UNIVERSITÁ DI PISA

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea

Progettazione e sviluppo di una architettura di controllo per piattaforme di Stewart in simulatori di motociclo

Relatori:

Prof. Massimo Bergamasco

Prof. Paolo Ancilotti

Ing. Carlo Alberto Avizzano

Candidato:

Federico Paolinelli

Anno Accademico 2002/2003

Alla mia famiglia,

ai miei nonni,

e a Laura

Ringraziamenti

Vorrei ringraziare tutti gli amici del laboratorio PERCRO per il bel periodo passato in questi mesi. In particolare vorrei ringraziare l'Ing. Carlo Alberto Avizzano per avermi seguito ed aiutato nei momenti più difficili di questo percorso, il prof. Massimo Bergamasco per avermi messo a disposizione le strutture del laboratorio e tutti quelli che mi sono stati di aiuto con i loro consigli e la loro esperienza.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

1.1 I simulatori

Un simulatore è un sistema in grado di replicare un'attività senza l'uso dei normali strumenti utilizzati per eseguire l'attività stessa.

I vantaggi di un'esperienza simulata sono numerosi:

- Controllabilità: in ogni momento la simulazione può essere interrotta.
- Analizzabilità: i dati registrati durante la simulazione possono essere utilizzati per studi successivi, come ad esempio analisi delle prestazioni dell'utilizzatore.
- Controllabilità degli eventi: tutti gli eventi che occorrono durante la simulazione sono determinati, e possono essere controllati.
- Sicurezza: l'utilizzatore del simulatore non è esposto alle condizioni di rischio che potrebbero esserci nel caso dell'esperienza reale.

Affinchè la simulazione sia verosimile, il simulatore deve rispondere in modo realistico alle "azioni virtuali". Per questo motivo, il simulatore dovrebbe contenere sia modelli che sono una buona rappresentazione della realtà, che nuovi strumenti e sistemi di controllo ad elevata qualità.

Un simulatore ha quindi il vantaggio di trasferire un problema fisico in un ambiente virtuale, dove è possibile effettuare tentativi finché non sono ottenuti risultati

incoraggianti, ma possiede anche lo svantaggio che è necessario operare con strumenti e modelli capaci di rappresentare correttamente lo svolgimento delle cose nella dimensione "virtuale".

Il buono sviluppo di un simulatore dipende dalle capacità dell'uomo di studiare, realizzare e implementare modelli che riproducono la realtà.

La possibilità di usare strumenti di calcolo come i computer ha avviato le attività di ricerca sui simulatori negli ultimi diciotto anni. È possibile simulare, prima dell'esecuzione reale, un circuito elettrico, l'attività di un'impianto chimico, un'operazione chirurgica e così via.

1.1.1 Simulatori di guida

La topologia più diffusa di simulatori è quella dei simulatori di guida.

Nei simulatori di guida è riprodotta l'interazione di una persona con un veicolo, e si realizza che il problema della simulazione è limitato a tre sensi di una persona: tatto, vista e udito. Un'altra componente dell'apparato umano da tenere in considerazione è l'apparato vestibolare, che percepisce il feedback inerziale.

I simulatori di guida si dividono in due grandi categorie:

- *fixed-base*, dove non è fornito nessun feedback di movimento al guidatore, che è solidale con il terreno. Gli effetti dinamici delle condizioni reali (accelerazioni e velocità) sono replicate solo con feedback visivo e auditivo;

- *motion-based*, dove tra il guidatore e il terreno è inserito un dispositivo meccanico che riproduce gli effetti dinamici, in aggiunta a feedback visivi e auditivi.

I benefici dell'utilizzo di un simulatore motion based si individuano nella riduzione del tempo di apprendimento, migliore performance dell'operatore, minore disagio causato dalla discordanza degli stimoli visivi con quelli di percezione del movimento.

1.2 Il simulatore MORIS

Il simulatore di motocicli *MORIS* (Motorcycle Rider Simulator) è stato sviluppato a seguito del progetto europeo *ESPRIT 20521*, in una collaborazione tra il laboratorio

PERCRO (PERCeptual RObotics) della Scuola Superiore S.Anna, dell'università di Halmstad (Svezia) e del gruppo Piaggio, con lo scopo di creare uno strumento di ausilio per la progettazione e la realizzazione di veicoli a due ruote. Si tratta in pratica della riproduzione di un motociclo sensorizzata ed attuata, in grado di ricavare valori ottimali di alcuni parametri meccanici, per ridurre il numero di test su strada richiesti dopo la fabbricazione di un prototipo (figura 1.1).

Figura 1.1. Il simulatore MORIS

MORIS è un simulatore "Motion-based", ovvero è fornito di parti meccaniche con lo scopo di riprodurre, con buona approssimazione, la dinamica di un vero motociclo. Il sistema finale è costituito dal mock-up di un veicolo a 2 ruote; per mock-up si intende una struttura rigida che è mossa rispetto al terreno da un meccanismo (sistema di attuazione) che possiede il numero necessario di gradi di libertà.

L'operatore umano è in contatto con la struttura meccanica al livello delle mani (con il manubrio), dei piedi (con la pedaliera) e della schiena (con un inclinometro).

Il sistema prevede inoltre un'interfaccia grafica che permette all'operatore di ottenere una riproduzione dell'ambiente virtuale in cui si trova il veicolo.

1.3 Simulatori e didattica

L'idea di utilizzare simulatori e realtà virtuale per addestrare piloti non è nuova. In particolare, i simulatori di volo sono stati per anni il punto di riferimento nel campo della simulazione di veicoli, a causa della differenza di costo tra un veicolo reale ed un simulatore, e dell'alto rischio derivante da un cattivo utilizzo dello stesso. Nel campo dei veicoli automobilistici invece, la fase di apprendimento è facilitata dal fatto che la guida reale viene assistita da un istruttore a bordo del veicolo, che in caso di situazione critica prende il controllo del mezzo; questo non è possibile nel caso di motocicli, perchè la presenza di un istruttore a bordo del veicolo nella fase di apprendimento non solo renderebbe più difficile la guida del veicolo, ma non sarebbe comunque in grado di recuperare le situazioni critiche come nel caso degli autoveicoli.

La fase di apprendimento per la guida di motocicli è ancora più importante se si pensa che nella maggior parte dei casi costituisce la prima esperienza reale di guida su strada.

Con l'imminente introduzione dell'obbligo di corsi di educazione stradale e di guida per i giovani, e con l'intenzione da parte di enti pubblici di promuovere tali corsi, un simulatore di guida si rivela essere un mezzo sicuro ed utile per la didattica: sicuro, poiche' permette agli utilizzatori di fare pratica con i mezzi a due ruote, senza essere esposti ai rischi delle prime esperienze con un mezzo reale; utile per la didattica, in quanto permette di ricreare particolari situazioni ambientali che il "candidato" si deve trovare ad affrontare, ad esempio situazioni di rischio. Il simulatore permette infatti di replcare le sensazioni di guida, sia in termini di accelerazioni e decelerazioni, di controllo e di contatto della struttura fisica, che in termini di feedback visivo. Può essere inoltre utilizzato anche come strumento di valutazione della capacità di un candidato ad affrontare la guida di un mezzo a due ruote senza dover ricorrere ad un test su strada.

1.4 Obiettivi

Il progetto MORIS è stato completato nell'anno 2000. Alla data di ultimazione dei lavori, esso era organizzato con una struttura Hardware/Software (descritta nei capitoli successivi) in grado di effettuare la riproduzione di un'esperienza di guida, ma poco flessibile per una riconfigurabilità in applicazioni interattive, come ad esempio la didattica.

Come la quasi totalità dei simulatori esistenti, il sistema MORIS è un simulatore di guida di tipo "chiuso", ovvero la cui struttura di controllo è stata predefinita per operare nell'ambito della replica verosimile di guida un un ambiente virtuale prestrutturato.

La didattica tramite strumenti di simulazione impone invece una nuova concezione degli strumenti di simulazione che siano quindi basati su:

- *flessibilità,* ovvero la capacità del sistema di potersi predisporre per esercizi e test di varia natura in maniera veloce ed agevole, e soprattutto in base alle esigenze didattiche del corso.

- *Difficoltà graduale*, ovvero la capacità di sfruttare il sistema di simulazione a livelli intermedi di complessità in maniera da fornire un supporto propedeutico alla guida che richiede una abilità proporzionata al livello di apprensione del sistema.

- *Interazione multipla*, in modo da consentire all'istruttore di interagire con il pilota durante la fase di simulazione.

Lo scopo di questa tesi è quello di fornire uno strumento *sicuro* per il controllo e la gestione di un simulatore di motocicli, in modo da far fronte a tutte le situazioni che possono costituire un rischio per la sicurezza di chi utilizza il sistema; questo è un requisito fondamentale, data l'elevato grado di interazione tra il sistema e l'utilizzatore, e la presenza di parti in movimento.

Vogliamo poi che l'utilizzo del sistema sia *facile* ed *intuitivo*, in modo da non richiedere la presenza costante di personale qualificato per il suo funzionamento.

Si desidera che le modifiche che verranno apportate al sistema, oltre a migliorarne il funzionamento e l'usabilità, ne facilitino anche lo sviluppo e la manutenzione. Pertanto sono stati identificati anche altri requisiti che il sistema deve avere.

Il sistema deve essere *modulare*, per facilitare lo sviluppo e la verifica del sistema nelle sue parti, e in seguito nella sua interezza.

Il sistema deve essere facilmente *diagnosticabile* e *verificabile*, per poter trovare rapidamente le cause di eventuali malfunzionamenti.

Il sistema deve essere *estensibile*, per aggiungervi nuove caratteristiche senza grossi sforzi e senza grandi modifiche al sistema attuale, ad esempio per poter sfruttare nuove tecnologie a disposizione.

Lo strumento deve essere facilmente *riconfigurabile*, in modo da poter apportare in maniera semplice modifiche necessarie a soddisfare nuove possibili esigenze.

Il sistema deve rispettare degli *standard*, per essere in grado di sfruttare senza alcun costo aggiuntivo i miglioramenti che vengono apportati agli standard utilizzati.

Infine, il sistema di controllo del simulatore deve essere *portabile,* ovvero deve essere il più possibile indipendente dalla specifica architettura di questo simulatore, in modo da poter essere sfruttato per lo sviluppo di altri sistemi simili.

Capitolo 2

Descrizione del sistema MORIS

In questo capitolo si procederà alla descrizione della struttura iniziale del sistema MORIS.

Approfondiremo in particolare quella che era l'architettura di calcolo e controllo del sistema prima dell'intervento descritto in questa tesi.

2.1 Descrizione generale del sistema

Il sistema MORIS puo' essere definito come composizione di vari elementi.

2.1.1 Il sistema meccanico

Il sistema meccanico è costituito dalla riproduzione della moto e dal meccanismo utilizzato per muoverla.

2.1.1.1 Il sistema di attuazione

La cinematica di un simulatore di guida richiede un largo numero di gradi di libertà (sei o più). Sei gradi di libertà sono i minimi necessari alla rappresentazione di un qualsiasi atto di moto per un oggetto rigido nello spazio. Un numero minore di gradi di libertà può simulare accuratamente solo una parte delle manovre possibili del veicolo.

Nel sistema MORIS il movimento del veicolo è reso possibile da una piattaforma di Stewart sviluppata appositamente (figura 2.1).

Figura 2.1. La piattaforma di Stewart

Una Piattaforma di Stewart è l'architettura maggiormente usata nel campo dei simulatori di volo. Essa è costituita da una piattaforma collegata ad una base per mezzo di un sistema di attuazione parallela, costituito da sei attuatori lineari. L'impiego di una struttura parallela facilita in questo caso il raggiungimento di elevate prestazioni dinamico di movimento necessarie allo sviluppo di simulatori.

La piattaforma è dotata di sei gradi di libertà (x, y, z e roll, pitch e yaw), e di uno spazio di funzionamento limitato dall'estensione massima degli attuatori (workspace). La conformazione della piattaforma è tale per cui una configurazione dell'estensione dei sei pistoni determina univocamente la posizione e l'orientamento della piattaforma nello workspace.

Poiché nella realtà, i motocicli rispondono rapidamente alle variazioni di posizione del pilota e ai comandi impartiti (a differenza ad esempio di un aereo), e a causa del fatto che il simulacro della moto pesa circa 400 Kg, la piattaforma di Stewart è stata progettata per avere prestazioni ottimali.

I sei attuatori lineari della piattaforma di MORIS sono mossi da un sistema idraulico ad alta pressione (120 atmosfere).

Figura 2.2. L'impianto idraulico

Ogni attuatore è guidato da due servovalvole, il corpo dell'attuatore, un canale di sfogo, più un componente elettronico che acquisisce il segnale di un sensore LVDT (per leggere l'estensione dell'attuatore) e che comanda le servovalvole; andando a pilotare l'apertura delle servovalvole, si regola il flusso di olio che attraversa il pistone, e di conseguenza la sua estensione.

Le specifiche principali della piattaforma di Stewart sono:

- Dimensioni: raggio della base superiore 0.3 m, raggio della base inferiore 0.6 m, semi-angolo tra gli attuatori 15 gradi, altezza di riferimento 1.4 m.
- Spinta degli attuatori: 9 kN
- Corsa degli attuatori: 600 mm
- Velocità degli attuatori: 0.4 m/s
- Accuratezza 1%

- Carico in movimento: 600 Kg

2.1.1.2 Il mock-up del veicolo

Il mock-up costituisce l'interfaccia fisica naturale tra il pilota e il sistema MORIS, con le stesse componenti presenti su un veicolo reale (manubrio e relativi comandi come leve dei freni, acceleratore, sedile, pedaliera). Trasferisce i comandi del pilota al sottosistema di acquisizione dei dati. La parte inferiore è composta da una piattaforma con una protezione laterale per evitare possibili cadute del pilota.

Il mock-up del sistema MORIS è stato progettato e sviluppato modificando il corpo di un veicolo Piaggio Hexagon 150 in alcune sue parti.

Figura 2.3. Il mockup della moto di MORIS

Le ruote e le sospensioni anteriore e posteriore sono state rimosse, e sostituite con due gambe rigide, che collegano il mockup alla piattaforma di Stewart. Il motore è stato rimosso, e sostituito con un'unità che provvede riprodurre la vibrazione del motociclo in movimento.

2.1.1.3 Lo sterzo

Per rendere più realistico il sistema, lo sterzo è stato modificato per poter essere connesso in asse ad un motore brushless e ad un resolver, in modo da simulare il ritorno di forza sullo sterzo durante le fasi di guida. La trasmissione della coppia meccanica dal motore all'asse dello sterzo è di tipo diretto.

2.1.2 Il sistema di acquisizione dei dati

Il sistema di acquisizione dei dati denominato *rider command acquisition* (RCA) ha il compito di leggere i dati immessi dal pilota, ovvero

- Forza impartita sulle leve dei freni anteriore e posteriore
- Posizione dell'acceleratore
- Controlli vari, come i tasti di accensione, delle luci, del clacson, la chiave
- Inclinazione del pilota

In modo da poterli inoltrare al modello dinamico del sistema e quindi al sottosistema di attuazione.

Le quantità analogiche che rappresentano l'acceleratore, le forze sui freni e l'inclinazione del pilota sono acquisite per mezzo di un set di encoder piazzati sotto il mock-up. Gli impulsi lineari sono trasformati in rotazioni mediante una puleggia. Inoltre la puleggia è collegata anche ad una molla per simulare la forza di reazione dell'acceleratore e dei freni. Il sistema di acquisizione è rappresentato in figura 2.4.

2.1.3 L'architettura di calcolo esistente

L'architettura di calcolo del simulatore è un'architettura distribuita costituita dai seguenti moduli:

2.1.3.1 Console (CONS)

È il modulo responsabile dell'interazione dell'operatore con il sistema. Durante la simulazione permette il monitoraggio dei dati più significativi. Inoltre l'operatore

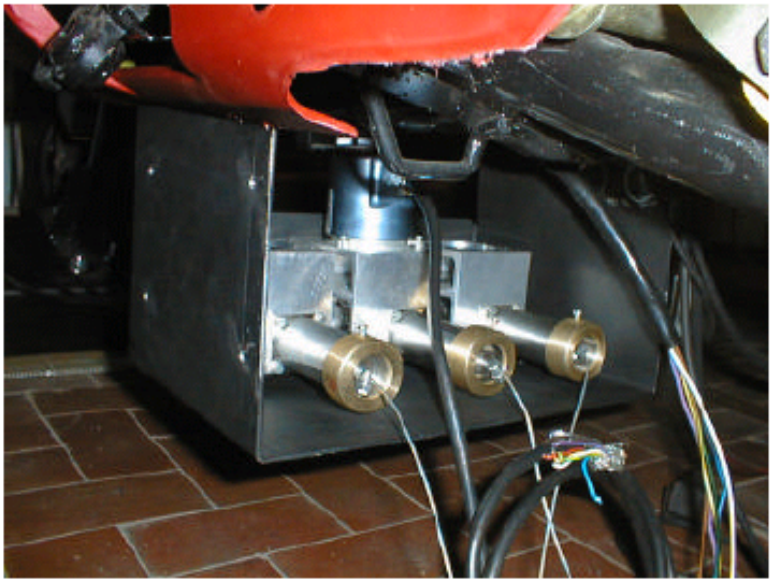

Figura 2.4. Il sistema di acquisizione dei dati

può dare in qualunque momento un allarme che ferma la simulazione. È costituita da un Pc con processore Intel e sistema operativo Linux RedHat con ambiente grafico Gnome.

2.1.3.2 Real Time System (RT)

Ha il compito di calcolare il feedback inerziale, di coordinare la sincronizzazione di tutte le attività e di controllare lo stato del sistema.

È costituito da un Pc Digital AXP-Pci 33 con processore Alpha 21064 a 166 MHz, diskless e con sistema operativo VxWorks.

2.1.3.3 Model subsystem (MOS)

Calcola, sulla base di un modello matematico della dinamica del motociclo, i nuovi riferimenti di coppia e posizione per l'impianto a partire dai comandi impostatati dal pilota.

2.1.3.4 Frame Controller (FC)

Implementa il controllo di posizione e di coppia dell'impianto.

È costituito da un Pc Digital AXP-VME con processore Alpha 21064 a 166 MHz, diskless, con bus PCI e VME, equipaggiato con sistema operativo montato è VxWorks. Sul bus VME sono poi montate le due schede di Input/Output digitale e analogico TVM-745 (figura 2.5).

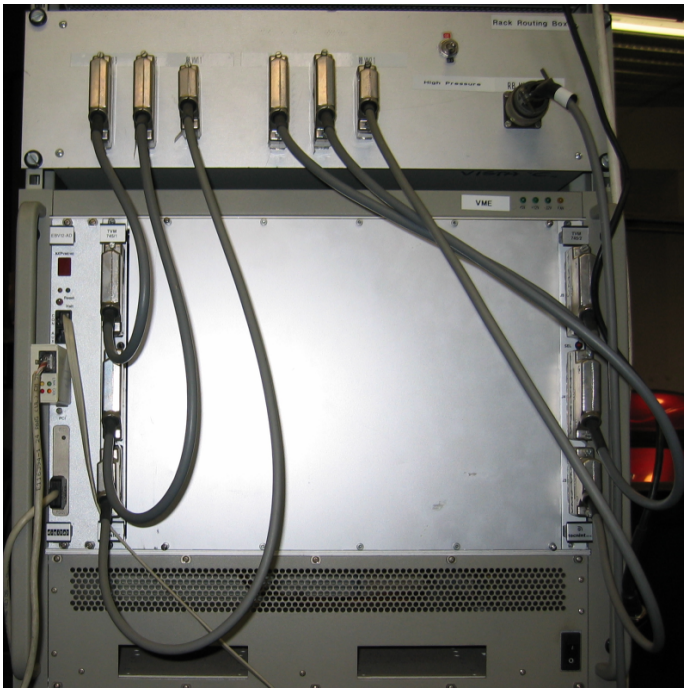

Figura 2.5. La macchina MorisFC

2.1.3.5 Graphical Subsystem (GS)

Ha il compito di fornire il feedback visivo all'utilizzatore del simulatore MORIS. Il GS è connesso con il RTS da cui acquisisce la posizione assoluta della testa del pilota nell'ambiente virtuale e valuta il punto di vista.

È costituito da una WorkStation Onix della Sylicon Graphics, equipaggiata con quattro processori 4400 a 250 MHz, 256 MB di RAM, 16 MB di texture memory.

2.1.3.6 Development Subsystem (MorisDev)

Ospita i file system dei sottosistemi RTS e FC e i relativi tool di sviluppo.

È costituito da un Pc Digital AXP-Pci 33, con hard disk scsi e sistema operativo DEC Unix della Digital. Oltre ad ospitare i file system dei due sottosistemi diskless, fornisce loro il sistema operativo VxWorks tramite richiesta di tipo BOOTP e trasferimento tramite protocollo TFTP.

2.1.4 L'interconnessione dei sottosistemi

Tutti i moduli descritti nell'architettura di calcolo sono collegati tramite una rete locale ethernet a 10 Mbit. I due moduli FC ed RT sono collegati inoltre tramite un'interfaccia FDDI a 100Mbit per garantire alte performance (fase di simulazione) e maggiore affidabilità rispetto alla ethernet.

Il modulo FC è poi connesso al sistema di attuazione della piattaforma per mezzo di due schede di IO montate su bus VME. Le connessioni del sistema sono descritte in figura 2.6.

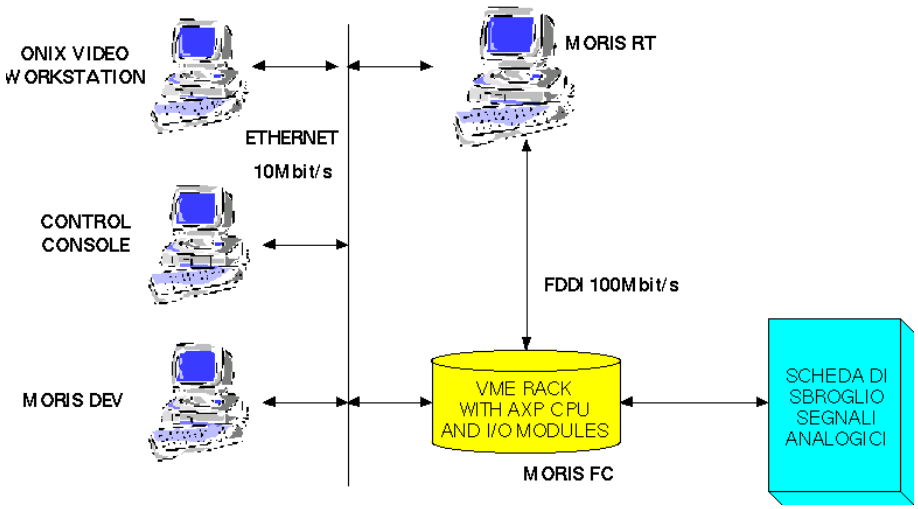

Figura 2.6. L'interconnessione dei sistemi

2.1.4.1 L'interfaccia FDDI

L'interfaccia FDDI (Fiber Distributed Data Interface) specifica una LAN a 100 Mbps di tipo doppio token ring; era spesso usata come tecnologia per WAN (wide area networks) per la larga banda supportata e perché le fibre ottiche supportavano distanze superiori a tecnologie a base di rame.

FDDI usa un'architettura a doppio anello, con il traffico che scorre in direzioni opposte sui due anelli. Gli anelli sono divisi in primario e secondario; durante il normale funzionamento, l'anello primario è usato per la trasmissione dei dati, e il secondario rimane inattivo. Lo scopo principale del doppio anello è quindi quello di fornire maggiore affidabilità e robustezza al sistema. Oltre al doppio anello, FDDI fornisce anche altri accorgimenti atti ad aumentare l'affidabilità del mezzo.

2.1.4.2 Il bus VME

Il bus VME (VERSA Module Eurocard) è un bus a 32 bit utilizzato prevalentemente per applicazioni industriali. Consente il collegamento, l'alimentazione la coordinazione e lo scambio di informazioni tra le schede collegate ad esso.

Nel caso specifico della macchina Alpha-VME della Digital, il bus Vme è collegato al processore attraverso un bridge situato nel bus PCI.

2.2 Introduzione alla architettura software

Procediamo adesso ad analizzare i moduli logici del sistema, che svolgono compiti diversi e sono ospitati di sottosistemi di calcolo diversi.

La struttura generale del sistema è rappresentata in figura 2.7.

Le interazioni dell'utente con il mockup del veicolo vengono registrate dal modulo RCA, e fornite in ingresso ad un modello che calcola le dinamiche del veicolo. Queste vengono elaborate dal modulo Washout filter, che fornisce in uscita la posizione che la piattaforma di Stewart deve assumere, il nuovo punto di vista da fornire alla grafica e i valori di coppia dello sterzo.

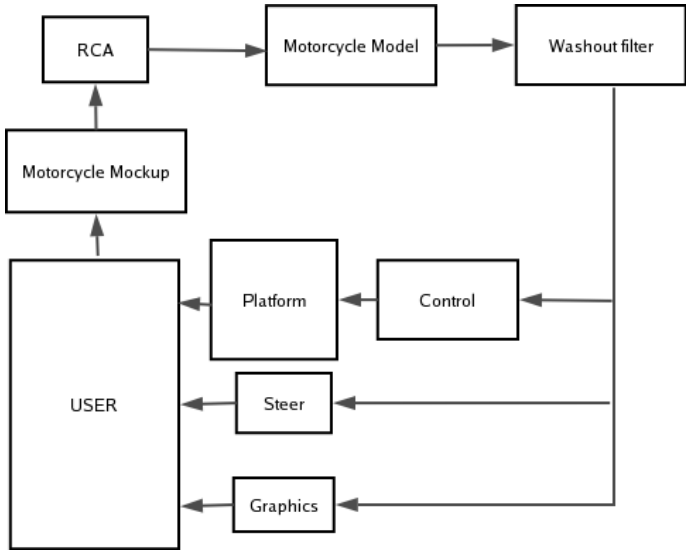

Figura 2.7. I moduli logici del sistema

2.2.1 Modello dinamico del veicolo

Il nucleo del software di MORIS è il software che simula la dinamica del veicolo a due ruote. Questo modulo risolve in tempo reale le equazioni del moto calcolate per il sistema complesso che approssima le caratteristiche meccaniche, geometriche e inerziali del veicolo che verrà guidato dall'operatore.

Il modello semplifica la moto in quattro corpi rigidi (fig. 2.8):

1. Il blocco posteriore
2. Il blocco anteriore
3. La ruota posteriore
4. La ruota anteriore

Il blocco anteriore è connesso al blocco posteriore per mezzo di un cardine cilindrico. La ruota anteriore ha un grado di libertà traslazionale rispetto al blocco anteriore. Lo stesso vale per la ruota posteriore rispetto al blocco posteriore.

Per quanto riguarda la replicazione delle forze di inerzia, sono considerate solo le parti del motociclo che sono in contatto con il pilota. L'ipotesi fatta è che il pilota

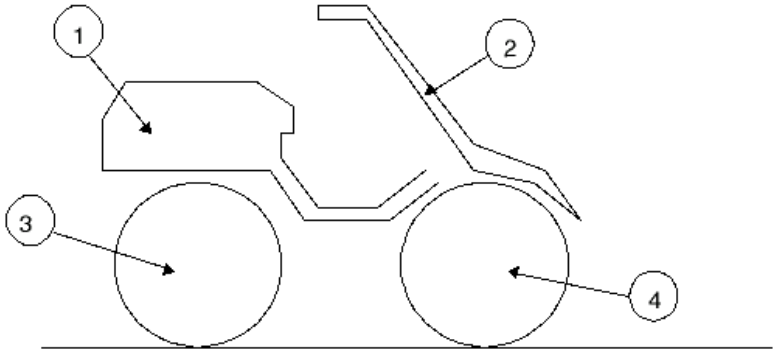

Figura 2.8. La divisione del modello della moto

sia in contatto solo con il blocco anteriore e posteriore del motociclo, e quindi, indipendentemente dal numero di corpi che schematizzano il motociclo, l'informazione necessaria per la corretta riproduzione delle forze inerziali è solo quella del moto dei blocchi posteriore e anteriore.

Il modello permette di generare profili di strada in salita e in discesa. Per includere questo tipo di scenario, tutte le equazioni del moto sono state scritte in un sistema di riferimento non inerziale, seguendo il profilo della strada.

I dati di Input del modello sono:

- La velocità laterale del vento
- L'angolo di sterzo del pilota
- L'inclinazione del pilota
- L'apertura dell'acceleratore
- La forza sulla leva del freno

I dati suddetti sono forniti al modello dai trasduttori montati sul simulatore. Inoltre sono forniti al modello anche le caratteristiche geometriche, aerodinamiche e inerziali del veicolo, insieme alle caratteristiche delle gomme.

I dati forniti in uscita dal modello sono:

- Accelerazione, velocità e posizione longitudinale
- Accelerazione, velocità e posizione laterale
- Orientamento e velocità angolare del veicolo

Per fornire sensazioni realistiche al pilota, come profilo del terreno e percorsi in salita o in discesa, sono stati implementati altri due gradi di libertà che sono:

- La direzione verticale
- La direzione di pitch

Il modello dinamico utilizzato nel sistema è stato sviluppato presso il laboratorio PERCRO sulla base di un modello sviluppato precedentemente da uno dei partner del progetto (figura 2.9).

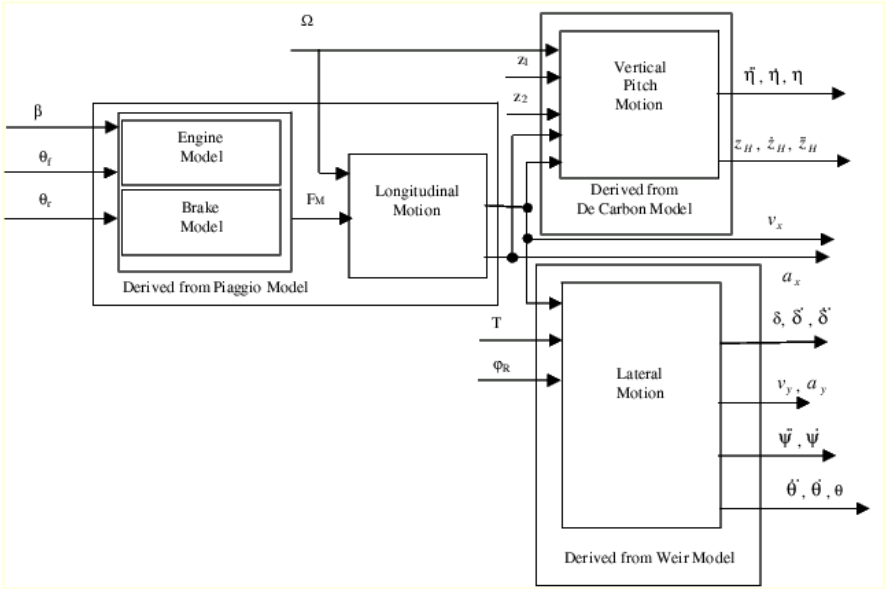

Figura 2.9. Il modello dinamico della moto

Il modello dinamico è costituito da un blocco relativo al movimento longitudinale, che elabora la dinamica del veicolo lungo la direzione longitudinale del motociclo.

L'informazione così ottenuta è trasferita a due modelli disaccoppiati che elaborano il movimento verticale (un modello con due gradi di libertà basati sulla teoria di *De Carbon*) e il movimento laterale (un modello con quattro gradi di libertà).

2.2.2 La Strategy Manager Unit

Lo scopo principale della Strategy Manager Unit è quello di riprodurre le accelerazioni lineari e angolari che il pilota avvertirebbe se la simulazione fosse reale, usando un sistema motion-based che ha però uno workspace limitato. Questo tipo di unità è nota in letteratura sotto il nome di *washout filter* ed è oggetto di studio nel campo della progettazione dei simulatori di volo.

Compito dello washout filter è quello di sincronizzare i movimenti della piattaforma con quelli richiesti alla proiezione grafica effettuata in MORIS su schermo fisso.

L'organo umano che percepisce il feedback inerziale è l'*apparato vestibolare*, costituito da due recettori (otolito e canali semicircolari) posti nelle orecchie. L'apparato vestibolare raccoglie l'informazione riguardante i movimenti della testa e li fornisce al cervello. L'apparato vestibolare è connesso con l'apparato oculare attraverso il riflesso vestibulo-oculare, che gestisce la maggior parte delle attività che richiedono entrambe i sensi, come la stabilizzazione dell'immagine e la coordinazione del movimento. L'incoerenza tra le immagini viste e il feedback inerziale avvertito possono generare sensazioni di pesante malessere.

Durante un'esperienza reale di guida, le accelerazioni hanno anche durate molto lunghe, che generano il movimento del veicolo nell'ambiente reale.

L'integrazione diretta delle accelerazioni lineari fornite dal modello dinamico non è possibile, perché richiederebbe spostamenti che non possono essere replicati da un simulatore; una possibilità per ridurre gli spostamenti necessari è quella di usare la forza di gravità per fornire un'accelerazione al pilota, ruotando la base del simulatore di un angolo di pitch addizionale.

Ovviamente questa manovra non è realistica, ma dentro un ambiente immersivo composto da grafica, feedback acustico e inerziale i sensi umani possono essere ingannati con successo. Il costo di questa operazione è che l'operatore non avverte il corretto valore dell'accelerazione di gravità lungo l'asse Z.

L'idea di base di uno washout filter è quella di simulare i termini costanti dell'accelerazione lineare fornendo rotazioni addizionali della base del simulatore.

Lo scopo dello washout filter è quindi quello di convertire le traiettorie generate dal modello dinamico della moto, che includono spostamenti molto ampi, in comandi per gli attuatori, in modo da fornire feedback realistici al pilota *rimanendo nei limiti dello workspace* del simulatore.

Uno washout filter è un sistema di controllo molto complesso, di cui devono essere garantite robustezza e stabilità al fine di non causare danni di natura meccanica al simulatore. Inoltre, uno washout filter deve tenere conto della natura non deterministica dei comandi provenienti dal pilota, e deve essere in grado di fornire stimoli *realistici* al pilota.

Nel simulatore MORIS, l'algoritmo è stato sviluppato calcolando la *washout location* (la posizione di riferimento su cui la SMU funziona) sulla testa del pilota.

Le uscite del modello dinamico della moto sono valutate sulla testa del pilota e quindi processate dallo Strategy Manager Filter (figura 2.10).

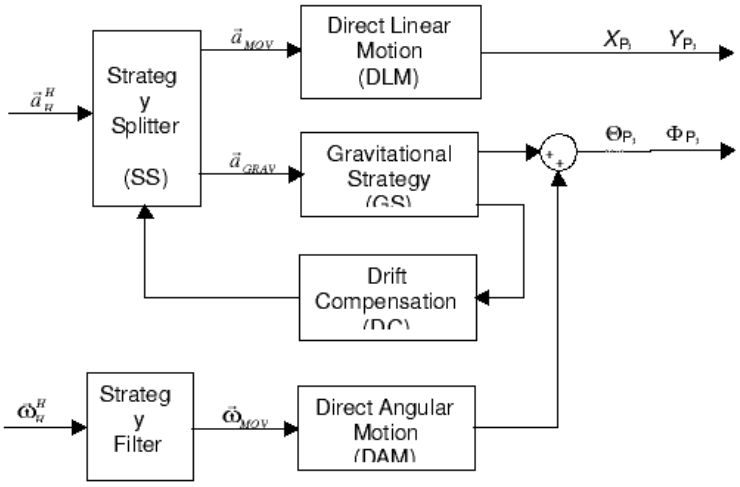

Figura 2.10. Lo schema del Washout Filter

Sulla base dei valori forniti dal modello dinamico della moto, il washout filter provvede poi a fornire al Frame Controller i riferimenti per muovere la piattaforma

di Stewart, e al sottosistema grafico i dati per aggiornare la posizione del punto di vista del pilota, e l'orizzonte del panorama.

2.2.3 Il controllo della piattaforma

Sulla base dei risultati forniti dallo washout filter, il modulo di controllo della piattaforma si occupa di aggiornare in tempo reale la posizione della piattaforma e dello sterzo.

Ha inoltre il compito di prelevare i dati dai sensori presenti sul mock-up e di fornirli al modello dinamico della moto.

Questa unità ha il controllo della parte meccanica del simulatore, pertanto deve essere in grado di reagire prontamente ad eventuali situazioni di errore interne od esterne, in modo da garantire la sicurezza del pilota. Per questo motivo è necessario che l'unità possa continuare a funzionare (ed agire in modo opportuno) anche in caso di crash degli altri sottosistemi.

L'unità di controllo gestisce varie attività, e quindi è costituita da vari moduli:

- Controllo in ciclo chiuso della posizione del telaio del veicolo nello spazio
- Controllo in ciclo chiuso della posizione angolare del manubrio
- Acquisizione dei comandi dinamici del veicolo: acceleratore, freno, inclinazione del pilota
- Acquisizione dello stato del quadro comandi: chiave, accensione, luci di posizione, fari

Poiché l'unità Strategy Manager fornisce la posizione della piattaforma, mentre il controllo agisce direttamente sui singoli pistoni, l'unità di controllo deve essere fornita di un ulteriore modulo provveda a calcolare la posizione lineare dei singoli pistoni in base alla posizione assoluta della piattaforma.

