

4.1 Gestione degli eventi

Prima di procedere con l'analisi dei canali di rete Wireless in SCNSL è opportuno soffermare l'attenzione sul funzionamento della classe `EventsQueue_t`

e quindi sulla gestione degli eventi all'interno del simulatore. Tale classe infatti è nata con l'obbiettivo principale di fornire un supporto per la gestione degli eventi all'interno di SCNSL. Il simulatore infatti gestisce come eventi le seguenti fasi della simulazione:

- l'inizio di ricezione di un pacchetto da parte di un nodo della rete;
- la fine ricezione di un pacchetto da parte di un nodo della rete;
- il movimento di un nodo.

Al fine di gestire tali eventi la classe `EventsQueue_t` implementa:

- la coda `_queue` la quale mantiene ordinati gli eventi in base al tempo in cui tali eventi verranno sollevati;
- dei metodi template per la creazione degli eventi;
- un metodo per l'eliminazione degli eventi;
- un metodo per la modifica del tempo relativo ad un evento;
- un metodo che restituisce il tempo relativo ad un evento.

La coda di eventi è una struttura `list` della libreria standard C++. La scelta di utilizzare questa struttura deriva dal fatto che la `list` permette di utilizzare il metodo `sort` per il riordino degli elementi che nel caso di SCNSL sarà un ordinamento in base al tempo. Gli eventi contenuti all'interno di tale lista sono di tipo `queue_event_t`. Il codice che li implementa è il seguente:

```
struct queue_event_t
{
    sc_core::sc_time time;
    void ( * action )( void * );
    void * data;
    data_type_t type;
    event_id_t id;

    queue_event_t():
        time(),
        action( NULL ),
        data( NULL ),
        type(),
        id()
    {}

    /// @brief Comparison operator.
    bool operator < ( const queue_event_t & e )
            const throw ();

    /// @brief Equality operator.
    bool operator == ( const queue_event_t & e )
        const throw ();
};
```

Gli elementi di maggior importanza in questa struttura sono l'elemento `time` che rappresenta il tempo per il quale tale evento viene lanciato e `id` che è l'identificativo dell'evento. Gli altri elementi della struttura sono necessari per istanziare la callback da richiamare. Si può notare che la struttura implementa inoltre un operatore di comparazione e un operatore di uguaglianza tra eventi utilizzati dall'algoritmo di sorting.

Dopo aver analizzato la struttura di un evento in SCNSL verranno descritti ora in dettaglio tutte le fasi relative ad un evento dalla sua creazione alla rimozione dalla coda di eventi.

4.1.1 Creazione di un evento queue_event_t

La creazione di un evento in SCNSL avviene invocando la funzione template `registerAction()`. Questa funzione utilizza i dati passati come parametro per creare un evento di tipo `queue_event_t` in questo modo:

```
queue_event_t queue_event;
queue_event.time = time;
queue_event.action = & _instance_callback_action_performer;
queue_event.data = reinterpret_cast< void * >( d );
queue_event.type = INSTANCE_METHOD_DATA;
queue_event.id = id;
```

Il metodo prevede quindi l'inizializzazione dell'evento utilizzando i parametri riferiti al tempo, alla callback da invocare e all'identificativo. Successivamente inserisce l'evento all'interno della coda `_queue` tramite una `push_back`:

```
// Storing the event in the _queue:
_queue.push_back( queue_event );
```

effettua un riordinamento degli eventi in base al tempo utilizzando la funzione `sort`:

```
// Sorting the _queue:
_queue.sort(_compareTime);
```

ed infine restituisce un identificativo di tale evento:

```
// Return the event-relative id:
return id;
```

Il riordino avviene attraverso l'utilizzo di un algoritmo tradotto nella funzione `_compareTime` la quale confronta e sistema gli eventi nella `_queue` in ordine crescente di tempo in maniera da ottenere in testa alla coda l'evento prossimo a dover essere sollevato.

4.1.2 Modifica di un evento queue_event_t

Nel caso avvenga un movimento di un nodo all'interno della rete simulata, SCNSL si occupa di modificare, se necessario, i tempi relativi agli eventi. Supponiamo di creare una simulazione che usa un canale con delay dei pacchetti variabile a seconda della distanza dei nodi. Se un nodo destinatario si

avvicina a un nodo mittente il pacchetto deve essere ricevuto prima, quindi gli eventi di inizio e fine ricezione dovranno essere anticipati. Per fare ciò SCNSL utilizza il metodo della classe `EventsQueue_t` chiamato `updateAction()`. Il codice della funzione è il seguente:

```
void EventsQueue_t::updateAction( event_id_t event_id,
                                  const sc_core::sc_time & time )
    throw ( std::domain_error )
{
    queue_event_t matcher;
    matcher.id = event_id;

    std::list< queue_event_t >::iterator it;

    // Find the id-relative event:
    it = find ( _queue.begin(), _queue.end(), matcher );
    if ( it == _queue.end() )
        throw std::domain_error("Event ID not found.");

    // Updating the simulation time into which fire the action:
    it->time = time;
    // Sorting the _queue:
    _queue.sort( _compareTime );
}
```

La prima fase è la ricerca dell'evento da modificare all'interno della `_queue` utilizzando l'id relativo passato come parametro. La seconda fase è la sostituzione del tempo dell'evento con il nuovo tempo e il riordino della `_queue` tramite il metodo `sort()` descritto precedentemente.

4.1.3 Esecuzione di un evento queue_event_t

Ogni evento `queue_event_t` contiene al suo interno una callback che viene richiamata quando il tempo dell'evento raggiunge il tempo corrente di simulazione. Il metodo privato `_process()` della classe `EventsQueue_t` si occupa di tale controllo attraverso le seguenti righe di codice:

```
time = sc_core::sc_time_stamp();

// Let's try to dispatch events (if any).
while ( ( ! _queue.empty() ) && time >=  _queue.front().time )
{
    SCNSL_TRACE_DBG( 5, "<> _process(): dispatching." );

    const queue_event_t e( _queue.front() );
    _queue.pop_front();
    ( * ( e.action ) ) ( e.data );
}
```

Quando il tempo relativo all'evento in testa alla coda raggiunge il tempo corrente di simulazione allora è il momento di sollevare tale evento. A questo punto i passi che si eseguono sono:

- estrazione dell'evento dalla `_queue`;

- esecuzione di una `pop_front()` sulla `_queue` per eliminare l'evento dalla coda.

- esecuzione della callback `action` con parametri `data`.

Infine il metodo ritorna in condizione di attesa finchè non sopraggiunge un altro evento da eseguire.

4.2 Il canale Shared

Il canale Shared (condiviso) è un canale half-duplex con tempo di propagazione dei pacchetti costante. Tale valore dovrà essere impostato dall'utente assegnando un valore preciso alla costante `delay` che nella simulazione con canale Shared contraddistingue proprio il tempo di propagazione di un pacchetto all'interno del range di trasmissione.

Oltre a questa costante l'utente dovrà impostare le coordinate tridimensionali che indicano la posizione di un nodo nella rete. Tali informazioni sono fondamentali alla simulazione, in quanto serviranno al canale per calcolare la posizione di un nodo destinatario rispetto al range di trasmissione di un nodo mittente. Infatti il canale dovrà verificare se il nodo ricevitore si trova all'interno o all'esterno di tale range in modo che il simulatore sia in grado di decidere se tale nodo dovrà ricevere o no i pacchetti. Per quanto riguarda l'esatta posizione del nodo quando questo si trova all'interno del range di trasmissione di un nodo mittente, non ha particolare rilevanza, in quanto il delay di propagazione è costante in ogni punto e quindi a livello di simulazione basta sapere che si è all'interno o all'esterno del range trasmissivo.

Altra caratteristica fondamentale del canale Shared è la possibilità di collisioni dei pacchetti all'interno della rete. Essendo un canale wireless half-duplex le cause che portano un pacchetto a collidere sono:

- la ricezione di un pacchetto da parte di un nodo mittente durante la fase di encoding. Ciò non è permesso in un canale half-duplex e quindi non si possono ricevere pacchetti da altri nodi mentre si stanno inoltrando pacchetti nella rete;

- la ricezione contemporanea di due o più pacchetti da nodi diversi della rete.

Descriviamo ora in dettaglio il funzionamento della classe che descrive il canale Shared, facendo riferimento anche a pezzi di codice della stessa per capire meglio le varie dinamiche del canale.

4.2.1 Inizializzazione della simulazione

La prima fase di una simulazione con SCNSL è la registrazione dei nodi attraverso la funzione `bind()` della classe `Scnsl_t`, la quale invoca indirettamente il metodo `addNode()` del canale scelto per la simulazione. In questo caso si tratta del canale Shared il quale esegue in ordine i seguenti task:

- assegna al nodo un identificativo di tipo `id_t` utilizzando la struttura `map` della libreria standard C++. Tale struttura è definita come segue:

  ```
  typedef std::map< const Node_t *, id_t > IdMap_t;
  IdMap_t _ids;
  ```

  La `_ids` memorizza elementi che sono la combinazione di un valore chiave e un valore chiamato `mapped value` (il nodo registrato). La seguente riga di codice mostra come avviene l'assegnazione:

  ```
  _ids[ n ] = _nodes_number ++;
  ```

- effettua una chiamata al metodo `addNode()` dell'interfaccia `Channel_if_t` mediante la quale viene aggiunto il nodo alla lista dei nodi della simulazione:

  ```
  Channel_if_t::addNode( n );
  ```

- aggiorna tutte le proprietà del nodo (posizione, bitrate, soglia di ricezione e raggiungibilità) mediante la chiamata alla funzione `updateProperties()`.

Tale metodo però, oltre ad essere utilizzato in fase di inizializzazione, viene richiamato ogni qualvolta avviene un movimento di un nodo all'interno della rete. Il canale Shared, infatti, è stato creato per supportare la mobilità dei nodi della rete. Ci occuperemo più avanti della mobilità dei nodi, concentriamoci ora sul funzionamento del metodo `updateProperties()`.

La prima parte del metodo è dedicata all'assegnazione di un `ID` al nodo passatogli come parametro e alla chiamata alla funzione `getProperties()` della classe `Node_t` in modo da salvarsi nella struttura `node_properties_t` tutte le proprietà del nodo. Fatto ciò, tramite un ciclo `for` per scorrere tutti i nodi della simulazione, si effettua una doppia chiamata al metodo di supporto `_updateSingleProperties` passando il nodo ora identificato come `ID` prima come `sender` e poi come `receiver`. In sequenza dunque la funzione `_updateSingleProperties()` si può descrivere tramite queste fasi:

- verifica se la coppia di nodi trasmette pacchetti con lo stesso bitrate:

  ```
  _transmissions[ ID ][ recv_id ].same_encoding_speed =
          (fabs( sp.bitrate - rp.bitrate) < _EPSILON );
  ```

- calcola la posizione del nodo da aggiornare e aggiorna anche il suo valore di raggiungibilità nei confronti degli altri nodi della rete:

```
const position_t dx = sp.x - rp.x;
const position_t dy = sp.y - rp.y;
const position_t dz = sp.z - rp.z;
const position_t d2 = dx*dx + dy*dy + dz*dz;
const double distance = sqrt( d2 );

// Check if the nodes have the same coordinates:
if ( distance < _EPSILON )
{
    // The nodes have the same coordinates.
    _transmissions[ ID ][ recv_id ].reachable = true;
}
else
{
    // The nodes have differents coordinates.
    const bool pow_reac = sp.transmission_power /
        pow( distance, _ALPHA ) >= rp.receiving_threshold;
    _transmissions[ ID ][ recv_id ].reachable = pow_reac;
}
```

Si noti come il controllo riguardante la raggiungibilità dei nodi sia diviso in due casi. Si è pensato che in fase di testing l'utente possa dare a più nodi della rete le stesse coordinate tridimensionali. Tale assunzione è impossibile nella realtà, in quanto due dispositivi fisici non potranno mai trovarsi esattamente nello stesso punto dello spazio, ma in SCNSL si è tenuta in considerazione anche tale opportunità. Nel codice viene quindi controllato se ciò accade mediante il confronto con una costante infinitesimale chiamata `_EPSILON`. Se l'esito del controllo è positivo e quindi le posizioni sono identiche, i due nodi vengono considerati come raggiungibili, altrimenti il loro valore di raggiungibilità viene calcolato mediante l'operazione all'interno del ramo dell'`else` che usa le proprietà (`transmission_power`, `receiving_threshold`) rispettivamente del nodo `sender` e del nodo `receiver`;

- verifica la presenza di pacchetti corrotti. Tale controllo avviene solo in caso di aggiornamento delle proprietà del nodo in un momento successivo all'inizializzazione della simulazione, in quanto in fase di inizializzazione i nodi devono ancora cominciare a trasmettere. Ciò significa che questa parte di codice verrà eseguita solo nel caso del movimento di un nodo della rete.

4.2.2 Spedizione e ricezione di pacchetti

Questa sezione descrive in dettaglio il comportamento del simulatore SCNSL durante l'invio e la ricezione dei pacchetti da parte dei nodi della rete e vista la possibilità di collisioni, effettua un controllo accurato sui tempi riferiti ai pacchetti, in modo da gestire tali collisioni.

Le funzioni che tratteremo in questa sezione sono quindi dedicate a tali scopi e sono rispettivamente:

- funzione `send()`: gestisce l'invio dei pacchetti;

- funzioni `_startSingleSending()` e `_completeSingleSending()`: metodi di supporto per la ricezione del pacchetto.

Verrà ora descritto in dettaglio il funzionamento delle sopraccitate funzioni. Il metodo `send()` è il cuore del canale Shared. Tale metodo si suddivide in tre fasi:

Fase 1 - Scheduling del pacchetto: in questa fase avviene il calcolo di tutti i tempi di simulazione riferiti al pacchetto da spedire. Si salva il tempo corrente di simulazione in una variabile chiamata `currentTime` e poi si passa al calcolo dei tempi di inizio decodifica e di fine decodifica del pacchetto. Tali valori sono salvati rispettivamente nelle variabili `interferenceStart` e `interferenceEnd`:

```
const sc_core::sc_time currentTime
        ( sc_core::sc_time_stamp() );
const sc_core::sc_time interferenceStart
        ( currentTime + _DELAY );
const sc_core::sc_time interferenceEnd
        ( interferenceStart + encodingTime );
```

Come si può vedere dallo spezzone di codice qui sopra che descrive tali passaggi, `interferenceStart` viene calcolato tenendo conto del tempo corrente di simulazione più il delay del canale, in modo da ottenere l'esatto momento nel quale il nodo ricevitore comincia a decodificare il pacchetto. Come sappiamo il tempo di propagazione del pacchetto è costante in questo canale e quindi conosciuto a priori. Per il calcolo di `interferenceEnd` invece si somma al tempo di inizio decodifica il vero e proprio tempo di decodifica salvato nella variabile `encodingTime`. Questi valori vengono quindi salvati nella struttura `_single_infos[ ID ]` che tiene traccia di tutte le informazioni del singolo pacchetto inviato dal nodo identificato come `ID`. Tra i valori della struttura troviamo i tempi di inizio e fine decodifica del pacchetto che sono necessari sia per verificare la presenza di collisioni, sia come riferimento agli eventi di inizio e fine codifica di un pacchetto. Tale necessità deriva dal fatto che la funzione `send()` registra l'inizio e la fine ricezione di un pacchetto come eventi distinti usando la `_eventsQueue` come supporto e i tempi `interferenceStart` e `interferenceEnd` per sollevare gli eventi associati al pacchetto. Possiamo vedere qui sotto il pezzo di codice per la registrazione di tali eventi:

```
typedef void (SharedChannel_t::* mptr_t)( Node_t * );
_eventsQueue->registerAction(
        interferenceStart,
        this,
        reinterpret_cast< mptr_t >
            ( & SharedChannel_t::_startSending),
        n );
_eventsQueue->registerAction(
        interferenceEnd,
        this,
```

```
reinterpret_cast< mptr_t >
    ( & SharedChannel_t::_completeSending ),
n );
```

Si può notare come a tempo `interferenceStart` verrà sollevato l'evento che simulerà l'inizio della ricezione del pacchetto. Tale evento come si vede effettua una chiamata al metodo `_startSending()`, il quale a sua volta invoca la funzione `_startSingleSending()` dedicata all'inizio ricezione del pacchetto. A tempo `interferenceEnd` verrà inceve sollevato l'evento di completa ricezione del pacchetto, il quale produce una chiamata al metodo `_completeSending()`, il quale a sua volta effettua una chiamata alla funzione `_completeSingleSending()` che gestisce la fine ricezione di un pacchetto.

Fase 2 - Attesa del tempo di decodifica: in questa fase si esegue una `wait()` per il tempo necessario alla codifica (`encodingTime`) del pacchetto.

```
wait( encodingTime );
```

Fase 3 - Verifica delle collisioni: tale fase funge da controllo sui tempi dei pacchetti presenti nella rete che interessano il nodo mittente. Essendo il canale Shared un canale Half-duplex non esiste la possibilità, per un qualsiasi nodo, di ricevere pacchetti nell'arco di tempo in cui lo stesso sta codificando un pacchetto da spedire. Avviene quindi una verifica di collisione controllando se i tempi di inizio e fine ricezione dei pacchetti inviati dagli altri nodi intersecano i tempi di codifica di un pacchetto spedito del nodo mittente.

I metodi di supporto al canale che si occupano della ricezione dei pacchetti, sono come accennato precedentemente il metodo `_startSingleSending()` e il metodo `_completeSingleSending()`. In entrambi i metodi per l'identificzione dei nodi si usano rispettivamente `ID` per il nodo mittente e `recv_id` per il nodo ricevitore.

Il compito principale della `_startSingleSending()` è verificare a lato ricevitore se vi sono state collisioni di pacchetti dovute alla ricezione contemporanea di due o più pacchetti derivanti da nodi diversi della rete. Per controllare ciò, si fa un controllo sui tempi di inizio e fine ricezione del pacchetto spedito dal nodo `ID` e dei tempi dei pacchetti spediti dagli altri nodi della rete. Con questo controllo infatti si verifica se esiste una intersezione tra i tempi dei vari pacchetti destinati al nodo `recv_id` e se ciò avviene il pacchetto verrà segnato come colliso e perciò scartato. Secondo compito della funzione, è quello di incrementare il `carrier` del nodo ricevitore per indicare che il canale è occupato e quindi evitare trasmissioni da parte di quest'ultimo:

```
++ _single_infos[ recv_id ].active_carriers;
```

Il metodo `_completeSingleSending()`, chiamato come detto prima a tempo `interferenceEnd` identifica la completa ricezione del pacchetto. Esso effettua gli stessi controlli descritti precedentemente nella `_startSingleSending()` tra i nodi `ID` e `recv_id` per quanto riguarda le collisioni, mentre per quanto i carrier del nodo ricevitore, in questo caso vengono decrementati come segue:

```
-- _single_infos[ recv_id ].active_carriers;
```

Il ruolo principale di questo metodo però, è la chiamata alla funzione `receive()` della classe `Node_t`, dopo aver controllato l'integrità del pacchetto tramite un controllo sui campi `collided` e `corrupted` relativi al pacchetto, e sul campo `same_encoding_speed` riguardante la velocità di codifica dei due nodi. Oltre a ciò, per la corretta gestione dei pacchetti, dovranno essere fatti alcuni controlli dovuti al possibile movimento dei nodi. Il primo è la verifica della presenza di un movimento del nodo `recv_id` dall'interno all'esterno dal range di trasmissione del nodo `ID` allo stesso istante della chiamata alla `_completeSingleSending()`. Se ciò avviene il pacchetto viene assunto come valido e quindi non verrà scartato. Il secondo è la verifica della presenza di una errata corruzione del pacchetto dovuta allo spostamento del nodo `recv_id` dall'esterno all'interno del range di trasmissione del nodo `ID`. Infatti se tale spostamento è avvenuto allo stesso istante della chiamata alla `_startSingleSending()` il pacchetto non deve essere etichettato come corrotto e quindi ricevuto dal nodo `recv_id`.

Si può dire quindi che questa chiamata è la vera e propria traduzione della completa ricezione di un pacchetto da parte del nodo `recv_id`.

4.2.3 Mobilità dei nodi nello Shared channel

In questa sezione verrà descritto come viene gestita la mobilità dei nodi all'interno della rete con le dovute conseguenze sulla gestione dei pacchetti. La libreria SCNSL infatti fornisce una struttura per simulare il movimento dei nodi nelle loro tre coordinate e si occupa di effettuare una serie di controlli specifici per verificare la presenza o meno di pacchetti corrotti dovuti allo spostamento dei nodi.

Registrazione del movimento

Nel canale Shared, la registrazione del movimento di un nodo avviene attraverso la funzione `registerNodeProperties()` definita nella classe `EventsQueue_t`:

```
eventsQueue->registerNodeProperties(
        sc_core::sc_time( time, sc_core::SC_SEC ),
        n,
        np,
```

```
        ch
)
```

Tramite questo codice è possibile settare delle nuove proprietà per il nodo desiderato in un momento preciso della simulazione. L'utente dovrà prima di tutto impostare tutte le proprietà del nodo tramite una struttura di tipo `node_properties_t`. Tale struttura è identificata da `np` e i suoi campi sono:

- le tre coordinate (x, y, z) del nodo;
- il bitrate di trasmissione;
- la potenza trasmissiva;
- il valore della soglia di ricezione.

Successivamente dovrà passare tale struttura al metodo `registerNodeProperties()`, assieme al tempo in cui avverrà lo spostamento `time`, al nodo da muovere `n` e al canale `ch`. Il codice della funzione `registerNodeProperties()` è il seguente:

```
event_id_t registerNodeProperties( const sc_core::sc_time & time,
                                   Node_t * n,
                                   node_properties_t np,
                                   Channel_if_t * ch )
    throw ()
{
    const sc_core::sc_time currentTime =
            sc_core::sc_time_stamp();

#if (SCNSL_LOG >= 1)
    if ( time == currentTime )
    {
    std::stringstream ss;
    ss<<"Warning: a node movement was registered at current time.
          The simulation may not be correct.";
    SCNSL_TRACE_LOG( 3, ss.str().c_str() );
    }
#endif

    SCNSL_TRACE_DBG( 3, "<> registerNodeMovement():
                                        instance callback." );

    // Storing the event-relative id:
    event_id_t id = _event_id;

    node_instance_data_t * nd = new node_instance_data_t();

    nd->n = n;
    nd->np = np;
    nd->ch = ch;

    queue_event_t queue_event;
    queue_event.time = time;
    queue_event.action = & _instance_callback_action_performer;
    queue_event.data = reinterpret_cast< void * >( nd );
    queue_event.type = INSTANCE_METHOD_DATA;
    queue_event.id = id;
```

```
    ++_event_id;

    // Storing the event in the _queue:
    _queue.push_back( queue_event );
    // Sorting the _queue:
    _queue.sort(_compareTime);
    if ( time <= _queue.front().time )
                _headInsertedEvent.notify();

    // Return the event-relative id:
    return id;
}
```

Il primo controllo effettuato verifica se il movimento del nodo avviene nello stesso istante nel quale la simulazione si trova. Può capitare infatti che un Task decida ad un certo momento di modificare la posizione attuale del nodo relativo e quindi di registrare un movimento del nodo al tempo corrente. Se ciò avviene il simulatore avvisa l'utente della possibile presenza di anomalie nella simulazione. Un movimento registrato a tempo corrente infatti può essere schedulato successivamente al completo aggiornamento delle proprietà di altri nodi e dei relativi eventi di inizio e fine ricezione dei pacchetti e quindi il comportamento della simulazione non è prevedibile a priori.

Si nota successivamente che all'interno del metodo si crea una struttura di tipo `node_instance_data_t` la quale viene inserita all'interno di un evento.

La struttura è definita in questo modo:

```
template< class T, class D >
struct instance_data_t
{
    T * obj;
    void (T::* callback)( D * );
    D * data;

    instance_data_t():
        obj( NULL ),
        callback( NULL ),
        data( NULL )
    {}

    virtual
    ~instance_data_t(){}

    virtual
    void call()
        throw ()
    {
        (obj->*callback)( data );
    }

private:

    /// @brief Disabled.
    instance_data_t( const instance_data_t & );
```

```
        /// @brief Disabled.
        instance_data_t & operator =
                    ( const instance_data_t & );
};
```

Lo spostamento di un nodo, infatti, è gestito come un evento dalla `_queue` e quindi, a tempo `time`, tale evento viene sollevato. La struttura si occupa di effettuare una chiamata al metodo `setProperties()` della classe `Node_t`, il quale a sua volta, attraverso una chiamata al metodo `updateProperties()` definito nel canale `ch` utilizzato nella simulazione, aggiornerà le proprietà del nodo e verificherà la presenza di pacchetti corrotti grazie alla funzione `_updateSingleProperties()` descritta in dettaglio nella prossima sezione.

Aggiornamento delle proprietà dei nodi

La funzione `_updateSingleProperties()` ha lo scopo di inizializzare i nodi all'avvio della simulazione e di gestirne il movimento durante tutto l'arco della simulazione. All'inizio del capitolo si era descritto come in fase di inizializzazione viene gestito esclusivamente l'aggiornamento delle proprietà del nodo, in quanto in questa fase non vi sono pacchetti presenti nella rete, ma durante la simulazione e precisamente quando avviene uno spostamento di un nodo, è necessario controllare la presenza di pacchetti corrotti.

In questo canale i pacchetti che possono risultare corrotti sono esclusivamente quelli in fase di decodifica e più precisamente la corruzione può avvenire solo nel caso in cui durante la decodifica il nodo ricevitore effettua un movimento. Tramite il movimento il nodo può uscire o entrare nel range di trasmissione del nodo mittente che sta effettuando una trasmissione del pacchetto e dunque il controllo di corruzione viene effettuato in due casi:

1. Il nodo ricevitore entra nel range di trasmissione di un nodo mittente e decodifica solo parte del pacchetto. Se ciò accade il pacchetto viene etichettato corrotto attraverso la seguente riga di codice:

   ```
   _transmissions[ ID ][ recv_id ]
       .packetInfos.front().corrupted = true;
   ```

   Successivamente grazie ad una chiamata alla funzione `_startSingleSending()` avviene l'aggiornamento delle informazioni relative alla spedizione del pacchetto che comincerà ad essere decodificato, sebbene solo in parte

   ```
   if ( _single_infos[ ID ].packetInfos.front().send_started )
   {
       _startSingleSending( ID, recv_id );
   }
   ```

   Come si nota dal codice qui sopra però tale chiamata avviene solo nel caso in cui il campo `send_started`, che indica l'inizio della ricezione de pacchetto, sia già stato impostato a `true`.

