UNIVERSITÀ DI PISA

Scuola di Ingegneria

Corso di Laurea Magistrale in Ingegneria Robotica e dell'Automazione

Tesi di Laurea Magistrale

Progettazione e sviluppo di un controllore tempo reale Multi-OS per bracci a cedevolezza variabile.

Relatore:
Prof.ssa Lucia Pallottino
Ing. Riccardo Schiavi

Controrelatore:
Prof. Antonio Bicchi

Candidato:
Mariella Foddai

Sommario

La richiesta di applicazioni sempre più complicate e che soddisfino determinati requisiti porta chi sviluppa tali applicazioni a cercare metodi alternativi che gli consentano di visualizzare meglio l'intero modello da gestire, evidenziando non solo le relazioni con l'ambiente esterno, ma anche le relazioni tra i vari sottomodelli da cui è composto. Oltre a questo si richiede anche l'utilizzo di dispositivi che siano in grado di garantire determinate prestazioni. L'approccio "Model-Based" si pone come una risposta al modo di affrontare la gestione di problemi che risultano sempre più complessi, fornendo una maggiore facilità di aggiornamento dei sistemi e una maggiore facilità nell'ottenere altri sistemi derivati con funzionalità estese.
La comodità principale della progettazione "model-based" risiede nel fatto che chi sviluppa il progetto non ha la necessità di concentrarsi sulla fase di programmazione ma piuttosto sulla progettazione, consentendo al programmatore di concentrarsi sulle funzionalità piuttosto che sulle funzioni.
La board UdooNeo, invece, caratterizzata da un chip con due processori si propone come dispositivo in grado di garantire determinate prestazioni, consentendo la gestione di applicazioni a frequenza diversa.
Partendo da queste considerazioni, questo lavoro di tesi mostra come generare automaticamente del codice partendo dallo schema "Model Based" dell'intero sistema, utilizzando il generatore di codice E4CoderCG, per la board UdooNeo in modo da controllare un braccio a cedevolezza variabile realizzato con i dispositivi *qbmove*.

Indice

Capitolo 1

Introduzione

Obiettivo della tesi è quello di controllare un braccio robotico a cedevolezza variabile, con 4 gradi di libertà, tramite codice generato automaticamente per un sistema multi-OS, a partire da uno schema a blocchi del sistema.
La progettazione "model-based" è un metodo matematico di affrontare i problemi connessi principalmente con la progettazione di controlli complessi, con l'elaborazione dei segnali e dei sistemi di comunicazione in ambiente automotive, aerospaziale e più in generale estendibile ad ogni ambito industriale. Ma l'applicazione più importante della progettazione model-based si riscontra nello sviluppo e nella progettazione di software embedded.
La progettazione model-based fornisce un approccio efficace per stabilire un completo quadro della comunicazione in tutto il processo di progettazione supportando il diagramma di sviluppo a "V", ovvero il modello di progettazione di un software che anzichè essere un modello a cascata, dopo la fase di progettazione risale verso l'alto (da cui il nome modello a "V"), mostrando per ogni fase del processo di sviluppo la relazione tra la fase di sviluppo del software e la fase di test.[12]
Nella progettazione "model-based" dei sistemi di controllo, lo sviluppo si ottiene attraverso quattro passi:

- modellazione del sistema da controllare;
- analisi e sintesi di un controllore per il sistema;
- simulazione del modello del sistema e del controllore;
- integrazione di tutte queste fasi implementando il controllore.

Il paradigma di progettazione "model-based" è diverso dalla metodologia di programmazione tradizionale; invece di utilizzare strutture complesse e elevate quantità di codice software, i progettisti possono utilizzare la progettazione "model-based" per definire modelli con caratteristiche funzionali avanzate utilizzando differenti blocchi per il tempo continuo e per il tempo discreto. Modelli così costruiti, utilizzati con strumenti di simulazione portano ad una più rapida prototipizzazione, test dei software e verifiche migliori.
I principali vantaggi della progettazione "model based" risultano, quindi, nel fatto che tale progettazione offre un ambiente di progettazione comune che facilita la comunicazione, l'analisi dei dati e la verifica dei sistemi tra i gruppi di sviluppo; in questo modo, inoltre, gli sviluppatori possono individuare e correggere errori iniziali di progettazione del sistema, tenendo conto del tempo e dell'impatto finanziario nelle modifiche che devono essere minimi. Infine, un altro vantaggio si riscontra nella facilità di aggiornamento dei sistemi e nella facilità con cui si possono ottenere altri sistemi derivati con funzionalità estese.
La comodità della progettazione "model-based" risiede nel fatto che chi sviluppa il progetto non ha la necessità di concentrarsi sulla fase di programmazione ma piuttosto sulla progettazione, consentendo al programmatore di concentrarsi sulle funzionalità piuttosto che sulle funzioni. Questo permette anche a chi non ha un'elevata conoscenza dei linguaggi di programmazioni di poter ricavare in autonomia il codice sorgente per un sistema ottenuto

attraverso la progettazione model-based.
Con il termine multi-OS si indica la coesistenza di due o più sistemi operativi all'interno di uno stesso dispositivo. Nella maggior parte dei casi all'avvio del dispositivo l'utente può scegliere quale sistema mandare in esecuzione a seconda delle proprie esigenze. Un altro modo in cui si può realizzare un ambiente multi-OS è attraverso la macchina virtuale, ambiente nel quale è possibile installare più di un sistema operativo, in questo modo l'utente, una volta avviato il dispositivo con il sistema operativo principale, potrà scegliere quale altro sistema operativo avviare. Utilizzando una macchina virtuale la coesistenza di due sistemi operativi è più visibile e più contemporanea, in quanto la macchina virtuale risulta essere un processo del sistema operativo principale.
La diffusione di sistemi multi-OS si ha in seguito alla necessità di garantire un maggiore e sempre affidabile controllo in dispositivi e macchine che attualmente risultano sempre più complesse; in particolar modo, con lo sviluppo di piattaforme multi-OS si ha la possibilità di monitorare più processi con periodicità diversa a seconda delle esigenze. Ad esempio, nel caso di automobili moderne in cui la maggior parte dei sistemi di controllo sono computerizzati e dotate delle moderne tecnologie si ha la necessità di garantire l'efficienza e la robustezza nell'esecuzione dei sistemi di controllo; si ha la necessità che i task che si occupano del controllo del sistema di frenata o dell'ABS non vengano interrotti rallentati nel loro funzionamento da task che, ad esempio, si occupano della riproduzione video o della riproduzione audio.
Nell'ottica di questo nuovo tipo di richieste si colloca lo sviluppo di nuove boards multiprocessore pensate proprio per lo sviluppo di ambienti multi-OS pensati non solo per l'ambiente automotive ma anche per il costante sviluppo di applicazioni legate all'IoT, Internet of Things, in crescente sviluppo anche nell'ambiente quotidiano basta pensare agli elettrodomestici di ultima generazione o agli smartwatch.
All'interno di queste nuove boards si colloca la scheda UdooNeo, che unisce le caratteristiche principali di Arduino, il pinout e la semplicità in fase di programmazione, con la possibilità di programmare in qualsiasi linguaggio, caratteristica principale di una Raspberry-Pi, a cui si aggiunge la presenza di un chip caratterizzato da due processori, un cortex-M4, su cui si trova il sistema operativo tempo reale e un ARM cortex-A9 su cui si trova un sistema operativo general purpose; in questo caso la coesistenza tra i due sistemi operativi è ancora più evidente in quanto non è necessario avviare una macchina virtuale per avere in esecuzione contemporaneamente i due sistemi operativi ma una volta avviata la scheda entrambi i sistemi operativi risultano essere attivi. Questo lavoro di tesi si introduce in questo contesto con l'obiettivo di generare automaticamente del codice per un sistema realizzato con un approccio Model-Based, per un sistema multi-OS. In particolare lo schema a blocchi del sistema è stato realizzato su ScicosLab, che supporta il tool E4Coder utilizzato per generare il codice sorgente per ambiente Linux che dove essere compilato ed eseguito sulla board UdooNeo.
Il braccio che si vuole controllare è, invece, realizzato con i dispositivi qbmove, ovvero dei dispositivi a cedevolezza variabile che attraverso delle flange possono essere montati orientando l'asse di rotazione del dispositivo a piacere in base alla configurazione desiderata per il braccio.

Capitolo 2

"Model based systems"

Il "Model based systems engineering" (MBSE) è una metodologia del sistema ingegneristico che si focalizza sulla creazione e sulla valorizzazione del "domain models", ovvero di un modello concettuale come mezzo principale per lo scambio di informazioni maggiore rispetto a quello basato sulla semplice documentazione.
Con il termine sistema ingegneristico si intende il campo interdisciplinare dell'ingegneria che si focalizza su come progettare e gestire complessi sistemi e il loro ciclo di vita. In questo modo, requisiti ingegneristici quali l'affidabilità, la logistica, il coordinamento delle diverse componenti, il test e la valutazione, la manutenibilità e molte altre discipline necessarie per il successivo sviluppo del sistema, la progettazione, l'implementazione, che in sistemi di grandi dimensioni sono difficili e complessi da gestire, diventano più semplici da gestire.
Il "domain models" è un modello concettuale, un modello di astrazione che descrive aspetti legati al comportamento del sistema, ai dati coinvolti, agli aspetti specifici della sfera di conoscenza, dell'influenza, o dell'attività. Tale modello può quindi essere utilizzato per risolvere i problemi connessi a tale dominio; esso è una rappresentazione di significativi concetti reali pertinenti al dominio che devono essere modellati nel software.
La modellazione viene fatta attraverso un modello ad oggetti composto da uno strato di basso livello a cui si aggiunge sopra uno strato ad alto livello gestito da API per l'accesso ai dati e al comportamento del modello. Recentemente, l'attenzione del "Model-based" si è focalizzata sul coprire gli aspetti legati alla realizzazione del modello per un esperimento di simulazione al computer, superando, così, il divario tra le specifiche del modello di sistema ed il relativo software di simulazione.
Come già detto il "Model-based systems engineering" (MBSE) è l'applicazione formalizzata di modellazione per supportare i requisiti di sistema, la progettazione, l'analisi, la verifica e le attività di convalida a partire dalla fase di progettazione concettuale e per tutto lo sviluppo delle successive fasi del ciclo di vita; quindi, un modello è un'approssimazione, la rappresentazione o l'idealizzazione di aspetti specifici della struttura, del comportamento, del funzionamento, o di altre caratteristiche del processo reale. In questo modo, tale tecnica può essere usata in tutte le aree scientifiche e non solo, come riportato in figura 2.1.

Figura 2.1.: *"Grafico di applicazione della tecnica del "Model-Based Systems Engineering"*

Alla base del "model-based" e del "domain models" si trovano dei linguaggi di modellazione come il linguaggio unificato di modellazione (UML, "Unified Modeling Language") e il "Systems Modeling Language" (SysML), entrambi linguaggi di modellazione per scopi generali. L'UML è un linguaggio di modellazione nel campo dell'ingegneria del Software, che fornisce un modo standard per visualizzare meglio il progetto di un sistema. Tale linguaggio offre un modo per visualizzare l'architettura del sistema in un diagramma che mette in evidenza:

- le attività del sistema;
- i componenti individuali del sistema e come possono interagire con le altre componenti software;
- il comportamento del sistema in esecuzione;
- quali altre componenti e interfacce interagiscono con esso;
- le altre interfacce utente.

È importante distinguere tra modello UML e l'insieme dei diagrammi di un sistema, un diagramma infatti è una parziale rappresentazione grafica del modello di un sistema e non è detto che uno o più diagrammi riescano a coprire l'intero modello.
Il diagramma UML rappresenta due differenti aspetti del modello di un sistema:

- l'aspetto statico o strutturale, che enfatizza la struttura statica del sistema utilizzando oggetti, attributi, operazioni, relazioni e realizzata con diagrammi di classi, con tipi o strutture definite, e con diagrammi di strutture composite, ovvero, una struttura che mostra il contenuto interno di una classe e le collaborazioni che può effettuare;
- l'aspetto dinamico o comportamentale che enfatizza il comportamento dinamico del sistema mostrando le collaborazioni con gli oggetti e le variazioni all'interno degli stati degli oggetti; questo viene fatto utilizzando sequenze di diagrammi, cioè l'interazione tra diagrammi durante tutto il processo, l'attività di diagrammi e macchine a stati.

Il linguaggio di modellazione di sistema, SysML (Systems Modeling Language) è anch'esso un linguaggio di modellazione per applicazioni di sistemi ingegneristici che si occupa di

supportare le specifiche di analisi, progettazione, verifica e validazione di una rete di sistemi e sistemi di sistemi. Tale linguaggio è definito come un'estensione di un sottoinsieme dell'UML. Rispetto all'UML il SysML presenta diversi vantaggi come:

- la semantica del linguaggio SysML è più flessibile ed espressiva; in particolare riduce le restrizioni legate al linguaggio UML aggiungendo due nuovi tipi di diagrammi fornendo la possibilità di effettuare analisi delle performance e analisi quantitative;
- è un linguaggio con una sintassi relativamente piccola e facile da imparare e applicare;
- fornisce supporto per diversi tipi di allocazione tabellare, mentre l'UML fornisce solo supporti limitati per la notazione tabellare. SysML fornisce allocazioni flessibili di tabelle che supportano richieste di allocazione, allocazioni funzionali e allocazioni strutturali; in questo modo risultano più semplici operazioni quali la verifica e la validazione di un sistema e l'analisi di eventuali gap.

I vantaggi del linguaggio SysML rispetto al linguaggio UML per sistemi ingegneristici diventano ovvi se si considera la modellazione di un sistema automotive, infatti, con SysML si possono usare diagrammi di requisiti per catturare in modo efficiente richieste di funzionalità, performance e interfaccia superando i limiti imposti dall'UML sull'utilizzo di diagrammi per definire richieste di funzionalità ad alto livello.

2.1 "Model Based" per sistemi "embedded"

Uno dei campi che per primo ha sposato la filosofia del "model-based" per i sistemi embedded è il campo dell'automotive.
Lo sviluppo di affidabili sistemi automotive embedded si ritrova a dover affrontare sfide derivanti dalla crescente complessità, criticità e dalla domanda di certificabilità. Inoltre lungo l'intero ciclo di vita e di sviluppo è necessario garantire l'efficienza e la coerenza del sistema sviluppato. Le soluzioni esistenti al momento risultano essere spesso insufficienti quando si trasformano modelli di sistemi con un alto livello di astrazione in modelli ingegneristicamente più concreti (come i modelli di ingegneria del software). Di fatto in generale l'industria cerca di standardizzare le strutture in modo da facilitare lo scambio di informazioni.
La tendenza a sostituire i sistemi meccanici tradizionali con i moderni sistemi embedded consente l'implementazione di strategie di controllo più avanzate, che forniscono ulteriori vantaggi sia al cliente sia all'ambiente, ma al tempo stesso il più alto grado di integrazione e criticità dell'applicazione di controllo portano a dover affrontare nuove sfide. Per gestire i problemi relativi ai moderni sistemi real-time, si utilizza lo sviluppo model-based in modo da supportare, in modo più strutturato, la descrizione del sistema in fase di sviluppo. Lo sviluppo dell'approccio basato sul "model based" consente lo sviluppo da diversi punti di vista delle diverse parti interessate, ovvero consente lo sviluppo di diversi livelli di astrazione e di raccogliere quelle che sono le informazioni centrali del sistema. Questo migliora la consistenza, la correttezza, la completezza del sistema in esame e quindi supporta le esigenze richieste dal mercato. Tuttavia la progettazione di un sistema basandosi sul "model based" rappresenta spesso un'eccezione piuttosto che la regola.
Con il lavoro di Sporer, Macher & Co., raccolto nell'articolo "Incorporation of Model-based System and Software Development Environments" [9] si vuole cercare di colmare il divario esistente tra gli strumenti dell'ingegneria dei sistemi "model-based" e gli strumenti dell'ingegneria del software. In particolare, tale approccio si basa sulla valorizzazione di un sistema ingegneristico "model-driven" con la capacità di progettare un'architettura software. Inoltre, un modello di trasformazione permette la descrizione del software per la valutazione della

sicurezza critica, dai requisiti a livello di sistema fino all'implementazione della componente software, in modo bidirezionale. Lo strumento che hanno preso in considerazione come generatore automatico di architetture software è realizzato su Matlab / Simulink, ed è descritto attraverso un modello di un sistema di controllo di alto livello in formato SysML.
Gli obiettivi principali dello studio di Sporer & Co. sono, quindi:

- sostenere un perfezionamento costante e tracciabile dalla fase di semplice concezione del sistema fino all'implementazione del software;
- stabilire la funzione di aggiornamento bidirezionale della struttura di trasformazione, ottenendo benefici reciproci per i software di base e per l'applicazione dello sviluppo del software dalla coesistenza di entrambe le informazioni all'interno del database centrale.

Lo sviluppo model-based è attualmente il migliore approccio per gestire la grande quantità di informazioni e la complessità dei moderni sistemi embedded con vincoli di sicurezza. Un problema che è stato risolto riguarda il corretto trasferimento del modello di progettazione del sistema nel modello di ingegneria del software, e le successive modifiche devono comunque essere coerenti; per questo motivo alcuni studiosi propongono l'approccio con un modello di sincronizzazione che consiste in un insieme di adattatori tra i modelli SysML e i modelli di ingegneria del software nelle rappresentazioni AUTOSAR. L'inconveniente di questo approccio, è che ogni fase di trasformazione è una potenziale sorgente per una mappatura ambigua e un modello non adatto allo scopo.
Da un lato, linguaggi di modellazione generici (come UML o SysML) forniscono potenza di modellazione adatta a catturare ampi vincoli di sistema e comportamenti, ma presentano lo svantaggio della sintetizzabilità. D'altra parte, ambienti di modellazione per scopi speciali (come Matlab / Simulink e ASCET) sono ottimizzati per una progettazione piuttosto dettagliata ed una minore efficienza nella progettazione di alto livello.
L'idea è, quindi, quella di avere un repository di informazioni coerenti come fonte centrale di informazioni, per memorizzare le informazioni di tutte le discipline ingegneristiche coinvolte nello sviluppo del sistema automobilistico integrato. Ciò permette a chi sviluppa il software di farlo in modo specifico, fornendo tracce e analisi delle dipendenze di funzioni riguardanti il sistema nel suo complesso, ad esempio la sicurezza o l'affidabilità. Nel dettaglio ciò che vogliono realizzare è:

- framework software di modellazione UML: apportando miglioramenti nel profilo UML per la definizione di manufatti di sviluppo software, più precisamente, per la definizione della composizione componenti interfacce e architettura SW;
- definire un esportatore per architettura software: strumento in grado di generare l'architettura SW, progettato per lo strumento da terze parti, come Matlab / Simulink;
- definire un importatore dell'architettura software: strumento che importa l'architettura software e l'interfaccia dello strumento di sviluppo software.

In questo modo, l'obiettivo di Sporer & Co. è quello di eliminare il divario, citato precedentemente anche da altri, tra lo sviluppo a livello di sistema, come rappresentazione astratta con linguaggio UML, e lo sviluppo a livello software attraverso gli strumenti di modellazione (come appunto Matlab / Simulink).
Il ponte supporta la consistenza nel trasferimento di informazioni tra gli strumenti di ingegneria dei sistemi e gli strumenti di ingegneria del software e riduce al minimo lo scambio di informazioni manuale ridondante tra questi strumenti. Ciò contribuisce a semplificare le

argomentazione sulla sicurezza per il sistema sviluppato. Inoltre, la mancanza di strumenti di supporto per il trasferimento di informazioni tra gli strumenti di sviluppo del sistema e gli strumenti di sviluppo del software può essere dissipato con questo metodo. La realizzazione del ponte, attraverso l'utilizzo di librerie versatili (DLL) e Matlab COM Automation Server, garantisce l'indipendenza dello scopo generale dello strumento di modellazione UML (come Enterprise Architect o Artisan Studio) da Matlab / Simulink, attraverso comandi API di implementazione.
La prima parte del lavoro di Sporer consiste nello sviluppo di un framework di modellazione UML specifica con uno strumento di sviluppo dello stato dell'arte del sistema. Questo profilo rende la rappresentazione UML del sistema più gestibile per le esigenze di progettazione di una architettura software automotive sfruttando un livello di astrazione AUTOSAR allineati. Inoltre, il profilo consente una definizione esplicita delle componenti, delle interfacce dei componenti e delle connessioni tra le interfacce. Ciò fornisce la possibilità di definire un'architettura software e assicura la corretta definizione della comunicazione tra le architetture realizzate, comprese le specifiche di interfaccia. Quindi, la rappresentazione dell'architettura software all'interno di tale strumento può essere collegata agli sviluppi del sistema, consentendo la possibilità di tenere traccia delle richieste e portando ad ulteriori benefici in termini di vincoli di controllo, di tracciabilità delle decisioni di sviluppo e di riutilizzo.
Nella seconda parte si concentra sulla realizzazione dell'architettura di un esportatore software, che è in grado di esportare il progetto software, le componenti contenute e le loro interconnessioni specificate in SysML, nel "tool" di sviluppo software Matlab / Simulink. L'implementazione dell'esportatore si basa su Matlab COM Automation Server e genera modelli attraverso l'implementazione dei comandi API. Per l'input dell'utente, la rappresentazione dell'architettura software da trasferire viene selezionata, attivando un task in background che genera un corrispondente modello di Matlab / Simulink. Ogni modello realizzato, i parametri e le connessione vengono trasferite su Matlab / Simulink, direttamente in blocchi di dimensioni corrette e nella giusta posizione.
L'ultima parte del lavoro riguarda, invece, la realizzazione dell'importatore dell'architettura software, o meglio della sua funzionalità. Questa funzionalità, in combinazione con la funzione di esportazione stessa, consente un aggiornamento bidirezionale della rappresentazione dell'architettura software nello strumento di sviluppo del sistema e nei moduli software di Matlab/Simulink. L'importatore identifica i vari collegamenti della rappresentazione, permettendo in questo modo, di differenziare i nuovi modelli realizzati da quelli modificati. L'attivazione tramite l'ingresso dell'utente, avviene attraverso un'interfaccia all'interno dello strumento di sviluppo del sistema che raffigura modifiche tra le due rappresentazioni, classificandole come aggiunte, cancellate, o aggiornate, e consentendo l'aggiornamento selettivo della rappresentazione dell'architettura software basato sul linguaggio UML.
Tutto ciò garantisce la coerenza tra i modelli realizzati per lo sviluppo del sistema e i cambiamenti fatti nello strumento per lo sviluppo del software, mentre, la funzionalità di importazione consente il riutilizzo di moduli software disponibili, garantendo la coerenza delle informazioni attraverso vincoli imposti dagli strumenti, e condividendo l'informazione in modo più preciso e meno ambiguo.
Questo è solo uno dei lavori di ricerca che sono stati svolti negli ultimi anni e che riguardano il "model-based" applicato ai sistemi embedded. Il sempre più continuo sviluppo dell'approccio "model based" nell'ambiente embedded e dei lavori di ricerca su questo argomento, ha, quindi, portato alcuni studiosi a voler esaminare quali sono le tecniche più diffuse utilizzate nello sviluppo e nella progettazione di un sistema mediante l'approccio "model-based", in modo da analizzarne vantaggi e svantaggi [10].

Lo sviluppo dei sistemi embedded è sempre in continua variazione e per questo motivo è sempre maggiore la probabilità di commettere errori in fase di progettazione sia a causa del comportamento della scheda, sia a causa degli aspetti temporali del sistema embedded; per superare questi problemi sono stati adottati dei meccanismi che permettono di effettuare delle semplici operazioni di verifica e validazione del sistema embedded durante la fase di sviluppo. E la tecnica offerta dall'approccio model-based è una di queste.
L'attività primaria del "Model-Based" è quella di specificare le richieste del sistema ambedded; vengono modellati entrambi gli aspetti strutturali e comportamentali. Tuttavia come già accennato gli approcci utilizzati per ottenere la rappresentazione di un sistema "model-based" possono variare gli uni dagli altri in base alle tecniche utilizzate, ad esempio possono variare per la scelta del linguaggio UML o SYSML, o ancora a seconda delle tecniche di conversione che possono essere tecniche di conversione Model-to-Model o Model-to-text.
Come accennato, il lavoro svolto da tre ricercatori arabi, si propone di analizzare i lavori di ricerca basati sul model-based dal 2008 al 2014 e di classificarli in base alle tecniche adottate; in particolare le domande alle quali cercano di rispondere sono:

- quali lavori di ricerca svolti tra il 2008 e il 2014 che utilizzano l'approccio MBSE ("Model Based System Engeneering") sono finalizzati a sistemi embedded;
- in quali di questi lavori si utilizza il profilo UML e in quali quello SYSML;
- quale approccio di trasformazione è più frequentemente usato tra quello Model-to-Model e quello Model-to-Text.

Come si può osservare anche dalla figura 2.2, gli articoli esaminati sono stati raccolti dai quattro principali canali di raccolta degli articoli universitari e di ricerca come "SPRINGER", "IEEE", "ELSEVIER", "ACM". Dagli studi effettuati si è, quindi, osservato che tra il 2008 e il 2014 sono stati ben 61 i lavori di ricerca basati sull'approccio MBSE e finalizzati a sistemi embedded; tra tutti i lavori esaminati la maggior parte di quelli finalizzati ad un sistema embedded tendono ad utilizzare un profilo SYSML; infine, la tecnica di conversione più utilizzata è quella Model-to-Model, anche se negli ultimi anni sono stati osservati lavori in cui vengono utilizzate entrambe le tecniche.
Tutti gli articoli esaminati vengono poi classificati in diverse categorie:

- categoria generale che racchiude tutti i lavori di ricerca che forniscono soluzioni basate sul "model-based" e finalizzati ai sistemi embedded;
- categoria di modellazione: come già accennato lo scopo principale del "model-based" è quello di focalizzarsi sulle richieste di modellazione dei sistemi embedded, quindi, questa categorie raccoglie tutti gli articoli che discutono sulle tecniche di modellazione e in particolare quelli che usano un profilo UML piuttosto che uno SYSML;
- categoria delle proprietà specifiche, ovvero è la categoria che contiene tutti i lavori di ricerca in cui si discute di tutte le specifiche tecniche adottate, come ad esempio il profilo di linguaggio, in base al comportamento e agli aspetti temporali del sistema embedded in esame;
- categoria delle trasformazioni di modello che contiene tutti i lavori di ricerca in cui vengono affrontate discussioni sulla scelta della tecnica di trasformazione del modello, quindi, discussioni sulla tecnica "Model-to-Model" o "Model-to-Text";

- categoria di simulazione, raccoglie tutti gli articoli che realizzano delle simulazioni con la rappresentazione "model-based" del sistema embedded.

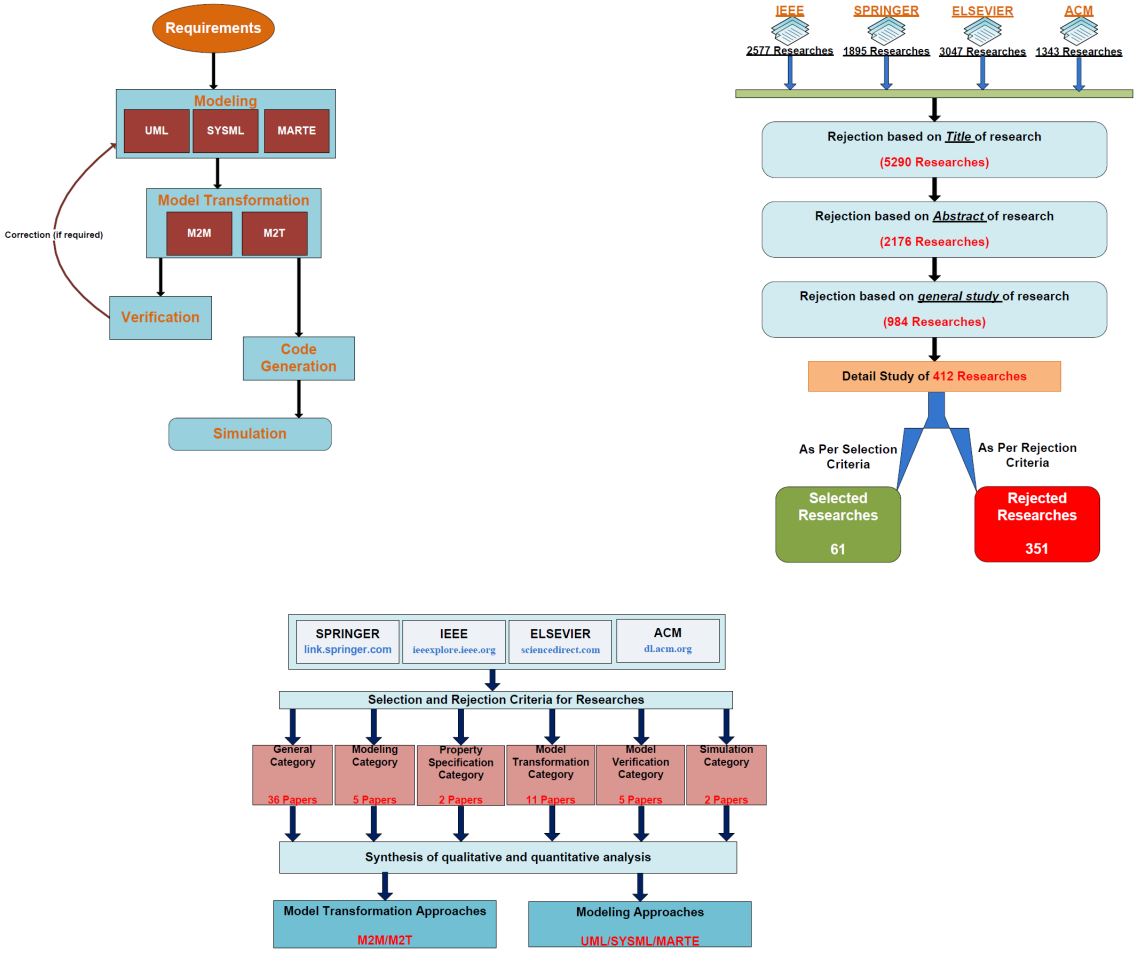


Figura 2.2.: *"Schema per la selezione dei lavori di ricerca sul "Model-based" finalizzati ai sistemi embedded"*

Quindi, quello che si può osservare è che, per quanto riguarda i modelli di trasformazione, oltre a rappresentare la chiave per la successiva verifica e validazione del sistema/modello, realizzano quello che è lo scopo principale cioè generare il codice sorgente per la simulazione del modello esaminato. Come già accennato quello che si può notare è che la tecnica di trasformazione più usata è quella Model-to-Model, rispetto a quella Model-to-Text, questo perchè l'accuratezza nella trasformazione della tecnica "M2M" permette di ridurre significativamente gli errori in fase di trasformazione; tuttavia, lo svantaggio principale che si ha nell'utilizzo di questa tecnica consiste nella complessità di implementazione, che di conseguenza facilita l'utilizzo della tecnica "M2T", che pur non avendo la stessa accuratezza in fase di trasformazione, risulta essere molto più semplice da implementare. Per questo motivo, negli ultimi lavori di ricerca si è potuto osservare un contemporaneo utilizzo di entrambe le tecniche soprattutto nel caso di complesse trasformazioni.
Per quanto riguarda i profili UML e SYSML/MARTE, si può osservare che questi sono quelli più frequentemente utilizzati per modellare gli aspetti strutturali e comportamentali del sistema embedded in esame sia separatamente che contemporaneamente. L'integrazione di questi profili fornisce flessibilità aggiuntive per i diversi aspetti comportamentali e strutturali

del modello del sistema, anche se alcuni aspetti sono comunque da migliorare. Nonostante il profilo MARTE supporti bene i vincoli temporali non è in grado di manipolare tutti gli aspetti strutturali e comportamentali del modello e quindi, viene utilizzato meno rispetto agli altri due.