2.2.4 Il sottosistema grafico

Il sottosistema grafico ha il compito di fornire al pilota il feedback visuale dell'ambiente virtuale in cui il veicolo è immerso.

Per soddisfare i requisiti di MORIS sono usate alcune tecniche software come:

Collezione degli oggetti non visibili

Il volume dell'oggetto virtuale è confrontato con il campo visivo. Se l'oggetto è fuori dalla visuale non viene disegnato. Con questa tecnica siamo in grado di gestire scenari virtuali ampi e complessi.

Livelli di dettaglio variabili

Ogni oggetto virtuale è definito da vari livelli di dettaglio. Il livello di dettaglio è scelto in base alla distanza dall'osservatore.

Frame rate costante

Se il frame rate è troppo basso, il software disegna gli oggetti con meno dettagli, per guadagnare performance. Se invece il frame rate è troppo alto, il software aumenta i dettagli per rallentarlo. In questo modo siamo in grado di raggiungere un frame rate costante.

Automatic Strips Finder

Le sequenze di poligoni con vertici in comune sono rilevate automaticamente e disegnate, in modo da avere un trasferimento più veloce delle primitive geometriche all'acceleratore grafico.

Supporto per Multiprocessori simmetrici (SMP)

Il software è in grado di sfruttare le workstation multiprocessore per gestire scenari ampi e complessi.

Programmazione Soft-Real-Time

Tutti i processori disponibili sono usati solo dai processi che devono fornire il feedback visivo. L'unico processore libero è usato per le attività standard del sistema. Ogni processo della grafica è allocato staticamente su un processore, e le priorità dei processi sono statiche. Tutta la memoria dei processi è segnata come non swappabile.

Precalcolo dell'illuminazione

L'illuminazione degli oggetti è precalcolata offline per tutte le sorgenti di luce statiche, per non sovraccaricare la CPU e l'acceleratore grafico.

Texture Mapping

Sono supportati tutti i tipi di texture mapping per fornire un feedback visivo di qualità.

Rispetto alla data di sviluppo del simulatore ad oggi le tecnologie di generazione e controllo di ambienti virtuali sono profondamente evolute e cambiate. Alla data di sviluppo iniziale di MORIS, gli unici sistemi grafici in grado di gestire le specifiche operazioni grafiche richieste da un simulatore erano workstation grafiche di alta fascia. Attualmente invece un normale PC di fascia medio-alta è in grado di fornire grafica 3D di alto livello. La struttura imposta al sistema MORIS è ad oggi inadeguata rispetto alla situazione tecnologica. Il lavoro svolto per adeguare il sistema offre una soluzione aperta per le tecnologie attuali e future.

2.2.5 Engine di ambiente

L'engine di ambiente si occupa di fornire informazioni riguardo le caratteristiche dell'ambiente virtuale in cui il veicolo si trova immerso.

La descrizione del circuito da percorrere viene precaricata in fase di avvio dalla macchina che ospita l'applicazione Console. Il circuito è logicamente suddiviso in segmenti aventi caratteristiche diverse, e le caratteristiche del segmento in cui il veicolo si trova vengono fornite istante per istante al modello della moto, che le usa per elaborare l'uscita insieme ai dati acquisiti dal pilota. Oltre alle caratteristiche fisiche del percorso, è possibile definire altri aspetti della simulazione, come condizioni atmosferiche o vento. La descrizione dell'ambiente viene inoltrata anche al sottosistema grafico che provvede ad aggiornare il feedback visivo che viene fornito all'utente.

2.3 Problematiche di utilizzo e di manutenzione del sistema attuale

Procediamo adesso ad analizzare quali sono i principali difetti presenti nel sistema allo stato attuale. Il sistema presenta problemi sia nella fase di utilizzo, sia nella fase di manutenzione e aggiornamento.

2.3.1 Problematiche di utilizzo

La prima difficoltà incontrata nell'utilizzo del sistema MORIS sta nella complessità della fase di startup. Per attivare il sistema e renderlo operativo, occorre una precisa e lunga sequenza di operazioni, tra cui l'accensione e l'avvio delle macchine coinvolte, avvio del software sulle singole macchine, nell'ordine prestabilito, e avvio del sistema dalla Console grafica.

Specialmente nel caso delle due macchine diskless, la fase di avvio è poco intuitiva. Si tratta infatti di inserire, tramite una console testuale, una lunga linea di avvio che specifica alla macchina la serie di parametri necessari per l'avvio, come il nome del file contenente il sistema operativo da caricare, l'indirizzo nella rete locale dell'host remoto che svolge il servizio di BOOTP (assegnamento di indirizzo IP) e che fornisce il kernel del sistema operativo.

Una volta avviate le due macchine diskless, e lanciati i rispettivi software, questi si bloccano in attesa che sulla rete locale arrivi un comando di inizio simulazione.

La fase di simulazione è attivata da una funzione fornita dall'applicazione utente che gira sul sottosistema Console, mentre tramite un'altra funzione fornita da questa applicazione è possibile terminare la fase di avvio.

In pratica quindi, oltre all'utilizzatore del simulatore, è sempre necessaria la presenza di un altro operatore che comandi l'inizio e la fine della simulazione tramite la console.

Un'altro difetto riscontrato nell'architettura attuale del simulatore è una certa instabilità del software presente sulla macchina che fornisce il controllo della piattaforma. Questo è un problema critico per la gestione del simulatore, in quanto un arresto della piattaforma di controllo in certe condizioni potrebbe determinare la rottura della piattaforma.

L'unico strumento che tenta di arginare questo problema è al momento attuale la presenza di un meccanismo detto watchdog-timer: in pratica è presente un hardware che monitorizza l'attività della CPU. Nel caso in cui si accorge di una situazione di errore (tipicamente la cpu non svolge più attività) comanda un rientro in posizione base dei pistoni; purtroppo anche questo sistema non è completamente sicuro per l'integrità della piattaforma: il rientro in posizione base è comandato per tutti i pistoni con la stessa velocità, *in qualunque posizione si trovi la piattaforma;* per certe configurazioni della piattaforma quindi il rientro potrebbe essere dannoso.

Al momento attuale, non è stata trovata la causa degli sporadici arresti del sistema di controllo. Sono state fatte ipotesi circa la stabilità del sistema operativo utilizzato (VxWorks), o l'esecuzione di operazioni critiche per il sistema operativo da parte del software di basso livello sviluppato.

2.3.2 Problematiche di manutenzione del sistema

Il sottosistema incaricato di fornire il controllo della piattaforma è stato realizzato andando a modificare modo invasivo e integrato il codice prodotto in maniera semi-automatica dai tool di sviluppo utilizzati con del codice scritto a mano; se da una parte questo ha permesso di soddisfare dei requisiti del sistema, dall'altra ha reso difficile l'aggiornamento e la manutenzione dello stesso: in caso sia necessaria la modifica di un modulo prodotto con i tool di sviluppo, sarà necessario modificare anche tutto il codice aggiuntivo che interagisce con quel modulo, e probabilmente a seguito di questa modifica sarà necessario anche cambiare parte del codice che è stato aggiunto agli altri moduli. Per contro, se è necessario modificare o aggiornare parte del codice che è stato aggiunto, sarà necessario modificare anche tutti quei moduli che vanno a interagire con il sottostrato di codice aggiunto.

Con questo tipo di soluzione, modificare o aggiornare il sistema risulta pertanto faticoso e macchinoso, e aumenta pertanto la probabilità di commettere degli errori durante lo sviluppo, oltre a favorire l'utilizzo di soluzioni di ripiego per evitare di fare modifiche troppo drastiche.

Una soluzione alternativa sarebbe quella di usare dei tool di sviluppo aggiornati che riescano a soddisfare in maniera elegante e non intricata i requisiti del sistema, ma è resa difficile a causa della particolarità e della arretratezza dell'hardware e del

software utilizzato, e quindi dall'assenza di questi tool di sviluppo moderni per questi tipi di architettura. La manutenzione è resa difficile anche dall'assenza di adeguati strumenti di debug, e dalla natura del codice in esame: si tratta infatti generalmente di programmazione concorrente, e quindi è difficile riuscire a stabilire e ad analizzare il flusso di programmazione nella sua interezza. Mettendo dei check point (come ad esempio stampe su video) si rischia di alterare l'esecuzione di alcuni task, e pertanto di andare a cambiare l'ordine di esecuzione di alcuni processi rispetto ad altri.

Un'altra difficoltà operativa nella manutenzione del software sta nella mancanza di risorse della macchina che ospita i tool di sviluppo: in particolare è sensibile la lentezza di tool che utilizzano approcci grafici, e altrettanto sensibili e fastidiosi sono i lunghi tempi di compilazione di applicazioni complesse.

2.4 Interazione con altri sistemi di Realtà Virtuale

Il sistema MORIS può essere integrato con altri sistemi di realtà virtuale senza grosso carico aggiuntivo. Il sottosistema RT è infatti in grado di fornire la posizione aggiornata del veicolo all'interno dell'ambiente virtuale, la sua velocità e la sua accelerazione; possono essere fornite sia ad altri processi presenti sulla stessa macchina, sia (come nel caso della grafica) a processi residenti su macchine diverse, tramite l'interfaccia ethernet (e protocollo UDP o TCP). Sulla base di queste informazioni, si possono produrre altri elementi utili a rendere più realistica l'esperienza di guida del pilota.

Una prima possibilità è quella di fornire al pilota un feedback audio, oltre che visivo. Potrebbe essere possibile definire delle sorgenti di rumore all'interno del percorso, e per mezzo di sistemi audio stereofonici, localizzare queste sorgenti nell'ambiente virtuale. Si potrebbero inoltre fornire varie sfumature del suono del motore, sulla base della velocità e dell'accelerazione dello stesso.

In ogni modo, qualunque sia il modo per riprodurre l'ambiente virtuale in cui il veicolo è immerso (visione stereo per mezzo di occhiali, riproduzione per proiezione su schermo, ecc), il sistema utilizzato per tale riproduzione ha bisogno soltanto dell'informazione relativa alla posizione del veicolo nell'ambiente; per questo motivo, il sistema MORIS è del tutto indipendente dal sistema di realtà virtuale che gli viene

affiancato. Conseguenza diretta di questo approccio, è la possibilità di affiancare al simulatore nuove tecnologie di RV riducendo al minimo lo sforzo di adattamento.

Capitolo 3

Analisi del sistema

In questo capitolo ci occuperemo di un'analisi approfondita del sistema, e in particolare dei sottosistemi coinvolti nel lavoro di questa tesi, ovvero i moduli ospitati dalla macchina Frame Controller (FC) e dalla macchina Real Time subsystem (RT).

Sulla macchina RTS, oltre al software necessario per coordinare le attività dei vari moduli, sono ospitati anche i sottosistemi logici del modello dinamico del motociclo, e dello Strategy Manager.

3.1 Le risorse a disposizione

I due sottosistemi di cui ci andiamo ad occupare sono costituiti da macchine equipaggiate con processori Alpha a 64 bit, diskless e con sistema operativo VxWorks; in questa prima parte ci occuperemo pertanto di una descrizione generale dell'architettura utilizzata.

3.1.1 VxWorks

VxWorks, distribuito attualmente dalla Wind River Systems, è un sistema operativo real-time, progettato per essere usato in un ambiente distribuito.

La versione impiegata sul sistema è tuttora il primo (ed ultimo) supporto preliminare per architettura Alpha fornito dalla Digital Equipment Corporation (DEC) prima dell'acquisto della compagnia da parte della Compaq.

Un sistema operativo real-time è un sistema operativo in cui l'esecuzione dei task può essere prevista deterministicamente, sulla base delle conoscenze del software e dell'hardware a disposizione.

A differenza dei sistemi operativi ordinari, dove l'esecuzione è governata da uno scheduler di processi e un insieme di servizi prioritari in grado di interrompere l'esecuzione dei processi, in un sistema RT tutte le risorse di calcolo sono garantite in modo deterministico consentendo in questo modo l'esatta predizione dell'ordine di esecuzione e il relativo soddisfacimento implicito dei tempi richiesti per l'esecuzione delle procedure (deadlines). In termini di controllo, la presenza di un controllo deterministico delle operazioni software, come vedremo, consentirà di considerare affidabili e quindi stabili, le analisi tecniche di sicurezza e performance del sistema.

Nella versione configurata per il sistema MORIS, VxWorks ha bisogno di una workstation di appoggio per lo sviluppo del software: sulla macchina che ospita il sistema operativo, non è presente nessun tool di sviluppo di software come compilatori, linker o editor. L'ambiente di sviluppo è basato su cross-compilazione (compilazione del software per l'architettura che ospita VxWorks su una macchina con architettura diversa) o comunque su metodi di sviluppo remoti.

In questo sistema è necessaria una macchina di appoggio di qualche tipo per eseguire i tool di sviluppo, ad esempio il compilatore. L'applicazione compilata può essere quindi inviata alla macchina con VxWorks, e viene eseguita come parte del sistema operativo.

Vediamo alcune delle caratteristiche principali:

- VxWorks mette a disposizione dell'utente molte routines compatibili con quelle utilizzate nell'ambiente Unix nelle librerie fornite.

- Il sistema operativo non prevede commutazione da modalità protetta a modalità utente, ma viene eseguito con un'unica modalità. In questo modo le chiamate di sistema non sono implementate con interrupt software, per cui l'overhead è ridotto. Non c'è poi allocazione di risorse, per cui il tempo necessario per il cambio di contesto è ridotto.

- Nella programmazione multitasking, i processi sono tutti a memoria condivisa,

e quindi si comportano in maniera analoga a quelli che nella terminologia comune sono detti "thread". Non esistono invece i cosiddetti "task", cioè processi con spazio di indirizzamento proprio.

- Le librerie di networking e di sistema fornite sono estremamente compatibili con quelle di UNIX. È quindi possibile implementare facilmente codice basato su socket per la comunicazione interprocessore.

- Il software che viene sviluppato e caricato diventa parte integrante dell'ambiente del sistema operativo e le variabili pubbliche possono essere accedute da ogni funzione, procedura o modulo caricato separatamente. Le funzioni pubbliche possono essere chiamate da linea di comando, senza dover essere eseguite da una funzione "main()".

- Il kernel del sistema operativo viene fornito in parte già compilato, e in parte sotto forma di codice sorgente. Questo per rendere possibile la personalizzazione del front-end di sistema in modo da soddisfare le esigenze applicative.

- Non essendoci divisione tra spazio kernel e spazio utente, è possibile, in fase di compilazione del kernel, aggiungere codice da noi prodotto. Le funzioni con visibilità pubblica definite nel codice aggiunto diventano parte integrante dell'ambiente e possono essere chiamate in qualsiasi momento.

- Non è necessario linkare librerie e produrre eseguibili: semplicemente caricando il file compilato nel sistema operativo, questo avrà accesso a tutte le funzioni disponibili in quel momento.

- Il sistema operativo mette a disposizione una shell primitiva, da cui caricare il software compilato sulla workstation per lo sviluppo, eseguire funzioni e altro.

In sostanza il sistema analizzato, pur offrendo i servizi di un sistema operativo quasi completo, è per certi aspetti "embedded". nel senso che lo sviluppatore è libero di selezionare le componenti di sistema necessarie a completarle con componenti e servizi in un'applicazione chiusa.

L'interfaccia fornita da ognuna delle due macchine in esame, è costituita da una Console a caratteri, costituita da un monitor e da una tastiera, e collegata

tramite interfaccia seriale. Tramite un'opportuna combinazione di tasti, è possibile visualizzare sul monitor (e comandare con la tastiera) la macchina desiderata (figura 3.1).

Figura 3.1. Il terminale a caratteri

In fase di boot viene caricata da una memoria ROM interna la shell base delle macchine Alpha (SRM console). Questa fornisce dei comandi di base, tra cui quelli per fare caricare il sistema operativo da remoto. È possibile inoltre specificare alcuni parametri da fornire al kernel del sistema operativo in fase di avvio.

3.1.2 L'ambiente di sviluppo

Come detto in precedenza, il sistema operativo VxWorks ha bisogno di una macchina esterna su cui venga fatto lo sviluppo. Anche nel sistema MORIS è presente una macchina che fornisce questo servizio. È denominata MORISDEV (MorisDEVelopment), ed è anch'essa montata sopra un'architettura Alpha.

Sia MORISDEV che RT e FC sono dotate di interfaccia di rete Ethernet, e si trovano sulla stessa sottorete, che permette loro di comunicare.

A differenza dei sottosistemi RT e FC, su questa workstation (MORISDEV) è presente un disco fisso SCSI, e sistema operativo DEC UNIX (OSF/1).

I servizi offerti da MORISDEV sono molteplici:

- Si preoccupa di rispondere alla richieste di tipo BOOTP, per assegnare un indirizzo IP alle macchine diskless.

- Fornisce a RT e a FC le relative immagini del sistema operativo, attraverso un protocollo TFTP (trivial FTP, una variante del protocollo FTP semplificata, non richiede autenticazione).

- Esporta parte del proprio file system e lo mette a disposizione delle macchine diskless tramite servizio NFS.

- Fornisce strumenti di sviluppo che come già detto sono assenti in ambiente VxWorks. In particolare fornisce il compilatore di linguaggio C, gli header delle librerie di VxWorks, in modo da rendere possibile la compilazione, oltre ad altri strumenti comuni all'ambiente Unix come l'editor di testo VI o il tool Make.

- Ospita un'altro tool di sviluppo, ovvero MathWorks Matlab, che è stato usato per sviluppare parte del sistema di controllo della piattaforma.

Una volta compilato il software scritto, viene prodotto un oggetto di tipo *loadable object* (l'equivalente di un eseguibile in ambiente Unix). In pratica questo oggetto può essere importato nel sistema operativo dopo la fase di boot, tramite la direttiva load (ld) che può essere chiamata da shell, oppure può venire aggiunto nell'immagine del kernel e quindi caricato ad ogni avvio del sistema. In entrambi i casi, come già detto, gli oggetti a visibilità globale (funzioni e variabili, ad esempio) possono essere acceduti dall'esterno (linea di comando, oppure altre funzioni).

Tra i vantaggi offerti da un sistema di questo genere, c'è la capacità di procedere alla fase di codifica degli schemi di controllo del sistema con una "minima" fase manuale, che limita quindi la probabilità di commettere errori. Nella versione considerata, come sarà meglio illustrato in seguito, le funzionalità offerte dal sistema di sviluppo erano alquanto limitate e hanno richiesto un grosso sforzo manuale per integrare negli schemi di controllo le procedure asincrone che gestiscono le fasi del sistema.

3.1.3 Matlab, Simulink e Real Time Workshop

Matlab/Simulink è un modulo applicativo per la simulazione di sistemi dinamici. Mette a disposizione dell'utente un ambiente grafico per la costruzione del sistema dinamico come composizione di blocchi. A questo scopo fornisce delle librerie di blocchi standard. Simulink da poi la possibilità all'utente di creare nuovi blocchi che potranno poi essere inseriti nel diagramma che rappresenta il modello. Questa funzionalità è fornita tramite il meccanismo delle S-Functions, che permette di incorporare codice C scritto dall'utente in uno schema simulink come se fosse un blocco.

Il modulo applicativo Real-Time Workshop collegato al programma Matlab/Simulink è un ambiente di prototipazione rapida di applicazioni di controllo in tempo reale. Si basa sulle funzionalità di Simulink e permette la generazione automatica di codice sorgente che implementa applicazioni di controllo a partire dalla loro progettazione grafica Simulink e dal sistema operativo residente sul calcolatore obiettivo. Il codice generato costituisce un'applicazione in tempo reale completa, che può essere eseguita su un calcolatore obiettivo dotato delle risorse strettamente necessarie.

3.1.4 La console

Oltre alle tre macchine Alpha descritte, il sistema MORIS è costituito anche da un PC classico equipaggiato con distribuzione Linux, anch'esso collegato alla rete locale ethernet.

Lo scopo di questo elaboratore è quello di fornire un'interfaccia grafica per l'interazione tra un operatore (esterno al veicolo) e il sistema console. In particolare una volta lanciate le applicazioni su FC ed RT, è da questa macchina che partono i segnali di inizio e fine simulazione, e che vengono trasferiti sugli altri sistemi dei parametri come la descrizione della geometria del circuito e descrizione della moto.

Sull'interfaccia grafica è inoltre possibile visualizzare in tempo reale alcuni parametri della simulazione (figura 3.2).

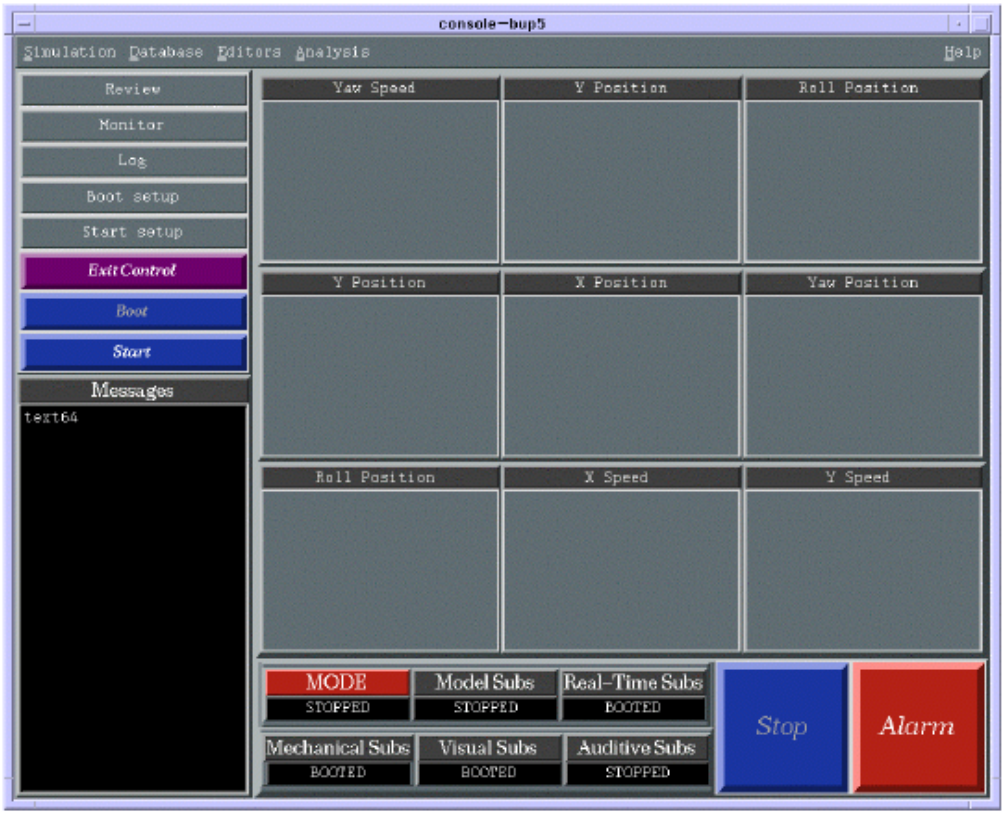

Figura 3.2. La console grafica

3.1.5 Il processore Alpha

Alpha è il nome dato alla architettura RISC a 64 bit della DEC (Digital Equipment Corporation). Oltre ad essere stato il primo processore del mondo a 64 bit naturali, è stato per un lungo periodo di tempo il processore in grado di operare alla frequenza più alta.

Le workstation FC e RT sono equipaggiate entrambe con processori di questo tipo; questa scelta è stata fatta in seguito alle aspettative collegate al sistema in sviluppo nel 1996.

Scelte politiche e commerciali hanno poi penalizzato questo tipo di architettura, al punto che ad oggi la linea di processori Alpha è fuori commercio.

Durante la fase iniziale del lavoro descritto, si è cercato di trovare una soluzione

alternativa all'architettura esistente. L'analisi ha interessato quali interventi Hardware e Software potessero essere effettuati sul sistema considerando anche i vincoli economici e strutturali legati agli investimenti. Si è cercato di sostituire le componenti di sistema dove l'intervento hardware non fosse eccessivo e in modo che la compatibilità software potesse essere garantita. Una prima fase è stata quella di cercare un sistema operativo alternativo a VxWorks, che pero' ne possedesse i requisiti Real-Time, e che potesse essere fatto girare su macchine prive di disco fisso. Altro requisito del sistema è quello di mantenere l'interfaccia FDDI per garantire la robustezza e la velocita' della comunicazione tra i due sottosistemi.

Durante la prima fase dell'analisi, si è cercato un sistema operativo che offrisse la possibilità di caricare l'immagine del sistema operativo da kernel e che supportasse il processore Alpha; a causa dell'uscita di produzione delle macchine in questione, e della loro scarsa diffusione, non è stato possibile trovare molte alternative. In particolare, l'attenzione è stata rivolta alle varianti real-time del sistema operativo Linux.

Dopo aver verificato la capacità di Linux di supportare boot remoto, e la presenza di versioni precompilate per processori Alpha, si è proceduto all'analisi delle varianti RealTime di Linux, ovvero RTLinux ed RTAI. Ad oggi, entrambe le versioni non supportano in modo soddisfacente il processore Alpha.

Parallelamente si è verificata la possibilità di supporto del bus VME da parte di Linux. Anche in questo caso la risposta è stata negativa, poiché il tipo di bridge PCI-VME montato sulla macchina FC non è tra quelli per cui sono stati sviluppati i driver dal progetto Linux-VME.

Dopo questa prima analisi, è risultato troppo oneroso fornire un'alternativa a VxWorks per le macchine RT e FC.

3.2 Lo studio del software esistente

Le applicazioni presenti sui due sottosistemi RT ed FC sono state realizzate utilizzando gli strumenti di sviluppo descritti precedentemente. Al fine di comprendere pienamente il funzionamento del sistema, è stato necessario lo studio del sistema operativo VxWorks, delle possibilità che offre e del significato delle API fornite dalle

librerie. Sebbene per la parte generale esse siano simili alle chiamate UNIX, le chiamate non seguono nessuno standard per quanto riguarda la creazione e l'interazione tra i processi (il sistema non è POSIX-compliant).

Oltre allo studio del sistema operativo VxWorks, è stato necessario anche lo studio del tool di sviluppo Matlab e del suo modulo Simulink, con cui è stata sviluppato parte del software che gestisce il controllo della piattaforma.

3.2.1 Analisi del sottosistema RT

Come descritto in precedenza, il sottosistema RT ha il compito di coordinare le attività di tutti i sottomoduli del sistema completo; inoltre, nell'ultima versione del software, esso ospita anche i moduli che implementano lo Strategy Manager (washout filter) e il modello dinamico della moto.

I moduli da gestire sono sia esterni come:

- Console, a cui invia e da cui riceve le informazioni di sistema e i comandi di controllo dello stato di simulazione (scenario, azioni, interazioni).
- Frame Controller, a cui trasmette le informazioni di controllo per lo sterzo e per la piattaforma, e da cui riceve i comandi di guida del pilota.
- Grafica, a cui invia la posizione della moto nell'ambiente virtuale.

che interni, come:

- Modello dinamico della moto, in grado di integrare le equazioni dinamiche della moto,
- Strategy Manager, in grado di trasformare le informazioni inerziali di movimento in traiettorie per la piattaforma di Stewart,
- Terrain Manager, in grado di determinare le caratteristiche del segmento del circuito virtuale in cui il veicolo si trova,
- Weather Manager, che determina la situazione climatica dell'ambiente virtuale,

- Dynamic Object Manager (DO Manager), che gestisce la presenza di elementi attivi nell'ambiente, come altri veicoli.

Tutti i moduli implementati su RT, e in particolare il modello dinamico della moto e lo strategy manager sono stati sviluppati per funzionare in ciclo chiuso e con frequenza di 100 Hz; L'esecuzione del sistema a regime è asincrona e comandata dalla ricezione da parte del modulo Frame Controller dei dati acquisiti dai sensori presenti sulla moto ogni 10 millisecondi. Il Frame Controller è stato progettato per fornire alla frequenza desiderata i dati sull'interfaccia FDDI. È particolarmente importante osservare come le prestazioni richieste al sistema RT sono di un ordine di grandezza inferiore rispetto a quelle offerte da FC (100 Hz contro 1 KHz). Ciò avrà un particolare rilievo nella scelta e nella verifica di una architettura alternativa.

In fase di avvio, il sottosistema RT provvede a caricare dalla Console una serie di dati utili alla simulazione, tra cui la descrizione della moto e la descrizione del circuito virtuale in cui sarà immerso il veicolo. Il circuito è idealmente diviso in segmenti, ognuno dotato di proprietà come inclinazione e lunghezza. A seconda della posizione della moto, è possibile stabilire in quale segmento si trova.

Vediamo adesso una descrizione a livello qualitativo dell'implementazione dei moduli interni al sottosistema RT.

3.2.1.1 Motorcycle Dynamic Model

Il modello dinamico della moto è stato sviluppato con il tool grafico Simulink e con il tool Real Time Workshop. Il real time workshop consente di generare codice per eseguire la simulazione dei modelli sviluppati in modo indipendente da simulink . Poiché nel caso del modello dinamico della moto non si era interessati ad un'esecuzione indipendente del modello, ma alla sua interazione con altri moduli esterni, è stata generata un'interfaccia che permette di eseguire un solo passo di simulazione per volta, fornendo al modello gli ingressi adeguati provenienti dalla lettura dei sensori, e che permette di leggerne i dati di uscita.

A livello implementativo quindi, si tratta di fornire al modello gli ingressi in una struttura allocata staticamente, chiamare la funzione di interfaccia, e quindi andare a leggere l'uscita in un'altra struttura preallocata.

I dati di ingresso necessari al modello sono i valori letti dell'acceleratore, dei freni anteriore e posteriore, dell'inclinazione del pilota sulla moto e della torsione dello sterzo; oltre a questi dati che arrivano dal frame controller attraverso l'interfaccia fddi, il modello accetta in ingresso anche gli angoli di inclinazione del terreno lungo l'asse X e Y, e la posizione lungo l'asse Z della ruota posteriore del veicolo (che vengono forniti dal modulo Terrain Manager).

I dati forniti in uscita sono invece sono le posizioni, le velocità e le accelerazioni lungo gli assi X,Y e Z, le posizioni, velocità angolari e accelerazioni angolari lungo gli angoli di roll, pitch e yaw, e la posizione angolare dello sterzo.

Il modello dinamico genera le informazioni di movimento della moto in maniera dinamica e interattiva. Lavora in ciclo chiuso sulla base dei parametri interni del motociclo. È particolarmente importante che l'esecuzione del modello dinamico avvenga in tempo reale alla stessa frequenza con cui è stato sviluppato il modello. Ogni alterazione della frequenza di simulazione rischia non solo di alterare la qualità del feedback, ma anche di compromettere la stabilità.

3.2.1.2 Strategy Manager

L'implementazione del modulo strategy manager è la stessa del modulo Dynamic Model, e cioè è stato modificato il codice generato in maniera automatica da RTW, in modo da ottenere l'interfaccia per eseguire un singolo passo della simulazione.

I dati richiesti in ingresso sono i dati letti dai sensori della moto (ricevuti via FDDI dal modulo FC), alcuni dei dati prodotti dal modello dinamico della moto, mentre in uscita produce i riferimenti per la nuova posizione della piattaforma (Coordinate X,Y,Z e angoli di Roll, Pitch e Yaw). Oltre ai dati per la piattaforma, inviati al modulo FC attraverso l'interfaccia FDDI, lo strategy manager fornisce anche la posizione presunta della testa del pilota, e un vettore di controllo della terna grafica.

Come per il Dynamic Model, l'esecuzione dello Strategy Manager in tempo reale è particolarmente importante affinché la piattaforma sia mossa in modo conforme alle traiettorie valutate e quindi che il pilota percepisca le accelerazioni stimate dal controllo.

3.2.1.3 Le componenti di ambiente virtuale

Terrain Manager

Il terrain manager provvede a verificare che il veicolo si trovi effettivamente all'interno del circuito virtuale, e che non ne esca fuori.

Sulla base della posizione del veicolo nell'ambiente virtuale, è poi in grado di determinare i segmenti del circuito coinvolti nella simulazione (quello che contiene la ruota anteriore, e quello che contiene la posteriore).

Sulla base dei segmenti su cui si trovano le due ruote del veicolo, è in grado di determinare l'inclinazione della strada lungo le due direzioni X e Y (parametri necessari in ingresso al modello dinamico della moto) e l'altezza delle due ruote (la posizione lungo l'asse Z della moto è un altro parametro necessario al modello).

Wheater Manager

Tra i parametri definibili per la simulazione, ci sono anche le condizioni atmosferiche dell'ambiente virtuale. Da console è possibile associare a momenti diversi della simulazione delle condizioni climatiche diverse.

Il wheater manager non fa altro che verificare se il passo attuale di simulazione prevede cambiamenti delle condizioni atmosferiche, e in caso positivo attuare questi cambiamenti.

Dynamic object manager

Il dynamic object manager dovrebbe occuparsi degli oggetti dinamici dell'ambiente virtuale, che possono interagire con il veicolo. In realtà questa funzione non è stata implementata nel sistema MORIS, e questo modulo non svolge nessuna funzione.

3.2.1.4 La gestione delle interazioni

Vediamo adesso a livello qualitativo come è stata implementata la gestione delle interazioni tra i moduli del sistema all'interno di RT.

Il funzionamento di RT può essere diviso in due fasi, ovvero una fase di preparazione e avvio del sistema, e un'altra fase in cui il sistema è in fase di simulazione e quindi tutti i moduli sono attivi.

La fase di avvio: Durante la fase di avvio vengono testate le componenti Hardware di sistema, e quindi create ed aperte le connessioni TCP sulla sottorete ethernet con la Console e con il modulo FC. Viene inoltre predisposto il modulo per la comunicazione sulla sottorete FDDI, e infine vengono inizializzate le strutture dati necessarie al funzionamento del sottosistema (strutture di scambio messaggi, e strutture di sincronizzazione tra processi).

La prima azione svolta durante la fase di avvio è la sincronizzazione con il modulo Console, e la ricezione dei dati utili alla simulazione, tra cui la descrizione del circuito. Il protocollo di avvio prevede lo scambio di messaggi di avvenuta ricezione dei dati.

In seguito, RT provvede a creare i processi necessari per svolgere le funzioni richieste dalla fase di simulazione, che si bloccano appena svegliati.

Una volta ricevuto il messaggio di inizio simulazione, RT provvede a propagarlo agli altri moduli interessati, e a svegliare i task per dare avvio alla fase di simulazione.

Dal punto di vista implementativo, la fase di avvio è divisa in sottofasi distinte, definite da procedure differenti. Il concludersi di una sottofase determina, in caso di successo, l'inizio della sottofase successiva.

La fase di simulazione: È in questa fase che il sottosistema svolge le sue funzioni, e che sono attivi i processi che sono stati creati nella fase di avvio. Questi processi sono stati creati con priorità differenti quindi, sebbene vengano risvegliati tutti insieme, l'ordine di esecuzione è deterministico e predeterminato. In ognuno di questi processi è presente inoltre un meccanismo di controllo del corretto ordine di esecuzione; in caso di ordine errato viene sollevata un'eccezione.

L'esecuzione è ciclica. Ogni processo svolge il suo compito e poi si riaddormenta, in attesa del suo nuovo turno.

L'inizio di un nuovo ciclo è determinato dal processo con priorità più alta che si blocca in attesa che arrivino dati da FC. A seguito dell'avvenuta ricezione, il sistema svolge le seguenti azioni:

1. Tra i dati provenienti dalla lettura dei sensori, c'è lo stato della chiave del quadro comandi. In caso di chiave in posizione ON, il sistema procede.

2. Viene interrogato il modulo Wheater Manager, per determinare se all'istante corrente è necessario cambiare le condizioni atmosferiche dell'ambiente virtuale.

3. I dati provenienti da FC vengono passati al modello dinamico della moto, che esegue un passo della simulazione.

4. Viene eseguito il gestore degli oggetti dinamici, che ad oggi è costituito da una chiamata vuota.

5. Con il risultato del modello dinamico e i dati letti dai sensori viene interrogato il modulo Strategy Manager che fornisce la nuova posizione della piattaforma, e i nuovi riferimenti per la grafica.

6. Il risultato fornito dal modulo Strategy Manager viene inviato attraverso FDDI al modulo Frame Controller. Inoltre i dati utili alla grafica e alla Console vengono inviati in broadcast sulla sottorete Ethernet.

7. Il ciclo si chiude, e si attende un nuovo messaggio su FDDI.

Oltre a questi task, è presente anche un task che viene svegliato in caso di errore, e che provvede a mandare in broadcast un messaggio di allarme.

Vale la pena di evidenziare come MORIS, già nella prima versione, preveda una frequenza di controllo del frame di 100 Hz. Questa frequenza è circa il doppio delle frequenze di funzionamento dei simulatori esistenti ad oggi, ma è fondamentale per la rappresentazione di dinamiche elevate quali il tipo di strada (pavimentazione) e la dinamica laterale (ad alta banda) della moto.