2. Il nodo ricevitore esce dal range di trasmissione di un nodo mittente e quindi codifica solo parte del pacchetto. Tale pacchetto dovrà essere etichettato come corrotto:

```
_transmissions[ ID ][ recv_id ]
    .packetInfos.front().corrupted = true;
```

A differenza del caso precedente qui viene solamente decrementato il carrier del nodo ricevitore verificando inoltre che il pacchetto non sia continuo:

```
-- _single_infos[ recv_id ].active_carriers;
if (_single_infos[ recv_id ].active_carriers == 0 )
{
    // Do we have a continuous packet?
    const bool continuous =
    (_single_infos[ ID ].packetInfos.size() > 1 )
    &&( (++_single_infos[ ID ]
                .packetInfos.begin())->sendStart ==
    _single_infos[ ID ].packetInfos.front().sendEnd );

    if ( ! continuous )
    {
        _single_infos[ recv_id ].old_carrier = false;
        _single_infos[ recv_id ]
                    .node->setCarrier( this, false );
    }
}
```

4.3 Validazione del canale Shared

Questa sezione è dedicata a descrivere i test più interessanti, creati per validare il canale Shared. Si tratta comunque di semplici test, pensati però per riassumere casi specifici e particolari in ambienti wireless. La loro esecuzione darà risultati che ora andremo ad esporre grazie a delle descrizioni e delle immagini.

4.3.1 Test_shared_tlm

Questo test è stato creato per simulare la spedizione di pacchetti tra due Task TLM. I due Task hanno ruoli differenti nei vari scenari che sono stati creati. In particolare sono stati creati tre diversi scenari che si possono riassumere così:

1. il Task0 spedisce pacchetti mentre il Task1 riceve solamente. Non ci sono collisioni quindi i pacchetti vengono ricevuti tutti;

2. il Task0 spedisce pacchetti per tutta la durata della simulazione. Il Task1 inizialmente riceve, poi inizia a spedire pacchetti per un breve periodo di tempo e infine ritorna a ricevere solamente. Si può notare

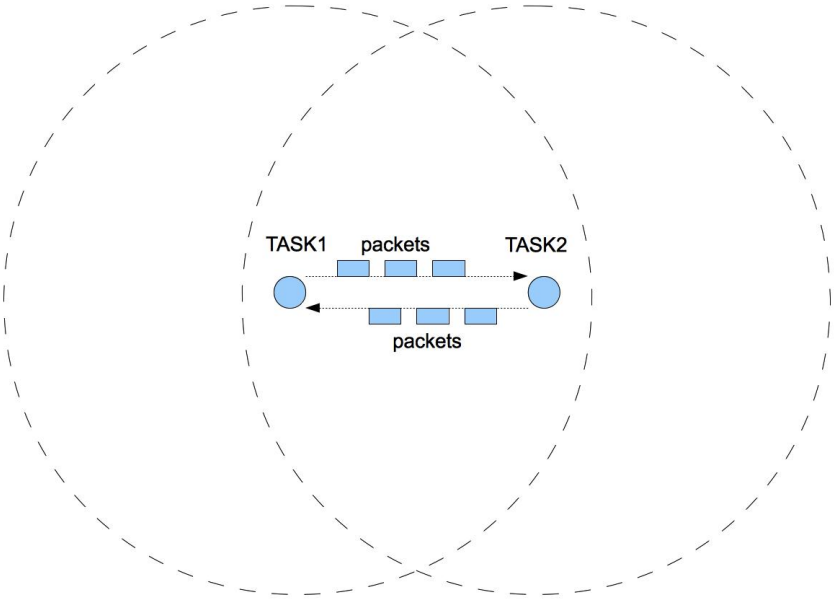


Figura 4.1: test_shared_tlm

che dall'istante `170 ms` il Task1 inizia a spedire e smette di ricevere. Questo a causa del canale Half Duplex;

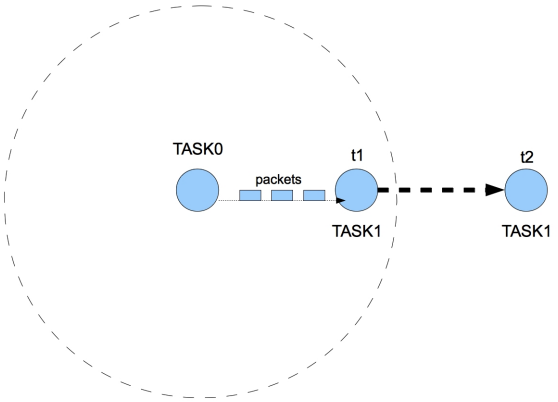
3. il Task0 e il Task1 sono sincronizzati dunque ci può essere solamente un pacchetto in trasmissione nel canale. Questa situazione è stata creata per evitare le collisioni e permettere a tutti e due i Task di inviare pacchetti.

Dalla Figura 4.1 si può vedere come i due nodi sono disposti nella rete.

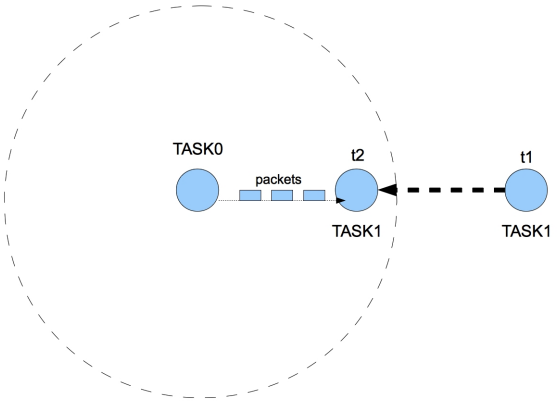
4.3.2 Test_shared_mobility_tlm

Con questo test si è voluto simulare quattro diversi scenari nei quali sono presenti due nodi che trasmettono pacchetti tra di loro. A differenza del test precedente dove i nodi rimanevano fermi qui si aggiunge la mobilità dei nodi nella rete, grazie alla quale si possono notare i diversi casi di corruzione dei pacchetti dovuti allo spostamento dei nodi. In particolare vedremo gli spostamenti eseguiti dal nodo ricevitore dall'interno all'esterno del range di trasmissione e viceversa. I casi presi in esame sono i seguenti:

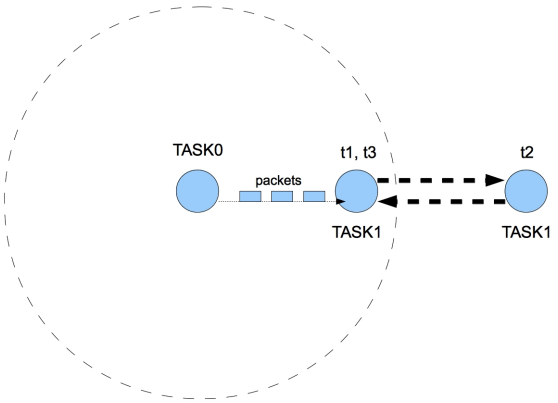
1. il Task0 spedisce due pacchetti a tempo 120 ms e 240 ms. Inizialmente il Task1 è raggiungibile, codifica interamente il primo pacchetto



(a)



(b)



(c)

Figura 4.2: test_shared_mobility_tlm

e quindi lo riceve. Successivamente quest'ultimo Task si muove all'esterno del range di trasmissione e diventa non raggiungibile; il secondo pacchetto quindi non viene ricevuto;

2. il Task0 spedisce pacchetti per tutta la durata della simulazione mentre il Task1 riceve solamente. Dopo 200 ms quest'ultimo esce dal range di trasmissione del Task0. A questo istante il Task1 sta decodificando uno dei pacchetti che risulterà corrotto perchè decodificato in parte e quindi non ricevuto. Dal tempo 200 ms in poi nessun pacchetto verrà più ricevuto dal Task1;

3. il Task0 spedisce continuamente pacchetti mentre il Task1 riceve soltanto. Inizialmente il Task1 non è raggiungibile poiché fuori dal range trasmissivo, ma successivamente ad uno spostamento esso entra nel range di trasmissione del Task0 riuscendo a codificare così soltanto una parte di uno dei pacchetti trasmessi il quale risulterà corrotto e quindi non ricevuto. I pacchetti seguenti verranno decodificati interamente e quindi ricevuti.

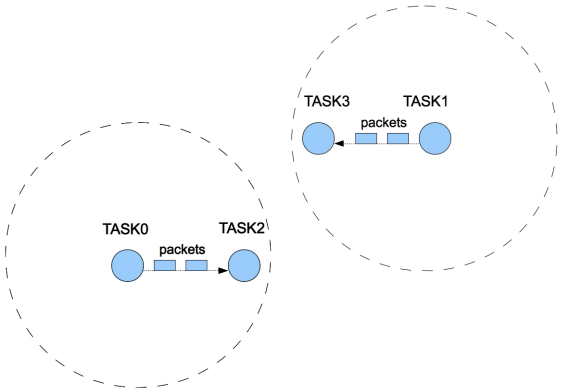
4. il Task0 spedisce pacchetti mentre il Task1 riceve. Durante la decodifica di un pacchetto il Task1 esce ed entra dal range di trasmissione del Task0 perdendo una piccolissima parte del pacchetto. Il pacchetto non completamente decodificato quindi sarà segnato come corrotto e quindi non verrà ricevuto dal Task1.

In Figura 4.2 sono rappresentati gli scenari che mostrano come un pacchetto può risultare corrotto, quindi i punti 2, 3 e 4. Il primo scenario non è rappresentato in quanto è stato pensato per mostrare solamente che se un Task ricevitore non è all'interno del range trasmissivo, esso non riesce a ricevere i pacchetti inviati dal sender.

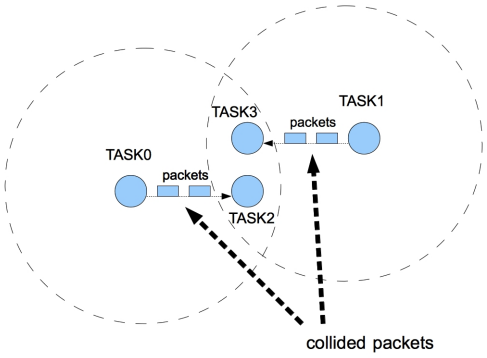
4.3.3 Test_shared_mobility_out_in_out_tlm

Questo test è una sensibile complicazione del test precedente. Vengono infatti aggiunti nella simulazione due Task che formano una coppia mittente/destinatario. Sfruttando la mobilità questo test crea varie situazioni che si possono vedere in Figura 4.3. I vari Task sono stati posizionati in modo che il tempo di delay del canale tra la coppia (Task0, Task2) e la coppia (Task1, Task3) sia lo stesso. Gli spostamenti vengono effettuati ogni 500 ms e quindi si avranno queste tre situazioni:

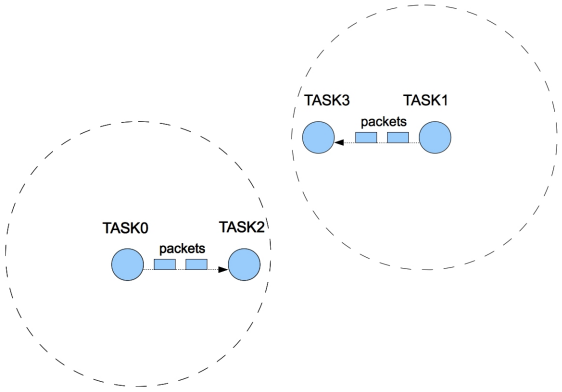
1. 0 ms: il Task0 spedisce pacchetti al Task2 mentre il Task1 spedisce pacchetti al Task3. I range di trasmissione dei nodi mittente non sono intersecati, quindi i Task ricevitori decodificheranno i pacchetti in modo corretto e li riceveranno.



(a)



(b)



(c)

Figura 4.3: test_shared_mobility_out_in_out_tlm

2. 500 ms: il Task0 e il Task2 si spostano. Ora il Task2 e il Task3 sono raggiungibili sia dal Task0 che dal Task1 che continuano a spedire pacchetti. Da questo momento in poi tutti i pacchetti ricevuti dai due Task saranno segnati come collisi, in quanto un nodo non può decodificare due o più pacchetti contemporaneamente.

3. 1 sec.: il Task0 e il Task2 tornano nelle loro posizioni iniziali e quindi i due Task ricevitori tornano a decodificare correttamente i pacchetti inviati dai sender.

4.3.4 Test_shared_mobility_hidden_exposed_tlm

Il seguente test è stato pensato per riprodurre due casi particolari che si verificano in ambienti wireless. Stiamo parlando del problema del nodo nascosto (hidden node) e del nodo esposto (exposed node).

Con problema del nodo nascosto si intende una situazione nella quale vi sono due nodi sufficientemente distanti da non essere in grado di sentirsi, ma che entrambi comunicano con un nodo nel mezzo in grado di sentire entrambi i mittenti.

Il problema del nodo esposto invece accade quando viene negata la possibilità di spedire pacchetti da parte di un nodo verso un nodo receiver per la presenza di un altro nodo mittente che con la sua trasmissione va a disturbare l'altro nodo mittente, ma in realtà non disturba il nodo receiver. Quindi è un errore per il sender disturbato pensare di non poter inviare a nessuno. Questo test riproduce in sequenza 4 situazioni di rete:

1. 0 ms: il Task0 spedisce pacchetti al Task2 mentre il Task1 spedisce pacchetti al Task 3. La posizione dei nodi nella rete fa si che i due receiver siano all'interno del range di trasmissione di un solo sender e quindi decodificano e ricevono tutti i pacchetti inviati;

2. 500 ms: il Task0 e il Task2 si spostano. Ora il Task2 e il Task3 sono raggiungibili sia dal Task0 che dal Task1 che continuano a spedire pacchetti. Si presenta proprio la situazione del nodo nascosto; ogni pacchetto ricevuto dai due Task risulta quindi essere colliso;

3. 1 sec.: il Task0 e il Task2 si spostano. In questa situazione i nodi mittente si trovano uno all'interno del range di trasmissione dell'altro, mentre i due receiver sono all'interno del range di trasmissione di un solo nodo. Si tratta del caso particolare del nodo esposto. In questo test entrambi i task inviano lo stesso pacchetti perché non viene utilizzato nessun tipo di protocollo carrier sense. Durante l'esecuzione comunque si nota che i carrier dei due nodi sender vengono sollevati per segnalare che il canale non è più libero;

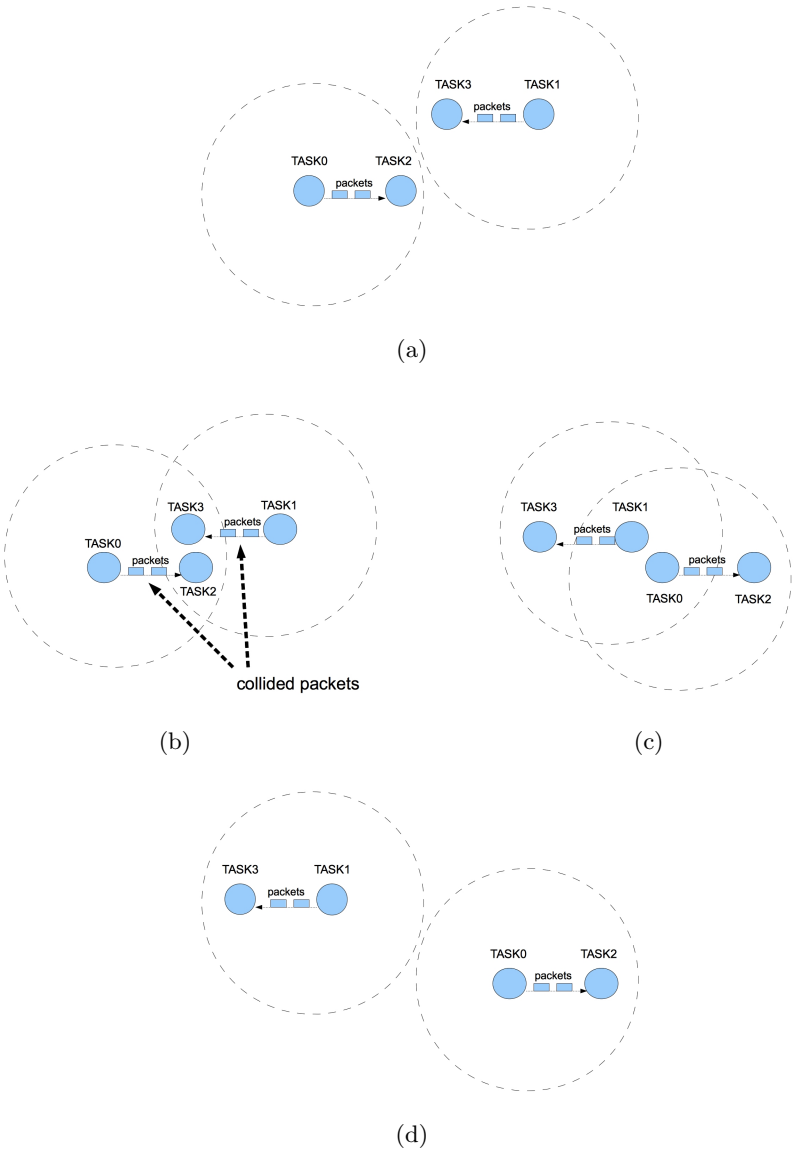


Figura 4.4: test_shared_mobility_hidden_exposed_tlm

4. 1,5 sec.: il Task0 e il Task2 si spostano nuovamente e nella loro nuova posizione i range non sono più intersecati e quindi i due Task receiver ricominciano a ricevere i pacchetti.

Le 4 situazioni descritte precedentemente sono rappresentate in Figura 4.4, dove si possono vedere anche graficamente le situazioni particolari del nodo nascosto e del nodo esposto.

4.4 Il canale Delayed Shared

Tra i canali di rete wireless è presente il canale Delayed Shared (condiviso con ritardo) il quale ha la particolarità di essere un canale half-duplex con tempo di propagazione dei pacchetti variabile in base alla distanza dei nodi. Ciò che differenzia questo canale dal canale Shared descritto in precedenza, è proprio il tempo di propagazione dei pacchetti. Nel canale Shared questo tempo infatti è costante e la posizione dei nodi non influisce sui ritardi. Nella simulazione con canale Delayed Shared, l'utente ha la possibilità di scegliere una costante di propagazione tra quelle riportate in Tabella 4.1, ed inoltre deve impostare le proprietà dei nodi tra le quali le loro coordinate tridimensionali. La costante indica la velocità di propagazione dei pacchetti nella rete, mentre le coordinate serviranno al canale per calcolare la posizione del nodo nella rete, verificare se due o più nodi che comunicano tra loro sono all'interno dei range trasmissivi ed infine se ciò è verificato calcolare il tempo di delay di trasmissione in base alla distanza tra i nodi e alla costante di propagazione scelta.

Descriviamo ora in dettaglio la classe che descrive il canale Delayed Shared facendo riferimento anche a pezzi di codice per capire meglio tutte le dinamiche del canale.

4.4.1 Inizializzazione della simulazione

La fase di inizializzazione della simulazione avviene come nel canale Shared mediante il metodo specifico `addNode()` che differisce solo per la chiamata al metodo `_inizializeProperties()` invece della chiamata al metodo `updateProperties()`. A differenza del canale Shared, qui il metodo `updateProperties()` si occupa della gestione del movimento dei nodi, con tutto ciò che uno spostamento comporta e verrà trattata più avanti.

Il metodo `_initializePorperties()` inizia con l'assegnazione di un ID al nodo passatogli come parametro e con la chiamata alla funzione `getProperties()` della classe `Node_t` per salvarsi nella struttura `node_properties_t` tutte le proprietà del nodo:

```
const id_t ID = _ids[ n ];
```

Tabella 4.1: Costanti di propagazione

Tipo	Valore (m/s)	Descrizione
WATER_SOUND_SPEED	1480	Velocità suono nell'acqua
AIR_SOUND_SPEED	320	Velocità suono nell'aria
EM_SPEED	300000000	Velocità luce nell'aria
OPTICAL_FIBER_SPEED	200000000	Velocità luce nella fibra ottica
COPPER_SPEED	200000000	Velocità luce nel rame

```
const Node_t::node_properties_t & np = n->getProperties( this );
```

Successivamente all'interno di un ciclo `for` per scorrere tutti i nodi della presente simulazione viene effettuata una doppia chiamata al metodo `_initializeSingleProperties()` con il nodo `n` passato prima come sender e poi come receiver:

```
for ( register unsigned int i = 0; i < _NODES_NUMBER; ++i )
{
    const Node_t::node_properties_t & n2p =
        _single_infos[ i ].node->getProperties( this );

    // Initializing properties with n as sender.
    _initializeSingleProperties( ID, i, np, n2p );

    // Initializing properties with n as receiver.
    _initializeSingleProperties( i, ID, n2p, np );
}
```

Tale metodo viene invocato per inizializzare tutte le proprietà tra tutte le coppie di nodi presenti nella simulazione e ha il compito principale di calcolare il delay del canale per ogni copia di nodi `( ID, recv_id )`:

```
delay_time = distance / _PROPAGATION;
```

Tale valore verrà poi salvato nella matrice che tiene traccia delle trasmissioni come possiamo vedere in queste righe di codice:

```
_transmissions[ ID ][ recv_id ].channelDelay =
    sc_core::sc_time( delay_time, sc_core::SC_SEC );
```

Un'altra operazione di cui si occupa questo metodo è il calcolo del tempo massimo che un pacchetto impiega per raggiungere il limite del range trasmissivo del nodo sorgente:

```
const double max_delay =
    pow(sp.transmission_power/rp.receiving_threshold,
            1/_ALPHA)/_PROPAGATION;
```

Questa informazione è fondamentale per alcuni controlli sui pacchetti che verranno eseguiti a livello di `_startSingleSending()` e `_completeSingleSending()`. Come ultimo punto tale metodo si occupa di calcolare la distanza tra i due nodi passati come parametro e controllarne la raggiungibilità grazie a calcoli effettuati in base alla potenza trasmissiva del sender e alla soglia di ricezione del receiver. Possiamo vedere qui sotto il codice che descrive tali calcoli:

```
const bool pow_reac = sp.transmission_power / pow( distance, _ALPHA ) >=
                      rp.receiving_threshold;
_transmissions[ ID ][ recv_id ].reachable = pow_reac;
```

4.4.2 Spedizione e ricezione di pacchetti

Questa sezione descrive come avviene l'invio e la ricezione di pacchetti in SCNSL sul canale DelayedShared. Le funzioni che si dedicano a questa parte della simulazione sono:

- la funzione `_send()`;
- la funzione `_startSingleSending()`;
- la funzione `_completeSingleSending()`.

Oltre all'invio e alla ricezione di pacchetti questi metodi implementano un meccanismo per il controllo di collisione. Essendo il canale Delayed Shared di tipo wireless infatti non è possibile la codifica di un pacchetto mentre se ne sta già codificando un altro, ed essendo anche half duplex, non si può ricevere mentre si sta inviando.

Come per il canale Shared la funzione `_send()` è il cuore del canale Delayed Shared e si compone fondamentalmente di tre fasi:

Fase 1 - Scheduling dei pacchetti: a differenza del canale Shared dove tutti i tempi di trasmissione di un pacchetto si salvano a lato mittente nella struttura `single_packet_infos_t`, qui i tempi di trasmissione vengono salvati a lato destinatario. Il nodo sorgente tiene solo traccia dell'inizio e della fine della codifica del pacchetto per effettuare i successivi controlli di collisione:

```
_single_infos[ ID ].sendStart = currentTime;
_single_infos[ ID ].sendEnd = sendEnd;
```

mentre per ogni nodo destinazione si calcolano e si salvano nella matrice `_transmissions` tutti i valori della trasmissione

```
const sc_core::sc_time delay
                ( _transmissions[ ID ][ i ].channelDelay );
const sc_core::sc_time interferenceStart
                ( currentTime + delay );
const sc_core::sc_time interferenceEnd
                ( interferenceStart + encodingTime );

// Storing times & packet:
infos.sendStart = interferenceStart;
infos.sendEnd = interferenceEnd;
infos.encodingTime = encodingTime;
infos.packet = p;
infos.packet.setChannel( this );

_transmissions[ ID ][ i ].packetInfos.push_back( infos );
```

Dopo questi primi passaggi si effettua la registrazione delle callback

```
const event_id_t start_event_id =
    _eventsQueue->registerAction(
        interferenceStart,
        this,
        reinterpret_cast< mptr_t >
          ( & DelayedSharedChannel_t::_startSingleSending ),
        & _transmissions[ ID ][ i ].idsInfos
        );
```

```
const event_id_t complete_event_id =
    _eventsQueue->registerAction(
        interferenceEnd,
        this,
        reinterpret_cast< mptr_t >
          ( & DelayedSharedChannel_t::_completeSingleSending ),
        & _transmissions[ ID ][ i ].idsInfos
        );
```

Il metodo `registerAction` della `_eventsQueue` ritorna un intero che indica l'id dell'evento, il quale viene salvato in una delle due code specifiche. Qui infatti si hanno due code:

```
EventsIdsQueue_t startEventsIds;

EventsIdsQueue_t completeEventsIds;
```

per tener traccia degli eventi di inizio e fine trasmissione. Tali code verrano poi gestite per catturare gli id degli eventi che dovranno essere modificati a causa di spostamenti dei nodi.

Fase 2 - Attesa del tempo di codifica: si attende il tempo di codifica definito dalla variabile `encodingTime` tramite una `wait()`.

```
wait( encodingTime );
```

Fase 3 - Verifica delle collisioni: come accennato precedentemente il canale Delayed Shared è un canale wireless half duplex, nel quale non esiste la possibilità per i nodi, di ricevere pacchetti nell'arco di tempo in cui stanno codificando un pacchetto da inviare. Viene quindi fatto un controllo sui tempi di spedizione e ricezione dei pacchetti. Nel caso specifico si controlla se i tempi di inizio e fine ricezione dei pacchetti inviati da altri nodi intersecano con i tempi di codifica di un pacchetto spedito dal nodo mittente.