2.2 "Model Based" per sistemi embedded tempo reale

Il "model-based" per lo sviluppo di un software di controllo è ampiamente utilizzato in una varietà di domini di sicurezza critica, tra cui automotive, aerospaziale e dell'automazione industriale. Gli algoritmi di controllo sono in genere sviluppati utilizzando la combinazione di due classici tipi di modelli: un diagramma a blocchi tempo discreto (BD) e una macchina a stati finiti discreta (FSM).
L'approccio model-based garantisce elevata coerenza tra i modelli di base e il codice prodotto, evitando gli errori che potrebbero essere introdotti in caso di sviluppo manuale del software; tale metodo delinea un modello probabilistico basato sull' analisi della propagazione dell' errore per algoritmi di controllo costruiti da modelli ibridi tempo discreto (BD) e macchine a stati finiti discrete (FSM).
La progettazione di sistemi embedded moderni diventa più complessa ogni giorno, a causa della crescente quantità di componenti e funzionalità distinte, incorporate in un unico sistema; per affrontare questa situazione, il livello di astrazione dei progetti viene continuamente sollevato, generando, di conseguenza, un aumento delle tecniche per la generazione di codice. Tutto ciò porta ad affrontare una questione importante che riguarda lo sviluppo di un consistente processo ingegneristico, per la progettazione coerente di sistemi, dalla fase dell'analisi dei requisiti fino alla produzione del codice sorgente generato. In questo contesto, l'UML è un'opzione interessante per la progettazione di sistemi embedded, che permette di ottenere tecniche per generare codice sorgente sia per hardware sia per software.
Attualmente, ci sono diversi strumenti per generare il codice sorgente da specifiche UML in linguaggi tradizionali, come C++ e Java. Tuttavia, ci sono anche tools che offrono la possibilità di generare automaticamente del codice sorgente per il linguaggio VHDL, linguaggio ampiamente utilizzato nello sviluppo dei sistemi integrati.
Il VHDL (VHSIC Hardware Description Language, dove VHSIC è la sigla di Very High Speed Integrated Circuits) è un linguaggio di descrizione dell'hardware (HDL) sviluppato negli Stati Uniti; è uno dei linguaggi più usati nella progettazione dei moderni circuiti integrati digitali e le sue applicazioni spaziano dai microprocessori (DSP, acceleratori grafici), comunicazioni (cellulari, TV satellitare), automobili (navigatori, controllo di stabilità) a molte altre. Il VHDL per alcuni aspetti si presenta simile a un vero e proprio linguaggio di programmazione, usa, infatti, alcuni tipici costrutti (if-then-else), tuttavia, essendo un linguaggio che descrive il funzionamento e la struttura di componenti hardware, ha alcune caratteristiche distintive rispetto ai linguaggi software. La principale è la concorrenzialità con cui si indica il fatto che diverse parti di un codice scritto in VHDL, una volta tradotte in un circuito elettronico, funzionano contemporaneamente, in quanto dispongono di hardware dedicato. Al contrario, in un linguaggio software, le funzioni descritte dal codice sono generalmente eseguite sequenzialmente, riga dopo riga, in quanto dispongono di un unico processore fisico. Il VHDL permette di modellare facilmente l'interazione tra i vari blocchi funzionali che compongono un sistema; queste interazioni sono essenzialmente lo scambio di segnali di controllo e di dati tra i vari oggetti che costituiscono il sistema. In un sistema hardware, infatti, ogni oggetto da modellare, sia esso una semplice porta logica o un complesso microprocessore, reagisce istantaneamente ai cambiamenti di stato dei propri ingressi producendo dei cambiamenti sulle proprie uscite. Ogni blocco funzionale, a sua volta, è descritto nella relazione

ingressi-uscite, usando i classici costrutti del linguaggi di programmazione (if, for, while). Nel lavoro di ricerca svolto da Moreira & Co. [8] viene proposta una metodologia per generare automaticamente codice sorgente VHDL da specifiche UML. Questa metodologia è supportato dallo strumento GenERTiCA, uno strumento per la generazione di codice.
Un sistema embedded è un sistema informatico progettato per eseguire solo una o poche funzioni dedicate, contenente componenti prevalentemente digitali, che consiste in una piattaforma hardware su cui i programmi applicativi software sono eseguiti. Nei sistemi embedded distribuiti (DES), la funzionalità del sistema si sviluppa su diversi nodi di calcolo che devono cooperare per raggiungere gli obiettivi del sistema. L'infrastruttura di comunicazione dei sistemi embedded distribuiti, costituita dalla sua piattaforma hardware e dal software embedded, è diversa dal sistema distribuito convenzionale a causa dei requisiti/vincoli dei sistemi embedded. Tali sistemi vengono ampiamente utilizzati anche nel settore industriale, nel quale sono chiamati componenti intelligenti in quanto prendono decisioni e svolgono le loro attività in maniera autonoma.
I sistemi embedded distribuiti di utilizzo industriale, supportano le funzioni tradizionali o innovative: le funzioni tradizionali sono quelle legate alle funzioni di controllo semplice, mentre le funzioni innovative rappresentano le funzioni più elaborate, come per esempio le funzioni per la manutenzione e le funzioni per la prognosi che permettono di eseguire funzioni per il monitoraggio delle condizioni, per la valutazione del corretto funzionamento e per la diagnostica. Il livello di intelligenza delle componenti è definito dalla quantità dei diversi servizi richiesti dall'utente che vengono implementati come funzioni delle componenti. Le componenti intelligenti sono sistemi, normalmente sistemi embedded, che utilizzano capacità di elaborazione per svolgere attività specifiche, come la manutenzione intelligente; essi infatti passano attraverso un processo di degradazione misurabile prima di danneggiarsi. L'idea di base dei sistemi di manutenzione e di diagnostica intelligenti è quello di utilizzare sensori e componenti informatiche, incorporate in apparecchiature, per raccogliere le sue informazioni. Sulla base di queste informazioni, gli algoritmi possono essere applicati per stimare lo stato di salute e per predire guasti, valutando il livello di degradazione della macchina. Pertanto, componenti informatiche integrate, cioè componenti intelligenti, come sensori incorporati, attuatori intelligenti ed elementi di elaborazione, giocano un ruolo fondamentale nello sviluppo di sistemi di manutenzione intelligenti.
Un tale scenario di progetto richiede strumenti e tecniche per aiutare lo sviluppatore a gestire la complessità del progetto. Un'idea comunemente accettata è quella di aumentare il livello di astrazione della lingua in cui il sistema può essere sviluppato. Recentemente, vi è stato un maggiore movimento verso l'utilizzo di linguaggi di modellazione grafici. Un approccio per aumentare il livello di astrazione è quello di usare un linguaggio di progettazione di alto livello standardizzato, che comprende le nozioni di orientamento dell'oggetto, e quindi, una mappatura specifica di alto livello fino a un tradizionale linguaggio di descrizione hardware e/o un linguaggio di descrizione software embedded, come C/C++ o Java. L'approccio proposto da Moreira & Co, propone un metodo per la generazione di descrizione hardware, più precisamente basato sul VHDL che permette di ottenere una stima del comportamento del sistema descritto, modellato, verificato e simulato, prima di tradurre gli strumenti di sintesi utilizzati in un progetto hardware reale (porte e cavi).Inoltre, in questo modo, si ha una riduzione del tempo di scrittura e degli errori di codifica che si otterrebbero con una produzione manuale del codice basata su specifiche di alto livello.
L'obiettivo è, quindi, quello di generare codice sorgente VHDL per le funzioni logiche di un sistema embedded, che finora è stato attuato solo per la parte software. Come già accennato il VHDL è un linguaggio di descrizione hardware utilizzato per l'automazione della proget-

tazione elettronica per descrivere sia sistemi con segnali digitali sia sistemi con segnali misti. Il recente sviluppo di questo linguaggio nella comunità dei sistemi embedded è dovuto alla facilità in fase di progettazione hardware e alla possibilità di validare il comportamento dei sistemi modellati durante la fase di simulazione.
Anche in questo caso per modellare i sistemi embedded si utilizza il linguaggio UML, linguaggio grafico che ha il vantaggio di diminuire la complessità del processo di progettazione, consentendo un metodo più intuitivo per la progettazione di tali sistemi. Tuttavia, la trasformazione da modelli costruiti con il linguaggio UML a quelli con il codice VHDL non è ancora ben diffuso. Per far fronte a questa sfida, nell'articolo di Moreira & Co si propone un approccio per la generazione automatica di codice sorgente da specifiche UML in codice VHDL per essere utilizzato in sistemi di FPGA (Field Programmable Gate Array), ovvero in sistemi integrati le cui funzionalitá sono programmabili attraverso software, fornendo un processo di progettazione che copre sia le fasi di analisi dei requisiti sia quella di modellazione UML per la generazione del codice VHDL.
Questo articolo presenta il caso di un sistema embedded distribuito utilizzato per sistemi di manutenzione (componenti intelligenti), che integra sia le funzioni tradizionali e innovative. L'approccio proposto è supportato da uno strumento chiamato GenERTiCA (Generazione di codice embedded in tempo reale sulla base di aspetti), che combina le tecniche di Model-Driven Engineering (MDE) con Aspect-Oriented Design (AOD) che in questo modo viene esteso con una serie di nuove regole di mappatura per consentire la generazione del codice VHDL.
Sono diversi gli approcci che si possono utilizzare per generare il codice sorgente partendo da modelli UML: alcuni di questi usano un solo schema (ad esempio diagramma delle classi) per la generazione del codice, mentre altri usano una combinazione di diagrammi distinti (ad esempio, di classe e di stato o diagrammi di sequenza). Altri ancora, invece, tendono ad elaborare quadri sviluppati per derivare le specifiche VHDL dalle classi UML e dai diagrammi di stato; essi utilizzano mappature tra strutture differenti, mantenendo le associazioni di classi meta-modello in un modo che ricorda la tecnica MDA, ovvero la tecnica che senza l'utilizzo di nessun supporto grafico permette di visualizzare testo monocromatico disposto su 80 colonne e 25 righe ad alta risoluzione. Tuttavia, in questo caso, il codice VHDL generato è focalizzato sulla simulazione e sulla verifica dei modelli UML invece di concentrarsi sulla sintesi hardware.
La tecnica MDA è utilizzata per definire descrizioni di sistemi model-based di alto livello che possono essere implementati sia come hardware sia come software. In questo modo è possibile trasformare diagrammi di stato UML direttamente in sintesi VHDL. Le macchine di stato UML sono usate per descrivere i comportamenti del sistema. Con questa metodologia, i requisiti funzionali sono mappati in componenti UML, come classi e diagrammi di stato, mentre i requisiti non funzionali sono specificati come annotazioni UML che descrivono vincoli sulle prestazioni, utilizzando coppie di valori di proprietà definiti dai profili UML. Tuttavia, questo approccio copre solo un piccolo sottoinsieme dei diagrammi di stato UML, senza supportare né la gerarchia né la concorrenza.
Nelle regole sviluppate, che consentono la generazione automatica di codice VHDL sintetizzabile da specifiche UML, il processo di progettazione utilizzato è basato su meta-modelli. I concetti racchiusi in un diagramma di stato UML, meta-modello, vengono mappati in concetti di meta-modello VHDL. Come si osserva anche dalla figura 2.3, le trasformazioni tra modelli realizzate sono due: la prima trasformazione converte il modello UML principale in un modello a diagramma di stato; la seconda, mappa questi modelli dei diagrammi di stato sui concetti della lingua VHDL. Una trasformazione model-to-text viene, infine, utilizzata

per generare il codice VHDL sintetizzabile dal modello VHDL.
Tra gli altri strumenti che consentono la generazione automatica di codice VHDL troviamo Simulink che genera codice VHDL sintetizzabile dai modelli Simulink e dai grafici "Stateflow". Simulink genera anche gli script di simulazione e di sintesi, che permettono di simulare e sintetizzare rapidamente un progetto sviluppato. Tuttavia il codice VHDL generato da Simulink, non deriva da specifiche UML.
Il limite di queste tecniche è quello di coprire solo specifici sottogruppi delle strutture UML; inoltre, essi generano solo codice per diagrammi di stato UML. Il "tool" GenERTiCA, usato nello studio di Moreira, consente, invece, l'utilizzo di diagrammi UML distinti, combinandoli per modellare la struttura, il comportamento e requisiti non funzionali che regolano i sistemi complessi. Le linee guida seguite, sono semplici e intuitive, e devono essere seguite per attivare la generazione automatica di codice. Inoltre, si ha la possibilità oltre alla generazione di codice VHDL, di ottenere codice Java e C/C++ dello stesso modello.

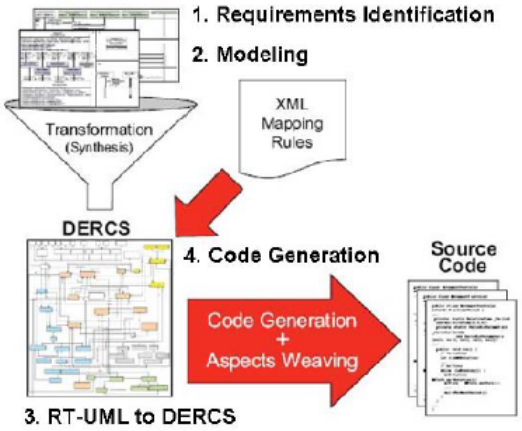


Figura 2.3.: *"Schema generazione codice per sistemi embedded"*

Più nel dettaglio, la tecnica descritta da Moreira, Wehrmeister & Co. segue il flusso di progettazione chiamato "Aspect-oriented Model-Driven Engineering for Real-Time systems" (AMoDE-RT), tecnica che è supportata dal generatore di codice GenERTiCA, che utilizza il mappaggio degli script per produrre file di codice sorgente, in diversi linguaggi di programmazione (tra cui C, C++, Java, ecc.) per una data piattaforma target, partendo da un modello UML, ottenuti con il profilo MARTE, "Modeling and Analysis of Real-Time and Embedded Systems". Tuttavia il processo per la generazione di codice per differenti linguaggi è lo stesso, quindi tale processo è del tutto generale.
I punti principali con i quali si realizza questa tecnica sono:

- Analisi dei requisiti e Identificazione: in questa fase, vengono analizzati tutti i requisiti ed i vincoli del sistema embedded real-time distribuito. Questo si ottiene attraverso l'analisi dei requisiti, che produce un insieme di documenti che descrivono i requisiti, le funzionalità e i vincoli del sistema. In seguito, si creano diagrammi che descrivono tutte le funzionalità previste per il sistema embedded distribuito e gli elementi esterni che interagiscono con il sistema.

- Modellazione: in questa fase vengono specificati gli elementi per gestire i requisiti funzionali e non funzionali raccolti nella fase precedente, ovvero nell'analisi dei requisiti. I

requisiti funzionali sono principalmente modellati utilizzando diagrammi di classi per descrivere le strutture e le sequenze dei diagrammi usati per descrivere il comportamento del metodo. Le classi e le sequenze di diagrammi sono obbligatori per descrivere la struttura e il comportamento di tutto il sistema con correttezza. Questi diagrammi UML sono ottenuti con lo stereotipo del profilo MARTE per specificare le proprietà in tempo reale degli elementi dei sistemi embedded distribuiti, DES. Durante questa fase, la gestione dei requisiti non funzionali è specificata utilizzando aspetti che derivano dal "Distributed Embedded Real-Time Aspects Framework".

- Trasformazioni da UML a DERCS: in questa fase le specifiche del sistema, vale a dire il modello UML, vengono trasformate in un altro modello chiamato DERCS, "Distributed Embedded Compact Specification", che rappresenta un sistema embedded privo di informazioni sovrapposte. Una specifica UML può contenere diversi elementi, che rappresentano lo stesso elemento; così questi elementi che possono risultare ambigui nel modello UML, vengono combinati e mappati in un solo elemento equivalente nel modello DERCS. Eliminando tali ambiguità, è possibile generare il codice senza generare errori di interpretazione della semantica del modello. Quando viene rilevata un'incoerenza e non può essere risolta, l'algoritmo di trasformazione da UML a DERCS viene interrotto ed è richiesto l'intervento del progettista per risolvere il problema.

- Generazione del codice:in questa fase il processo di generazione di codice esegue una serie di script, cioè regole di mappatura, che guidano la trasformazione del "model-to-text" dagli elementi del modello DERCS agli elementi della piattaforma di destinazione. Inoltre, il processo di generazione di codice incorpora tutti gli elementi con coi agisce il sistema: se l'elemento in fase di valutazione è influenzato da altri aspetti, questi vengono presi in considerazione modificando il codice generato in base a ciò che è descritto nelle regole di mappaggio.

2.3 Esempi di applicazioni dell'utilizzo del MBSE

Uno degli esempi più classici in cui si può osservare l'utilizzo della rappresentazione "model-based" di un sistema, è quello in cui tale rappresentazione viene utilizzata per effettuare dei controlli sulla diagnostica [11]. Sebbene lo sviluppo di sistemi applicativi per la diagnosi dei guasti abbia attirato l'interesse di tutto il mondo in diversi settori della ricerca, e nonostante sia nel campo del Controllo Ingegneristico che in quello dell'Intelligenza Artificiale siano state ricavate tecniche avanzate per trovare malfunzionamenti mediante l'utilizzo di modelli comportamentali e strutturali espliciti del sistema fisico diagnosticato, alcune delle quali basate su modelli (MBDS), non esiste ancora una chiara metodologia disponibile per la selezione di un adeguato approccio per risolvere un determinato problema diagnostico.
Il lavoro di Chantler, Leitch, Shen e Coghill ha proprio lo scopo di sviluppare una specifica metodologia che essenzialmente comprende un insieme di tasks diagnostici, un insieme di sistemi basati sul modello, e una serie di linee guida che forniscono una mappatura tra i task e la rappresentazione model-based del sistema.
Il loro scopo è quello di fornire un metodo con cui strumenti e tecniche esistenti sul "model-based" possono essere combinate con un'architettura generica in modo da produrre sistemi diagnostici efficaci per determinate applicazioni. La struttura della metodologia è stata derivata in parte da uno strumento diagnostico "model-based" generico che unisce una grande varietà di strumenti diagnostici model-based. Questa architettura si basa su tre principali tipi di conoscenza che sono necessarie per la costruzione di sistemi diagnostici basati sul modello.

Il primo tipo di conoscenza si basa sulle caratteristiche distintive dei sistemi model-based che permettono di realizzare modelli espliciti del sistema fisico e del suo comportamento. Tale conoscenza è generalmente nota al progettista o ingegnere di controllo responsabile del funzionamento sicuro ed efficiente del sistema. Esso può assumere la forma di equazioni o relazioni tra le principali variabili del modello. Il secondo tipo di conoscenza riguarda le informazioni sulle possibili anomalie che possono verificarsi durante il suo comportamento. Questa conoscenza può essere ottenuta da informazioni su guasti noti, a volte confusamente chiamati 'modelli di guasto' o potrebbe anche essere disponibile come insieme di relazioni tra sintomo e causa e questa è la conoscenza utilizzata in 'prima generazione' nei sistemi di diagnosi o più in generale negli approcci di classificazione basati sulla diagnosi. Se il comportamento osservato non corrisponde al comportamento previsto, allora viene rilevato un conflitto e la parte del modello che potrebbe generare un conflitto viene quindi identificato come parte sospetta. Tuttavia, dalla parte del modello sospetto non è immediato risalire alla parte reale del sistema mal funzionante. Quindi, è difficile individuare una conoscenza esplicita dei guasti da tali approcci; questo si potrebbe ottenere più facilmente se tali approcci fossero estesi per includere le conoscenze dei modelli di guasto.
La terza categoria di conoscenza determina come il processo diagnostico deve essere intrapreso. Questa conoscenza include la strategia diagnostica di base, che indica come i vari moduli sono combinati per eseguire la diagnosi attuale. Le ipotesi diagnostiche di base come guasti singoli o multipli e non intermittenti sono esempi di questo tipo di conoscenza. Inoltre, la conoscenza di eventuali limitazioni delle risorse in termini di vincoli temporali che influenzano l'utilità della diagnosi finale è cruciale per l'esecuzione del processo diagnostico. Questa conoscenza può essere utilizzata per mettere meglio in luce sia il modello del sistema sia le conoscenze sul guasto al fine di migliorare l'efficienza complessiva del sistema diagnostico. Quindi, il primo compito di questa metodologia è quello di fornire un insieme di requisiti che verranno utilizzati per definire i singoli problemi diagnostici che andranno a modificare il tipo di conoscenze richieste in seguito. A questo punto dato un insieme di richieste che caratterizzano lo spazio del problema è necessario ricavare quello che è lo spazio delle soluzioni, ovvero l'insieme di tutti i possibili sistemi "model-based", che una volta definito permette di ottenere la correlazione tra lo spazio del problema e quello delle soluzioni.

Capitolo 3

Multi-OS

Con il termine Multi-OS, si intende un dispositivo in cui si concretizza la possibilità di avere in esecuzione due sistemi operativi differenti contemporaneamente sullo stesso processore. Solitamente, dato un dispositivo su cui sono installati diversi sistemi operativi, è possibile scegliere quale sistema operativo mandare in esecuzione tramite il "Boot Manager". Il "Boot Manager" è un programma installato in un computer che durante l'accensione della macchina si avvia automaticamente e che permette di scegliere o selezionare quale sistema operativo avviare, ovvero ci permette di specificare al dispositivo quale kernel deve caricare. Una volta effettuata tale scelta viene avviato il boot loader del sistema operativo, ovvero il programma che nella fase di avvio del pc, carica il kernel del sistema operativo dalla memoria secondaria, o memoria di massa, alla memoria primaria, quella che lavora a più diretto contatto con il processore, permettendone così la sua esecuzione da parte del processore e il conseguente avvio del sistema.
Il compito principale di un boot loader è, quindi, quello di caricare ed eseguire il kernel di un sistema operativo, insieme ai processi e ai servizi secondari, compito che richiede l'accesso alla memoria di massa per leggere il kernel di sistema operativo e nel caso, altri file necessari. L'accesso al disco avviene solitamente tramite le funzioni fornite dal firmware o attraverso l'interpretazione di file system per trovare i file da caricare, nel caso di boot loader con tale capacità. I boot loader più moderni sono anche in grado di sfruttare le funzionalità fornite dalle schede di rete per scaricare un kernel dalla rete attraverso un protocollo di trasferimento file di livello applicativo (TFTP).
Il "boot manager", invece, agli inizi poteva essere avviato solo dalla RAM o dalla ROM, mentre nei moderni sistemi questo è sempre installato in una zona ben precisa dell'hard-disk, il master boot record, ovvero il settore di avvio principale. Senza un "boot manager" è impossibile far convivere due o più sistemi operativi sulla stessa macchina.
Il processore o unità di elaborazione è, come già stato accennato, un dispositivo hardware adibito all'esecuzione di istruzioni, all'elaborazione dati sotto la supervisione del sistema operativo; ogni processore lavora ad una certa frequenza di clock che rappresenta uno dei suoi parametri prestazionali in termini di capacità di processamento. Nel caso in cui un computer, o più in generale un altro dispositivo, sia fornito di più processori si parla di multiprocessore. Attualmente la maggior parte dei sistemi multi-OS si realizza mediante l'utilizzo di macchine virtuali, un software che, attraverso un processo di virtualizzazione, crea un ambiente virtuale che emula tipicamente il comportamento di una macchina fisica grazie all'assegnazione di risorse hardware tra cui una porzioni di disco rigido, RAM e risorse di processamento ed in cui alcune applicazioni possono essere eseguite come se interagissero con tale macchina; infatti se dovesse andare fuori uso il sistema operativo che gira sulla macchina virtuale, il sistema di base non ne risentirebbe affatto. Tra i vantaggi vi è il fatto di poter offrire contemporaneamente ed efficientemente a più utenti diversi ambienti operativi separati, ciascuno attivabile su effettiva richiesta, senza sporcare il sistema fisico reale con il partizionamento del disco rigido oppure fornire ambienti clusterizzati su sistemi server. Il software che rende possibile tale divisione è chiamato "virtual machine monitor" o "hypervisor". La virtualizzazione può essere suddivisa in diversi modi:

- Paravirtualizzazione: la macchina virtuale non simula un hardware ma offre speciali API che richiedono modifiche nel sistema operativo;

- Virtualizzazione completa (o nativa): la macchina virtuale esercita una completa virtualizzazione dell'hardware, tramite un Hypervisor di tipo 1 o 2;

- Virtualizzazione non nativa: la macchina virtuale emula il software ed il sistema operativo scritti per un altro processore permettondogli di venire eseguiti, è possibile che siano incluse componenti hardware che necessitano di microcodice;

- Virtualizzazione a livello di sistema operativo: la macchina virtuale adotta una virtualizzazione a livello kernel che permette a un server di virtualizzarne altri.

L'hypervisor, è il componente chiave per un sistema basato sulla virtualizzazione; esso deve operare in maniera trasparente senza pesare con la propria attività sul funzionamento e sulle prestazioni dei sistemi operativi. Svolge attività di controllo al di sopra di ogni sistema, permettendone lo sfruttamento anche come monitor e debugger delle attività dei sistemi operativi e delle applicazioni in modo da scoprire eventuali malfunzionamenti ed intervenire rapidamente. I requisiti richiesti a questo scopo sono quelli di compatibilità, performance e semplicità.
Gli ambiti di applicazione delle macchine virtuali sono molteplici ed eterogenei fra loro, poichè la virtualizzazione sta diventando sinonimo di sicurezza informatica ed affidabilità del sistema.
L'hypervisor può controllare ed interrompere eventuali attività pericolose, e ciò fa sì che si usino macchine virtuali sempre più frequentemente in ambito di ricerca e collaudo di software.
L'hypervisor può, inoltre, allocare le risorse dinamicamente quando e dove necessario, può ridurre in modo drastico il tempo necessario alla messa in opera di nuovi sistemi, può isolare l'architettura nel suo complesso da problemi a livello di sistema operativo ed applicativo, può abilitare ad una gestione più semplice di risorse eterogenee e, come già accennato, può facilitare collaudo e debugging di ambienti controllati.
L'hypervisor può essere classificato in due categorie:

- hypervisor nativo, "bare-metal" o di tipo 1: in questo caso l'hypervisor esegue direttamente nell'hardware del sistema principale per controllare l'hardware e manipolare il sistema operativo ospite; per questo motivo è spesso chiamato hypervisor "bare-metal". Un sistema operativo ospite che viene eseguito in questo modo non è altro che un processo che viene eseguito nel sistema operativo principale;

- hypervisor "hosted" o di tipo 2: questo hypervisor esegue in un sistema operativo convenzionale; l'hypervisor di tipo 2 astrae il sistema operativo ospite dal sistema operativo principale.

La distinzione tra i due tipi di hypervisor non è del tutto chiara.
Tuttavia queste tecniche sono adatte per realizzare un ambiente multi-OS nei personal computer, ma non sempre tali tecniche sono adatte per l'ambiente embedded.

3.1 La virtualizzazione nei sistemi embedded

La virtualizzazione di sistemi, che gode di grande popolarità negli ambienti industriali e nell'ambito dei personal computer, sta recentemente guadagnando notevole importanza nel dominio embedded, in quanto, è in grado di soddisfare le particolari esigenze dei sistemi

embedded.
La virtualizzazione del sistema è diventata uno strumento tradizionale nel settore informatico. Il disaccoppiamento delle piattaforme di calcolo virtuali e fisiche tramite la macchina virtuale (VM) di sistema supporta una varietà di usi, di cui i più popolari sono:

- consolidare i servizi che utilizzavano i singoli computer all'interno delle singole macchine virtuali sullo stesso computer. Questo sfrutta il forte isolamento, risorsa garantita dalle macchine virtuali al fine di raggiungere le qualità di servizio di isolamento tra i server;

- bilanciamento del carico tra i cluster, con la creazione di nuove macchine virtuali a richiesta su un host leggermente usato, o anche la migrazione verso macchine virtuali. Questo utilizza l'astrazione della piattaforma fornita dalla virtualizzazione;

- gestione dell'alimentazione in cluster, spostando le macchine virtuali leggermente caricate, che possono poi essere spente;

- servizi di firewall che hanno un alto rischio di compromissione per proteggere il resto del sistema;

- sistemi operativi diversi che vengono eseguiti sulla stessa macchina fisica (ad esempio Windows, Linux e MacOS), tipicamente al fine di eseguire le applicazioni, che sono specifiche per un particolare sistema operativo. Questo utilizzo è rilevante per personal computer (desktop o laptop) ed è anche abilitato per l'isolamento delle risorse.

Una caratteristica principale di tale utilizzo è che in genere tutte le macchine virtuali eseguono lo stesso sistema operativo (o sistemi operativi "simili", nel senso che essi forniscono approssimativamente lo stesso tipo di capacità e livelli di astrazione simili). Un'altra caratteristica di questo scenario è che la comunicazione nella macchina virtuale, avviene proprio come nella macchine fisica attraverso interfacce di rete (virtuale, compresi i file-systems di rete). Questo è coerente con la visione che si ha della macchina virtuale (VM), che, per definizione, è come quella di una macchina fisica.
Chiaramente, la maggior parte dei casi di utilizzo elencati sopra non hanno degli equivalenti nei sistemi embedded attuali (anche se alcuni diventano rilevanti con l'avvento di chip multicore). Per capire il motivo per cui gli sviluppatori di sistemi embedded si stanno sempre più interessando all'utilizzo delle macchine virtuali è necessario osservare quelle che sono le caratteristiche dei moderni sistemi embedded, e individuare punti in comune e le differenze dei sistemi di elaborazione aziendali.
I sistemi embedded sono usati per essere relativamente semplici e sono dominati da vincoli hardware quali la memoria, la potenza di elaborazione, la carica della batteria. La loro funzionalità è stata anche in gran parte determinata dall'hardware, con software costituito in gran parte da driver per le periferiche, schedulatori e un pò di logica di controllo. Come risultato, i sistemi embedded sono dotati di software che vanno da una bassa a una moderata complessità. Tali sistemi sono stati sottoposti a vincoli di tempo reale, che pone domande sui sistemi operativi general purpose che invece sono inusuali nel calcolo tempo reale. I sistemi embedded tradizionali sono anche chiusi: lo stack software completo è fornito dal produttore del dispositivo, caricato in fase di prevendita e non cambia (tranne che per gli aggiornamenti del firmware). I moderni sistemi embedded, tuttavia, stanno sempre più assumendo caratteristiche dei sistemi general-purpose. La loro funzionalità è in crescita, e così è la quantità e la complessità del loro software.