Nella figura 3.3 è evidenziato come il sottosistema RT abbia a disposizione 10 millisecondi per compiere le azioni richieste, mentre il sottosistema FC deve essere in grado di rispondere ogni millisecondo.

3.2.1.5 Il modulo di comunicazione

Per rendere più veloce la comunicazione durante la fase di simulazione, i driver dell'interfaccia FDDI sono stati modificati: i dati scambiati non viaggiano sul livello

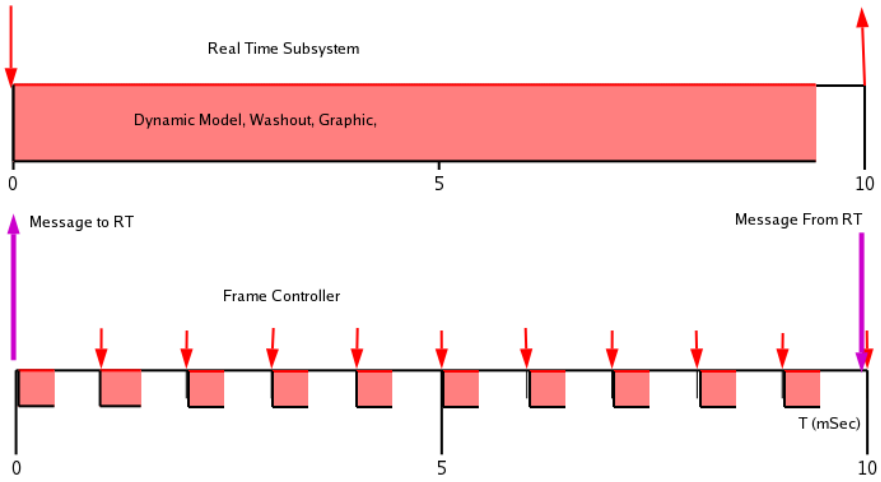

Figura 3.3. Le temporizzazioni di Rt e Fc a confronto

network, ma costituiscono il payload del pacchetto del livello *datalink*. Le interfacce quindi non vengono riferite tramite l'IP address, ma tramite il MAC address.

Questo è stato possibile grazie alle proprietà di personalizzabilità di VxWorks citate in precedenza.

Per verificare il corretto scambio dei messaggi, tra i dati scambiati è presente un contatore che viene incrementato ad ogni trasmissione. Questo fornisce un meccanismo di controllo dell'errore, e della operatività del canale di trasmissione.

3.2.1.6 Aspetti da migliorare nel sottosistema RT

Architettura obsoleta: Le prestazioni offerte dalla macchina che ospita il sottosistema RT sono di gran lunga inferiori a quelle che possono essere offerte da un PC di ultima generazione. È auspicabile quindi la sostituzione di questa macchina con una più moderna.

Sviluppo Off-Line: Il bisogno di una macchina esterna che ospita i tool di sviluppo è, come descritto in precedenza, una delle caratteristiche di VxWorks. Questo rende più lungo il processo di verifica del codice, e più difficile la procedura di debug, che va svolta da remoto.

Non linearità del codice: Sebbene nelle operazioni svolte dal sistema si possa individuare un flusso logico lineare, queste vengono svolte da processi distinti che vengono lanciati insieme (ma eseguiti a tempi diversi); questo è un ostacolo alla leggibilità e alla facilità di manutenzione del codice.

Comunicazione via FDDI a livello Datalink: La scelta di non utilizzare le normali procedure per comunicare a livello IP (socket, comunicazione TCP e UDP) sull'interfaccia FDDI, limita la portabilità del codice, rendendo il sistema inutilizzabile in presenza di altre interfacce di comunicazione. Per contro, l'utilizzo delle chiamate classiche renderebbe il sistema indipendente dal tipo di interfaccia di comunicazione utilizzata.

Dipendenza dalla Console: A causa della limitatezza di funzioni grafiche offerte da VxWorks, non è possibile implementare su RT un valido editor di circuiti, che al momento attuale è presente sulla console, né ospitare un database di circuiti e di descrizioni di veicoli. Per questo motivo, in fase di avvio il sottosistema è costretto a farsi fornire dall'esterno i parametri necessari alla simulazione.

Macchina Diskless: Poiché la macchina non è dotata di disco fisso, è necessario in fase di boot specificare i parametri per l'avvio da rete. La macchina ha quindi bisogno di una macchina esterna che gli fornisca l'immagine del sistema operativo e che gli metta a disposizione un file system distribuito sulla rete.

La velocita' di accesso al file system è quindi limitata dal mezzo di trasmissione (la rete).

Inoltre il funzionamento di questa macchina è subordinato al funzionamento dell'altra macchina che fornisce questi servizi.

3.2.2 Analisi del sottosistema FC

Generalmente le applicazioni in tempo reale possono essere catalogate in applicazioni *clock based* (le elaborazioni del sistema sono cicliche e devono essere eseguite in un periodo costante) o applicazioni *event based* (le elaborazioni del sistema devono essere eseguite in risposta ad eventi non predicibili, entro un tempo massimo). Nel

caso del sistema FC, si ha l'esigenza di presentare nello svolgimento delle attività che lo costituiscono, comportamenti temporali diversi:

Le attività *clock based* sono quelle che interagiscono direttamente con l'impianto e data la loro criticità devono essere progettate fornendo garanzie sul periodo di esecuzione e sulla durata di esecuzione. Il loro compito è quello di far tenere all'impianto il comportamento desiderato.

Le attività *clock based* sono inoltre controllate costantemente dal sistema in maniera che le deadline vengano rispettate. Un segnalatore specifico è in grado di rilevare il non rispetto di una deadline (miss) e di segnalarla a livelli di controllo superiore.

La gestione degli errori, l'interazione con altri sottosistemi e la coordinazione dei vari moduli invece sono eseguite da attività *event based*, in cui la risposta ad un certo evento deve avvenire entro un tempo massimo.

Il sottosistema FC si occupa di gestire il controllo della piattaforma di Stewart. Il software è stato sviluppato con l'intento di rendere questo sottosistema in grado essere indipendente da eventuali malfunzionamenti del sottosistema RT.

Per facilitare lo sviluppo del sistema, è stato seguito un approccio *modulare*: il sistema è stato suddiviso in vari sottosistemi che offrono e richiedono servizi (ingressi e uscite).

Per rendere il sistema pronto a rispondere eventuali situazioni di errore, critiche perchè potenzialmente dannose per l'utente e per la piattaforma, i moduli implementati sono eseguiti con una frequenza di 1 KHz. Questo significa che le interazioni con il sottosistema RT, che abbiamo detto funzionare a 100 Hz, non devono essere gestite ad ogni passo della simulazione, ma ogni 10 passi.

Procediamo adesso ad analizzare i moduli che gestiscono il sistema meccanico:

3.2.2.1 Il controllo della piattaforma

L'applicazione che gestisce il controllo della piattaforma è stata sviluppata per mezzo del tool Matlab/Simulink, concettualmente per il sistema di controllo. La piattaforma può trovarsi in tre diverse situazioni di utilizzo: nella posizione *base* gli attuatori sono completamente chiusi; è la posizione in cui si trova la piattaforma prima di attivare il sistema, e in cui la piattaforma si deve trovare prima di terminare la

simulazione. Nella posizione *centrale* l'estensione degli attuatori è a metà corsa (è la posizione in cui il sistema si trova prima di effettuare la simulazione vera e propria). Nella fase di *simulazione* infine, la posizione degli attuatori è definita dai dati provenienti dallo washout filter.

Nella prima fase (chiusa) la piattaforma è in uno stato sicuro, nel senso che il controllo può essere interrotto senza rischi per la struttura (stabilità meccanica).

Nella seconda fase, il controllo mantiene la piattaforma in posizione e si cura della sua eventuale chiusura. In fase di simulazione, il controllo impone alla piattaforma la traiettoria fornita dal Washout Filter, controllando l'integrità dei dati, il workspace della piattaforma e la continuità di posizione nelle transizioni di fase.

Il controllo ad una frequenza elevata consente in linea di principio anche il controllo attivo "MIMO" (multiple in/out) delle relazioni tra le dinamiche dei pistoni, prevenendo in tal modo le interferenze dinamiche del modello, presenti in caso di controllo "diagonale" asse per asse.

Di seguito si discuteranno quindi le componenti presenti in FC.

La cinematica inversa

Il modulo *Washout Filter* fornisce la posizione e l'orientamento che la piattaforma deve avere nello spazio, mentre nel sottosistema FC si vanno a pilotare direttamente le estensioni dei pistoni.

Poichè la corrispondenza tra la posizione della piattaforma e la lunghezza degli attuatori è biunivoca, dai dati forniti dallo washout è possibile calcolare la configurazione richiesta per i pistoni. Il modulo che si occupa di fare questo è quello della cinematica inversa.

Questo modulo è stato implementato con il meccanismo delle S-Function, e oltre a fornire l'estensione dei pistoni, si preoccupa di verificare che la configurazione richiesta non sia dannosa per la piattaforma, e cioè che non si verifichino intersezioni tra gli attuatori. Nel caso in cui il modulo si accorga di una potenziale situazione di errore, provvede a generare un messaggio di allarme.

Il controllo dei pistoni

Il modulo che implementa il controllo dei pistoni è descritto in figura 3.4.

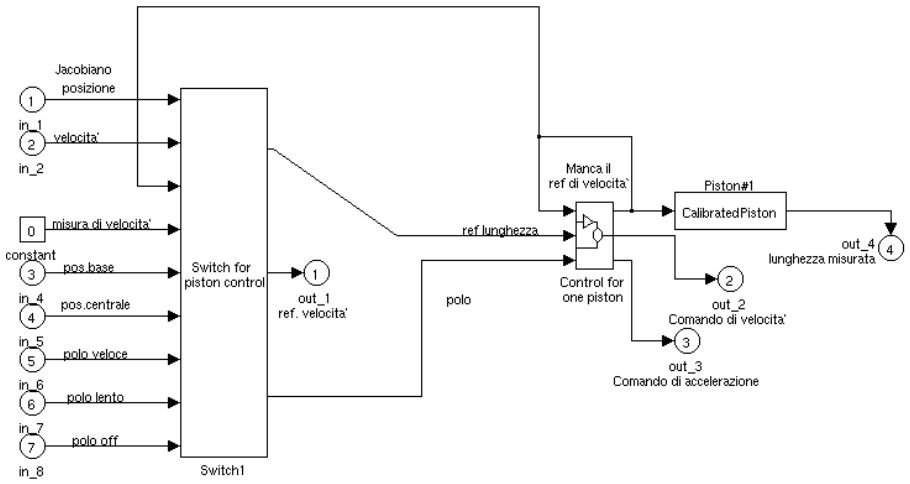

Figura 3.4. Il controllo dei pistoni

In fase di simulazione, l'uscita generata dalla cinematica inversa costituisce il riferimento di posizione che i pistoni devono inseguire.

Vogliamo però essere in grado di comandare per ogni pistone il raggiungimento delle posizioni base e centrale.

Attualmente i sei pistoni sono controllati in ciclo chiuso fornendo un riferimento di posizione.

La soluzione implementata è quella di intercettare (per ogni pistone) il riferimento di posizione fornito dal modulo che implementa la cinematica inversa e fornirlo in ingresso ad un servocontrollo PID solo in fase di simulazione. Nelle altre fasi in cui si deve trovare la piattaforma, i valori che le lunghezze dei pistoni devono assumere sono note e costanti, e generate internamente in funzione della postura richiesta. In pratica, si antepone al controllore del pistone un altro modulo detto *piston switch*, sempre implementato come S-Function, che realizza uno switch intelligente. Il modulo ha in ingresso i valori costanti di posizione e velocità che il pistone deve assumere negli stati noti. Un'ulteriore uscita del piston switch costituisce il polo della funzione di trasferimento del controllore del pistone. Come si nota nello schema di figura 3.4 si può, tramite questa, comandare la velocità di inseguimento

della posizione di riferimento fornita.

Le configurazioni dei sei piston switch (e quindi gli ingressi dei controllori dei pistoni) sono sei:

Stato dello switch	Riferimento di posizione	Riferimento di velocità	Polo
OFF	Pos. Base	0	Lento
STARTUP	Pos. Centrale	0	Lento
PRESIM	Pos. Centrale	0	Veloce
SIM	Riferimento da Cin. Inv	Riferimento da Cin. Inv	Veloce
STOP	Pos. Centrale	0	Lento

Tabella 3.2. Le configurazioni del piston switch

La transizione dello switch da una configurazione all'altra è determinata dallo stato generale in cui si trova il sistema. Le possibili transizioni sono rappresentate in figura 3.5.

Il controllo del pistone lavora come un servoregolatore, la cui banda può essere controllata da un polo esterno definito tra gli output dello switch (vedi tabella 3.2). La banda aggiunta limita la responsività della piattaforma, ed è utilizzata durante le fasi in cui le transizioni richieste devono essere morbide. L'aggiunta del polo del controllore in uscita infatti è voluta perchè si desidera una commutazione lenta tra la posizione di base e quella centrale (fase di avvio), ma si vuole che il sistema sia reattivo in fase di simulazione (polo veloce), a causa della rapidità di movimento richiesta al simulatore di motociclo.

La sostituzione del polo lento con quello veloce è fatto nella transizione dalla configurazione di startup alla configurazione di presimulazione: la posizione è la stessa, ma il polo del controllore cambia. Questo per far si che l'errore tra la posizione comandata al pistone, e la posizione richiesta sia nullo e il cambio di polo sia possibile. La posizione assunta dal pistone, misurata dal sensore lvdt montato sullo stesso viene fornita in ingresso al modulo piston switch, al fine di verificare che il pistone abbia assunto la posizione richiesta entro un intervallo di tempo determinato. Se questo non accade, il modulo notifica una situazione di errore.[1]

[1]La gestione delle condizioni di errore (hw, elettrico, meccanico e software) è un elemento fondamentale nello sviluppo di FC, a causa della potenza elevata della piattaforma (spinge fino a 80 KN), e le condizioni di sicurezza sono essenziali.

Il modulo piston switch ha inoltre la capacità di aprire e chiudere il relay che abilita il pilotaggio dei pistoni, e in caso di situazione di errore ne comanda l'apertura.

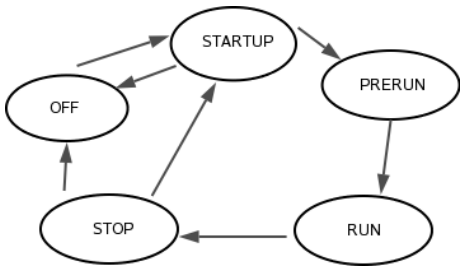

Figura 3.5. Le possibili transizioni del *piston switch*

Il ciclo di controllo

Il riferimento della posizione desiderata per il pistone viene fornito in ingresso al sistema di controllo dell'estensione del pistone, che è un sistema passabasso del secondo ordine, la cui banda può essere cambiata in tempo reale. Il ciclo di controllo assicura la continuità della posizione e della velocità, anche in presenza di discontinuità in ingresso, e di variazioni della frequenza. La frequenza viene fornita in ingresso dal piston switch.

Infine, come si può vedere nella figura 3.4, la posizione che il pistone deve assumere viene fornita ad un altro modulo, sempre implementato come S-Function, che interagisce con i driver delle schede di IO, e che comanda l'estensione effettiva del pistone.

L'ingresso del modulo è la posizione desiderata per il pistone, e l'uscita è la posizione misurata dal sensore LVDT relativo al pistone. Il modulo si occupa di effettuare la conversione dalla posizione desiderata alla tensione che l'uscita analogica della scheda di IO relativa al pistone deve avere. Inoltre, sempre interagendo con i driver, il modulo riporta la posizione assunta dal pistone in ogni istante. Questa viene utilizzata per chiudere il ciclo di controllo, e dal Piston Switch per verificare eventuali situazioni di errore.

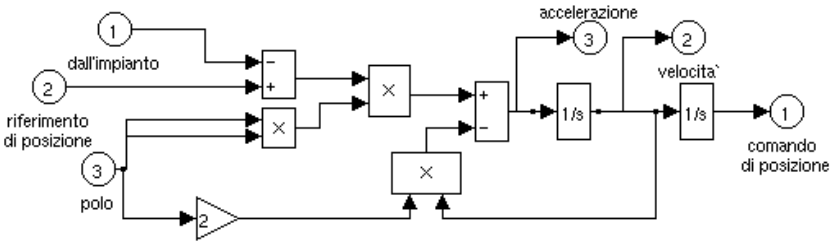

Figura 3.6. Il ciclo di controllo del pistone

3.2.2.2 Il controllo dello sterzo

In maniera analoga alla posizione della piattaforma, la posizione desiderata per lo sterzo è fornita dal Dynamic Model, e viene fornita al sottosistema FC dal sottosistema RT. Questa viene fornita al modulo di controllo che pilota in ciclo chiuso il motore brushless collegato allo sterzo. Il modulo è rappresentato nel sottosistema PID rappresentato in figura 3.7, i cui parametri sono stati regolati sulla base delle caratteristiche meccaniche dello sterzo del mock up.

L'interazione con il motore dello sterzo è realizzata con il blocco SteerSfun, che utilizza i driver delle schede di IO. Il modulo, realizzato con una S-Function, pilota l'uscita relativa al motore dello sterzo con il valore di tensione desiderato, e fornisce in uscita il valore di rotazione letto.

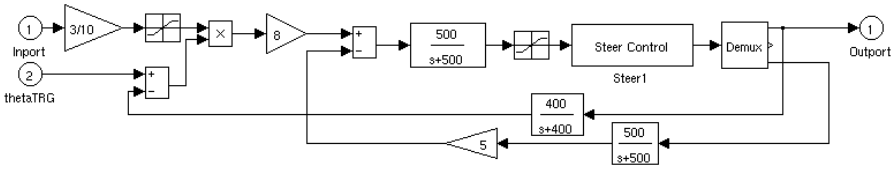

Figura 3.7. Il controllo dello sterzo

Inoltre, il modulo è in grado di leggere un'ingresso digitale pilotato direttamente dal motore, che segnala eventuali condizioni di malfunzionamento, e può pilotare un

relay che abilita o meno il funzionamento del motore dello sterzo. In situazioni di errore, il relay viene aperto, e un messaggio di errore viene notificato all'esterno.

3.2.2.3 L'interazione con il sottosistema meccanico

Il sistema FC interagisce con il sistema meccanico, leggendo le posizioni dei pistoni della piattaforma di Stewart, e una serie di parametri dal mock up della moto: la rotazione della manopola dell'acceleratore, la pressione dei freni, l'inclinazione del pilota, la torsione dello sterzo, oltre alla pressione dei tasti e della chiave di accensione.

Oltre a leggere dati, il sistema è in grado di comandare le posizioni dei sei pistoni, e dello sterzo.

Le schede di acquisizione

Il sottosistema FC è dotato di due schede di IO TVM745, montate sul bus VME. Queste schede sono dotate di:

- 4 uscite digitali di tipo relay
- 12 ingressi digitali
- 4 uscite analogiche, pilotate da altrettanti convertitori DAC
- 4 ingressi analogici, che pilotano convertitori ADC di tipo differenziale
- 4 ingressi di encoder

I driver per pilotare queste schede sul sistema VxWorks sono stati interamente sviluppati nel corso del progetto MORIS.

L'interfaccia fornita ad alto livello dai driver è costituita da delle variabili a visibilità globale, che costituiscono i puntatori ai registri delle schede dove è possibile scrivere (uscite, digitali o analogiche) o leggere (ingressi).

I nomi di questi puntatori sono tali da indicare quale componente del simulatore è collegato all'ingresso o all'uscita relativa.

Nelle tabelle 3.3 e 3.4 sono indicati gli ingressi e le uscite del sottosistema verso l'impianto.

Ingresso	Segnale (A/D)
6 misure di posizione dei pistoni	Analogico
Angolo dello sterzo	Encoder
Acceleratore	Encoder
Freno anteriore	Encoder
Freno posteriore	Encoder
Marcia	Analogico
Inclinazione del pilota	Encoder
Clacson	On/Off
Starter	On/Off
Fari	On/Off
Accensione	On/Off
Luci di posizione	On/Off
Faults	On/Off (n bit)

Tabella 3.3. Gli ingressi del sistema FC

Uscita	Segnale(A/D)
Comandi per l'attuazione dei piston (6)	Analogico
Comando per l'attuazione della coppia dello sterzo(1)	Analogico
Comando per il sottosistema di replica delle vibrazioni (1)	Analogico
Comando per il contachilometri (1)	Digitale
Comando per il tachimetro (1)	Digitale
Comando per il contagiri (1)	Digitale

Tabella 3.4. Le uscite del sistema FC

Oltre a queste variabili globali, sono esportate anche due funzioni: una serve per inizializzare i driver, assegnare una locazione nello spazio di indirizzamento IO alle due schede, e ad indirizzare i puntatori a visibilità globale sulla giusta posizione nello spazio di IO, in modo che si riferiscano ai registri corretti delle schede. L'altra funzione invece, viene chiamata quando si vuole comandare l'aggiornamento dei registri da leggere con i valori presenti sugli ingressi, e l'aggiornamento delle tensioni di uscita con i valori presenti nei registri di output.

Date le caratteristiche di VxWorks enunciate in precedenza, i puntatori ai registri e le funzioni possono essere acceduti da qualsiasi modulo del sistema essendo variabili

globali; pertanto, per i moduli che ne hanno bisogno, è facile aggiornare i valori delle lunghezze dei pistoni, ad esempio, semplicemente andando a scrivere il valore desiderato nell'indirizzo corrispondente.

L'integrazione con Simulink

Come detto, ogni modulo del sistema può avere accesso ai puntatori ai registri delle schede di IO.

Questa caratteristica rende molto semplice integrare (a memoria comune) codice sviluppato con applicazioni e strumenti differenti.

Data la possibilità di integrare moduli simulink forniti dalla libreria con moduli prodotti in linguaggio C, l'integrazione tra i driver delle schede di IO e gli schemi di controllo è banale.

Tra i moduli descritti in precedenza a proposito del controllo della piattaforma, il modulo Piston Switch ha accesso al registro che controlla l'apertura del relay relativo al funzionamento dei pistoni, il modulo che comanda l'estensione del pistone ha accesso in scrittura al registro relativo al convertitore DAC che controlla il pistone, e in scrittura al registro ADC dove viene riportata la lettura del sensore lvdt del pistone.

Allo stesso modo, la S-Function all'interno dello schema che controlla lo sterzo ha accesso al registro che controlla la tensione da fornire al motore, e ai registri di ingresso con la torsione del motore e i messaggi di errore.

Oltre ai moduli già descritti, c'è un altro modulo che si preoccupa di raccogliere i segnali rimanenti, ed è il modulo RCA (Rider Command Acquisition). Questo modulo (implementato ancora con il meccanismo delle S-Function), non ha ingressi, ma solo uscite, costituite dai valori letti nei registri relativi all'acceleratore, ai freni, all'inclinazione del pilota, ai tasti del quadro comandi e alla chiave.

Il codice prodotto da matlab, è stato poi modificato inserendo prima dell'inizio della simulazione la chiamata alla funzione che inizializza le schede di IO, e ad ogni passo della simulazione la chiamata che aggiorna le uscite con i valori dei registri, in modo che, istante per istante, i valori comandati alla piattaforma siano quelli richiesti dalla simulazione.

Tutti i moduli matlab accedono inoltre alle funzioni e alle variabili della macchina

a stati (descritte in seguito) per il coordinamento del loro funzionamento nell'ambito dello schema di controllo generale.

3.2.2.4 La comunicazione con moduli esterni

Il sistema comunica con il modulo CONSOLE attraverso l'interfaccia Ethernet e con il modulo RT attraverso l'interfaccia FDDI.

Tutti i messaggi sono costituiti da un header più un corpo.

Header	Corpo

L'header è costituito da 8 bytes

Versione	Tipo	Num oggetti	4 bytes riservati

Nella fase iniziale, avviene uno scambio di messaggi in cui FC comunica il corretto funzionamento, e poi si mette in attesa di un messaggio che determini l'azione che deve eseguire. In particolare, può essere richiesto l'avvio della fase di simulazione, o di una fase di test.

Una volta avviata la fase di simulazione, ha inizio la comunicazione con RT, dove i requisiti di velocità e robustezza del mezzo di comunicazione sono più stringenti; per questo motivo l'interazione tra i due sistemi avviene attraverso l'interfaccia FDDI.

La fase di simulazione può venire interrotta da un'eccezione interna, oppure dalla ricezione (sull'interfaccia FDDI) di un messaggio di Stop.

Il formato dei messaggi scambiati

La comunicazione sulla sottorete Ethernet è asincrona con acknowledge di avvenuta o mancata ricezione. Il protocollo utilizzato è il TCP/IP. Viene creata una connessione con il modulo CONSOLE che ha il ruolo di server. Lo stream socket di comunicazione è non bloccante.

I messaggi inviati dal sistema RT al sistema CONSOLE possono indicano avvenuta o mancata ricezione, o messaggi di notifica di errore interno. Il valore del campo *type* discrimina il tipo di messaggio.

Dal sottosistema CONSOLE invece, i messaggi inviati a FC possono essere di inizio e fine simulazione, spengimento e inizio fase di test.

La comunicazione su FDDI è vincolata da requisiti temporali. In fase di simulazione i messaggi vengono inviati e letti con una frequenza di 100 Hz (T=10msec). La trasmissione su FDDI è di tipo point-to-point e consente una comunicazione bidirezionale. Per la comunicazione si utilizza il *livello applicativo.* Le primitive utilizzate sono non bloccanti.

Su questa linea viaggiano solo due formati di messaggi: *Control Data* e *Sensory Data*.

L'acknowledge al RTS viene inviato tramite il campo ack del Sensory Data, e in base al valore codificato nel campo *type* dell'header del *Control Data* si possono avere messaggi di tipo *Control Data*, *Alarm Message* e *End Sim Message*.

Il formato del messaggio Sensory Data è indicato in figura 3.5.

Campo	Lunghezza (in bytes)
Header	4
Posizione testa	4
Freno anteriore	4
Freno posteriore	4
Coppia sul volante	4
Angolo di inclinazione del pilota	4
Accelerazione angolare del pilota	4
Marcia	1
Acceleratore	1
Clacson	1
Starter	1
Luci	1
Luci di posizione	1
Accensione	1
Acknowledgment	1

Tabella 3.5. Il formato del messaggio *Sensory Data*

Il formato del messaggio Control Data è descritto nella tabella 3.6.

L'interfaccia FDDI

Come detto in precedenza, i driver originali dell'interfaccia FDDI sono stati modificati per poter trasportare i messaggi nel payload del messaggio a livello Datalink.

Campo	Dimensione (bytes)
Header	4
Riferimento di posizione e di coppia	28
Frequenza di vibrazione	4
Ampiezza della vibrazione	4
Contachilometri	4
Tachimetro	4
Contagiri	4

Tabella 3.6. Il formato del messaggio *Control Data*

La fase di invio di un messaggio Sensory Data prevede il riempimento di una struttura predefinita nello spazio di memoria condivisa (*dataToRT*) che ha gli stessi campi del messaggio da inviare al sistema RT. Una volta riempita la struttura con i valori corretti, la chiamata di una funzione fornita dai driver (*sendToRT()*) permette l'invio della struttura al sistema RT.

Allo stesso modo, c'è un'altra struttura di tipo Control Data nella memoria condivisa (*dataFromRT*). Questa struttura viene riempita ogni volta che viene chiamata la funzione *receiveFromRT()*, ed è quindi disponibile per i processi che la richiedono.

L'integrazione con Matlab/Simulink

Per permettere ai moduli Simulink di inviare e ricevere messaggi attraverso l'interfaccia FDDI, è stato implementato un modulo (fddi) apposito tramite il meccanismo delle S-Function.

I dati letti dai sensori della moto (provenienti dai moduli RCA, Steer e Platform) sono raccolti e inviati in ingresso a questo modulo. Dato che il sistema funziona a frequenza di 1 KHz, mentre lo scambio di messaggi con il sistema RT deve avvenire con frequenza di 100 Hz, questo modulo effettua le sue operazioni solo ogni 10 passi.

Poichè l'applicazione di controllo funziona in modo periodico, è necessario che non ci siano procedure che possono determinare lo stallo del sistema. Per questo motivo, le procedure di invio e di ricezione di messaggi sono *non bloccanti*.

Ogni 10 passi di simulazione, il modulo

1. Riempie la struttura *dataToRT* con i dati letti in ingresso e chiama la funzione *sendToRT()*.

2. Chiama la funzione *receiveFromRT()*, per ricevere i dati prodotti dallo washout filter.

Il campo *reserved* (4 bytes) dell'header di *control Data* è utilizzato come contatore e viene incrementato ad ogni invio da RT.

Una volta ricevuto il messaggio, il modulo controlla il campo *type* dell'header: se il messaggio è di tipo *Control*, allora viene controllato il campo reserved. Se il contatore non ha il valore corretto, e sono stati già persi 5 messaggi consecutivi, viene sollevata un eccezione e il sistema va in protezione. Se invece il contatore ha il valore aspettato, i valori letti dalla struttura *dataFromRT* vengono propagati in uscita. Se il contatore non ha il valore atteso, ma sono stati persi meno di cinque pacchetti consecutivi, viene mantenuta l'uscita del passo precedente.

Infine, se il messaggio è di tipo *End of Simulation*, la fine della simulazione viene notificata all'esterno.

Moduli di conversione Spesso, le uscite di un modulo sono espresse in un formato diverso da quello richiesto in ingresso da un altro modulo. Per ovviare a questo problema, sono stati inseriti dei moduli intermedi che realizzano la conversione da un formato all'altro, moltiplicando o sommando le costanti opportune agli ingressi dei moduli che ne hanno bisogno.

3.2.2.5 La gestione delle interazioni

Come specificato in precedenza, il sistema FC ha bisogno di gestire anche eventi asincroni, come le interazioni con gli altri sistemi o la gestione di situazioni di errore.

Il formalismo utilizzato per definire il comportamento *event based* del sistema FC è quello degli *automi a stati finiti* o *macchine a stati*. Il sistema è definito attraverso gli *stati* in cui si può trovare, le *transizioni*, cioè i passaggi da uno stato all'altro, le *uscite* del sistema, che possono essere associate alle transizioni o agli stati, e lo *stato iniziale* da cui parte l'evoluzione del sistema.

Nel caso particolare del sistema FC, la macchina a stati che descrive il comportamento asincrono del sistema è rappresentata nella figura 3.8.

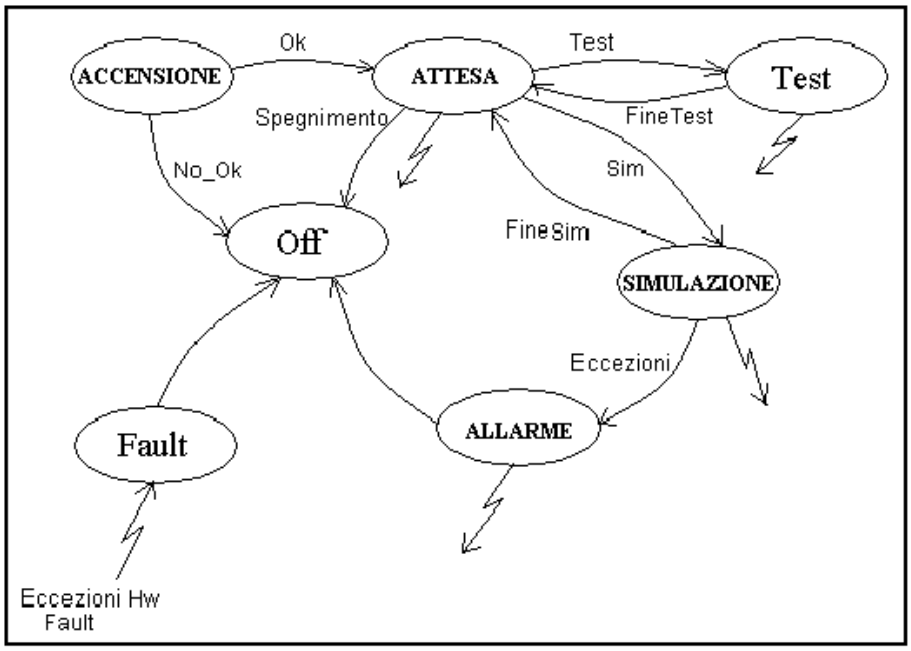

Figura 3.8. La macchina a stati ad alto livello

Vediamo di descrivere il significato dei vari stati:

Accensione: All'avvio, l'applicazione verifica la connessione con il modulo CONSOLE ed il funzionamento dell'impianto. Porta l'impianto in posizione centrale. Nel caso in cui si presentino problemi, attende il primo messaggio da CONS, risponde con un acknowledge negativo e attiva la procedura di spengimento.

Attesa: Attende un messaggio da CONS e risponde con un acknowledge per segnalare che l'impianto è in condizioni di sicurezza (il pilota può salire o scendere).

Test: Se non riceve messaggi da CONS, acquisisce i dati dai sensori e li invia su Ethernet. Il test termina con la ricezione di un messaggio di FineTest.

Simulazione: Viene eseguita la simulazione della guida. Il colloquio con Rt avviene attraverso l'interfaccia FDDI. Si esce dalla simulazione in seguito ad un messaggio di fine simulazione, oppure per il verificarsi di un'eccezione.

Allarme: Quando si presenta un problema in fase di simulazione, il telaio viene

riportato in posizione centrale; si invia un acknowledge a CONS per segnalare che l'impianto è in condizioni di sicurezza e si rimane in attesa di un acknowledge da CONS per poter avviare la procedura di spengimento.

Fault: Si entra in questo stato quando viene recepito un fault dall'impianto. Si invia un acknowledge negativo al modulo CONS e si attiva la procedura di spengimento.

L'implementazione della macchina a stati

Il tool Matlab/Simulink, alla data della realizzazione del sistema MORIS, si presta bene ad implementare le attività periodiche come blocchi simulink predefiniti o come blocchi S-function, ma non permette di codificare attività aperiodiche all'interno di uno schema.

La soluzione adottata è stata quella di implementare la macchina a stati descritta precedentemente in codice C che va ad integrare il codice prodotto automaticamente dal tool Matlab/RTW. La state-machine costituisce un task che si aggiunge ai task generati dal codice di Matlab.

Tutti i blocchi scritti come S-Function interagiscono con la State-Machine: le transizioni del sistema vengono inoltrate a tutti gli altri moduli, e la State Machine attende che tutti i moduli autorizzino le transizioni. Prima di autorizzare la transizione, un modulo può dover compiere delle azioni legate alla transizione. Una volta che tutti i moduli hanno autorizzato la transizione, lo stato del sistema viene cambiato nel nuovo stato. Inoltre la State-Machine fornisce ad ogni modulo la possibilità di "proporre" una transizione dallo stato corrente, invece della transizione che dovrebbe confermare; questo è un meccanismo utile per la notifica di situazioni di errore.

L'applicazione State-Machine è ciclica. Ad ogni passo della simulazione del modello simulink, la state machine viene svegliata, controlla lo stato del sistema, e se necessario effettua una transizione.

Una transizione può essere attivata in diversi modi: nelle transizioni definite "naturali", lo stato attuale del sistema determina lo stato futuro, e la transizione è attivata automaticamente. Ci sono poi degli stati in cui la transizione è determinata dal tipo di messaggio che viene ricevuto su ethernet da CONS (come lo

stato di *attesa*). Quando il sistema si trova in uno di questi stati, la State-Machine provvede a verificare se ha ricevuto messaggi di cambiamento di stato su ethernet. Infine, la transizione può essere proposta dai moduli, in risposta ad un tentativo di cambiamento di stato; questa situazione si verifica nei casi in cui i moduli vogliono notificare una situazione di errore.

Tutti i moduli, prima della fase di simulazione, eseguono una "registrazione" presso la macchina a stati attraverso una funzione fornita. Il comportamento di ogni modulo può venire influenzato dallo stato attuale del sistema, ed eventi registrati da un modulo possono determinare la transizione dello stato globale.

Vediamo adesso in particolare quali sono i comportamenti di ogni modulo in relazione allo stato del sistema.

PistonSfun: il pistone viene pilotato solo se il sistema si trova in fase di simulazione. Se si verifica una situazione di errore, propone la transizione nello stato Fault.

FDDISfun: lo scambio di messaggi sulla fddi viene effettuato solo se il sistema si trova in fase di simulazione. Se viene ricevuto un messaggio di tipo END_SIM, viene proposta la transizione del sistema dallo stato di *simulazione* allo stato di *attesa*. Se viene riscontrato un errore sul tipo di messaggi ricevuti, o se si rileva che un numero di pacchetti persi maggiore di cinque, viene proposta la transizione del sistema dallo stato di *simulazione* allo stato di *allarme*. Nei passaggi intermedi delle transizioni, il modulo si occupa di inviare messaggi di ack o di nack, a seconda della transizione che viene effettuata.