I metodi `_startSingleSending()` e `_completeSingleSending()` sono metodi di supporto al canale dedicati a gestire la ricezione dei pacchetti e a maneggiare le code degli eventi come era stato accennato in precedenza.
La funzione `_startSingleSending()` prevede fondamentalmente tre fasi:

Fase 1 - Aggiornamento delle trasmissioni: in questa fase si calcola il tempo corrente di simulazione e il tempo di inizio della trasmissione

```
    // Updating transmissions:
const sc_core::sc_time currentTime = sc_core::sc_time_stamp();

    // Check if the movement of the node was performed after
    // the packet has reached the limit of the transmission range.
const sc_core::sc_time startTime =
  _transmissions[ ID ][ recv_id ].packetInfos.front().sendStart -
  _transmissions[ ID ][ recv_id ].channelDelay;
```

Questi dati servono per effettuare un particolare controllo riguardante i tempi di movimento dei nodi e la gestione della coda `startEventsIds`. Bisogna infatti confrontare l'istante di tempo in cui avviene il movimento del nodo e l'istante di tempo in cui il pacchetto raggiunge il limite del range trasmissivo. Nel caso in cui il movimento avvenga anche solo un attimo dopo il raggiungimento del limite del range da parte del pacchetto, quest'ultimo dovrà essere segnato come corrotto e tolto dalla coda degli eventi riguardante l'inizio della trasmissione come si nota dal seguente pezzo di codice:

```
if ( currentTime > ( _transmissions[ ID ][ recv_id ].maxDelay +
     startTime ) )
  {
  _transmissions[ ID ][ recv_id ].packetInfos.front().corrupted =
          true;

  _transmissions[ ID ][ recv_id ].startEventsIds.pop_front();
  return;
  }
```

Fase 2 - Verifica delle collisioni: si controlla che non ci siano collisioni tra i vari pacchetti presenti nella rete che coinvolgono il nodo identificato come `recv_id`. Infatti il procedimento prevede un controllo sui tempi di inizio e fine ricezione del pacchetto inviato dal nodo `ID` e dei tempi dei pacchetti spediti dagli altri nodi della rete.

Fase 3 - Modifica dei carrier: viene aggiornato il carrier del nodo che sta ricevendo il pacchetto incrementando il contatore dei carrier, per comunicare che il canale non è più libero.

Il metodo `_completeSingleSending()` ha come ruolo fondamentale nella simulazione la chiamata alla funzione `_receive()`, ma si dedica anche a controlli relativi alle collisioni e alla modifica dei carrier. Vediamo ora nello specifico le varie fasi che caratterizzano tale metodo:

Fase 1 - Aggiornamento delle trasmissioni: come accade nella `_startSingleSending()`, anche qui si calcola il tempo corrente di simulazione e il tempo in cui inizia la trasmissione del pacchetto. Grazie a questi tempi si potrà effettuare il medesimo confronto descritto in precedenza, avendo l'accortezza di aggiungere il tempo di encoding al tempo che impiega il pacchetto per raggiungere il limite del range trasmissivo. L'idea qui è stata quella di eliminare le informazioni del pacchetto contenute in `packetInfos` ed eliminare l'evento dalla `startEventsIds` nel caso in cui il movimento sia effettuato un istante dopo che l'intero pacchetto abbia superato il limite del range trasmissivo come si vede nel seguente codice:

```
if ( currentTime > ( _transmissions[ ID ][ recv_id ].maxDelay +
     startTime +
     _transmissions[ID][recv_id].packetInfos.front().encodingTime))
```

```
{
    _transmissions[ ID ][ recv_id ].packetInfos.pop_front();

    _transmissions[ ID ][ recv_id ].completeEventsIds.pop_front();
        return;
}
```

Si ha la certezza di poter eliminare le informazioni del pacchetto e anche l'id dell'evento poichè in questo caso il nodo che entra nel range trasmissivo non sentirà nemmeno una piccolissima parte del pacchetto inviato dal nodo mittente.

Fase 2 - Verifica delle collisioni: si controlla che non ci siano collisioni tra i vari pacchetti presenti nella rete come avviene nella `_startSingleSending()` ed inoltre si fa un controllo particolare riguardante il ricevitore. Si controlla infatti se chi deve ricevere il pacchetto sta inviando qualcosa nella rete. Se ciò accade, causa il canale half duplex, il pacchetto deve essere segnato come colliso.

Fase 3 - Forwarding del pacchetto: nel caso in cui il pacchetto non sia colliso, non sia corrotto e i due nodi abbiano la stessa velocità di encoding si chiama il metodo `receive()` della classe `Node_t`, che significa completa ricezione del pacchetto. Prima di effettuare la chiamata a tale metodo però si effettua un controllo per verificare se il tempo di simulazione in cui è stato fatto il movimento di un nodo ha un valore maggiore del tempo di fine ricezione del pacchetto per quello stesso nodo. In caso affermativo un warning avvisa che la simulazione potrebbe non essere corretta. Questo controllo viene effettuato in quanto il simulatore permette movimenti dei nodi a velocità anche altissime, che nella realtà invece non sono possibili. Causa di questi movimenti velocissimi è la ricezione di pacchetti in realtà non validi. Il warning avvisa proprio di questa situazione.

Fase 4 - Modifica dei carrier: viene aggiornato il carrier del nodo che ha ricevuto il pacchetto decrementando il contatore in modo da segnalare che il canale è libero.

4.4.3 Mobilità dei nodi nel canale Delayed Shared

Il metodo che gestisce la mobilità dei nodi nel canale Delayed Shared è il metodo `updateProperties()` e il metodo di supporto `_updateSingleProperties()`. La registrazione del movimento avviene come nel canale Shared attraverso il metodo `_registerNodeProperties()` definito nella classe `EventsQueue_t`. Anche qui avviene la chiamata al metodo `setProperties()` della classe `Node_t`, il quale a sua volta fa una chiamata al metodo `updateProperties()` del canale utilizzato nella simulazione, quindi in questo caso il canale è il Delayed Shared.

Tale metodo come primo punto controlla che un nodo non faccia due spostamenti allo stesso tempo. Se ciò avviene solleva una eccezione avvisando l'utente che un nodo ha fatto più di un movimento allo stesso tempo. Fisicamente un dispositivo non può trovarsi in due posizioni diverse allo stesso tempo, quindi la simulazione risulterebbe errata e perciò viene terminata. Fatto ciò all'interno di un ciclo for necessario per scorrere tutti i nodi si effettua una doppia chiamata al metodo `_updateSingleProperties()` passando il nodo `n` prima come sender e poi come receiver. Questo ultimo metodo è dedicato all'aggiornamento delle proprietà dei nodi ed inoltre effettua una chiamata alla funzione `_updateEvents()` per l'aggiornamento degli eventi associati al nodo. Descriveremo in dettaglio il funzionamento di tali metodi nelle prossime sezioni.

Aggiornamento delle proprietà dei nodi

La funzione `_updateSingleProperties()` come precedentemente accennato si occupa dell'aggiornamento delle proprietà dei nodi, in particolare di coppie di nodi `( sender, receiver )`, passati come parametro alla funzione. Si salvano nella matrice `_transmissions` tutti i dati necessari alla simulazione.

Il primo dato di interesse è il tempo massimo che un pacchetto impiega per raggiungere il limite del range di trasmissione.

```
const double max_delay =
    pow(sp.transmission_power/rp.receiving_threshold, 1/_ALPHA)/
        _PROPAGATION;
_transmissions[ ID ][ recv_id ].maxDelay =
    sc_core::sc_time( max_delay, sc_core::SC_SEC );
```

Questo dato servirà successivamente per il controllo di corruzione dei pacchetti e per l'aggiornamento degli eventi. Si passa poi a salvare il valore di raggiungibilità tra i vari nodi prima dello spostamento e si controlla se essi hanno la stessa velocità di encoding. Fatto ciò si passa a calcolare la nuova distanza tra i due nodi e di conseguenza il nuovo tempo di delay tra i due nodi come si vede dal seguente codice:

```
const position_t dx = sp.x - rp.x;
const position_t dy = sp.y - rp.y;
const position_t dz = sp.z - rp.z;
const position_t d2 = dx*dx + dy*dy + dz*dz;
const double distance = sqrt( d2 );

delay_time = distance / _PROPAGATION;
```

Come sappiamo per questo canale il tempo di delay tra due nodi varia in base alla posizione dei due e alla costante di propagazione che si è scelta per la simulazione. A questo punto si salva il vecchio valore del delay assegnandolo al campo `oldChannelDelay` della matrice `_transmissions` e si aggiorna il valore `channelDelay` con quello appena calcolato:

```
    // Updating the channel delay:
_transmissions[ ID ][ recv_id ].channelDelay =
    sc_core::sc_time( delay_time, sc_core::SC_SEC );
```

Questi sono tutti i dati necessari al canale per dedicarsi poi all'aggiornamento degli eventi relativi ad un nodo.

Viene a questo punto chiamato il metodo `registerAction()` della classe `_EventsQueue_t` che registra una callback al metodo `_updateEvents()` a tempo corrente di simulazione:

```
// After updating the nodes properties we register a method
// that updates the nodes-relative events:

_eventsQueue->registerAction(
    currentTime,
    this,
    reinterpret_cast< mptr_t >( & DelayedSharedChannel_t::_updateEvents ),
    & _transmissions[ ID ][ recv_id ].idsInfos
    );
}
```

Aggiornamento degli eventi associati al nodo

La funzione `_updateEvents()` è adibita all'aggiornamento degli eventi già registrati nella Events Queue relativi ai nodi che compiono un movimento. Come già accennato precedentemente si hanno due code di eventi:

- startEventsIds: tiene traccia degli eventi relativi all'inizio della trasmissione;
- completeEventsIds: tiene traccia degli eventi relativi alla fine della trasmissione.

Descriviamo passo per passo il funzionamento del sopraccitato metodo:

Fase 1 - Aggiornamento delle trasmissioni: in questa fase si calcola il tempo corrente di simulazione, necessario nella quarta e ultima fase del metodo per fare i controlli di corruzione sui pacchetti.

Fase 2 - Aggiornamento degli eventi relativi alle _startSingleSending: si controlla che la coda relativa agli eventi di inizio trasmissione non sia vuota. Fatto ciò si scorrono tramite un ciclo for tutti gli elementi e si salva per ogni evento il vecchio valore in cui esso doveva essere scatenato. A questo punto si calcola il nuovo tempo determinato dalle nuove coordinate e si chiama il metodo `_updateAction()` della Events Queue per modificare il tempo in cui l'evento dovrà essere scatenato. Per ultimo, tramite un ciclo for si scorre la lista dei pacchetti associati al nodo e si cambiano i tempi di invio.

Fase 3 - Aggiornamento degli eventi relativi alle _completeSingleSending: i passaggi fatti in questa fase sono i medesimi della fase precedente, ma c'è una differenza all'interno del ciclo for che scorre la lista degli eventi. In questo caso infatti se in testa alla lista si trova un evento relativo ad un pacchetto che è in fase di codifica, questo evento non deve essere aggiornato.

Fase 4 - Controllo di corruzione dei pacchetti: il controllo di corruzione viene effettuato solo nei pacchetti in fase di codifica. Ci sono essenzialmente due condizioni per il controllo di corruzione. Nella prima, si controlla se un nodo che ha effettuato un movimento si è spostato dall'esterno all'interno del range di trasmissione di un altro nodo. In questo caso bisogna controllare se il nodo aveva già iniziato la trasmissione di un pacchetto e in caso positivo tale pacchetto dovrà essere marcato come corrotto. La seconda, contrariamente, controlla se un movimento di un nodo ha portato lo stesso dall'interno all'esterno del range di trasmissione mentre stava codificando un pacchetto. In questo caso il pacchetto deve essere marcato come corrotto in quanto una parte di esso non è stato codificato. In entrambi i casi comunque il fatto di aver marcato il pacchetto come corrotto permette al simulatore di non invocare la `receive()` su tale pacchetto.

4.5 Validazione del canale Delayed Shared

In questa sezione vengono descritti i test creati per validare il canale Delayed Shared. Alcuni test non verranno descritti in quanto sono esattamente gli stessi usati per il canale Shared, quindi ne analizzeremo solo alcuni, che saranno comunque molto rappresentativi per le caratteristiche del canale. Anche qui verranno utilizzate alcune immagini per aiutarci nel comprendere meglio tutte le dinamiche del canale.

4.5.1 Test_delayedshared_tlm

Questo test è stato creato per mettere in luce la cosa più importante che distingue i due canali: il tempo di propagazione dei pacchetti nella rete. Si noterà infatti che Task a distanze diverse avranno tempi di ricezione diversi. In questo test sono infatti presenti tre Task TLM, in tre punti punti diversi dello spazio che comunicano tra loro. In particolare sono stati creati due diversi scenari che ora descriviamo:

1. il Task0 spedisce pacchetti mentre il Task1 e il Task2 ricevono solamente. Non vi è nessun tipo di collisione quindi i pacchetti vengono ricevuti tutti. La peculiarità di questo test è far vedere che i due Task ricevitori hanno tempi di ricezione diversi a seconda della loro posizione. In Figura 4.5 si può vedere la posizione dei nodi nella rete simulata ed inoltre possiamo vedere nel seguente spezzone di file di LOG i tempi di invio e ricezione dei pacchetti:

```
LOG 0 s Task0 SEND data: I size: 1.
LOG 0 s Task0 <> send().
LOG 0 s DelayedSharedChannel packet 0 <> send().
LOG 58515229985 ps Task1 carrier: 1.
LOG 80 ms Task0 SEND data: L size: 1.
```

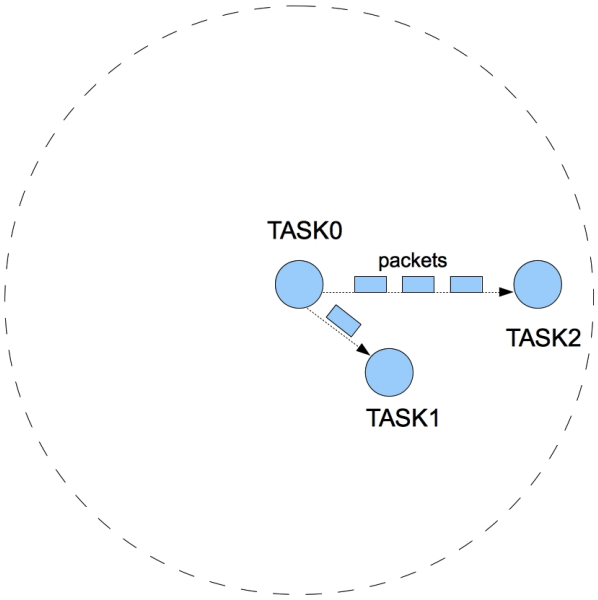


Figura 4.5: test_delayed_shared_tlm.

```
LOG 80 ms Task0 <> send().
LOG 80 ms DelayedSharedChannel packet 1 <> send().
LOG 117030459971 ps Task2 carrier: 1.
LOG 138515229985 ps Task1 RECEIVE data: I, size: 1.
LOG 160 ms Task0 SEND data: C size: 1.
LOG 160 ms Task0 <> send().
LOG 160 ms DelayedSharedChannel packet 2 <> send().
LOG 197030459971 ps Task2 RECEIVE data: I, size: 1.
LOG 218515229985 ps Task1 RECEIVE data: L, size: 1.
LOG 240 ms Task0 SEND data: P size: 1.
LOG 240 ms Task0 <> send().
LOG 240 ms DelayedSharedChannel packet 3 <> send().
LOG 277030459971 ps Task2 RECEIVE data: L, size: 1.
LOG 298515229985 ps Task1 RECEIVE data: C, size: 1.
LOG 320 ms Task0 SEND data: S size: 1.
LOG 320 ms Task0 <> send().
LOG 320 ms DelayedSharedChannel packet 4 <> send().
LOG 357030459971 ps Task2 RECEIVE data: C, size: 1.
```

Si nota che il Task1, essendo più vicno del Task2 rispetto al sender, riceve i pacchetti a tempi inferiori. Questo è dovuto al tempo di propagazione dei pacchetti variabile in base alla distanza tra i Task.

2. il Task0 spedisce pacchetti mentre il Task1 e il Task2 inizialmente ricevono solamente. Ad un certo istante iniziano a trasmettere, ma solo per un pò di tempo e poi tornano a ricevere solamente. La posizione dei nodi è la stessa del punto precedente, quindi si può vedere in Figura 4.5. Ciò è stato creato per far vedere le collisioni tra i pacchetti nel

periodo in cui tutti stanno trasmettendo. Si vedrà nello spezzone del file di LOG inoltre come vengono gestiti i carrier dei Task.

```
LOG 0 s Task0 SEND data: I size: 1.
LOG 0 s Task0 <> send().
LOG 0 s DelayedSharedChannel packet 0 <> send().
LOG 58515229985 ps Task1 carrier: 1.
LOG 80 ms Task0 SEND data: L size: 1.
LOG 80 ms Task0 <> send().
LOG 80 ms DelayedSharedChannel packet 1 <> send().
LOG 117030459971 ps Task2 carrier: 1.
LOG 138515229985 ps Task1 RECEIVE data: I, size: 1.
LOG 160 ms Task0 SEND data: C size: 1.
LOG 160 ms Task0 <> send().
LOG 160 ms DelayedSharedChannel packet 2 <> send().
LOG 170 ms Task2 SEND data: P size: 1.
LOG 170 ms Task2 <> send().
LOG 170 ms DelayedSharedChannel packet 3 <> send().
LOG 170 ms Task1 SEND data: S size: 1.
LOG 170 ms Task1 <> send().
LOG 170 ms DelayedSharedChannel packet 4 <> send().
LOG 228515229985 ps Task0 carrier: 1.
LOG 240 ms Task0 SEND data: K size: 1.
LOG 240 ms Task0 <> send().
LOG 240 ms DelayedSharedChannel packet 5 <> send().
LOG 250 ms Task2 SEND data: L size: 1.
LOG 250 ms Task2 <> send().
LOG 250 ms DelayedSharedChannel packet 6 <> send().
LOG 250 ms Task1 SEND data: R size: 1.
LOG 250 ms Task1 <> send().
LOG 250 ms DelayedSharedChannel packet 7 <> send().
```

In questa prima parte si vede che a tempo 170 ms anche il Task1 e il Task2 iniziano a spedire. Da questo punto in poi infatti non ci sono più `RECEIVE`. Si vede inoltre il carrier del Task0 che viene settato a 1, cioè canale occupato. Questa situazione di collisione continua fino a tempo 490 ms, quando il Task1 e il Task2 eseguono il loro ultimo invio. A tempo 687 ms si nota come il carrier del Task0 venga settato a 0 quindi canale libero e a tempo 757 ms i nodi ritornino a ricevere in quanto non vi sono più collisioni:

```
LOG 490 ms Task2 SEND data: B size: 1.
LOG 490 ms Task2 <> send().
LOG 490 ms DelayedSharedChannel packet 15 <> send().
LOG 490 ms Task1 SEND data: P size: 1.
LOG 490 ms Task1 <> send().
LOG 490 ms DelayedSharedChannel packet 16 <> send().
LOG 560 ms Task0 SEND data: B size: 1.
LOG 560 ms Task0 <> send().
LOG 560 ms DelayedSharedChannel packet 17 <> send().
LOG 640 ms Task0 SEND data: W size: 1.
LOG 640 ms Task0 <> send().
LOG 640 ms DelayedSharedChannel packet 18 <> send().
LOG 687030459971 ps Task0 carrier: 0.
LOG 720 ms Task0 SEND data: L size: 1.
LOG 720 ms Task0 <> send().
LOG 720 ms DelayedSharedChannel packet 19 <> send().
LOG 757030459971 ps Task2 RECEIVE data: B, size: 1.
LOG 778515229985 ps Task1 RECEIVE data: W, size: 1.
```

```
LOG 800 ms Task0 SEND data: L size: 1.
LOG 800 ms Task0 <> send().
LOG 800 ms DelayedSharedChannel packet 20 <> send().
LOG 837030459971 ps Task2 RECEIVE data: W, size: 1.
LOG 858515229985 ps Task1 RECEIVE data: L, size: 1.
LOG 880 ms Task0 SEND data: S size: 1.
LOG 880 ms Task0 <> send().
LOG 880 ms DelayedSharedChannel packet 21 <> send().
LOG 917030459971 ps Task2 RECEIVE data: L, size: 1.
```

4.5.2 Test_delayedshared_collision_tlm

Il corrente test è uno dei più rappresentativi per questo tipo di canale. E' stato creato infatti per vedere come incide la posizone dei nodi nella ricezione dei pacchetti. Si hanno qui quattro nodi posizionati come si vede in Figura 4.6. Due di questi hanno il ruolo di sender (Task0, Task3), mentre gli altri due hanno il ruolo di receiver (Task1, Task2). Tutti e quattro i nodi sono all'interno del range di trasmissione di entrambi i nodi sender. Possiamo riassumere il test in questi punti:

1. il Task0 invia un pacchetto a tempo 0 ms;
2. il Task1 inizia a sentire il pacchetto inviato dal Task0 a tempo 898 ms;
3. il Task3 invia un pacchetto a tempo 900 ms;
4. il Task1 esegue la `RECEIVE` del pacchetto inviato dal Task0 a 978 ms;
5. il Task2 sente il pacchetto inviato dal Task0 a 1351 ms, ma poco dopo arriva anche il pacchetto inviato dal Task3, quindi si crea una collisione e non riceve nessuno dei pacchetti;
6. il Task3 esegue la `RECEIVE` del pacchetto inviato dal Task0 a tempo 1884 ms;
7. il Task1 esegue la `RECEIVE` del pacchetto inviato dal Task3 a tempo 1885 ms;
8. il Task0 esegue la `RECEIVE` del pacchetto inviato dal Task3 a tempo 2784 ms.

La posizione del Task2 fa si che lui non riceva nessun pacchetto a causa delle collisioni, mentre gli altri Task non sono coinvolti in collisioni e quindi ricevono i pacchetti come desiderato. Vediamo nel file di LOG sia le posizioni esatte dei nodi nella rete, sia le dinamiche descritte sopra:

```
LOG 0 s DelayedSharedChannel Position of Node0: x = 0 y = 0 z = 0.
LOG 0 s DelayedSharedChannel Position of Node1: x = 1330 y = 0 z = 0.
LOG 0 s DelayedSharedChannel Position of Node2: x = 2000 y = 0 z = 0.
LOG 0 s DelayedSharedChannel Position of Node3: x = 2670 y = 0 z = 0.
LOG 0 s Task0 SEND data: I size: 1.
LOG 0 s Task0 <> send().
```

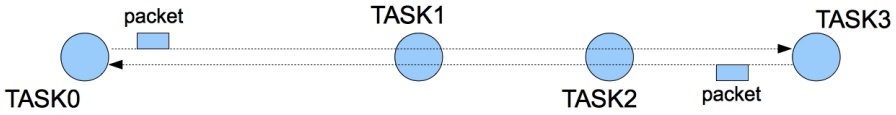


Figura 4.6: test_delayed_shared_collision_tlm. Non vengono riportati i range di trasmissione, tutti sentono tutto.

```
LOG 0 s DelayedSharedChannel packet 0 <> send().
LOG 898648648649 ps Task1 carrier: 1.
LOG 900 ms Task3 SEND data: L size: 1.
LOG 900 ms Task3 <> send().
LOG 900 ms DelayedSharedChannel packet 1 <> send().
LOG 978648648649 ps Task1 RECEIVE data: I, size: 1.
LOG 978648648649 ps Task1 carrier: 0.
LOG 1351351351351 ps Task2 carrier: 1.
LOG 1432702702703 ps Task2 carrier: 0.
LOG 1804054054054 ps Task3 carrier: 1.
LOG 1805405405405 ps Task1 carrier: 1.
LOG 1884054054054 ps Task3 RECEIVE data: I, size: 1.
LOG 1884054054054 ps Task3 carrier: 0.
LOG 1885405405405 ps Task1 RECEIVE data: L, size: 1.
LOG 1885405405405 ps Task1 carrier: 0.
LOG 2704054054054 ps Task0 carrier: 1.
LOG 2784054054054 ps Task0 RECEIVE data: L, size: 1.
```

Capitolo 5

Cosimulazione e tool di simulazione

In questo capitolo si descrive come avviene il meccanismo della co-simulazione Matlab/Simulink - SCNSL e come applicare tale metodica al problema del controllo di formazione. In sezione 5.1 sono specificate le decisioni progettuali prese allo scopo di creare ambienti di simulazione essenziali a dimostrare come incide l'introduzione della rete in un controllo di formazione. Nella sezione successiva 5.2 viene effettuata una descrizione del controllore Matlab necessario alla creazione delle traiettorie dei veicoli usati nei vari test. Tale controllore ha lo scopo principale di aggiornare la velocità di ogni veicolo per portarlo nel più breve tempo possibile in formazione rispetto ad un veicolo leader. Portarsi in formazione significa rispettare alcuni vincoli di distanza rispetto ad altri veicoli che la compongono. La sezione 5.3 è dedicata alla co-simulazione; si analizzano quindi gli strumenti necessari per introdurre una rete wireless per lo scambio di messaggi tra i veicoli e si descrive come l'utente finale può far uso di questi strumenti. Nell'ultima sezione 5.4 si descrive quali sono le scelte progettuali che hanno portato alla creazione di uno scenario utilizzato per la validazione degli strumenti di simulazione.

5.1 Assunzioni di progetto

Da questo capitolo in avanti tratteremo il problema di controllo di formazione utilizzando scenari progettati sulla base delle seguenti ipotesi:

- i veicoli sono descritti dal loro modello cinematico senza tenere conto di alcuna dinamica (punto materiale);
- i veicoli possono ruotare liberamente senza essere limitati da vincoli di non olonomia;
- non sono presenti forze di attrito e turbolenze;

- i sensori per il rilevamento della posizione forniscono le misure senza ritardo e senza rumore;
- si usa un modello di decadimento isotropo dell'energia del segnale wireless;
- il calcolo della raggiungibilità di un nodo ricevitore avviene considerando la potenza trasmissiva del sender e la soglia di ricezione del receiver, come già specificato in sezione 4.4.1 nella quale si descrive la fase di inizializzazione dei nodi in SCNSL, grazie alla seguente formula:

```
const bool pow_reac = sp.transmission_power /
        pow( distance, _ALPHA ) >= rp.receiving_threshold;
```

  con l'esponente `_alpha` impostato al valore 1. Questa impostazione permette una notevole agevolazione dei calcoli in quanto è possibile misurare il raggio del range trasmissivo di un veicolo in metri. Per esempio se un veicolo possiede una potenza trasmissiva con valore 100 significa che il suo range trasmissivo ha un raggio esattamente di 100 metri.

Queste assunzioni di progetto sono da considerare di notevole impatto per la progettazione di scenari per il controllo di formazione ma sono giustificate in quanto l'obiettivo della tesi non è progettare un controllo di formazione ottimale ma studiare l'incidenza del canale di trasmissione sulle prestazioni.

5.2 Il controllore

Negli ambienti di co-simulazione creati si è scelto di utilizzare un sistema di controllo dei veicoli come quello rappresentato in Figura 5.1.