Sempre più spesso, i sistemi embedded eseguono applicazioni originariamente sviluppate per l'ambiente dei PC e nuove applicazioni (ad esempio giochi) sono sempre scritte portando alla richiesta di sistemi operativi orientati ad applicazioni di alto livello con API appropriate. I dispositivi embedded sono ancora sistemi in tempo reale (o almeno una parte del software è in tempo reale) e sono ancora dotati di risorse limitate: aumento della capacità della batteria solo lentamente nel tempo, per cui i dispositivi mobili hanno bilanci energetici ristretti. Inoltre, la maggior parte dei sistemi embedded sono venduti a poco prezzo, quindi sono dotati di poca memoria; la memoria è infatti un fattore di costo (oltre a consumare energia). Allo stesso tempo, i sistemi embedded, già piuttosto diffusi, stanno diventando sempre più parte della vita quotidiana; inoltre essi sono sempre più utilizzati in scenari critici e di conseguenza diventano elevati e sempre più crescenti i requisiti in materia di sicurezza e affidabilità.
La rilevanza della virtualizzazione nei sistemi embedded deriva dalla capacità di affrontare alcune delle nuove sfide poste da loro. Una è il supporto per gli ambienti dei sistemi operativi eterogenei, come un modo per affrontare le esigenze contrastanti di API di alto livello specializzati per la programmazione di applicazioni, prestazioni in tempo reale. Tradizionali applicazioni del sistema operativo non hanno il supporto per la vera risposta in tempo reale e non sono adatti per sostenere la grande quantità di firmware in esecuzione su dispositivi presenti attualmente. La virtualizzazione può aiutare, consentendo l'esecuzione contemporanea di un sistema operativo di applicazione (Linux, Windows, Symbian) e di un sistema operativo real-time (RTOS) sullo stesso processore come si può osservare nella figura 3.1.

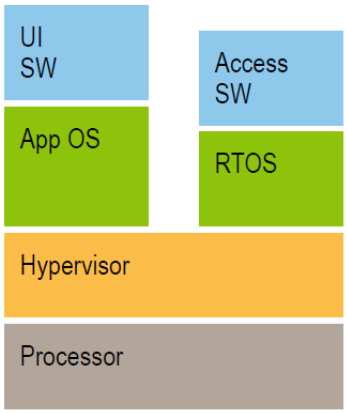


Figura 3.1.: *"Primo caso di utilizzo della virtualizzazione nei sistemi embedded che consiste nella coesistenza di due sistemi operativi completamente differenti sullo stesso processore."*

A condizione che l'hypervisor di fondo sia in grado di fornire in modo affidabile e veloce gli interrupt per l'RTOS, quest'ultimo può quindi continuare a eseguire gli elementi nello "stack" fornendo funzionalità delle periferiche tempo reale [13]. L'applicazione del sistema operativo può fornire le funzionalità delle API richieste e dell'alto livello di funzionalità adatto per la programmazione delle applicazioni.
Lo stesso può essere raggiunto utilizzando più core di processore, ognuno dei quali esegue il proprio sistema operativo, a condizione che vi sia un supporto hardware per il partizionamento in modo sicuro della memoria. Chip multi-core si stanno sviluppando sempre più. Inoltre, due core di performance inferiori (che possono essere disabilitati singolarmente) tendono ad avere un consumo di potenza media inferiore rispetto ad uno solo ad alte prestazioni,

a causa di forti non linearità nella gestione dell'energia. Un'altra caratteristica importante è che la virtualizzazione supporta l'astrazione architettonica, come la stessa architettura software può essere spostata sostanzialmente in modo invariato tra un multicore e un single core virtualizzato.
I chip multi-core più recenti forniscono una diversa, e più a lungo termine motivazione per l'utilizzo della virtualizzazione. Con un gran numero di processori, è probabile che i sistemi embedded dovranno affrontare problemi non dissimili dalle ragioni che si trovano dietro l'utilizzo della virtualizzazione nello spazio "enterprise". Un hypervisor scalabile potrebbe costituire la base per l'implementazione di sistemi operativi su un gran numero di core, suddividendo il chip in diversi domini multiprocessore più piccoli. Un hypervisor può aggiungere dinamicamente core per un dominio di applicazione che richiede potenza maggiore rispetto a quella fornita dal processore, o in grado di gestire il consumo di energia, eliminando processori da domini e spegnendo core inattivi. L'hypervisor può essere utilizzato anche per la creazione di domini ridondanti per la tolleranza di errore in una configurazione particolari.
Probabilmente la motivazione più forte per la virtualizzazione è la sicurezza.
Con lo sviluppo dei sistemi aperti, la probabilità che un'applicazione del sistema operativo venga compromessa aumenta notevolmente. Il danno risultante può essere minimizzato eseguendo tale sistema operativo in una macchina virtuale che limita l'accesso al resto del sistema, come mostrato in figura 3.2.

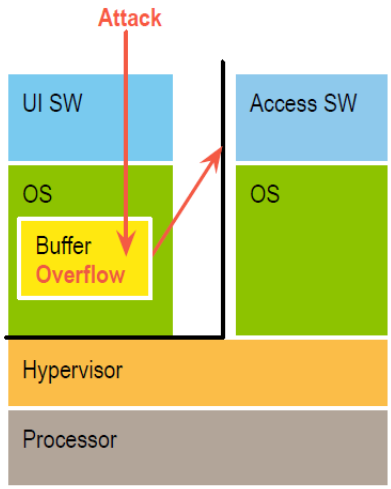


Figura 3.2.: *"Utilizzo della macchina virtuale che mostra come in presenza di un attacco esterno, questo non vada a coinvolgere tutto il sistema ma solo una sua parte."*

La sicurezza del sistema è verificata solo se:

- l'hypervisor di fondo è molto più sicuro del sistema operativo ospite (il che significa prima di tutto che l'hypervisor deve essere molto più piccolo);
- funzionalità critiche possono essere suddivise in macchine virtuali diverse dall'utente.

Se queste condizioni non sono soddisfatte, l'hypervisor aumenterà semplicemente le dimensioni della TCB (Trusted Computing Base), che è controproducente per la sicurezza.
Il "Trusted Computing Base", TCB, è l'insieme di tutte le componenti hardware, software e/o firmware che sono critiche per la sicurezza del dispositivo in cui si trovano, nel senso che i problemi o le vulnerabilità che possono verificarsi all'interno della TCB possono compromettere il funzionamento dell'intero sistema. Al contrario, le parti di un sistema che non

sono comprese nella TCB non dovrebbero essere affetti da comportamenti scorretti, in modo che possano godere di particolari privilegi in accordo alle politiche di sicurezza dell'intero sistema. L'attenta progettazione e la realizzazione del TCB di un sistema è, quindi, fondamentale per la sua sicurezza complessiva. I sistemi operativi moderni cercano di ridurre le dimensioni del TCB in modo che un esame esaustivo del suo codice (mediante controllo del software manuale o assistita da computer o attraverso verifica del programma) diventi possibile. Infine, un ampio e standardizzato supporto per la virtualizzazione consente un nuovo modello per la distribuzione di applicazioni software, si ha quindi che il programma viene inviato insieme con la propria immagine del sistema operativo, comportando così una minore probabilità di guasto del software distribuito a causa della mancata corrispondenza di configurazione.
Da ciò che è stato detto fino ad ora, si può osservare che la virtualizzazione può fornire alcuni vantaggi interessanti per i sistemi embedded. Tuttavia, ci sono significative limitazioni sull'uso della macchina virtuale nei sistemi embedded, infatti, tali limitazioni sono una conseguenza diretta di ciò che rende la virtualizzazione così popolare.
La virtualizzazione si basa completamente sull'isolamento, per definizione infatti, una macchina virtuale gira su suo hardware virtuale come se avesse l'uso esclusivo dell'hardware fisico. Ma questo tipo di pseudo-virtualizzazione non risolve nulla, rappresenta solo il caso più semplice di sistema operativo eterogeneo. Tuttavia il modello di macchine virtuali fortemente isolato non soddisfa i requisiti richiesti dai sistemi embedded.
Per loro natura i sistemi embedded sono altamente integrati, tutti i loro sottosistemi devono cooperare al fine di contribuire alla funzione generale del sistema. Isolando tra loro le varie componenti, si interferisce con i requisiti funzionali del sistema stesso. La cooperazione tra i vari sottosistemi di un sistema embedded richiede un'efficiente condivisione delle risorse. Ciò è necessario per il trasferimento dei dati, un cellulare, ad esempio, può ricevere un file video attraverso la rete cellulare (ossia tramite il sistema operativo in banda base), che viene poi visualizzata sullo schermo (da un lettore multimediale in esecuzione sul sistema operativo applicazione). Il trasferimento di questi dati in "stile macchina virtuale" tramite un'interfaccia di rete virtuale tra il tempo reale e l'applicazione, implica almeno un'operazione di copia in più, e cioè uno spreco significativo di cicli del processore (e quindi un ulteriore consumo della carica della batteria). La soluzione a questo problema si può trovare chiaramente, utilizzando un buffer condiviso a bassa latenza con primitive di sincronizzazione/messaggistica. Tuttavia, tali operazioni non si adattano al modello della macchina virtuale.
Un altro disallineamento tra le richieste dei sistemi embedded e il modello della macchina virtuale è evidente in fase di programmazione. Le macchine virtuali, per loro natura, sono programmate dall'hypervisor come scatole nere, con il sistema operativo ospite responsabile delle attività di pianificazione all'interno di tale struttura. Tuttavia, questo non è adatto per sistemi embedded; la loro caratteristica di essere un sistema fortemente integrato richiede un approccio integrato, quindi globale, di programmazione. Mentre le attività in tempo reale (in esecuzione in un RTOS) generalmente devono avere la massima priorità di pianificazione, il dominio RTOS avrà anche attività in background a bassa priorità. Queste non dovrebbero essere in grado di fare "premption" sull'interfaccia utente in esecuzione con il sistema operativo dell'applicazione. Allo stesso modo, nell'applicazione del sistema operativo potrebbero essere in esecuzione alcune attività tempo reale, ad esempio, il lettore multimediale, che può avere la precedenza su un'altra attività tempo reale nell'ambiente RTOS. Le caratteristiche dei sistemi embedded, ovviamente, non possono essere affrontate con il modello di schedulazione decentrato e gerarchico che è insito nella virtualizzazione.
La gestione dell'energia nei sistemi embedded coinvolge frequentemente le operazioni, an-

dando a influenzare notevolmente quello che è il limite di confine tra perfomance e potenza, considerando il vincolo di rispettare le varie scadenze. Tuttavia, l'energia è una risorsa fisica globale che non può essere scambiata con il tempo, che risulta essere virtuale. Inoltre, la virtualizzazione fa ben poco per affrontare quello che probabilmente è il problema più importante dei sistemi embedded: il notevole incremento di software ad elevata complessità, che rischia di minare la solidità e la sicurezza del dispositivo. L'approccio software-engineering utilizzato per cercare di affrontare la sfida dell'aumento della complessità dei software è quella di utilizzare componenti incapsulate per il contenimento dei guasti. Mentre la virtualizzazione fornisce l'incapsulamento, la sua granularità è troppo grossolana per fare una grande differenza; una macchina virtuale emula l'hardware ed è progettata per eseguire un sistema operativo che supporti il relativo software. Ciò significa che le macchine virtuali sono abbastanza pesanti, e sistemi embedded non possono ragionevolmente eseguire più macchine virtuali (a questo si riconduce anche il discorso sul costo del dispositivo, sulla memoria disponibile e quindi sula potenza del dispositivo).
Ultimo ma non meno importante, è il problema del controllo del flusso. Osservati quelli che sono i problemi posti dai sistemi embedded moderni, questi vengono tradotti in requisiti per un'adeguata tecnologia di sistema operativo. La tecnologia ideale fornirà componenti fortemente incapsulati, adatti per il contenimento di un eventuale guasto e l'isolamento per questioni di sicurezza. Inoltre deve supportare la virtualizzazione del sistema più tradizionale, in particolare la possibilità di eseguire istanze in modo isolato, del sistema operativo ospite e le sue (non modificate) applicazioni nello stack. Questo deve essere fatto mantenendo la reattività in tempo reale per i sottosistemi con tempistiche critiche.
Allo stesso tempo, la bassa latenza controllata e la comunicazione ad alta larghezza di banda devono essere disponibili tra i vari componenti, a cui si aggiunge la memoria condivisa, oggetto di una politica di sicurezza a livello di sistema, imposto da una piccola e affidabile TCB. I componenti devono essere sufficientemente leggeri per renderli adatti per incapsulare singoli fili, per imporre un criterio di pianificazione globale.
Queste richieste superano le funzionalità di un hypervisor e richiedono più meccanismi rispetto a quelli offerti da un sistema operativo general-purpose. Microkernel ad alte prestazioni sembrano offrire le giuste capacità. Tali microkernel hanno una comprovata esperienza come base per hypervisor e sistemi operativi in tempo reale. In particolare i vari membri della famiglia L4 di microkernel sono caratterizzati da un passaggio del messaggio con un "overhead" estremamente basso, leggeri spazi di indirizzi che costituiscono la base di incapsulamento, meccanismi per la creazione di regioni di memoria condivisa e driver di periferica a livello utente ad alte prestazioni. L'efficiente meccanismo del passaggio del messaggio (IPC) è il fattore chiave per qualsiasi sistema basato sul microkernel e questo include anche le macchine virtuali.
I problemi dovuti alla virtualizzazione sono catturati dal kernel e convertiti in messaggi di eccezione inviati ad un livello utente del monitor della macchina virtuale. Infine, il kernel rende driver di periferiche a livello utente fattibili, come le interrupt vengono convertiti in messaggi IPC e inviati al driver.
Le regioni di memoria condivisa sono alla base per una efficiente condivisione dei buffer in modo da annullare le operazioni di copia. Questo è importante anche per la condivisione di dispositivi tra le macchine virtuali e altri componenti. La condivisione del driver può essere applicata anche tra varie macchine virtuali, per esempio un driver può, inizialmente, essere lasciato all'interno del suo sistema operativo originale, ovvero all'interno del quale è stato sviluppato, ma successivamente può essere reso disponibile ad altri componenti del sistema. Questo, naturalmente, significa che il sistema operativo ospite in cui verrà eseguito il driver

deve essere attendibile per operare correttamente sul particolare dispositivo.
Un "software stack", ovvero un insieme di programmi che lavorano insieme per raggiungere un determinato obiettivo o per produrre determinati risultati può essere facilmente portato su un microkernel incapsulando lo stack completo all'interno di una macchina virtuale. Le componenti critiche, come la crittografia e gestione delle chiavi, possono essere spostati in componenti separati che funzionano direttamente sopra il microkernel con una TCB minima. Tali componenti possono essere attivi, cioè contengono proprio un thread schedulabile per l'esecuzione o passivo, invocati in modo simile ad una chiamata di funzione e l'esecuzione è determinata nel contesto di pianificazione del chiamante. La capacità di assegnare individualmente una priorità di schedulazione di un componente attivo supera i limiti di schedulazione della macchina virtuale.
Nel corso del tempo, più parti possono essere estratte fuori dalla macchina virtuale in componenti separate, portando ad un sistema altamente strutturato con una maggiore robustezza. Il progettista del sistema è in grado di decidere il compromesso più appropriato tra sicurezza, robustezza e costo ingegneristico.
Questo approccio è già attivamente applicato nel settore dei sistemi embedded. Tuttavia, l'aspetto più interessante del piccolo TCB abilitato dalla tecnologia microkernel è la possibilità di stabilire la sua affidabilità al di là di ogni dubbio. Un microkernel ben progettato è abbastanza piccolo per essere completamente formalmente verificato attraverso una dimostrazione matematica in cui si mostra che la realizzazione è conforme alla specifica (e quindi è in un certo senso garantito il fatto che sia "bug-free").
Questo microkernel rappresenta un salto di qualità dalla tecnologia del sistema operativo stabilito. Un kernel formalmente verificato può essere trattato in modo molto simile all'hardware: cambia solo ogni pochi anni (e non post-distribuzione). Inoltre esso permette al software sulla parte superiore di essere aggiornato.
La virtualizzazione ha molti aspetti interessanti per il mondo embedded, ma di per sè è un match povero per sistemi embedded moderni; infatti per poter utilizzare meglio tale tecnica è necessaria una maggiore tecnologia generale del sistema operativo, che supporta un grado di incapsulamento maggiore, la programmazione integrata e il controllo del flusso di informazioni.
I microkernel ad alte prestazioni sono in grado di fornire tutte le caratteristiche richieste e consentire una migrazione da componenti stack monolitici a componenti "software stack" più robuste. La prospettiva di una verifica formale dell'attuazione del microkernel offre un'interessante strada verso sistemi di affidabilità senza precedenti. Sembra che ci sia una tendenza che porta l'hypervisor a diventare più simile ai microkernel, tra cui l'aggiunta di primitive microkernel come per superare alcune delle carenze della virtualizzazione. I microkernel, tuttavia, hanno il vantaggio intrinseco di consentire un più piccolo TCB; inoltre, i benefici di verifica formale possono, per il prossimo futuro, essere raggiunti solo con un vero e proprio microkernel, a causa della scarsa adattabilità delle tecniche di verifica formale. Da quanto detto quello che si può osservare è che ci sono buone ragioni per la distribuzione di alcune forme di virtualizzazione nei sistemi embedded future. Tuttavia, il ruolo della macchina virtuale rimane limitato e le loro prestazioni migliori vengono raggiunte pienamente solo nel contesto di un cambiamento generale della tecnologia del sistema operativo basata sul microkernel ad alte prestazioni.

3.2 Vulnerabilità di un sistema multi-OS in un ambiente embedded

Negli ultimi anni sta sempre più prendendo piede l'impiego del multi-Os nei diversi ambiti dell'ambiente embedded. Ciò è motivato dalla necessità di soddisfare diverse esigenze alle quali si aggiungono quelle legate alla sicurezza critica degli ambienti automotive, che si traducono in elevate esigenze di sicurezza dei sistemi embedded realizzati. A seconda dell'architettura hardware, è possibile utilizzare diverse tecniche per isolare i sistemi operativi, aspetto che deve essere garantito per motivi di sicurezza. Nonostante i meccanismi di virtualizzazione siano quelli più diffusi, l'idea di multiprocessore asimmetrico può essere usato per dividere le risorse hardware di un sistema, il che rende la virtualizzazione dell'hardware obsoleta. Tuttavia, i dispositivi indipendenti come co-processori potrebbero aggiungere potenziali rischi per la sicurezza.
Nel lavoro svolto da degli studiosi tedeschi [14], viene mostrato un vettore di attacco, che utilizza un co-processore per rompere l'isolamento di un dominio del sistema operativo. Utilizzando un ambiente multi-OS, un coprocessore viene manipolato al fine di eludere i meccanismi di isolamento per conto di un sistema operativo "d'attacco".
All'interno dell'ambiente automobilistico, funzioni indipendenti distribuite, come la gestione audio, la gestione video, la navigazione, il telefono, e altri, sono stati consolidati in un unico sistema, la cosiddetta unità principale (HU). Negli ultimi decenni il numero di queste funzioni è aumentato. Le nuove applicazioni sono destinate all'uso nei futuri sistemi per le auto. Una delle funzioni più importanti è la connessione Internet per supportare lo streaming audio, la navigazione web, e l'utilizzo dei social network come Twitter e Facebook. Inoltre, i futuri sistemi per le auto sono in grado di visualizzare contenuti su più schermi che sono dedicati al conducente o ai passeggeri nella parte posteriore. Ad esempio, il conducente vede un tachimetro 3D animato e altre informazioni correlate alla guida, mentre i passeggeri possono guardare i film. Come risultato, tali sistemi diventano ancora più complessi. Tutte le parti che costituiscono tali sistemi, ma anche quelli più critici e indispensabili come quelli per la gestione della frenata o dell'ABS, hanno le proprie esigenze di risorse specifiche. Questi requisiti possono essere tradizionali, come la frequenza della CPU, la quantità di memoria utilizzata, i driver, la funzionalità in tempo reale e la connettività; o requisiti che dipendono dall'utilizzo di piattaforme con altri sistemi operativi (come ad esempio, Google Android o iOS di Apple).
L'integrazione di applicazioni mobili è una delle principali caratteristiche a cui mirano i vari produttori automobilistici. Tuttavia, a causa dei costi in termini di complessità di sviluppo e di complessità ingegneristica, nella prossima generazione di software HU, le applicazioni per dispositivi mobili sono ancora tralasciate. Questo è anche motivato dal fatto che il normale ciclo di vita di un'applicazione mobile differisce da quella di un software per auto. Inoltre, per garantire ai passeggeri la funzionalità, le applicazioni hanno bisogno di essere eseguite nel loro ambiente di sviluppo.
I sistemi operativi di oggi possono essere classificati in tre classi: sistemi operativi real-time (RTOS), i sistemi operativi general purpose (GPOS) e sistemi operativi mobili (Mobile-OS). Quindi, i futuri sistemi sviluppati per le auto, faranno uso delle funzionalità di tutte e tre le categorie di sistemi operativi; anche se, la complessità del sistema operativo è in aumento con i requisiti desiderati. Per questo motivo i sistemi operativi general-purpose (Linux, Windows) e i sistemi operativi embedded, comunemente destinati all'ambiente automotive potrebbero non essere in grado di soddisfare tutti i requisiti per i futuri sistemi multimediali desiderati.

L'idea è, quindi, quella di combinare tutti i sistemi operativi necessari in modo da realizzare un unico multi sistema operativo (multi-OS). In questo modo, ogni sistema ha una complessità limitata e offre vantaggi dal punto di vista della sicurezza, il che significa che la connessione ad internet e l'utilizzo di applicazioni vulnerabili non vanno ad influire sulla sicurezza dell' unità principale (HU). Di conseguenza, l'isolamento affidabile di tutto il multi-OS è necessario per soddisfare le esigenze del settore automobilistico. In particolare per motivi di prestazioni, la tecnica utilizzata per fornire un ambiente adeguato per più sistemi operativi gioca un ruolo importante. Ciò vale in particolare per l'isolamento.
Contrariamente ai metodi comunemente disponibili di virtualizzazione per i personal-computer, i sistemi embedded spesso si basano su tecniche di virtualizzazione hardware supportate, al fine di ottenere un sistema operativo ospite in esecuzione che sia il più possibile vicino all'hardware. Inoltre, periferiche o dispositivi come "controller" grafici dovrebbero essere accessibili in un ambiente il più possibile vicino all'ambiente in cui sono stati sviluppati. Lo scopo dello studio di alcuni studiosi tedeschi è quello di rendere evidente la vulnerabilità in base a circostanze specifiche, combinando diversi sistemi operativi. Per dimostrare ciò, tali studiosi propongono l'utilizzo di un vettore di attacco per sfruttare la vulnerabilità in modo da dimostrare che i rischi sulla sicurezza devono essere affrontati in futuro tenendo conto dell'ambiente proposto. Per questo motivo è importante concentrarsi sulle architetture hardware a disposizione del pubblico dominio.
Per fornire la capacità di eseguire più sistemi operativi contemporaneamente, è implementato un sistema per adattare il concetto di base di multiprocessore asincrono (AMP). L'idea alla base è quella di dividere tutte le risorse disponibili e di assegnarle a un sistema operativo specifico. Ciò è contrario ad un sistema virtualizzato in cui l'hardware in genere viene estratto da un monitor della macchina virtuale.

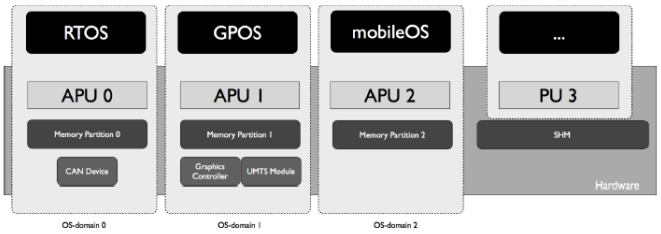


Figura 3.3.: *"Schema composizione di un sistema Multi-OS"*

Generalmente, si assumono due o più sistemi operativi che girano contemporaneamente sulla piattaforma hardware. Questi sistemi operativi sono classificati come sistemi operativi tempo reale, sistemi operativi general purpose, di uso generale, e sistemi operativi mobili in base alle loro capacità. Ogni sistema operativo gestisce una propria partizione di memoria, come riportato in figura 3.3. I dispositivi fisici della piattaforma hardware vengono mappati staticamente ad un sistema operativo specifico. La combinazione del sistema operativo, della memoria e dei dispositivi costituisce un ambiente-OS indipendente e con auto-organizzazione. I vari ambienti-OS sono legati staticamente ad una unità di elaborazione (PU). Questa è di solito una delle unità centrali di elaborazione (CPU) o CPU-core, della piattaforma hardware. La configurazione del sistema è staticamente definita. Ciò significa che non è destinato ad ampliare le partizioni nella memoria principale o di riorganizzare le assegnazioni dei dispositivi durante la fase di run-time, di esecuzione. Durante la fase di avvio tutte le inizializzazioni e le configurazioni necessarie sono impostate nell'ordine per abilitare i diversi

ambienti-OS.
Tecnicamente, la possibilità di isolare tali ambienti multi-OS si basa su due concetti. In primo luogo, ci si avvale di una unità di gestione della memoria a due stadi (MMU), che l'hardware fornisce. La prima fase prevede uno spazio di indirizzamento virtuale per la CPU-core. La seconda fase gestisce la segmentazione e l'isolamento degli spazi di indirizzi fisici e viene utilizzato dal sistema operativo "guest", quello ospite. La configurazione viene impostata durante la fase di avvio in una speciale modalità "hypervisor" privilegiata. Questo livello di sicurezza aggiuntiva protegge la configurazione della fase due dall'accesso degli stessi sistemi operativi ospiti, che sono in esecuzione in un livello meno privilegiato. Ciò consente una forte applicazione del sistema di isolamento. Poichè la piattaforma hardware deve fornire questi due meccanismi per l'approccio multi-OS, nessun monitor della macchina virtuale scritto in software è necessario.

3.3 Architetture multi-OS utili per applicazioni robotiche

Ultimamente sono state sviluppate nuove architetture multi-OS progettate in particolare per la piattaforma SMP, in cui su diverse CPU corrono un sistema operativo real-time RTOS e un sistema operativo general purpose GPOS che non causa modifiche nell'architettura. Con questa nuova architettura è possibile raggiungere un livello elevato di prestazioni a poco costo ingegneristico.
Un prototipo multi-OS chiamato RGMP (un multi processore con un sistema operativo tempo reale, RTOS e uno general purpose, GPOS), in cui i due sistemi non si influenzano a vicenda è stato implementato utilizzando Linux come sistema operativo general puropose, GPOS, e μC OS-II come sistema real-time, RTOS, in modo tale da poterne verificare la fattibilità e questo risulta essere più veloce di RTAI.
Attualmente esistono molti sistemi operativi real-time (RTOS) che forniscono servizi in tempo reale sufficienti per le applicazioni robotiche. Essi sono progettati per una risposta rapida e per scopi deterministici che conducono a un minor supporto per le applicazioni di uso generale. D'altra parte, i sistemi operativi general purpose (GPOS) come Linux hanno numerose risorse per la programmazione, ma forniscono scarsi vincoli per il tempo reale; da qui nasce la necessità di eseguire RTOS e GPOS simultaneamente sulla stessa piattaforma in modo da poter soddisfare le nuove applicazioni di uso generale.
La ricerca del multi-OS è iniziata quando le piattaforme multi-core non erano ancora ampiamente disponibili. Diversi progetti multi-OS (RTOS e GPOS) sono stati originariamente progettati per funzionare sulle piattaforme monoprocessore. Uno dei metodi più utilizzati come architettura per supportare un sistema operativo ibrido è rappresentato di seguito in figura 3.4:

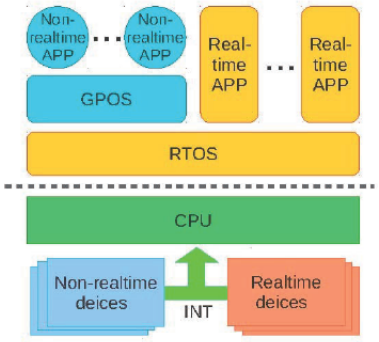


Figura 3.4.: *"Architettura ibrida per il supporto del Multi-OS"*

Tale architettura gestisce un RTOS al livello più basso e un GPOS che viene eseguito a più bassa priorità del processo RTOS in modo che il GPOS possa essere interrotto da un altro processo tempo reale in qualsiasi momento. Sistemi operativi come RTLinux e RTAI sono basati su questa architettura [12].
Questo progetto originariamente prevedeva l'esecuzione dei due sistemi RTOS e GPOS su delle piattaforme monoprocessore, ma ciò causava alcuni problemi come:

- Quando il carico di lavoro del RTOS è pesante il GPOS avrà a disposizione pochi quanti temporali di accesso alla CPU rendendo necessarie applicazioni del GPOS la cui affidabilità è dichiarato.
- La modifica dei GPOS è necessaria per condividere CPU con RTOS e ciò può comportare un carico di lavoro elevato e bug per la complessità dei GPOS.
- Il tempo di reazione all'allarme del GPOS diventa più lungo, perchè RTOS assume il controllo del sistema di allarme che porta "overhead" aggiuntivo.

L'altra tecnica che si utilizza per far coesistere due sistemi operativi è la virtualizzazione; come si può notare dalla figura 3.5, in questo caso si introduce uno strato software tra l'hardware e sistemi operativi chiamato "hypervisor" la cui funzione è di dispacciamento hardware tra sistemi operativi in modo che possano condividere la CPU, la memoria e i dispositivi di I/O. La virtualizzazione può essere di diversi livelli: virtualizzazione completa e para-virtualizzazione.

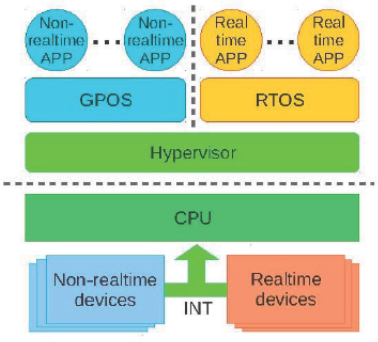


Figura 3.5.: *"Architettura ibrida per il supporto del Multi-OS"*

La virtualizzazione completa ha bisogno di un supporto hardware come la tecnologia VT Intel; in questo caso l'hypervisor è in esecuzione a un più alto livello di privilegio e sotto si trova il sistema operativo ospite. Ogni volta che il sistema operativo ospite accede alla parte comune dell'hardware si genererà una trappola che privilegia e manda in esecuzione "l'hypervisor" che arbitra l'accesso all'hardware. I vantaggi della virtualizzazione completa sono:

- i sistemi operativi ospiti possono essere eseguiti simultaneamente senza alcuna modifica.
- i sistemi operativi ospiti sono isolati e forniscono una protezione più sicura.

Mentre i principali svantaggi risultano essere:

- L'hypervisor si introduce molto in alto tra hardware e sistemi operativi e ciò ridurrà le prestazioni tempo reali.
- Se il sistema operativo ospite esegue sulla stessa CPU, esiste un problema di schedulazione, su come suddividere il tempo della CPU.