PistonSwitchSfun: come analizzato in precedenza, il modulo stesso possiede uno stato interno. Dallo stato interno dipendono le uscite del modulo, e quindi i riferimenti forniti in ingresso al controllore del pistone. Lo stato interno del modulo dipende dallo stato globale del sistema. In caso di transizione dello stato del sistema, il modulo provvede ad effettuare la transizione interna corrispondente. Nel caso in cui la transizione interna avvenga entro il tempo stabilito, il modulo autorizza la transizione del sistema, altrimenti propone la transizione nello stato di *fault*. Inoltre, se il sistema si trova nello stato di *simulazione*, il modulo chiude il relay che permette il pilotaggio dei pistoni.

SteerControlSfun: se il sistema si trova in fase di *simulazione*, il modulo attiva il *relay* che permette il pilotaggio del motore dello sterzo. Inoltre, se rileva un messaggio di errore dai sensori presenti sul motore, propone la transizione nello stato *fault* del sistema.

EthernetSfun: a seconda della transizione che viene proposta, si occupa di inviare un messaggio di Acknowledgement o di Non Acknowledgement al modulo CONS.

3.2.2.6 Analisi globale del sistema

Come visto, il sistema è in grado di svolgere attività periodiche e attività asincrone. Il comportamento asincrono è regolato dalla macchina a stati descritta nel paragrafo precedente.

Il codice prodotto in maniera automatica dal tool Real-Time Workshop è stato modificato, in modo che nella fase precedente all'avvio della simulazione vengano inizializzati i driver delle periferiche di rete e delle schede di IO, e che ad ogni passo della simulazione viene chiamata la funzione che effettua l'aggiornamento dei dati di input e output sulle schede di IO. Inoltre, il codice originale prevede la verifica che tutte le operazioni necessarie al sistema vengano eseguite nella durata del periodo. In caso contrario, la simulazione viene interrotta. Nel caso del sistema Moris, l'interruzione della fase di simulazione può essere dannosa per il sistema meccanico e per il pilota: per questo motivo il codice è stato modificato in modo che il superamento della deadline non determini la fine della simulazione.

Una visione globale dello schema simulink che realizza il sistema, è rappresentata nella figura 3.9.

Una volta raggiunta la fase di simulazione, il flusso logico delle operazioni è il seguente:

- I dati letti dai sensori della moto vengono inoltrati al sottosistema RT per mezzo del modulo FDDI.

- Il sistema RT invia le elaborazioni dei dati, che costituiscono i segnali di controllo per lo sterzo e la piattaforma.

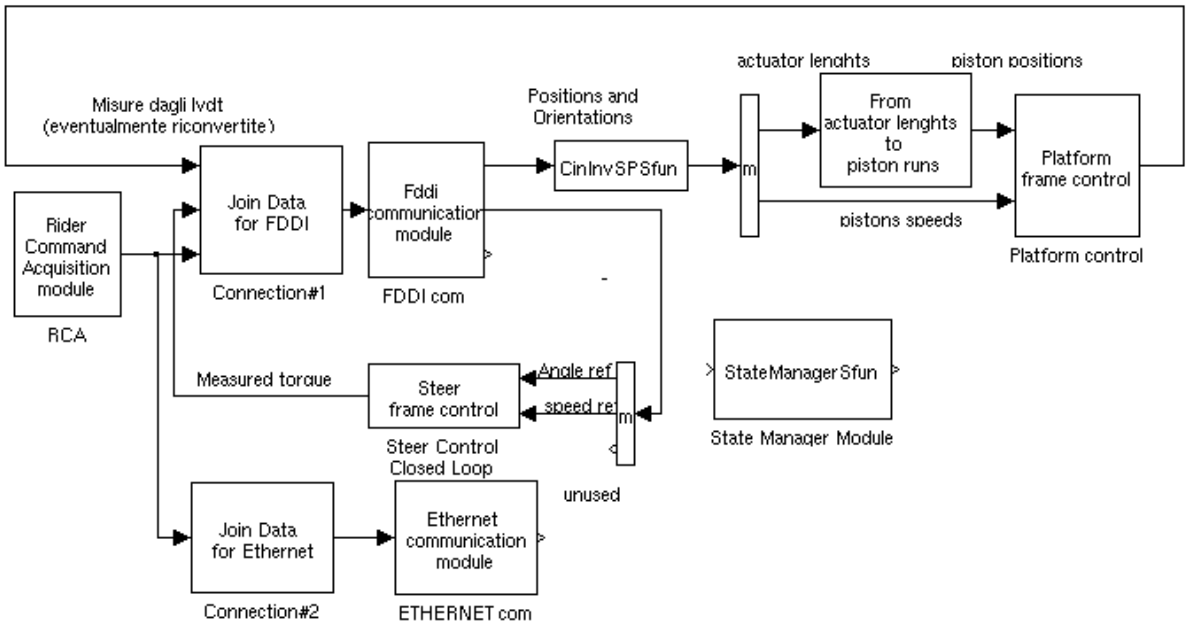

Figura 3.9. Il vecchio FC

- I dati relativi alla piattaforma vengono processati dal modulo che ne realizza la cinematica inversa, e il risultato viene inoltrato al modulo che gestisce il controllo della piattaforma.

- I dati relativi allo sterzo vengono inoltrati al modulo che gestisce il controllo dello sterzo.

3.2.2.7 Aspetti del sistema FC da migliorare

Robustezza del sistema: La struttura ottenuta integrando il codice relativo alla macchina a stati con il codice prodotto automaticamente da Matlab produce instabilità del sistema. Questa è una situazione inaccettabile, in quanto il corretto funzionamento del sistema FC è indispensabile per la sicurezza e l'integrità del sistema meccanico.

Autonomia: La macchina che ospita il sistema FC non è dotata di disco rigido, pertanto dipende da un'altra macchina (*morisdev*) per il caricamento del sistema operativo e per il filesystem. La macchina è equipaggiata con una memoria Flash su cui può essere installato il sistema operativo da avviare. In questo modo si eliminerebbe la dipendenza di FC da altre macchine.

Diagnostica: Come descritto per il modulo RT, il sistema operativo VxWorks ha bisogno di una macchina di supporto su cui viene effettuato lo sviluppo. Questo rende più difficile la fase di debug, che va fatta in remoto. Inoltre la natura real time e concorrenziale del software che gira sul sistema FC ostacola il debug classico step by step, o con istruzioni di stampa su schermo, perché si potrebbe andare a modificare il corretto ordine di funzionamento delle varie operazioni che vengono eseguiti in parallelo.

Per questo motivo, per effettuare diagnostica sui vari valori che coinvolgono la simulazione, ci si affida a soluzioni "offline", come ad esempio la scrittura dei dati raccolti su file (a fine simulazione). La diagnostica delle varie situazioni prevede l'utilizzo di soluzioni *ad hoc*, rallentando pertanto la fase di sviluppo.

Modularità: Sebbene il tool Matlab/Simulink permetta in maniera intuitiva un approccio modulare nello sviluppo del sistema, l'aggiunta del codice che implementa la macchina a stati aggiunge un legame più stretto e meno comprensibile tra i singoli moduli, che non interagiscono più soltanto tramite i normali ingressi e uscite, ma anche tramite i loro stati interni. La rimozione di un modulo può comportare anche la necessità di andare a cambiare la parte di codice relativa alla macchina a stati di altri moduli.

La presenza di interdipendenze più complesse tra i moduli ostacola la verificabilità del sistema, e quindi la sua affidabilità.

Riconfigurabilità: È difficile apportare modifiche al sistema, a causa della sua scarsa modularità. Se si vogliono cambiare le caratteristiche, aggiungere o rimuovere un modulo, oltre all'interfaccia visibile in simulink, è necessario tenere conto delle complesse interazioni che il sistema ha con gli altri moduli per mezzo della macchina a stati.

Analisi delle interazioni: Il fatto che il codice relativo alla macchina a stati sia distribuito tra i vari moduli, e che non sia visibile nella sua interezza, rende difficile la comprensione del funzionamento della macchina a stati e delle sue evoluzioni nel tempo. Conseguenza diretta di questa scarsa comprensibilità è ancora la difficoltà di manutenzione e di verifica del sistema.

Analisi delle fasi del sistema: La presenza di un'unica macchina a stati che si occupa di gestire lo stato del sistema è un limite: ogni sottomodulo del sistema può essere descritto mediante un insieme di stati di funzionamento, ma è la macchina a stati generale che si deve occupare di gestire gli stati interni e le transizioni di tutti i singoli moduli. L'insieme delle possibili configurazioni degli stati dei sottomoduli determina i possibili stati del sistema. Questo aspetto della macchina a stati limita la scalabilità del sistema: l'aggiunta di nuovi moduli provocherebbe un notevole aumento della complessità del sistema.

Il fatto che la macchina a stati sia distribuita su tutti i moduli del sistema rende più difficile la manutenzione e la verifica del sistema: non è possibile andare a verificare il funzionamento di un sottoinsieme del sistema, e in caso di errore occorre identificare quale parte di codice (e quindi quale modulo) ha generato tale errore.

Trasparenza di funzionamento Il codice che implementa la macchina a stati per rendere possibile la gestione degli eventi asincroni è mascherato all'interno dei vari moduli che compongono il sistema. Questa dipendenza dei singoli moduli dalla macchina a stati rende difficile sia la verifica della presenza di errori, sia l'aggiornamento dei moduli e della macchina a stati stessa. Se si desidera modificare il comportamento del sistema (ad esempio aggiungendo uno stato alla macchina), sarà necessario controllare e modificare il codice di tutti i moduli che possono essere stati influenzati da questo cambiamento. Per contro, cambiare il comportamento di un modulo può avere ripercussioni sul flusso di transizioni della macchina a stati.

Portabilità della comunicazione: Come per il sistema RT, i driver originali della periferica per la comunicazione su fddi sono stati modificati, in modo che i dati viaggino contenuti nel payload del pacchetto a livello datalink. Questo è stato reso possibile data la natura *point-to-point* del link in questione. Se da un lato

questo aumenta la velocità con cui vengono trasmessi i dati (si evita il trasferimento dei dati dal livello Network al livello Datalink), dall'altro è limitata la portabilità del codice (funziona solo con i driver modificati, e con questa specifica interfaccia). Inoltre, se l'interfaccia venisse sostituita con una nuova, sarebbe necessario andare a modificare direttamente nei driver il MAC address della vecchia interfaccia con quello della nuova. L'utilizzo delle chiamate standard con i socket avrebbe permesso la portabilità totale del codice utilizzato per la comunicazione. Infine, nei driver modificati non viene fatto alcun controllo sull'integrità dei dati trasmessi e ricevuti. L'unico controllo effettuato è quello ad alto livello sul numero di sequenza del pacchetto ricevuto. Utilizzando il livello network, il controllo di integrità sarebbe stato fornito dal campo Checksum del datagram UDP.

Tool di sviluppo superati: I tool installati sul sistema su cui viene effettuato lo sviluppo delle applicazioni per il sottosistema FC risalgono agli anni precedenti la data di ultimazione del sistema MORIS.

L'utilizzo di tool grafici come Simulink è condizionato anche dalla lentezza dell'ambiente grafico presente sulla macchina *morisdev*.

Sono state rilasciate nuove versioni dei tool Matlab/Simulink, il cui utilizzo è più intuitivo; le versioni più recenti offrono anche la possibilità di sviluppare in modo naturale diagrammi di stato e di integrarli come se fossero blocchi Simulink. Inoltre, le nuove versioni permettono una maggiore verificabilità delle applicazioni prodotte, attraverso vari tool. Purtroppo, data l'architettura software e hardware del sottosistema *morisdev*, è impossibile effettuare un aggiornamento del tool Matlab.

Capitolo 4

Vincoli di progetto e selezione dell'approccio

In questo capitolo ci occuperemo di descrivere e motivare l'approccio seguito nel progetto della nuova architettura del sistema MORIS, e di descrivere gli strumenti utilizzati per svilupparla.

4.1 Approccio utilizzato

Tra le caratteristiche che determinano la qualità del prodotto o del progetto, citiamo l'*affidabilità*, la *modificabilità*, la *comprensibilità* e la *riusabilità*. L'esperienza accumulata finora dimostra che questa proprietà dipendono fortemente da un'altra, la *modularità*.

Si desidera cioè dividere il sistema in moduli che svolgono compiti distinti, e caratterizzati soltanto dalle loro interfacce con il mondo esterno.

Il primo aspetto delicato di un sistema modulare è la *granularità*: dobbiamo regolare le dimensioni dei componenti del sistema in modo che non siano troppo grandi ne troppo piccole: la comprensione di un sistema diviso in tanti sottomoduli che realizzano piccole funzioni è tanto difficile quanto quella di un sistema monolitico che in cui sono ammassate tutte le funzioni svolte.

Al fine di poter comprendere chiaramente l'implementazione dei moduli in cui è

diviso il sistema, questi dovranno essere a loro volta divisi in sottomoduli; pertanto, la modularità del sistema deve essere *gerarchica*.

Per progettare l'interfaccia del modulo occorre tenere presente i principi di *divisione delle responsabilità* e *information hiding*. Secondo il primo, occorre assegnare il lavoro svolto dal sistema (o dal modulo, ricorsivamente) tra i vari moduli che lo compongono, in modo che ognuno abbia un compito definito e limitato. Occorre poi specificare gli obblighi reciproci dei vari moduli, e il comportamento che ogni singolo modulo deve tenere in caso si verifichi una situazione di errore (può risolvere la situazione da solo, o semplicemente notificare l'errore a qualche altro modulo che si occuperà di risolverlo).

Secondo il principio dell'*information hiding*, bisogna rendere inaccessibile dall'esterno tutto ciò che non è strettamente necessario all'interazione con gli altri moduli, in modo che le dipendenze siano ridotte al minimo e che quindi ogni modulo sia implementabile e testabile in maniera indipendente dagli altri.

Si è deciso di utilizzare la stessa divisione logica del sistema utilizzata nella sezione 2.2, e la stessa interfaccia dei vari moduli, andandone però a modificare l'implementazione.

4.1.1 La portabilità

Vogliamo che il nuovo sistema sia dipendente dalla specifica architettura su cui è applicato soltanto per lo stretto necessario. Qualsiasi modulo può essere scomposto in *strati,* individuati in base al livello di astrazione dei servizi richiesti per realizzare il modulo. Uno strato richiede servizi allo strato inferiore e ne fornisce altri allo strato superiore. In questo modo, solo lo strato di livello più basso (quelli che interagiscono direttamente con la piattaforma hardware/software) devono essere implementati in maniera dipendente dall'architettura. In questo modo si può utilizzare tutto il sistema andando a sostituire soltanto gli strati di basso livello, purché i nuovi offrano al resto del sistema la stessa interfaccia dei vecchi moduli.

La suddivisione del sistema in moduli che svolgono funzioni logicamente diverse permette poi di sostituire soltanto i moduli che non sono più adatti al nuovo sistema, e di mantenere intatti quelli che non dipendono dall'architettura del sistema.

4.1.2 Test e verifiche di sistema

Nella fase di test si esegue la parte di sistema da verificare per osservare se i risultati corrispondono a quelli previsti. È impossibile poter testare il sistema (o il singolo sottomodulo) con tutti i valori possibili dei dati di ingresso. Occorre quindi selezionare un insieme di dati di ingresso che sia piccolo per permettere l'esecuzione entro limiti di tempo accettabili, ma che sia anche significativo. La selezione dei dati di ingresso può essere guidata dalle specifiche del sistema, o dalla sua struttura.

La correttezza dei risultati invece può essere decisa dall'utente, dal confronto con le specifiche, o dal confronto con versioni precedenti (test di *regressione*).

La suddivisione del sistema in moduli (e ricorsivamente, in sottomoduli) permette il test del sistema nelle sue parti, prima di effettuarne il test globale.

Il test di unità di ciascun modulo richiede l'uso di moduli ausiliari che simulano il corretto funzionamento di quei moduli che nel sistema reale vanno ad interagire con il modulo che stiamo testando. I moduli che simulano i moduli cliente sono detti *stub*, mentre i moduli che simulano i moduli servente sono detti *driver*. La realizzazione di questi moduli è più semplice di quelli reali (generalmente sono realizzati mediante tabelle, o con l'interazione con il collaudatore).

Una volta verificato il corretto funzionamento dei singoli moduli, si deve provvedere a verificare la correttezza delle interazioni tra moduli diversi. Una possibilità è quella di provare il sistema completo (*big-bang* test), ma è preferibile un approccio incrementale, in cui i driver e gli stub sono sostituiti gradualmente dai moduli veri.

In questo modo è più facile localizzare gli errori di interfacciamento.

4.2 Utilizzo di nuovi strumenti

Dalla data di ultimazione del sistema MORIS, gli strumenti che assistono lo sviluppo hanno subìto notevoli evoluzioni. In particolare, le versioni più recenti del tool Matlab (attualmente è presente sul mercato la versione 6.5) offrono un numero di funzionalità maggiore rispetto alla versione utilizzata per lo sviluppo del sistema (4.1). Inoltre, anche le prestazioni delle macchine attuali sono decisamente maggiori rispetto a quelle utilizzate per il sistema. Le attività svolte dalle cinque macchine

che costituivano il sistema distribuito (grafica, console, sviluppo, gestione delle interazioni, controllo della piattaforma) possono adesso essere gestite da un moderno Personal Computer senza che le prestazioni del sistema ne risentano.

L'intenzione è quindi quella di ridurre al minimo il numero di macchine che costituiscono il sistema, e di avere a disposizione i moderni tool per sviluppare il nuovo sistema, tenendo conto delle osservazioni fatte nelle sezioni 3.2.1.6 e 3.2.2.7.

Infine, sul mercato sono presenti sistemi operativi real-time più aggiornati rispetto alla versione di VxWorks presente su alcune macchine del sistema. Un altro miglioramento dell'architettura del sistema consiste quindi nel sostituire il vecchio sistema operativo VxWorks con un sistema operativo aggiornato.

Durante l'analisi del vecchio sistema, sono stati individuati alcuni moduli che potranno essere riutilizzati nel nuovo sistema, altri che dovranno essere reimplementati in modo da garantire la modularità del sistema, e altri che non possono essere sostituiti perchè legati a vincoli progettuali.

4.2.1 Vincoli di progetto

Il nuovo sistema deve potersi interfacciare con il sottosistema elettro-meccanico esattamente come il vecchio. La maggior parte del costo di lavoro del sistema MORIS è infatti dovuto al sistema meccanico ed elettronico. In particolare, i segnali che attraversano le connessioni sono stati dimensionati e raccolti in maniera adatta alle schede di IO presenti sul sistema: è necessario pertanto mantenere queste stesse schede di IO, al fine di non dover modificare i valori dei segnali e il loro instradamento.

Le schede di IO presenti sul sistema FC, come detto in precedenza, alloggiano sul bus VME, che non è presente nei normali PC.

Per poter riutilizzare le schede della macchina FC quindi si può riutilizzare la stessa macchina, o provvedere alla sua sostituzione con un'altra equipaggiata con lo stesso tipo di bus; al fine di ridurre i costi di aggiornamento del sistema, si è scelto di non sostituire la macchina. Conseguenza diretta di questa decisione è l'impossibilità di sostituire il sistema operativo che gira su questa macchina con uno più aggiornato: il sistema operativo che gira sulla macchina FC deve innanzitutto supportare processore Alpha montato, deve avere caratteristiche Real-Time, e deve supportare il bus Vme.

L'unico sistema operativo che sembra poter soddisfare queste caratteristiche è Linux, utilizzandone una variante Real-Time tra quelle disponibili gratuitamente in rete. L'ostacolo che ha impedito l'utilizzo del sistema operativo Linux sulla macchina FC è l'assenza di supporto del bridge PCI-VME di cui la macchina è equipaggiata (DECchip 7407B). Il progetto Linux-VME infatti prevede l'utilizzo di bridge con il chipset distribuito dalla Tundra-Universe, con cui sono equipaggiate le moderne architetture fornite di bus VME.

4.2.2 La nuova architettura di calcolo

Come specificato in precedenza, le operazioni che venivano fatte dai vari sottosistemi del sistema MORIS possono essere tutte svolte da un moderno Personal Computer. Nel progettare l'architettura Hardware del nuovo sistema di controllo, si è deciso di affiancare la vecchia macchina che ospitava il sistema FC (DEC Alpha-VME) con un PC equipaggiato con processore Intel Pentium 4 a 3 GHz e con una scheda video di ultima generazione NVidia GeForce 5600 Fx, e con sistema operativo Linux (distribuzione *Fedora Core 1.0*).

Il nuovo Pc (NewRT) deve ospitare i moduli del vecchio sistema RT (Dynamic Model e Washout Filter, gestione delle interazioni), il software per l'interazione con l'operatore (nuova console) e la grafica, mentre il controllo della piattaforma sarà ancora delegato al sottosistema FC.

Su NewRT si potranno utilizzare tool di sviluppo aggiornati, tra cui le nuove versioni dei tool con cui era stato sviluppato il vecchio sistema; ricordiamo inoltre che il sistema operativo VxWorks ha bisogno di essere affiancato ad un'altra macchina che gli fornisce gli strumenti di sviluppo remoti: anche questo servizio sarà svolto da newRT.

Si è deciso di utilizzare l'interfaccia FDDI per la comunicazione tra i due moduli, perchè garantisce prestazioni elevate (comunicazione bidirezionale), robustezza (grazie al doppio canale) e basse latenze di trasmissione, rispetto ad una ethernet in cui è presente il problema delle collisioni dei messaggi sul mezzo condiviso.

Il sistema newRT è poi equipaggiato con un'interfaccia ethernet a 100 Mbit che può essere utilizzata per interagire con altri sistemi.

La scelta del sistema operativo Linux è stata determinata dalla sua elevata configurabilità, dalla presenza nel pacchetto di tool di sviluppo configurabili e personalizzabili (come compilatore, assemblatore e librerie di sistema). Il sistema Linux fornisce inoltre inoltre i driver per utilizzare l'interfaccia FDDI che era montata sul vecchio sistema RT, permettendo in tal modo l'interazione richiesta tra i due sistemi. Infine, utilizzando il sistema Linux è possibile, con piccolo sforzo, utilizzarne le varianti Real-Time, che sono spesso fornite sotto forma di *patch* da applicare direttamente al codice sorgente del kernel.

4.2.3 Caratteristiche del nuovo sistema

Si è scelto di riprogettare le componenti del sistema in maniera modulare. Questo garantisce flessibilità, facilità di manutenzione, riusabilità. L'utilizzo di tool aggiornati facilita lo sviluppo, la manutenzione e la verifica del software prodotto.

La nuova architettura prevede sempre un sistema distribuito a due nodi (figura 4.1).

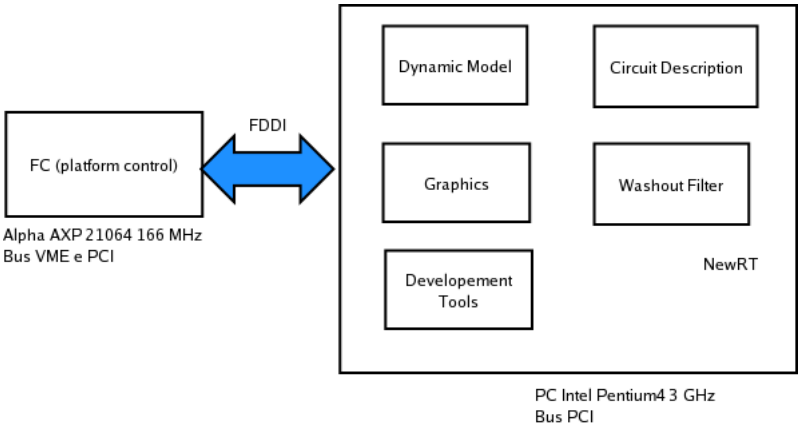

Figura 4.1. L'architettura di calcolo del nuovo sistema

La vecchia macchina che ospitava FC, che continua a gestire il controllo della piattaforma e l'acquisizione dei dati provenienti dal mockup del veicolo, e il nuovo modulo newRT, che si occupa invece di gestire le interazioni tra i vari moduli, di ospitare i moduli *dynamic model* e *strategy manager*, e di gestire la *grafica*, oltre che di ospitare i tool di sviluppo e il file system di FC.

I due nodi dell'architettura possono essere collegati attraverso l'interfaccia ethernet equipaggiata; inoltre è necessario che la comunicazione in fase di simulazione avvenga attraverso l'interfaccia FDDI.

Infine, sarebbe gradito un aumento della facilità di utilizzo del sistema rispetto al precedente.

4.2.4 Moduli da riutilizzare

Procederemo adesso allo studio dei moduli dei sistemi RT e FC che è possibile riutilizzare nel nuovo sistema.

4.2.4.1 Sistema RT

Tra i moduli del sottosistema troviamo che moduli Dynamic model e Washout Filter sono di facile portabilità, in quanto il codice con cui sono stati realizzati è indipendente dalla piattaforma su cui vengono compilati. Oltre a questi moduli è necessario però esportare anche le strutture dati utilizzate per fornire loro i dati in ingresso, e dove prelevare i dati in uscita.

Altri aspetti del sistema che è possibile mantenere sono il Terrain Manager e il Weather Generator, a patto di mantenere la stessa implementazione della descrizione del circuito (segmenti) e delle condizioni atmosferiche dell'ambiente virtuale.

Infine, la porzione di software che gestisce il protocollo di avvio con la Console e con FC è riutilizzabile in quanto utilizza chiamate classiche di comunicazione via socket. Sono invece da implementare del tutto gli oggetti dinamici dell'ambiente virtuale.

Nel riutilizzare il codice prodotto in precedenza per VxWorks per produrre file oggetto da utilizzare nel nuovo ambiente Linux, occorre tenere presente la diversa struttura dei due ambienti e dei tipi di file prodotti. In dettaglio, la visibilità di variabili gestite nello spazio di indirizzamento globale in VxWorks è risolta a livello di spazio di visibilità comune in assenza di conflitti di nomi, e a livello di modulo altrimenti. Questo aspetto di VxWorks viene a mancare in Linux, dove normalmente in fase di linking non possiamo avere variabili condivise con lo stesso nome nei diversi file oggetto che andiamo a linkare insieme.

4.2.4.2 Sottosistema FC

La presenza invasiva del codice della macchina a stati all'interno di alcuni moduli simulink, ne rende difficile la portabilità e la riusabilità in altri sistemi, senza che vi sia apportato nessun cambiamento.

Senza dubbio sono riutilizzabili i moduli che non hanno interazione diretta con la state machine; in particolare, sono direttamente riutilizzabili:

- I driver delle schede di IO, di cui è nota l'interfaccia con il mondo esterno (strutture dati preposte alla comunicazione, funzioni che inizializzano le schede e che avviano l'input e l'output),
- il modulo che gestisce la comunicazione su ethernet (strutture dei messaggi scambiati, funzioni per la comunicazione), dato che utilizza il protocollo TCP/IP e le chiamate standard con socket,
- le strutture di alto livello per la comunicazione su fddi, ma non i driver di basso livello perché specifici dell'hardware e del tipo di rete (point to point) utilizzato.

Ulteriormente, sono riutilizzabili i moduli che effettuano la conversione delle uscite dei moduli negli ingressi di altri, e i moduli che implementano gli schemi di controllo dei singoli pistoni e dello sterzo.

I moduli che interagiscono con la macchina a stati sono tutti realizzati con il meccanismo delle S-Function, e quindi implementati con codice C.

È possibile riutilizzare la parte di codice che ne implementa il funzionamento, mentre va scartata del tutto la parte che realizza la comunicazione con la macchina a stati; di ognuno di questi moduli si può mantenere l'interfaccia visibile a livello Simulink, vale a dire gli ingressi e le uscite che interagiscono con altri moduli.

In particolare, in questo modo è possibile riutilizzare

- La cinematica inversa
- Il modulo che realizza il servocontrollo diretto di ogni pistone e che legge il valore dell'LVDT relativo

- Il modulo che realizza il controllo diretto dello sterzo e la lettura del resolver
- Il modulo che gestisce la comunicazione in fase di simulazione con il sottosistema Fc
- Il modulo che acquisisce i dati dai sensori

La gestione delle interazioni sulla rete Ethernet con il modulo CONS dipende dalla macchina a stati, pertanto è difficile riutilizzarla.

Anche il funzionamento del modulo che fornisce al controllo dei singoli pistoni il polo e i riferimenti di posizione e velocità dipende fortemente dallo stato globale del sistema, per cui è di difficile riutilizzo.

4.3 Introduzione a Case di Sviluppo

Procederemo adesso alla descrizione dei tool di sviluppo utilizzati nello sviluppo della nuova architettura del sistema MORIS.

4.3.1 Matlab Simulink

Simulink è un pacchetto software per la modellazione, la simulazione e l'analisi di sistemi dinamici. Supporta sistemi lineari e non lineari, modellati in tempo continuo, tempo discreto o un ibrido dei due.

Simulink fornisce un'interfaccia grafica per costruire i modelli come diagrammi a blocchi (figura 4.2).

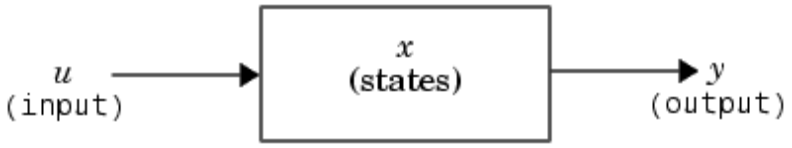

Figura 4.2. Un blocco simulink

Con questa interfaccia, i modelli possono essere disegnati in modo semplice ed intuitivo. Simulink include una libreria di blocchi, tra cui sorgenti, componenti lineari e non lineari, e connettori.

Un'altra funzionalità offerta (e largamente utilizzata nello sviluppo del sistema Moris) è quella di poter creare blocchi personalizzati con il meccanismo delle S-Function, che andremo ad analizzare in seguito.

I modelli sono gerarchici, quindi si possono utilizzare modelli utilizzando approcci "top-down" (scomposizione di modelli in modelli più semplici) o bottom-up (composizione di modelli semplici per ottenere modelli più complessi). Il sistema può essere visto ad alto livello, e si può scendere nel dettaglio dei singoli blocchi.

Una volta che il modulo è stato definito, può essere simulato. Utilizzando delle sonde (scopes) si possono andare a vedere i risultati mentre la simulazione è ancora in corso.

4.3.1.1 La modellazione di un sistema dinamico

Un sistema dinamico reale può essere modellato selezionando ed interconnettendo i blocchi simulink appropriati. Uno schema simulink è un modello grafico di un sistema dinamico (fig. 4.3).

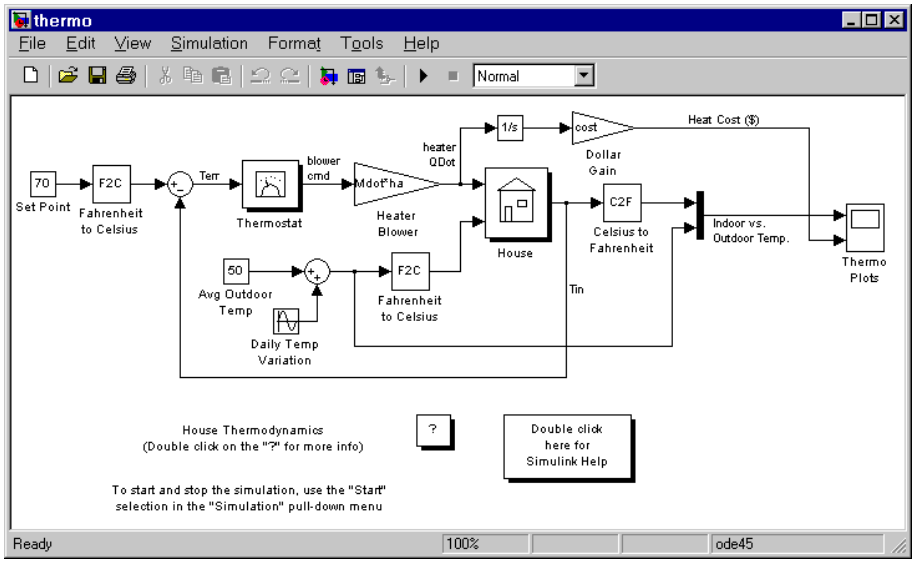

Figura 4.3. Un esempio di sistema dinamico modellato con simulink

Consiste di un set di simboli (blocchi) interconnessi da linee. Ogni blocco rappresenta un sistema dinamico che produce un output. Le linee rappresentano connessioni degli ingressi di un blocco con le uscite di un altro. Il tipo di blocco determina la relazione tra la sua uscita, il suo ingresso, il suo stato e il tempo.

Un blocco può essere caratterizzato da parametri. Questo incrementa la riusabilità dei blocchi definiti.

Simulink permette poi di modellare un sistema complesso come un set di sottosistemi interconnessi, ognuno dei quali è rappresentato da un diagramma a blocchi. Questo approccio incrementa la *modularità* del sistema. Un sottosistema può essere implementato a sua volta da una serie di sottosistemi, a qualsiasi profondità, per creare modelli gerarchici (es., figura 4.4).

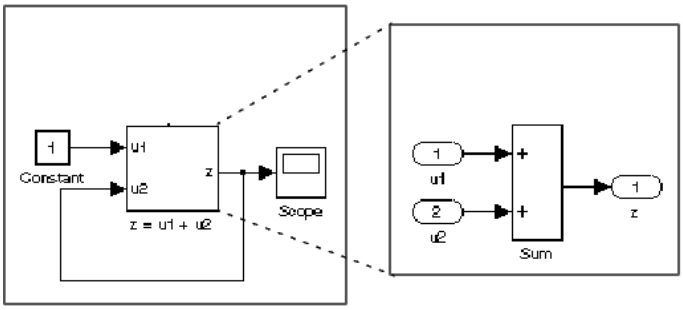

Figura 4.4. Un modello gerarchico di simulink

4.3.1.2 L'analisi del sistema dinamico

Una volta creato il modello, è possibile analizzare il sistema dinamico grazie alle funzionalità offerte da Simulink. Gli strumenti di analisi forniti consentono la simulazione, la linearizzazione ed il calcolo dei punti di equilibrio del modello da studiare.

La simulazione implica l'integrazione numerica di sistemi di equazioni differenziali. Sono disponibili diversi algoritmi per l'integrazione da scegliere a seconda delle caratteristiche del modello.

Le librerie di Simulink offrono blocchi per generare segnali con cui si possono

generare ingressi al fine di verificare il funzionamento del sistema (*sources*), e blocchi per leggere in fase di simulazione i valori dei segnali presenti nel sistema (*sinks*).

4.3.1.3 Le S-Function

Le S-Function sono uno strumento offerto da Matlab/Simulink per permettere all'utente di definire nuovi blocchi Simulink che possono essere utilizzati insieme ad altri. Una S-Function è una descrizione di un blocco simulink in un linguaggio di programmazione (nel caso del sistema MORIS, il linguaggio utilizzato è il C).

Le S-function utilizzano una sintassi particolare, che permette al programmatore di interagire con Simulink.

Durante lo sviluppo del vecchio sistema Moris è stato fatto largo utilizzo di queste funzioni, al fine di:

- Interagire con l'hardware (con i driver delle schede di IO e delle interfacce di rete)
- Gestire gli eventi asincroni e l'interazione dei moduli con la state machine
- Implementare blocchi in grado di effettuare funzioni complesse (come la cinematica inversa)
- Rappresentazione di stati complessi dei moduli (come il pistonSwitchModule)

Nel nuovo sistema si è scelto di utilizzare le S-Function solo per l'interazione con l'hardware e per l'implementazione di eventi asincroni, demandando a moduli sviluppati con le nuove funzionalità offerte da Matlab (come vedremo in seguito) gli altri compiti.

4.3.1.4 Vantaggi offerti da Simulink

Come detto, Simulink offre un approccio naturalmente modulare (e gerarchico) per lo sviluppo di modelli. Utilizzando questo tool si riescono pertanto a soddisfare le esigenze descritte all'inizio di questo capitolo. La possibilità di utilizzare le S-Function per realizzare blocchi che interagiscono con l'hardware specifico, incrementa la portabilità del sistema: nel caso in cui si voglia sostituire l'hardware, basta modificare

l'implementazione interna del blocco, lasciandone immutata l'interfaccia. In questo modo le modifiche da apportare al resto del sistema sono nulle.

Occorre precisare poi che la simulazione del sistema è indipendente dalla piattaforma che ospita il tool. In pratica, una volta definito il modello, questo può essere simulato su qualsiasi macchina.

Le librerie di blocchi fornite con le versioni aggiornate di Simulink sono compatibili con quelle fornite con la versione utilizzata per lo sviluppo del vecchio sistema; è possibile pertanto riutilizzare moduli sviluppati con vecchie versioni del tool.

4.3.2 Matlab/Stateflow

Stateflow (SF) è uno strumento grafico per la modellazione e lo sviluppo di automi a stati finiti; si possono quindi modellare sistemi basati sulla teoria delle macchine a stati, e integrare questi sistemi con moduli dell'ambiente Simulink; è stato introdotto con la versione 5.2 di Matlab, e non era quindi disponibile alla data di sviluppo del sistema MORIS.