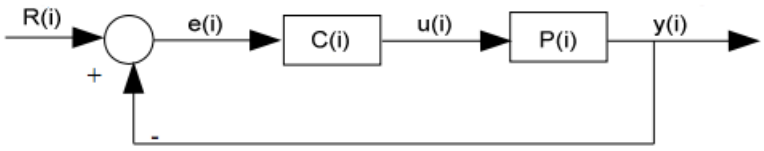


Figura 5.1: Schema a blocchi semplificato di controllo di un veicolo.

Tale schema rappresenta un controllo in retroazione nella quale il valore in uscita dipende sia dal valore della variabile controllata sia da valori di altre variabili dipendenti del sistema controllato per cui è sempre presente almeno un percorso di segnale chiuso detto *anello di retroazione*.

Le principali componenti di questo sistema sono:

- $R(i)$: è il valore che si vuole fare assumere alla variabile controllata;

- $Y(i)$: il valore effettivamente assunto dalla variabile controllata;
- $e(i)$: il segnale di errore $e(i) = R(i) - y(i)$ calcolato dal controllore $C(i)$;
- $C(i)$: il controllore che ha il compito di fornire un comando $u(i)$ appropriato in funzione dell'errore riscontrato;
- $P(i)$: il processo che fornisce il segnale d'uscita $y(i)$;
- $u(i)$: il segnale fornito in ingresso al processo $P(i)$.

Il modulo di maggior interesse è il controllore $C(i)$ che è di tipo PID. Il controllore proporzionale-integrale-derivativo è un meccanismo di controllo ampiamente utilizzato nei sistemi di controllo industriali. Esso acquisisce in ingresso un valore da un processo, e lo confronta con un valore di riferimento. La differenza, il cosiddetto segnale di errore $e(i)$, viene quindi usata per determinare il valore della variabile di uscita del controllore, che è la variabile manipolabile del processo. L'algoritmo di calcolo di un controllore PID coinvolge tre azioni:

- azione proporzionale $u_P = K_P e$ dove e è l'errore e K_P una costante;
- azione integrale $u_I = K_I \int e(t)\,dt$ proporzionale all'integrale nel tempo del segnale di errore e, moltiplicato per la costante K_I;
- azione derivativa $u_D = K_D \frac{de}{dt}$ con K_D costante;

Tali valori vengono sommati algebricamente per creare il segnale di uscita $u(i)$.

Per il lavoro di questa tesi si è deciso di utilizzare Matlab/Simulink per modellare il controllore e la cinematica del veicolo da controllare. Tale codice verrà descritto dettagliatamente nella prossima sezione.

5.2.1 Analisi del codice

Dopo aver descritto in Sezione 5.2 il funzionamento di un controllore di tipo PID generico procediamo a questo punto con l'analisi del controllore PID implementato per questo lavoro di tesi descrivendone il funzionamento passo passo a partire dai dati di input sino ad arrivare ai dati in output e mostrando il codice utilizzato per l'implementazione del controllore.

Per procedere con l'analisi consideriamo il caso di invio di un pacchetto all'istante t da parte di un veicolo *Leader* e la ricezione di tale pacchetto da parte di uno dei suoi inseguitori (veicolo *follower*). Indichiamo con L ed F i veicoli *Leader* e *Follower* rispettivamente. Nell' idea di formazione illustrata in Figura 5.2 ogni veicolo deve seguire il proprio *Leader* in modo da raggiungere e mantenere una distanza da esso prefissata. Il controllore del

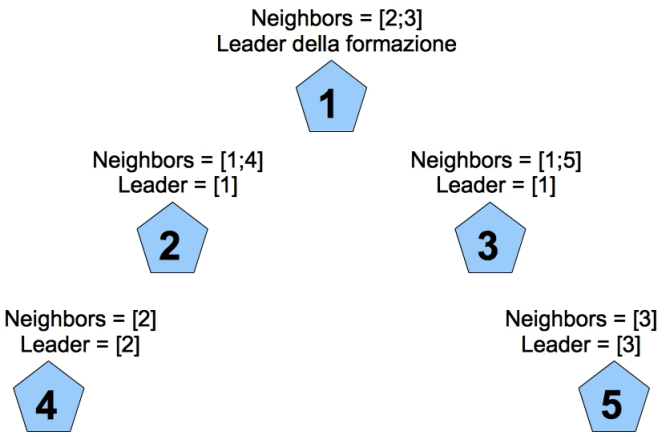


Figura 5.2: Veicoli in formazione con indicazione del Leader e dei Neighbors.

Follower tiene traccia dell'indice corrispondente a tale veicolo e interviene nel momento in cui riceve un pacchetto da quest'ultimo.

Per verificare che il pacchetto proviene effettivamente dal veicolo *Leader* si effettua un controllo grazie al seguente codice:

```
flag_id = find( (Leader-vehicle_id) == 0, 1 );
```

La variabile `vehicle_id` contiene l'identificatore del veicolo che ha inviato il pacchetto mentre la variabile `Leader` contiene l'indice del leader del veicolo. Se nella variabile `flag` viene caricato un valore ho la sicurezza che il messaggio è stato inviato dal *Leader* quindi procedo con il calcolo del nuovo comando altrimenti scarto il pacchetto restituendo il comando precedente. Il controllore $C(F)$ all'istante t riceve quindi in ingresso due vettori. Il primo vettore $R_k(L)$ proviene dal pacchetto ricevuto e contiene le informazioni relative al veicolo *Leader*. Tale vettore ed è definito nel seguente modo:

$$R_k(L) = \begin{bmatrix} x_k(L) \\ v_k(L) \end{bmatrix}$$

dove $x_k(L)$ è la posizione e $v_k(L)$ la velocità scomposte nei due assi del veicolo *Leader*. Il secondo vettore $y_k(F)$ giunge dall'*anello di retroazione* definito in Sezione 5.2 ed è definito come segue:

$$y_k(F) = \begin{bmatrix} x_k(F) \\ v_k(F) \end{bmatrix}$$

dove $x_k(F)$ e $v_k(F)$ sono rispettivamente la posizione e la velocità scomposte nei due assi del veicolo *Follower*. A questo punto il controllore procede

con il calcolo del vettore d'errore $E_k(F)$ come differenza tra le velocità e posizioni dei veicoli *Follower* e *Leader*:

$$E_k(F) = \begin{bmatrix} e_k(F) \\ \dot{e}_k(F) \end{bmatrix} = \begin{bmatrix} x_k(L) \\ v_k(L) \end{bmatrix} - \begin{bmatrix} x_k(F) \\ v_k(F) \end{bmatrix}$$

Il vettore $e_k(F)$ contiene l'errore di posizione mentre $\dot{e}_k(F)$ l'errore di velocità. I valori $e_k(F)$ e $\dot{e}_k(F)$ tendono al valore `delta(i)` che verrà descritto nel prossimo paragrafo. Tale calcolo nel codice del controllore viene suddiviso in due fasi. La prima fase calcola i valori d'errore di posizione nei due assi:

```
error_x = vehicle_x + delta(1) - my_x;
error_y = vehicle_y + delta(2) - my_y;
```

dove `vehicle_x` e `vehicle_y` sono le posizioni negli assi del veicolo *Leader* mentre `my_x` e `my_y` sono le posizioni negli assi del veicolo *Follower*. Il valore `delta(i)` rappresenta l'offset tra il veicolo *Leader* e il veicolo *Follower* nel sistema di riferimento globale; più precisamente il prodotto dell'offset caratteristico della formazione per la matrice di rotazione all'istante k che lega il sistema di riferimento del veicolo con quello globale. La seconda fase invece calcola i valori d'errore delle velocità i quali sono da considerarsi come la derivata d'errore:

```
error_vel_x = vehicle_vel_x - my_vel_x;
error_vel_y = vehicle_vel_y - my_vel_y;
```

dove `vehicle_vel_x` e `vehicle_vel_y` sono le velocità del *Leader* mentre `my_vel_x` e `my_vel_y` sono le velocità del *Follower* nei due assi.

Successivamente vengono coinvolte le tre azioni del controllore PID descritte in Sezione 5.2. Tali azioni sono necessarie per il calcolo di $u_k(F)$ (l'output del controllore) da passare in input al processo $P(F)$ il quale restituirà in uscita $y_{k+1}(F)$ cioè il nuovo comando. La formula per il calcolo di $u_k(F)$ è la seguente:

$$u_k(F) = (K_P + K_I T_s)e_k(F) + K_D\dot{e}_k(F) + u_{k-1}(F)$$

definita quindi come somma algebrica delle tre azioni:

- azione proporzionale: $K_P e_k(F)$ dove K_P è una costante detta *costante di proporzionalità* e $e_k(F)$ corrisponde all'errore di posizione;
- azione integrale: $K_I T_s e_k(F) + u_{k-1}(F)$ dove K_I è una costante detta *costante integrale*, $u_{k-1}(F)$ è il comando precedente mentre T_s è il tempo di campionamento del controllore a tempo discreto;
- azione derivativa: $K_D\dot{e}_k(F)$ dove K_D è una costante detta *costante derivativa* e $\dot{e}_k(F)$ corrisponde all'errore di velocità che è la derivata dell'errore di posizione;

La costante K_P, detta anche costante di proporzionalità, è quella costante che permette al controllore di agire rapidamente sul sistema sotto controllo. Più precisamente, se si aumenta K_P, diminuisce il tempo di salita (ossia aumenta la prontezza di risposta del mio sistema). La costante derivativa K_D invece aumenta lo sforzo del controllo e rende minore il tempo di assestamento.

Il codice che implementa il calcolo di $u_k(F)$ è mostrato di seguito:

```
command_x = 0.01*error_x + 0.1*error_vel_x + x(1);
command_y = 0.01*error_y + 0.1*error_vel_y + x(2);

sys = [command_x; command_y; x(3)+1];
```

Si nota che l'implementazione del controllore prevede una suddivisione dei calcoli per ogni asse di riferimento. La costante 0.01 rappresenta la costante di proporzionalità più la costante integrale mentre la costante 0.1 è la costante derivativa scelta. Infine `sys` corrisponde esattamente a $u_k(F)$ e cioè al segnale fornito in ingresso al processo $P(F)$.

5.3 Cosimulazione mediante utilizzo dei tool

In questo capitolo fino ad ora si è trattato esclusivamente l'aspetto di controllo dei veicoli. Per procedere con il lavoro di tesi è necessario a questo punto introdurre la simulazione della rete per lo scambio di messaggi. Per fare ciò è necessario adottare la tecnica della co-simulazione ampiamente descritta in Sezione 2.3. La tecnica di co-simulazione applicata in questo lavoro di tesi fa uso di un wrapper Matlab il quale comunica con un wrapper SCNSL attraverso porte di input e di output mappate. Tali wrapper verranno descritti nelle prossime sezioni di questo capitolo.

5.3.1 Il wrapper Matlab

Per permettere la co-simulazione è stato necessario lo sviluppo di un wrapper Matlab per garantire la comunicazione tramite socket bloccanti tra Matlab/Simulink e SystemC/SCNSL. Il wrapper deve essere adattato alle esigenze di progetto, in particolare devono essere mappate le porte di input e le porte di output.

Nei successivi casi di studio il wrapper possiede per ogni veicolo 15 porte di input e 10 porte di output definite tramite dei blocchi Matlab. Tra le porte di input si trovano le porte di controllo e le porte dati. Nello specifico, come si può notare in Figura 5.3, si hanno 8 porte di controllo:

1. `_label:` indica la priorità con cui deve essere inviato il pacchetto;
2. `_start_send:` è un numero progressivo che viene incrementato ogniqualvolta si spedisce un pacchetto;

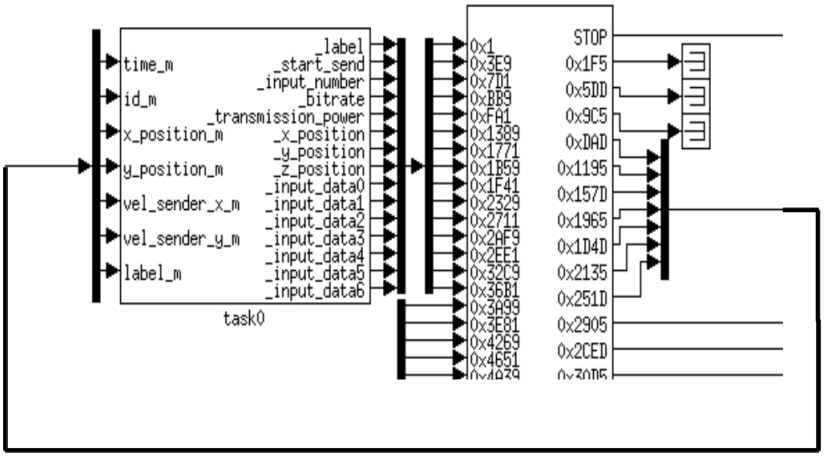


Figura 5.3: Parte del wrapper Matlab. Si notano le porte in input verso SCNSL (sinistra), le porte in input verso Matlab (destra) e i relativi indirizzi.

3. `_input_number:` contiene un numero intero che indica il numero di dati presenti nel pacchetto. Le porte dati dovranno essere quindi in numero uguale a quello indicato in questa porta;

4. `_bitrate:` indica con quale bitrate sta trasmettendo il veicolo;

5. `_transmission_power:` indica la potenza trasmissiva con cui sta trasmettendo il veicolo;

6. `_x_position:` contiene la posizione nell'asse x del veicolo al momento della creazione del pacchetto;

7. `_y_position:` contiene la posizione nell'asse y del veicolo al momento della creazione del pacchetto;

8. `_z_position:` contiene la posizione nell'asse z del veicolo al momento della creazione del pacchetto.

Tra le porte di input ci sono, come accennato in precedenza, anche le porte dati. Esse sono 7 e sono:

1. `_input_data0:` contiene il tempo di simulazione in cui si spedisce il pacchetto;

2. `_input_data1:` contiene l'identificativo del veicolo che ha creato il pacchetto;

3. `_input_data2:` contiene la posizione nell'asse x del veicolo che ha creato il pacchetto;

4. `_input_data3:` contiene la posizione nell'asse y del veicolo che ha creato il pacchetto;

5. `_input_data4:` contiene la velocità nell'asse x del veicolo al momento della creazione del pacchetto dati inviato;

6. `_input_data5:` contiene la velocità nell'asse y del veicolo al momento della creazione del pacchetto dati inviato;

7. `_input_data6:` contiene la label del pacchetto, che indica con quale priorità si deve inviare lo stesso.

Ogni porta ha un suo indirizzo univoco che servirà poi a lato SCNSL, descritto nella prossima sezione, per leggere e scrivere sulle porte.

5.3.2 Il wrapper Systemc

In aggiunta al wrapper Matlab e stato necessario sviluppare un wrapper SystemC tramite la creazione di un particolare Task SCNSL di tipo `SimulinkTask_t`. Tale wrapper è costituito da porte di ingresso e di uscita e da thread SystemC che svolgono differenti task. Ingressi e uscite del wrapper Systemc sono aree di memoria riservate alle quali sono mappate ingressi e uscite del wrapper Matlab e altri elementi del modello Simulink. Questa configurazione permette quindi al wrapper SystemC di:

- elaborare le informazioni contenute nel payload dei pacchetti ricevuti e scrivere tali informazioni nelle aree di memoria utilizzate da Matlab come porte di input;

- ricevere i dati scritti sulle porte output di Matlab e creare il pacchetto destinato alla rete SCNSL;

- ricevere da Matlab i dati aggiornati delle proprietà del nodo e settare tali proprietà.

Le porte utilizzate dal wrapper SystemC sono create all'interno del costruttore del Task SCNSL. A questo scopo sono utilizzati due offset, uno per le porte d'ingresso e uno per le porte d'uscita, i quali indicano il punto di partenza per l'allocazione della memoria. Tali valori sono passati come parametro al costruttore e sono di tipo `offset_t`. Essi sono necessari in quanto possono essere presenti più nodi all'interno della rete simulata e quindi ad ogni nodo saranno riservate delle aree di memoria differenti.

I task che il wrapper deve svolgere sono riservati alle thread chiamate `_manager`, `_thread` e `_updateThread`. La `_manager` è di fondamentale importanza in quanto svolge un ruolo particolare di gestione delle altre due thread sollevando, quando necessario, eventi che permettono alle thread di

iniziare la loro computazione. Nel momento in cui il metodo `_sendingRoutine()` viene invocato, la thread `_manager` decide di sollevare o meno l'evento `_data_arrived` che sblocca la `_thread` e l'evento `_update_properties` che sblocca la `_updateThread` . Verranno analizzate ora le implementazioni dei task svolti da tali thread e descritti all'inizio della sezione.

Ricezione dei pacchetti e scrittura sulle porte di output

Il wrapper SystemC prevede l'utilizzo del metodo `b_transport` della classe `SimulinkTask_t` modificato ad hoc per gestire la co-simulazione con Matlab. Come primo compito tale metodo, al momento della ricezione di un pacchetto, effettua la lettura della label dal payload e scrive tale valore sulla porta dedicata chiamata `_output_label` in questo modo:

```
_output_label.write( &matlab_label, sizeof( double ) );
```

Il passo successivo è avvisare Matlab della quantità di dati del payload che verranno scritti sulle porte di output. Ciò è necessario in quanto Matlab conoscerà il numero di porte dati totali sulle quali effettuare la lettura (ogni singolo dato è scritto su una specifica porta di output). Si effettua quindi la scrittura sulla porta `_output_number` del valore della dimensione dei dati del payload:

```
length = p.get_data_length() / sizeof( double );
double matlab_length =
    to_systemc( static_cast< double >( length ));

_output_number.write( &matlab_length, sizeof( double ) );
```

Il task più importante del metodo `b_transport` è la scrittura in output del contenuto del payload. Si memorizzano su un buffer temporaneo i dati del payload e successivamente si scrive ogni singolo dato sulla sua porta di output associata avvisando l'utente tramite file di log dell'avvenuta scrittura:

```
buffer = reinterpret_cast< double * >( p.get_data_ptr() );

for ( register unsigned int i = 0; i < length; ++i )
{
    datum = to_systemc( buffer[ i ] );
    _output_data[ i ].write( &datum, sizeof( double ) );
}
```

Creazione dei pacchetti da Matlab e inoltro nella rete

Nel momento in cui l'evento `_data_arrived` viene sollevato significa che Matlab ha scritto nuovi dati nelle porte di input ed è necessario effettuare una lettura per poi creare il pacchetto da inoltrare nella rete. La thread `_thread`, inizialmente, legge dalla porta `_start_send` per verificare se effettivamente sono stati scritti dei nuovi dati confrontando il precedente valore memorizzato con il nuovo valore appena letto:

```
if( ! ( _old_start_send > systemc_start_send
    || _old_start_send < systemc_start_send ) )
    continue;
```

Se effettivamente sono stati scritti nuovi dati da Matlab si procede leggendo dalla porta `_input_number` il numero di dati scritti e successivamente si effettua la lettura dalle porte `_input_data` memorizzando i dati in un buffer temporaneo. Tutto ciò avviene attraverso il seguente codice:

```
_input_number.read( & length );

for ( register unsigned int i = 0; i < systemc_length; ++i )
{
    _input_data[ i ].read( & datum );
    buffer[ i ] = to_systemc( datum );
}
```

Infine la thread, dopo aver memorizzato la label dalla lettura sulla porta `_input_label`, effettua una chiamata al metodo `send()` che si occupa di creare il pacchetto e inoltrarlo nella rete attraverso le seguenti righe di codice:

```
_input_label.read( & label );

TlmTask_if_t::send(
    0,
    reinterpret_cast<Scnsl::Core::byte_t*>( buffer ),
    static_cast< size_t >( systemc_length * sizeof( double ) ),
    static_cast< label_t >( systemc_label ) );
```

Aggiornamento delle proprietà dei nodi

La thread `_updateThread` ha il compito di leggere e aggiornare, se necessario, le proprietà del singolo nodo. In ogni momento della simulazione Matlab può scrivere sulle porte di input nuovi valori per specifiche proprietà dei nodi come ad esempio potenza trasmissiva, posizione, bitrate ecc... Se l'evento `_update_properties` viene sollevato la thread legge e memorizza le proprietà attuali del nodo associato:

```
const Scnsl::Core::Node_t::node_properties_t & np =
    _node->getProperties( _channel );
```

Successivamente effettua una lettura completa su tutte le porte di input relative alle proprietà dei nodi:

```
_bitrate.read( & datum );
_transmission_power.read( & datum );
_x_position.read( & datum );
_y_position.read( & datum );
_z_position.read( & datum );
```

Le proprietà attuali del nodo vengono ricavate con lo scopo di confrontarle con i valori delle proprietà appena letti. Se almeno un valore risulta essere diverso rispetto al valore attuale, è necessario aggiornare le proprietà dei nodi attraverso la chiamata al metodo `setProperties()` della classe `Node_t`. Di seguito il codice che descrive tali passaggi:

```
if( systemc_bitrate < np.bitrate || systemc_bitrate > np.bitrate
    || systemc_transmission_power < np.transmission_power
    || systemc_transmission_power > np.transmission_power
    || systemc_x_position < np.x || systemc_x_position > np.x
    || systemc_y_position < np.y || systemc_y_position > np.y
    || systemc_z_position < np.z || systemc_z_position > np.z )
{
    Scnsl::Core::Node_t::node_properties_t np1;

    // Update properties.
    np1.bitrate = systemc_bitrate;
    np1.transmission_power = systemc_transmission_power;
    np1.x = systemc_x_position;
    np1.y = systemc_y_position;
    np1.z = systemc_z_position;
    np1.receiving_threshold = np.receiving_threshold;

    // Set the node properties.
    _node->setProperties( np1, _channel, false);
}
```

5.3.3 I tool di simulazione

Nella precedente sezione sono stati analizzati gli strumenti di controllo dei veicoli e gli strumenti che permettono di effettuare la co-simulazione Matlab/Simulink - SCNSL. Per permettere all'utente di poterli utilizzare sono stati creati tre ambienti di simulazione:

- ambiente con Matlab puro;
- ambiente di cosimulazione Matlab/Simulink - SCNSL con shared channel;
- ambiente di cosimulazione Matlab/Simulink - SCNSL con delayed shared channel.

Il primo ambiente lavora in totale assenza di rete SCNSL, il secondo applica la tecnica della co-simulazione utilizzando il canale Shared come canale trasmissivo mentre il terzo applica anch'esso la tecnica della co-simulazione ma utilizzando il canale Delayed Shared.

L'obiettivo di questa sezione è descrivere innanzitutto il procedimento che permette all'utente finale di effettuare una simulazione utilizzando ognuno dei tre ambienti sopra citati. Successivamente validare questi tool di simulazione allo scopo di dimostrare che i risultati ottenuti sono tra loro diversi dando prova che, grazie a tool che sfruttano la tecnica della co-simulazione, è possibile ottenere risultati più precisi e svolgere analisi più accurate.

Durante lo sviluppo di questi tool uno dei punti fondamentali è stato rendere il più possibile automatico il processo di simulazione in modo da facilitare il lavoro all'utente finale. Grazie a un lavoro di scripting l'avvio

delle simulazioni avviene attraverso l'esecuzione di uno script bash (uno per ogni test) situato all'interno della cartella *scnsl/tests/Scripts*. Nella prima sezione di tali scripts, denominata *General Settings*, sono definite le variabili d'ambiente e i path necessari all'esecuzione delle simulazioni. In questo punto l'utente dovrà quindi impostare manualmente:

- il path del simulatore SCNSL;
- il path del software Matlab;
- il path della libreria `ld` della macchina ospite;
- le variabili necessarie all'esecuzione di SCNSL.