A causa del costo delle prestazioni, molte ricerche preferiscono utilizzare la para-virtualizzazione. L'hypervisor è in esecuzione allo stesso livello di privilegio del sistema operativo ospite come modalità kernel, in modo che tutti i sistemi operativi ospiti debbano essere modificati per utilizzare le API fornite dall'hypervisor, invece, di operare direttamente sull'hardware condiviso.
Esiste ancora un altro tipo di virtualizzazione chiamata virtualizzazione asimmetrica, che è un compromesso tra la virtualizzazione completa e la para-virtualizzazione. La sua architettura è come quella della virtualizzazione completa tranne per il fatto che il sistema operativo tempo reale esegue allo stesso livello dell'hypervisor. Questo metodo non solo riduce lo "switch" del RTOS nella virtualizzazione completa, ma mantiene anche il GPOS non modificato. Tuttavia tale tecnica eredita anche la parte di svantaggi di entrambe le virtualizzazioni su cui si basa, come ad esempio, il problema di condivisione del tempo della CPU, elevato overhead del GPOS e ridotto overhead del RTOS.
Con la diffusione di piattaforme multi-core, il problema di condivisione della CPU in sistemi multi-OS può essere risolto facilmente specificando ogni esecuzione del sistema operativo su una CPU che non solo migliora le prestazioni del sistema operativo individuale ma semplifica anche l'implementazione. Tuttavia la maggior parte dei progetti multi-OS nella piattaforma multi-core derivano ancora da metodi monoprocessore. Attualmente alcuni sistemi operativi ibridi sono supportati dalle piattaforme multi-core. Un'attenta configurazione può essere realizzata per isolare i processi in tempo reale da quelli non in tempo reale che vengono eseguiti in diverse CPU risolvendo il problema della suddivisione della CPU. Ma gli altri problemi esistono ancora.
Alcuni studi di ricerca utilizzano il metodo di virtualizzazione nella piattaforma multi-core. Ma ogni CPU ha ancora un hypervisor che introduce overheader e ulteriori modifiche ad ogni sistema operativo ospite.
In questo caso con il termine Robot-OS si intende un sistema operativo progettato per l'esecuzione su robot che fornisce servizi sufficienti per il controllo dei robot.
I sistemi operativi robotici più popolari al giorno d'oggi, come ROS e OROCOS sono costruiti su sistemi operativi general purpose come Linux e Windows, al fine di utilizzare le ricche risorse di programmazione che forniscono. Così essi non possono coprire l'utilizzo in

tempo reale che risulta in un campo di applicazione su cui l'architettura multi-OS può essere applicata.
Le caratteristiche principali della nuova architettura multi-OS appositamente implementata per la piattaforma SMP risultano essere:

- GPOS e RTOS corrono parallelamente sulla propria CPU;
- non c'è nessun strato software aggiuntivo inserito tra il sistema operativo e l'hardware;
- nessuna modifica per il GPOS.

Il prototipo implementato di tale sistema operativo, si chiama RGMP (RTOS e GPOS su multi-processore) e con questo si cerca di dimostrare la sua fattibilità e di misurare le sue prestazioni usando Linux come sistema operativo general purpose, GPOS e μC/OS-II come sistema operativo real time, RTOS.
Questa architettura può essere applicata a qualsiasi piattaforma multi-core, ma il prototipo in questione si basa sull'IA32 SMP.

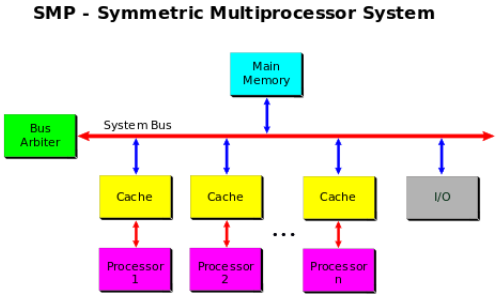


Figura 3.6.: *"Architettura SMP, Sistema Multiprocessore Simmetrico"*

I computer con architettura SMP hanno due o più processori identici connessi ad un'unica memoria principale condivisa. Tradizionalmente essi sono controllati da un unico sistema operativo per eseguire diversi compiti (thread). Attualmente la maggior parte dei sistemi multiprocessore utilizzano un'architettura SMP come quella riportata in figura 3.6.
Nel caso di processori multi-core, l'architettura SMP si applica ai cores, trattandoli come processori separati. Rispetto ai computer con una comune architettura SMP, i computer con architettura IA32 SMP hanno un unico sistema di manipolazione delle interruzioni chiamato architettura APIC, ovvero è un sistema di controllori avanzati di interruzioni programmabili, utilizzato in sistemi di elaborazioni che prevedono sistemi multiprocessore simmetrici. Ogni CPU ha una APIC locale per ricevere messaggi di interruzione generati dall'I/O APIC che raccolgono e gestiscono segnali di interruzione inviati dalle periferiche di I/O. Ogni APIC locale ha un unico ID in questo modo è possibile programmare le I/O APIC per fornire specifiche interruzioni per una specifica CPU e quindi a un determinato sistema operativo (obiettivo 1). Questo viene utilizzato per implementare un controllore delle interruzioni per quello che è il secondo obiettivo e che nelle architetture monoprocessore viene effettuata da un software (RTOS o hypervisor).
Il processo di avvio del computer IA32 SMP avviene nel seguente modo:

- quando il computer viene avviato, una sola CPU, chiamata CPU di "bootstrap" (BSP) inizierà ad eseguire il programma;

- dopo che il BSP ha inizializzato i sistemi hardware e software, allora verranno svegliate le altre CPU, chiamate processori applicativi (AP) da IPI.

L'abilitazione di una delle CPU avviene tramite una funzionalità propria del kernel di Linux chiamata CPU-hotplug, che è appunto utilizzata per attivare o disattivare dinamicamente un core della CPU. Questo può essere fatto chiamando due funzioni del kernel "cpu_up()" o "cpu_down()", rispettivamente per l'abilitazione e la disabilitazione. Quando si chiama la funzione "cpu_down()" con parametro il numero della CPU, i processi e le interruzioni della CPU verranno dirottate ad altre CPU e la CPU verrà disabilitata fino a quando non verrà chiamata la funzione "cpu_up()".
Questa caratteristica è importante per il raggiungimento di quello che è il terzo obiettivo. Con la funzionalità del kernel CPU-hotplug, si può avviare un primo kernel di Linux non modificato, quindi, inserire un modulo del kernel caricabile che chiama la funzione "cpu_down()" per allocare qualsiasi CPU per l'avvio di un RTOS (in questo caso μC/OS-II). Infine, se non abbiamo più bisogno del sistema operativo tempo reale, si può chiamare la funzione "cpu_up()" per tornare indietro e riavviare il sistema operativo general purpose, ovvero Linux.
L'IPI, ovvero l'Inter-Processor Interrupt, un sistema in grado di generare interruzioni che interrompono il normale funzionamento di un processore in un ambiente multiprocessore, è generato da APIC locale della propria CPU ed è gestito da un'altra CPU come interrupt normale. Esiste un IPI speciale, chiamata INIT IPI; se una CPU ha ricevuto una INIT IPI, essa eseguirà un processo di boot, di avvio, dall'indirizzo che gli è stato passato dalla INIT IPI. Questo meccanismo è usato da CPU-hotplug e dalla fase di avvio, quindi dal boot del sistema operativo tempo reale. In questo modo è possibile emettere una INIT IPI con in ingresso l'indirizzo dell'immagine RTOS precaricata per l'avvio in una seconda CPU.
In generale la IPI viene inviata con un numero del vettore di interruzione e la CPU ricevente gestirà l'IPI come una interruzione normale. L'IPI generale viene utilizzata nella comunicazione tra le varie CPU/sistema operativo che costituiscono il sistema. Entrambi RTOS o GPOS possono utilizzare normali IPI per inviare messaggi a vicenda e gestirli immediatamente.

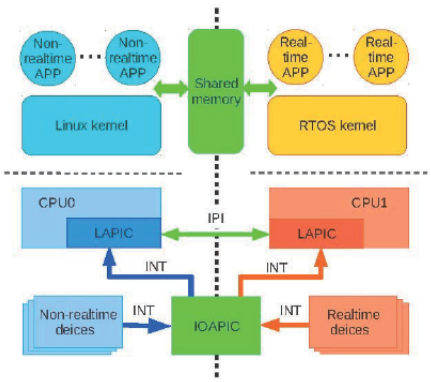


Figura 3.7.: *"Architettura del Multi-OS"*

Come si può ossevare dalla figura 3.7, GPOS e RTOS vengono eseguiti su CPU separate indipendenti. I dispositivi sono divisi in dispositivi in tempo reale e dispositivi non in tempo reale gestiti rispettivamente da RTOS o da GPOS. L'APIC è usato come un router di interruzione per consegnare direttamente ai due tipi di dispositivi le interruzioni dei sistemi

operativi corrispondenti.
Dato che il problema chiave del multi-OS è la condivisione dell'hardware, l'obiettivo è quello di minimizzare l'hardware condiviso in modo tale da ridurre la complessità e l'attuazione aumentando le prestazioni. In questo modo il sistema può essere diviso in due parti controllate individualmente da GPOS e RTOS che hanno solo la memoria, come hardware comune. GPOS e RTOS comunicano con la memoria condivisa. Questo è il modo più veloce, più efficiente e conveniente di comunicare nel campo dei sistemi distribuiti rispetto alla rete o alle altre vie di comunicazione I/O. Il modello di programmazione effettiva può variare (messaggio FIFO,ecc), ma attualmente per il passaggio delle informazioni ciò che viene usato è la memoria condivisa dell'architettura SMP.
Entrambi i sistemi operativi RTOS e GPOS prendono tutto il controllo della propria CPU, in modo che nessun livello software possa essere inserito tra l'hardware e il sistema operativo. L'interruzione è gestita direttamente dal gestore delle interruzioni di ogni sistema operativo senza causare ritardi nell'esecuzione dei software nei sistemi operativi ibridi o nei metodi di virtualizzazione. Inoltre, il fatto che i due sistemi operativi siano eseguiti su diverse CPU non solo risolve il problema di condivisione del tempo della CPU, ma assicura anche che due sistemi operativi non si influenzeranno a vicenda.
Quindi, in un sistema multi-OS i problemi che si cercano di risolvere sono quelli che rispondono a queste domande:

- Come avviare due sistemi operativi su CPU separate in modo indipendente?
- Come gestire i dispositivi di I/O del sistema?
- Come comunicare tra due sistemi operativi?

Al fine di avviare due sistemi operativi e assicurarsi che nessuna modifica venga apportata al kernel di Linux, il modo più semplice e flessibile consiste nell'usare il modulo caricabile e il CPU-hotplug caratteristico di Linux. Non appena viene avviato Linux, esso prende il controllo di tutto l'hardware, quindi si inserisce il modulo del kernel RGMP e si avvia il monitor RTOS. Il monitor RTOS controllerà il modulo del kernel RGMP per caricare l'immagine nella memoria RTOS, chiama la funzione "cpu_down()" per disabilitare una CPU ed emettere un INIT IPI con in ingresso l'indirizzo dell'immagine RTOS che quella CPU dovrà avviare, dopo di che l'RTOS in esecuzione sulla CPU sarà disabilitato da Linux. L'RTOS può anche essere spento o riavviato in modo dinamico. Questo viene fatto dal monitor RTOS per controllare il modulo del kernel RGMP per disattivare l'RTOS in esecuzione e chiamare la funzione "cpu_up()" per allocare la CPU per la nuova esecuzione di Linux.
In questo caso, i dispositivi sono divisi in dispositivi tempo reale e quelli non in tempo reale che sono controllati da Linux e hanno normali driver di periferiche Linux; i dispositivi in tempo reale, invece, sono controllati da RTOS e hanno driver di periferica RTOS. I driver Linux di dispositivi tempo reale sono configurati per essere disabilitati. Il driver di periferica tempo reale di RTOS imposterà una I/O APIC per instradare la sua interruzione al sistema operativo tempo reale, RTOS. Quando si avvia RTOS, Linux passerà la tabella delle interruzioni IRQ (Interrupt Request) che contiene gli indirizzi di memoria dei gestori delle interruzioni, come parametro. Ciò può essere utilizzato dai driver delle periferiche del sistema operativo tempo reale in fase di assegnazione dei numeri delle IRQ per evitare che questi creino conflitto con i numeri delle interruzioni IRQ dei driver delle periferica Linux.
Se due periferiche di tipo diverso hanno bisogno di condividere lo stesso numero di IRQ, allora quella non in tempo reale deve essere disabilitata. Tuttavia, se l'hardware MSI (Message Signaled Interrupt) è supportato, il dispositivo può essere programmato per fornire le

interruzioni direttamente alla corrispondente CPU senza scavalcare APIC I/O che risolve il problema.
La memoria condivisa dell'architettura SMP è utilizzata per implementare un gestore dei messaggi bidirezionale, una FIFO (First In First Out). Il più grande problema della comunicazione è la sincronizzazione dei due compiti non solo perchè lavorano su diverse CPU, ma anche su diversi sistemi operativi. Per questo motivo si utilizzano due approcci: la mutua esclusione e la sincronizzazione.
Gli accessi in mutua esclusione sono adatti per il breve termine e la sincronizzazione multiprocessore. In questo caso è stato implementato un nuovo metodo di "spinlock" usato da Linux e RTOS. Quando si accede ad una variabile condivisa come i puntatori di lettura e scrittura della coda FIFO condivisa, allora lo "spinlock" deve essere acquisito prima.
Per quanto riguarda la sincronizzazione, assumiamo che A e B siano due task che eseguono su diversi sistemi operativi e che hanno stabilito una coda FIFO per comunicare tra di loro, ad esempio il task A scrive nella coda e il task B legge da essa. La sincronizzazione avviene quando il task A sta per scrivere, ma il buffer è pieno o il task B sta per leggere, ma il buffer è vuoto. Nel primo caso, il task A chiamerà la funzione "sleep()" e rimarrà sospesa fino a quando il task B non avrà terminato la sua operazione di lettura e chiamerà una funzione con il compito di risvegliare il task A che quindi verrà rimesso in esecuzione. Il problema è che le funzioni che possono chiamare i vari task sono relative al sistema operativo su cui eseguono e non hanno influenza sugli altri, quindi il task B non può risvegliare il task A che esegue su un altro sistema operativo. La soluzione è quella di utilizzare una IPI: ovvero si divide la funzione che dovrebbe risvegliare il task in attesa in due parti, una che esegue sul sistema operativo del task B e che una volta chiamata, invia una IPI al sistema operativo del task A; quando il sistema operativo del task A riceve la IPI, il suo gestore delle interruzioni, genererà l'interruzione in grado di risvegliare il task A e rimetterlo in esecuzione per portare a termine il suo compito. Lo stesso avviene nel caso opposto, cioè se si blocca il task B e il task A lo deve risvegliare dopo aver eseguito il suo compito. In questo modo, nessun sistema operativo verrà a contatto con il codice degli altri sistemi, garantendo così un buon isolamento e garantendo che le performance di entrambi i sistemi operativi dipendano solo dalla loro implementazione.

Capitolo 4

Hardware e Software utilizzati

4.1 E4Coder

E4Coder è un insieme di strumenti che può essere usato per simulare gli algoritmi di controllo e per generare codice per microcontrollori embedded che eseguono in presenza o in assenza di un sistema operativo tempo reale.
Esso è pensato come un insieme di pacchetti di "toolbox" per Scicos e Scilab, un software libero con il quale è possibile realizzare delle simulazioni e creare modelli di sistema. In particolare E4Coder include:

- E4Coder Code Generation, un efficiente generatore di codice embedded per microcontrollori e per sistemi con scopi generali;
- SMCube, un modello di una macchina a stati finiti e un generatore di codice;
- E4CoderGui, uno strumento di prototipizzazione per la generazione di un'interfaccia utente grafica.

Le principali caratteristiche di E4Coder sono:

- supporto per hosts Linux e Windows;
- simulazione di progetti tempo continui e tempo discreti;
- debug, simula e genera codice per una macchina a stati finiti (FSM) con stati parelleli supportati;
- debug, simula e genera codice per pannelli di interfaccia utente embedded personalizzati attraverso l'utilizzo di E4CoderGui;
- genera codice C/C++ da Scilab e diagrammi XCOS per qualsiasi piattaforma embedded;
- genera codice multithread compatto e rileggibile per modelli embedded con frequenze multiple;
- utilizzo di RAM e FLASH ottimizzato;
- semplice possibilità di introdurre nuove MCUs o nuovi hardware personalizzati;
- generazione di librerie di codice per piccoli microcontrollori senza un sistema operativo tempo reale (RTOS);
- generazione di un progetto completo tempo reale per:
  - OSEK/VDX basati su RTOS che includono l'RTOS Erika;
  - target RTAI o Interfacce di Applicazione Real Time;

  - target BareMetal che includono supporti per la semplice generazione di codice nell'anello principale;
  - Windows soft real-time che includono pannelli E4Coder GUI per generare semplici prototipi GUI

- generazione del codice personalizzata con dati personalizzati e nomi di variabili; crea blocchi personali per integrare le applicazioni esistenti, funzioni e dati nei progetti Scicos o Scilab.

Tale strumento è semplice da usare perchè i flussi di sviluppo sono basati sul modello; si passa dalla modellizzazione di un sistema complesso alla sua implementazione in uno o più sistemi microcontrollori. Un'altra caratteristica che ne semplifica l'utilizzo è la prototipizzazione dell'interfaccia grafica, ovvero si ha l'interazione tra il modello e lo stato della macchina attraverso la progettazione di pannelli GUI [7].
L'idea che guida E4Coder è la necessità di un leggero insieme di strumenti basati su librerie open-source, che ci permettono di usare un software open-source nel target microcontrollore.

4.1.1 Generazione automatica di codice da modelli per diverse piattaforme

La metodologia di sviluppo della progettazione basata sui modelli (Model-Based Design) è ampiamente utilizzata in diversi domini di applicazioni come l'automotive aerospaziale e il controllo. Tale tipo di progettazione può fornire una significativa riduzione del tempo di sviluppo e del costo dei software embedded che fa leva sulla verifica e sulla validazione.
Nella progettazione basata sui modelli, un modello della funzione di controllo è sviluppato insieme al modello del sistema da controllare; il comportamento di tale sistema all'interno e possibilmente anche al di fuori dell'intervallo delle specifiche è verificato da simulazioni, e da modelli di controllo nel caso di sistemi di sicurezza critici. La tecnica della generazione automatica di codice è, allora, usata per generare un'implementazione del software che risulta essere equivalente al modello riducendo la possibilità di aggiungere errori durante la fase di scrittura manuale del codice [6].
Sono diversi i prodotti commerciali che permettono di eseguire modellazione, simulazione e verifica del sistema tra i quali si ricordano "Simulink", "LabView", "Scade"; di questi fa parte anche "Scicoslab", il cui utilizzo per lo sviluppo industriale di controlli embedded è stato limitato a causa di un numero di problemi che comprendono la mancanza di supporto per la modellazione della macchina a stati finiti, un generatore di codice non adatto a sistemi con disponibilità di memoria limitata, nessun supporto per l'implementazione multithread e specifiche piattaforme per il codice che includono l'utilizzo di periferiche di ingresso/uscita.
Gli strumenti sviluppati per cercare di risolvere il problema comprendono:

- uno strumento per la modellazione, simulazione e sintesi automatica software di una macchina a stati finiti (FMS), chiamata SMCube; tale strumento supporta macchine a stati finiti concorrenti e gerarchiche e fornisce una semplice interfaccia grafica per la modellazione e la simulazione, producendo animazioni e tracciati;

- E4CoderGUI, un prototipo di GUI basato sulle librerie grafiche Qt, con un editor grafico, un simulatore e un generatore di codice "run-time" per piattaforme Linux;

- E4CoderCG, un framework per la generazione di codice basato su tecniche di trasformazione "model-to-model" e "model-to-code", o ancora, per un'efficiente generazione di codice di una implementazione multithread o di un modello multirate;

- supporto per periferiche di ingresso/uscita nel modello e in fase di generazione di codice attraverso l'uso di uno strato di astrazione hardware; l'integrazione con le periferiche di ingresso/uscita e l'acquisizione dati su Linux e RTAI è fornita da insiemi di blocchetti personalizzabili in base alle proprie necessità

4.1.2 Progettazione del sistema

Gli strumenti SMCube e E4CoderGUI sono sostituiti da due moduli:

- un modulo eseguito al tempo di simulazione;
- un modulo per la generazione del codice da eseguire nel target.

In figura 4.1 si riporta lo schema di realizzazione di un'applicazione con il generatore di codice:

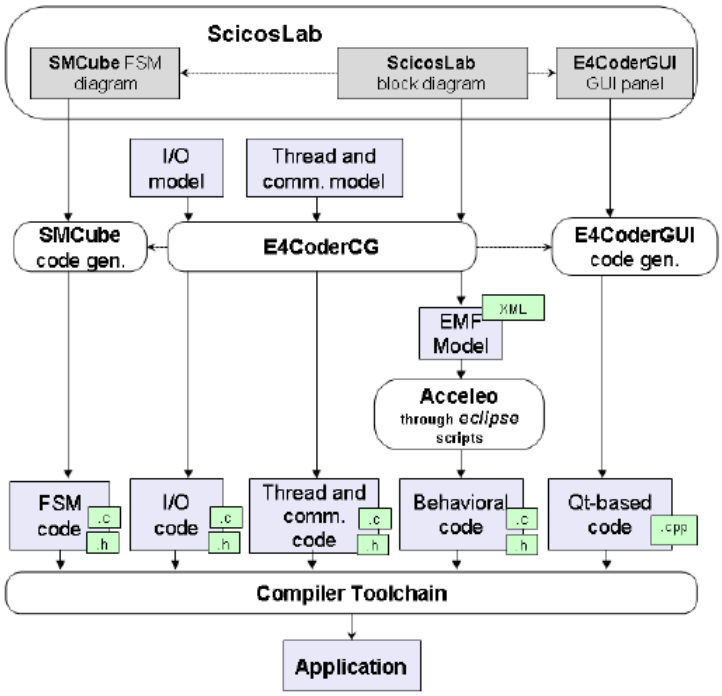


Figura 4.1.: *"Schema per la realizzazione di una applicazione"*

Il meccanismo standard della definizione dei blocchi personalizzabili, definito nel modello per ogni macchina a stati o sottosistema GUI, collega i moduli, eseguiti in fase di simulazione, con l'ambiente ScicosLab. Tale blocco è collegato agli altri blocchi del modello attraverso porte di ingresso/uscita standard e apre reti locali di comunicazione con un programma esterno, fornendo la simulazione del sottosistema a stati finiti o dell'interfaccia GUI.
Il codice per ogni sottosistema SMCube e E4CoderGui è generato da una macchina dedicata che processa le informazioni di modello (codificate in un file XML) e genera una singola funzione step che calcola in fase di esecuzione lo stato e l'uscita aggiornata della funzione del sottosistema. Il codice generato per il modello costruito può essere usato per scopi differenti. La generazione di codice è supportata da tre differenti moduli quali la definizione del flusso di dati del modello e della sua struttura, setup del thread e definizione delle periferiche di ingresso/uscita. In ogni caso la prima fase della generazione di codice è eseguita attraverso trasformazioni "model-to-model" e "model-to-text".

Il modello della creazione del thread ha una macchina dedicata e l'accesso alle periferiche di ingresso/uscita è eseguito attraverso uno strato di astrazione che permette una visione di tali periferiche nel modello della piattaforma. Un generatore separato fornisce la configurazione dello strato di astrazione basato sulle informazioni specifiche dell'hardware [6].

4.1.2.1 Generatore di codice E4CoderCG

Il generatore di codice E4CoderCG genera l'implementazione del codice C del modello costruito su ScicosLab. La generazione del codice può essere eseguita per l'intero modello o per un sottoinsieme selezionato dei suoi sottosistemi. Il modello è esportato (attraverso un formato XML) sull'ambiente di sviluppo Eclipse EMF come un modello o un file ".ecore", file principale di eclipse con il quale è possibile gestire tutto il modello, e processato da una trasformazione "model-to-text" definita utilizzando il tool "Acceleo", un generatore di codice che implementa le conversioni "model-to-text". Disaccoppiando la generazione di codice attraverso un modello intermedio, si ottiene una maggiore flessibilità e un maggiore controllo nella fase di generazione, aprendo la porta per la generazione di codice da altri linguaggi e o tools con una maggiore integrazione con altri tool quali Eclipse.
La progettazione di un sistema di controllo basata sul modello, consiste tipicamente di una fase di simulazione iniziale in cui la logica di controllo è verificata contro il modello del sistema che deve essere controllato. Durante questa fase iniziale la piattaforma hardware può non essere disponibile ed è virtualizzata o rimpiazzata da un ambiente prototipizzato, permettendo così l'astrazione e la generalizzazione dei blocchi che rappresentano le periferiche. Un'aggiuntiva porta di ingresso o di uscita permette allora di agire come un qualcosa che gli passa attraverso al tempo di simulazione.
In fase di generazione di codice, ogni generica periferica è mappata sull'attuale periferica di ingresso/uscita dell'hardware considerato; in questo modo, lo stesso diagramma può essere usato per differenti target hardware senza apportare modifiche. Inoltre lo stesso diagramma può essere usato sia in fase di generazione che di simulazione.
Nella generazione di una implementazione multithread di un modello multirate, il modello complessivo deve essere composto da un insieme di superblocchi di primo livello, sottosistemi, ognuno attivato da un singolo blocco di clock, in questo modo ogni sottosistema avrà la sua frequenza di campionamento. Tali sottosistemi comunicano tra di loro attraverso opportune porte e con l'esterno attraverso porte rimappate sulle periferiche di ingresso/uscita. La semantica di esecuzione sincrona e la dipendenza dalle porte di ingresso/uscita determinano un ordine parziale nell'esecuzione dei blocchi; quando due superblocchi di primo livello scambiano dati utilizzando oppotune porte dati, il generatore di codice contrassegna ogni porta utilizzando un blocco di lettura/scrittura, inserendolo automaticamente se questo non è presente. Tale blocco è implementato usando un buffer di memoria protetto da mutex o dalla disabilitazione delle interruzioni, che dipendono dal target scelto. Miglioramenti futuri includeranno la comunicazione utilizzando le risorse libere in attesa, come "fieldbus" o altri bus di comunicazione permettendo così la distribuzione del codice generato nei sistemi distribuiti.
Come già accennato, il generatore di codice fornisce supporto per diversi target che operano su sistemi e altri prodotti attraverso un modello di esecuzione comune che consiste di un insieme di astrazioni che devono essere implementate da qualsiasi piattaforma supportata. Nonostante ogni sistema operativo abbia caratteristiche proprie, ognuno di essi richiede solo l'implementazione di un numero limitato di meccanismi base come richiesto dall'esecuzione di un modello.
Alla base del codice generato si trova:

- un task, come una sequenza di chiamate di funzioni derivate dall'esecuzione del superblocco di primo livello in un dato ordine; ogni task ha bisogno di essere attivato in accordo al suo periodo dato dalla frequenza di campionamento del superblocco mappato in esso;

- una priorità assegnata ad ogni task e usata per garantire il corretto ordine di esecuzione dei superblocchi e assicurarne la schedulabilità;

- un background time, ovvero il tempo in cui il task non esegue, utilizzato dal modello per eseguire gli aggiornamenti come quelli delle GUI;

- inizializzazione e fine di un task che include anche l'inizializzazione delle periferiche.

Durante il tempo di esecuzione è necessario garantire la consistenza dei dati condivisi, implementando segnali scambiati tra superblocchi mappati su task differenti.
In particolare E4CoderCG è in grado di generare codice per i seguenti target:

- Linux con librerie pthread;

- RTAI (Real Time Application Interface);

- OSEK/VDX Erika Enterprise (sistema operativo realtime);

- Bare Metal;

- Windows.

Per permettere un più semplice accesso all'acquisizione dei dati dalla scheda, il codice generato per i sistemi operativi Linux e RTAI è supportato dalle librerie Comedi; inoltre nel caso in cui il codice sia generato per Linux, il task è implementato usando i pthread e la sua priorità è mappata nella piorità realtime del pthread. La periodicità è implementata dalla chiamata della funzione superblocco all'interno di un ciclo continuo. Per quanto riguarda la condivisione dei dati tra task in un sistema multirate, si ha che questa è implementata utilizzando un mutex a "priority inheritance" per proteggere i dati durante l'accesso concorrente.
Come mostrato in figura 4.2, l'implementazione per RTAI è simile a quella per sistemi Linux, ma in questo è necessario considerare la comunicazione con l'interfaccia GUI. La comunicazione tra un task periodico real-time e il principale ciclo Qt è fatto attraverso slots Qt, anche se dal codice Qt è possibile richiamare normali primitive Linux portando così ad una automatica transizione del task da RTAI real-time a Linux. Per evitare questo rischio il generatore di codice aggiunge un thread ponte aggiuntivo per ogni blocco E4CoderGUI, che è responsabile della comunicazione tra il task realtime e il framework Qt. La comunicazione con il task realtime è gestita da un buffer circolare non bloccante per le operazioni di scrittura, quindi, nel caso in cui il buffer risulti pieno e si va ad effettuare un'altra operazione di scrittura i dati saranno sovrascritti.

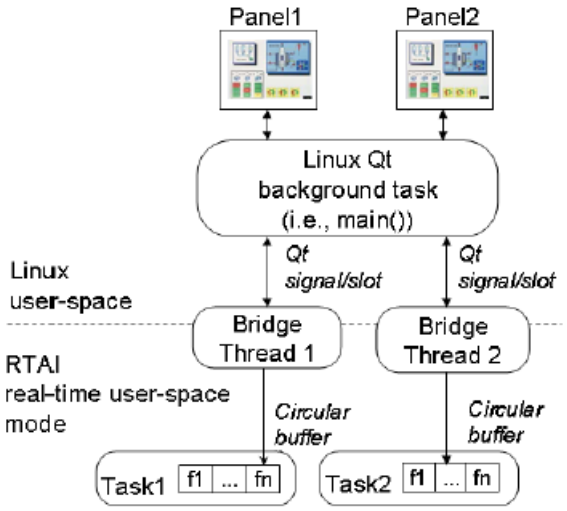


Figura 4.2.: *"Flusso di generazione di codice per un'applicazione Linux RTAI"*

Nel caso in cui si abbia a che fare con sistemi operativi realtime OSEK/VDX come Erika, si deve considerare la limitazione sulla quantità di primitive disponibili oltre alle limitate risorse dell'hardware; per questo motivo non tutti i supporti "run-time" disponibili in Linux sono anche disponibili nei sistemi OSEK/VDX. Nonostante queste limitazioni il codice generato è sempre caratterizzato da un task che in questo caso sarà implentato usando i task OSEK/VDX, la priorità, la periodicità e l'offsets che vengono implementati dalla configurazione di un allarme in un oppotuno file (file oil). Tale allarme è connesso ad un contatore attivato periodicamente da un timer; la periodicità del task è espressa come multipli della frequenza del timer. L'inizializzazione di ogni superblocco è fatta in un task dedicato, attivato all'avvio del sistema prima che partano tutte le attività periodiche, mentre la funzione che fa terminare il task non risulta implementata, in quanto, si suppone che in una esecuzione tipica del microcontrollore il task non smetta mai di eseguire. Per quanto riguarda la condivisione dei dati, invece, questa è implementata disabilitando le interruzioni in quanto la dimensione dei dati e il tempo necessario per leggere o scrivere è trascurabile rispetto ai tempi richiesti dalle primitive RTOS. In questo caso, invece, non vengono gestite attività background e l'interfaccia GUI.
Infine, nei sistemi "Bare Metal", ovvero quelli senza sistema operativo, l'implementazione di un sistema realtime non preemptive si realizza con una sequenza di chiamate di funzioni all'interno della funzione principale. Tuttavia questo approccio funziona bene per piccoli impianti con funzionalità limitate. In questo caso il task è implementato nel main, non c'è schedulabità, la periodicità e l'offset sono implementati usando degli interi che vengono incrementati dalle interruzioni di un timer. L'inizializzazione dei superblocchi è fatta sempre all'interno del main e la condivisione dei dati è implementata con una semplice copia dei dati. Anche in questo caso non vengono utilizzate le interfacce GUI [6].