Utilizzando Stateflow/Simulink, la progettazione di una macchina a stati avviene in maniera grafica e intuitiva, e la rappresentazione della macchina a stati è quella classica utilizzata nella teoria.

In figura 4.5 si può vedere un semplice diagramma a stati realizzato con SF di un interruttore.

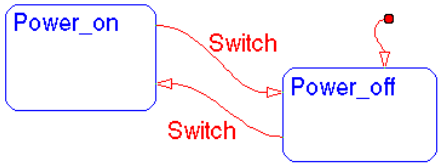

Figura 4.5. Una semplice macchina a stati realizzata con Stateflow

Lo sviluppo di una macchina a stati con Matlab/SF è semplice ed intuitivo. Si possono creare stati e transizioni in maniera gerarchica, all'interno di uno stato, assegnare azioni che il sistema deve compiere in caso di ingresso, uscita o permanenza

in uno stato, o in caso di transizione, si possono associare delle condizioni o eventi che determinano la transizione da uno stato in un altro, e tutti gli altri aspetti previsti dalla teoria degli automi a stati finiti.

SF permette anche di verificare il corretto funzionamento della macchina a stati; è possibile infatti simulare l'esecuzione della macchina a stati, osservandone graficamente l'evoluzione.

Infine, le macchine a stati prodotte con SF possono essere integrate completamente in modelli Simulink. La macchina a stati può essere rappresentata come un normale blocco simulink, con ingressi e uscite. Le transizioni degli stati possono essere influenzate da particolari valori degli ingressi, e le uscite possono essere pilotate da azioni associate agli stati o alle transizioni.

Anche questo aspetto incoraggia la modularità (e il principio dell'*information hiding*): l'implementazione del sistema esterno dipende solamente dall'interfaccia fornita dalla macchina a stati, e non dalla sua rappresentazione interna.

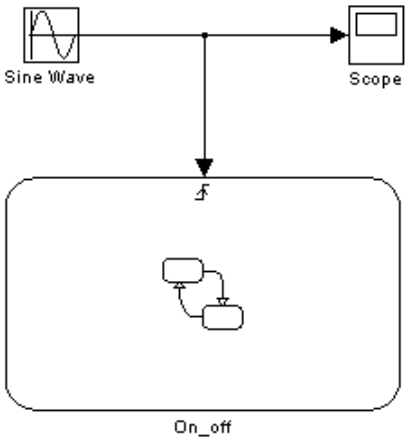

Figura 4.6. Esempio di interazione tra simulink e stateflow

Utilizzando la combinazione Simulink e SF, si è in grado quindi di descrivere in maniera intuitiva sistemi in grado di rispondere a stimoli *clock-based* (utilizzando i normali blocchi *simulink*), *event based* (utilizzando i blocchi SF) e possibili combinazioni dei due (figura.4.6).

4.3.3 RealTime Workshop

Il modulo applicativo Real-Time Workshop (RTW) collegato al programma Matlab/Simulink, è un ambiente per la prototipazione rapida di applicazioni di controllo in tempo reale. Permette la generazione automatica di codice sorgente (in linguaggio C o ADA) che implementa algoritmi di controllo a partire dalla loro specifica in termini di schemi a blocchi Simulink (e Stateflow).

Consente inoltre di gestire automaticamente la compilazione ed il linking del codice generato, in modo da ottenere codice eseguibile su varie piattaforme hardware e con diversi sistemi operativi, eventualmente dotati di possibilità per la gestione del tempo reale.

Il codice generato costituisce un'applicazione real-time *stand alone*, che può essere eseguita su un calcolatore target dotato delle risorse strettamente necessarie. L'applicazione è costituita da un numero di processi dipendente dal modello Simulink, che possono comunicare tra loro e sincronizzarsi.

Il codice viene prodotto integrando una base generica (template) che dipende dal tipo della macchina che andrà ad ospitare l'applicazione, con i file sorgente ricavati dalla conversione dello schema Simulink. In particolare, per ogni architettura è fornito un *makefile*, un file (*rt_ main.c*) che contiene il corpo dei task che costituiscono l'applicazione e la direttiva main() che ne determina l'esecuzione; in particolare il makefile provvede ad effettuare il linking ed includere i file necessari, e a procedere alla compilazione dell'applicazione.

Insieme a real time workshop sono forniti una serie di file template per generare applicazioni per diversi tipi di architetture. Nella versione di real time workshop utilizzata per lo sviluppo del nuovo sistema MORIS, sono presenti file template per la nuova versione di VxWorks (Tornado) e sono incompatibili con la versione presente sul sistema FC. Sarà quindi necessario uno sforzo per adattare questi file alle esigenze del sistema (le operazioni effettuate saranno descritte in seguito).

L'utilizzo di Real Time Workshop favorisce la portabilità dell'applicazione sviluppata: il sistema è modellato mediante schemi simulink, che sono del tutto indipendenti dal tipo di architettura.[1] È poi il modulo RTW che si occupa di tradurre

[1] L'utilizzo di codice dipendente dalla specifica macchina nei blocchi realizzati come S-Function limita la portabilità dell'applicazione; in questo caso è necessario adeguare i blocchi alla nuova architettura.

questi schemi per l'architettura richiesta; pertanto, è possibile, senza alcuno sforzo aggiuntivo, trasportare l'applicazione su tutti i tipi di architettura supportati dal RTW.

4.4 Linux RTAI

RTAI (Real Time Application Interface), è una variante real time del kernel di linux sviluppata dal dipartimento di ingegneria aerospaziale del politecnico di Milano.

RTAI offre gli stessi servizi del kernel di Linux, aggiungendo alcune caratteristiche tipiche dei sistemi operativi real time. In pratica, RTAI intercetta i segnali di interruzione e, se necessario, li indirizza a Linux; quindi, RTAI considera Linux come un task in background, da eseguire quando non sono attive le attività real-time.

RTAI mette a disposizione dell'utente una serie di API per creare task real-time, per gestire la comunicazione inter-processo (pipe, shared memory) e la sincronizzazione dei vari processi (semafori e mutex). Inoltre RTAI permette di utilizzare i servizi forniti da RTAI e dai suoi scheduler nello spazio utente, in processi in soft o hard real time, grazie ad un modulo aggiuntivo (RTAI-LXRT). Il vantaggio principale di questo tipo di approccio (l'alternativa è quella di generare le applicazioni direttamente come processi del kernel) è che si hanno direttamente a disposizione tutti i meccanismi di controllo e protezione della memoria tipici dello spazio utente.

Capitolo 5

Descrizione del lavoro svolto

Dopo aver proceduto all'analisi del sistema preesistente, sono state definite le possibili soluzioni alternative per l'implementazione del nuovo sistema.

Il trasferimento del sistema RT sulla nuova piattaforma ha reso immediatamente possibile l'utilizzo dei tool di sviluppo descritti nel capitolo precedente, quali i vari pacchetti messi a disposizione da Matlab (Simulink, StateFlow, Realtime Workshop); questo non è stato possibile per il sistema FC: parte del lavoro è stato rivolto alla configurazione degli strumenti a disposizione per permetterne l'utilizzo con l'architettura di FC.

Nel tentativo di risolvere i problemi presenti nel sistema FC (descritti nella sezione 3.2.2.7), si è deciso di definirne nuovamente la suddivisione in moduli preposti allo svolgimento di funzioni distinte, e di riprogettarne interamente la gestione delle attività *clock based* con l'utilizzo dei nuovi strumenti di sviluppo (stateflow), riutilizzando i componenti utili.

Si è proceduto quindi alla riprogettazione generale del sistema RT utilizzando i tool offerti dal pacchetto GNU/Linux e i tool del pacchetto Matlab/Linux, fornendo uno strumento pronto per essere utilizzato dalle varianti Real Time di Linux.

5.1 Configurazione degli strumenti di sviluppo

Come detto in precedenza, il sistema *newRT* deve ospitare gli strumenti di sviluppo sia per le applicazioni residenti sul sistema stesso, sia per quelle relative al sistema

FC.

I tool forniti insieme alla distribuzione Linux e quelli del pacchetto Matlab/Simulink sono già configurati per sviluppare applicazioni per la piattaforma newRT.

5.1.1 Configurazione dell'ambiente di sviluppo per il sistema FC

Uno degli obiettivi di sviluppo è quello di poter utilizzare le potenzialità offerte da newRT sia per lo sviluppo delle applicazioni (compito svolto in precedenza dalla piattaforma *morisdev*), sia per svolgere le funzioni di RT. Il sistema newRT è montato su una piattaforma Intel equipaggiata con architettura Linux.

Il primo ostacolo per lo sviluppo su newRT di applicazioni per FC, sta nella differenza del tipo di architettura del processore. In particolare, la macchina AXP-VME che ospita il sistema FC è equipaggiata con un processore Alpha a 64 bit reali, mentre la macchina newRT, sulla quale lo sviluppo deve essere effettuato, è equipaggiata con un processore Pentium Intel a 32 bit.

Le applicazioni di basso livello (driver delle schede di IO e delle interfacce di rete) sono state sviluppate direttamente in linguaggio C. Inoltre, per poter generare applicazioni real-time, il pacchetto Matlab/Simulink/RTW/Stateflow effettua una conversione intermedia del modello sviluppato graficamente in codice C . Per questi motivi, in fase di sviluppo di applicazioni per la macchina FC, occorre poter effettuare la traduzione da linguaggio C a linguaggio macchina (compilazione) compatibile con il processore Alpha.

Durante la fase di compilazione si distinguono la macchina *host* (su cui viene eseguito il compilatore) e la macchina *target* (su cui andrà eseguita l'applicazione generata dal compilatore).

5.1.1.1 Il cross-compiler per Alpha

Il primo passo per poter sviluppare su newRT applicazioni per FC, è stata la configurazione di un cross-compiler.

Nei compilatori utilizzati comunemente, le architetture *host* e *target* coincidono;

nel caso in cui *host* e *target* siano di due tipi di architettura diverse, si parla di *cross-compiler* o di *trans-compiler*.

Oltre ad essere fornito sotto forma di eseguibile insieme alla distribuzione Linux utilizzata, il compilatore *gcc* della *GNU* è disponibile in rete sotto forma di codice sorgente. La possibilità di compilare il compilatore, ne permette una maggiore configurabilità secondo le esigenze: prima della fase di compilazione, si impostano vari parametri con cui va configurato il compilatore, tra cui il tipo di architettura della macchina *host* e della macchina *target*, e i tipi di linguaggi che deve supportare.

La particolare versione del processore ALPHA di interesse insieme con il formato dei file oggetto in uso sul sistema operativo *target* ha richiesto una attenta analisi della versione del compilatore. È stato infatti necessario trovare una versione dei sorgenti GCC che fornissero pieno supporto ad entrambe (piattaforma Alpha e OS VxWorks).

Si è utilizzata la versione 2.95 del compilatore gcc, in quanto il supporto per la cross compilazione verso macchine alpha è stato infatti rimosso nelle versioni più recenti.

5.1.1.2 Compilare per VxWorks

Generalmente, alla fase di compilazione, segue una fase di linking dei vari file oggetto necessari per generare l'eseguibile.

Nel caso specifico di VxWorks, la fase di linking non è necessaria. Lo spazio di indirizzamento è condiviso e generale, per cui un modulo che viene caricato nel sistema operativo (tramite la direttiva ld) ha il diretto accesso a tutte le funzioni disponibili in quel momento.

I file prodotti sono di tipo "*loadable object*", vale a dire oggetti che possono essere dinamicamente caricati (oppure inclusi direttamente in fase di compilazione del kernel) e le cui funzioni esportate diventano disponibili per tutti gli altri moduli. Per eseguire un'applicazione occorre quindi chiamare da linea di comando la funzione che la esegue.

Allo stesso modo è possibile utilizzare tutte le librerie offerte da VxWorks, chiamandole semplicemente. Resta comunque il fatto che il compilatore dovrà avere

visibili i prototipi delle funzioni esterne e le dichiarazioni delle strutture dati condivise che il nostro modulo dovrà utilizzare; per questo motivo, sono state copiati sul sistema newRT tutti gli *header file* delle librerie di sistema presenti sul vecchio sistema *morisdev*, e in fase di compilazione si dovrà indicare al compilatore il percorso corretto di questi file.

5.1.2 La personalizzazione di Matlab/RTW

Il tool Real Time Workshop permette la traduzione in maniera automatica dei modelli simulink in codice C da compilare per ottenere un'applicazione *stand alone*. RTW permette di selezionare il tipo di codice prodotto, in relazione al tipo di architettura del sistema su cui va eseguita l'applicazione.

La versione del sistema operativo VxWorks installata sulla macchina FC non è tra le architetture supportate direttamente da RTW, ma ne è supportata l'evoluzione (Wind River Tornado).

Per riuscire a generare codice compatibile con il sistema FC è stato necessario modificare alcuni dei template generici utilizzati da matlab per produrre il codice sorgente.

La sintassi delle chiamate utilizzate nel codice che costituisce l'applicazione (rt_main.c) è compatibile con quella del sistema operativo VxWorks, pertanto l'adattamento è stato limitato alla modifica del makefile (che viene utilizzato dal tool GNU make, per la gestione automatica della compilazione). In particolare, il compilatore da utilizzare (quello di default) è stato sostituito con il cross compiler, e sono stati modificati i flag di compilazione (è stata rimossa la fase di linking). Sono inoltre stati cambiati i percorsi dei file da includere (si includono adesso gli header file di VxWorks copiati dal vecchio morisdev).

Grazie a queste modifiche, è possibile adesso generare in maniera automatica il codice per il sistema FC a partire da un modello Simulink, ed è possibile compilare questo codice utilizzando il cross-compiler per produrre un *loadable object* che può essere caricato (ed eseguito) sul sistema FC (figura 5.1).

Ulteriori modifiche sono state apportate per poter effettuare operazioni (come inizializzazioni) prima della simulazione vera e propria oppure ad ogni passo della

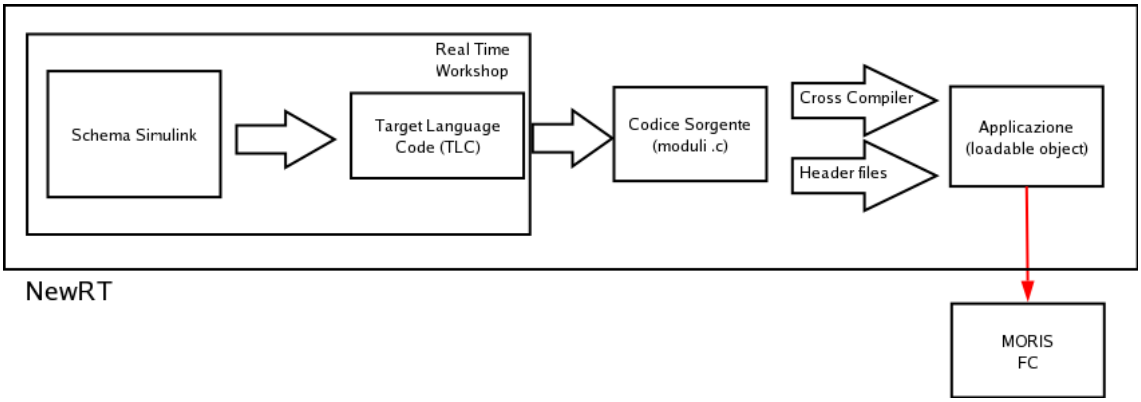

Figura 5.1. La catena di sviluppo di un modello dinamico per FC

simulazione.

In figura 5.2 sono rappresentati i moduli del nuovo sistema, distribuiti sulla relativa piattaforma.

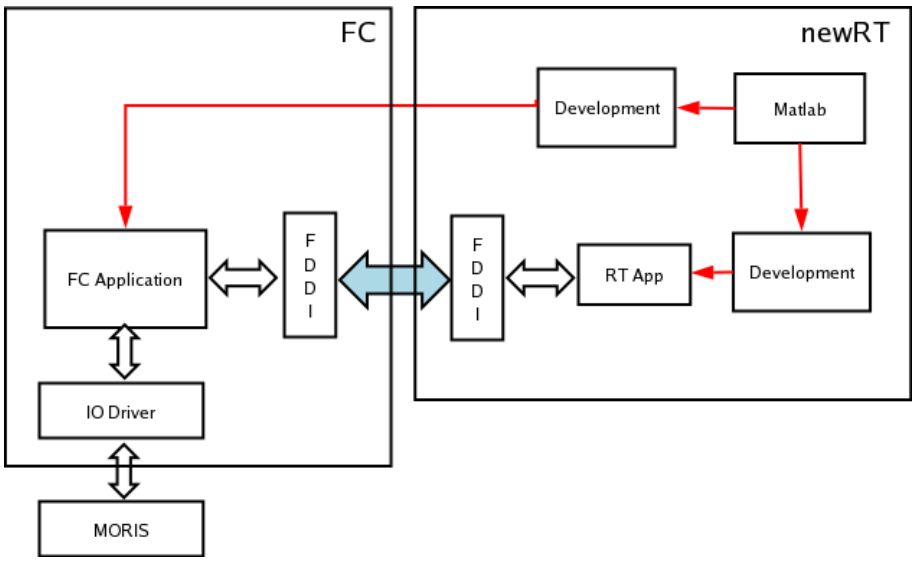

Figura 5.2. L'architettura del sistema di sviluppo e di simulazione

5.2 Sviluppo del Middleware di controllo/sistema

Dato che anche il nuovo sistema è di tipo distribuito, è stato necessario definire e sviluppare il software che provvede alla connessione dei due nodi che compongono il sistema, e definire un protocollo di comunicazione.

Nel vecchio sistema il nodo FC aveva il ruolo di *slave*, nel senso che le transizioni nei suoi vari stati di funzionamento erano comandate dall'esterno (dal modulo Console). Per questo motivo era necessario, oltre al pilota, un operatore esterno che comandasse l'inizio e la fine della simulazione tramite l'interazione con la console.

Nella nuova architettura invece si desidera che sia direttamente il pilota, tramite l'interazione con il veicolo (chiave di accensione, tasto di start) a gestire le varie fasi della simulazione. Pertanto il nodo FC si trova adesso ad assumere il ruolo di *master*, che determina i vari stati in cui si deve trovare newRT. Il flusso di dati di controllo (asincroni) si trova adesso a scorrere nel verso opposto, cioè da FC a newRT.

L'aver assegnato a FC il ruolo di master di simulazione ha anche il vantaggio di irrobustire la struttura della simulazione consentendo la centralizzazione sul nodo più prossimo al controllo del movimento le fasi di rilevamento dello stato del sistema.

5.2.1 Il nuovo protocollo di comunicazione

Adesso che le varie fasi di funzionamento del sistema generale sono comandate direttamente da FC, l'utilità del modulo console è demandata ad operazioni secondarie e non di controllo, quali database management, data view e recording; nel nuovo sistema il nodo console è stato rimosso, e le sue funzionalità assegnate ad un processo a bassa priorità su newRT. In questo modo le uniche comunicazioni di controllo avvengono tra il modulo FC e il modulo RT che gestisce il funzionamento degli altri moduli (dynamic model, washout, grafica). I flussi di informazioni scambiate tra FC ed RT possono essere di due tipi:

- **Messaggi di servizio**: sono i messaggi che determinano la transizione del sistema RT dallo stato di funzionamento allo stato inattivo, asincroni e diretti da FC a RT

- **Messaggi di simulazione**: sono i tipi di messaggi scambiati da RT e FC durante la fase di simulazione, e hanno requisiti di trasmissione più stringenti.

5.2.1.1 Il formato dei messaggi

Il formato dei messaggi dei sistemi RT ed FC è analogo al formato della vecchia versione, descritti nella sezione 3.2.2.4, di cui se ne ricorda solo la struttura generale.

Tutti i messaggi sono costituiti da un header più un corpo.

Header	Corpo

L'header è costituito da 8 bytes

Versione	Tipo	Num oggetti	4 bytes riservati

Per la definizione del corpo dei messaggi, si rimanda alla sezione citata in precedenza.

Nel caso di messaggi che vanno dal sistema FC al newRT, il campo *type* assume il valore TYPE_SENS, e nel corpo ci sono i valori dati dalle letture dei sensori.

Nei messaggi che vanno dal sistema newRT al sistema FC, il campo type assume il valore TYPE_CON, e nel corpo ci sono i riferimenti per la nuova posizione della piattaforma di Stewart.

I 4 bytes riservati (campo *reserved*) sono utilizzati come elemento di verifica della robustezza di simulazione; essi includono un contatore, e servono per controllare che non ci siano stati pacchetti persi. Il sistema FC incrementa il campo di ogni pacchetto che invia a RT, e RT risponde con lo stesso numero che riceve (figura). Se dopo lo scadere del tempo a disposizione di RT per effettuare un passo della simulazione (10 msec) il numero di pacchetto ricevuto da FC non è uguale a quello inviato, si assume che il pacchetto sia perso.

In linea di principio, con il protocollo utilizzato il pacchetto potrebbe essere sia perso che in ritardo. In entrambi i casi risulta non disponibile per la simulazione e la sua assenza viene registrata come comunicazione persa.

Se vengono persi più di cinque pacchetti consecutivi la comunicazione viene interrotta (e il sistema va in stato di protezione, come vedremo in seguito).

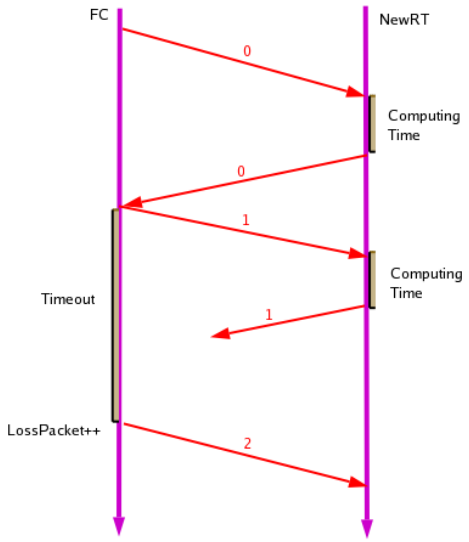

Figura 5.3. Esempio di comunicazione in fase di simulazione

Si procede ora a definire il formato che devono avere i pacchetti di servizio: dato che sono previsti solamente due tipi di configurazioni di funzionamento per il sistema newRT (attivo, e non attivo), ci saranno solamente due formati diversi di messaggio. Al fine di utilizzare la stessa chiamata per inviare e ricevere messaggi, si è scelto di utilizzare lo stesso formato dei messaggi che il sistema FC invia all'altro in fase di simulazione. In questo caso, il corpo del messaggio è privo di significato, mentre il campo type dell'header può assumere il valore START o STOP. A seguito della corretta avvenuta ricezione del messaggio, il sistema newRT invia a FC un messaggio di Acknowledgement riempiendo il campo type del messaggio che invia con il valore TYPE_ACK.

Una volta ricevuto il segnale di START, il sistema RT si mette in attesa (bloccato) dei messaggi di tipo TYPE_SENS da FC. Il sistema FC invece, una volta ricevuto il messaggio di ACK da FC, provvede ad iniziare l'invio dei messaggi di tipo TYPE_CON.

5.2.1.2 Comunicazione e vincoli temporali

FC esegue il ciclo di comunicazione ad una priorità più bassa rispetto al ciclo di controllo. Questo assicura che in nessun caso problemi di comunicazione possano

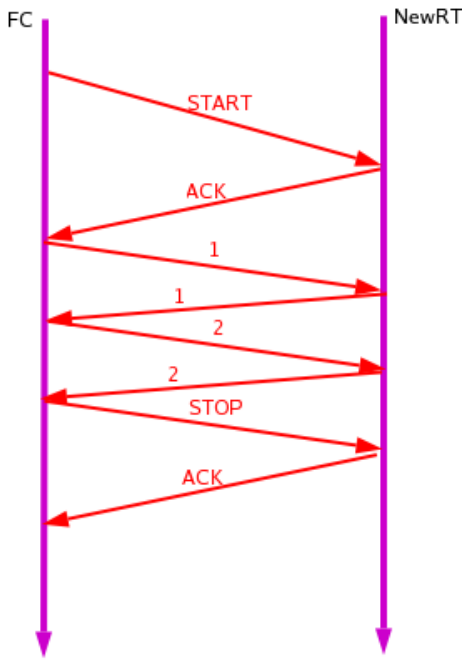

Figura 5.4. Esempio di comunicazione con inizio e fine

alterare il controllo del sistema (figura 5.5).

Inoltre, la trasmissione e la ricezione dei messaggi non sono immediate, e non dipendono solamente dai tempi necessari per trasmettere e ricevere i messaggi.

La frequenza di funzionamento del sistema FC è di 1 KHz, ma la sincronizzazione con newRT avviene ogni 10 msec (100 Hz); questo implica che ogni dieci passi di simulazione, FC invia i dati letti dai sensori, e si aspetta di aver ricevuto l'elaborazione dei dati inviati al passo precedente. Se il pacchetto non viene ricevuto in tempo, si utilizza quello del passo precedente.

Per questo motivo, la somma dei tempi necessari per effettuare l'invio e la ricezione su FC, più i tempi necessari a RT per effettuare la ricezione, i calcoli richiesti e l'invio del nuovo pacchetto, deve essere inferiore al periodo di funzionamento richiesto per RT (10 Msec). Se questo non avviene, il pacchetto sarà considerato perduto. Occorre considerare poi che il sistema FC deve eseguire, a priorità più alta, le operazioni di controllo a frequenza di 1 KHz, che possono ritardare l'esecuzione dell'invio e della ricezione.

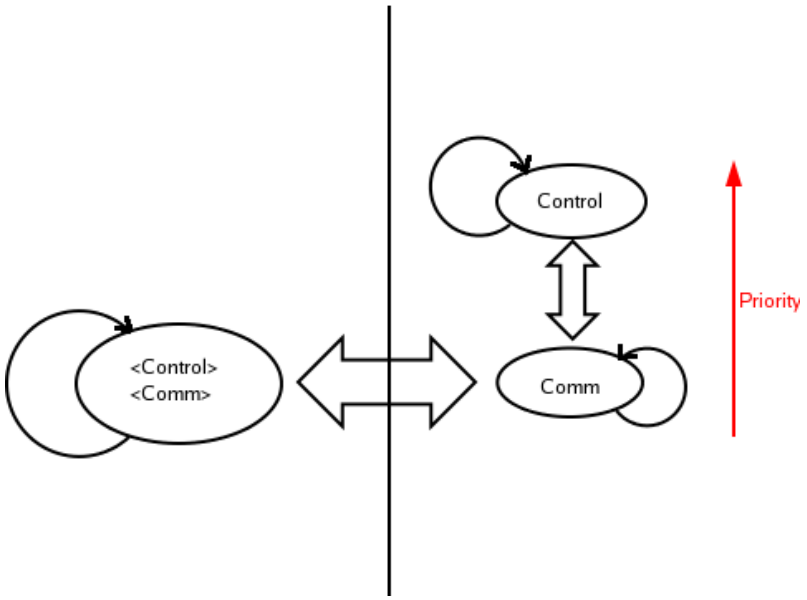

Figura 5.5. I processi coinvolti nella comunicazione

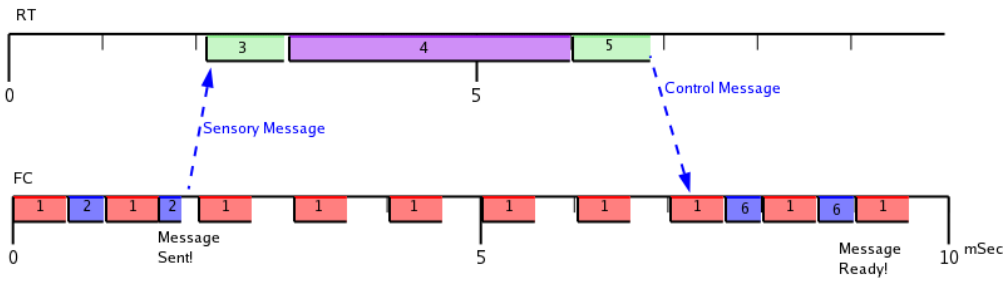

Figura 5.6. Vincoli temporali tra trasmissione e ricezione dei dati

In figura 5.6 sono rappresentati 10 passi di simulazione del sistema FC. I tempi raffigurati hanno il seguente significato:

1. Tempo necessario a FC per svolgere le attività di controllo.
2. Tempo necessario a FC per effettuare la Send. Come si può notare, l'operazione di invio del messaggio ha priorità minore delle operazioni di controllo, per cui può venire deschedulata.
3. Tempo che intercorre tra la ricezione del dato e la sua disponibilità.

4. Tempo necessario per elaborare i dati ricevuti.

5. Tempo per trasmettere il messaggio.

6. Tempo necessario a FC per effettuare la Receive. Come si può notare, l'operazione di ricezione del messaggio ha priorità minore delle operazioni di controllo, per cui può venire deschedulata.

Si potrebbe immaginare un protocollo per cui, essendo la send di tipo non bloccante, questa venga effettuata ad alta priorità nel ciclo di controllo; tuttavia, il profiling delle funzioni ha mostrato come l'uso di questa primitiva, pur essendo non bloccante, richieda in media 230 microsecondi per essere completata. Questa latenza non è accettabile per le necessità di controllo.

L'esecuzione del sistema newRT è sincronizzata dalla ricezione del messaggio, per cui la chiamata della ricezione deve essere bloccante. Nel caso di FC invece, il sistema deve funzionare anche in caso di mancata ricezione; per questo motivo, la ricezione non è bloccante, e allo scadere dei 10 millisecondi si controlla se il pacchetto è stato ricevuto o meno. Inoltre, in caso di trasmissioni multiple da parte di newRT, FC deve ricevere l'ultimo pacchetto inviato.[1]

5.2.1.3 La sincronizzazione iniziale

Una volta che il sistema FC ha ricevuto l'ACK del segnale di inizio trasmissione, occorre verificare che la comunicazione inizi correttamente. Per questo motivo, il pacchetto trasmesso non viene incrementato finché non viene ricevuto il primo messaggio di tipo TYPE_CON. Solo in questo momento la comunicazione inizia correttamente (figura 5.7).

Aver reso master FC rende più agevole la sincronizzazione di sistema e lo startup anche in caso di problemi (come ad esempio ritardi) di risposta da parte di RT. La politica di controllo della comunicazione anche in caso di pacchetti persi consente a FC sia di inoltrare di nuovo richieste di inizio (in caso di perdita della risposta), sia di eliminare le eventuali risposte in surplus (pacchetti ritardati).

[1] Sebbene il normale networking preveda l'arrivo di messaggi fuori ordine (per cui l'ultimo ricevuto può non essere l'ultimo inviato), qua siamo in presenza di una condizione point to point. Sono assenti pertanto le cause che potrebbero determinare un'inversione dell'ordine dei pacchetti.

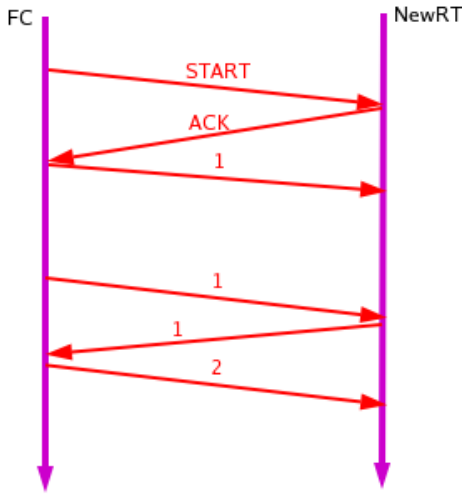

Figura 5.7. La sincronizzazione iniziale

5.2.2 L'implementazione

Mentre nella vecchia versione del sistema le informazioni venivano scambiate nel corpo di segmenti a livello Datalink, nel nuovo sistema si è deciso di utilizzare le chiamate standard di comunicazione su socket. Questa scelta incrementa la portabilità del sistema, perchè lo rende indipendente dal tipo di interfaccia di comunicazione utilizzata (e dai relativi driver). Il protocollo di comunicazione utilizzato è l'UDP, a causa dei suoi minori overhead di comunicazione rispetto al Tcp/Ip.

Sul sistema FC sono state rimosse le modifiche apportate ai driver FDDI originali, mentre su newRT sono stati utilizzati i driver per l'interfaccia FDDI forniti insieme alla distribuzione di Linux.

Le operazioni del sistema newRT sono sequenziali e sincronizzate dalla ricezione del messaggio; per questo motivo, l'operazione di ricezione deve essere bloccante. Una volta ricevuto il messaggio, l'applicazione fa le dovute elaborazioni (Modello dinamico e Washout) e poi invia il messaggio di controllo a FC. Prima di bloccarsi nuovamente, può svolgere le altre operazioni richieste, come la gestione della grafica o di altri sottosistemi. La struttura dell'applicazione è pertanto sequenziale:

- *receiveFromFC()* (bloccante)
- <elaborazioni>

- *sendToFC()* (non bloccante)
- <interazioni con altri sistemi>

Si noti che, essendo più stringenti le esigenze in termini di tempo di FC, dopo la ricezione vengono fatte soltanto le elaborazioni necessarie per l'invio della risposta, postponendo tutte le altre operazioni.

Nel caso di FC invece, il sistema deve continuare a funzionare indipendentemente dal successo della trasmissione e della ricezione dei dati. Le operazioni che il sistema compie per il controllo della piattaforma hanno la precedenza sulla comunicazione con newRT. Inoltre, le chiamate delle funzioni di send e di receive direttamente nel codice che effettua il controllo (sebbene "non bloccanti"), ne avrebbero aumentato il tempo di esecuzione, rischiando di provocare il superamento della deadline nei passi di simulazione sono richieste l'invio e la ricezione.

La soluzione implementata, è stata quella di affiancare al processo generato da Matlab (*controlTask*) altri due processi che si occupano rispettivamente dell'invio e della ricezione di messaggi. Questa soluzione permette di sfruttare al meglio il tempo macchina: le attività di comunicazione vengono svolte nei momenti in cui il processore non è impegnato nelle attività di controllo.

La comunicazione tra i task avviene tramite memoria condivisa: ci sono due zone (*dataToRT* e *dataFromRT*) dove vengono memorizzati rispettivamente i messaggi da inviare e quelli ricevuti da RT. Il *controlTask* (che si occupa di gestire l'input/output verso le schede di IO) utilizza queste zone di memoria per comunicare con i due task addetti alla comunicazione.

Il processo che si occupa dell'invio (*senderTask*) è in un ciclo, bloccato su un semaforo. Quando il controlTask vuole inviare un messaggio, riempie la zona di memoria condivisa con il messaggio, e sblocca il semaforo. Il senderTask ha priorità minore del controlTask, per cui la send viene effettuata soltanto nei momenti il processore non è occupato dal controlTask; in questo modo, l'operazione di invio del messaggio non provoca il superamento della deadline da parte del controlTask.

Per quanto riguarda la ricezione invece, oltre a non appesantire l'esecuzione del controlTask, si ha l'esigenza di avere sempre nel buffer di ricezione (dataFromRT) l'ultimo messaggio ricevuto, per evitare che, a seguito di un ritardo nell'invio da parte di RT, si formino code di pacchetti ricevuti (ma ancora da elaborare). Il

receiverTask quindi è un task sempre attivo, con priorità minore sia del *controlTask* che del *senderTask*, che effettua in ciclo delle receive bloccanti. In questo modo, il task "consuma" le code di messaggi eventualmente accumulate mettendole nel buffer (dataFromRT), e si blocca quando la coda dei messaggi da ricevere è vuota. Quando il controlTask accede al buffer di ricezione, vi troverà sempre l'ultimo messaggio ricevuto, senza overhead aggiuntivo.

Pertanto, ogni dieci passi di simulazione, il controlTask deve effettuare un'operazione di *signal* sul semaforo su cui è bloccato il senderTask, e semplicemente accedere al buffer di ricezione.

5.2.3 Riconfigurazione dello scheduler

In tutte le versioni di Linux ad ora disponibili, il driver della scheda FDDI è accessibile soltanto tramite il protocollo TCP/IP completo; questo comporta che, anche strutturando in tempo reale i servizi su RT, sarà necessaria una componente di comunicazione che gira come processo non real time.

Al fine di verificare che la comunicazione tra i due sottosistemi funzioni entro i tempi richiesti, sono state fatte dei test di invio e ricezione da entrambe le parti. In particolare, il sistema newRT eseguiva una receive bloccante e di seguito una send, mentre il sistema FC eseguiva la send e poi la receive bloccante (figura 5.8).

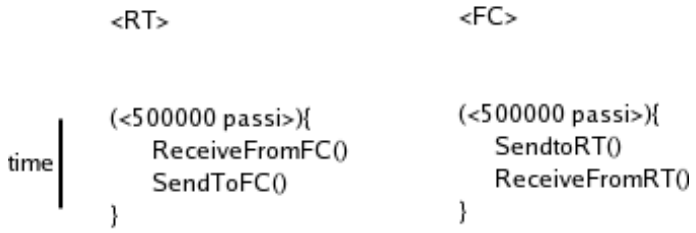

Figura 5.8. Il test di comunicazione

Si è poi analizzato il campione di tempi registrati su 500.000 ricezioni e trasmissioni, espressi in secondi (figura 5.9)

Come si può vedere dalla figura, il comportamento dei ritardi è generalmente corretto, tranne in casi in cui ci sono ritardi superiori a 10 mSec.