Di seguito sono mostrati i comandi contenuti all'interno dello script:

```
# Set the SCNSL path:
export SCNSL_TESTS_PATH=/home/giacomo/scnsl/tests

# Set the Matlab path:
export MATLAB_PATH=/home/giacomo/Matlab/bin/matlab

# Change LD_LIBRARY_PATH with the path of the libraries on your machine:
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/giacomo/scnsl/obj

# Set the system variables:
export ISS_PORT_BASE=4455
export ISS_NUM=1
export COSIM_TYPE=3
export VERBOSE=1
export VIRTUAL_TICK=0.005
```

Successivamente è presente il comando per l'esecuzione delle simulazioni. Nel caso di un test eseguito in ambiente Matlab puro è presente solamente il comando per l'esecuzione dello script Matlab:

```
# Start MATLAB:
time -f "& %Es & %Us & %Ss \\\\" $MATLAB_PATH -nodesktop -nosplash -r
"run $SCNSL_TESTS_PATH/test_validation_tools/$SIM_NAME.m" 2>>
$SCNSL_TESTS_PATH/test_validation_tools/Results/total_performance.tex
```

Lo script Matlab in questione è un file .m che contiene la configurazione dell'aspetto di controllo e ne esiste uno per ogni testcase che si desidera simulare nel test. Con testcase si intende un set di condizioni e variabili che l'utente utilizza per lo stesso test in modo da verificarne il comportamento sotto diverse condizioni. Il comando `time -f` permette di salvare i tempi di simulazione nel formato *Real time*, *User time* e *System time* all'interno del file `total _performance.tex`. Nel caso invece siano necessarie simulazioni con la presenza di SCNSL, quindi utilizzando l'ambiente di co-simulazione, si aggiunge in testa al precedente codice il comando per l'esecuzione dell'eseguibile del simulatore di rete SCNSL:

```
# Start SCNSL:
cd $SCNSL_TESTS_PATH/obj
./test_validation_tools $COSIM_TYPE $VERBOSE $VIRTUAL_TICK >
$RESULTS_FOLDER/log.txt &
```

Se si desidera eseguire più volte un singolo testcase in sequenza si utilizza un ciclo `while` che permette di svolgere un numero predefinito di simulazioni. La struttura dello script bash diventa quindi la seguente:

```
export SEED=1
export MAX_SIM_NUMBER=10
export MIN_SIM_NUMBER=0

...

SIM_NAME=caseCurveAlgorithm

while [  $MIN_SIM_NUMBER -lt $MAX_SIM_NUMBER ]; do

...

# Start SCNSL

# Start MATLAB

...

# New seed for rand():
SEED=$(( SEED + 1 ))

# Increase simulations number
MIN_SIM_NUMBER=$(( $MIN_SIM_NUMBER + 1))

done
```

L'insieme dei source file SCNSL che permettono la creazione dell'eseguibile e gli script Matlab per ogni testcase sono raggruppati in un'unica cartella all'interno di *scnsl/tests* rinominata con il nome del test. All'interno di tale cartella sono presenti i file `main.cc`, `SimulinkTask_t.cc` e `SimulinkTask_t.hh` i quali permettono di configurare la rete SCNSL per la simulazione e sono necessari per la creazione dell'eseguibile. Si trova inoltre il file `total_performance_template.tex`, un template di un Latex file nel quale il test inserisce i risultati delle simulazioni. I restanti file presenti in questa cartella si suddividono in script Matlab e modelli Matlab. Come già accennato ogni singolo testcase è descritto da uno di questi Matlab script il quale descrive la configurazione della parte di controllo in Matlab/Simulink. Tali scripts si suddivide in 3 sezioni:

1. Testcase settings;
2. Testcase configuration;
3. Figures and performances;

La sezione *Testcase settings* si occupa del caricamento nel Matlab workspace delle variabili utilizzate per la configurazione dell'ambiente di simulazione tramite l'esecuzione del Matlab scripts *settings*. Il contenuto di tale script è il seguente:

```
% Simulations settings
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Set Matlab Wrapper paths:
addpath /home/giacomo/esercitazione-ncs/matlab/Matlab-HSN/addons
addpath /home/giacomo/esercitazione-ncs/matlab/Matlab-HSN/classes
addpath /home/giacomo/esercitazione-ncs/matlab/Matlab-HSN/functions
addpath /home/giacomo/esercitazione-ncs/matlab/Matlab-HSN/models

% Set SCNSL path:
SCNSL_path = '/home/giacomo/scnsl';

% Set Tw:
Tw = 0.05;

% Set Ts:
Ts = 2e-1;

% esponente di attenuazione
att_exponent = 1;

% Set timeRange:
timeRange = Ts * 359;

% Set TsSub:
TsSub = 10;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Si nota come vengono inizializzati in questo punto il timestamp `Ts` e il tempo totale di simulazione `timeRange`. Come conseguenza al fatto che i path e il tempo di simulazione sono gli stessi per ogni testcase il file `settings` è unico e viene quindi caricato ad ogni esecuzione con gli stessi parametri.

La sezione *Testcase configuration* contiene la configurazione dell'utente per i veicoli utilizzati nel testcase. In questo punto l'utente specifica per ogni veicolo:

- le coordinate del punto che il veicolo deve mantenere in formazione;
- i veicoli `Neighbors` cioè i veicoli dai quali deve ricevere pacchetti;
- la velocità del veicolo nell'asse X e nell'asse Y (solo per il veicolo Leader).

Come esempio completo di codice viene riportata di seguito la configurazione del primo veicolo della formazione:

```
% Set Leader
x0_car1_basic = [30;30];
Formation1 = [0;0];
Neighbors1 = [];
command1.time = [0:Ts:timeRange];
l = length(command1.time);
x = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2
     3.5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5];
y = [5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
     5 3.5 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];
len_x = l/length(x);
len_y = l/length(y);
```

```
vel_x = transpose(interp(x,len_x));
vel_y = transpose(interp(y,len_y));
command1.signals.values = [vel_x vel_y];
command1.signals.dimensions = 2;
v = [x(1); y(1)];
d = v/norm(v);
rotation = [d(1) -d(2); d(2) d(1)];
```

Dopo la configurazione dei veicoli si procede con l'avvio vero e proprio della simulazione tramite il comando `sim` passando come parametro il modello Matlab che si desidera utilizzare. Al termine del comando `sim` si entra nella sezione *Figures and performance* la quale semplicemente esegue gli scripts *figures* e *performances* che, rispettivamente, visualizzano le serie di dati risultanti dall'esecuzione del testcase e calcolano le statistiche memorizzandole su file. E' interessante notare come avviene l'automatizzazione del calcolo delle performance. Si calcola per ogni veicolo il vettore dell'errore di posizione attraverso un calcolo che confronta i valori di posizione del veicolo nella simulazione con i suoi valori ideali:

```
error_2(k) = sqrt((X1(k) + gamma(1) - X2(k))^2 +
                  (Y1(k) + gamma(2) - Y2(k))^2);
```

Grazie a questi vettori è possibile calcolare per il testcase alcuni indici di particolare interesse come ad esempio l'errore medio, la varianza e lo scarto quadratico medio e in automatico inserirli all'interno di un file Latex per poterli visualizzare in forma tabellare. I comandi utilizzati per quest'ultimo punto sono i seguenti:

```
performance_file =
    fopen(strcat(Results_path,'/total_performance.tex'),'a');
fprintf(performance_file,'%s & %6.4f & %6.4f & %6.4f ',
        testcase_name,mean_error_tot,variance_tot,rms_tot)
fclose(performance_file)
```

Riassumendo, con l'automatizzazione del processo di simulazione l'utente, una volta configurate le variabili di sistema e i percorsi, deve semplicemente eseguire lo script bash il quale, in sequenza, esegue i processi di seguito elencati per ogni testcase:

1. creazione della cartella dei risultati;
2. caricamento della configurazione del testcase (script *settings*);
3. caricamento delle configurazioni dei veicoli;
4. esecuzione del testcase;
5. visualizzazione dei grafici e salvataggio in file JPG (script *Figures*);
6. calcolo delle performance (script *performances*).

Conclusi tali processi, l'utente visualizza nella cartella */Results* del test una sottocartella rinominata con la data e l'ora di partenza dello script bash

all'interno della quale si trovano tante sottocartelle quanti sono i testcase eseguiti più un file Latex, ottenuto dal file `total_performance_template.tex` contenente i risultati delle performance di ogni testcase eseguito. In ognuna di queste sottocartelle sono presenti il log file, le immagini JPG dei grafici, e il Matlab workspace. La scelta di rinominare le sottocartelle dei test eseguiti con la data e l'ora di partenza della simulazione del primo testcase permette all'utente finale di poter lanciare le stesse simulazioni quante volte desidera senza sovrapporre i risultati finali e quindi creare confusione con i risultati ottenuti.

5.4 Lo scenario di rete

Per la validazione dei tool di simulazione e per tutti i test svolti in questa tesi si è deciso di creare uno scenario che descriva il più possibile un ambiente marino nel quale sono presenti 5 AUV (Autonomus underwater vehichle), ovvero dispositivi sottomarini comandati a distanza, che trasmettono informazioni sotto forma di onde sonore. A questo scopo, tenendo conto delle assunzioni di progetto enunciate a inizio capitolo, i veicoli e lo scenario di rete sono stati progettati con le seguenti caratteristiche:

- i veicoli percorrono traiettorie con una velocità media di 5 m/s;
- la formazione mantiene i veicoli ad una distanza di 28 metri l'uno dall'altro;
- i pacchetti viaggiano ad una velocità di trasmissione di 1480 m/s che è esattamente la velocità di un onda sonora nell'acqua;

Questa scelta è dovuta al fatto che il controllo a distanza di veicoli AUV sono un'applicazione dei Networked Control System di particolare interesse ed è perfetto per riprodurre il problema di controllo di formazione. Per la rete simulata si è deciso di utilizzare il protocollo IEEE 802.15.4 che è nato per definire i livelli MAC e fisico per la comunicazione wirless low-power e low-rate utilizzata nelle reti di sensori.

Capitolo 6

Validazione dei tool

Prima di procedere con il lavoro di tesi occorre avere certezza sulla "bontà" degli strumenti utilizzati. Effettueremo in questo capitolo quindi la validazione dei tool descritti nel precedente capitolo con l'obiettivo di dimostrare che, a partire da un tool per simulazioni con ambiente Matlab puro, è possibile utilizzare tool che simulano ambienti più vicini al caso reale e che permettono di ottenere risultati più informativi. Lo scenario che utilizzeremo in questa fase è caratterizzato da 5 veicoli che partono in formazione perfetta. Il veicolo leader esegue in 70 secondi una traiettoria curva continua a velocità costante di 5 m/s. I veicoli, nel caso in cui sia presente la rete simulata, mantengono una potenza trasmissiva uguale per tutta la simulazione. Tale potenza è impostata in modo da coprire ogni altro veicolo presente nello scenario. Lo studio si è focalizzato principalmente sui seguenti aspetti:

- analisi dell'errore di formazione medio dei veicoli followers;
- analisi sulla variabilità dei valori di errore di formazione;
- analisi sui tempi dei tool per l'esecuzione delle simulazioni;
- analisi sul numero di pacchetti persi e corrotti nella rete.

Il testbench preparato per la validazione dei tool consiste in 10 simulazioni dello stesso scenario appena descritto. Negli ambienti di co-simulazione i Task fanno uso di generatori di numeri pseudo-casuali per generare i valori di Backoff del protocollo MAC. Per questo motivo è stato necessario inserire un seed diverso per ogni simulazione. Come seed sono stati scelti numeri interi da 0 sino al numero massimo di simulazioni eseguite (nel nostro caso 10) incrementando di 1 il valore ad ogni simulazione. In questo modo è possibile riprodurre le stesse sequenze di numeri casuali ogni qualvolta si desidera riavviare le 10 simulazioni dello stesso testcase.

Un problema fondamentale in questi test di validazione è simulare il delay di propagazione negli ambienti di simulazione che non utilizzano il canale delayed shared il quale, come già descritto in precedenza, modifica il delay di propagazione dei pacchetti dipendentemente alla distanza dei veicoli. Per risolvere tale problema si è dovuto analizzare i tempi di delay in questo scenario e progettare dei meccanismi per generare tali tempi. Nell'ambiente di simulazione Matlab puro si è aggiunto un nuovo modulo Simulink al modello che riproduce un ritardo costante determinato dalle dinamiche di una rete wireless a pacchetto:

- tempo di codifica/decodfica;
- tempo di CCA;
- tempo medio di Backoff;
- tempo medio di propagazione dei pacchetti.

Dopo un'attenta analisi si è deciso di introdurre un ritardo di propagazione dei segnali pari a 27 ms. Tale tempo è frutto della somma dei ritardi descritti precedentemente. In particolare si nota che la distanza media tra i veicoli è di 70 metri che coincide con un ritardo di 19 ms di propagazione dei pacchetti in un ambiente in cui si usano onde sonore nell'acqua. Per l'ambiente di co-simulazione con canale Shared invece si è dovuto simulare esclusivamente il tempo medio di propagazione dei pacchetti di 19 ms in quanto le altre dinamiche di rete sono già considerate in questo ambiente di simulazione grazie a SCNSL.

Analizzeremo ora le formule utilizzate per il calcolo dei risultati di ogni simulazione le quali ci forniscono i dati necessari per la validazione.

L'errore di formazione medio indica in media, per tutti i veicoli, la differenza tra la posizione reale ad ogni Ts e la posizione dove avrebbero idealmente dovuto trovarsi per essere in formazione. Si precisa che per tale misura si considera la posizione ideale come la posizione ideale rispetto al proprio leader e non al leader di posizione. Come già descritto in precedenza la formazione possiede un Leader che detta la traiettoria ma ogni veicolo ha un proprio leader dal quale riceve messaggi per calcolare le proprie velocità negli assi. La formula utilizzata è la seguente:

$$\frac{1}{T}\frac{1}{N}\sum_{N}\sum_{T}|p^{reale}(t,i) - p^{ideale}(t,i)|$$

dove T indica il tempo di simulazione, N indica il numero dei veicoli, $p^{reale}(t,i)$ la posizione reale dove si trova il veicolo i a tempo t e $p^{ideale}(t,i)$ la posizione ideale dove si dovrebbe trovare il veicolo i a tempo t.

La formula utilizzata per il calcolo della varianza σ è invece la seguente:

$$\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

dove N indica il numero totale di valori di errore dei 5 veicoli e x_i è l'i-esimo valore di errore sul totale. La varianza fornisce una misura di quanto siano vari i valori, ovvero di quanto si discostino dalla media.

Si è ritenuto inoltre necessario il calcolo dello scarto quadratico medio ad ogni simulazione prendendo sempre come popolazione di dati i valori dell'errore di formazione:

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i^2)}$$

Questi indici finora descritti si riferiscono, come già specificato, all'errore medio di formazione.

Si è deciso inoltre di calcolare i tempi di esecuzione di ogni tool tramite l'utilizzo del seguente comando bash:

`time "-f & %Es & %Us & %Ss"`

questo comando misura il tempo di esecuzione di un processo nel formato *System time*, *User Time* e *Real Time*.

Tutti i test effettuati sono stati impostati per eseguire 10 simulazioni di 70 secondi affinché la media dell'errore di formazione rientrasse all'interno dell'intervallo di confidenza al 95% per media sconosciuta e deviazione standard sconosciuta (nel senso che non è possibile avere un valore prefissato indipendente dalla simulazione). Il calcolo dell'intervallo di confidenza avviene grazie allo script bash del test che carica uno script Matlab dedicato esclusivamente al calcolo della media dell'errore di formazione totale e dell'intervallo di confidenza. In questo caso con media dell'errore di formazione totale indichiamo la media dell'errore di formazione di tutti i veicoli di tutte le 10 simulazioni eseguite. Si riporta di seguito il codice dello script in questione contenente i relativi comandi:

```
path = '/home/giacomo/scnsl/tests/test_validation_tools/Results';
load(strcat(path,'/errorVectors.mat'),'errorVector');

% Calculate confidence interval
[h,p,ci] =  ttest(errorVector);
mean_error = mean(errorVector);

% Print result in total_performance.tex file
performance_file = fopen(strcat(path,'/total_performance.tex'),'a');
fprintf(performance_file,'Media totale d''errore = %6.4f \n',mean_error);
fprintf(performance_file,'Intervallo di confidenza al 95\% =
[%6.4f ',ci(1));
fprintf(performance_file,'%6.4f] \n',ci(2));
```

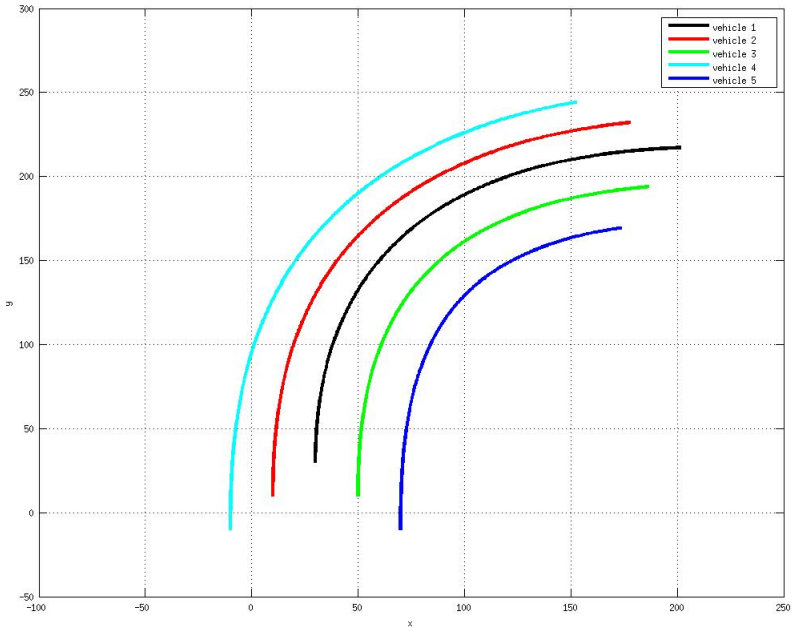


Figura 6.1: Traiettorie eseguite dai veicoli nella simulazione n.1 in ambiente Matlab puro.

```
fclose(performance_file);