4.1.2.2 SMCube

SMCube è un modellatore di una macchina a stati finiti gerarchica e concorrente che permette la simulazione interattiva e la generazione di codice in C, che offre un editor e una visuale della simulazione. Tale tools fa particolare attenzione nel ridurre il numero di possibili insidie legate alla semantica e per garantire che il codice generato abbia lo stesso comportamento del modello simulato.
Un ordine di esecuzione deve essere assegnato per tutte le transizioni e per gli stati paralleli

per garantire un'esecuzione deterministica del codice; in più le azioni associate con le transizioni non possono produrre eventi che innescano altre transizioni della stessa macchina [6]. In figura 4.3, si può osservare l'ambiente di sviluppo per macchine a stati finiti:

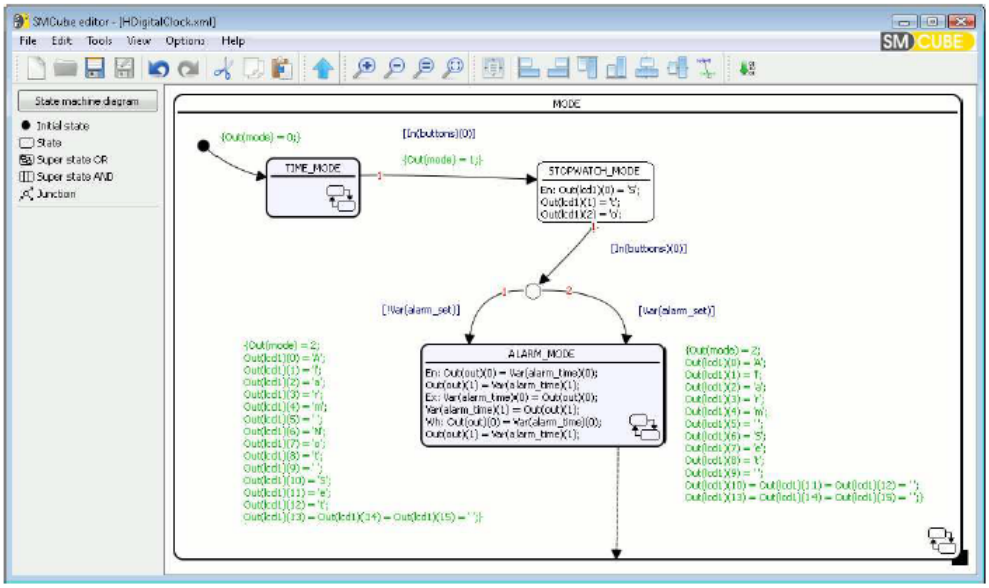


Figura 4.3.: *"Editor SMCube per la creazione di un modello a stati finiti"*

4.1.2.3 E4CoderGUI

E4CoderGUI è uno strumento per la progettazione e la simulazione di una interfaccia utente grafica (GUIs). La parte di progettazione su E4CoderGUI è realizzata attraverso un editor che consente l'utilizzo di testo, immagini e un numero di altri "windgets" che associano un insieme di azioni predefinite al valore dell'ingresso e dell'uscita o al verificarsi di alcune condizioni su di essi.
L'inserimento di un blocco di E4CoderGUI in un diagramma ScicosLab e di conseguenza la sua simulazione fornisce all'utente mezzi per interagire con la simulazione, analizzando i risultati o simulando le operazioni di un touch-screen o di un front-end fisico. In questo modo, si permette all'utente di testare il comportamento del sistema prima della sua distribuzione.
Il blocco E4CoderGUI può essere usato in fase di generazione di codice e in questo caso il risultato è un'applicazione Qt che mostra lo stesso pannello costruito con l'editor e validato durante la simulazione. Il codice così generato può essere usato nelle piattaforme e applicazioni dove il controllo real-time e l'interfaccia uomo-macchina (HMI) risiedono nello stesso eseguibile.
Particolare attenzione deve essere fatta quando si genera codice per sistemi in tempo reale a causa del rischio di compromettere le performance real-time della parte di controllo nel momento in cui si esegue il codice GUI generato [6].

4.1.3 Strutture generate dalla generazione di codice

Il codice che si ottiene dalla generazione automatica, quindi, risulta essere strutturato nei seguenti file:

- *data.h* un file header che contiene i prototipi delle funzioni contenute e implementate nel file data.c

- *data.c* file che contiene il codice delle funzioni ottenute dal diagramma Scicos; contiene le funzioni di inizializzazione, aggiornamento e fine di ogni superblocco generato;
- *types.h* file che contiene la definizione dei tipi usati e specificati nell'ultimo pannello che si ottiene durante la generazione di codice;
- *target.c* file che contiene l'implementazione delle funzioni specifiche del target;
- *target.h* file che contiene i prototipi delle funzioni implementate nel file *target.c*;
- *target_name.h* file che contiene le inclusioni dei file header e le specifiche macro del target selezionato;
- *task.c* file che contiene l'implementazione del task come specificato dall'utente durante la fase di generazione del codice;
- *main.c* file che contiene il punto di ingresso dell'applicazione.

In base al sistema operativo per il quale si vuole generare il codice tali file potrebbero aumentare.

4.2 UdooNeo

La board UdooNeo, che si osservare in figura 4.4 nasce dall'unione e dalla fusione di:

- Arduino;
- Raspberry-Pi;
- sensore a 9 assi;
- Wi-fi;
- Bluetooth 4.0.

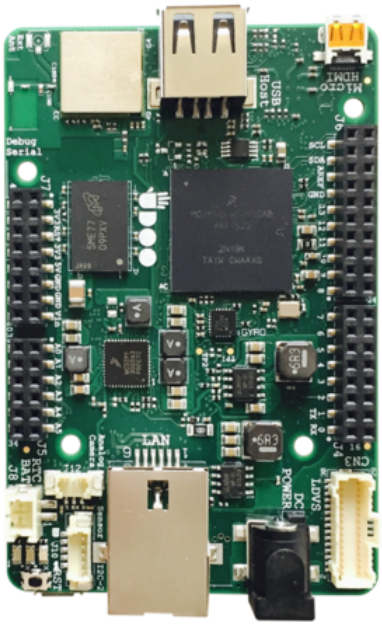


Figura 4.4.: *"Scheda UdooNeo"*

Tale scheda, come mostrato in figura 4.5, incarna un nuovo concetto, un singolo computer board appropriato per l'era post-PC, infatti:

- come una Raspberry-Pi può essere programmata con un qualsiasi linguaggio di programmazione ed eseguito in un ambiente completamente linux con interfacce grafiche;
- si riscontra tutta la semplicità di una scheda compatibile con Arduino, grazie al Cortex-M4 e alla configurazione pinout di ArduinoUno, con la possibilità di aggiungere sensori e attuatori sia digitali che analogici;
- un incredibile "smoothly-running" Android offre la possibilità di costruire nuove periferiche smart basate su Android;
- è dotata di un modulo Wi-fi, un modulo ethernet e un modulo bluetooth a bassa energia;
- la presenza dei sensori di moto a 9 assi sono montati per creare robot/droni 3D o per creare nuovi tipi di iterazioni con il mondo reale;
- hardware open-source.

L'idea dello sviluppo di tale scheda nasce dalla considerazione che in questi anni continuano ad aumentare sempre più gli aspetti della vita quotidiana in cui è necessaria l'iterazione con un dispositivo intelligente; nei prossimi anni le periferiche collegate aumenteranno sempre più fino ad arrivare ad un mondo pieno di iterazioni automatiche tra oggetti intelligenti, "Internet of Things". UdooNeo viene progettata proprio per portarci verso questo nuovo ambiente e per rendere più facile la gestione di queste iterazioni [5].

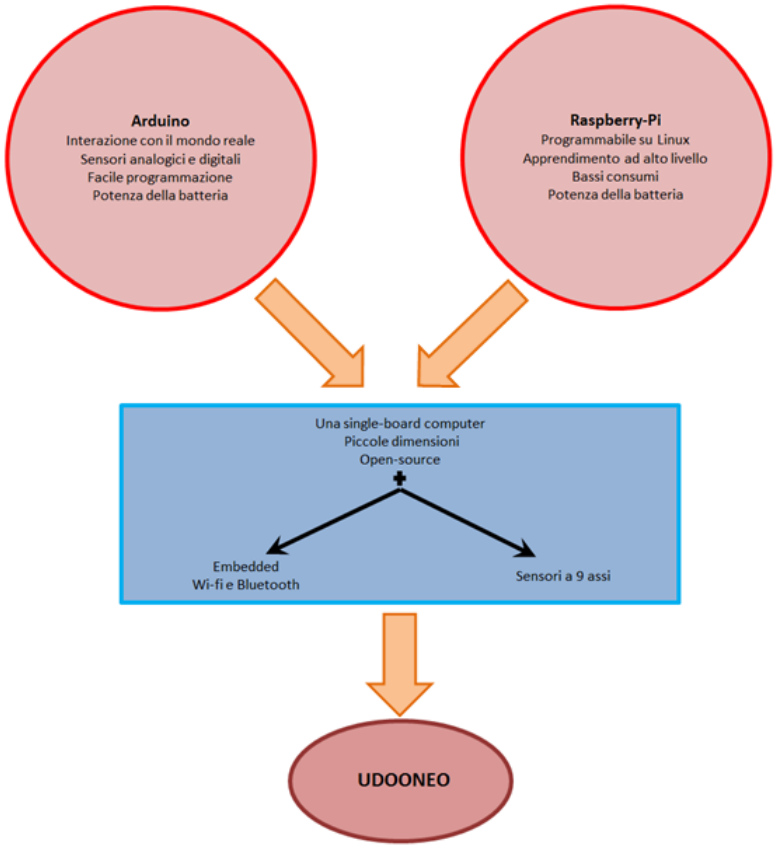


Figura 4.5.: *"Caratteristiche della board UdooNeo"*

La scheda UdooNeo offre un nuovo paradigma nel nel panorama delle single-board computer, da un processore omogeneo a uno eterogeneo, ottenuto incastrando due core nello stesso processore: un ARM Cortex-A9 a 1 GHz ed un Cortex-M4 I/O a 166 MHz co-processore real-time in uno stesso chip, l'iMX6 Solo-X della Freescale.

4.3 QBMOVE

L'idea dei cubotti a cedevolezza variabile nasce dalla considerazione che nell'iterazione tra l'uomo e un braccio robotico è necessario garantire un elevato margine di sicurezza, ovvero, è necessario garantire che in nessuna circostanza, che sia essa relativa al normale funzionamento o al verificarsi di una situazione non prevista, il braccio robotico causi danni alle persone con cui interagisce direttamente o indirettamente. Tenendo presente questo come obiettivo principale, l'altro obiettivo importante riguarda il garantire l'accuratezza e la rapidità nel compiere movimenti quando questi vengono richiesti.
Garantire la sicurezza del robot comporta diverse considerazioni che dipendono da vari fattori che variano dall'affidabilità del software, ai possibili mal funzionamenti meccanici, agli errori umani che si possono commettere nel collegare il braccio con la macchina. Per questo motivo è necessario effettuare un'analisi completa dei rischi secondo procedure metodiche che variano a seconda degli scopi per cui viene richiesto il braccio.
Nello specifico, si riserva particolare interesse nel valutare la situazione di rischio in cui durante l'esecuzione di un movimento pianificato del manipolatore, si verifica una collisione tra un link del braccio e l'umano con cui interagisce, portando, quindi, ad un'analisi su come dovrebbero essere progettati i meccanismi dei robot e i controllori per garantire una migliore e più sicura iterazione tra l'uomo e il manipolatore. I manipolatori progettati per condividere l'ambiente con gli umani, ad esempio, in ambiente domestico, in ambiente assistito per la riabilitazione o altre applicazioni mediche devono soddisfare requisiti diversi rispetto ai manipolatori progettati per un uso industriale.
Una soluzione per aumentare la sicurezza e l'affidabilità di tali manipolatori può essere quella di aumentare il numero di sensori a disposizione del manipolatore e utilizzare degli algoritmi di controllo più appropriati; tuttavia questo approccio non può dimostrarsi robusto per l'intero manipolatore, bisogna, infatti, tenere in considerazione quelle parti del braccio che non sono dotate di sensori e delle limitazioni al tipo di controllo che si può usare dovute all'inerzia del corpo meccanico e dell'attrito [1].
Un altro approccio per aumentare il livello di sicurezza dei bracci robotici che interagiscono con gli umani è quello di introdurre la cedevolezza meccanica direttamente nella fase di progettazione del robot, in modo tale da riuscire a disaccoppiare dinamicamente nel momento in cui si verifica un impatto, l'inerzia del rotore dell'attuatore dal link del braccio. Naturalmente la presenza della cedevolezza andrà ad influire poi sulle performance del braccio, principalmente, in termini di incremento delle oscillazioni. Compito del controllo sarà quello di garantire il raggiungimento della posizione e della rigidezza accordata, nonostante la maggiore cedevolezza. Una delle tecniche che si propone per garantire comunque elevate performance nel funzionamento del braccio robotico è, quindi, quella della cedevolezza variabile (VSA).

4.3.1 Indici di sicurezza

Come già detto sopra, il principale obiettivo nell'interazione uomo/robot è quello di garantire la sicurezza durante la durata dell'intera collaborazione.
Diversi studi ed esperimenti sono stati condotti per valutare il rischio e il danno che si avrebbe nel caso in cui si verifichi un incidente durante il regolare funzionamento del braccio

robotico; in particolare, nel caso peggiore si assume che l'impatto con l'uomo possa verificarsi in qualsiasi istante durante l'inseguimento di una traiettoria pianificata. L'entità del danno è stata studiata in ambiente biomeccanico portando all'introduzione di diversi indici che permettono di valutare in modo più immediato la gravità dell'impatto [1]. I principali indici utilizzati sono:

- Indice di gravità di Gadd, GSI (Gadd Severity Index)

  $$GSI = \int_0^t a^{2.5} d\tau$$

  dove a è l'accelerazione della testa e t rappresenta la durata della collisione;

- criterio del danno alla testa, HIC (Head Injury Criterion)

  $$HIC = T\left[\frac{1}{T}\int_0^T a(\tau)d\tau\right]^{2.5}$$

  dove T rappresenta l'istante finale dell'impatto;

- criterio della viscosità, VC (Viscous Injury);

- criterio "3 ms";

- Indice del Trauma Toracico, TTI (Thoracic Trauma Index).

Nel caso dell'indice GSI, si ha che il valore GSI = 1000 è considerato il livello di soglia o il limite di tolleranza da non superare per non avere un danno grave alla testa. Invece, per quanto riguarda l'indice HIC richiede particolare attenzione la scelta di T, per cui, considerando il caso peggiore, si procede scegliendo T come l'istante di tempo in cui il movimento della testa raggiunge la massima velocità, v(T), con T che solitamente risulta essere minore di 15 ms. Un valore dell'indice HIC pari a 1000 indica un danno piuttosto grave alla testa, mentre un valore dell'indice pari a 100 è associato alle normali collaborazioni uomo/robot. L'indice HIC può essere generalizzato se come zona coinvolta nell'impatto non consideriamo più la testa ma una qualsiasi altra parte del corpo e in questo caso l'indice viene espresso come:

$$HIC' = T^{1-\alpha} v(T)^{\alpha}$$

4.3.2 VSA: Attuatore a rigidità variabile (Variable Stiffness Actuator)

Oltre all'elevata sicurezza, l'altro principale obiettivo che si vuole raggiungere nelle collaborazioni uomo/robot è quello della performance, che spesso, a seconda del manipolatore con cui si interagisce, viene espressa in termini di velocità del moto; più in generale si può affermare che una macchina che si muove piano è si, meno pericolosa di una che si muove più veloce, ma spesso, a seconda delle applicazioni delle macchine troppo lente risultano inaccettabili in termini di qualità e quantità del lavoro effettuato.
Oltre alla sensorizzazione della parte in movimento e al controllo attivo del braccio, le altre soluzioni che fino ad ora sono state adottate per garantire la sicurezza consistono nella trasmissione elastica passiva o nella minimizzazione dell'inerzia del link e del motore con il progetto di una nuovo meccanismo di trasmissione.

Una delle tecniche studiate di recente, che cerca di garantire contemporaneamente sia la sicurezza che la performance è l'approccio a impedenza variabile, tecnica che durante l'esecuzione di un compito ci permette di variare l'impedenza meccanica del sistema di attuazione; in particolare il modo in cui varia l'impedenza è determinato dalla soluzione del problema di controllo ottimo. Dalle simulazioni effettuate in tali condizioni, si può notare che il meccanismo ad impedenza variabile impone elevati valori di cedevolezza a basse velocità e bassi valori di cedevolezza per elevati valori della velocità, consentendo, così, il disaccoppiamento dell'inerzia del link del braccio da quella dell'attuatore [2].
L'implementazione del principio dell'impedenza variabile da un punto di vista meccanico, porta allo sviluppo degli attuatori a rigidezza variabile (VSA).

4.3.2.1 VSA-I

Almeno nella fase iniziale, l'attuatore a rigidezza variabile è stato pensato per la sicurezza e le performance dei robot nell'ambiente condiviso con gli umani, tuttavia in seguito è stata osservata l'importanza e l'utilità di tale meccanismo anche in ambienti non strettamente connessi all'uomo, in cui è importante variare la rigidezza dell'attuatore durante l'esecuzione di un determinato compito [2].

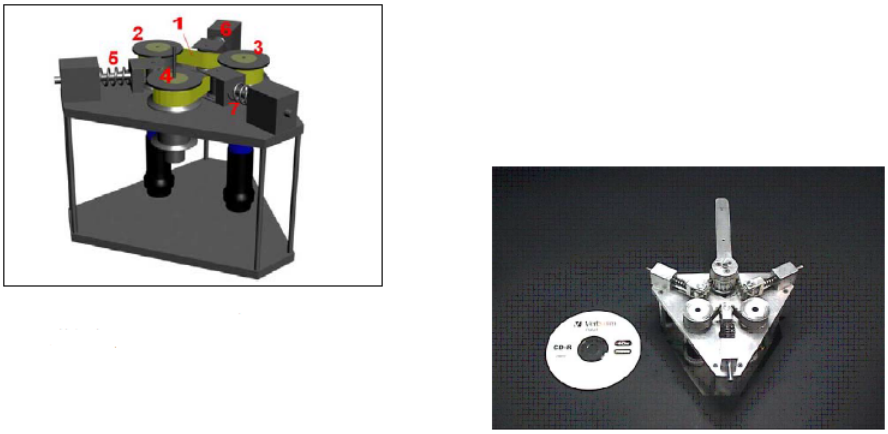


Figura 4.6.: *"Attuatore a rigidezza variabile"*

Nella figura 4.6, si può osservare il prototipo dell'attuatore a rigidezza variabile; in particolare nell'immagine sulla sinistra si ha una visione prospettica dell'attuatore, in cui la cinghia di trasmissione 1 collega le pulegge dei motori 2-3 all'albero del giunto 4, ed il tutto è messo in tensione dagli elementi elastici passivi 5-6-7. Variazioni angolari, δq_1 e δq_2, nello stesso verso dei motori 2 e 3 generano una variazione angolare nell'albero principale, δq_m, mentre, una variazione angolare dei due motori in direzioni opposte genera una variazione della rigidezza meccanica, $\delta\sigma$.
La principale differenza tra questo meccanismo e gli altri sviluppati è che questo si presta maggiormente ad una implementazione più compatta, permettendo così, una più rapida e continua variazione della rigidezza durante tutto lo svolgimento dell'attività. Per calcolare la rigidezza meccanica σ dell'attuatore VSA, si procede calcolando la coppia meccanica τ, generata dalla molla con costante elastica K, all'albero motore e la sua derivata rispetto alla variazione angolare che risulta essere proprio la rigidezza meccanica:

$$\sigma = -\frac{\delta\tau}{\delta q_m} \tag{4.1}$$

In figura 4.7 si riporta il meccanismo con cui si realizza la variazione di stiffness nel dispositivo:

Figura 4.7.: *Variazione della rigidezza*

4.3.2.2 VSA-II

VSA-II si presenta sempre come un attuatore a rigidezza variabile che ha lo scopo di superare le limitazioni riscontrate nel caso del VSA-I e dovute principalmente alle limitazioni della capacità di carico e di implementazione in un braccio robotico [3].

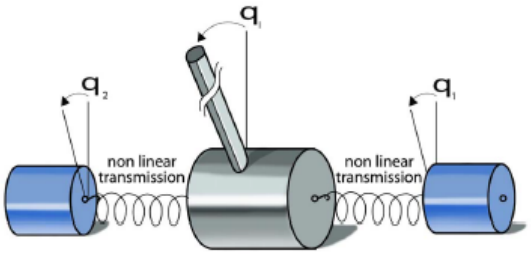


Figura 4.8.: *"Schema VSA-II"*

Il nuovo prototipo dell'attuatore a rigidezza variabile, come mostrato in figura 4.8, è pensato per riuscire a supportare una maggiore capacità di carico rispetto al precedente prototipo e per essere più compatto. Il sistema di trasmissione del VSA-II è basato su un meccanismo 4-bar come quello mostrato in figura 4.9:

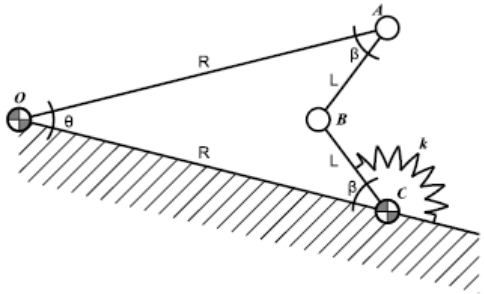


Figura 4.9.: *"Meccanismo 4-bar"*

costituito da due link più lunghi, di lunghezza pari a R e due link più corti, di lunghezza L e dove k rappresenta la costante elastica della molla e β è l'angolo di trasmissione. Lo scopo di questo sistema di trasmissione è quello di ottenere una caratteristica coppia/spostamento non lineare tra il carico applicato in ingresso dai motori e lo spostamento angolare dell'albero motore; quindi, l'attuatore VSA-II non è altro che un attuatore antagonista che ha due motori in opposizione.
Indicando con q_1 e q_2 gli angoli dei due motori, con q_l la posizione angolare del link e definendo $\theta_{i,j}$=q_i-q_j, si ha che la coppia di carico è:

$$\tau_l = 2M(\theta_{1,l}) + 2M(\theta_{2,l}) = 2M_{1,l} + 2M_{2,l}, \tag{4.2}$$

dove $M(\theta)$ è la derivata dell'energia potenziale immagazzinata nella molla

$$P = \frac{1}{2}k\beta^2 \tag{4.3}$$

$$M(\theta) = \frac{\delta P}{\delta \theta}. \tag{4.4}$$

L'espressione della rigidezza σ risulta essere anche in questo caso, espressa come la derivata della coppia di carico rispetto alla posizione angolare del link:

$$\sigma = \frac{\delta \tau_l}{\delta q_l} = 2\sigma_{1,l} + 2\sigma_{2,l}. \tag{4.5}$$

Confrontando i due attuatori si può osservare che l'attuatore VSA-I ha un accoppiamento incrociato tra i motori, mentre l'attuatore VSA-II implementa un meccanismo antagonista, che risulta meno complesso nella ottimizzazione dei parametri. Inoltre, VSA-I ha bisogno di una differenza tra gli angoli dei due attuatori per ottenere rigidità minima, mentre nel caso della VSA-II si ha rigidità minima quando la differenza tra i due angoli è nulla. VSA-I e II hanno diversi range di rigidità. Se i dispositivi sono azionati da motori con coppie limitate, la rigidità ottenibile viene limitata in quanto, condizioni di rigidità infinita richiederebbero anche valori di coppia infinita.
I motori devono, inoltre, bilanciare la coppia di disturbo esterna che agisce sul link, τ_{load}, riducendo in questo modo, la coppia disponibile per il controllo della rigidità. All'aumentare del valore della coppia di disturbo diminuisce in entrambi i casi il range dei valori ammissibili per la rigidezza, anche se nel caso della VSA-I all'aumentare della coppia esterna aumenta il massimo valore ammissibile della rigidezza, nel caso della VSA-II tale valore diminuisce.
Nel caso degli attuatori VSA-II, la rigidezza massima è limitata dal valore della coppia di stallo (o coppia di avviamento) del motore, mentre nel caso dell'attuatore VSA-I dipende anche dal carico applicato.
Tali considerazioni sono state effettuante valutando le performance dei due attuatori considerando il valor medio del range della rigidezza,σ_m e del valore dell'ampiezza relativa,Δ_σ:

$$\Delta_\sigma = \frac{(\sigma_{MAX} - \sigma_{MIN})}{2\sigma_m} \tag{4.6}$$

all'aumentare del valore di Δ_σ, si osserva un aumento delle prestazioni nel caso degli attuatori VSA-II.

4.3.3 VSA-CubeBot

La più diretta implementazione degli attuatori a cedevolezza variabile si riscontra nei dispositivi chiamati "VSA-Cube" [4], come mostrato in figura 4.10.

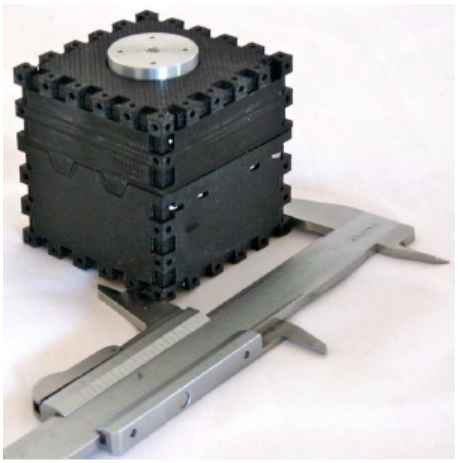


Figura 4.10.: *"Attuatore a cedevolezza variabile: VSA-Cube"*

Le principali caratteristiche di progettazione di tale dispositivo, risultano essere:

- l'elevata modularità;
- le piccole dimensioni;
- il basso costo.

In realtà questa singola unità è pensata per far parte di un più ampio kit robotico. L'idea di base dietro alla progettazione di tale dispositivo è quella di presentare un sistema che è simile ad un servomotore, in modo che il montaggio di un sistema robotico ci permetta di poter prendere in considerazione le prestazioni e le capacità dell'intero sistema, come ad esempio: l'energia immagazzinata, il range di cedevolezza, il tempo di assestamento della cedevolezza e il range dei valori cedevolezza/coppia.
Come già detto il VSA-Cube è un attuatore a cedevolezza variabile, che risulta essere composto da un motore (sono due servomotori), un riduttore, un sensore di posizione, una scheda elettronica e da alcuni algoritmi per controllare la posizione dell'albero di uscita; quindi, il VSA-Cube incorpora le caratteristiche di un servomotore con la possibilità di aggiustare la rigidezza dell'albero di uscita.
Questo dispositivo può essere visto come un sistema con una trasmissione non lineare che trasforma la coppia di ingresso e la velocità del motore primo in un insieme di quattro nuove variabili relative all'albero d'uscita come la coppia, la velocità, la cedevolezza e la variazione di cedevolezza. La composizione dell'attuatore VSA è riportata di seguito nella figura 4.11:

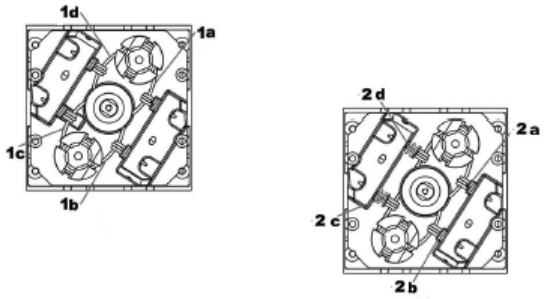


Figura 4.11.: *"Composizione interna di un attuatore VSA-Cube"*

Dalla figura si può osservare che la trasmissione elastica è realizzata attraverso quattro tendini (1a,1b,1c,1d) e quattro molle (1a,2b,2c,2d). Il meccanismo a molla non lineare è realizzato mediante un meccanismo antagonista.
In una configurazione dell'attuatore corrispondente ad un basso valore di cedevolezza, la molla e i tendini non sono carichi, in questo caso i due servomotori e, quindi, le pulegge ad essi connessi ruotano nello stesso verso. Invece, quando le due pulegge ruotano in direzioni opposte, due dei quattro tendini risultano sottoposti ad una forza, allungando così le molle collegate e realizzando, in questo caso, una configurazione ad elevata cedevolezza. Il movimento dell'albero di uscita si ottiene muovendo i due motori nello stesso verso.
Da un punto di vista elettrico il sistema è attuato da due servomotori e la posizione dell'albero motore è letta da un potenziometro; ogni unità è controllata da un microcontrollore che interpreta i dati letti dal potenziometro, controlla i motori e si occupa di gestire la comunicazione con l'ambiente esterno. L'interfaccia elettrica di ogni attuatore VSA è costituita da un bus a cinque fili, di cui uno va messo a massa, uno è per l'alimentazione del motore, uno è per l'alimentazione della logica e due implementano un bus I^2C.
Inoltre più attuatori VSA possono essere connessi in serie tramite questo bus, creando una catena di attuatori; in questo caso, ogni attuatore ha un suo indirizzo nel bus per cui un'operazione di scrittura verso ogni unità si traduce nell'invio all'unità corrispondente di un determinato comando, mentre, un'operazione di lettura si traduce nella ricezione da parte dell'unità desiderata di informazioni sulla configurazione interna dell'attuatore.
Sono possibili diverse modalità di controllo dell'attuatore che coinvolgono la servounità:

- la prima tecnica regola la posizione e la cedevolezza dell'albero motore;
- altre tecniche, invece, permettono di regolare la posizione dell'albero di uscita (controllo in anello chiuso come se l'attuatore non fosse un attuatore a cedevolezza variabile);
- infine, si hanno delle tecniche che permettono di regolare indipendentemente la posizione dei due servo motori; tale strategia di controllo è resa disponibile in modo da dare all'utente la possibilità di sviluppare altre strategie di controllo a più basso livello.