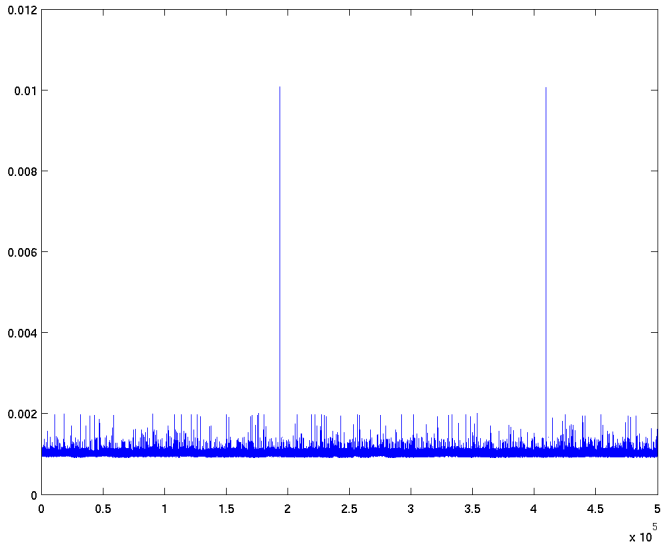

Figura 5.9. I tempi di comunicazione

Il processo residente su newRT è non real time. Questo significa che è soggetto a deschedulazioni in favore di attività del sistema operativo, che hanno priorità maggiore dei processi dello spazio utente. Una volta deschedulato, il processo utente deve aspettare il successivo quanto temporale per poter tornare attivo.

Normalmente, il quanto temporale di Linux è esattamente di 10 mSec.

Per ottenere quanti temporali inferiori, si è quindi proceduto alla ricompilazione del kernel di Linux, andando a modificare il campo che definisce la frequenza di schedulazione da 100 a 1000 Hz.

Andando a ripetere il test eseguito in precedenza, il risultato è stato quello indicato in figura 5.10.

Adesso il ritardo maggiore è dell'ordine di due mSec. Le operazioni richieste a newRT devono essere effettuate entro 10 mSec, tenendo conto dei tempi di ricezione e trasmissione. Per questo motivo, i tempi di comunicazioni ottenuti sono ritenuti adeguati alle esigenze della nostra applicazione.

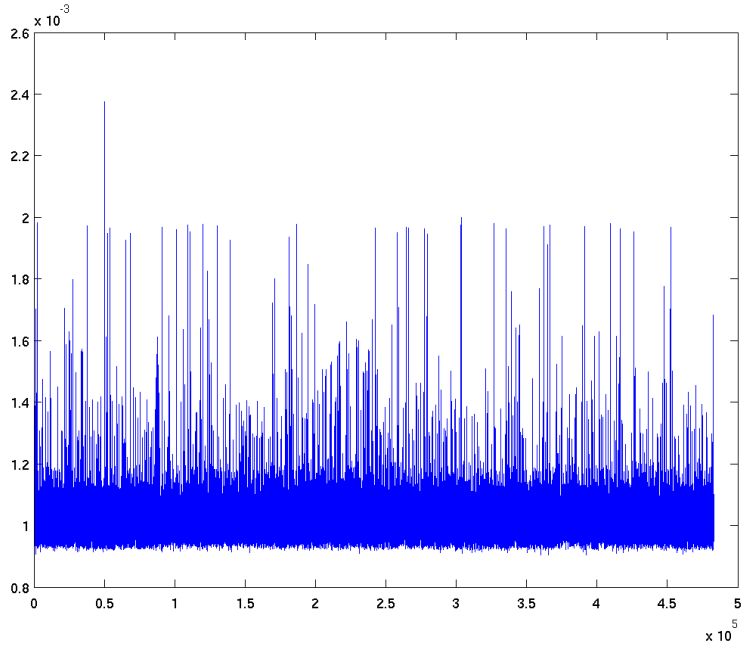

Figura 5.10. I ritardi con kernel a frequenza 1000 Hz

5.3 Sviluppo della nuova architettura di controllo

Si procede adesso ad analizzare la nuova architettura di controllo sviluppata per il sistema FC. Durante questa fase sono stati utilizzati gli strumenti personalizzati descritti nella sezione 5.1.

Durante lo sviluppo, è stato ridefinito lo schema generale (figura 4.1) e la divisione in moduli del sistema, andando a riutilizzare (con le opportune modifiche), alcuni dei moduli presenti nel vecchio sistema (vedi sez.4.2.4.2).

Prima di definire l'implementazione dei sottosistemi, è stata definita l'interfaccia con il resto del sistema, tenendo ben presente i principi della *divisione dei compiti* e dell'*information hiding*.

Ogni modulo sarà descritto secondo la sua interfaccia esterna e secondo i suoi compiti, e quindi ne sarà fatta una descrizione qualitativa dell'implementazione.

5.3.1 La nuova gestione degli eventi asincroni

Come illustrato nella sezione 3.2.2.5, la gestione delle attività asincrone era demandata ad un unico modulo centrale realizzato in una funzione C esterna. Sarebbe stato possibile implementare questo modulo utilizzando i nuovi strumenti a disposizione, ma un approccio di questo genere avrebbe penalizzato la modularità del sistema: nella vecchia architettura, lo stato generale del sistema determinava le attività dei vari sottomoduli.

Durante lo sviluppo del nuovo sistema si è cercato di dare una struttura gerarchica e generale anche alla gestione delle interazioni, in modo che i vari moduli potessero interagire soltanto con i moduli direttamente connessi e quelli di livello inferiore (figura 5.26)

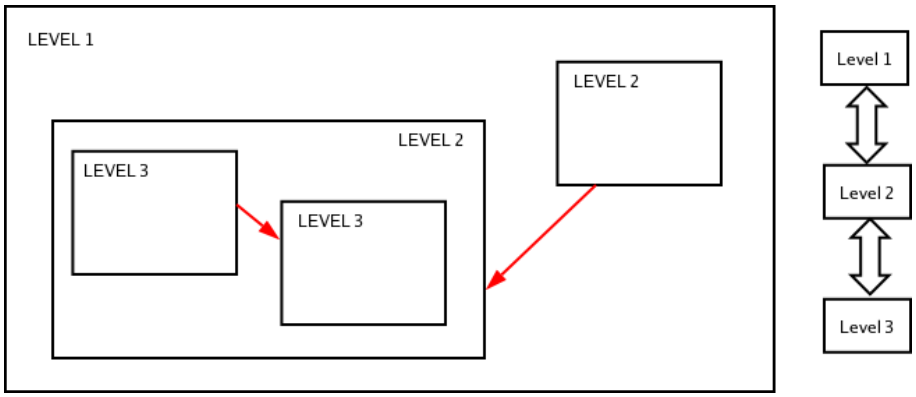

Figura 5.11. I vari livelli di interazione

Oltre all'interfaccia utilizzata per le interazioni relative al controllo, per ogni modulo è stata definita un'interfaccia standard per le attività asincrone.

5.3.1.1 L'interfaccia per le transizioni

Oltre agli ingressi e alle uscite relative al funzionamento sincrono, ogni modulo viene dotato di un ulteriore ingresso che definisce lo stato in cui si desidera che si trovi (stato desiderato, o *desired state*), e di un'uscita definisce lo stato in cui il modulo si trova attualmente (stato riportato, o *reported state*). (figura 5.12).

Il *desired state* di un modulo può assumere tre valori distinti:

- OFF, in cui il modulo è inattivo, e aspetta di essere attivato (è lo stato iniziale),
- READY, in cui il modulo è attivo, e pronto per andare in fase di funzionamento,
- RUN, in cui il modulo sta svolgendo le sue attività.

Il *reported state* di un modulo invece può assumere quattro valori, di cui tre sono quelli del *desired state*, più uno stato aggiuntivo

- ERROR, in cui il modulo notifica che si è verificato un'errore interno.

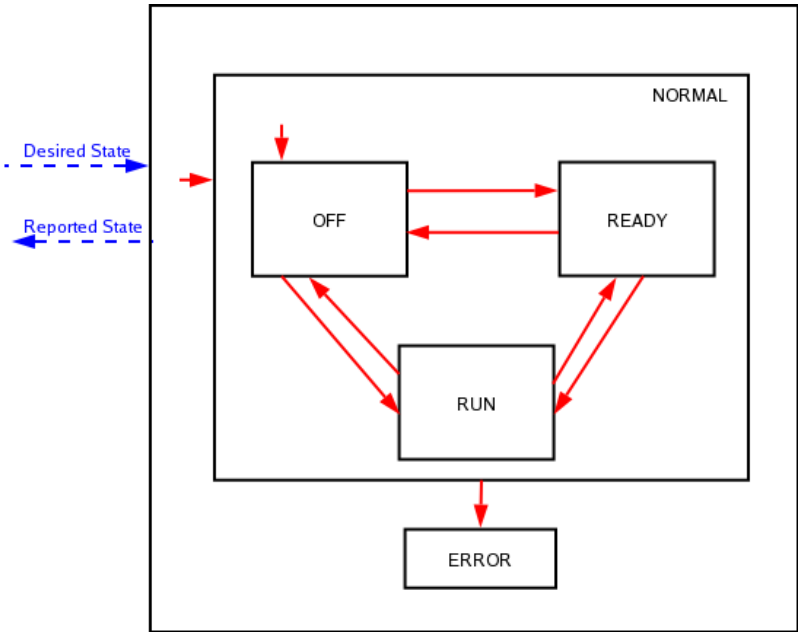

Figura 5.12. L'interfaccia del modulo

Il valore dell'ingresso *desired state* determina la transizione da uno stato all'altro del sistema.

5.3.1.2 Un approccio gerarchico

Dal momento in cui viene richiesta la transizione di un modulo, questo compie tutte le operazioni necessarie per transire nel nuovo stato; tra queste operazioni, è

previsto che ci siano aggiornamenti degli stati dei moduli a livello inferiore. Una volta che tutti i moduli di livello inferiore hanno notificato l'avvenuta transizione, il modulo notifica all'esterno il nuovo stato. Ogni modulo interagisce quindi soltanto con i moduli di livello superiore (riportando il suo stato) e inferiore (richiedendo transizioni di stato); rispetto ad un sistema in cui lo stato generale determina le azioni che devono fare tutti i singoli moduli, la modularità è incrementata (figura 5.13).

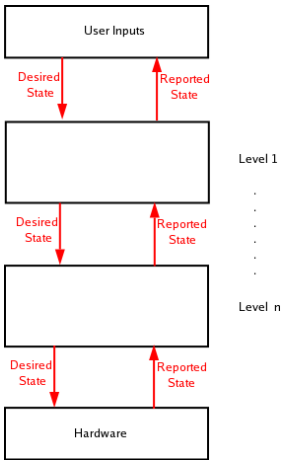

Figura 5.13. Le interazioni gerarchiche

Lo stato generale del sistema è determinato dalle interazioni con l'utente del sistema, mentre al livello più basso si determina lo stato di funzionamento dell'hardware; ovviamente, ogni livello può avere più di un modulo nei livelli inferiori.

Si è deciso di delegare il singolo modulo per la gestione delle situazioni di errore, e di notificarle all'esterno.

5.3.1.3 La verifica di un modulo

Utilizzando questo tipo di approccio, è possibile verificare il corretto funzionamento dei singoli moduli in maniera indipendente: la verifica può avvenire pilotando il *desired state* di un modulo, e andandone a leggere il *reported state*. Il livello inferiore può essere rimosso semplicemente cortocircuitando il *desired state* e il *reported state* (figura 5.14).

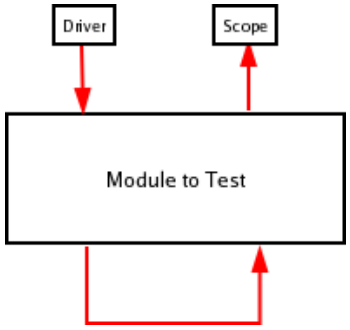

Figura 5.14. Il test del modulo

Ovviamente, per non determinare errori interni al modulo, gli ingressi relativi al controllo devono essere pilotati correttamente.

5.3.2 Il controllo della piattaforma

Il controllo della piattaforma costituisce un modulo logico del sistema globale (*platform subsystem*).

L'interfaccia di questo modulo (figura 5.15) è costituita da

- Posizione desiderata della piattaforma, è il risultato delle elaborazioni del modello dinamico e dello washout (IN);
- Stato desiderato della piattaforma, è lo stato di funzionamento in cui si desidera che il modulo che controlla la piattaforma si trovi (IN);
- Estensione dei pistoni, è il valore dell'estensione dei sei attuatori della piattaforma, riportato dai sensori LVDT (OUT) ;
- Stato riportato della piattaforma (OUT).

Come per tutti gli altri moduli, sono previsti tre stati di funzionamento per la piattaforma (OFF, READY, RUN), più lo stato di ERRORE. Andiamo a descrivere adesso quale deve essere il comportamento per ognuno di questi stati (figura 5.16).

Off: Nello stato di OFF (A) la piattaforma è chiusa. I dati provenienti dalla cinematica inversa sono ignorati, e i pistoni sono comandati in stato chiuso.

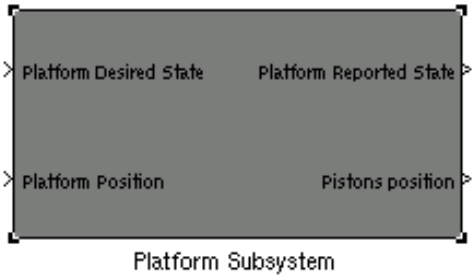

Figura 5.15. L'interfaccia esterna della piattaforma

Ready: Nello stato READY (B) la piattaforma è in posizione centrale (pistoni a mezza corsa). I dati provenienti dal Washout filter sono ignorati.

Run: Nello stato RUN (C), la piattaforma insegue il riferimento generato dal Washout filter (Simulazione).

Error: Si è verificato un errore critico, viene comandata l'apertura del relay che attiva il controllo dei pistoni.

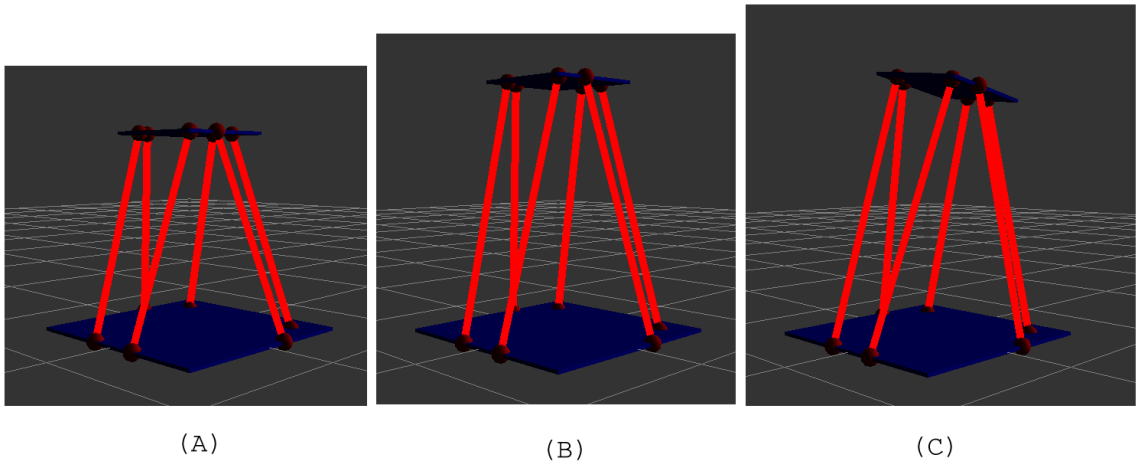

Figura 5.16. I tre possibili stati della piattaforma

Vengono distinte poi due tipologie di errore: errori *soft*, in cui l'eccezione è sollevata ad esempio da un errato valore dei dati di ingresso, ed errori *hard* in cui si

rileva un malfunzionamento della parte meccanica del sistema. In caso di errori *soft* è possibile riportare il sistema in uno stato coerente; in caso di errori *hard* invece, il sistema meccanico viene disattivato tramite l'apertura del relay che ne comanda l'alimentazione, ed è impossibile recuperare lo stato del sistema.

Subito dopo l'accensione del sistema, la piattaforma entra in fase di test per verificare il corretto funzionamento dell'hardware. Se la fase di test ha successo, si può procedere al normale utilizzo; altrimenti la piattaforma notifica una situazione di errore.

Il modulo *platform subsystem* è a sua volta composto da vari sottomoduli:

5.3.2.1 Cinematica inversa

La cinematica inversa è il modulo che si occupa di convertire i riferimenti di posizione, velocità e accelerazione generati per la piattaforma, nei riferimenti per i singoli pistoni. Il modulo è implementato con il meccanismo delle S-Functions.

Si è deciso di riutilizzare il vecchio modulo, epurato del codice che implementava l'interazione con la state machine aggiungendo un'uscita, che serve a notificare eventuali situazioni di errore rilevate (il modulo è in grado di accorgersi se la configurazione richiesta alla piattaforma può provocare intersezioni negli attuatori).

5.3.2.2 Il controllo del pistone

Le informazioni prodotte dalla cinematica inversa vengono trasmesse ad un modulo che contiene al suo interno i sei moduli che gestiscono il controllo dei pistoni (*piston module*). Il *piston module* è dotato della normale interfaccia per la gestione degli eventi asincroni, più l'interfaccia relativa al controllo:

- Riferimento di posizione desiderato (IN),
- Lettura del sensore di posizione del pistone (OUT).

Lo stato desiderato per ciascuno dei sei pistoni coincide con lo stato desiderato per l'intera piattaforma. Poiché, a seguito di una richiesta di transizione, ogni pistone dovrebbe effettuare la transizione nello stato richiesto, il *reported state* della piattaforma viene cambiato solo quando tutti e sei i pistoni hanno notificato l'avvenuta

transizione. Inoltre, se uno solo dei sei pistoni segnala lo stato di errore, l'intera piattaforma segnala esternamente lo stato di errore.

Andiamo adesso ad analizzare l'implementazione di ogni *piston module*.

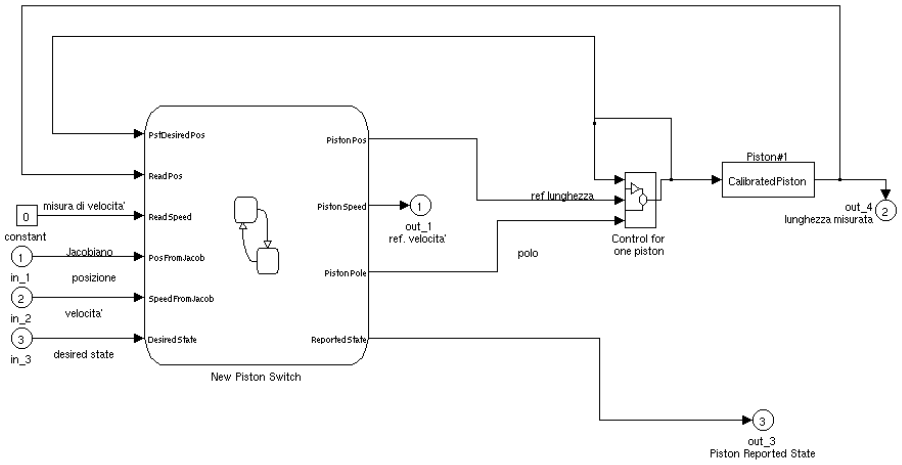

Figura 5.17. Il controllore del pistone

Confrontando la nuova implementazione in figura 5.17, con la struttura precedente (figura 3.4), si può osservare che la parte che gestisce il controllo di basso livello è rimasta immutata (il ciclo di controllo del riferimento di posizione e il blocco S-Function che interagisce con i driver di basso livello). L'utilizzo del tool *Stateflow* ha permesso la rimozione di codice C che veniva integrato nei vari moduli con lo scopo di gestire le interazioni con la macchina a stati.

Il vecchio PistonSwitch, che gestiva le interazioni con la macchina a stati globale e comandava lo stato di funzionamento del sistema, è stato sostituito con un modulo Stateflow che rende il sottosistema compatibile con la nuova struttura gerarchica. Al livello inferiore troviamo direttamente il pistone, mentre al livello superiore troviamo il modulo che realizza il controllo di tutta la piattaforma.

Le corrispondenze tra gli stati in cui si trova il *piston module* e i dati forniti in ingresso al controllore sono rappresentate nella tabella 5.1.

Procediamo adesso a descrivere il comportamento del modulo:

	OFF	READY	RUN
Riferimento di posizione	Posizione Base	Posizione Centrale	Posizione dalla cinematica
Riferimento di velocità	0	0	Velocità dalla cinematica
Polo	Lento	Lento	Veloce

Tabella 5.1. Relazione tra stato del modulo pistone e valori forniti al controllore

Per poter notificare all'esterno la realizzazione delle transizioni richieste, è necessario che le transizioni comandate al pistone (fisico) siano avvenute. Se le transizioni richieste al pistone non avvengono entro un intervallo di tempo prestabilito, il modulo va in stato di errore.

Una volta avviato il sistema, ha inizio la fase di test. Il pistone viene scostato di 10 mm dalla posizione base, per poi esservi riportato. Se questa operazione non ha successo entro un intervallo di tempo prestabilito, il modulo del pistone transisce nello stato ERROR, altrimenti il modulo va in stato di OFF.

In stato di OFF, il pistone è comandato in posizione chiusa.

Se viene richiesta al sistema la transizione nello stato READY, il modulo fornisce il riferimento della posizione centrale e il polo lento al controllore del pistone. Quando la posizione misurata dal sensore del pistone si discosta dalla posizione centrale di un valore inferiore alla tolleranza, si assume che il pistone abbia raggiunto la posizione centrale, e il modulo notifica all'esterno l'avvenuto cambiamento di stato.

Quando il sistema si trova in stato READY, può essere richiesta la transizione nello stato OFF. In questo caso, si fornisce al controllo in posizione il riferimento della posizione base, e il polo lento. Se il pistone non raggiunge la posizione richiesta entro il timeout, il modulo va in stato di errore.

Se invece viene richiesta dallo stato READY allo stato RUN, il polo lento fornito in ingresso al controllore viene sostituito con un polo veloce.

L'ingresso in fase di RUN è delicato: il riferimento di posizione costante viene sostituito con quello proveniente dallo Washout Filter. Una discontinuità tra le due posizioni in ingresso potrebbe essere pericolosa per il sistema e per l'impianto. Per questo motivo, prima di fornire al controllo del pistone il riferimento proveniente dalla cinematica inversa, si aspetta un numero massimo di passi di simulazione, verificando la continuità di posizione; se la posizione comandata alla piattaforma,

e quindi ai pistoni (attraverso il modulo della cinematica inversa), differisce di un valore troppo elevato da quella della posizione di READY, il sistema va in stato di errore; in caso contrario, il riferimento di posizione fornito in ingresso al controllore è quello fornito dalla cinematica inversa (fase di simulazione).

Durante la fase di RUN viene effettuato un continuo controllo dei dati (monitoring), e se la posizione comandata al pistone e la posizione letta sul sensore relativo si discostano di un valore troppo grande, significa che c'è qualche problema nell'hardware (meccanico, o nel sensore di lettura dell'estensione). In questo caso, il modulo va in stato di fault.

Se viene richiesto al modulo di transire dallo stato di RUN allo stato di READY, il riferimento di posizione fornito al controllore dalla cinematica inversa viene sostituito con il riferimento della posizione centrale, e il polo veloce con quello lento. Una volta che la posizione del pistone letta dal sensore raggiunge la posizione centrale, il modulo notifica all'esterno l'avvenuta transizione.

Procediamo ora con l'analisi dell'interfaccia di questo modulo. Gli ingressi sono:

- PstDesiredPos, è l'estensione imposta al pistone dal ciclo di controllo,
- ReadPos, è l'estensione del pistone misurata dal sensore relativo,
- ReadSpeed, è la velocità del pistone letta dal sensore (inutilizzata),
- PosFromJacob, è il riferimento di posizione generato dalla cinematica inversa,
- SpeedFromJacob, è il riferimento di velocità generato dalla cinematica inversa,
- DesiredState, è lo stato desiderato del sistema.

Le uscite invece sono:

- PistonPos, è il riferimento di posizione che si vuole che il pistone insegua,
- PistonSpeed, è il riferimento di velocità che si vuole che il pistone insegua (inutilizzato),
- PistonPole, è il polo che viene fornito in ingresso al controllore per determinare la velocità di inseguimento,

- ReportedState, è lo stato in cui il modulo si trova.

I valori dei riferimenti di posizione centrale e base, e dei poli da fornire in ingresso al controllore, sono definiti in costanti all'interno del modulo Stateflow.

5.3.2.3 RelaySfun

Il modulo relaySfun controlla il relay che comanda le uscite relative alla posizione dei pistoni. Se l'ingresso assume il valore della codifica di errore, il relay viene aperto, le schede disattivate e i pistoni comandati in posizione di homing. Nella versione precedente, questa funzione era integrata nel PistonSwitch. Adesso, per effettuare una corretta divisione dei compiti, questa funzione è stata inserita in un modulo a parte.

5.3.2.4 L'integrazione dei sottomoduli

Il modulo complessivo che realizza il controllo della piattaforma è rappresentato in figura 5.18.

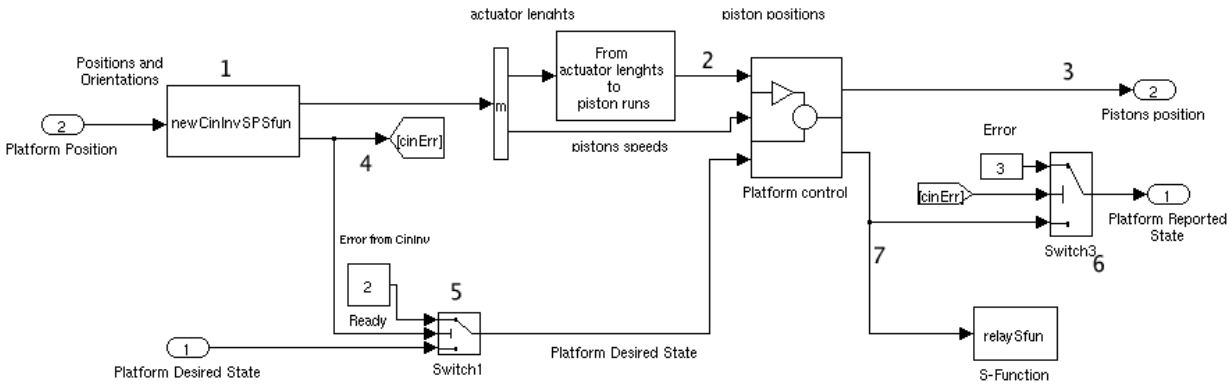

Figura 5.18. Il nuovo controllo della piattaforma

Vediamo di descriverne il funzionamento globale.

Lo stato desiderato per la piattaforma viene trasferito direttamente sullo stato desiderato per i singoli pistoni.

Il riferimento di posizione della piattaforma proveniente dallo Washout Filter viene tradotto nelle lunghezze dei pistoni dal modulo che realizza la cinematica inversa (1). Le lunghezze dei pistoni vengono poi convertite nelle corse degli attuatori (2) e trasferite in ingresso al modulo che controlla il pistone. Le posizioni dei pistoni lette dai sensori vengono fornite in uscita al modulo (3).

Se il modulo che si occupa di realizzare la cinematica inversa rileva una situazione di errore (4), viene comandata la transizione dei controllori dei pistoni nello stato di Ready (5), e notificata all'esterno uno stato di errore (6).

Se invece è uno dei moduli che effettua il controllo dei pistoni a notificare lo stato di errore (errore hard), il relay viene aperto, e viene notificato all'esterno lo stato di errore (7).

5.3.3 Il controllo dello sterzo

L'interfaccia fornita dal modulo che gestisce il controllo dello sterzo (steer subsystem) è rappresentata dalla figura 5.19.

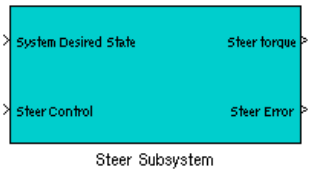

Figura 5.19. L'interfaccia dello *steer subsystem*

L'interfaccia è costituita da

- Lo stato desiderato per lo sterzo (IN)
- I riferimenti di posizione e velocità angolare dello sterzo (IN)
- La coppia torcente dello sterzo misurata dal resolver (OUT)

Lo steer subsystem può trovarsi in due stati di funzionamento (RUN e OFF), e la commutazione tra questi stati è automatica. Per questo motivo si assume che lo

stato riportato sia sempre uguale allo stato desiderato. Durante lo stato di ON, i riferimenti di posizione e velocità provenienti dall'ingresso del modulo sono comandati al motore dello sterzo, mentre il valore di torsione dello sterzo è fornita in uscita dal modulo. Durante lo stato di OFF invece, l'uscita del modulo è nulla, e i riferimenti di posizione sono ignorati.

L'implementazione dello *steer subsystem* è rappresentata in figura 5.20.

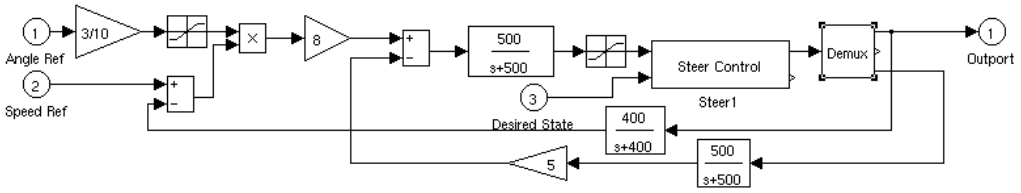

Figura 5.20. Il controllo dello sterzo

Il ciclo di controllo PID è lo stesso sviluppato per il precedente sistema.

Il blocco Steer Control interagisce direttamente con i driver delle schede di IO, e può comandare l'uscita relativa alla coppia dello sterzo, e leggere gli ingressi relativi all'angolo di sterzo forniti dal resolver montato sul motore; oltre a questi valori, il modulo controlla l'uscita di tipo *relay* che controlla l'alimentazione del motore, e i quattro ingressi digitali su cui il motore notifica malfunzionamenti (tipicamente surriscaldamento).

Se lo stato desiderato del sistema è RUN, la posizione angolare fornita dal ciclo di controllo viene comandata al motore dello sterzo.

I valori di posizione e velocità letti dal resolver montato sullo sterzo vengono utilizzati nel ciclo di controllo, e la posizione angolare è fornita anche in uscita al modulo globale.

Se lo stato desiderato del sistema è OFF, i dati provenienti dal controllore sono ignorati, e l'uscita è nulla.

In condizioni normali, il relay è aperto in stato di OFF, e chiuso in stato di RUN; se però gli ingressi digitali notificano una situazione di errore, il modulo provvede ad aprire il relay per interrompere l'alimentazione del motore. Si è scelto di non

notificare la situazione di errore ai moduli di livello superiore, perchè questo tipo di errore non costituisce rischio per la struttura e per il pilota.

5.3.4 Il RCA module

Il modulo RCA si occupa di fornire al resto del sistema i valori dei dati letti dai sensori presenti sul mockup della moto.

L'interfaccia è descritta in figura 5.21.

Figura 5.21. L'interfaccia del RCA module

Il modulo non è dotato di ingressi rispetto al resto del sistema simulink, perchè preleva i valori direttamente dal mockup della moto. Le uscite del modello sono i valori delle letture dei sensori, e in particolare:

- Acceleratore,
- Freni anteriore e posteriore,
- Inclinazione del pilota,
- Chiave di accensione, starter, clacson, luci.

All'interno del modulo (figura 5.22) c'è un blocco simulink realizzato come S-Function, che ha accesso alle zone di memoria condivisa utilizzate dai driver delle schede di IO, e aggiornate con i valori letti provenienti dal mockup.

Le letture dell'acceleratore e dei freni vengono poi elaborate per essere idonee ad essere elaborate dal resto del sistema[2].

[2]Nella precedente versione, il RCA subsystem forniva anche i valori di posizione letti dai sensori

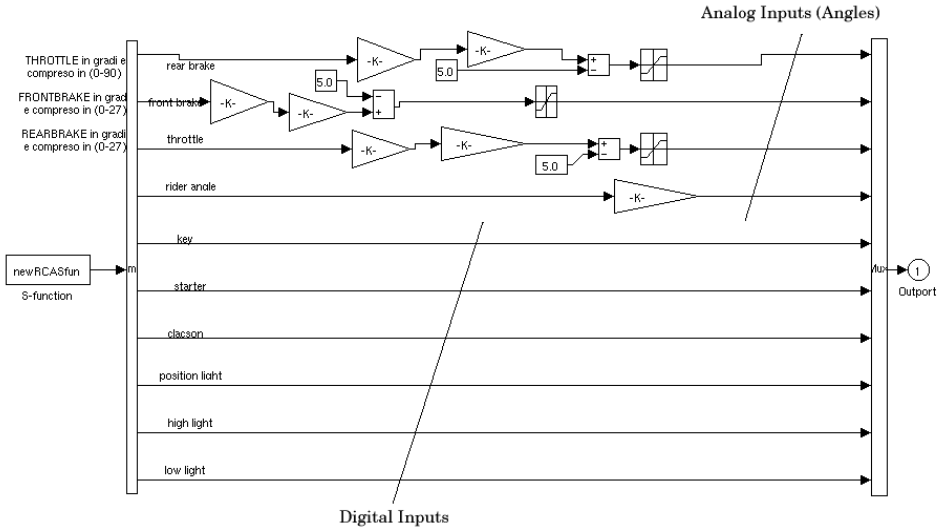

Figura 5.22. L'implementazione del RCA subsystem

5.3.5 La comunicazione

Il modulo *communication module* si occupa di gestire la comunicazione con il sistema newRT. Gestisce sia l'interazione asincrona (messaggi di START e di STOP) che quella asincrona (invio periodico delle letture dei vari sensori, e ricezione del riferimento di posizione della piattaforma).

L'interfaccia esportata del modulo è quella in figura 5.23.

Gli ingressi del modulo sono relative alle letture dei vari sensori degli altri moduli del sistema (posizioni dei pistoni, torsione dello sterzo, letture del modulo RCA) e allo stato in cui si desidera che il modulo si trovi.

Le uscite del modulo invece sono relative ai dati provenienti dal sistema newRT, in particolare i riferimenti di posizione, velocità e accelerazione della piattaforma, e dello sterzo.

Inoltre, sono presenti i normali ingressi e uscite relativi alla gestione delle attività asincrone.

Il modulo prevede due stati di funzionamento:

presenti sui pistoni della piattaforma. Si è deciso di rimuovere questa uscita per una corretta suddivisione dei compiti tra i moduli.

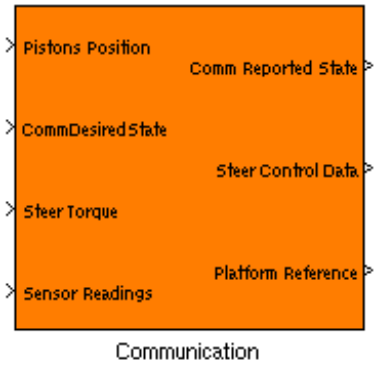

Figura 5.23. L'interfaccia del communication module

OFF: In cui non avviene interazione con il sistema newRT

ON: In cui le letture dei vari sensori vengono inviate al sistema newRT, e le posizioni di riferimento ricevute.

Procediamo adesso ad esaminare il comportamento del modulo.

Durante lo stato di OFF i valori di ingresso sono ignorati, e le uscite sono nulle.

Se viene richiesta al modulo la transizione nello stato di ON, il modulo provvede ad inviare un messaggio di START al sistema newRT. Quando avviene la ricezione del messaggio di Acknowledgement del messaggio di START, il modulo transisce nello stato di ON e notifica l'avvenuta transizione all'esterno. Se i messaggi di conferma non arrivano entro un timeout prestabilito, il modulo transisce nello stato ERROR.

Durante lo stato di ON, ogni dieci passi di simulazione, i dati letti in ingresso vengono spediti al modulo newRT, e i riferimenti di posizione, velocità e accelerazione della piattaforma vengono ricevuti secondo il protocollo descritto nella sezione 5.2.1. Se durante lo stato di ON vengono persi più di 5 pacchetti consecutivi, il modulo va in stato ERROR, e invia un messaggio di STOP al newRT.

Se viene richiesta la transizione dallo stato ON allo stato OFF, il modulo interrompe il flusso di comunicazione con newRT, e invia ad esso un segnale di STOP. La transizione ha successo in seguito alla ricezione del messaggio di conferma dello STOP; in questo caso, la transizione viene notificata all'esterno.

Vediamo adesso una descrizione qualitativa dell'implementazione del modulo, facendo riferimento alla figura 5.24.