% Quit Matlab
quit
```

La funzione `ttest(x)` esegue un test di ipotesi su un vettore random di valori x da una distribuzione normale con varianza non nota. Tale funzione permette inoltre di calcolare l'intervallo di confidenza al 95%. Nel codice mostrato precedentemente tale intervallo è indicato con *ci*. Il risultato del test ritorna un valore h. h = 1 indica che l'ipotesi è stata accettata mentre h = 0 indica che l'ipotesi del test è stata rifiutata.

6.1 Validazione dell'ambiente con Matlab puro

In questa sezione vengono mostrati e commentati i risultati delle 10 simulazioni eseguite per la validazione dell'ambiente con Matlab puro. Viene inoltre mostrata un'immagine delle traiettorie dei veicoli in Figura 6.1.

Analizzando i risultati ottenuti e visualizzati in Tabella 6.1 si nota che l'errore medio, la varianza e lo scarto quadratico medio sono totalmente identici in ogni simulazione. Questa situazione fa notare che non è possibile con Matlab puro riprodurre più simulazioni e ottenere risultati differenti di-

	Errore medio	Varianza	SQM
caseMaxPowerInit_Matlab	12 m	44.8169	13.7445
caseMaxPowerInit_Matlab	12 m	44.8169	13.7445
caseMaxPowerInit_Matlab	12 m	44.8169	13.7445
caseMaxPowerInit_Matlab	12 m	44.8169	13.7445
caseMaxPowerInit_Matlab	12 m	44.8169	13.7445
caseMaxPowerInit_Matlab	12 m	44.8169	13.7445
caseMaxPowerInit_Matlab	12 m	44.8169	13.7445
caseMaxPowerInit_Matlab	12 m	44.8169	13.7445
caseMaxPowerInit_Matlab	12 m	44.8169	13.7445
caseMaxPowerInit_Matlab	12 m	44.8169	13.7445

Tabella 6.1: Risultati dei valori di errore medio, varianza e scarto quadratico medio in ambiente Matlab puro.

versamente della realtà nella quale se si necessita di riprodurre esattamente lo stesso scenario si ottengono sicuramente risultati ogni volta diversi causati principalmente dalla presenza della rete. Il vantaggio di utilizzare questo tipo di ambiente è la velocità del tool di simulazione, come si nota in Tabella 6.2, che si aggira intorno ai 22 secondi. La media totale dell'errore risultante è 12 metri con un intervallo di confidenza al 95% di [11.8m 12.1m]. Un intervallo di confidenza cosi ristretto è il risultato di una bassa variabilità dei valori dell'errore di formazione. Questi risultati anche se a prima vista risultano molto buoni (la media dell'errore è molto bassa così come la variabilità dei dati) non sono tuttavia precisi in quanto, come descritto a inizio capitolo, la rete sottostante è simulata tramite un semplice modulo Matlab che crea un delay costante di trasmissione e non tiene conto delle dinamiche di rete.

6.2 Validazione dell'ambiente con canale Shared

In questa sezione vengono mostrati e commentati i risultati delle 10 simulazioni eseguite per la validazione dell'ambiente di co-simulazione che utilizza il canale shared.

Grazie alla tecnica di co-simulazione si può notare come la rete incide notevolmente sulle simulazioni. Tale conclusione è dedotta dal fatto che i risultati ottenuti in un ambiente con Matlab puro sono diversi rispetto ai risultati ottenuti con l'ambiente di co-simulazione appena validato utilizzando il canale Shared. In questo ultimo caso infatti, grazie all'introduzione della rete SCNSL, si introducono dinamiche di rete che rendono più precise e informative le simulazioni. Si nota dalla Tebella 6.3 come l'introduzione del protocollo di rete IEEE 802.15.4 alteri in maniera notevole il risultato

	Real time	User time	System time
caseMaxPowerInit_Matlab	0:22.86s	21.95s	1.52s
caseMaxPowerInit_Matlab	0:23.16s	21.89s	1.87s
caseMaxPowerInit_Matlab	0:23.11s	22.18s	1.56s
caseMaxPowerInit_Matlab	0:23.17s	22.08s	1.72s
caseMaxPowerInit_Matlab	0:23.17s	22.28s	1.56s
caseMaxPowerInit_Matlab	0:24.24s	23.30s	1.80s
caseMaxPowerInit_Matlab	0:26.56s	25.77s	1.90s
caseMaxPowerInit_Matlab	0:25.18s	24.42s	1.84s
caseMaxPowerInit_Matlab	0:24.29s	23.36s	1.80s
caseMaxPowerInit_Matlab	0:23.57s	22.85s	1.71s

Tabella 6.2: Risultati dei tempi di simulazione in formato Real time, User time e System Time in ambiente Matlab puro.

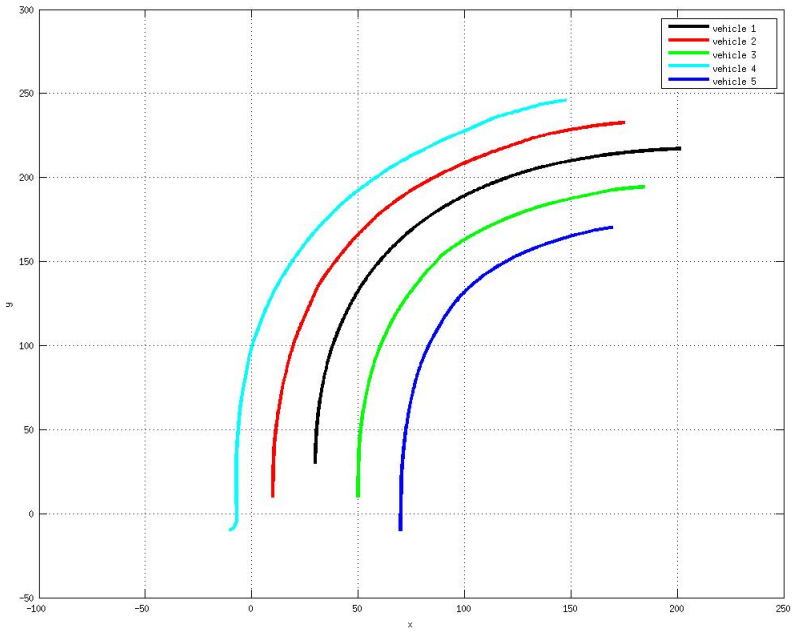


Figura 6.2: Traiettorie eseguite dai veicoli nella simulazione n.1 in ambiente di co-simulazione con canale Shared.

	Errore medio	Varianza	SQM
caseMaxPowerInit_SCNSL	5.5 m	4.5658	5.9177
caseMaxPowerInit_SCNSL	6.8 m	23.4481	8.4021
caseMaxPowerInit_SCNSL	5.8 m	9.4094	6.6071
caseMaxPowerInit_SCNSL	5.3 m	4.9019	5.7516
caseMaxPowerInit_SCNSL	5.5 m	6.1874	6.1070
caseMaxPowerInit_SCNSL	5.8 m	7.1796	6.4299
caseMaxPowerInit_SCNSL	5.7 m	7.3967	6.4028
caseMaxPowerInit_SCNSL	5.0 m	3.0037	5.3141
caseMaxPowerInit_SCNSL	5.4 m	3.6370	5.7384
caseMaxPowerInit_SCNSL	5.6 m	11.2812	6.5671

Tabella 6.3: Risultati dei valori di errore medio, varianza e scarto quadratico medio in ambiente co-simulazione con canale Shared.

	Real time	User time	System time
caseMaxPowerInit_SCNSL	24:28.54s	1412.80s	35.18s
caseMaxPowerInit_SCNSL	25:35.11s	1475.06s	35.81s
caseMaxPowerInit_SCNSL	25:19.76s	1463.04s	34.73s
caseMaxPowerInit_SCNSL	25:34.73s	1469.79s	35.21s
caseMaxPowerInit_SCNSL	25:21.68s	1463.01s	34.57s
caseMaxPowerInit_SCNSL	26:00.61s	1475.80s	52.67s
caseMaxPowerInit_SCNSL	25:04.67s	1445.61s	35.54s
caseMaxPowerInit_SCNSL	24:37.18s	1419.23s	33.55s
caseMaxPowerInit_SCNSL	24:09.32s	1394.48s	33.18s
caseMaxPowerInit_SCNSL	24:11.23s	1394.78s	34.71s

Tabella 6.4: Risultati dei tempi di simulazione in formato Real time, User time e System Time in ambiente co-simulazione con canale Shared.

	Collisioni	Corruzioni	MaxBackoff
caseMaxPowerInit_SCNSL	16568	0	0
caseMaxPowerInit_SCNSL	15970	0	0
caseMaxPowerInit_SCNSL	15887	0	0
caseMaxPowerInit_SCNSL	16276	0	0
caseMaxPowerInit_SCNSL	16459	0	0
caseMaxPowerInit_SCNSL	16112	0	0
caseMaxPowerInit_SCNSL	15952	0	0
caseMaxPowerInit_SCNSL	16114	0	0
caseMaxPowerInit_SCNSL	16665	0	0
caseMaxPowerInit_SCNSL	15843	0	0

Tabella 6.5: Risultati del numero di collisioni di pacchetti, corruzione di pacchetti e numero massimo di Backoff raggiunti in ambiente co-simulazione con canale Shared.

delle simulazioni a causa dell'utilizzo del tempo di Backoff e cioè di un timer di durata casuale che modifica i tempi di invio dei pacchetti. La media dell'errore di formazione infatti non è identica in ogni simulazione a differenza dell'ambiente Matlab puro il quale, con l'assenza della rete, ritorna sempre lo stesso risultato. Qui infatti la media di errore di formazione totale è 5.6 metri con un intervallo di confidenza al 95% = [5.6m 5.7m].

Una notevole caratteristica introdotta nel tool di co-simulazione con canale Shared grazie a SCNSL è la possibilità di rilevare il numero di collisioni dei pacchetti, il numero dei pacchetti corrotti e il numero di volte che i veicoli raggiungono il tempo massimo di Backoff.

Lo svantaggio principale nell'utilizzo di questo tool è il tempo di simulazione totale che aumenta in maniera drastica a causa della complessa elaborazione dati di SCNSL che si è sostituita a un semplice modulo presente nell'ambiente Matlab puro. I tempi calcolati in questo test di validazione sono presenti in Tabella 6.4. Un altro svantaggio è l'impossiblità di riprodurre fedelmente scenari wireless a causa del canale Shared che prevede esclusivamente delay di trasmissione dei pacchetti costanti. Questa lacuna è colmata grazie all'ambiente di co-simulazione che utilizza il canale Delayed Shared creato grazie al lavoro di questa tesi e che sarà validato e analizzato nella successiva sezione.

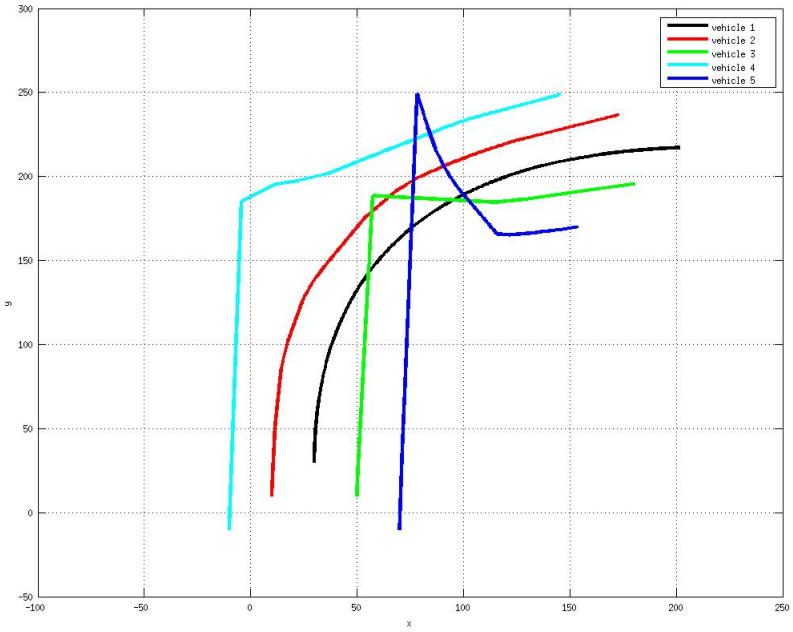


Figura 6.3: Traiettorie eseguite dai veicoli nella simulazione n.1 in ambiente di co-simulazione con canale Delayed Shared.

6.3 Validazione dell'ambiente con canale Delayed Shared.

Verranno ora mostrati i risultati della validazione dell'ambiente di co-simulazione che utilizza il canale Delayed Shared.

Questo strumento per la simulazione ritorna come risultato una media totale degli errori di 18.1 metri e un intervallo di confidenza al 95% = [17.8m 18.4m]. Questo valore sta ad indicare che li delay di trasmissione in una simulazione di una rete wireless incide notevolmente sulle performance confrontandolo con il risultato del tool precedente. Non è possibile quindi utilizzare un canale Shared per simulare reti wireless in maniera precisa e realistica ma ciò è possibile grazie al canale Delayed Shared che calcola per ogni pacchetto un delay di trasmissione basato sulla distanza dei veicoli e sulla velocità di propagazione del segnale nel canale trasmissivo.

Inoltre il numero di collisioni risulta essere diminuito notevolmente come si nota in Tabella 6.8. Questo perchè nel caso si utilizzi un delay costante per la trasmissione di dati è molto più probabile che due o più pacchetti siano decodificati a destinatario nello stesso istante e quindi risultino collisi. Questo caso è comune in uno scenario wired ma non in uno scenario wireless

	Errore medio	Varianza	SQM
caseMaxPowerInit_SCNSL	22.5 m	401.4930	30.1257
caseMaxPowerInit_SCNSL	23.6 m	480.1988	32.2117
caseMaxPowerInit_SCNSL	15.2 m	287.7385	22.7899
caseMaxPowerInit_SCNSL	16.1 m	182.7452	21.0663
caseMaxPowerInit_SCNSL	17.7 m	327.3423	25.3418
caseMaxPowerInit_SCNSL	15.1 m	234.5940	21.5086
caseMaxPowerInit_SCNSL	16.5 m	295.5806	23.8268
caseMaxPowerInit_SCNSL	13.7 m	248.8377	20.9304
caseMaxPowerInit_SCNSL	19.4 m	299.4153	26.0345
caseMaxPowerInit_SCNSL	21.7 m	288.9876	27.5880

Tabella 6.6: Risultati dei valori di errore medio, varianza e scarto quadratico medio in ambiente co-simulazione con canale Delayed Shared.

	Real time	User time	System time
caseMaxPowerInit_SCNSL	28:10.85s	1516.32s	37.32s
caseMaxPowerInit_SCNSL	28:22.00s	1589.36s	36.78s
caseMaxPowerInit_SCNSL	28:11.60s	1588.55s	39.24s
caseMaxPowerInit_SCNSL	29:11.38s	1645.69s	39.41s
caseMaxPowerInit_SCNSL	29:19.84s	1649.93s	37.91s
caseMaxPowerInit_SCNSL	29:28.28s	1665.68s	39.17s
caseMaxPowerInit_SCNSL	29:14.64s	1652.97s	39.41s
caseMaxPowerInit_SCNSL	29:03.02s	1637.33s	38.89s
caseMaxPowerInit_SCNSL	29:35.35s	1641.80s	57.47s
caseMaxPowerInit_SCNSL	28:58.60s	1626.18s	37.86s

Tabella 6.7: Risultati dei tempi di simulazione in formato Real time, User time e System Time in ambiente co-simulazione con canale Shared.

	Collisioni	Corruzioni	MaxBackoff
caseMaxPowerInit_SCNSL	4503	0	0
caseMaxPowerInit_SCNSL	4729	0	0
caseMaxPowerInit_SCNSL	6125	0	0
caseMaxPowerInit_SCNSL	5462	0	0
caseMaxPowerInit_SCNSL	5492	0	0
caseMaxPowerInit_SCNSL	6371	0	0
caseMaxPowerInit_SCNSL	6050	0	0
caseMaxPowerInit_SCNSL	6039	0	0
caseMaxPowerInit_SCNSL	5106	0	0
caseMaxPowerInit_SCNSL	5191	0	0

Tabella 6.8: Risultati del numero di collisioni di pacchetti, corruzione di pacchetti e numero massimo di Backoff raggiunti in ambiente co-simulazione con canale Shared.

nel quale il delay di trasmissione è quasi sempre diverso salvo il caso in cui due o più dispositivi trasmettitori si trovano alla stessa identica distanza dal dispositivo ricevitore.

Di conseguenza si è dimostrato che se si desidera simulare scenari come quello descritto in sezione 5.4 l'ambiente di co-simulazione con canale Delayed Shared risulta essere migliore in quanto più preciso e realistico per i motivi appena citati. Utilizzeremo dunque questo ambiente di simulazione per testare la validità dell'algoritmo di regolazione della potenza trasmissiva descritto nel prossimo capitolo.

Capitolo 7

Algoritmo di adattamento della potenza trasmissiva

Questo capitolo è interamente dedicato all'algoritmo prodotto dal lavoro di questa tesi. In sezione 7.1 si descrivono le motivazioni che hanno portato all'idea dell'algoritmo di adattamento della potenza trasmissiva e gli obiettivi che si desiderano raggiungere con tale algoritmo. La sezione 7.2 è dedicata completamente alla descrizione del codice dell'algoritmo ed infine la sezione 7.3 visualizza i risultati del test di validazione dell'algoritmo e contiene l'analisi effettuata su tali risultati assieme alle dovute conclusioni.

7.1 Obiettivi

Dai test eseguiti durante la validazione dei tool è risultato che la presenza di una rete wireless per lo scambio di messaggi tra i veicoli provoca evidenti disturbi nel raggiungimento/mantenimento della formazione voluta in quanto la media totale dell'errore nell'ambiente di co-simulazione con canale Delayed Shared risulta molto più elevata rispetto agli altri ambienti di simulazione. La presenza della rete, inoltre, può provocare collisioni di pacchetti oltre ad un ritardo dovuto alla codifica/propagazione/decodifica di un pacchetto. Il destinatario di tali pacchetti viene quindi influenzato in modo determinante.

Si può immaginare come il range di trasmissione in una comunicazione wireless influisce molto sull'aspetto riguardante le collisioni. Si provi ad immaginare due coppie di nodi che cercano di comunicare come in Figura 7.1. La situazione ottimale si ottiene nel caso in cui i due range di trasmissione non vadano ad intersecarsi, come mostrato in Figura 7.2. Questa situazione annulla la probabilità di collisioni dei pacchetti scambiati tra le due coppie. Nel caso in cui i due range si intersechino si crea la situazione dell'hidden node visibile in Figura 7.3. In questo caso la possibilità di collisioni è elevata e ciò influisce pesantemente sulle prestazioni del controllore Matlab.

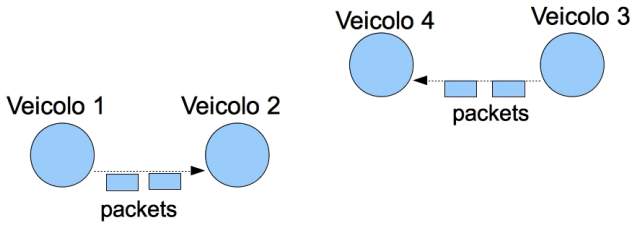


Figura 7.1: Coppie di veicoli che si spediscono messaggi.

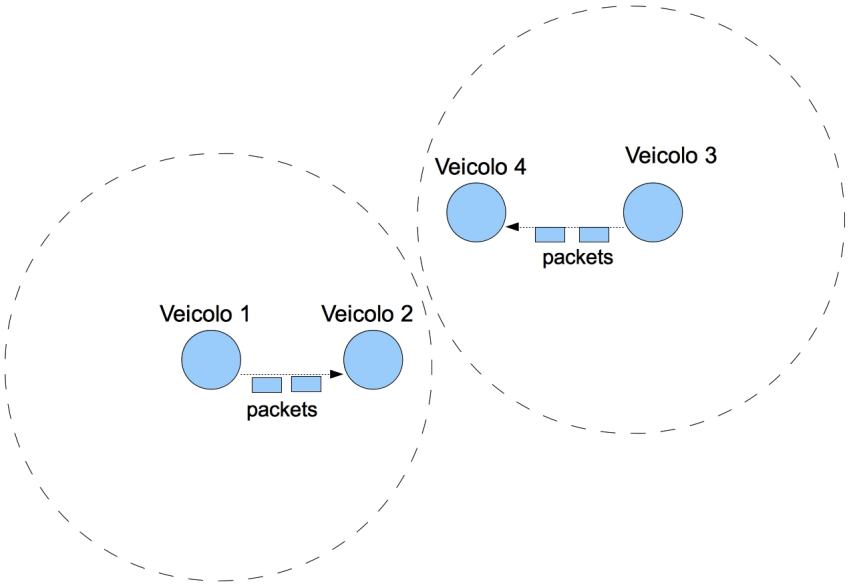


Figura 7.2: Coppie di veicoli che si spediscono messaggi con range separati.

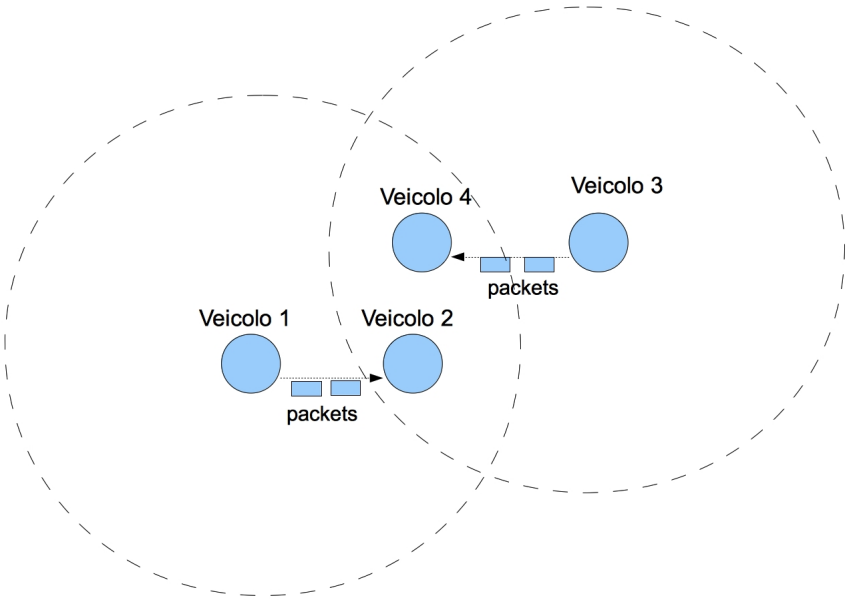


Figura 7.3: Coppie di veicoli che si spediscono messaggi. I range di trasmissione sono intersecati e causano il problema dell'hidden node.

Per risolvere tale problema si può dapprima pensare di progettare i dispositivi per il controllo di formazione in modo che mantengano una potenza trasmissiva ad un livello necessario a comunicare con i veicoli *Neighbors*. In questo modo i range trasmissivi sono ridotti al minimo così come il consumo di potenza e la possibilità di collisione dei pacchetti.

Poniamo dunque il range trasmissivo di ogni veicolo ad un livello di potenza che rappresenti la configurazione appena descritta e che sia costante per tutta la simulazione. I veicoli nello scenario utilizzato partono in formazione perfetta a distanza di 28,28 metri dal proprio *Leader*; di conseguenza il range trasmissivo viene impostato ad un minimo che permette di contenere questi veicoli.

Per complicare lo scenario e renderlo di maggior interesse per l'analisi dei risultati, si è inserito un disturbo ad uno dei veicoli in modo che al secondo 35 subisca uno spostamento laterale di 5 m/s per la durata di 3 secondi. Il motivo di questo disturbo è studiare il comportamento del veicolo disturbato nella configurazione appena descritta. Si può immaginare questo fenomeno come un ostacolo alla formazione che nel caso reale descritto nella sezione 5.4 si traduce in una forte corrente marina inaspettata.

Di seguito si riportano i risultati delle 10 simulazioni effettuate sullo scenario utilizzando l'ambiente di co-simulazione con canale Delayed Shared. Si riporta inoltre un'immagine delle traiettorie dei veicoli.

Come si può notare dalla Tabella 7.1 i valori di errore di formazione

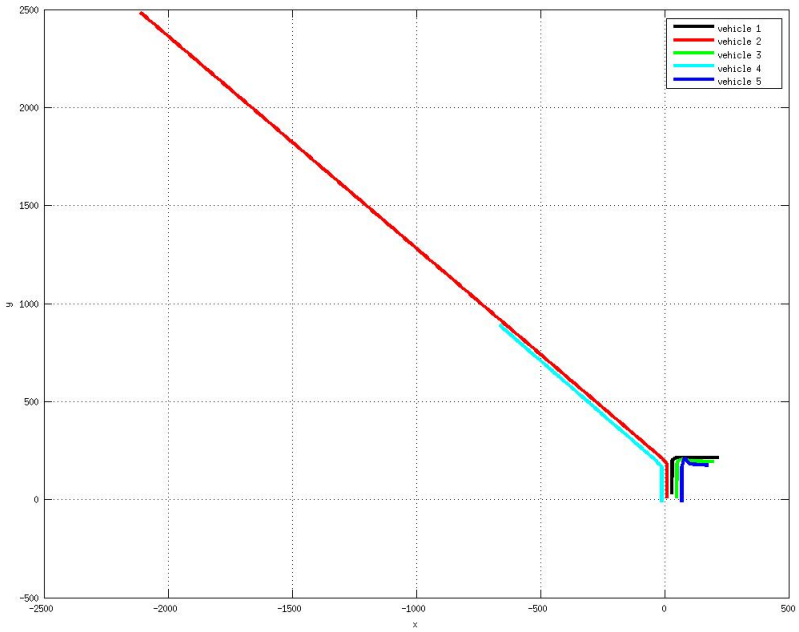


Figura 7.4: Traiettorie eseguite dai veicoli nella simulazione n.1 con scenario con potenza trasmissiva minima.

	Errore medio	Varianza	SQM
caseCurveMinPowerInit	303.9 m	473184.6562	751.8291
caseCurveMinPowerInit	304.2 m	473724.1626	752.2894
caseCurveMinPowerInit	305.2 m	471065.9661	750.9524
caseCurveMinPowerInit	304.0 m	473403.8849	751.9955
caseCurveMinPowerInit	303.7 m	468835.7503	748.8365
caseCurveMinPowerInit	303.7 m	472983.5115	751.6269
caseCurveMinPowerInit	304.3 m	473290.8860	752.0425
caseCurveMinPowerInit	301.0 m	464357.3042	744.7503
caseCurveMinPowerInit	303.7 m	471333.6505	750.5326
caseCurveMinPowerInit	304.4 m	473033.2763	751.9352

Tabella 7.1: Risultati dei valori di errore medio, varianza e scarto quadratico medio nello scenario con veicoli con potenza trasmissiva minima.

	Collisioni	Corruzioni	MaxBackoff
caseCurveMinPowerInit	1611	0	0
caseCurveMinPowerInit	1590	0	0
caseCurveMinPowerInit	1478	0	0
caseCurveMinPowerInit	1417	0	0
caseCurveMinPowerInit	1623	0	0
caseCurveMinPowerInit	1559	0	0
caseCurveMinPowerInit	1670	0	0
caseCurveMinPowerInit	1553	0	0
caseCurveMinPowerInit	1468	0	0
caseCurveMinPowerInit	1415	0	0

Tabella 7.2: Risultati del numero di collisioni di pacchetti, corruzione di pacchetti e numero massimo di Backoff raggiunti nello scenario con veicoli con potenza trasmissiva minima.

medio, varianza e scarto quadratico medio sono elevatissimi. L'errore di formazione totale medio inoltre è di 303.8 metri con un intervallo di confidenza al 95% = [292.6m 315.0m]. Il motivo è molto semplice e lo si deduce dalla figura 7.4. La scelta di impostare la potenza dei range trasmissivi ad un valore costante ma basato sulla distanza di partenza dei veicoli non risolve le problematiche causate da improvvisi ostacoli che deviano le traiettorie. Infatti il secondo veicolo, nel momento in cui il disturbo causato dalla corrente marina improvvisa smette di deviare il suo percorso, non risiede all'interno di nessun range trasmissivo quindi smette di ricevere pacchetti. Di conseguenza il controllore non è più in grado di calcolare il vettore di velocità che permette di riportare in formazione il veicolo il quale viaggerà costantemente verso la traiettoria creata dal disturbo.

Un'alternativa per risolvere questo problema e permettere quindi di non perdere nessun veicolo in caso di ostacoli è porre la potenza trasmissiva al massimo in ogni dispositivo. Si è deciso quindi di ripetere le 10 simulazioni precedenti con una configurazione nella quale la potenza trasmissiva è impostata a livelli altissimi per permettere anche a veicoli lontani centinaia di metri di poter essere raggiunti e di conseguenza riportarsi in formazione. In Tabella 7.3 e 7.4 sono mostrati i risultati ottenuti.

Il problema della perdita dei veicoli è stato risolto in quanto si nota dalla figura 7.5 che il veicolo numero 2, dopo aver subito la spinta dalla corrente marina improvvisa per 3 secondi, continua a ricevere pacchetti e si riporta lentamente in formazione. La media totale dell'errore di formazione si è abbassata a 42.5 metri con un Intervallo di confidenza al 95% = [40.8m 44.1m]. Questa configurazione risulta però svantaggiosa in termini di numero di collisioni dei pacchetti i quali, come si deduce dai valori in Tabella 7.4,

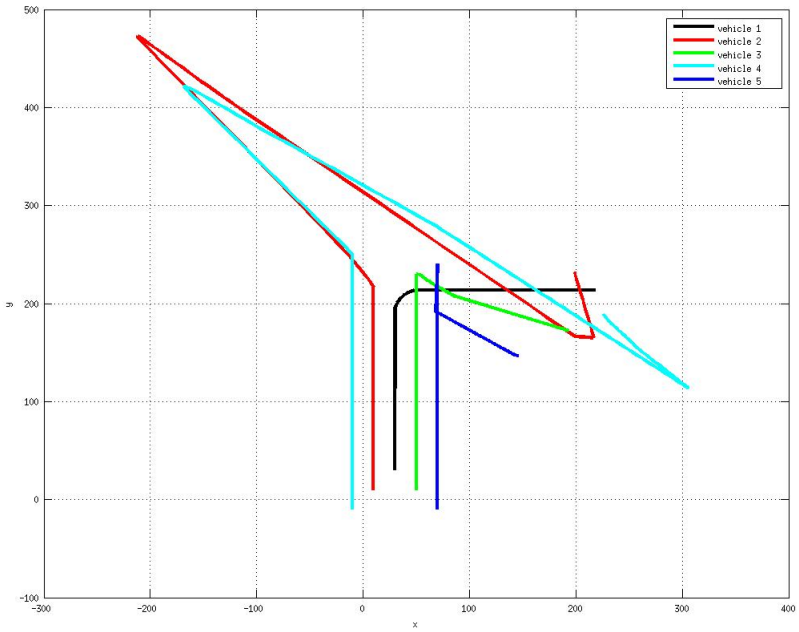


Figura 7.5: Traiettorie eseguite dai veicoli nella simulazione n.1 con scenario con potenza trasmissiva massima.

	Errore medio	Varianza	SQM
caseCurveMaxPowerInit	33.3 m	2388.5883	59.1540
caseCurveMaxPowerInit	33.3 m	2149.5112	57.0748
caseCurveMaxPowerInit	28.3 m	2546.7122	57.8898
caseCurveMaxPowerInit	38.2 m	3109.0252	67.5785
caseCurveMaxPowerInit	23.9 m	753.8759	36.4272
caseCurveMaxPowerInit	26.0 m	736.2162	37.6069
caseCurveMaxPowerInit	29.7 m	2395.1674	57.2629
caseCurveMaxPowerInit	31.5 m	2506.3466	59.1518
caseCurveMaxPowerInit	38.6 m	3461.4888	70.3736
caseCurveMaxPowerInit	34.7 m	2316.2767	59.3581

Tabella 7.3: Risultati dei valori di errore medio, varianza e scarto quadratico medio nello scenario con veicoli con potenza trasmissiva massima.

	Collisioni	Corruzioni	MaxBackoff
caseCurveMaxPowerInit	3103	0	0
caseCurveMaxPowerInit	3080	0	0
caseCurveMaxPowerInit	3393	0	0
caseCurveMaxPowerInit	2912	0	0
caseCurveMaxPowerInit	3152	0	0
caseCurveMaxPowerInit	1942	0	0
caseCurveMaxPowerInit	3041	0	0
caseCurveMaxPowerInit	3024	0	0
caseCurveMaxPowerInit	3143	0	0
caseCurveMaxPowerInit	2947	0	0

Tabella 7.4: Risultati del numero di collisioni di pacchetti, corruzione di pacchetti e numero massimo di Backoff raggiunti nello scenario con veicoli con potenza trasmissiva massima.

sono praticamente raddoppiati. Il consumo di potenza per la trasmissione di pacchetti in questa situazione è posta al massimo livello causando un consumo energetico elevato e una ridotta durata della batteria dei dispositivi. L'elevato numero di collisioni è causato dall'intersezione di tutti i range trasmissivi i quali, come si può immaginare, creano una situazione nella quale i dispositivi ricevono numerosi pacchetti da veicoli non *Neighbors*, pacchetti che successivamente verrebbero scartati e quindi inutilizzati.

Dalle precedenti analisi è nata dunque l'idea di sviluppare un algoritmo in grado di modificare la potenza trasmissiva di ogni veicolo durante la simulazione, in modo da creare il minor disturbo possibile verso gli altri componenti della formazione e in grado di mantenere una copertura di comunicazione adeguata per l'intera simulazione. In questo modo il consumo energetico sarebbe ridotto al minimo così come il numero delle collisioni dei pacchetti. Nella successiva sezione verrà descritto tale algoritmo di regolazione della potenza trasmissiva.

7.2 Descrizione dell'algoritmo

Per il funzionamento dell'algoritmo si è diviso il range di trasmissione di ogni veicolo in tre regioni di interesse. Tali regioni vanno sotto il nome di:

- *Near Region:* se il veicolo inseguitore si trova in questa regione, il veicolo leader può diminuire la sua potenza trasmissiva;
- *Safe Region:* se il veicolo inseguitore si trova in questa regione, il veicolo leader non deve cambiare la sua potenza trasmissiva;

- *Unsafe Region:* se il veicolo inseguitore si trova in questa regione, il veicolo leader deve aumentare la sua potenza trasmissiva.

La Figura 7.6 ci illustra chiaramente la suddivisione del range trasmissivo nelle regioni sopraccitate. Ogni regione ha una dimensione dipendente dalla potenza trasmissiva con la quale un determinato veicolo sta trasmettendo in un determinato momento. In particolare, calcolata la potenza trasmissiva, la *Near Region* copre un'ampiezza direttamente proporzionale all'80% della potenza trasmissiva, mentre *Safe Region* e *Unsafe Region* coprono entrambe un'ampiezza proporzionale al 10% della potenza trasmissiva. La formula per calcolare l'ampiezza delle regioni è ricavata dalla formula di decadimento del segnale wireless utilizzata nella rete simulata da SCNSL. Questa divisione è pensata per ovviare ad alcuni problemi nei quali potrebbero incorrere i veicoli durante il loro cammino.

La regione chiamata *Unsafe Region* è una zona di pericolo per un veicolo che sta seguendo il suo leader. Esso infatti si trova in una zona limite del range trasmissivo del veicolo che deve "ascoltare" e quindi rischia da un momento all'altro di uscire completamente dal range e non ricevere più messaggi. Questo è un rischio troppo grande poiché l'inseguitore, non ricevendo più messaggi dal leader, non ha modo di modificare la proprià velocità e direzione perdendo definitivamente la possibilità di rimanere in formazione. Le cause che possono portare un veicolo in questa regione sono molteplici: pensiamo ad esempio alla necessità di evitare un ostacolo che nel caso fosse colpito porterebbe alla distruzione del veicolo e comprometterebbe definitivamente la formazione. L'idea è quella che un veicolo rimanga in questa zona il meno possibile, onde evitare tragiche conseguenze come ad esempio la perdita del veicolo accaduta nel test in sezione 7.1. L'algoritmo deve quindi intervenire nel minor tempo possibile, cambiando la potenza trasmissiva dei dispositivi in questione per diminuire le probabilità di uscita dal range di trasmissione.

La zona più ampia a livello di dimensioni è la *Near Region.* In questa regione non ci sono pericoli di uscire dal range di trasmissione, anzi, il trasmettitore sta sprecando troppa energia poichè la distanza tra i veicoli potrebbe essere colmata con una potenza trasmissiva minore. Ecco che l'algoritmo, seguendo la strategia di ottimizzazione della potenza trasmissiva, interviene e diminuisce la potenza trasmissiva delle entità in gioco in modo da ridurre sia il consumo di energia, sia il disturbo verso altri veicoli della formazione.

La zona più sicura per un veicolo è la *Safe Region.* Essa è una zona di mezzo, nella quale i veicoli sono "al sicuro". In caso di imminente cambio di traiettoria per evitare un ostacolo, per esempio, essi avrebbero una minima percentuale di possibilità di uscita dal range trasmissivo. Infatti, prima di uscire completamente dovrebbero attraversare l'intera *Unsafe Region.* È auspicabile che i veicoli si trovino in questa zona per il maggior tempo possi-

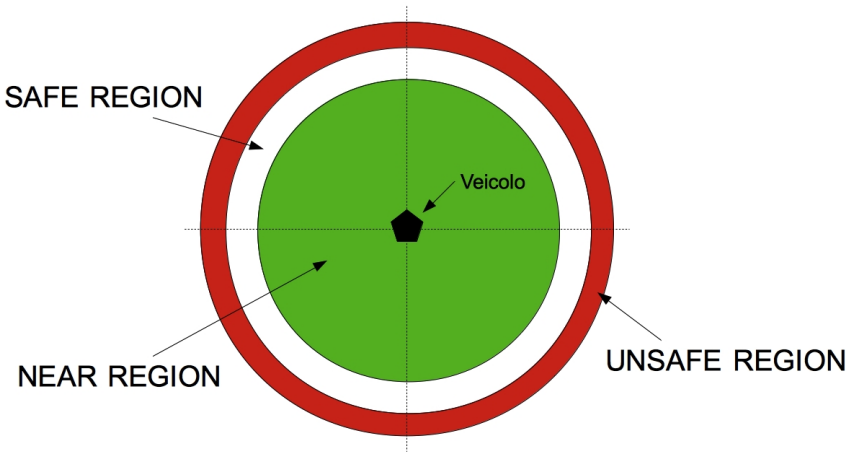


Figura 7.6: Suddivisione del range trasmissivo in regioni.

bile in quanto il pericolo di uscire dal range di trasmissione è minimizzato ed inoltre la potenza trasmissiva utilizzata è perfetta per la situazione corrente. In *Safe Region* siamo quindi nella posizione perfetta.

Andiamo ora ad analizzare in dettaglio il funzionamento dell'algoritmo basandoci sul codice Matlab.

E' noto che ogni veicolo ha una lista dove mantiene gli identificatori dei suoi "vicini", cioè tutti i veicoli con cui il nodo interessato deve scambiare messaggi. Nel caso in cui un nodo abbia più di un "vicino" l'algoritmo deve memorizzare le informazioni di tali veicoli per utilizzarle successivamente per il calcolo della nuova potenza trasmissiva con cui deve trasmettere.

A questo scopo in fase di inizializzazione del controllore Matlab si creano e si inizializzano un numero di stati discreti proporzionale al numero di elementi contenuti nell'array *Neighbors*:

```
% Creazione stati discreti
sizes.NumDiscState = 3 + 2 * length( Neighbors );

% Inizializzazione stati discreti
x0_2 = [];
for i=1:length(Neighbors)
    x0_2 = [ x0_2; 0; t_power_init];
end
x0 = [ 0 0 t_power_init x0_2 ];
```

Si nota che se l'array *Neighbors* contenesse 1 solo elemento, gli stati creati e poi inizializzati sarebbero 5, mentre se *Neighbors* avesse dimensione 2, gli stati creati e poi inizializzati sarebbero 7, ecc... Il primo stato contiene la velocità che deve mantenere il veicolo nell'asse X; il secondo stato memorizza la velocità nell'asse Y e il terzo stato contiene valore della potenza trasmissiva. Le funzioni degli stati creati dinamicamente grazie alla dimensione dell'array *Neighbors* verranno spiegate nel proseguo di questa sezione.

Nel momento in cui arriva un pacchetto si passa a controllare se arriva da uno dei "vicini" contenuti nell'array *Neighbors*. Solo in questo caso infatti esso è un pacchetto di interesse per l'algoritmo. Tramite la funzione `find()` si carica nella variabile `flag` l'indice dell'array che corrisponde al veicolo che ha inviato il messaggio:

```
flag = find( (Neighbors-vehicle_id) == 0, 1 );
```

Il primo ciclo `if` controlla se la variabile `flag` contiene un valore:

```
if ~isempty(flag)
```

In caso positivo si entra nel ciclo e si calcolano come prima cosa tre grandezze:

- `max_distance_reachable`: la massima distanza in metri che il veicolo può coprire con la potenza trasmissiva attuale;
- `unsafe_region`: l'ampiezza in metri della *Unsafe Region*;
- `safe_region`: l'ampiezza in metri della *Safe Region*.

grazie ai comandi seguenti:

```
max_distance_reachable = ( x( 3 ) / Threshold )^( 1 / att_exponent );
unsafe_region = max_distance_reachable * bound;
safe_region = max_distance_reachable * bound;
```

A questo punto si salva in una variabile identificata da `j` il valore contenuto in `flag` e si calcolano i rispettivi valori per le variabili `t_index` e `p_index`:

```
j = flag;
t_index = 3 + ( ( j-1 ) * 2 ) + 1;
p_index = 3 + ( ( j-1 ) * 2 ) + 2;
```

Queste variabili servono per indicare in quale degli stati l'algoritmo deve salvare le informazioni. Gli indici presenti sono due in quanto, per ogni elemento in *Neighbors*, l'algoritmo deve memorizzare due stati: il primo tiene traccia del tempo dell'ultimo pacchetto ricevuto dal veicolo identificato dall'elemento presente in *Neighbors*; il secondo memorizza la potenza trasmissiva con cui il veicolo deve trasmettere in modo che tale veicolo sia all'interno della *Safe Region*. Di conseguenza per ogni elemento presente in *Neighbors* l'algoritmo riuscirà a tener traccia delle rispettive informazioni necessarie (tempo dell'ultimo pacchetto ricevuto e potenza trasmissiva).

Successivamente si memorizza nello stato identificato da `t_index` il tempo di arrivo del pacchetto attualmente in fase di elaborazione. Tramite un ciclo `for` si caricano in un array i tempi attualmente in memoria negli stati relativi al tempo di arrivo dei pacchetti:

```
x(t_index) = arrival_time;

array_time = [];

for i=1:length( Neighbors )
        array_time( i ) = x( end - ( i * 2 ) + 1 );
end
```

A questo punto si verifica tramite un costrutto `if` se l'elemento minore presente nell'array è più piccolo del tempo attuale meno una costante moltiplicata per `Ts`. Se ciò accade l'algoritmo imposta la potenza trasmissiva attuale con il valore della potenza trasmissiva iniziale, che è la massima possibile:

```
if ( min( array_time ) < ( actual_time - 10 * Ts )  )
        x( 3 ) = t_power_init;
```

Questo controllo sui tempi è necessario per evitare che un veicolo perda la formazione. La non ricezione di messaggi da parte dei veicoli è causata da due diversi fattori: errori di trasmissioni dovuti alla rete, oppure dall'uscita del veicolo dai range trasmissivi dei suoi "vicini". A lato Matlab non è possibile capire per quale dei due motivi precedenti un veicolo non riceve più pacchetti quindi l'algoritmo deve intervenire per cercare di risolvere tale situazione. L'implementazione prevede che se per un periodo di tempo definito da un multiplo di `Ts` un veicolo non riceve più pacchetti da uno dei veicoli nella lista *Neighbors*, esso trasmette con la massima potenza possibile, in modo da recuperare il veicolo perso nel caso sia uscito dal range trasmissivo.

Se il precedente controllo non va a buon fine si passa al ramo `else`, che è il cuore dell'algoritmo. In questa sezione si controlla in quale delle tre regioni si trova il veicolo che ha trasmesso il pacchetto. Nel caso in cui si trovi nella *Near Region* o nella *Unsafe Region* si calcola la nuova potenza trasmissiva con cui si deve trasmettere e si salva il valore nel relativo stato, altrimenti la potenza trasmissiva attuale rimane invariata . Di seguito sono riportate le righe di codice riferite a questa parte di algoritmo:

```
if (distance < ( max_distance_reachable - unsafe_region - safe_region))

        x( p_index ) = ( ( distance^( att_exponent ) ) * Threshold ) *
                    ( bound_power );

else if ( ( distance < max_distance_reachable ) &&
        ( distance > ( max_distance_reachable - unsafe_region ) ) )

        x( p_index ) = ( ( distance^( att_exponent ) ) * Threshold ) *
                    ( bound_power );

        else
        x( p_index ) = x( 3 );
end
```

Come ultima fase l'algoritmo, tramite un ciclo `for`, carica in un array i valori attualmente salvati negli stati relativi alla potenza trasmissiva e imposta come potenza trasmissiva attuale la massima tra quelle contenute negli stati:

```
rangeD = [];