Un semplice modello del meccanismo a cedevolezza variabile può essere rappresentato come in figura 4.12:

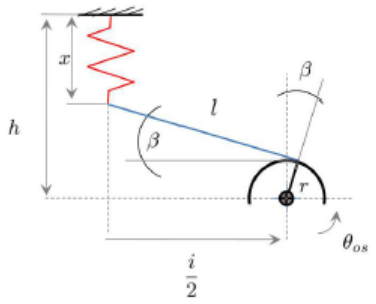


Figura 4.12.: *"Schema del meccanismo a cedevolezza variabile di un attuatore VSA-Cube"*

Come si può vedere dall'immagine, la funzione $\theta_{os} = f(x)$ descrive come viene caricata la molla quando l'albero o la puleggia ruota. Questa è ottenuta risolvendo la cinematica del meccanismo, descritta da:

$$\left\{ \begin{array}{c} \frac{i}{2} = l\cos(\beta) - r\sin(\beta) \\ h = x + l\sin(\beta) + r\cos(\beta) \\ (\theta_{os} - \theta_{m,i})r = 2(l + r(\frac{\pi}{2} - \beta)) - (i + r\pi) \end{array} \right\} \tag{4.7}$$

dove $\theta_{m,i}$ rappresenta la posizione angolare della puleggia. L'albero di uscita è posizionato simmetricamente rispetto all'asse delle molle e la sua cedevolezza può essere espressa come:

$$\sigma_{os} = \sum_{i=1}^{N_s} \frac{\delta^2 U_i}{\delta \theta_{os}^2} = \sum_{i=1}^{N_s} k_i \left(x_i \frac{\delta^2 x_i}{\delta {\theta_{os}}^2} + \left(\frac{\delta x_i}{\delta} \theta_{os} \right)^2 \right) \tag{4.8}$$

dove $x_i = f^{-1}(\theta_{os})$. La coppia esercitata sull'albero di uscita τ_{os} risulta essere la derivata prima dell'energia elastica rispetto alla variazione della posizione angolare dell'albero motore:

$$\tau_{os} = \sum_{i=1}^{N_s} k_i x_i \frac{\delta x_i}{\delta \theta_{os}}. \tag{4.9}$$

Capitolo 5

Realizzazione del software

Come già accennato, lo schema del modello del sistema da controllare è stato realizzato su ScicosLab, "tools" che, con il supporto di "E4Coder", oltre alla semplice modellazione con uno schema a blocchi, del sistema da controllare, utile per effettuare delle simulazioni, permette, anche di poter generare il codice del modello progettato per diversi sistemi operativi e per diverse "boards" a seconda delle esigenze.
Per realizzare il modello del sistema su ScicosLab, sia per la parte di simulazione che per la parte di generazione del codice è stato necessario implementare dei blocchetti custom da aggiungere a quelli già messi a disposizione dal "tools". In particolare tali blocchetti sono stati realizzati sia per modellare la comunicazione della scheda con i vari cubotti, sia per modellare il comportamento dei vari cubotti e la loro interfaccia con l'utente esterno.
Oltre a ciò si è reso necessario effettuare il "porting" della scheda su E4Coder, in modo da poter consentire la generazione di codice per tale board. Con il termine "porting" si indica un processo di trasposizione di un componente software, a volte con modifiche, in modo da consentirne l'utilizzo in un ambiente diverso da quello in cui è stato creato. Tale operazione può essere richiesta a causa delle differenze tra le CPU, dalle diverse interfacce dei sistemi operativi, dalla diversità dell'hardware o a causa di incompatibilità nel linguaggio di programmazione sul target considerato, ovvero sull'ambiente su cui dovrà essere compilato il programma generato.

5.1 Realizzazione del porting per la scheda

La realizzazione del supporto per un nuovo target su E4Coder è abbastanza semplice e non coinvolge nessun aspetto della programmazione in ScicosLab. Ogni scheda è definita da un target boards che viene immagazzinata in una directory separata all'interno della cartella "E4Coder-Customizations", come mostrato in figura 5.1. Tale directory contiene un file chiamato "board.txt" che descrive le varie proprietà della board. A questo punto ogni board ha un sottoinsieme di sottodirectory che contengono altri file tra cui il file "target.txt" che contiene le specifiche del codice custom generato per la board per uno specifico sistema operativo supportato dal generatore di codice (Linux, Linux RTAI, BareMetal).

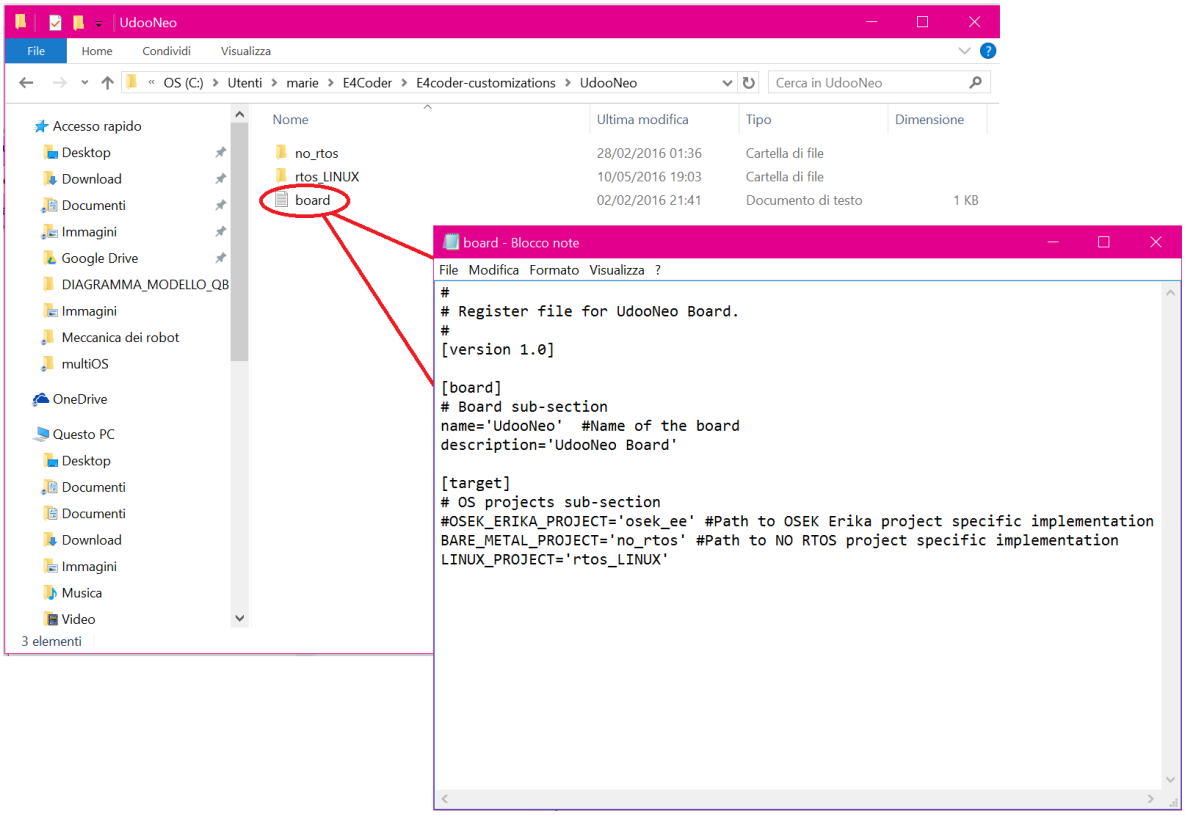


Figura 5.1.: *"Immagine directory target-Board e struttura del file 'board.txt'"*

Ogni directory contenente un file "board.txt" contenuta nella cartella E4coder-Customizations viene interpretata come una nuova board supportata dal generatore di codice; tale file, che viene interpretato nella fase iniziale, ogni volta che viene generato del codice, è suddiviso in sezioni tra cui:

- la sezione "version", in cui si esprime la versione della boards;
- la sezione "board" che contiene alcune informazioni sulla board, come ad esempio il nome, o la directory che contiene palette aggiuntive per tale board (quest'ultimo attributo risulta essere facoltativo);
- la sezione "target", sezione che contiene l'insieme di attributi che sono collegati al sistema operativo per cui vogliamo generare il codice; se una board supporta l'implementazione per uno specifico sistema operativo, allora il corrispondente attributo sarà specificato in questa sezione. Il valore di questi attributi rappresenta il nome di una directory contenente le specifiche informazioni che permettono al generatore di codice di generare i file dipendenti dal target; ognuna di queste directory conterrà un file target.txt. In particolare gli attributi supportati dalla sezione target sono:
  - LINUX_PROJECT, utilizzato per specificare il supporto per sistemi operativi GNU/Linux, dotati di librerie pthread e caratteristiche "soft-realtime";
  - OSEK_ERIKA_PROJECT, che specifica il supporto per il sistema operativo tempo reale Erika OSEK/VDX;
  - WINDOWS_PROJECT, utilizzato per specificare il supporto per il sistema operativo Windows;

  - LINUX_RTAI_PROJECT, utilizzato per specificare il supporto per il sistema operativo RTAI-Linux;
  - BARE_METAL_PROJECT, utilizzato per specificare il supporto per sistemi Bare Metal, ovvero per ambienti privi di sistemi operativi.

Come accennato precedentemente, nel momento in cui si aggiunge una board custom o un blocco custom è spesso utile definire una "palette" custom che contiene un'istanza del blocco che può essere utilizzata in un particolare contesto; ma questo è del tutto arbitrario.
Nel file "target.txt", mostrato in figura 5.2 e contenuto all'interno di ogni directory che specifica il target supportato dal generatore di codice per quella determinata board, si trova la lista delle varie periferiche supportate per ogni target, così come il codice aggiuntivo e le librerie che è necessario aggiungere a un progetto per poterlo compilare. La struttura del file "target.txt" risulta essere composta dalle seguenti sezioni:

- la sezione "version" che anche in questo caso è utilizzata per associare una versione al file "target.txt";
- la sezione "settings" che a sua volta contiene i seguenti attributi:
  - "librarycode" è il nome della directory che contiene il codice aggiuntivo e i file che devono essere aggiunti al codice generato quando si utilizza questo tipo di board; questo è un parametro opzionale e principalmente viene utilizzato per specificare quali librerie devono essere aggiunte;
  - "targettick" è l'attributo utilizzato per specificare in secondi il periodo minimo dell'interruzione di default della board; questo periodo di interruzione di default è utilizzato in alcuni sistemi operativi, come OSEK/VDX, per scalare tutti i periodi sulla base del "tick" di default. Tale valore deve comunque essere compatibile con le frequenze di lavoro della board;
  - "targetinclude" è un attributo utilizzato per specificare file aggiuntivi che devono essere aggiunti all'interno del file "target_name.h";
  - "targetconfigs" è un attributo utilizzato per specificare file C che dovranno essere aggiunti nel file "target.c";

  in particolare, questi ultimi attributi sono utilizzati per specificare un insieme di funzioni hooks e un insieme di altre funzioni che sono utilizzate dal generatore di codice per creare l'interfaccia tra il target e il codice generato, tra cui le funzioni per la gestione delle interruzioni.
- la sezione "build" è quella sezione in cui vengono specificati i comandi aggiunti, solitamente alcuni file, che devono essere eseguiti prima del processo di generazione di codice o dopo il processo di generazione di codice e compilazione; questo perchè solitamente il processo di generazione di codice è solo una parte del processo di compilazione, programmazione ed esecuzione di codice in ambiente embedded. Ad esempio, in alcuni casi potrebbe essere necessario fornire dopo la generazione di codice, ma prima della compilazione, insieme al codice generato un insieme di altri file necessari per l'ambiente di compilazione che sono contenuti in un altro repository. Tuttavia l'opzione di poter aggiungere dei file dopo la fase di generazione di codice non può essere utilizzata nel caso in cui il codice venga generato per un ambiente "Bare Metal", ovvero per un ambiente privo di sistema operativo o se la compilazione dopo la generazione di codice non è stata ben specificata nelle opzioni di compilazione per il target considerato. Per

ogni sistema operativo per cui è supportata la generazione del codice sotto la sezione "build" attraverso gli attributi "pre" e "post" è possibile aggiungere l'insieme dei path dei file necessari per la compilazione;

- la sezione "blocktype" è utilizzata per definire le varie periferiche che possono essere utilizzate per una specifica board: ADC, DAC, PWM, LED, ecc. Oltre a questo in tale sezione, per ogni periferica è possibile specificare altre informazioni attraverso gli attributi:
  - "librarycode" dove viene specificata la directory che contiene il codice aggiuntivo e i file che devono essere aggiunti per generare il codice quando viene utilizzata quella determinata periferica; se anche altre periferiche fanno riferimento alla stessa directory e tali periferiche vengono utilizzate contemporaneamente nel codice generato, allora tali file, verranno aggiunti nel codice generato solo una volta;
  - "targetcode" dove viene indicato il nome di un file che contiene le specifiche del codice che sarà generato per questo tipo di blocco e che si trova nel file "target.c";
  - "peripheral" attributo che può essere ripetuto più volte e che è utilizzato per elencare le diverse istanze della periferica per una data board;
- infine, si ha la sezione "oil" utilizzata quando viene generato codice per il sistema operativo OSEK/VDX Erika Enterprise, che permette di specificare il file di configurazione OIL contenente informazioni sulla configurazione del sistema operativo.

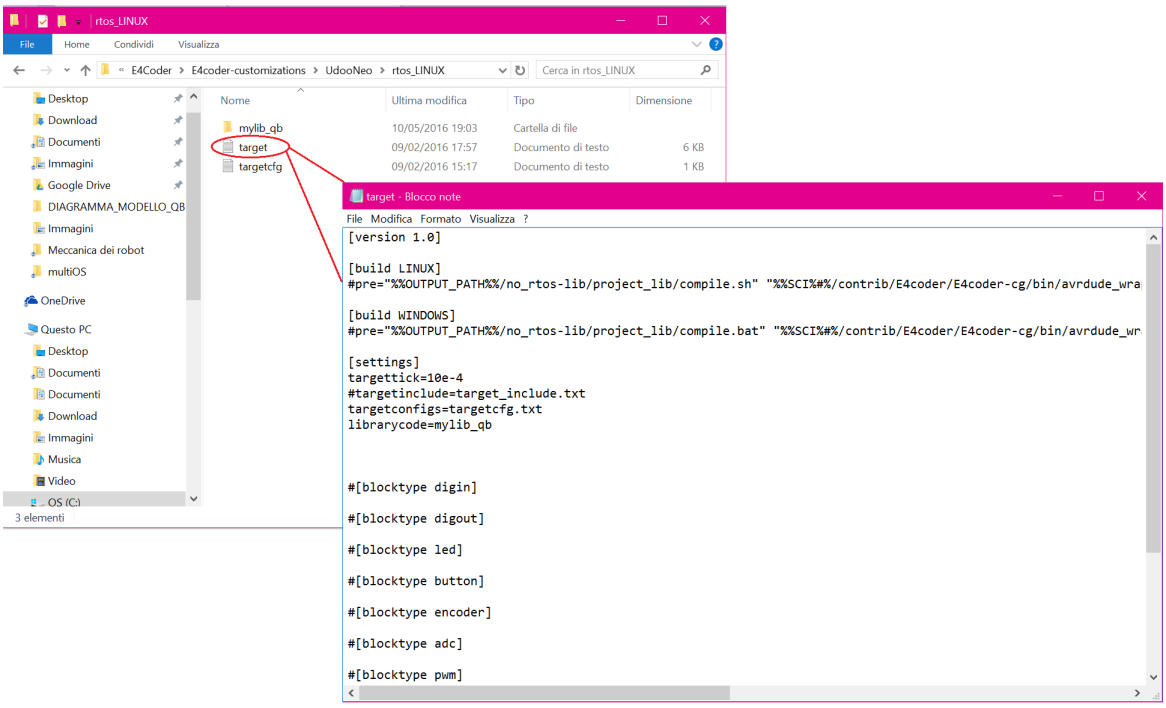


Figura 5.2.: *"Immagine dei target supportati per la board e struttura del file 'target.txt' per il target LINUX_PROJECT"*

5.2 Realizzazione dei blocchetti custom

Il vantaggio dei blocchi custom consiste nel fatto che con questi blocchi è possibile modellare un generico comportamento del sistema in base alle proprie esigenze, specificando

semplicemente 4 funzioni scritte in linguaggio C.
Il nome del blocco è anche utilizzato per indicare il file header che contiene la dichiarazione; inoltre, diversi blocchi "custom" possono condividere lo stesso nome e quindi, avere lo stesso comportamento generale. Le altre proprietà del blocco permettono di specificare altri parametri come il numero e il tipo di ingressi e uscite del blocco, eventuali parametri del blocco e stati interni, proprietà che contribuiscono a delineare gli aspetti e le funzionalità del blocco custom. Questo permette di poter ottenere diversi blocchi custom che hanno lo stesso nome ma differiscono tra di loro solo per i valori dei parametri.
Tali blocchi possono essere usati sia in fase di simulazione che in fase di generazione di codice, per questo motivo è necessario fornire un'implementazione del blocco per la fase di simulazione e una per quella di generazione.
Il supporto di nuovi blocchi custom per la generazione di codice richiede la creazione di una directory separata per ogni blocco, che verrà aggiunta nella cartella "custom-blocks" situata all'interno della directory E4Coder presente in Scicos:

directory_install_Scicos/contrib/E4Coder/E4Coder−cg/codegen/custom−blocks.

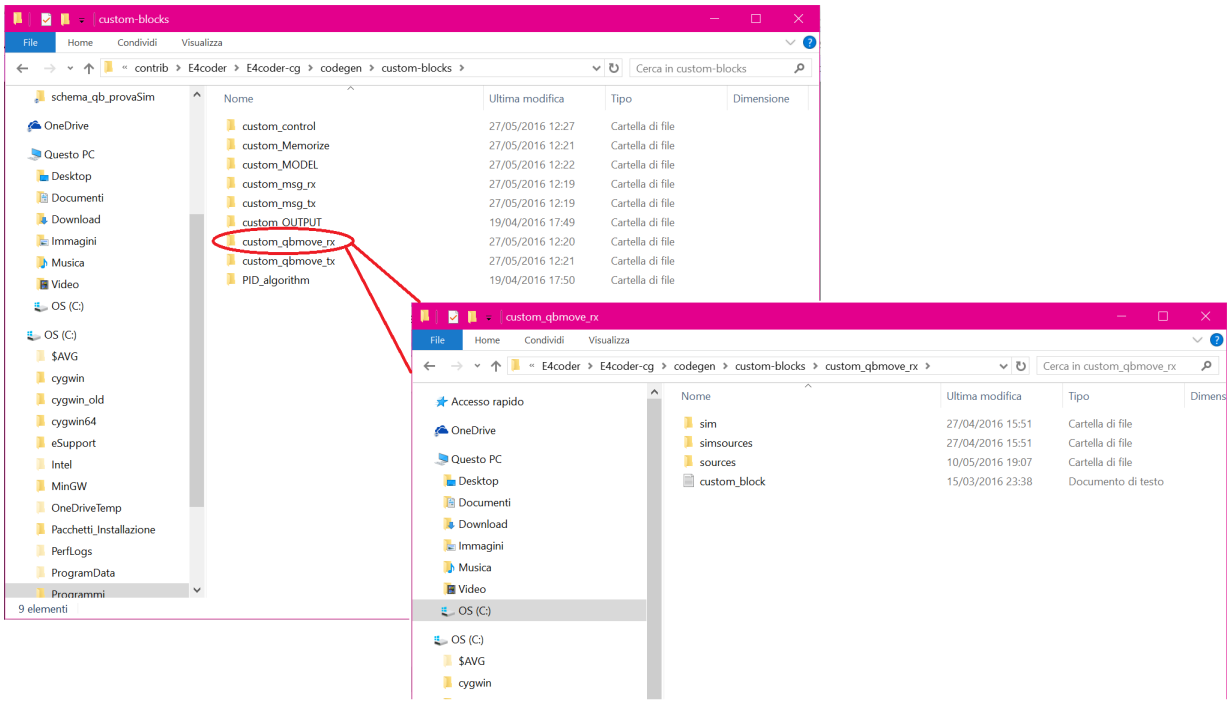


Figura 5.3.: *"Struttura della directory dei blocchi custom"*

In particolare, i blocchi aggiunti sono cinque:

- un blocco custom msg_rx che si occupa di gestire la ricezione dei messaggi dal cubotto;
- un blocco custom qbmove_rx che elabora i dati ricevuti;

- un blocco qbmove_tx che ci permette di selezionare alcune informazioni sul nuovo comando da spedire al cubotto;
- un blocco custom msg_tx che gestisce l'invio dei dati e dei comandi al cubotto;
- un blocco custom control nel quale si implementa l'algoritmo di controllo del sistema.

Come già accennato e come si può osservare anche dalla figura 5.3 nella directory di ogni blocco è stato necessario implementare la funzionalità del bloccho per la parte di generazione di codice, che si trova nella cartella sources, la funzionalità del blocco per la parte di simulazione, che si trova nella cartella simsources, e generare la libreria dll che verrà richiamata "run-time" in fase di simulazione.

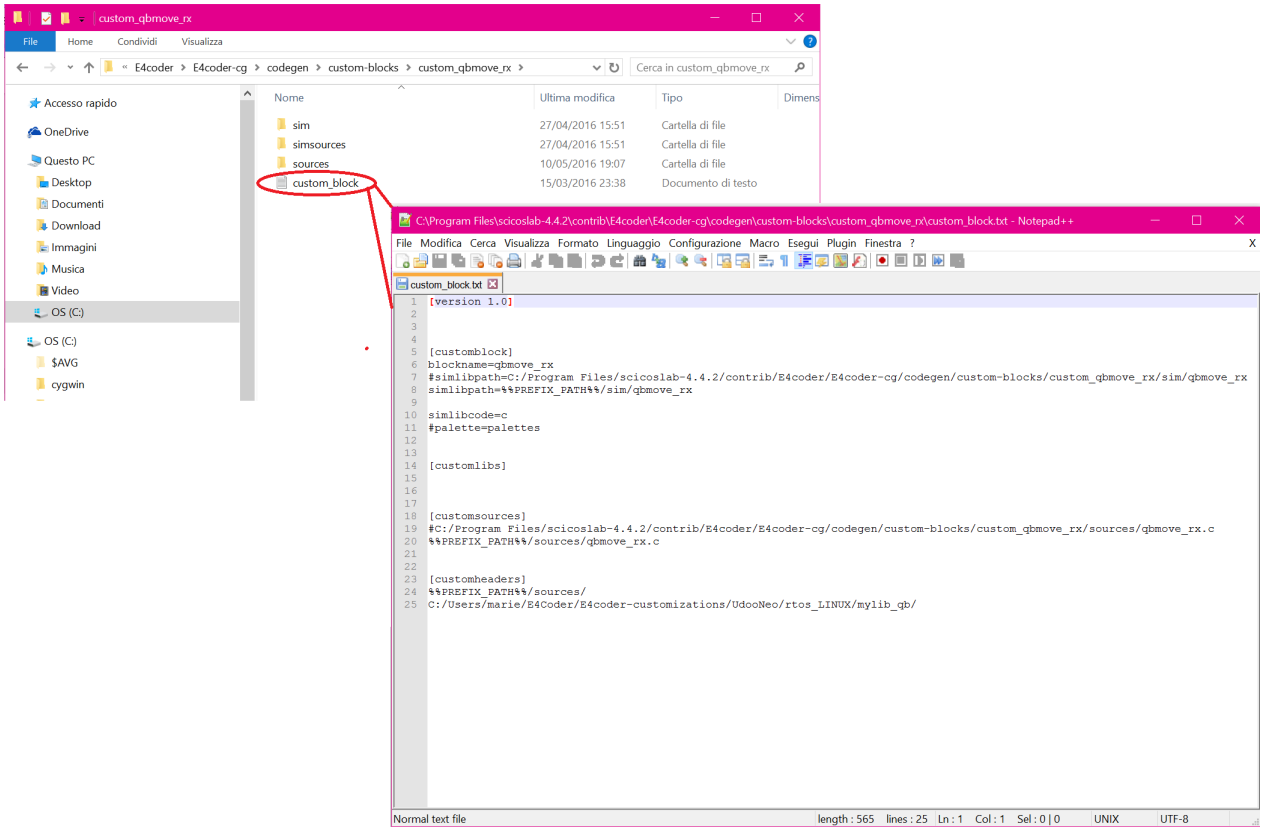


Figura 5.4.: *"Struttura del file custom_block"*

La realizzazione di ogni blocco è poi completata da un file ".txt", riportato in figura 5.4, sempre presente all'interno della directory, in cui vengono specificati i path in cui si trovano gli elementi descritti sopra; in particolare la struttura di tali file sarà costituita da:

- una sezione "version", in cui sarà indicata la versione corrente del blocco;
- una sezione "custom_block", in cui saranno indicati alcuni attributi tra cui "blockname", ovvero il nome del blocco custom, "simlibpath", ovvero il path della directory in cui si trova la libreria da utilizzare in fase di simulazione, che implementa il comportamento del blocco, e che sarà caricata all'avvio di E4Coder; in questo caso è importante che il nome della libreria sia specificato senza l'estensione; inoltre, oltre a questi, altri attributi sono "simlibcode" che indica il tipo di codice utilizzato per scrivere il codice

sorgente e l'attributo "palette" che indica se tale blocco è presente tra le palette del tools;

- una sezione "customlibs", in cui vengono specificate le librerie aggiuntive che devono essere caricate in aggiunta a quelle specificate nella sezione precedente;

- una sezione "customheaders", che contiene i path dei file headers che devono essere aggiunti;

- infine, la sezione "oil" che deve essere descritta per la generazione o la simulazione dei blocchi per il sistema operativo Erika.

Vediamo ora come è stato realizzato lo schema complessivo del sistema.

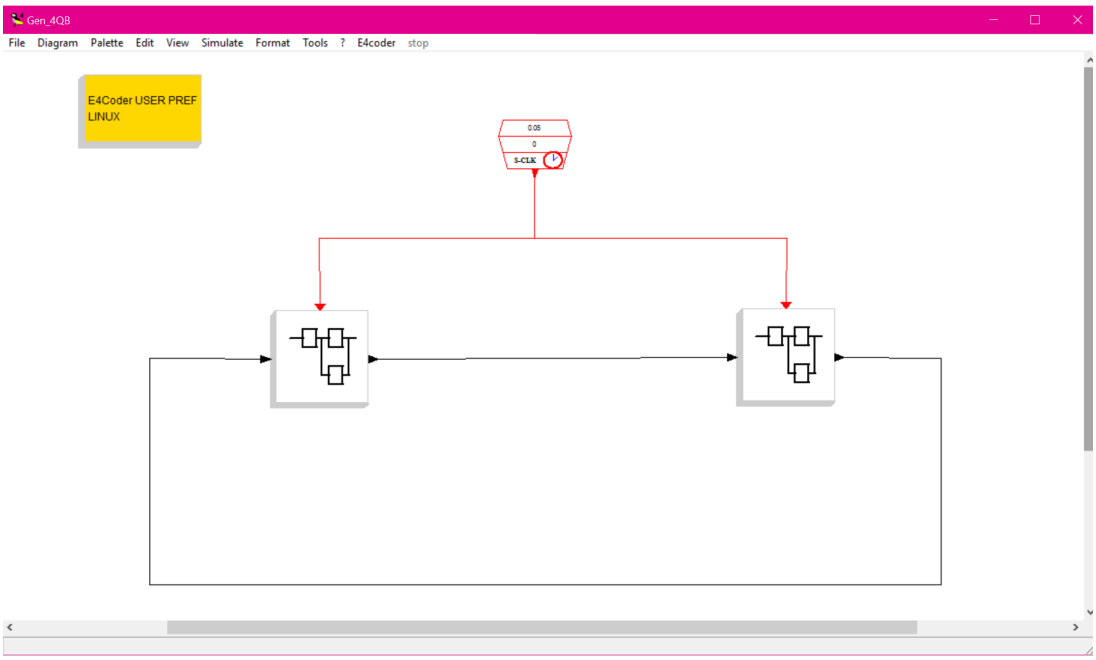


Figura 5.5.: *"Struttura del diagramma complessivo"*

Come si può osservare dall'immagine riportata in figura 5.5, lo schema del sistema complessivo è stato realizzato separando in due sottoblocchi la parte dello schema necessaria alla sola generazione di codice che comprende i blocchi relativi alla modellazione del comportamento dei cubotti, al controllo e alla comunicazione e la parte dello schema che è necessario aggiungere per realizzare la simulazione, comprendente il blocco in cui viene implementato il modello dinamico del sistema che, invece, non è necessario per la generazione di codice, in quanto, il sistema interagisce direttamente con il dispositivo.
Il primo sottoblocco, ovvero quello relativo alla generazione di codice, è costituito dai blocchi che implementano il funzionamento dei cubotti, la trasmissione dei messaggi e l'algoritmo di controllo.