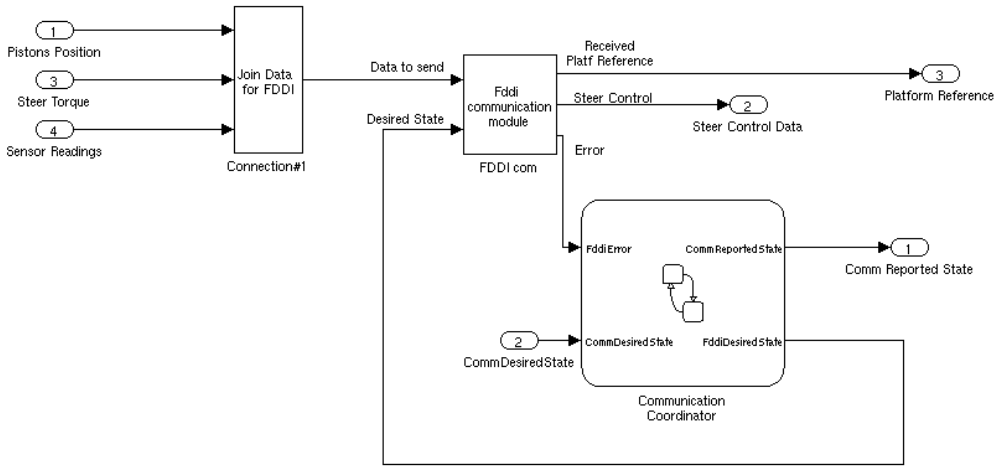

Figura 5.24. L'implementazione del *communication module*

Il *communication module* è composto da un modulo che gestisce la comunicazione via FDDI con NewRT in fase di simulazione (FDDI Com), e di un'altro modulo (realizzato con Stateflow) che regola le attività asincrone del sistema.

L'*FDDI Com* ha ingressi e uscite relativi ai dati da ricevere e da inviare in fase di simulazione, più un ingresso che indica se lo stato del modulo deve essere ON o OFF, e un'uscita su cui vengono notificate eventuali situazioni di errore. È realizzato come S-Function, e ha accesso alle zone di memoria condivisa su cui vengono registrati i dati ricevuti, e su cui vanno scritti i dati da inviare. Se lo stato del modulo è ON, ogni dieci passi di simulazione (quindi con frequenza di 100 Hz), aggiorna la sua uscita con il valore presente nel buffer di ricezione e sblocca il semaforo del task addetto all'invio dei dati.

Il *Communication Coordinator* è un modulo realizzato con il tool Stateflow, ed ha il compito di gestire le comunicazioni asincrone (messaggi di START e STOP, e Acknowledgement) con il sistema newRT, e determinare lo stato di funzionamento del modulo. È dotato di un ingresso su cui legge lo stato desiderato del modulo, di un'uscita su cui riporta lo stato di funzionamento, e di un'interfaccia con il modulo

FDDI Comm, attraverso cui ne attiva il funzionamento e ne registra le eventuali situazioni di errore.

5.3.6 L'interfaccia con il pilota

Facendo riferimento alla struttura gerarchica descritta nella sezione 5.3.1, il sistema MORIS può essere descritto nella sua interezza nei tre stati di funzionamento.

Nel vecchio sistema lo stato di funzionamento era imposto dall'operatore della Console grafica; per aumentare la facilità di uso del nuovo sistema, lo stato di funzionamento deve essere determinato in modo intuitivo direttamente dal pilota. Occorre pertanto definire l'interfaccia del sistema verso il pilota; le transizioni di stato sono determinate dalle azioni naturali che un pilota compie durante l'utilizzo di un motociclo. L'interazione con l'utente è realizzata tramite gli ingressi *key* e *starter*, con gli altri moduli con i relativi *desired state* e *reported state*. In dettaglio:

- Il passaggio della chiave da Off a On, determina la transizione del sistema dallo stato OFF allo stato READY,
- La pressione del pulsante Start per un tempo maggiore di un secondo, determina la transizione del sistema dallo stato READY allo stato RUN,
- Il passaggio della chiave da On a Off, determina la transizione del sistema dallo stato RUN allo stato OFF,
- La pressione del pulsante Start mentre il sistema è nello stato di RUN, determina la transizione nello stato di READY.

C'è bisogno quindi di un modulo collocato tra il livello utente e i vari moduli del sistema. Il modulo descritto è il *Coordinator.*

L'interfaccia con l'esterno è illustrata in figura 5.25.

Il comportamento del modulo coordinator è molto semplice: a seguito di un evento che determina la richiesta di transizione del sistema (si veda l'elenco descritto sopra), il *coordinator* propone la transizione a tutti i moduli di livello inferiore. Una volta che le transizioni di tutti i moduli hanno avuto successo, lo stato globale del sistema viene aggiornato.

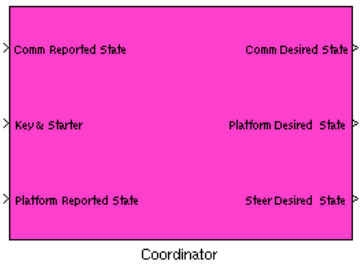

Figura 5.25. L'interfaccia del modulo *Coordinator*

Se si verifica che il *platform subsystem* notifica lo stato ERROR, viene richiesta la transizione del *communication module* in stato di OFF. Viceversa, se è il *communication module* a notificare lo stato di errore, viene richiesta la transizione del *platform subsystem* in stato di OFF.

È possibile adesso tracciare un grafico dei vari livelli di interazione di tutto il sistema (figura 5.26).

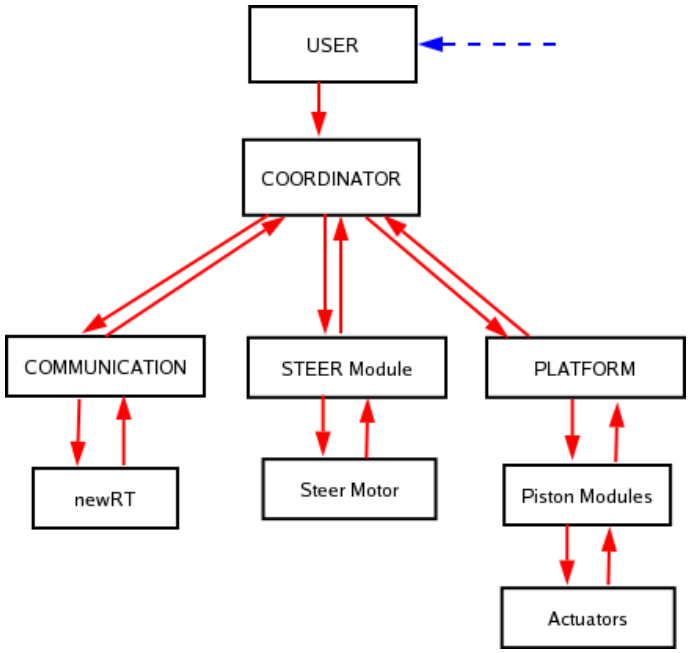

Figura 5.26. I vari livelli di interazione del sistema MORIS

Si noti come il *reported state* del sistema arrivi al pilota indirettamente dal comportamento del sistema (piattaforma in posizione base, centrale, o che risponde agli input), e come il sistema newRT sia definito solo dalle interazioni che ha con il modulo *communication*.

5.3.7 L'integrazione dei moduli del sistema FC

Il nuovo sistema FC è costituito dai moduli descritti finora. La visione globale del sistema è esposta in figura 5.27. Vediamo di riassumerne il comportamento:

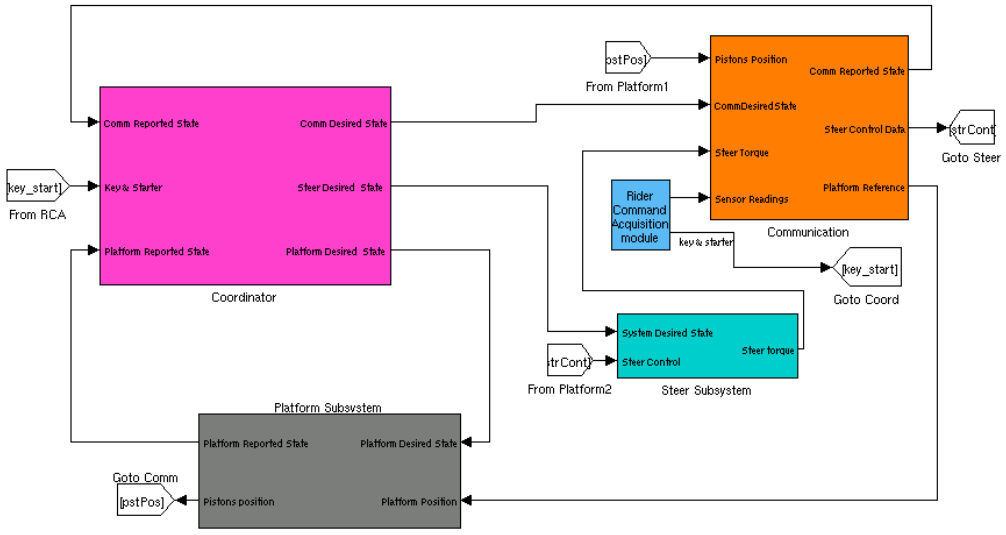

Figura 5.27. Il nuovo sistema FC

I valori letti di chiave e starter vengono utilizzati per determinare lo stato desiderato del sistema globale, nel modulo Coordinator. Sulla base del *desired state*, vengono generati gli stati desiderati dei singoli moduli, a meno di situazioni di errore.

Dal punto di vista *clock based*, è significativo il funzionamento nello stato di RUN: ogni dieci passi di simulazione (100 Hz), i dati provenienti dai vari moduli (sterzo, piattaforma, RCA) vengono raccolti e inviati al sistema newRT, che provvede a elaborarli. Contemporaneamente vengono ricevuti i riferimenti di posizione della piattaforma e dello sterzo relativi ai dati inviati al passo precedente. Questi dati

vengono quindi forniti in ingresso ai moduli relativi, che interagendo con le schede di IO pilotano gli attuatori e il motore collegato allo sterzo.

5.4 Il nuovo sistema RT

Sul nuovo sistema RT sono ospitati i moduli che realizzano il modello dinamico della moto (*dynamic model*), il *washout filter* e la grafica. Il sistema si occupa poi di coordinare le interazioni tra i vari moduli.

Nella versione precedente, RT aveva un ruolo attivo nella determinazione delle fasi del sistema; adesso, il sistema sta in attesa di un messaggio di inizio simulazione, e smette di operare in seguito ad un segnale di fine.

A regime le fasi di esecuzione del sistema sono periodiche, e scandite dalla ricezione dei messaggi dal sistema FC. Il ciclo di operazioni eseguite ad ogni passo sono:

- Prelievo dei dati ricevuti da FC;
- Elaborazione di questi dati da parte del modello dinamico;
- Elaborazione dei dati forniti dal modello dinamico da parte del modulo Washout filter;
- Invio dei riferimenti forniti dal Washout filter al sottosistema FC;
- Operazioni relative alla grafica.

Per il corretto funzionamento del sistema, queste operazioni devono essere terminate in un tempo inferiore a 10 msec: il sistema FC infatti si aspetta di ricevere i nuovi pacchetti con frequenza di 100 Hz.

Durante la fase precedente alla simulazione vera e propria, è possibile prevedere un'interazione a bassa velocità tra i due sottosistemi, ad esempio per operazioni di verifica del funzionamento del sistema, o per permettere all'utente di selezionare, interagendo con la moto, alcuni parametri della simulazione come scenario o la durata della simulazione.

5.4.1 Implementazione di dynamic model e washout

L'analisi dell'implementazione di *dynamic model* e *washout filter* non è stata oggetto di questa tesi, pertanto ci siamo limitati all'utilizzo delle versioni presenti nel vecchio sistema. Nella versione precedente, i moduli erano presenti sotto forma del codice sorgente C generato automaticamente da Matlab; ad entrambi era stata aggiunta una funzione che ne eseguiva un passo di simulazione. Lo stesso codice è stato ricompilato per essere utilizzato all'interno della nuova architettura.

Poiché il codice dei due moduli è stato generato in maniera automatica, alcune variabili globali dei due moduli sono state definite con lo stesso nome. Per evitare sovrapposizioni dei due codici, ed errori in fase di linking dei due distinti file oggetto, si è usato un'opzione del compilatore GCC che permette di allocare le variabili globali nella sezione *data* del file oggetto, invece di generarle come blocchi condivisi.

Si è quindi analizzata l'interfaccia dei due moduli, al fine di poterli utilizzare all'interno del nuovo sistema.

5.4.1.1 Dynamic Model

La funzione che esegue un passo del modello dinamico della moto accetta in ingresso 9 valori di tipo *double*, e fornisce in uscita un vettore di 39 valori di tipo *double*.

In tabella 5.2 sono indicati i significati degli ingressi.

# ingresso	Significato
1	Rotazione acceleratore
2	Posizione ruota posteriore lungo l'asse Z
3	Posizione ruota anteriore lungo l'asse Z
4	Inclinazione della strada lungo l'asse X
5	Inclinazione della strada lungo l'asse Y
6	Forza su freno anteriore
7	Forza su freno posteriore
8	Torsione dello sterzo
9	Inclinazione pilota

Tabella 5.2. Ingressi del dynamic model

Le posizioni delle ruote anteriore e posteriore, e gli angoli di inclinazione della

strada dipendono dalla rappresentazione del circuito. Attualmente questa funzione non è implementata, e questi ingressi sono posti identicamente nulli.

Tra le uscite significative della funzione, si citano le posizioni, velocità e accelerazioni lungo gli assi X, Y, Z, e le posizioni, velocità e accelerazioni angolari secondo gli angoli di Roll, Pitch e Yaw.

5.4.1.2 Washout filter

La funzione che implementa un passo di simulazione dello washout filter accetta in ingresso un vettore di 39 double (dello stesso formato dell'uscita del *dynamic model*), e in uscita fornisce un vettore di 18 *double* (riferimento di posizione, velocità e accelerazione della piattaforma lungo gli assi X,Y,Z e posizione, velocità e accelerazione angolare negli angoli di Roll, Pitch e Yaw).

I riferimenti della piattaforma possono essere inviati al sistema FC.

5.4.2 Integrazione con Matlab

Le azioni che l'applicazione residente su newRT deve compiere sono facilmente implementabili sotto forma di codice C, definendo un corpo *main()* in cui, ciclicamente, vengono chiamate le funzioni necessarie.

Questa soluzione è quella di più facile implementazione, ma non offre grandi possibilità di sviluppo, ne facilità di verifica; si è deciso quindi di utilizzare un approccio misto, utilizzando le funzioni esportate dai moduli *dynamic model* e *washout filter* all'interno di blocchi Simulink.

Si sono definiti, mediante il meccanismo delle S-Function, due blocchi Simulink che hanno gli stessi ingressi e le stesse uscite delle funzioni esportate dai rispettivi moduli; ad ogni passo di simulazione, le funzioni implementano quindi un passo dei due modelli.

Infine, sono stati definiti due blocchi Simulink per implementare rispettivamente la ricezione bloccante da FC e la conseguente trasmissione.

Il blocco che implementa la ricezione ha due uscite: ogni volta che la receive ha successo, la prima viene aggiornata con i valori dei dati ricevuti, e la seconda con il numero di sequenza.

Il blocco che implementa la trasmissione ha invece due ingressi. Ad ogni passo della simulazione (determinato dalla receive), invia i dati letti sul primo ingresso, con il numero di sequenza letto sul secondo ingresso.

In figura 5.28 è illustrato lo schema Simulink che implementa il sistema.

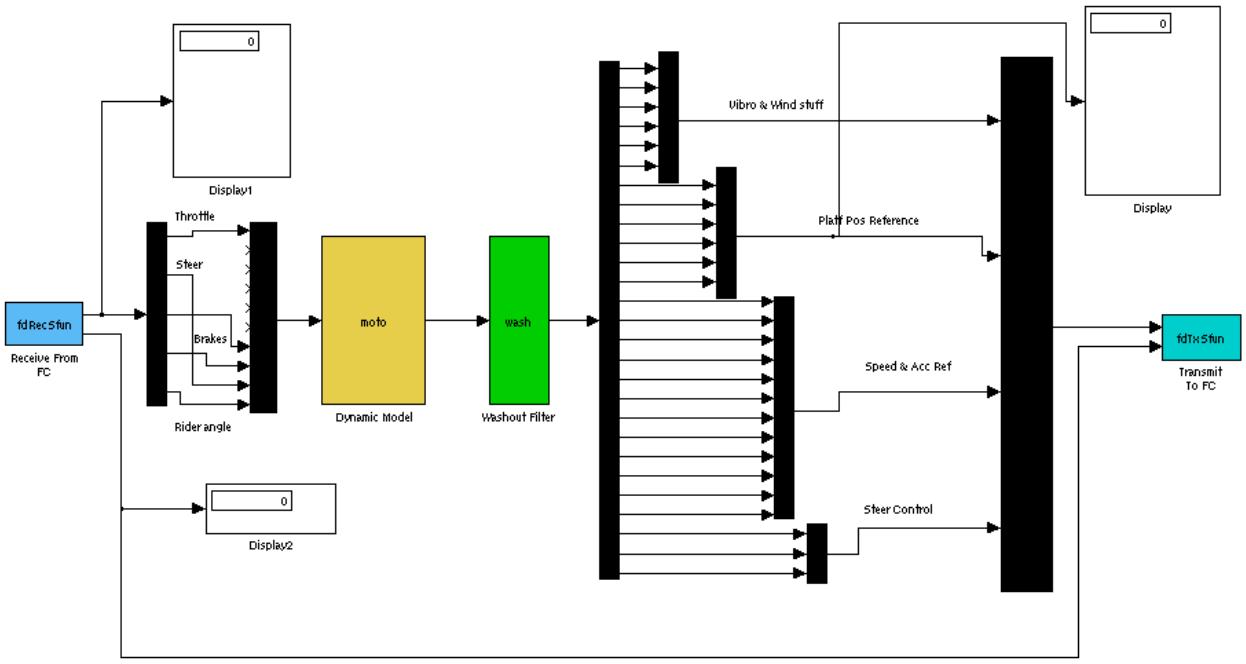

Figura 5.28. Il nuovo sistema RT

I vantaggi di questo tipo di approccio sono molteplici, dal punto di vista della verifica, dell'esecuzione e della portabilità del sistema:

- È possibile testare il funzionamento dei singoli moduli andando a rimuovere i blocchi che implementano ricezione e trasmissione.
- In fase di esecuzione si possono leggere (tramite i blocchi *display* in figura) i valori aggiornati di tutti i valori che i vari moduli si scambiano.
- Lo schema Simulink è generale, e indipendente dal tipo di architettura. Per questo motivo l'applicazione può essere trasferita su piattaforme diverse senza

sforzi di adattamento (anche il codice C che implementa i vari blocchi S-Function utilizza chiamate standard e *platform-indipendent*).

- Data l'elevata modularità del sistema, è possibile aggiungere nuove funzioni e interazioni con altri sistemi in maniera semplice (come ad esempio la grafica).

- Utilizzando il tool Realtime Workshop è possibile generare un'applicazione *stand alone* che permetta al sistema di funzionare su un'architettura Realtime.

5.4.3 Funzionamento soft real time

Durante il lavoro svolto, è stata effettuata la traduzione delle chiamate specifiche di VxWorks (per creare task, utilizzare semafori e strutture condivise) nelle chiamate offerte dalla variante *realtime* di Linux (Linux RTAI). Nel nuovo contesto Real Time non è stato possibile utilizzare l'interfaccia FDDI a causa dell'incompatibilità dei driver, quindi ci si è limitati alla verifica del funzionamento utilizzando il normale sistema operativo Linux.

Il sistema deve essere in grado di effettuare le operazioni richieste nell'arco di 10 msec, per garantire la corretta coordinazione con il sistema FC.

Al fine di verificare che il sistema sia in grado di soddisfare i requisiti di performance, sono state fatti una serie di test delle prestazioni. In particolare, si è verificato il tempo necessario al sistema per:

- Effettuare un passo di simulazione del modello dinamico della moto;

- Effettuare un passo di simulazione del *washout filter*.

In figura 5.29 sono illustrate le somme delle durate in micro secondi delle due operazioni descritte sopra, effettuate su campioni raccolti durante una simulazione di durata pari a 180 secondi.

Come si vede dalla figura, il tempo richiesto è generalmente inferiore a 100 microsecondi (si consideri che il tempo a disposizione di newRT è pari a 10 millisecondi).[3]

[3] I picchi di ritardo più alti sono troppo piccoli per essere attribuiti a deschedulazioni del processo. Sono quindi da imputarsi ad interruzioni software del processo, ad esempio legate al movimento del mouse.

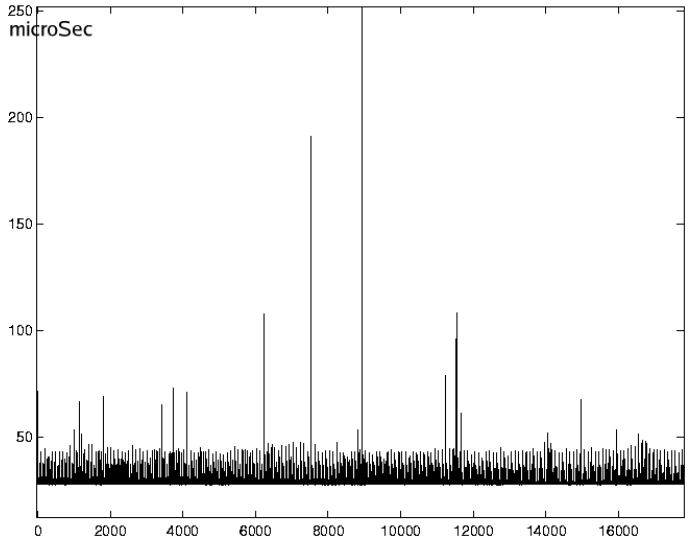

Figura 5.29. Tempo richiesto da dynamic model e strategy manager

Tuttavia, essendo il sistema "non real time", il processo che gestisce queste attività è soggetto a deschedulazione in favore di attività del sistema operativo.

Si può quindi concludere dicendo che il sistema funziona "generalmente bene" a meno di condizioni dovute al fatto che non è attualmente un sistema in tempo reale.

Si consideri inoltre che è ammessa la perdita (o il ritardo) di pacchetti, pur entro certi limiti (numero consecutivo di pacchetti persi non superiore a 5), e che un ritardo di risposta da parte del sistema newRT (o della trasmissione) non è un errore critico, tale da compromettere la sicurezza e l'integrità del sistema generale. Il sistema FC è infatti in grado di rilevare eventuali malfunzionamenti e porre la piattaforma in stato di sicurezza.

5.4.4 Integrazione della grafica

Il sistema prevede lo sviluppo di un nuovo ambiente grafico utilizzando strumenti di sviluppo grafici aggiornati.

La modularità del sistema sviluppato prevede una facile integrazione della grafica, che ha bisogno della posizione del veicolo nell'ambiente virtuale. Questa posizione può essere ricavata in maniera semplice dal riferimento di posizione generato

dal modello dinamico della moto, a meno di eventuali conversioni tra i due sistemi di riferimento utilizzati.

Si è aggiunto quindi un modulo che preleva questi dati con lo scopo di inviarli al modulo che gestisce la grafica. L'implementazione di questo modulo dipenderà dal formato dei dati richiesti dal modulo grafico, e dal mezzo utilizzato per inviare i dati: se ad esempio il modulo della grafica sarà residente sulla stessa piattaforma che ospita il sistema newRT, si utilizzeranno i meccanismi di comunicazione interprocesso; sarà altrimenti necessario utilizzare le direttive per la comunicazione remota (comunicazione su rete).

5.4.5 Possibili sviluppi

L'approccio modulare utilizzato prevede la possibilità di sostituire i moduli che implementano *washout filter* e *dynamic model* con versioni nuove e aggiornate senza apportare ulteriori modifiche al sistema, a patto di mantenerne la stessa interfaccia; ad esempio, è in corso presso il laboratorio PERCRO lo sviluppo di un nuovo modulo Washout Filter riconfigurabile secondo le caratteristiche della piattaforma. Questo modulo è implementato con il tool Matlab/Simulink e, modificandone l'interfaccia in modo che sia compatibile con il sistema newRT, può essere integrato senza sforzo con il sistema newRT.

Utilizzando il tool Real Time Workshop è possibile generare un'applicazione Real Time a patto di utilizzare un'interfaccia di rete supportata. Utilizzando le librerie di traduzione delle chiamate da VxWorks a RTAI, avendo a disposizione un'interfaccia supportata (o adattando i driver dell'interfaccia FDDI presente sul sistema), è possibile generare senza sforzo aggiuntivo l'applicazione real time.

Infine, implementando correttamente il modulo di interfaccia con la grafica, ne permette l'integrazione senza dover apportare ulteriori modifiche al sistema.

5.5 Test di sistema

Durante la fase di test del sistema, sono stati eseguiti i test dei singoli moduli (test di unità) seguendo un'approccio *bottom-up* (partendo cioè dai moduli di livello inferiore, per arrivare al livello più alto).

Una volta verificato il funzionamento dei singoli moduli, si è proceduto alla verifica del funzionamento in caso di integrazione graduale dei vari moduli.

Per i moduli critici per il funzionamento e la sicurezza del sistema, oltre ad effettuare il test in condizioni normali, è stato controllato il corretto comportamento in caso di dati errati in ingresso (*test di robustezza*).

Al fine di poter controllare il funzionamento dei moduli in maniera indipendente dal sistema, si sono utilizzati *driver* e *stub* (vedi sezione 4.1.2).

Il tool Simulink fornisce gli strumenti per generare i *driver* appropriati, e permette la verifica in fase di esecuzione di tutti i moduli Stateflow inseriti nel sistema.

5.5.1 Test del sistema FC

Il corretto funzionamento del sistema FC è indispensabile, perchè deve garantire stabilità e sicurezza nel funzionamento del sistema globale.

5.5.1.1 Test del Piston Module

Facendo riferimento alla figura 5.30, si è andati a pilotare gli ingressi di *Piston Reference*

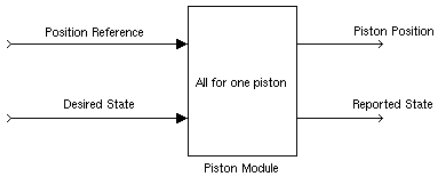

Figura 5.30. Schema per il test del pistone

(riferimento di posizione comandato al pistone) e *desired state* con dei segnali di prova, andando a leggere le risposte del modulo fornite sulle uscite *piston position* e *reported state*.

In particolare, l'ingresso *Desired State* è stato pilotato con i valori indicati in figura 5.31, mentre il riferimento di posizione è stato pilotato con la forma d'onda in figura 5.32.

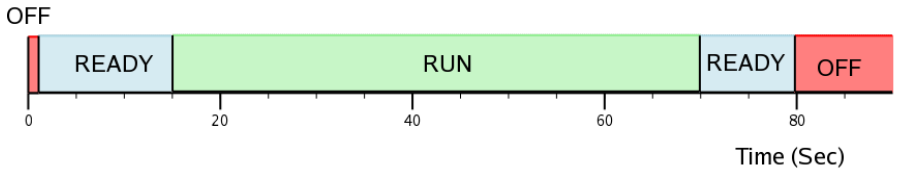

Figura 5.31. Il driver del desired state del piston module

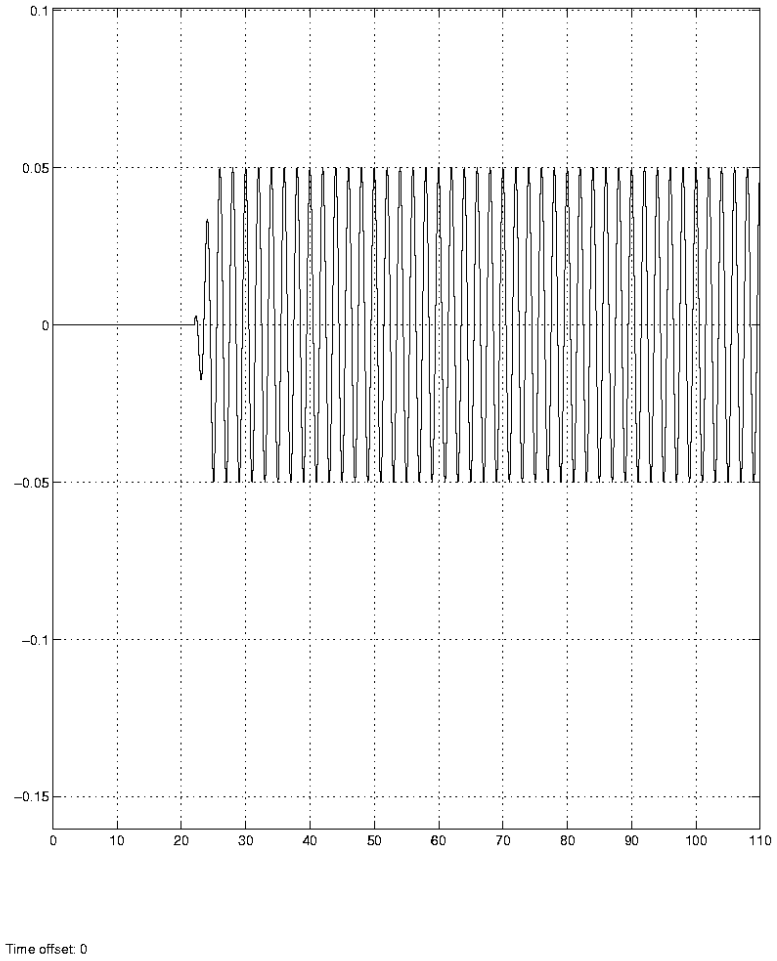

Figura 5.32. Il driver del riferimento di posizione del Piston Module

Si è simulato un normale ciclo di funzionamento (figura 5.31): lo stato del modulo è pilotato da OFF a READY , da READY a RUN, da RUN a READY, e da READY

a OFF.

Il riferimento di posizione fornito è prima nullo, e poi è costituito da una sinusoide di ampiezza 0.05 e frequenza pari al valore *pi greco*. Per poter effettuare la verifica senza dover interagire con l'hardware, l'ingresso e l'uscita della S-Function che interagisce con la scheda di IO sono stati cortocircuitati, in modo che la posizione letta del pistone sia esattamente quella che gli viene comandata.

I valori assunti dall'uscita relativa all'estensione del pistone sono rappresentati in figura 5.33 (A), nella figura (B) è indicato lo stato desiderato del sistema, e in figura (C) è indicato lo stato riportato.

Facendo riferimento alla figura, si possono evidenziare le varie zone di funzionamento:

Prima dell'istante 1, il pistone è in posizione base (-0.31 m); la fase che va da 1 a 2 è la fase di test, in cui il pistone è spostato dalla posizione base per poi tornarvi.

All'istante 2 è richiesto al pistone di andare in stato READY, e quindi assume la posizione centrale (0 m).

All'istante 3 comincia la fase di RUN. È importante notare come, nella commutazione da stato READY a stato RUN, sia richiesta la continuità tra la posizione centrale e il riferimento di posizione fornito in ingresso. Se ci fosse stata discontinuità, il modulo sarebbe andato in stato ERROR.

Nello stato RUN l'uscita del modulo insegue esattamente il riferimento di posizione fornito in ingresso.

All'istante 4, l'ingresso *desired state* assume il valore relativo allo stato READY, e il pistone assume la posizione centrale. Infine, a seguito della richiesta di transizione nello stato OFF, il pistone si porta in posizione base.

In figura 5.34, si può vedere come lo stato riportato del sistema (B) insegua, con i ritardi dovuti alle transizioni tra le varie posizioni da parte del pistone, lo stato in ingresso (A).

Si è provato quindi a fornire una discontinuità della posizione di riferimento rispetto alla posizione centrale, all'istante della transizione di stato da READY a RUN. Come previsto, il modulo riporta lo stato ERROR.

A seguito di questo test, si ritiene che il modulo *piston module* abbia un funzionamento conforme a quello richiesto.

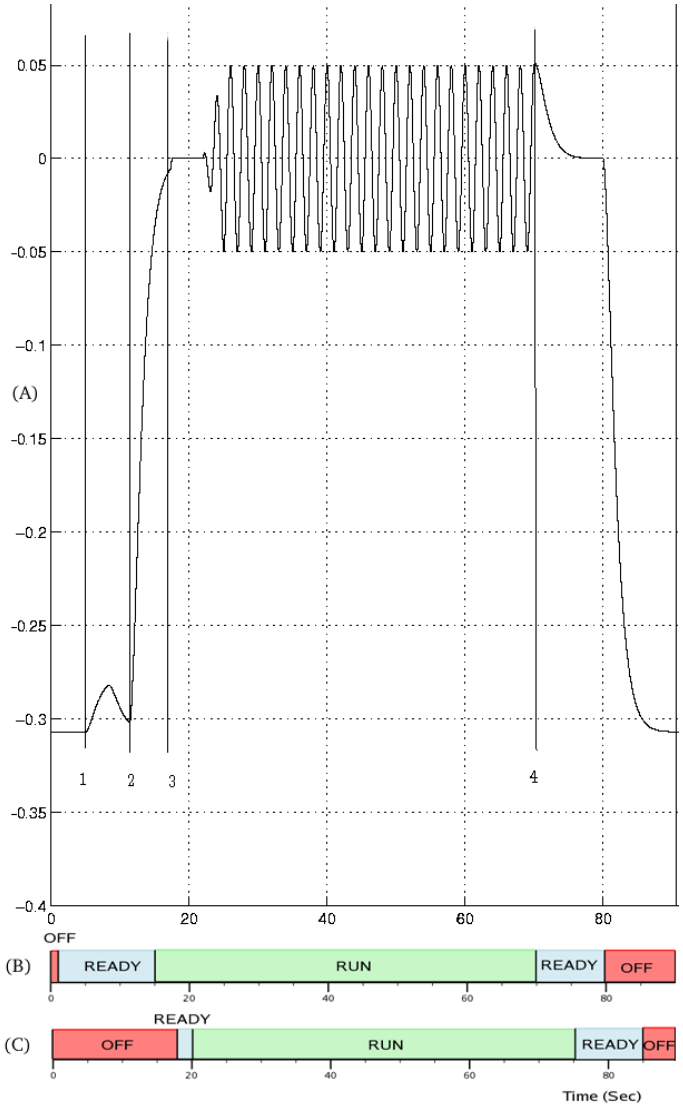

Figura 5.33. La posizione assunta dal pistone (A), lo stato desiderato (B) e lo stato riportato (C)

5.5.1.2 Test del Platform Subsystem

Per il modulo che gestisce la piattaforma, si è operato analogamente al *piston module.*

L'interfaccia del *Platform Subsystem* è illustrata in figura 5.35.

Per riprodurre una normale sessione di simulazione, siamo andati a pilotare lo

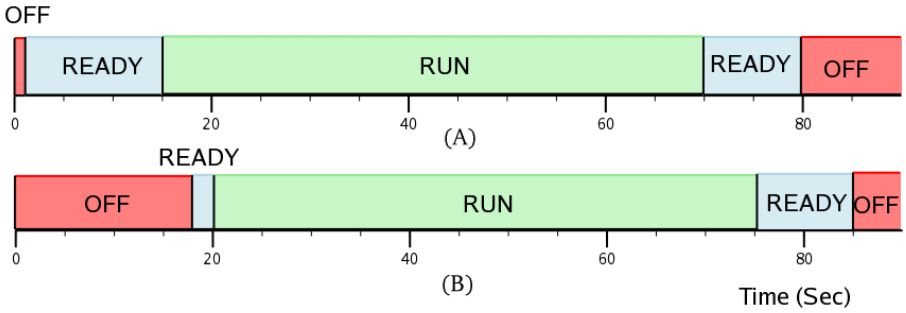

Figura 5.34. Il confronto tra lo stato riportato e lo stato desiderato

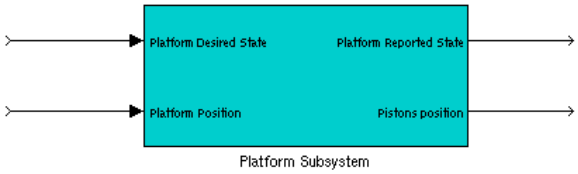

Figura 5.35. Interfaccia di test della piattaforma

stato desiderato del modulo con la stessa forma d'onda utilizzata per il test del *piston module* (figura 5.31).

Per quanto riguarda i riferimenti di posizione invece, si è proceduto a pilotare separatamente i vari assi. Riportiamo in figura 5.36 la posizione dei sei pistoni ottenuta fornendo nell'ingresso relativo al riferimento di posizione della piattaforma lungo l'asse Z, la forma d'onda di figura 5.31.

Si noti che, per la simmetria della piattaforma, le posizioni assunte dai sei pistoni al variare del riferimento lungo l'asse Z (ponendo tutti gli altri ingressi a 0) sono le stesse.