for i=1:length(Neighbors)
        rangeD( i ) = x( end - ( i - 1 ) * 2 );
end

x( 3 ) = max( rangeD );
```

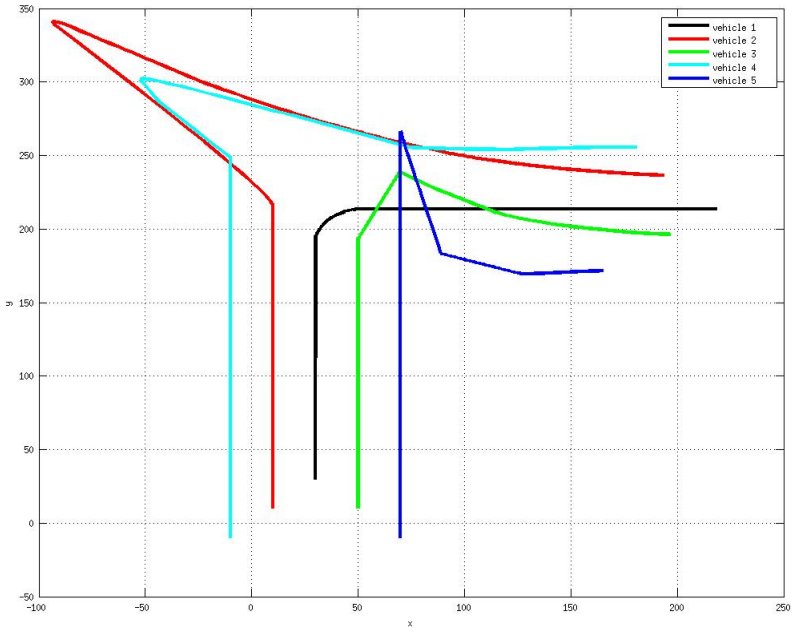


Figura 7.7: Traiettorie eseguite dai veicoli nella simulazione n.1 in ambiente di co-simulazione con canale Delayed Shared e algoritmo di regolazione della potenza trasmissiva.

Questo passaggio è necessario al corretto funzionamento dell'algoritmo, in quanto, data la possibilità di avere più di un vicino, l'algoritmo dovrà impostare la potenza trasmissiva in modo che il veicolo riesca a comunicare con tutti i suoi vicini compreso quello più lontano.

7.3 Validazione dell'algoritmo e analisi dei risultati

In questo capitolo viene testato il funzionamento dell'algoritmo di adattamento della potenza trasmissiva con lo stesso scenario utilizzato nella sezione precedente e ambiente di co-simulazione con canale Delayed Shared. Di seguito in Figura 7.7 è riportata un'immagine delle traiettorie dei 5 veicoli e in Tabella 7.5 e 7.6 i risultati ottenuti.

Dall'analisi dei risultati ottenuti si notano netti miglioramenti sulle performance del controllo di formazione. Si nota innanzitutto che l'errore di formazione medio è diminuito rispetto ai valori ottenuti nei test precendenti. Ora il valore è di 22.5 metri con con un intervallo di confidenza al 95% = [22.0m 22.9m]. Tali risultati sono maggiormente evidenti nella

	Errore medio	Varianza	SQM
caseCurveAlgorithm	27.0 m	750.6213	38.4799
caseCurveAlgorithm	36.3 m	1754.7176	55.4206
caseCurveAlgorithm	13.1 m	322.2366	22.2676
caseCurveAlgorithm	16.6 m	405.8393	26.1304
caseCurveAlgorithm	24.8 m	738.5313	36.7885
caseCurveAlgorithm	20.4 m	663.9727	32.8689
caseCurveAlgorithm	18.8 m	482.4016	28.9086
caseCurveAlgorithm	17.2 m	441.0913	27.1917
caseCurveAlgorithm	23.5 m	717.3377	35.6358
caseCurveAlgorithm	27.1 m	711.6992	38.0531

Tabella 7.5: Risultati dei valori di errore medio, varianza e scarto quadratico medio nello scenario con veicoli con algoritmo di regolazione della potenza trasmissiva.

	Collisioni	Corruzioni	MaxBackoff
caseCurveAlgorithm	1935	1	0
caseCurveAlgorithm	1109	0	0
caseCurveAlgorithm	2251	0	0
caseCurveAlgorithm	1837	0	0
caseCurveAlgorithm	1922	0	0
caseCurveAlgorithm	2216	0	0
caseCurveAlgorithm	2148	0	0
caseCurveAlgorithm	1971	0	0
caseCurveAlgorithm	1962	0	0
caseCurveAlgorithm	1757	0	0

Tabella 7.6: Risultati del numero di collisioni di pacchetti, corruzione di pacchetti e numero massimo di Backoff raggiunti nello scenario con veicoli con algoritmo di regolazione della potenza trasmissiva.

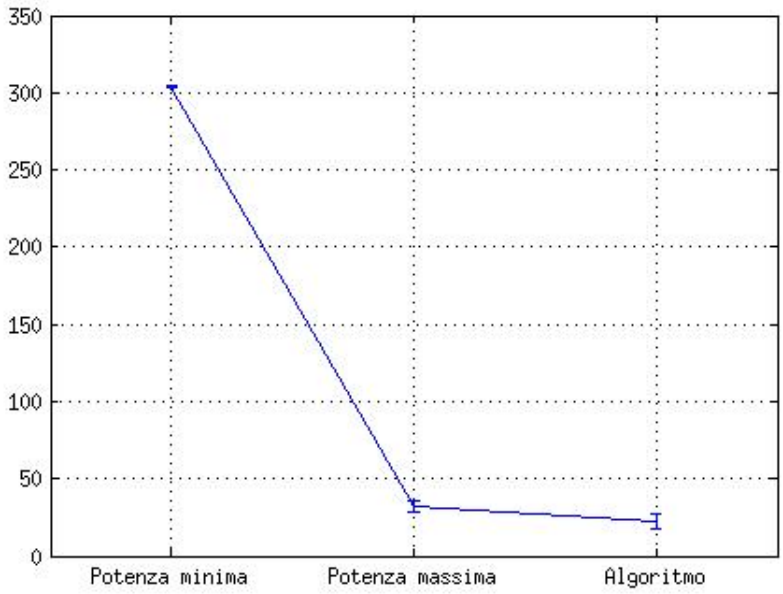


Figura 7.8: Da sinistra a destra gli intervalli di confidenza dei test MinPower MaxPower e Algorithm.

Figura 7.8 che mette a confronto i valori delle medie totali dell'errore di formazione e gli intervalli di confidenza calcolati. L'intervallo di confidenza relativo al test che utilizza l'algoritmo di adattamento della potenza trasmissiva risulta completamente disgiunto dagli intervalli di confidenza dei test precedenti e soprattutto dal test con potenza iniziale massima che fino ad ora sembrava essere il migliore dal punto di vista prestazionale. Quest'ultimo risultato è chiaramente raffigurato in Figura 7.9 che in pratica è uno zoom della Figura 7.8 sui test effettuati con potenza iniziale massima e con l'utilizzo dell'algoritmo. Possiamo dedurre quindi che l'utilizzo dell'algoritmo di adattamento della potenza trasmissiva per il controllo di formazione offre sempre performance migliori rispetto alle precedenti configurazioni con potenza trasmissiva iniziale minima e potenza trasmissiva iniziale massima.

Un altro risultato positivo ottenuto è la diminuzione netta del numero di collisioni di pacchetti. Per ogni testcase il numero di collisioni è dimezzato grazie all'adattamento della potenza trasmissiva durante la simulazione che abbatte la probabilità di intersezione dei range.

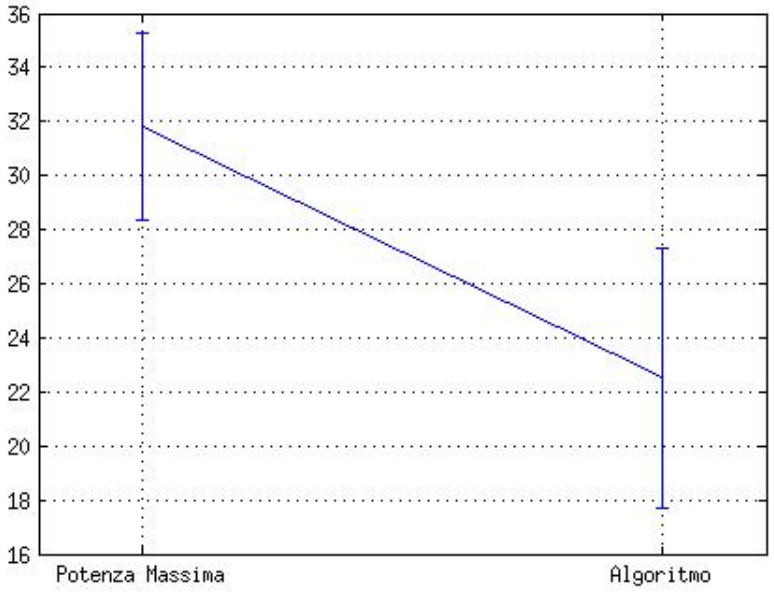


Figura 7.9: Intervalli di confidenza dei test MaxPower e Algorithm.

Capitolo 8

Istanze di configurazione dell'algoritmo

Successivamente alla validazione dell'algoritmo di adattamento della potenza trasmissiva, che ha portato ad un notevole miglioramento del controllo dei veicoli, si è deciso di analizzare diverse istanze di configurazione dell'algoritmo allo scopo di verificare se al variare di alcuni aspetti l'algoritmo tende ad peggiorare le performance o viceversa tende a migliorare le prestazioni.

Gli aspetti possibili da analizzare e modificare sono molti. Alcuni di questi risultano di maggior interesse rispetto ad altri e sono:

- l'ampiezza delle tre regioni *Near Region*, *Safe Region* e *Unsafe Region*;
- la politica di emergenza nel caso di perdita di un veicolo;

Per quanto riguarda l'ampiezza delle tre regioni *Near Region*, *Safe Region* e *Unsafe Region* non è detto che la configurazione attuale impostata al 10% della potenza trasmissiva per la *Safe Region*, 10% per la *Unsafe Region* e al 80% per la *Near Region* risulti la più performante. Si è pensato quindi di modificare tali grandezze ed effettuare dei test in modo da verificare se esistono altre istanze di configurazioni che ottimizzano l'algoritmo e quindi rendono migliore il controllo di formazione. La sezione 8.1 è dedicata esattamente a questo scopo.

Per la politica di emergenza nel caso di perdita di un veicolo, per esempio, si può pensare di modificarne il comportamento affinchè si comporti in maniera meno drastica come prevede la configurazione attuale. Tutt'ora infatti, in caso di mancata ricezione dei pacchetti da parte di un veicolo *Neighbor* per un tempo pari a 2 secondi, l'algoritmo imposta la potenza trasmissiva al valore massimo in modo da recuperare tale veicolo e permettergli di ritornare in formazione. L'idea in questo caso è di aumentare in maniera esponenziale il valore della potenza trasmissiva per permettere di minimizzare i consumi e soprattutto di creare meno intersezioni di range trasmissivi

con conseguente diminuzione del numero di collisioni. Tale configurazione è descritta e analizzata in sezione 8.2.

8.1 Ampiezza delle regioni

La scelta dell'ampiezza delle regioni trasmissive è un punto di fondamentale importanza nell'algoritmo di regolazione della potenza trasmissiva la cui modifica potrebbe avere un impatto non indifferente sulle prestazioni. Si è scelto di analizzare tre diverse istanze di configurazione dell'algoritmo con le regioni impostate nel seguente modo:

- Configurazione 1: 10% *Safe Region*, 10% *Unsafe Region*, 80% *Near Region*;
- Configurazione 2: 20% *Safe Region*, 20% *Unsafe Region*, 60% *Near Region*;
- Configurazione 3: 5% *Safe Region*, 5% *Unsafe Region*, 90% *Near Region*;

Queste configurazioni si basano su modifiche dell'ampiezza della *Safe Region*. La prima è la "classica" configurazione applicata finora all'algoritmo e di conseguenza in tutti i test precedentemente eseguiti in questa tesi. La seconda e terza configurazione prevedono, rispettivamente, un aumento della dimensione del del doppio e una diminuzione della metà della *Safe Region*. Come già specificato in Sezione 7.2 è auspicabile che i veicoli rimangano nella zona *Safe Region* il maggior tempo possibile in quanto il pericolo di uscire dal range di trasmissione è minimizzato. Da tale affermazione si è quindi scelto di procedere con la modifica delle ampiezze delle regioni di trasmissione ma non della politica di calcolo della nuova potenza trasmissiva. Infatti, in tutte e tre le configurazioni, il veicolo, in caso di adattamento della potenza trasmissiva, pone sempre il veicolo *Neighbor* esattamente a metà della *Safe Region* indipendentemente dall'ampiezza della regione. Nelle Figure 8.1, 8.2, 8.3 e 8.4 sono visualizzati i risultati dell'esecuzione dei test con le tre configurazioni sopra descritte. Tali risultati si riferiscono rispettivamente al valore totale d'errore di formazione, varianza, scarto quadratico medio e numero di collisioni. I risultati che riguardano il numero di pacchetti corrotti e il numero massimo di Backoff raggiunti non sono raffigurati in quanto i valori sono esattamente 0 in tutti i testcase.

Si nota che la seconda configurazione rispetto la configurazione "classica", e cioè la configurazione numero uno, sia leggermente peggiore in termini di errore totale di formazione e numero di collisioni. Precisamente l'errore totale di formazione passa da un valore di 12.4 metri della prima configurazione a un valore di 12.6 metri. La media del numero di collisioni passa da

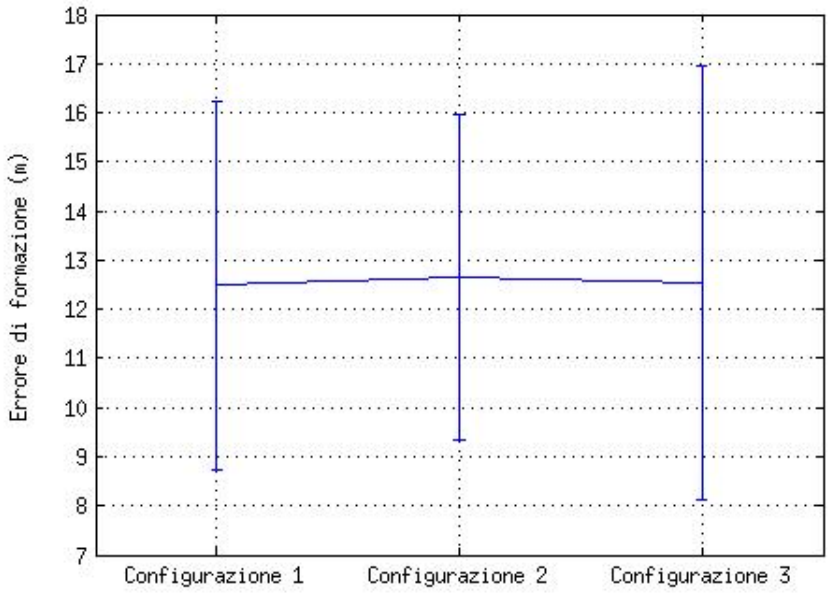


Figura 8.1: Errore totale di formazione al variare dell'ampiezza delle regioni.

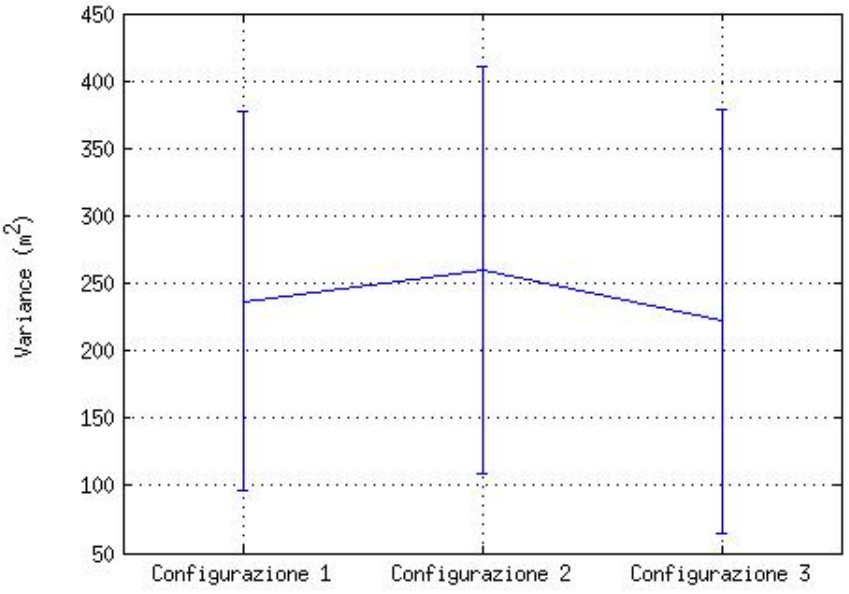


Figura 8.2: Varianza dei valori d'errore al variare dell'ampiezza delle regioni.

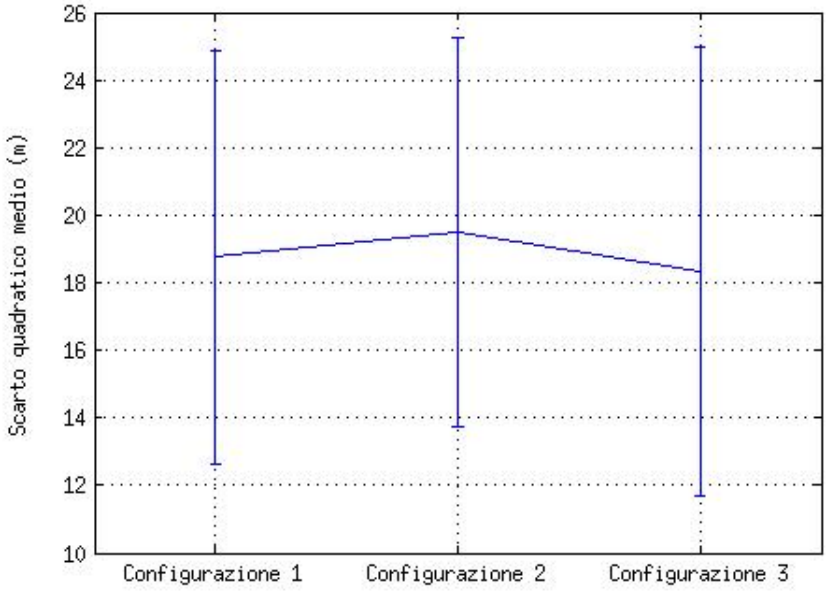


Figura 8.3: Scarto quadratico medio dei valori di errore al variare dell'ampiezza delle regioni.

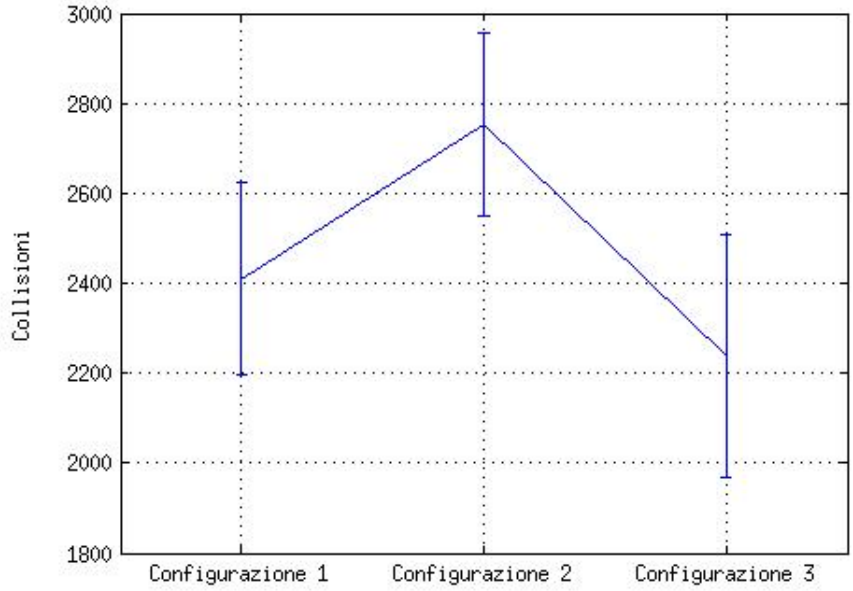


Figura 8.4: Numero totale di collisioni al variare dell'ampiezza delle regioni.

2410 a 2752. Questo ultimo risultato è dovuto all'aumento della dimensione della *Safe Region* e della *Unsafe region* che di conseguenza portano ad aumentare il range trasmissivo del veicolo trasmettitore il quale si interseca maggiormente con i range trasmissivi degli altri veicoli dello scenario.

La terza configurazione si è dimostrata migliore rispetto alle precedenti in termini di numero di collisioni. Si passa infatti da 2410 della prima configurazione, che fino ad ora era risultata la migliore, a 2239. Per quanto riguarda il valore d'errore totale di formazione la situazione non cambia rispetto alla seconda configurazione in quanto risulta un leggero peggioramento passando da un valore di 12.4 metri a un valore di 12.5 metri. L'intervallo di confidenza fa però notare che a confronto delle due configurazioni precedenti la terza configurazione genera una più alta variabilità dei dati d'errore.

In conclusione queste diverse configurazioni hanno migliorato il controllo di formazione sotto alcuni aspetti e peggiorato sotto altri. Si nota infatti come un miglior valore medio totale d'errore non corrisponda ad un minor numero di collisioni come ad esempio accade nella seconda configurazione. A questo punto è necessario un trade-off tra i diversi aspetti che porta ad eleggere la prima configurazione come la migliore in quanto risulta molto buona in termini di numero di collisioni, valore medio d'errore totale e soprattutto variabilità dei dati.

8.2 Perdita di un veicolo

La scelta di porre al massimo la potenza trasmissiva in caso di perdita di un veicolo non è detto sia la politica migliore da adottare nell'algoritmo di regolazione della potenza trasmissiva. E' noto infatti che l'utilizzo di un range trasmissivo molto ampio aumenta la probabilità di collisioni di pacchetti tra i veicoli in formazione. Ciò è causato dal problema dell' *Hidden node* che in una trasmissione wireless si presenta quando due nodi sufficientemente distanti da non essere in grado di "sentirsi" l'uno con l'altro comunicano con un *access point* comune. Entrambi i nodi quindi sentono il canale libero e trasmettono creando una collisione.

Un ragionamento interessante è il seguente: nel momento in cui si rileva la perdita di un veicolo e di conseguenza si pone al massimo la potenza trasmissiva come accade attualmente, la maggior parte della potenza utilizzata molto spesso viene sprecata. Questa conclusione deriva dal fatto che il veicolo disperso molto probabilmente si trova ad una distanza minore rispetto alla distanza coperta dalla potenza trasmissiva massima.

Da queste osservazioni si è deciso di creare un'istanza di configurazione dell'algoritmo la quale, in caso di perdita di un veicolo, aumenta la potenza trasmissiva a livelli discreti di potenza e ad ogni 5 Ts (1 secondo) e non più ponendola al livello massimo sin dal primo istante.

Tale configurazione prevede dieci livelli di potenza. I primi cinque livelli aumentano la potenza trasmissiva in maniera lineare, più precisamente aumentano di 20 metri il range trasmissivo ad ogni livello. I secondi cinque livelli invece aumentano in maniera esponenziale il range trasmissivo. Questa scelta deriva dal fatto che nella maggior parte dei casi il veicolo considerato "perso" è ancora molto vicino quindi non è necessario aumentare di molto la potenza nei primi livelli.

Una ulteriore modifica apportata grazie alla nuova configurazione, che chiameremo da qui in avanti configurazione a livelli, riguarda la scelta del range temporale oltre al quale un veicolo si può definire perso. Sino ad ora tale soglia era impostata in maniera totalmente arbitraria (2 secondi). Ora invece si è deciso di applicare una soglia pari al tempo in cui un veicolo percorre 60 metri alla velocità massima di 15 m/s. Il calcolo tiene conto del fatto che i due veicoli potrebbero percorrere direzioni esattamente opposte quindi la velocità nel calcolo deve essere raddoppiata. Il calcolo è il seguente:

$$t_{soglia} = \frac{60m}{30m/s} = 1s$$

Esattamente dopo un secondo dalla ricezione dell'ultimo pacchetto il veicolo risulta disperso e l'algoritmo applica la politica di aumento della potenza a partire dal livello numero 1.

Per confrontare l'efficacia dell'algoritmo con la configurazione a livelli rispetto all'algoritmo con la configurazione classica si è deciso di utilizzare uno scenario particolare aggiungendo un disturbo che crea la perdita del veicolo 2 della formazione. Tale scenario è esattamente lo scenario utilizzato per la validazione dell'algoritmo di regolazione della potenza trasmissiva in sezione 7.3. Nelle Figure 8.5, 8.6, 8.7 e 8.8 sono mostrati i risultati delle performance nelle due configurazioni. Tali risultati si riferiscono rispettivamente al valore totale d'errore di formazione, varianza, scarto quadratico medio e numero di collisioni. Il numero di pacchetti corrotti e il numero massimo di Backoff hanno restituito valori pari a 0 in tutti i testcase.

Le Figure 8.9 e 8.10 mostrano i valori assunti dalle potenze trasmissive del veicolo 1 della formazione nelle due configurazioni sopra descritte. Analizzando Figura 8.9 notiamo che la potenza trasmissiva viene posta al massimo anche in istanti della simulazione dove il disturbo al veicolo 2 (che nello specifico è *Follower* del veicolo 1) non è stato ancora provocato. Questa situazione è evidente nell'intervallo tra il quinto e il nono secondo. Tale situazione è dovuta ad una serie di collisioni continue di pacchetti in ricezione al veicolo 1 il quale, dopo un secondo dalla ricezione dell'ultimo pacchetto del veicolo 2, etichetta il veicolo 1 come "perso" ponendo la potenza trasmissiva al massimo. Il veicolo 2 però, tra il quinto e il nono secondo, si trova a pochi metri di distanza dal veicolo numero 1 dunque, come volevasi dimostrare, il fatto di porre la potenza trasmissiva a livello massimo provoca un enorme spreco di potenza e un elevata probabilità di intersezione con i

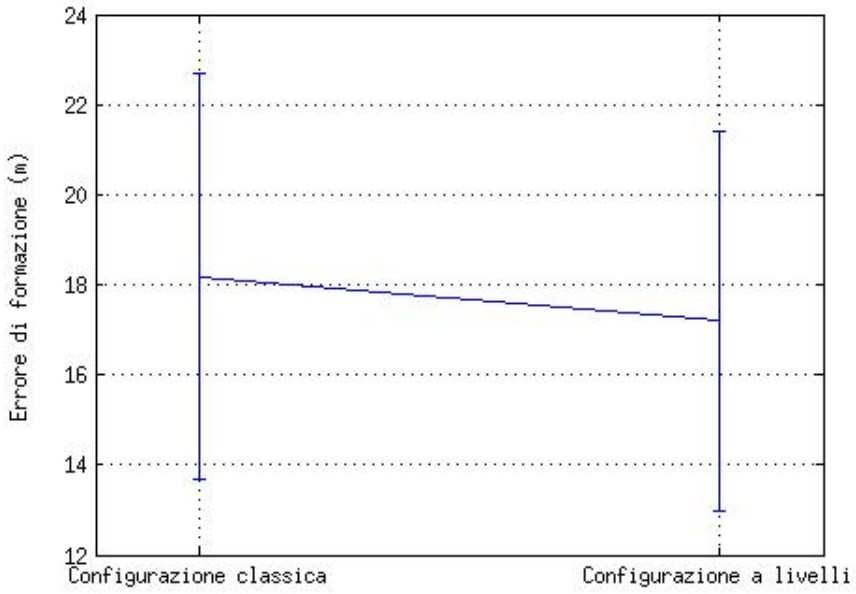


Figura 8.5: Errore totale di formazione al variare della politica trasmissiva in caso di perdita di un veicolo.

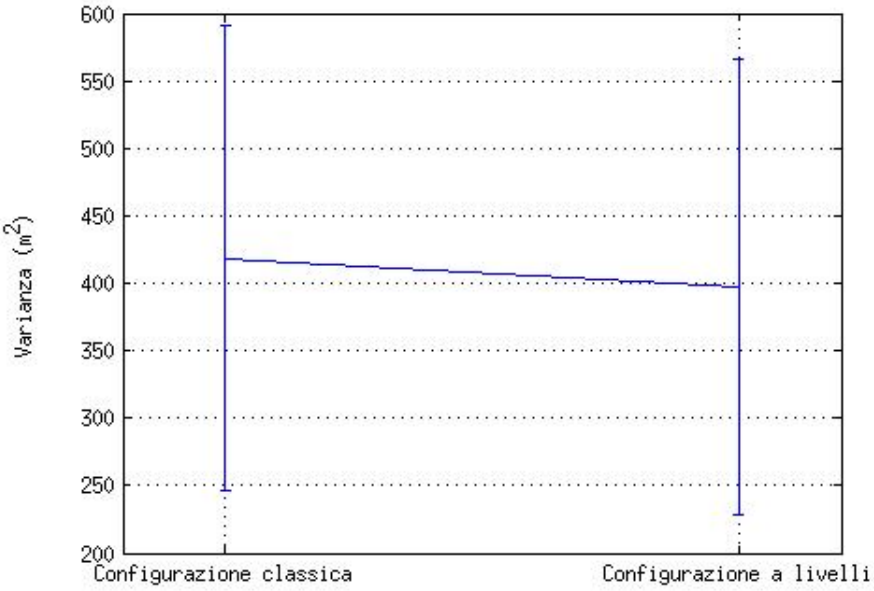


Figura 8.6: Varianza dei valori d'errore al variare della politica trasmissiva in caso di perdita di un veicolo.

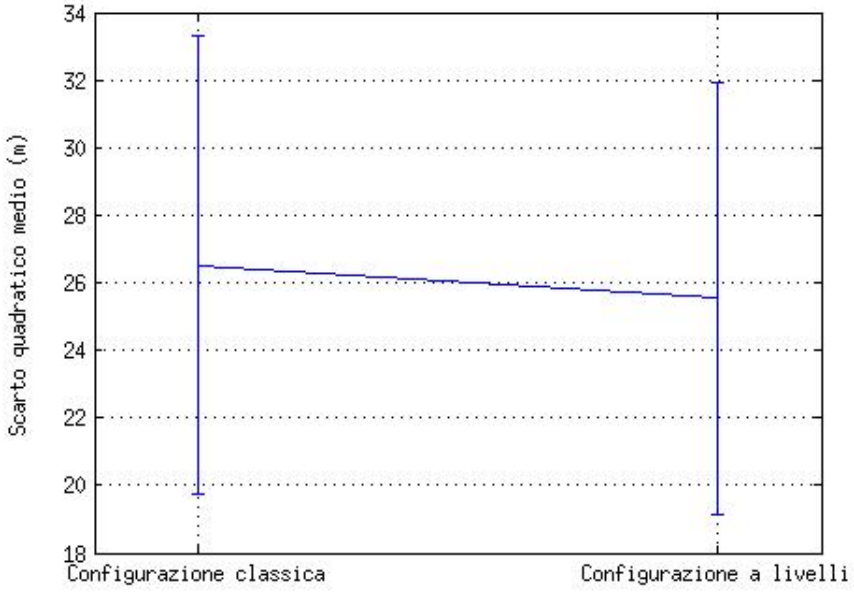


Figura 8.7: Scarto quadratico medio dei valori di errore al variare della politica trasmissiva in caso di perdita di un veicolo.

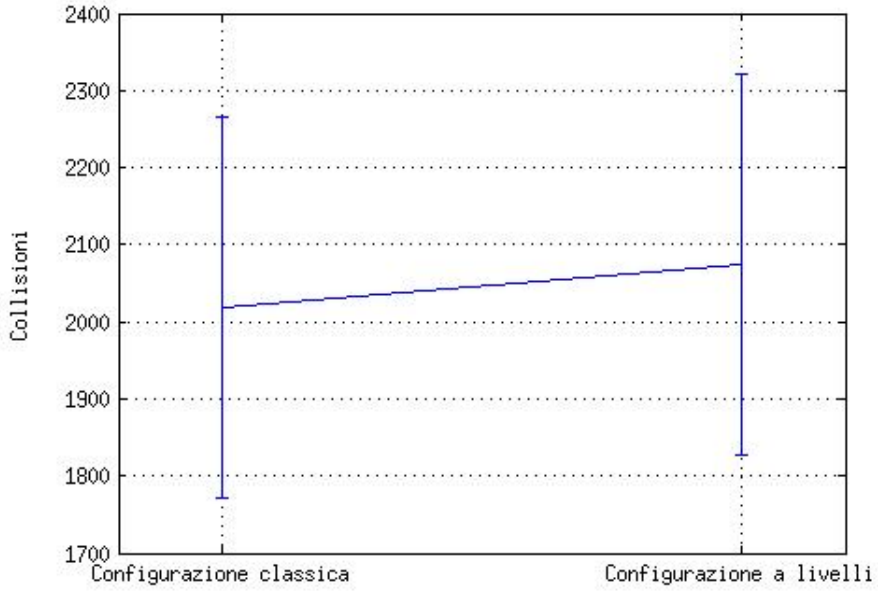


Figura 8.8: Numero totale di collisioni al variare della politica trasmissiva in caso di perdita di un veicolo.

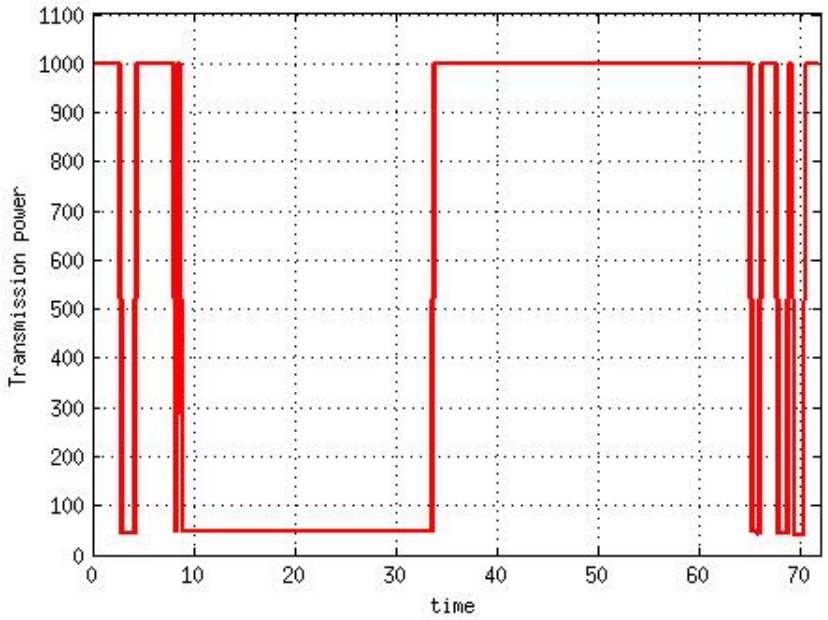


Figura 8.9: Potenza trasmissiva del veicolo numero 1 nella configurazione classica.

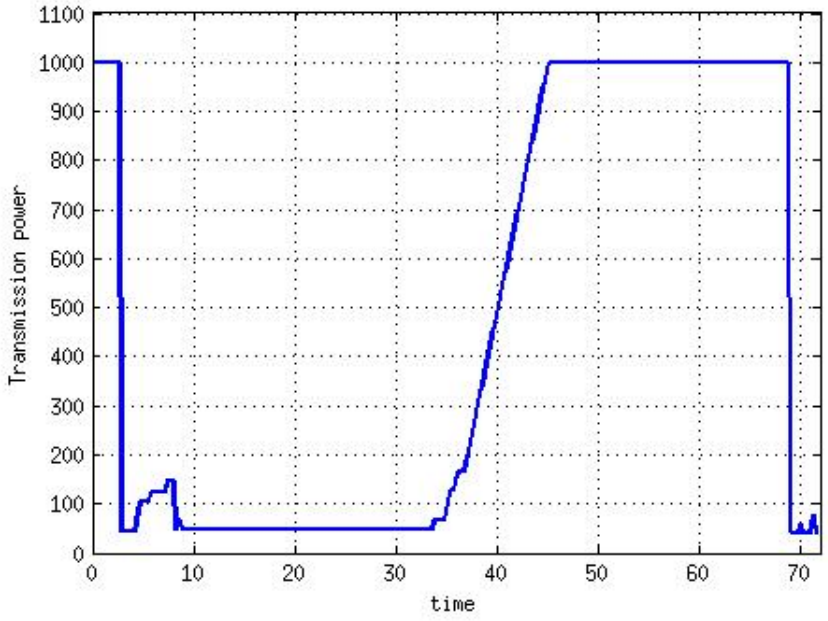


Figura 8.10: Potenza trasmissiva del veicolo numero 1 nella configurazione a livelli.

range trasmissivi di altri veicoli. Grazie alla Figura 8.10 notiamo invece che la configurazione a livelli dal quinto al nono secondo aumenta in maniera lineare la potenza trasmissiva fino ad un massimo di circa 200 metri, un range sufficientemente grande da poter rilevare il veicolo numero 2 evitando cosi sprechi eccessivi di potenza e diminuendo la probabilità di intersezioni con altri range trasmissivi rispetto la configurazione classica.

Si nota inoltre che nella configurazione a livelli i valori assunti dalla potenza trasmissiva presentano una variabilità molto bassa a differenza della configurazione classica che passa da valori vicini allo zero al valore 1000 molte volte durante le simulazioni.

Analizzando il comportamento delle due configurazioni durante il disturbo che provoca volutamente la perdita del veicolo 2 rilevata dal veicolo 1 (secondo 35 - 65) notiamo come tra il secondo 35 e il secondo 45 la configurazione classica imposta immediatamente al massimo la potenza trasmissiva ottenendo lo stesso risultato della configurazione a livelli la quale, incrementando linearmente e poi esponenzialmente la potenza trasmissiva, provochi un minor spreco di potenza e conseguentemente, come già specificato, una minor probabilità di collisioni.

Analizzando inoltre i valori d'errore di formazione la configurazione a livelli risulta migliore rispetto alla configurazione classica. Rispettivamente la media totale d'errore per la configurazione classica è di 18.1 metri con un intervallo di confidenza al 95% = [13.6m 22.6m] mentre per la configurazione a livelli è di 17.1 metri con un intervallo di confidenza al 95% = [12.9m 21.4m].

In conclusione di questa analisi si può affermare che la configurazione a livelli apporta netti miglioramenti al controllo di formazione in quanto, in caso di perdita di un veicolo, la politica prevede un minor spreco di potenza, range trasmissivi meno ampi e di conseguenza un minor numero di collisioni nonché ad un miglior controllo di formazione.

Capitolo 9

Conclusioni

Grazie al lavoro compiuto in questa tesi è stata estesa la libreria SCNSL al fine di modellare la velocità di propagazione del segnale fisico e di gestire l'impatto dei movimenti dei nodi sulle comunicazioni wireless. E' possibile ora estendere le simulazioni a scenari che utilizzano canali wireless nel caso della trasmissione radio e nel caso della trasmissione acustica. Per riprodurre fedelmente una trasmissione wireless è stato creato un meccanismo che gestisce la mobilità dei nodi all'interno della rete. In particolare è stata modificata la gestione degli eventi all'interno di SCNSL in modo da poter rischedulare gli stessi dopo uno o più movimenti dei nodi. Queste modifiche permettono ora la simulazione di scenari come la trasmissione mediante sonar.

Le innovazioni portate alla libreria SCNSL hanno reso possibile la creazione di scenari per studiare il problema del controllo di formazione. È stato creato quindi un modello Matlab per il controllo di formazione di tipo leader-following, che comprende un controllore incorporato in ciascuno dei veicoli followers per il raggiungimento/mantenimento della formazione. Successivamente, tramite il meccanismo della co-simulazione, si è introdotto il modello della rete per la comunicazione wireless tra i vari veicoli e si è potuto analizzare il suo impatto sull'algoritmo di controllo.

Inoltre è possibile eseguire un notevole numero di simulazioni di ambienti wireless grazie al tool creato in questa tesi. Il tool è risultato molto preciso nei risultati e realistico nella fase di simulazione, rivelandosi di grande utilità all'utente finale grazie alla completa automatizzazione del lavoro che va dall'esecuzione delle simulazioni alla raccolta dei dati.

Dall'analisi effettuata sui risultati, si è notato che si ottengono prestazioni migliori a livello di controllo quando l'interferenza trasmissiva tra i veicoli è minima. Si è sviluppato quindi un algoritmo che ottimizza la potenza trasmissiva in modo da creare la minor interferenza possibile nella rete. Ogni nodo in base alla distanza dei propri *vicini* regola la potenza trasmissiva in modo da garantire un margine di raggiungibilità per ciascuno di loro anche

in presenza di improvvisi spostamenti.

I risultati ottenuti dimostrano che tramite l'uso dell'algoritmo si ha una diminuzione dell'errore di formazione medio, del numero di collisioni e della variabilità dei dati.

La modifica di alcuni aspetti dell'algoritmo quali l'ampiezza delle regioni e la politica di trasmissione in caso di perdita di un veicolo hanno portato a netti miglioramenti nell'algoritmo dimostrando che la configurazione "classica" utilizzata per la quasi totalità del lavoro di tesi può essere migliorata.

L'adattamento della potenza trasmissiva come soluzione migliorativa per il controllo di formazione è un esempio di allargamento dello spazio di progetto di networked control system agli aspetti trasmissivi rispetto al tradizionale approccio focalizzato sulla progettazione del solo controllore. L'estensione dello strumento di simulazione e la sua applicazione ad uno scenario di controllo di formazione mostra che tale strumento può essre usato efficacemente per la realizzazione di NCS in cui sia gli aspetti di controllo sia quelli trasmissivi possono essere oggetto dell'esplorazione di progetto.

Bibliografia

[1] T. Balch, R.C. Arkin: "Behavior-based formation control for multirobot team", IEEE Trans. Robot. Autom., 1998, 14, (6), pp. 926-939.

[2] P. Ogren, M. Egrstedt, X. Hu: "A control Lyapunov function approach to multi-agent coordination", IEEE Trans. Robot. Autom., 2002, 18, (5), pp. 847-851.

[3] J. Shao, G. Xie, L. Wang: "Leader-following formation control for multiple mobile vehicles".

[4] R. T. Jonathan, R. W. Beard, B. J. Young, "A Decentralized Approach to Formation Maneuvers", IEEE Trans. Robot. and Automat., Vol. 19, pp. 933-941, 2003.

[5] P. K. C. Wang, "Navigation Strategies for Multiple Autonomous Mobile Robots Moving in Formation", J. Robot. Syst., Vol. 8, No. 2, pp. 177-195, 1991.

[6] P. K. C. Wang, F. Y. Hadaegh, "Coordination and Control of Multiple Microspacecraft Moving in Formation", J. Austronautical Sci., Vol. 44, No. 3, pp. 315-355, 1996.

[7] M. Anthony Lewis, Kar-Han Tan, "High Precision Formation Control of Mobile Robots Using Virtual Structures", The Commotion Laboratory, Computer Science Departement, Universitu of California, Los Angeles, 1997.

[8] F. Lavratti, A. R. Pinto, L. Bolzani, F. Vargas, C. Montez, F. Hernandez, E. Gatti, C. Silva, "Evaluating a Transmission Power Self-Optimization Technique for WSN in EMI Environments", 2010 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools.

[9] F. Lavratti, A. R. Pinto, D. Prestes, L. Bolzani, F. Vargas, C. Montez, "Towards a Transmission Power Self-Optimization in Reliable Wireless Sensor Networks", Proc. 11^{th} IEEE Latin American Test Workshop, 2010.

[10] A. R. Pinto, M. Camada, M. A. R. Dantas, C. Montez, P. Portugal, F. Vasques, "Genetic Machine Learning in the Optimization of Communication Efficiency in Wireless Sensor Networks", Proc. Annual Conference of the IEEE Industrial Electronics Society, IECON 2009.

[11] IEEE Std 1666 - 2005 IEEE Standard SystemC Language Reference Manual, IEEE Std 1666-2005 (2006) 1-423.

[12] F. Stefanni, D. Quaglia, F. Fummi, SystemC Simulation of Networked Embedded Systems, in: Languages for Embedded systems and their Applications, Springer Lecture Notes in Electrical Engineering, vol. 36, pp. 201-211, 2009.

[13] M.-Y. Chow, Z. Sun, H.Li, Optimal stabilizing gain selection for networked control systems with time delays and packet losses, Control Systems Technology, IEEE Transactions on 17 (5) (2009) 1154-1162.

[14] T. Matiakis, S. Hirche, M. Buss, Control of networked systems using the scattering transformation, Control Systems Technology, IEEE Transactions on 17 (1) (2009) 60-67.

[15] G. Walsh, H. Ye, L. Bushnell, Stability analysis of networked control systems, Control Systems Technology, IEEE Transactions on 10 (3) (2002) 438-446.

[16] F.-L. Lian, J. Moyne, D. Tilbury, Network design consideration for distribuited control systems, Control Systems Technology, IEEE Transactions on 10 (2) (2002) 297-307.

[17] Mathworks Co., Matlab, URL: http://www.mathworks.com.

[18] Davide Quaglia, Riccardo Muradore, Roberto Bragantini, Paolo Fiorini, "A SystemC/Matlab Co-Simulation Tool for Networked Control Systems", Departement of Computer Science, University of Verona, Italy.