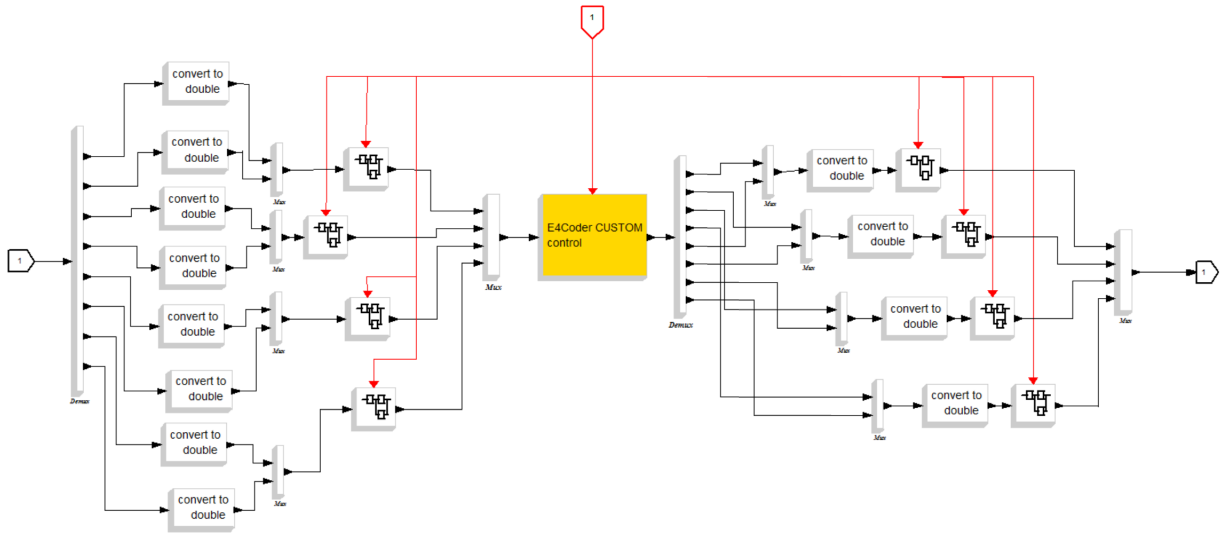


Figura 5.6.: *"Sottoblocco del sistema per la parte di controllo di cui viene generato il codice"*

Dalla 5.6 si può osservare che il primo sottoblocco è costituito dal blocco del controllo e da ulteriori sottoblocchi, che per ogni riga rappresentano il funzionamento di un cubotto e la sua comunicazione con la board utilizzata. In particolare, poichè ad ogni cubo non solo è possibile fornire informazioni ma anche richiederne e quindi, è dotato sia di sensori che di attuatori, nello schema, sulla sinistra si trovano i sottoblocchi relativi ai sensori, nei quali si trovano i blocchi custom msg_rx e qbmove_rx, e sulla destra i sottoblocchi relativi agli attuatori, in cui si trovano i blocchi custom qbmove_tx e msg_tx.
Nelle figure 5.7 e 5.8 si può osservare meglio come sono realizzati rispettivamente i sottoblocchi per la ricezione e per la trasmissione:

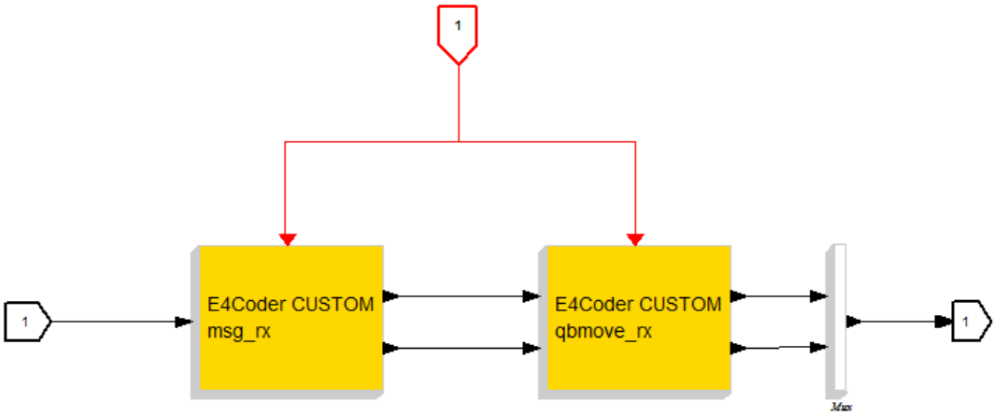


Figura 5.7.: *"Sottoblocco per la ricezione"*

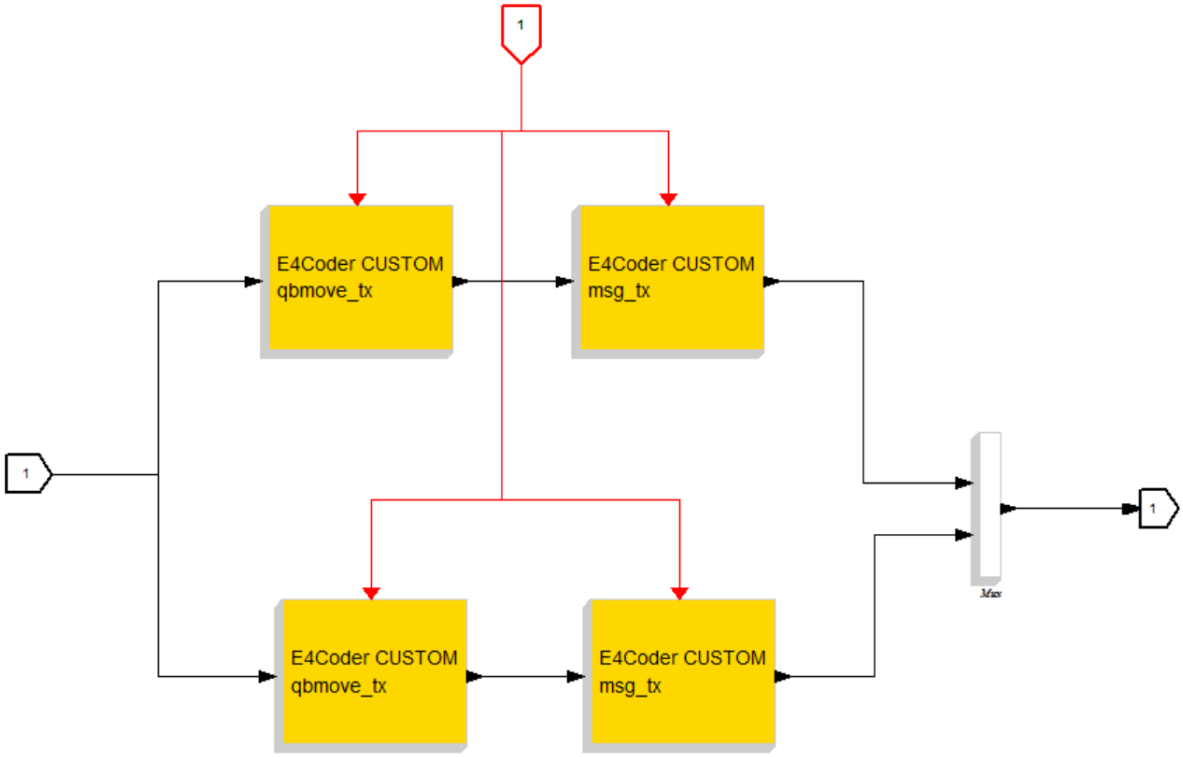


Figura 5.8.: *"Sottoblocco per la trasmissione"*

L'idea è quindi, quella di utilizzare una coppia di blocchi msg/qbmove per ogni comando che vogliamo inviare al dispositivo e il tipo di comando viene specificato mettendo il numero, corrispondente al comando desiderato, nei parametri del blocco; in questo modo, con una coppia di tali blocchi è possibile specificare qualsiasi funzione di comunicazione disponibile per il cubotto. In particolare, si ha che ogni funzione che ha il solo compito di settare nuovi parametri del cubotto senza ricevere una risposta da parte sua, viene implementata solo con i blocchi custom qbmove_tx e msg_tx, mentre le funzioni che chiedono informazioni al cubotto, come ad esempio informazioni sulla posizione, sulla velocità, sulla corrente, vengono implementate con i blocchi custom qbmove_tx e msg_tx ,che si occupano di inviare la richiesta e con i blocchi custom msg_rx e qbmove_rx che, invece, si occupano di recuperare la risposta ottenuta.
In questo caso, per ogni cubotto le funzioni utilizzate sono due, quella con cui si settano i nuovi valori della posizione dati dal controllo, ovvero la funzione "commSetInputs()" dalla quale non si ottiene una risposta da parte del cubotto e che quindi viene implementata solo tramite blocchi qbmove_tx e msg_tx e la funzione commGetMeasurements(), con la quale si richiedono al cubotto la posizione dell'albero centrale e dei due servomotori che costituiscono il cubotto e che viene implementata utilizzando tutti e quattro i blocchi custom. Quindi, è necessario che lungo ogni riga i due sottoblocchi per la ricezione e la trasmissione siano relativi allo stesso cubotto.
I valori in uscita dai quattro superblocchi di ricezione vanno in ingresso al blocco custom del controllo che in uscita restituisce i nuovi valori da comunicare al dispositivo.
Per disaccoppiare la parte di controllo da quella di comunicazione con i cubotti, la comunicazione viene gestita da due thread in background. Analizziamo quindi, il funzionamento dei vari blocchetti custom.

5.2.1 Blocchi custom per la comunicazione

Per la gestione della comunicazione tra il braccio e la scheda sono stati realizzati i due blocchi custom msg, di cui uno per la trasmissione e uno per la ricezione. I parametri di tali blocchi sono il BAUDRATE, l'ID, la posizione, POS, all'interno della catena e "Command", la codifica del comando che deve eseguire; la presenza dell'ID come parametro del blocco è necessaria perchè in fase di lettura dei dati ricevuti ogni elemento legge solo il pacchetto a lui destinato, identificato appunto dall'ID.
In realtà tali blocchi non implementano veramente la comunicazione seriale tra il cubotto e la scheda, ma semplicemente gestiscono delle code FIFO ("First-In First-Out"), in cui verranno messi i messaggi che dovranno essere spediti al cubotto o che vengono ricevuti dal cubotto. La comunicazione seriale è, invece, gestita da due thread in background, che rimangono sempre in ascolto sulla seriale. In particolare un thread è utilizzato per gestire la sola trasmissione dei dati, mentre l'altro gestisce sia la trasmissione che la ricezione dei dati; per questo motivo sono state create due code FIFO una in cui vengono messi i messaggi che dobbiamo inviare al cubotto e dai quali non riceviamo risposta e una dove vengono posti i messaggi da inviare e dai quali ci si aspetta una risposta.
Come già accennato per ogni blocco si è reso necessario implementare il comportamento sia, per la generazione di codice sia, per la simulazione.

5.2.1.1 Generazione di codice del blocco custom "msg_rx"

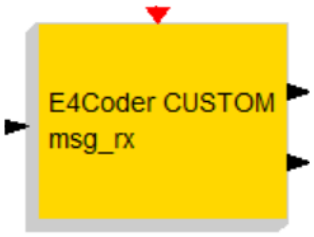


Figura 5.9.: *"Blocco custom msg_rx"*

Il blocco riportato in figura 5.9 è quello che gestisce la ricezione dei messaggi, in particolare si può notare che esso è caratterizzato da un ingresso e due uscite; l'ingresso viene utilizzato solo per la parte di simulazione, in quanto, nella parte di generazione di codice i messaggi vengono prelevati dalla struttura condivisa con il thread che gestisce la ricezione dei dati.

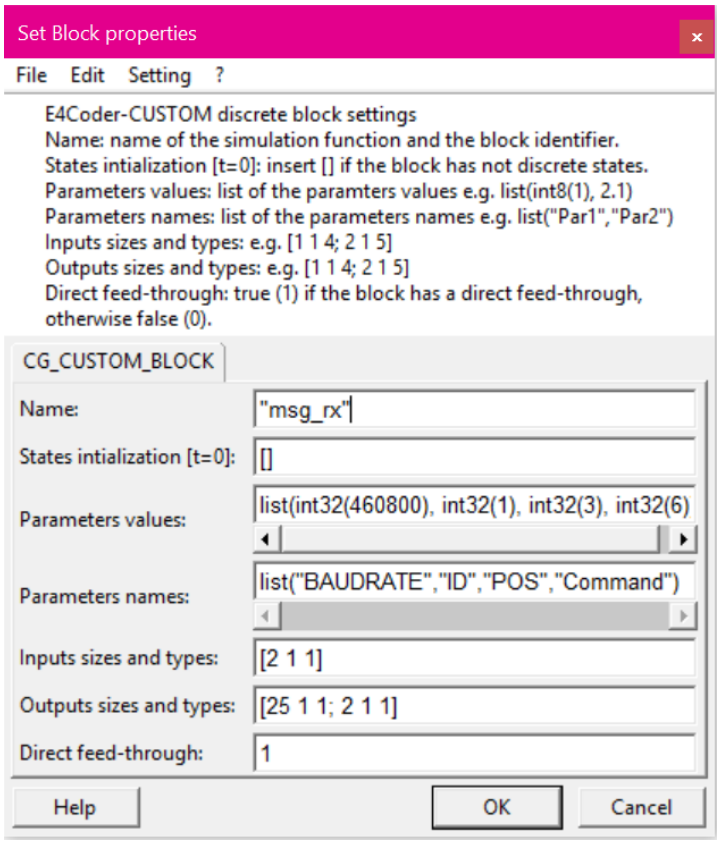


Figura 5.10.: *"Parametri del blocco custom msg_rx"*

I Parametri di tale blocco che si possono osservare nella figura 5.10, sono:

- BAUDRATE, parametro necessario per fissare la velocità di trasferimento dei dati sulla seriale;
- ID, parametro necessario per identificare il cubotto con il quale si vuole interagire; tale parametro è sempre presente nel protocollo di comunicazione, infatti se non venisse specificato l'ID verrebbe inviato lo stesso comando a tutti gli elementi che costituiscono la catena;
- POS, parametro necessario per indicare le posizione del cubotto nella catena, informazione necessaria per l'algoritmo di controllo;
- Command, paramentro con il quale viene specificato il comando o la richiesta che vogliamo inviare al cubotto.

Variando il valore dei parametri si può ottenere lo stesso funzionamento per diversi cubotti, ognuno identificato dal proprio ID.
Come già accennato precedentemente la costruzione di un blocco custom si ottiene mediante l'implementazione di quattro funzioni in C che, in questo caso, risultano essere:

void msg_rx_Init(msg_rx_Param *Param, real *In1, real *Out1, real *Out2);

void msg_rx_OutputUpdate(msg_rx_Param *Param, real *In1, real *Out1, real *Out2);

void msg_rx_StateUpdate(msg_rx_Param *Param, real *In1, real *Out1, real *Out2);

void msg_rx_Terminate(msg_rx_Param ∗Param, real ∗In1, real ∗Out1, real ∗Out2).

Con la funzione di inizializzazione, void msg_rx_Init(), si procede in primo luogo con l'apertura della porta seriale; se nel diagramma sono presenti più blocchi dello stesso tipo, come in questo caso, si ha che tale operazione viene eseguita solo dal primo blocco che si trova in sequenza. Inoltre, si procede all'inizializzazione di una struttura dati in cui verranno memorizzati, oltre ai valori letti, informazioni quali l'ID del destinatario, il comando relativo all'informazione servita e l'effettiva dimensione del pacchetto memorizzato; in questo modo, verrà effettuata la copia del dato ricevuto per lasciare libera la struttura dati condivisa in cui erano stati memorizzati dal thread gestore.
A questo punto, dopo aver inizializzato altre due strutture dati, si procede con la creazione del thread che si occupa della gestione della comunicazione ricevuta. La prima è una struttura in cui verranno memorizzati i pacchetti da inviare con i quali di chiedono informazioni al dispositivo; in questa coda troveremo quindi, solo pacchetti da inviare al dispositivo che richiedono in risposta misure effettuate da esso. L'altra, invece, è una struttura in cui vengono memorizzati in coda tutti i pacchetti ottenuti in risposta dal dispotivo, in particolare avremo una coda per ogni dispositivo. Quindi, il thread creato, chiamato "Gestore_Read" si occupperà di inviare al cubotto la richiesta e attendere la sua risposta con le informazioni richieste.
La funzione msg_rx_OutputUpdate(), invece, ha il compito di estrarre dalla struttura condivisa i nuovi dati ricevuti dal dispositivo; in uscita dal blocco si ottiene, infatti, il pacchetto ricevuto, sulla prima uscita, e la dimensione effettiva, sulla seconda.

5.2.1.2 Generazione di codice del blocco custom "msg_tx"

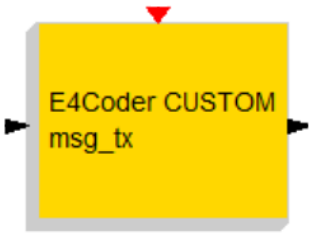


Figura 5.11.: *"Blocco custom msg_tx"*

Il blocco riportato in figura 5.11 gestisce la trasmissione dei messaggi, ed è caratterizzato da un ingresso e da un'uscita; l'uscita viene utilizzata solo per la parte di simulazione, in quanto nella parte di generazione di codice i messaggi vengono inseriti in una struttura condivisa con il thread che gestisce la trasmissione dei messaggi al dispositivo.

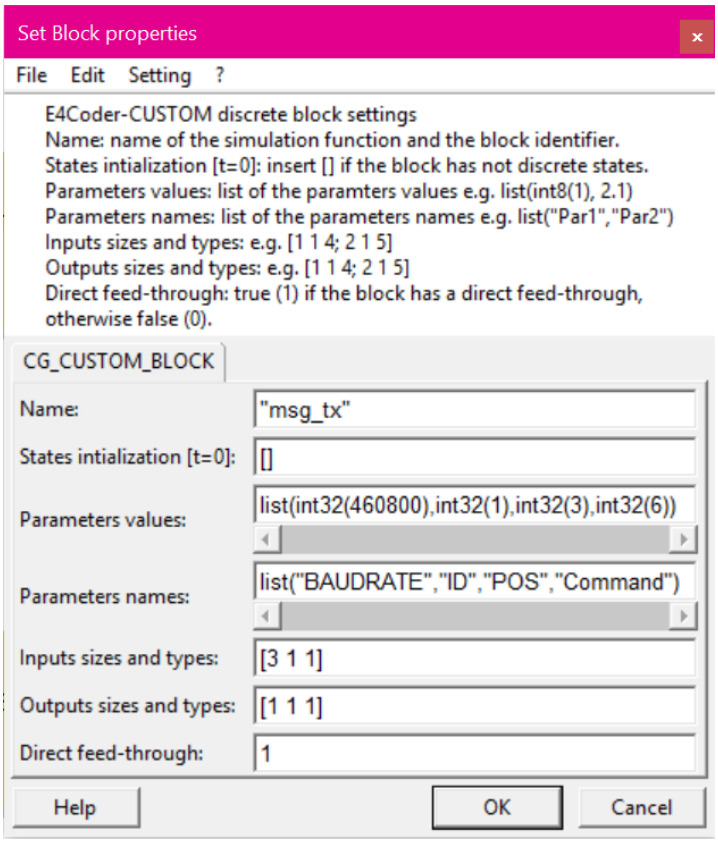


Figura 5.12.: *"Parametri del blocco custom msg_tx"*

I parametri del blocco che si possono osservare anche in figura **??**, sono gli stessi del blocco custom msg_rx, ovvero BAUDRATE, ID, POS e Command. Come accennato prima, variando il valore dei parametri è possibile ottenere lo stesso funzionamento per diversi dispositivi ognuno identificato dal proprio ID, ma anche ottenere diversi comandi per uno stesso ID come si verifica nel caso di ogni sottoblocco per la trasmissione in cui sono presenti due blocchi dello stesso tipo per lo stesso ID che però differiscono per il comando, con il primo si richiedono le misure di posizione e con l'altro si settano i nuovi ingressi dati dal controllo.
Anche in questo caso la costruzione di un blocco custom si ottiene mediante l'implementazione di quattro funzioni in C che risultano essere:

void msg_tx_Init(msg_tx_Param *Param, real *In1, real *Out1);

void msg_tx_OutputUpdate(msg_tx_Param *Param, real *In1, real *Out1);

void msg_tx_StateUpdate(msg_tx_Param *Param, real *In1, real *Out1);

void msg_tx_Terminate(msg_tx_Param *Param, real *In1, real *Out1).

Con la funzione msg_tx_Init() si procede in primo luogo ad attivare il dispositivo, aspettando la conferma dell'attivazione da parte dello stesso prima di procedere ulteriormente.
Attivato il dispositivo, si procede con la creazione del thread, chiamato Gestore_Write, che si occupa di inviare al dispositivo i pacchetti relativi a quei comandi che non richiedono una risposta da parte del dispositivo e che vengono inseriti in una struttura dati condivisa, sempre una coda FIFO, che contiene solo questi pacchetti. Anche in questo caso il thread

viene creato solo una volta.
La funzione msg_tx_OutputUpdate(), invece, ha il compito di selezionare il pacchetto da inviare relativo al comando corrispondente e di inserirlo nella rispettiva struttura dati condivisa a seconda che il comando preveda o meno una risposta da parte del dispositivo.

5.2.2 Blocchi custom per la rappresentazione del cubo VSA

Il funzionamento del cubo VSA è descritto attraverso i blocchi custom qbmove_rx e qbmove_tx. I parametri di questi blocchi sono l'ID, la posizione, POS, del cubo nella catena e il comando, Command, con il quale si identifica il comando (i valori di tali parametri sono uguali a quelli dei blocchi msg a cui sono legati).

5.2.2.1 Generazione di codice del blocco custom "qbmove_rx"

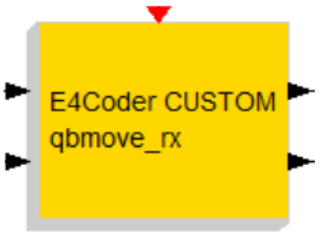


Figura 5.13.: *"Blocco custom qbmove_rx"*

Il blocco riportato in figura 5.13 si occupa di gestire i dati ricevuti in ingresso estrapolando da ogni pacchetto le informazioni richieste. Esso è caratterizzato da due ingressi che riceve dal blocco msg_rx e due uscite che costituiranno parte degli ingressi del blocco control.
I parametri del blocco si possono osservare nella figura 5.14:

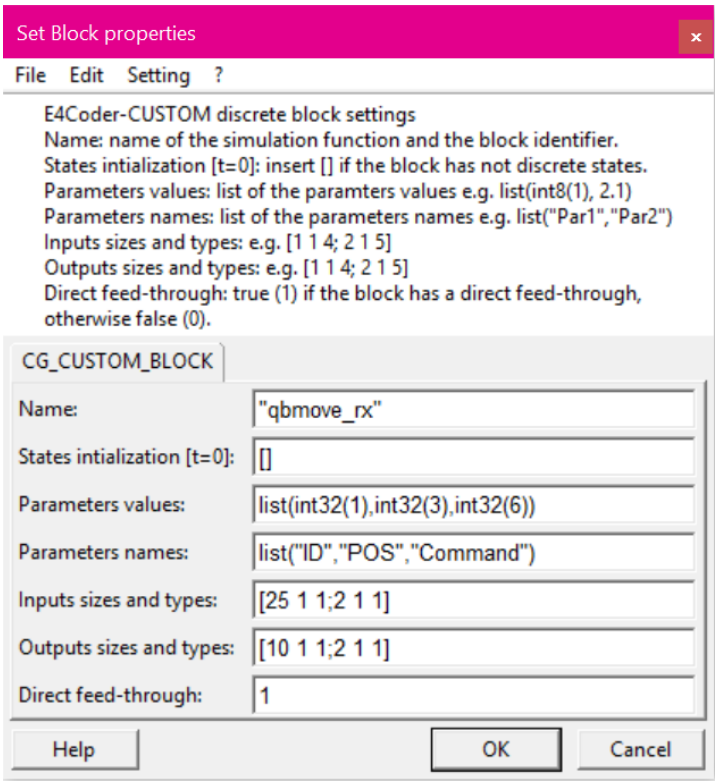


Figura 5.14.: *"Parametri del blocco custom qbmove_rx"*

Le funzioni che definiscono il comportamento del blocco sono:

void qbmove_rx_Init(qbmove_rx_Param ∗Param, real ∗In1, real ∗In2, real ∗Out1, real ∗Out2);

void qbmove_rx_OutputUpdate(qbmove_rx_Param ∗Param, real ∗In1, real ∗In2, real ∗Out1, real ∗Out2);

void qbmove_rx_StateUpdate(qbmove_rx_Param ∗Param, real ∗In1, real ∗In2, real ∗Out1, real ∗Out2);

void qbmove_rx_Terminate(qbmove_rx_Param ∗Param, real ∗In1, real ∗In2, real ∗Out1, real ∗Out2).

Come accennato sopra, il compito di questo blocco è quello di estrapolare dal pacchetto ricevuto le misure, in base al comando; questa operazione viene effettuata solo se la dimensione del pacchetto ricevuta in ingresso è diversa da zero. I valori estrapolati vengono poi rimandati sulla prima uscita, mentre sulla seconda uscita si trovano rispettivamente il numero delle misure ricevute, che varia in base ai sensori utilizzati dal dispositivo, che coincide con la dimensione effettiva del buffer in uscita, e la posizione del dispositivo nella catena, informazione necessaria per la fase di controllo.

5.2.2.2 Generazione di codice del blocco custom "qbmove_tx"

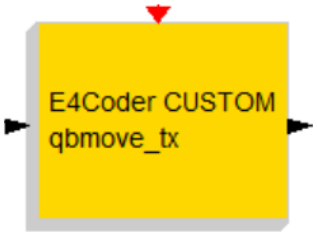


Figura 5.15.: *"Blocco custom qbmove_tx"*

Il blocco in figura 5.15 ha il compito di selezionare, in base al comando associato, una serie di informazioni quali l'ID, la codifica del comando e la dimensione del pacchetto del protocollo di comunicazione. Esso è caratterizzato da un ingresso che riceve dal blocco control, in cui sono contenuti i nuovi valori da passare al dispositivo e un'uscita da passare al blocco custom msg_tx, che in base al protocollo di comunicazione selezionerà il pacchetto da inviare.
Anche in questo caso i parametri del blocco si possono osservare nella figura 5.14

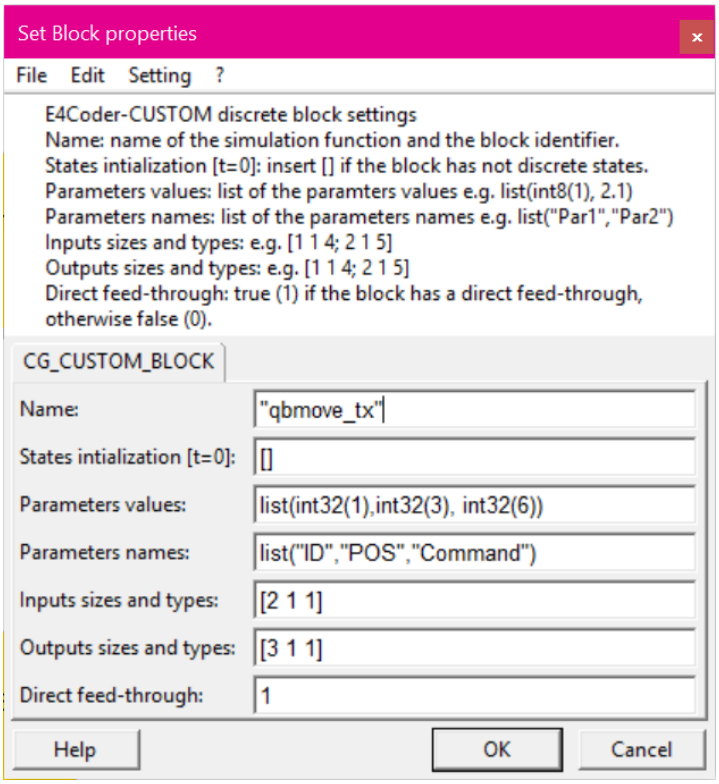


Figura 5.16.: *"Parametri del blocco custom qbmove_tx"*

I parametri di tale blocco sono sempre ID, POS e Command che anche in questo caso devono essere gli stessi del blocco msg_tx a cui è collegato. Le funzioni che descrivono il comportamento del blocco sono:

void qbmove_tx_Init(qbmove_tx_Param ∗Param, real ∗In1, real ∗Out1);

void qbmove_tx_OutputUpdate(qbmove_tx_Param ∗Param, real ∗In1, real ∗Out1);

void qbmove_tx_StateUpdate(qbmove_tx_Param ∗Param, real ∗In1, real ∗Out1);

void qbmove_tx_Terminate(qbmove_tx_Param ∗Param, real ∗In1, real ∗Out1).

Con la funzione qbmove_tx_Init() vengono passate al blocco custom msg_tx, le informazioni necessarie per selezionare il pacchetto da inviare al dispositivo per l'attivazione; con la funzione qbmove_tx_Terminate(), vengono passate le informazioni per disattivare il dispositivo.
Con la funzione qbmove_tx_OutputUpdate(), invece, oltre a mandare informazioni al blocco msg_tx relative al comando associato, quali la codifica del comando e la dimensione del pacchetto del protocollo di comunicazione, si aggiornano anche le strutture che contengono i nuovi parametri, ricavati dall'algoritmo di controllo da passare al dispositivo.

5.2.3 Blocco custom per l'algoritmo di controllo

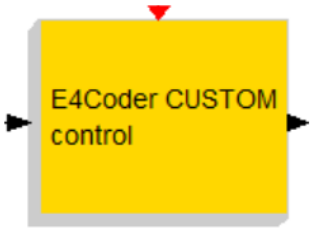


Figura 5.17.: *"Blocco custom control"*

Il blocco custom control, in figura 5.17, è il blocco utilizzato per implementare l'algoritmo di controllo.

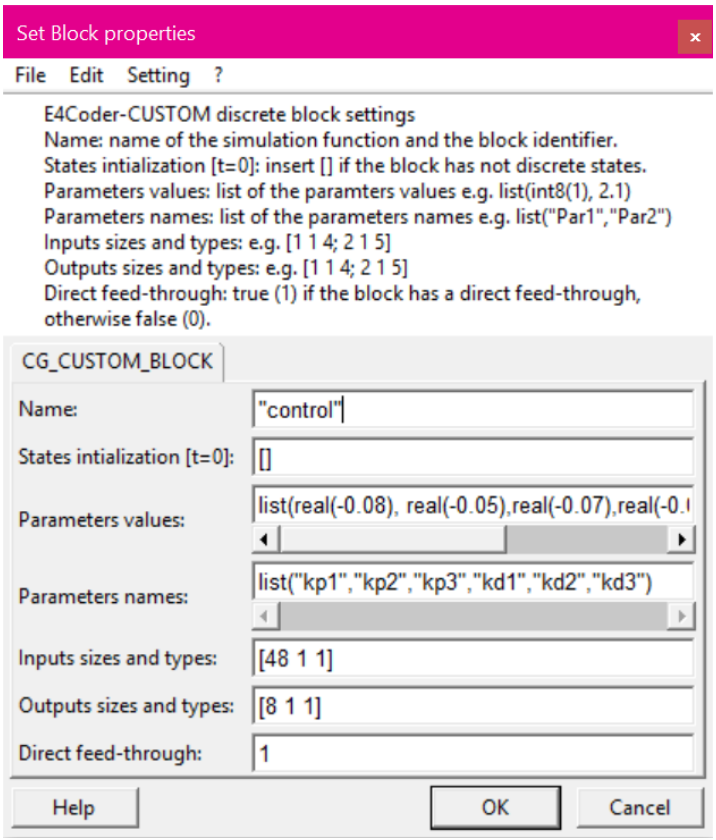


Figura 5.18.: *"Parametri del blocco custom control"*

I parametri di questo blocco, come mostrato in figura 5.18 sono solo i coefficienti del controllore; in questo caso, avendo scelto come controllore un controllore proporzionale derivativo con compensazione di gravità, i coefficienti saranno quelli della componente proporzionale e della componente derivativa.
Tale blocco è caratterizzato da un ingresso e un'uscita; in particolare, l'ingresso di tale blocco è dato dall'unione delle uscite dei blocchi a monte che contengono le misure richieste al dispositivo, ovvero, in questo caso le misure di posizione costituite dalla posizione dei due servomotori e dell'albero centrale.
Gli ingressi ricevuti sono in funzione della posizione dei blocchi nello schema, tuttavia, per l'algoritmo di controllo è necessario che tali ingressi siano in funzione della posizione dei dispositivi nella catena reale del braccio. Per questo motivo in ingresso, oltre alle informazioni sulle misure, si ottiene anche la posizione dei vari dispositivi nella catena, in questo modo, è possibile riorganizzare gli ingressi in funzione della loro posizione nella catena. Questo ci permette, facendo lo schema, di non essere vincolati a disporre i sottoblocchi, corrispondenti ai dispositivi, nell'ordine in cui questi si trovano nella catena lasciandoci liberi di posizionarli in base al valore del proprio ID o in qualsiasi altro modo.
Le funzioni che definiscono il comportamento del blocco custom control sono:

void control_Init(control_Param *Param, real *In1, real *Out1);

void control_OutputUpdate(control_Param *Param, real *In1, real *Out1);

void control_StateUpdate(control_Param *Param, real *In1, real *Out1);

void control_Terminate(control_Param *Param, real *In1, real *Out1).

Nella funzione control_OutputUpdate() una volta riorganizzati gli ingressi in funzione della loro posizione, viene richiamata la funzione control_qb() in cui viene implementato l'algoritmo di controllo vero e proprio; tale funzione riceve in ingresso le misure ricevute e i coefficienti del controllore e restituisce i nuovi valori elaborati dall'algoritmo di controllo da spedire al dispositivo.

5.3 Controllo Arimoto

Come già accennato, l'algoritmo di controllo implementato è un controllo proporzionale derivativo, PD, con compensazione di gravità realizzato nello spazio operativo, ovvero nello spazio in cui è definita la posa e l'orientazione dell'organo terminale.
L'idea alla base di un controllore PID è quello di avere un'architettura standard per il controllo di un generico processo; tale controllo, infatti, può essere applicato ai più svariati ambiti, dal controllo di una portata di fluido alla misurazione della temperatura in quanto, l'unico modo per variare il comportamento di un PID e adattarlo ai vari funzionamenti consiste nel variare i valori dei coefficienti dei vari termini. L'azione proporzionale è l'azione di controllo più semplice da utilizzare ed è quella che si basa sull'errore proporzionale tra il valore di riferimento e la variabile da controllare, in questo modo maggiore sarà l'errore, maggiore sarà l'azione di controllo svolta dal controllore stesso. Con l'azione proporzionale si produce una differenza tra il valore richiesto e quello effettivamente ottenuto, differenza che può essere ridotta aumentando il guadagno proporzionale del controllore, azione che però può provocare un aumento delle oscillazioni generate in seguito ai rapidi transitori; tale controllore è infatti solitamente utilizzato solo in processi che risultano essere asintoticamente stabili. Inoltre tale azione di controllo da sola non garantisce l'annullamento dell'errore a regime.
Con l'azione derivativa, invece, ciò che si vuole ottenere è un miglioramento della stabilità del sistema a ciclo chiuso; tale azione fornisce la derivata rispetto al tempo dell'errore. Tuttavia tale azione, in alcuni casi, può fornire l'inconveniente di amplificare i segnali con un contenuto armonico a frequenze elevate introducendo instabilità, per questo motivo tale termine non è sempre presente.
Il compito dell'azione integrale è quello di far si che a regime la variabile controllata assuma il valore di riferimento, in questo modo tale azione può essere vista come un'azione per l'azzeramento dell'errore a regime introdotto dall'azione proporzionale.