Anche in questo caso sono ben evidenti le varie fasi di funzionamento dei sei pistoni:

1. Test
2. Transizione da OFF a READY

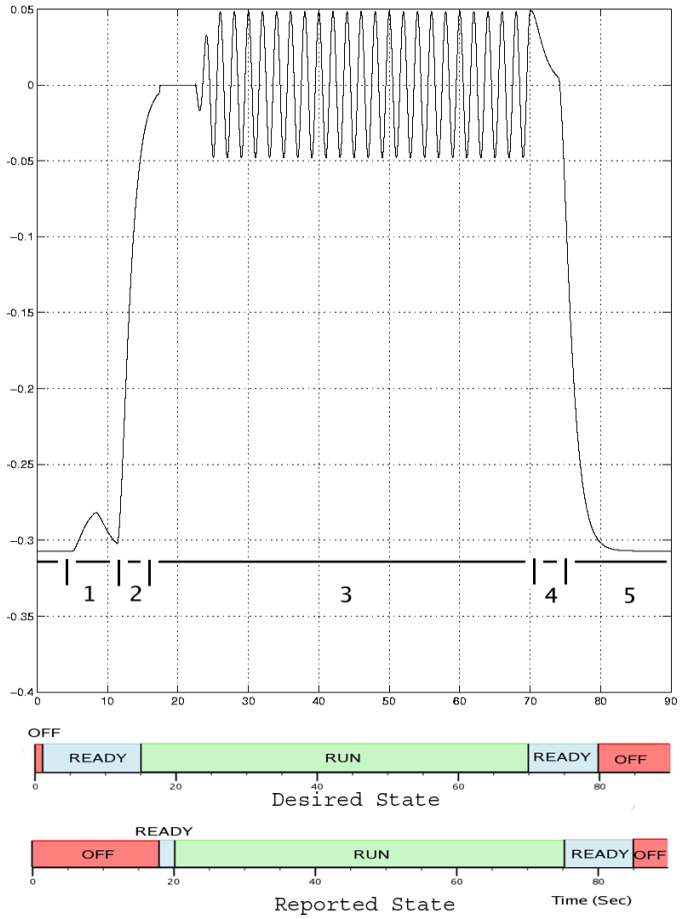

Figura 5.36. La posizione assunta dai sei pistoni della piattaforma

3. Transizione da READY a RUN, e fase di RUN

4. Transizione da RUN a READY

5. Transizione da READY a OFF

Lo stato riportato della piattaforma insegue, con un ritardo dovuto alle transizioni di stato, lo stato desiderato.

Se con la stessa forma d'onda proviamo a pilotare l'asse X, si vede che il comportamento dei sei pistoni durante lo stato di RUN è diverso (disegnati sovrapposti in figura 5.37).

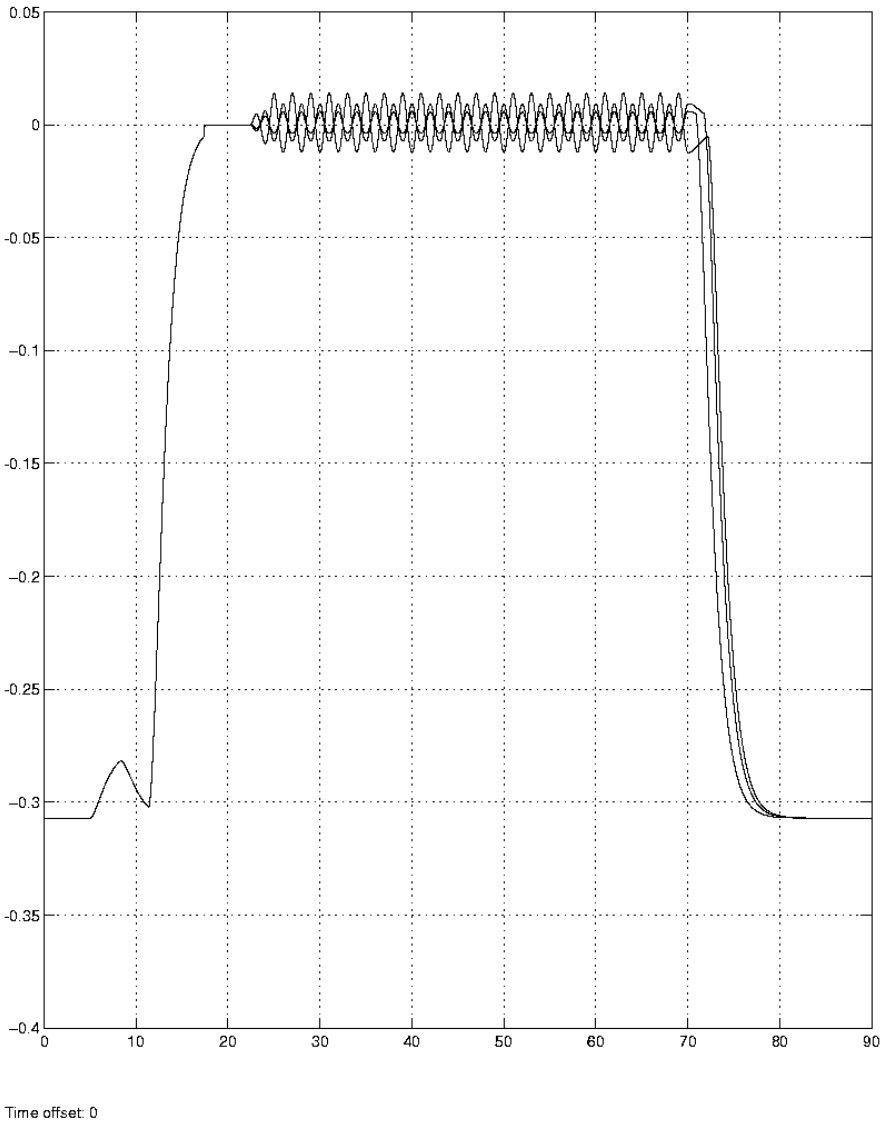

Figura 5.37. La posizione assunta dai pistoni in caso di variazione di riferimento di posizione della piattaforma lungo l'asse X

In figura figura 5.38 è raffigurato il comportamento che il modulo ha a seguito dell'aumento in modo indiscriminato l'angolo di Yaw (rotazione attorno all'asse Z): si vuole generare una configurazione in cui i sei pistoni andrebbero ad intrecciarsi; la piattaforma segnala lo stato ERROR. Questa configurazione viene riconosciuta dal modulo della cinematica inversa, pertanto l'errore è di classe *soft*; appena viene

notificato, la piattaforma viene pilotata in posizione base, senza interromperne il controllo.

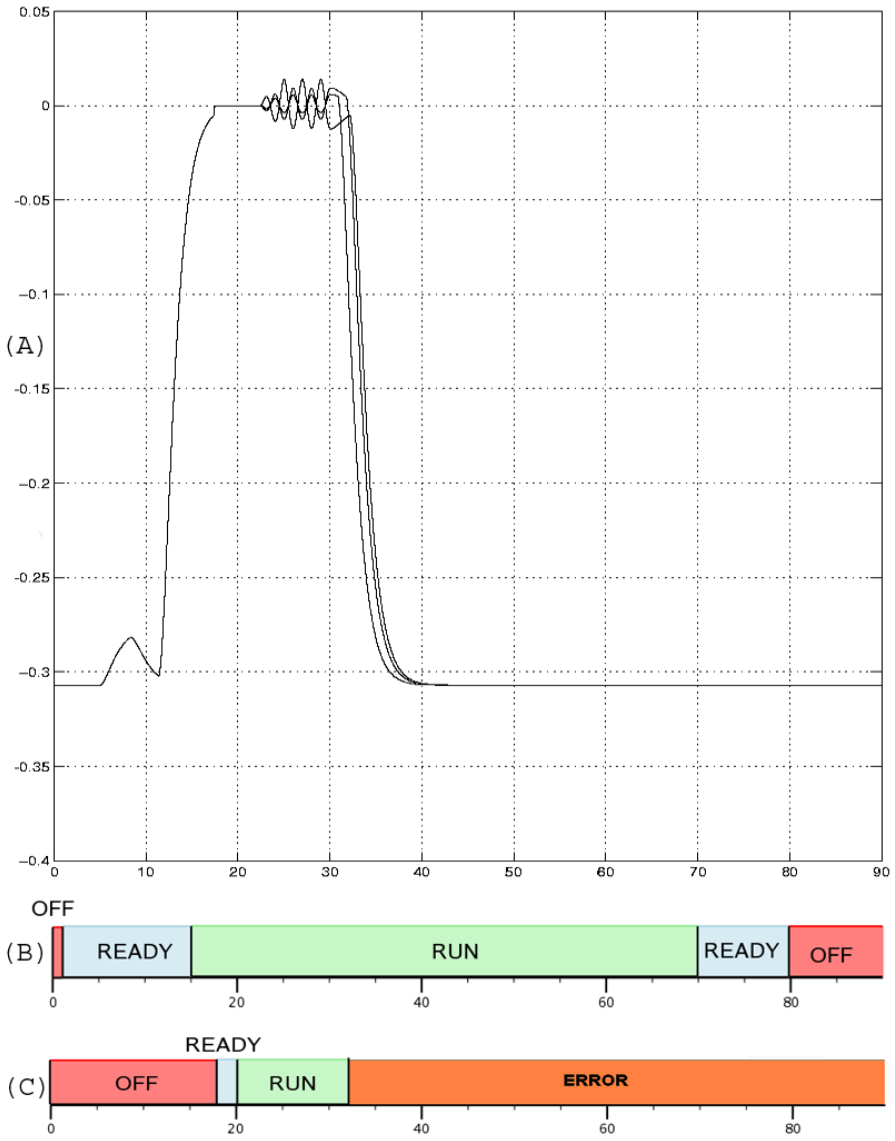

Figura 5.38. Le posizioni dei pistoni in caso di errore della cinematica inversa

5.5.1.3 Integrazione tra Coordinator e Platform Subsystem

Una volta verificato il corretto funzionamento del modulo Platform Subsystem, si è proceduto a verificare il funzionamento del modulo *Coordinator* con gli altri moduli del sistema, in maniera graduale. Il primo passo è stato quello di integrare i moduli *Coordinator* e *Platform Subsystem*, e osservarne il funzionamento dal punto di vista delle interazioni asincrone (i due sottosistemi non interagiscono dal punto di vista del controllo).

Per fare questo test, l'uscita relativa al *desired state* e l'ingresso del *reported state* relative al *communication module* nel *Coordinator* sono state cortocircuitate, in modo da simularne il corretto funzionamento (figura 5.39).

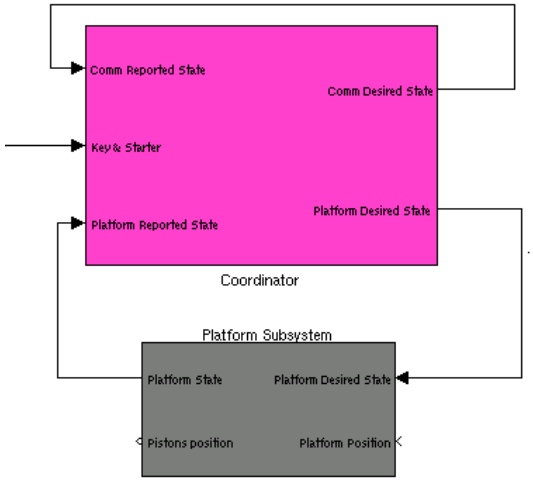

Figura 5.39. L'integrazione del coordinator e del platform subsystem

Si è quindi realizzato un *driver* che fornisce in ingresso al *Coordinator* i valori di chiave e starter che si avrebbero nel caso di una normale sessione di simulazione: l'ingresso relativo alla chiave viene posto al valore 1 (codifica che indica la chiave in posizione ON) al tempo di 10 secondi, e viene inviato un impulso di ampiezza 1 e della durata di un secondo sull'ingresso relativo allo starter al tempo di 20 secondi; infine, dopo 35 secondi dall'inizio della simulazione l'ingresso della chiave viene messo a 0 (chiave in OFF).

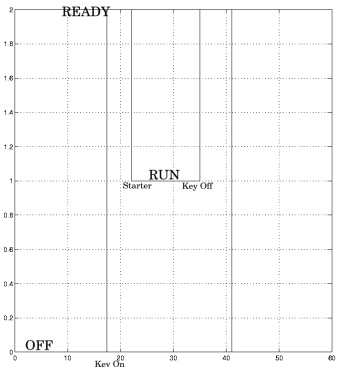

Figura 5.40. Il reported state del platform subsystem

Andando ad osservare i valori assunti dal *reported state* (figura 5.40), si vede che il comportamento assunto dai due moduli è quello aspettato: in seguito all'accensione della chiave, il sottosistema va in stato READY; quindi, in seguito alla pressione dello starter, va in stato di RUN e infine, quando la chiave viene girata in posizione di OFF, il sottosistema va prima in stato READY e poi direttamente in stato OFF.

5.5.1.4 Integrazione di Coordinator e Communication Module

Parallelamente al test di integrazione descritto nella sezione precedente, si è proceduto ad effettuare il test di integrazione tra il modulo Coordinator e il *communication module*.

Il driver che genera gli ingressi relativi a chiave e starter per il coordinator è lo stesso descritto in precedenza. L'ingresso e l'uscita del coordinator relativi rispettivamente a desired state e reported state del Platform Subsystem sono stati cortocircuitati per simularne il corretto funzionamento (figura 5.41).

Sulla macchina newRT poi è stato implementato un semplice programma in C che implementa il seguente flusso di operazioni scritte in pseudolinguaggio:

```
while(<messaggio != START>) ricevi;
<inviaAck();>
while(<messaggio!=STOP){
<ricevi>
```

```
<invia>
}
<inviaAck()>
```

Sugli ingressi del communication module relativi ai valori provenienti da tutti gli altri moduli sono stati immessi segnali nulli.

Andando ad analizzare i valori del reported state del communication module, si è verificato il corretto funzionamento a livello asincrono dei due moduli. Inoltre si è verificato il funzionamento del protocollo di comunicazione tra i due sistemi: il codice che è stato implementato su newRT permette di controllare le varie fasi di esecuzione: si può controllare quindi la ricezione dei messaggi di start e di stop, e il valore dei messaggi di simulazione ricevuti.

Anche in questo caso si è verificato il corretto funzionamento:

- Ricevuto messaggio di start
- Inviato Ack
- Ciclo di ricezione e invio messaggi
- Ricevuto messaggio di stop
- Inviato Ack

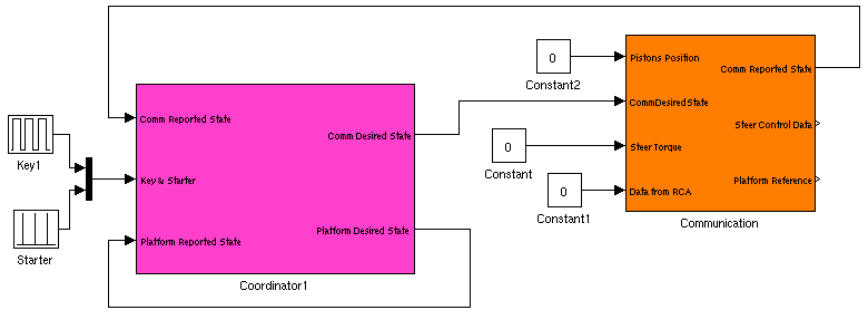

Figura 5.41. Test di integrazione di Coordinator e Communication Module

5.5.1.5 Moduli Steer e RCA

Data l'assenza di coordinazione con il resto del sistema (il modulo RCA è passivo, legge solo dati, mentre lo steer module, nello stato di funzionamento è trasparente rispetto al resto del sistema), non sono stati fatti particolari test di unità dei moduli *steer* e *RCA*. Si è proceduto quindi direttamente all'integrazione nel sistema e al test di integrità del sistema.

Una volta effettuati i test dei componenti ritenuti più critici e delicati per il funzionamento del sistema (coordinator, platform, communication), si è ritenuto opportuno e adeguato procedere all'integrazione totale del sistema, e alla verifica nella sua interezza (*big bang* test).

5.5.2 Test del sistema newRT

Durante la fase di test del sistema newRT ci si è limitati alla verifica del corretto funzionamento del *dynamic model* e del *washout filter,* fornendo in ingresso valori predefiniti e andando ad analizzarne le uscite significative.

Questi test sono stati ridotti considerando che si sono riutilizzati i moduli funzionanti del vecchio sistema.

5.5.2.1 Test del *dynamic model*

Per effettuare il test del modulo che implementa il modello dinamico della moto, si sono posti dei valori costanti negli ingressi relativi all'acceleratore della moto, dello sterzo e dell'inclinazione del pilota, andando ad osservarne le uscite.

Ad esempio, ponendo un valore costante e positivo nell'ingresso relativo all'acceleratore, si è verificato un aumento lineare nella posizione, un valore costante dell'accelerazione e un valore della velocità che in un primo momento sale per poi raggiungere una velocità di regime.

5.5.2.2 Test del *washout filter*

Una volta verificato il corretto funzionamento del modello dinamico della moto, la verifica del corretto funzionamento del washout filter è stata fatta ponendo in

ingresso le uscite del *dynamic model*, e mettendo in ingresso al modello gli ingressi costanti utilizzati al punto precedente. Questa soluzione è stata possibile grazie alla diretta dipendenza dei due moduli (l'uscita dell'uno è posta direttamente in ingresso all'altro) e alla loro indipendenza da tutti gli altri moduli del sistema (se non per gli ingressi del primo e per le uscite del secondo).

I valori forniti in uscita sono risultati essere conformi alle previsioni di comportamento. Questo risultato era prevedibile, dato che non sono stati effettuati interventi di nessun tipo atti a modificare i due moduli descritti.

5.5.3 Test di integrazione tra FC e newRT (input predefiniti)

Per permettere una migliore comprensione del funzionamento del sistema, e una verifica più comprensibile rispetto all'analisi dei valori numerici assunti dai vari segnali in gioco nei vari moduli, è stato sviluppato presso il laboratorio PERCRO un modello grafico della piattaforma di Stewart (figura 5.42), che, ricevendo in ingresso i valori delle estensioni dei sei pistoni, ne mostra la configurazione. Questo modello gira su ambiente operativo Windows e riceve le posizioni dei sei pistoni attraverso protocollo UDP.

Per questo motivo è necessaria (a meno di eventuali *porting* su piattaforma Linux come quella di newRT), un'ulteriore macchina che ospiti questo modulo. Per inviare i dati di cui il modulo ha bisogno, è bastato aggiungere un altro blocco simulink che prelevasse le posizioni dei sei pistoni (provenienti da FC) e che le spedisse via UDP alla macchina che ospita il modello grafico.

Utilizzando questo strumento, è stato possibile avere un'idea *qualitativa* oltre che *quantitativa* del comportamento del sistema in risposta ai vari ingressi forniti.

Ulteriormente, mentre sul sistema FC viene eseguita un'applicazione *stand alone* senza alcuna interfaccia grafica, sul sistema newRT invece si utilizza uno schema Simulink. Per questo motivo, è possibile leggere i valori di tutti i segnali che sono coinvolti nella simulazione.

Il primo test di integrazione dei due sistemi (newRT e FC) è stato realizzato sostituendo i valori provenienti dal modulo RCA con dei valori costanti, andando ad agire sul valore dell'acceleratore.

Sull'ingresso del modulo coordinator di FC corrispondente alla chiave e allo starter, è stata utilizzata la forma d'onda descritta nella sezione precedente, in modo da emulare una sessione di funzionamento.

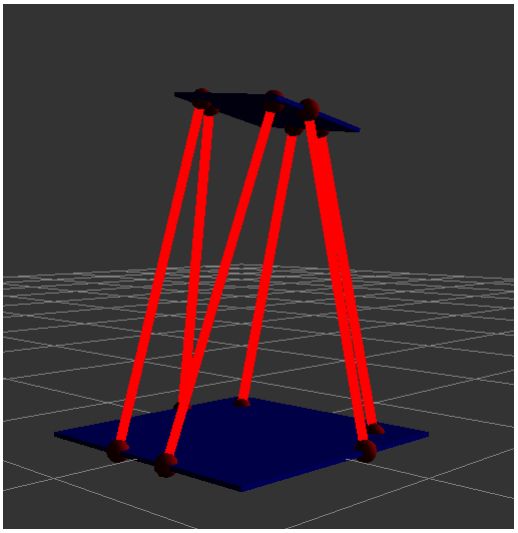

Figura 5.42. Il modello grafico della piattaforma

Dallo schema Simulink su newRT, andando ad osservare i riferimenti generati per la posizione e l'orientamento della piattaforma, si osserva il funzionamento dello washout, che genera un'angolo di Pitch, per poi ritornare in posizione centrale una volta che, secondo il modello della moto, si è raggiunta la velocità necessaria.

Il comportamento della piattaforma è più chiaro osservando il modello grafico della piattaforma, che viene prima inclinato indietro rispetto all'asse longitudinale, per poi tornare in posizione centrale.

5.5.3.1 Prestazioni

Il sistema newRT è attualmente implementato su una piattaforma non real-time. Il processo che realizza l'applicazione è stato portato, tramite la direttiva *nice*, alla massima priorità che un processo utente può avere. Questo non toglie che un qualsiasi processo del kernel di Linux, andando in esecuzione, possa determinare la preemption del nostro processo e quindi un ritardo nella sua risposta verso il sistema FC. Il sistema newRT ha dieci millisecondi di tempo per effettuare le sue operazioni,

e abbiamo visto come il tempo necessario ad eseguire i calcoli necessari a *dynamic model* e *washout filter* siano inferiori a 10 microsecondi. Se il processo viene deschedulato, e invia il messaggio a FC in un tempo superiore a quello consentito, questo messaggio viene ignorato e ritenuto perso. Comunque sia, c'è una certa tolleranza sul numero di pacchetti persi (si ammettono fino a quattro pacchetti consecutivi persi), ed eventuali disservizi del sistema newRT non costituiscono situazioni critiche per la sicurezza del sistema: se la comunicazione non funziona, il *communication module* del sistema FC va in stato di errore, e di conseguenza la piattaforma viene riportata in posizione base.

5.5.4 Test Interattivo

Durante l'ultima parte del lavoro, il mockup della moto è stato smontato dalla piattaforma di Stewart per essere messo su un supporto fisso, per effettuare una serie di test interattivi con la moto senza mettere attivare la piattaforma (figura 5.43).

Figura 5.43. Il mockup della moto messo a terra

Essendo rimasto inutilizzato per il periodo che va dalla fine dell'anno 2000 ad oggi, il sistema meccanico e idraulico richiede una serie di manutenzioni che non sono state ultimate nell'arco della durata del lavoro svolto. Non è stato possibile pertanto utilizzare la piattaforma completa; per questo motivo, i test sono stati limitati alla lettura dei dati provenienti dai vari sensori montati sul mockup.

5.5.4.1 Verifica del funzionamento dei sensori montati sul mockup, e delle schede di IO

Insieme ai driver delle schede di IO, nel precedente sistema è stata sviluppata un'applicazione che, istante per istante, mostra su video i valori letti dalle schede di IO. Utilizzando quest'applicazione si è verificato il corretto funzionamento dei sensori e delle schede (figura 5.44).

```
fold back         over temp         drive up
  1                    1                 1
steer command     current monitor   speed monitor     steer angle
    0                595                  603               8750
rear  brake       front brake       throttle          rider angle
    6                  -230              53                -652
 pve:vibro         km:drv_en         km:drv_rs         unused
   open              open              open              open
moog:out           unused            unused            mts:srv_en
   open              open              open              open
PST:      #1          #2          #3          #4          #5          #6
P/Q:      0           0           0           0           0           0
lvdt:     0.001 0.00    0.00    0.001   0.00    0.00
          key         start       clacson     light: pos  high        low
          off         off         off         off         off         off
0

Commands Available: [V]ibro [R]eset [E]ncoders, [r]elay[x],
                    [S]teer[C]ommand, [PQ]commad[x], [e]xit
Your Command / Value :
```

Figura 5.44. L'applicazione MorisMonitor

5.5.4.2 Il test di tutto il sistema

Si è proceduto quindi all'integrazione di tutto il sistema, esclusa la parte meccanica della piattaforma. È possibile comunque osservare la configurazione che la piattaforma assumerebbe in seguito alle interazioni con il veicolo, utilizzando il modello

grafico sviluppato, e i valori forniti in uscita dal modello dinamico e dal washout, osservando lo schema simulink che implementa l'applicazione sul sistema newRT.

È possibile eseguire una sessione completa di simulazione, eseguendo le azioni definite nella sezione 5.3.6.

Quindi, girando la chiave, e premendo lo starter, il sistema entra in fase di simulazione. Durante la fase di simulazione, interagendo con il veicolo (con acceleratore, freni e inclinometro) è possibile osservare il movimento della riproduzione della piattaforma e tutti i parametri di simulazione nello schema Simulink.

Portando di nuovo la chiave in posizione OFF, la simulazione viene terminata.

5.6 Esempio di interazione con un modulo di grafica

Presso il laboratorio PERCRO è stata sviluppata una riproduzione grafica di una città virtuale (figura 5.45). Attualmente l'applicazione gira su piattaforma Windows, ed è stata modificata per accettare in ingresso via UDP, su una determinata porta, i valori di posizione e orientamento nello spazio del punto di vista. Utilizzando i valori forniti dal modello dinamico della moto, e applicandovi le opportune trasformazioni, è possibile muovere punto di vista all'interno dell'ambiente virtuale interagendo con il veicolo (figura 5.46).

L'integrazione del sistema attuale con questo modulo è semplice: basta utilizzare il blocco simulink realizzato per interagire con il modulo grafico, descritto nella sezione 5.4.4, e modificarne l'implementazione in modo da convertire il sistema di riferimento della moto in quello della grafica.

Figura 5.45. La riproduzione della città virtuale

Figura 5.46. Un esempio di interazione con la grafica

Capitolo 6

Conclusioni

Prima dell'inizio di questo lavoro, il sistema MORIS era funzionante ma poco flessibile ad eventuali modifiche o variazioni di configurazione. Il codice che costituiva l'applicazione residente su FC era statico e "chiuso". A causa dell'utilizzo di tool superati, e della particolare struttura del codice con cui erano state sviluppate le applicazioni residenti sui vari nodi del sistema, qualsiasi aggiornamento mantenesse intatta l'architettura software/hardware sarebbe stato faticoso, avrebbe richiesto una fase di verifica lunga e difficile, e sarebbe stato a sua volta poco flessibile.

A seguito delle problematiche riscontrate, sono state valutate le possibili soluzioni ed è stata definita riprogettata l'architettura software e hardware per il sistema. La nuova architettura è costituita dai due nodi essenziali *newRT* e *FC* compatibili con i vincoli di progetto rilevati; in particolare, per il sistema FC è stata mantenuta la vecchia architettura Hardware.

Il sistema newRT è stato equipaggiato con versioni aggiornate dei tool Matlab/Simulink/Stateflow e dell'ambiente di sviluppo GNU per lo sviluppo delle applicazioni residenti, e che sono stati quindi personalizzati per permettere anche lo sviluppo di applicazioni per FC.

Per lo sviluppo della nuova applicazione che gestisce il controllo della piattaforma è stato utilizzato un approccio modulare, che facilita la modifica, la manutenzione e la verifica delle singole componenti del sistema in maniera del tutto indipendente, sia a livello di interazioni *intermodulari* (con gli altri moduli del sistema) sia a livello

di interazioni *intramodulari,* vale a dire tra i vari livelli di profondità che costituiscono un modulo. È possibile rimuovere o cambiare l'implementazione interna di un modulo senza dover modificare il resto del sistema.

Il vecchio codice che implementava la macchina a stati di FC, e che ostacolava una chiara visione del sistema e delle interazioni tra i moduli, oltre che la modularità stessa, è stato rimosso. La gestione degli eventi asincroni adesso è demandata a moduli realizzati con il nuovo tool *Stateflow*, che permette uno sviluppo intuitivo e verificabile di automi a stati finiti. La modularità del sistema è adesso garantita anche dal punto di vista delle interazioni *asincrone*, grazie all'approccio gerarchico utilizzato. Di ogni modulo sono adesso evidenti gli ingressi relativi ai flussi di dati sincroni e asincroni: questo favorisce la comprensibilità del funzionamento del sistema, e la manutenzione dello stesso, nella sua totalità e nei singoli moduli.

Infine, la macchina FC è stata resa autonoma da macchine esterne a seguito della memorizzazione del kernel e dei moduli sviluppati nella memoria flash di cui è equipaggiata.

La struttura hardware del sistema RT è stata sostituita con un nuovo Personal Computer ad alte prestazioni. I codici (relativi a modello dinamico e washout filter), scritto per essere eseguiti nel sistema operativo VxWorks sono stati portati alla nuova architettura e compilati in modo da generare applicazioni per il nuovo sistema operativo; sono stati inoltre integrati con Matlab/Simulink al fine di aumentare la verificabilità del sistema, permettere l'aggiunta di nuovi moduli (e funzionalità) in maniera semplice, e l'aggiornamento di quelli esistenti. Su *newRT* si è realizzato un servizio di tipo *best effort* in risposta alle richieste di FC, utilizzando il normale sistema operativo Linux. La profilazione delle operazioni da effettuare durante il funzionamento, e la stima dei ritardi dovuti alla comunicazione, ha mostrato che le prestazioni così ottenute sono in linea con le specifiche di funzionamento.

Un nuovo approccio a più livelli di priorità ha consentito di rimuovere alcuni vincoli stringenti di comunicazione; in conseguenza, la comunicazione tra i due sistemi è stata riscritta utilizzando il protocollo UDP e le chiamate standard su socket. In questo modo la portabilità è aumentata e il codice è conforme agli standard di programmazione. È stato definito un protocollo per determinare l'inizio e la fine

della simulazione, e in grado di verificare la perdita di pacchetti durante la fase di simulazione. Il corretto funzionamento del protocollo è stato verificato con successo.

La modularità del sistema ne permette una facile riconfigurabilità, modificando soltanto i moduli necessari, o aggiungendone altri necessari per svolgere nuove funzioni. Essendo la maggior parte delle applicazioni sviluppate ad alto livello su Matlab/Simulink, modificando lo strato inferiore è possibile utilizzare le stesse applicazioni su altri tipi di architettura.

La facilità di uso del sistema è notevolmente aumentata: non è più necessaria la presenza di un operatore esterno che gestisca le varie fasi della simulazione, che sono comandate direttamente dal pilota attraverso l'interazione diretta con il veicolo.

Sono stati eseguiti vari test di funzionamento con ingressi predefinti, sia a livello di singoli moduli, che di sistema globale, con risultati soddisfacenti. Utilizzando un modello ad elementi finiti della piattaforma è stato possibile verificare il comportamento del sistema idraulico anche senza accedere al sistema meccanico, attualmente in fase di manutenzione.

Dopo aver verificato il funzionamento dei singoli moduli software, si è proceduto alla fase di test interattiva con il mockup della moto. Si è accertato il funzionamento dei segnali provenienti dalla moto e, utilizzando il modello grafico della piattaforma si è osservato in modo chiaro e comprensibile il comportamento richiesto al sistema meccanico in seguito all'interazione con la moto.

Infine, è stata effettuata l'integrazione del sistema newRT con un modulo preposto alla riproduzione del feedback visivo verso il pilota. Questa integrazione è risultata semplice grazie alla struttura modulare con cui è stato progettato il sistema.

6.1 Sviluppi Futuri

Il sistema MORIS è stato riprogettato per essere totalmente flessibile a nuove possibili soluzioni e implementazioni dei singoli moduli. In particolare si prevede la sostituzione del modulo washout filter con quello in corso di sviluppo presso il laboratorio PERCRO.

I test effettuati hanno coinvolto anche le componenti non verificabili sperimentalmente ed assicurano con buona certezza che, una volta effettuata la manutenzione necessaria, il sistema possa tornare a funzionare integrato con il sistema elettromeccanico.

Il sistema newRT attualmente gira su un sistema operativo non real time. Nel corso del lavoro è stata predisposta una libreria per la conversione delle chiamate VxWorks in chiamate per Linux RTAI: implementando un driver dell'interfaccia di rete compatibile con Linux/RTAI, l'applicazione di newRT sarebbe eseguita su un sistema real-time, immune quindi ai problemi di ritardo dovuti alle deschedulazioni del processo.

A causa dei requisiti software dell'applicazione che realizza la grafica, l'interazione tra newRT e il modulo grafico è effettuata in remoto (ethernet). Un porting (già in corso di sviluppo) dell'applicazione per renderla compatibile con Linux ne permetterebbe l'esecuzione in locale, eliminando la necessità di un'ulteriore macchina. A livello di sistema, l'implementazione realizzata consentirà tali modifiche in maniera immediata, sostituendo l'indirizzo del modulo che ospita la grafica con l'indirizzo locale. Può essere poi sviluppato un ambiente virtuale *ad hoc*, con varie tipologie di scenario a disposizione, che riproduce percorsi e situazioni utili agli scopi del simulatore.

Utilizzando una descrizione dei vari segmenti dell'ambiente virtuale in grado di rilevare le proprietà stradali (quote delle ruote lungo l'asse Z, angolo di inclinazione lungo l'asse Y), l'integrazione del nuovo ambiente con il modello dinamico e quindi con tutto il sistema può avvenire in maniera naturale.

Bibliografia

[1] F. Barbagli D. Ferrazzin. A simplified dynamic model for a motorcycle. Technical Report MO-SS-ME-D-DM-01, PERCRO, Scuola Superiore S.Anna, Pisa, Italy, Sept. 1999

[2] F. Salsedo D.Ferrazzin and Others. Strategy manager as a design tool in a motorcycle driving simulator. In *Proceedings of 32nd ISATA*, *Wien*, number 99SI26, 1999

[3] D. Ferrazzin et al. Strategy Manager Implementation in a motion based two wheeled veichle simulator. In *32nd ISATA*, pages 281-288, June 1999

[4] D. Ferrazzin. Rider command acquisition subsystem. Technical Report MO-SS-ME-D-RCA-01, PERCRO, Scuola Superiore S.Anna, Pisa, Italy, Sept. 1997

[5] D.Ferazzin and others. MORIS Project Final Report, PERCRO, Scuola Superiore S.Anna, Pisa, Italy,Jan. 2001

[6] "MORIS PROJECT -Motorcycle Rider Simulator; Global Architecture an Subsystems Specification", MORIS Deliverable D3.2, PERCRO, Scuola Superiore S.Anna, Pisa, Italy, May. 1997

[7] C.A. Avizzano, F. Barbagli, D. Ferrazzin and M. Bergamasco, Washout filter design for a motorcycle simulator, IEEE VR2001 (2001)

[8] D. Ferrazzin, F.Barbagli, C.A. Avizzano, G. Di Pietro and M. Bergamasco, Designing new commercial motorcycles through a highly reconfigurable virtual-reality based simulator, Advanced Robotics number 4 2003, pages 293-318

[9] G. Di Pietro, Studio e implementazione di strumenti e metodi per la prototipazione rapida di applicazioni in tempo reale per il controllo di sistemi robotici, tesi di laurea, feb. 1999

[10] D. Ferrazzin, Study and design of motion base for driving simulators, Doctoral

thesis, Scuola Superiore S. Anna

[11] F. Barbagli, C.A. Avizzano, Strategy Manager Unit. Technical Report MO-SS-CO-D-SM-01, PERCRO, Scuola Superiore S.Anna, Pisa, Italy, Jul. 2000

[12] C.A. Avizzano, Frame Controller Software, Technical Report MO-SS-ME-V-SM-0, PERCRO, Scuola Superiore S.Anna, Pisa, Italy, Jul. 2000

[13] G. Borlizzi, Introduzione al bus VME, ARTS Lab, Rapporto interno, Scuola Superiore S.Anna, Pisa, Italy, Jun. 1994

[14] A. Domenici, Dispense del corso di Ingegneria del Software, Università di Pisa, Facoltà di Ingegneria, 2000

[15] L. Peterson, Computer Networks: a system approach, Ed.Kaufmann, may 2003

[16] "AXPvme Single-Board Computer User Guide", Digital Equipment Corporation, Maynard, Massachussets, Sept 1995

[17] "DECchip 21066 Alpha AXP Microprocessor Data Sheet", Digital Equipment Corporation, Maynard, Massachussets,May 1994

[18] "VxWorks: Introduction to VxWorks", Digital Equipment Corporation, Maynard, Massachussets, Nov. 1994

[19] "VxWorks Programmer's guide", Digital Equipment Corporation, Maynard, Massachussets, March 1992

[20] "Alpha VxWorks Hardware Supplement", Digital Equipment Corporation, Maynard, Massachussets, July 1995

[21] "Simulink User's Guide (ver 5.1)", MathWorks Inc., 2003

[22] "Real Time Workshop User's Guide (ver 5.0)", Mathworks Inc., 2003

[23] "Stateflow User's Guide (ver 5.1)", MathWorks Inc., 2003

[24] VITA (VME International Trade Association) Home page. Web page http://www.vita.com/vmefaq.html

[25] X86ToAlphaCross Compiler How To, Web page http://www.cse.unsw.edu.au/~cgray/crossdev/x86-alpha-xcc.html

[26] Alpha-Linux Homepage, Web page http://www.alphalinux.org/

[27] Linux VME How To, Web page http://www.tldp.org/HOWTO/VME-HOWTO.html

[28] RTAI - Realtime Application Interface Homepage, Web page http://www.rtai.org