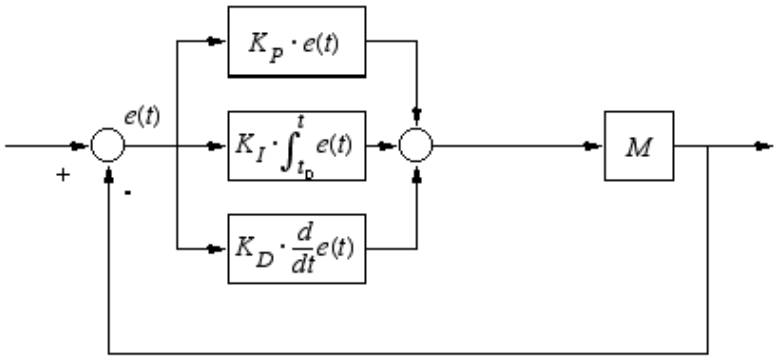


Figura 5.19.: *"Schema base di un controllore PID"*

Al posto di utilizzare un classico controllore PID, come quello mostrato in figura 5.19,

si procede utilizzando un controllore PD con compensazione di gravità, l'algoritmo di controllo, infatti è tanto più efficace quanto migliori sono le conoscenze sul modello che si vuole controllare e in questo caso si suppone di avere una perfetta conoscenza dell'azione gravitazionale che agisce sul sistema; tale controllo prende anche il nome di controllo Arimoto.
Partendo dall'equazione dinamica di un generico sistema:

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau \tag{5.1}$$

la coppia di controllo τ, si ottiene applicando la tecnica del controllo di Lyapunov e scegliendo come funzione di Lyapunov l'energia cinetica del sistema a cui si aggiunge un ulteriore termine

$$V(\dot{q},\tilde{q}) = \frac{1}{2}\dot{q}^T B(q)\dot{q} + \tilde{q}^T Q\tilde{q} \tag{5.2}$$

dove $\tilde{q}$=$q_D - q$ è l'errore tra la posizione desiderata, q_D, e quella ottenuta q. L'espressione della coppia τ risulta essere:

$$\tau = K_p\tilde{q} + K_d\dot{\tilde{q}} + G(q), \tag{5.3}$$

tale tipo di controllo risulta essere un controllo di tipo Coppia Calcolata inesatto, in quanto si ha una stima esatta solo di G(q), mentre non si hanno informazioni sulle altre matrici che esprimono la dinamica del sistema. Il fatto di prendere una funzione di Lyapunov, V(t) che sia funzione non solo delle velocità ai giunti ma anche dell'errore di posizione ci assicura che sotto l'influenza del controllo:

- il manipolatore tende asintoticamente ad una posizione di equilibrio data da $\dot{q} = 0$;
- a regime l'errore di posizione tende a zero, ovvero $q_D = q$.

Tuttavia, utilizzando il criterio di Lyapunov l'unica conclusione a cui si giunge è quella relativa al raggiungimento della posizione di equilibrio, in quanto, si ha che:

$$V(\dot{q}) > 0$$

$$\dot{V}(\dot{q},\tilde{q}) \leq 0.$$

L'asintotica stabilità del sistema e quindi, il fatto che a regime l'errore di posizione tende a zero, è dimostrata mediante l'utilizzo del lemma di Krasowskii; tale lemma afferma che se la funzione di Lyapunov $V(\dot{q})$, continua insieme alle sue derivate prime, è definita positiva in un punto q e la $\dot{V}(\dot{q},\tilde{q})$ è semi-definita negativa nello stesso punto, allora, se l'insieme dei punti in cui si annulla la $\dot{V}(\dot{q},\tilde{q})$:

$$S = \left\{\dot{q} : \dot{V}(\dot{q},\tilde{q}) = 0\right\}$$

non contiene traiettorie perturbate, si ha che il punto q è uno stato di equilibrio asintoticamente stabile.
In questo caso, infatti, considerando la coppia di controllo Arimoto, si ha che:

$$\dot{V}(\dot{q},\tilde{q}) = \dot{q}^T(-Q\tilde{q} + K_p\tilde{q} + K_d\dot{\tilde{q}})$$

da cui ponendo $Q = K_p$ si ricava che

$$\dot{V}(\dot{q},\tilde{q}) = \dot{q}^T(-K_p\tilde{q} + K_p\tilde{q} + K_d\dot{\tilde{q}})$$

$$\dot{V}(\dot{q},\tilde{q}) = \dot{q}^T K_d \dot{\tilde{q}}.$$

Inoltre, poichè $\tilde{q} = q_D - q$ e q_D è una traiettoria punto-punto, si ha che:

$$\dot{\tilde{q}} = -\dot{q}, \qquad \dot{q}_D = 0$$

$$\dot{V}(\dot{q},\tilde{q}) = -\dot{q}^T K_d \dot{q}$$

Quindi, applicando il lemma di Krasowskii si ha che:

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} = K_p\tilde{q} - K_d\dot{q}.$$

Da ciò considerando che

$$\lim_{t\to\infty} \dot{q} = 0;$$

$$\lim_{t\to\infty} \ddot{q} = 0;$$

si ricava che

$$K_p\tilde{q} = 0 \Longrightarrow \tilde{q} = 0$$

ovvero l'errore a regime è nullo, quindi, il controllo proposto assicura l'inseguimento asintotico di una traiettoria punto-punto.
Una scelta opportuna della matrice K_d ci permette di rendere più o meno elevata la velocià di convergenza a zero dell'errore di posizione $\tilde{q}$ da parte del manipolatore. A parità di $\dot{q}$, infatti, più i termini sulla diagonale risultano elevati, maggiore sarà la velocità di convergenza dell'errore.
Il controllore Proporzionale-Derivativo opera inserendo rigidezze K_p e viscosità, K_d virtuali a livello di giunto:

- la forza elastica $K_p\tilde{q}$, costringe la posizione di giunto q a inseguire il riferimento q_D;
- la forza di attrito $K_d\dot{\tilde{q}}$ tende, invece, a dissipare l'energia elastica accumulata durante l'evoluzione e quindi a smorzare le oscillazioni.

Tuttavia, in questo caso, per non incorrere nei problemi dovuti alle singolarità del sistema, il controllo non viene effettuato nello spazio dei giunti ma nello spazio operativo; pertanto l'algoritmo di controllo risulta essere:

$$\tau = G(q) + J_A^T(q)K_p\tilde{x} - J_A^T(q)K_d J_A\dot{q} \tag{5.4}$$

dove:

- G(q) è il contributo gravitazionale;
- $J_A^T(q)$ è la trasposta dello Jacobiano;
- $K_p\tilde{x}$ è il contributo della componente proporzionale nello spazio operativo con K_p matrice definita positiva dei coefficienti proporzionali e $\tilde{x} = x_d - x_e$;
- $K_d J_A\dot{q}$ contributo della componente derivativa riportata nello spazio operativo.

Nel caso della generazione di codice la dinamica del sistema che si utilizza per ricavare i nuovi ingressi da passare al dispositivo è data da:

$$\tau = k_1 \sinh(a_1(x - q_1)) + k_2 \sinh(a_2(x - q_2)) \tag{5.5}$$

dove:

- x è la posizione dell'albero motore;
- q_1 e q_2 sono le posizioni dei due servomotori;
- k_1,k_2,a_1,a_2 sono parametri utili per la descrizione della dinamica del sistema.

Se necessario, per ottenere la variazione della stiffness, invece si utilizza l'espressione:

$$\sigma = a_1 k_1 \cosh(a_1(x - q_1)) + a_2 k_2 \cosh(a_2(x - q_2)) \tag{5.6}$$

5.4 Simulazione

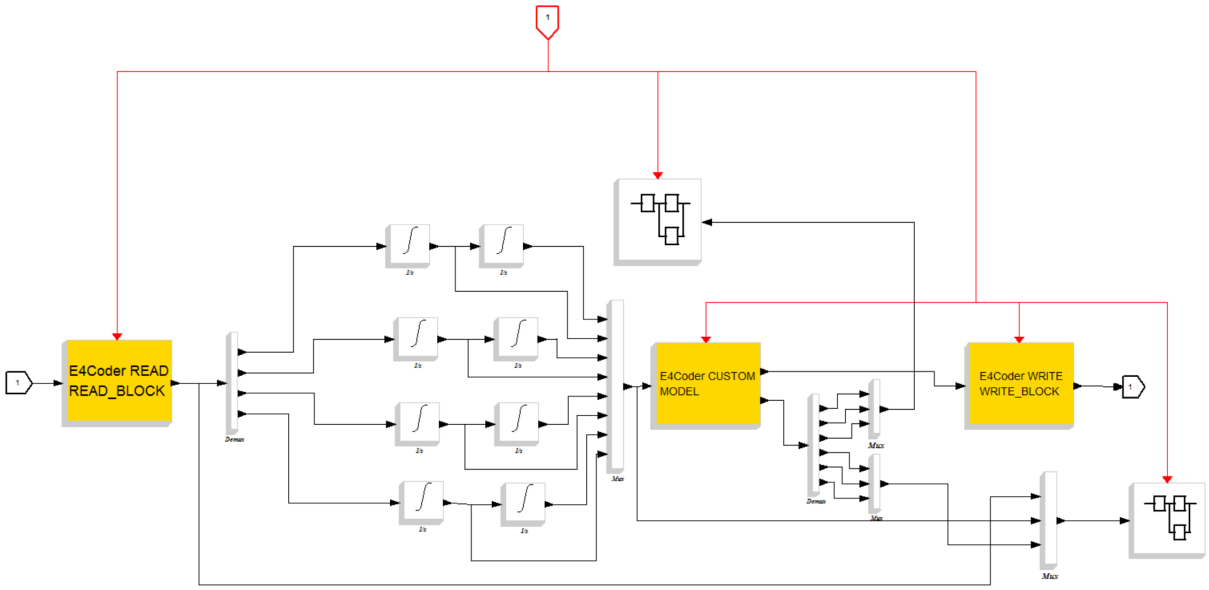


Figura 5.20.: *"Sottoblocco aggiunto per la simulazione"*

Per quanto la riguarda la parte di simulazione, di cui lo schema a blocchi viene mostrato in figura 5.20, il comportamento dei blocchi custom msg_tx, msg_rx, qbmove_tx e qbmove_rx è stato implementato in modo che il compito di tali blocchi sia quello di riportare in uscita il valore ricevuto in ingresso: i valori delle posizioni e delle velocità ai giunti per i blocchi custom msg_rx e qbmove_rx e i valori delle accelerazioni ai giunti prodotte dall'algoritmo di controllo per quanto riguarda i blocchi custom qbmove_tx e msg_tx; in fase di simulazione, infatti, non è necessario replicare la comunicazione tra la board e il dispositivo. In ingresso al sottoblocco della simulazione arrivano, quindi, le accelerazioni dei giunti, prodotte dal controllo, che vengono integrate una prima volta per ottenere le velocità e una seconda volta per ottenere le posizioni. I valori di velocità e posizione costituiscono gli ingressi del blocco custom MODEL, mostrato in figura 5.21, in cui viene implementato il

modello del braccio da cui si ricava la sua dinamica. Per la parte di simulazione è infatti, necessario conoscere la dinamica del sistema.

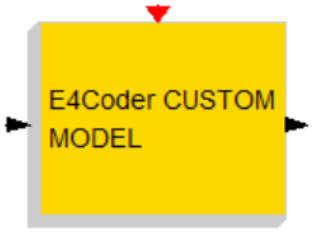


Figura 5.21.: *"Blocco custom MODEL."*

In questo caso l'implementazione del blocco custom MODEL viene fatta solo per la parte di simulazione e le funzioni che descrivono il suo comportamento sono:

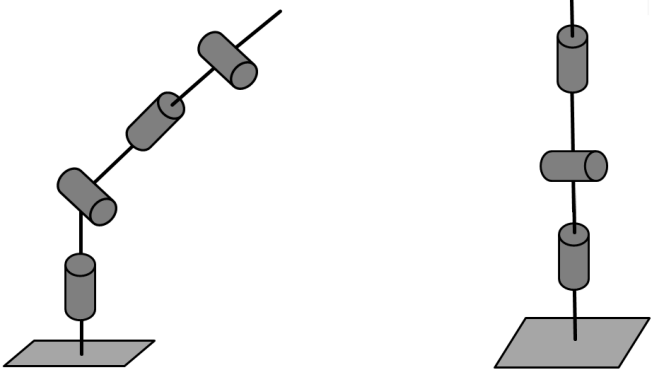
void MODEL_Init(MODEL_Param *Param, real *In1, real *Out1);

void MODEL_OutputUpdate(MODEL_Param *Param, real *In1, real *Out1);

void MODEL_StateUpdate(MODEL_Param *Param, real *In1, real *Out1);

void MODEL_Terminate(MODEL_Param *Param, real *In1, real *Out1).

Dalla funzione MODEL_OutputUpdate(), ad ogni iterazione si richiama la funzione in cui viene implementata la dinamica del sistema.
La dinamica del sistema si ricava dalla convenzione di Denavit-Hartenberg, convenzione con la quale è possibile esprimere la posa, ovvero la posizione e l'orientazione dell'organo terminale, End-Effector, in funzione della terna globale. Avendo a disposizione quattro dispositivi qbmove, il braccio di cui si vuole costruire il modello risulta composto da quattro giunti rotoidali come riportato nell'immagine 5.22:



j	theta	d	a	alpha	offset
1	q1	0	0	-1.571	0
2	q2	0	0	1.571	1.571
3	q3	0.14	0	-1.571	0
4	q4	0	0.066	0	-1.571

Figura 5.22.: *Schema del modello del braccio con quattro giunti rotoidali e relativa tabella di Denavit-Hartenberg.*

Nell'immagine troviamo anche la tabella di Denavit-Hartenberg nella quale si ritrovano tutti i parametri necessari per effettuare le trasformazioni da un sistema di riferimento a quello successivo.
Seguendo questa convenzione si procede numerando i link da 0 a n+1 e i giunti da 1 a n; in seguito si fissano i sistemi di riferimento solidali ad ogni link seguendo i seguenti passi:

- l'asse z_i del sistema di riferimento i-simo, viene fissato in modo che abbia la direzione dell'asse del giunto a valle, ovvero del giunto (i+1)-esimo. Se il giunto è rotoidale oltre alla direzione è fissato anche il verso dell'asse, invece, se il giunto è prismatico è fissata solo la direzione;

- l'asse x_i del sistema di riferimento i-simo ha la stessa direzione della normale comune agli assi z_{i-1} e z_i con direzione che va dal giunto i al giunto i+1;

- l'origine del sistema di riferimento viene posta sull'asse z_i nel punto di intersezione tra l'asse z_i e la normale comune agli assi z_{i-1} e z_i. Se gli assi z_{i-1} e z_i sono paralleli e il giunto è rotoidale, è possibile scegliere l'origine in modo tale da annullare d_i, ovvero la traslazione lungo l'asse z, mentre se il giunto è prismatico si sceglie una posizione di fine corsa;

- l'asse y_i viene posizionato in modo da completare la terna levogira;

- infine, la terna solidale all'organo terminale viene scelta posizionando l'asse z_n in modo che abbia la stessa direzione dell'asse z_{n-1}, l'asse x_n in modo che abbia direzione perpendicolare all'asse z_{n-1} e l'asse y_n in modo da ottenere una terna levogira.

Seguendo questi passi, le operazioni necessarie per ottenere la matrice di trasformazione da un sistema di riferimeno all'altro sono:

$$T_i^{i-1} = T_z(d) * R_z(\theta) * T_x(a) * R_x(\alpha).$$

Eventualmente, se il sistema di riferimento globale e quello solidale al primo link non coincidono, a queste matrici va aggiunta in cima una matrice che tenga della trasformazione tra il sistema di riferimento globale e quello solidale al primo link (link 0).
In questo caso, poichè il sistema di riferimento globale e quello solidale al primo link coincidono non è necessario effettuare ulteriori trasformazioni. Le altre matrici di trasformazione tra i vari sistemi di riferimento risultano essere:

$$A_1^0 = \begin{bmatrix} cos(q_1) & 0 & -sin(q_1) & 0 \\ sin(q_1) & 0 & cos(q_1) & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_2^1 = \begin{bmatrix} cos(q_2) & 0 & sin(q_2)ln_2 * cos(q_2) & \\ sin(q_2) & 0 & -cos(q_2)ln_2 * sin(q_2) & \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3^2 = \begin{bmatrix} cos(q_3) & 0 & -sin(q_3) & 0 \\ sin(q_3) & 0 & cos(_q3) & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_4^3 = \begin{bmatrix} cos(q_4) & -sin(q_4) & 0 & ln_4 * cos(q_4) \\ sin(q_4) & cos(q_4) & 0 & ln_4 * sin(q_4) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

mentre la matrice di trasformazione complessiva risulta essere:

$$A_4^0 = \begin{bmatrix} c_{q_1} * c_{q_2} * c_{q_4} + (-c_{q_1} * c_{q_3} * s_{q_2} - s_{q_1} * s_{q_3}) * s_{q_4} & c_{q_4} * (-c_{q_1} * c_{q_3} * s_{q_2} - s_{q_1} * s_{q_3}) - c_{q_1} * c_{q_2} * s_{q_4} & -c_{q_3} * s_{q_1} + c_{q_1} * s_{q_2} * s_{q_3} & d_3 * c_{q_1} * c_{q_2} + ln_4 * c_{q_1} * c_{q_2} * c_{q_4} + ln_4 * (-c_{q_1} * c_{q_3} * s_{q_2} - s_{q_1} * s_{q_3}) * s_{q_4} \\ c_{q_2} * c_{q_4} * s_{q_1} + (-c_{q_3} * s_{q_1} * s_{q_2} + c_{q_1} * s_{q_3}) * s_{q_4} & c_{q_4} * (-c_{q_3} * s_{q_1} * s_{q_2} + c_{q_1} * s_{q_3}) - c_{q_2} * s_{q_1} * s_{q_4} & c_{q_1} * c_{q_3} + s_{q_1} * s_{q_2} * s_{q_3} & d_3 * c_{q_2} * s_{q_1} + ln_4 * c_{q_2} * c_{q_4} * s_{q_1} + ln_4 * (-c_{q_3} * s_{q_1} * s_{q_2} + c_{q_1} * s_{q_3}) * s_{q_4} \\ -c_{q_4} * s_{q_2} - c_{q_2} * c_{q_3} * s_{q_4} & -c_{q_2} * c_{q_3} * c_{q_4} + s_{q_2} * s_{q_4} & c_{q_2} * s_{q_3} & -d_3 * s_{q_2} - ln_4 * c_{q_4} * s_{q_2} - ln_4 * c_{q_2} * c_{q_3} * s_{q_4} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Lo Jacobiano del sistema, ovvero la matrice che permette di rimappare la velcità dell'End-Effector nelle velocità ai giunti risulta essere:

$$J(q) = \begin{bmatrix} -d_3 * c_{q_2} * s_{q_1} - ln_4 * c_{q_2} * c_{q_4} * s_{q_1} + ln_4 * c_{q_3} * s_{q_1} * s_{q_2} * s_{q_4} - ln_4 * c_{q_1} * s_{q_3} * s_{q_4} & -d_3 * c_{q_1} * s_{q_2} - ln_4 * c_{q_1} * cq_4 * s_{q_2} - ln_4 * c_{q_1} * c_{q_2} * c_{q_3} * s_{q_4} & -ln_4 * c_{q_3} * s_{q_1} * s_{q_4} + ln_4 * c_{q_1} * s_{q_2} * s_{q_3} * s_{q_4} & -ln_4 * c_{q_1} * c_{q_3} * c_{q_4} * s_{q_2} - ln_4 * c_{q_4} * s_{q_1} * s_{q_3} - ln_4 * c_{q_1} * c_{q_2} * s_{q_4} \\ d_3 * c_{q_1} * c_{q_2} + ln_4 * c_{q_1} * c_{q_2} * c_{q_4} - ln_4 * c_{q_1} * c_{q_3} * s_{q_2} * s_{q_4} - ln_4 * s_{q_1} * s_{q_3} * s_{q_4} & -d_3 * s_{q_1} * s_{q_2} - ln_4 * c_{q_4} * s_{q_1} * s_{q_2} - ln_4 * c_{q_2} * c_{q_3} * s_{q_1} * s_{q_4} & ln_4 * c_{q_3} * s_{q_1} * s_{q_4} + ln_4 * c_{q_1} * s_{q_2} * s_{q_3} * s_{q_4} & -ln_4 * c_{q_3} * c_{q_4} * s_{q_1} * s_{q_2} + ln_4 * c_{q_1} * c_{q_4} * s_{q_3} - ln_4 * c_{q_2} * s_{q_1} * s_{q_4} \\ 0 & -d_3 * c_{q_2} - ln_4 * c_{q_2} * c_{q_4} + ln_4 * c_{q_3} * s_{q_2} * s_{q_4} & ln_4 * c_{q_2} * s_{q_3} * s_{q_4} & -ln_4 * c * q_2 * c_{q_3} * c_{q_4} + ln_4 * s_{q_2} * s_{q_4} \\ 0 & -s_{q_1} & c_{q_1} * c_{q_2} & (-c_{q_3}) * s_{q_1} + c_{q_1} * s_{q_2} * s_{q_3} \\ 0 & c_{q_1} & c_{q_2} * s_{q_1} & c_{q_1} * c_{q_3} + s_{q_1} * s_{q_2} * s_{q_3} \\ 1 & 0 & -s_{q_2} & c_{q_2} * s_{q_3} \end{bmatrix}$$

Note le matrici di trasformazione e lo Jacobiano del sistema si ricavano le altre matrici necessarie per esprimere l'equazione dinamica del sistema, ovvero:

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau. \tag{5.7}$$

In particolare la matrice B(q) si ottiene come:

$$B(q) = \sum_{i=1}^{N} \left\{ m_i J_{p_i}(q)^T J_{p_i}(q) + J_{O_i}(q)^T \left[{}^0R_{l_i}(q)\, {}^{l_i}I_{G_i}^0 R_{l_i}(q)^T \right] J_{O_i}(q) \right\}$$

dove N sono i giunti che costituiscono il sistema, m_i le masse, J_{p_i} e J_{O_i} sono rispettivamente lo Jacobiano di posizione e quello di orientazione, I_{G_i} è la matrice delle inerzie e ${}^0R_{l_i}(q)$ è la matrice di rotazione.
In questo caso la matrice delle inerzie B(q) risulta essere:

$$B(q) = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{24} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

dove:

$$\begin{aligned} b_{11} = {} & 0.00075504 + \cos(q_2)^2 * (0.0196 * m_3 + 0.021778 * m_4 + 0.01848 * m_4 * \cos(q_4) + 0.00218 \\ & * m_4 * \cos(2 * q_4)) + m_4 * \cos(q_2) * \cos(q_3) * (-0.01848 - 0.008712 * \cos(q_4)) * \sin(q_2) * \\ & \sin(q_4) + m_4 * (0.003267 - 0.001089 * \cos(2 * q_2) - 0.002178 * \cos(q_2)^2 * \cos(2 * q_3)) * \sin(q_4)^2; \end{aligned}$$

$$b_{12} = m_4 * \sin(q_3) * \sin(q_4) * ((0.00924 + 0.004356 * \cos(q_4)) * \sin(q_2) + 0.004356 * \cos(q_2) * \cos(q_3) * \sin(q_4));$$

$$b_{13} = +0.00037752 * \cos(q_2) + m_4 * (\cos(q_4)^2 * (0.002178 * \sin(q_2)) - 0.002178 * \sin(q_2) + 0.00924 * \cos(q_2) * \cos(q_3) * \sin(q_4) + (0.004356 * \cos(q_2) * \cos(q_3)) * \cos(q_4) * \sin(q_4) + (-0.002178 * \sin(q_2)) * \sin(q_4)^2);$$

$$b_{14} = (0.00018876 * \sin(q_2)) * \sin(q_3) + m_4 * \cos(q_2) * (0.004356 + 0.00924 * \cos(q_4)) * \sin(q_3);$$

$$b_{21} = m_4 * \sin(q_3) * \sin(q_4) * ((0.00924 + 0.004356 * \cos(q_4)) * \sin(q_2) + 0.004356 * \cos(q_2) * \cos(q_3) * \sin(q_4));$$

$$b_{22} = 0.00056628 + 0.0196 * m_3 + 0.022867 * m_4 + 0.01848 * m_4 * \cos(q_4) + m_4 * \cos(2 * q_3) * (0.001089 - 0.001089 * \cos(2 * q_4)) + 0.001089 * m_4 * \cos(2 * q_4);$$

$$b_{23} = m_4 * (-0.00924 - 0.004356 * \cos(q_4)) * \sin(q_3) * \sin(q_4);$$

$$b_{24} = 0.00018876 * \cos(q_3) + m_4 * (\cos(q_3) * (0.004356 + 0.00924 * \cos(q_4)));$$

$$b_{31} = 0.00037752 * \cos(q_2) + m_4 * (\cos(q_4)^2 * (+0.002178 * \sin(q_2)) - 0.002178 * \sin(q_2) + 0.00924 * \cos(q_2) * \cos(q_3) * \sin(q_4) + (0.004356 * \cos(q_2) * \cos(q_3)) * \cos(q_4) * \sin(q_4) + (-0.002178 * \sin(q_2)) * \sin(q_4)^2);$$

$$b_{32} = m_4 * (-0.00924 - 0.004356 * \cos(q_4)) * \sin(q_3) * \sin(q_4);$$

$$b_{33} = 0.00037752 + 0.004356 * m_4 * \sin(q_4)^2;$$

$$b_{34} = 0;$$

$$b_{41} = 0.00018876 * \sin(q_2) * \sin(q_3) + m_4 * \cos(q_2) * (0.004356 + 0.00924 * \cos(q_4)) * \sin(q_3);$$

$$b_{42} = 0.00018876 * \cos(q_3) + m_4 * (\cos(q_3) * (0.004356 + 0.00924 * \cos(q_4)));$$

$$b_{43} = 0;$$

$$b_{44} = 0.00018876 + m_4 * (0.004356);$$

La matrice di Coriolis $C(q, \dot{q})$ si ricava a partire dalla matrice delle inerzie B(q):

$$C(q, \dot{q}) = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \\ C_{41} & C_{42} & C_{43} & C_{44} \end{bmatrix} .$$

Un generico termine di tale matrice sarà del tipo:

$$C_{ij} = \sum_{k=1}^{n} \Gamma_{jk}^{i} \dot{q}_k$$

dove i termini Γ_{jk}^{i} si ottengono come

$$\Gamma_{jk}^{i} = \frac{1}{2} \left[\frac{\delta b_{ij}}{\delta q_k} + \frac{\delta b_{ik}}{\delta b_j} - \frac{\delta b_{jk}}{\delta q_i} \right].$$

In particolare i termini della matrice di Coriolis risultano essere:

$$C_{11} = \Gamma_{21}^{1} \dot{q}_2 + \Gamma_{31}^{1} \dot{q}_3 + \Gamma_{41}^{1} \dot{q}_4;$$

$$C_{12} = \Gamma^1_{12}\dot{q}_1 + \Gamma^1_{22}\dot{q}_2 + \Gamma^1_{32}\dot{q}_3 + \Gamma^1_{42}\dot{q}_4;$$
$$C_{13} = \Gamma^1_{13}\dot{q}_1 + \Gamma^1_{23}\dot{q}_2 + \Gamma^1_{33}\dot{q}_3 + \Gamma^1_{43}\dot{q}_4;$$
$$C_{14} = \Gamma^1_{14}\dot{q}_1 + \Gamma^1_{24}\dot{q}_2 + \Gamma^1_{34}\dot{q}_3 + \Gamma^1_{44}\dot{q}_4;$$
$$C_{21} = \Gamma^2_{11}\dot{q}_1 + \Gamma^2_{31}\dot{q}_3 + \Gamma^2_{41}\dot{q}_4;$$
$$C_{22} = \Gamma^2_{32}\dot{q}_3 + \Gamma^2_{42}\dot{q}_4;$$
$$C_{23} = \Gamma^2_{13}\dot{q}_1 + \Gamma^2_{23}\dot{q}_2 + \Gamma^2_{33}\dot{q}_3 + \Gamma^2_{43}\dot{q}_4;$$
$$C_{24} = \Gamma^2_{14}\dot{q}_1 + \Gamma^2_{24}\dot{q}_2 + \Gamma^2_{34}\dot{q}_3 + \Gamma^2_{44}\dot{q}_4;$$
$$C_{31} = \Gamma^3_{11}\dot{q}_1 + \Gamma^3_{21}\dot{q}_2 + \Gamma^3_{41}\dot{q}_4;$$
$$C_{32} = \Gamma^3_{12}\dot{q}_1 + \Gamma^3_{22}\dot{q}_2 + \Gamma^3_{42}\dot{q}_4;$$
$$C_{33} = \Gamma^3_{43}\dot{q}_4;$$
$$C_{34} = \Gamma^3_{14}\dot{q}_1 + \Gamma^3_{24}\dot{q}_2 + \Gamma^3_{34}\dot{q}_3;$$
$$C_{41} = \Gamma^4_{11}\dot{q}_1 + \Gamma^4_{21}\dot{q}_2 + \Gamma^4_{31}\dot{q}_3;$$
$$C_{42} = \Gamma^4_{12}\dot{q}_1 + \Gamma^4_{22}\dot{q}_2 + \Gamma^3_{32}\dot{q}_3;$$
$$C_{43} = \Gamma^4_{13}\dot{q}_1 + \Gamma^4_{23}\dot{q}_2 + \Gamma^4_{33}\dot{q}_3;$$
$$C_{44} = 0.$$

Il vettore dell'azione gravitazionale G(q), invece, si ottiene come:

$$G(q) = -\left[{}^0g^T \left(\sum_{i=1}^{n} m_i J_{p_i}\right)\right]^T$$

che in questo caso, traslando il centro di massa del terzo link di una distanza pari a dl_3, risulta essere:

$$G(q) = \begin{bmatrix} 0 \\ g*(c_{q_2}*(-0.1*m_2 - dl_3*(m_3+m_4) - ln4*m_4*c_{q_4}) + ln4*m_4*c_{q_3}*s_{q_2}*s_{q_4}) \\ g*ln4*m_4*c_{q_2}*s_{q_3}*s_{q_4} \\ g*ln4*m_4*((-c_{q_2})*c_{q_3}*c_{q_4} + s_{q_2}*s_{q_4}) \end{bmatrix}.$$

Ricavate le matrici necessarie per esprimere la dinamica del sistema e considerando come algoritmo di controllo quello proporzionale derivativo con compensazione di gravità si ha che le variabili di controllo si ottengono partendo da:

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau; \tag{5.8}$$

da cui sostituendo l'espressione della coppia di controllo

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = G(q) + J_A^T(q)K_p\tilde{x} - J_A^T(q)K_d J_A\dot{q}; \tag{5.9}$$

si ricavano

$$\ddot{q} = B_q^{-1}\left[J_A^T(q)K_p\tilde{x} - J_A^T(q)K_d J_A\dot{q} - C(q,\dot{q})\dot{q}\right]; \tag{5.10}$$

Capitolo 6

Risultati

Di seguito si riportano i risultati ottenuti prima in fase di simulazione e in seguito attraverso delle prove pratiche.
Per quanto riguarda le simulazioni, a causa di alcuni problemi derivanti dagli algoritmi di integrazione disponibili su ScicosLab, si è reso necessario fare alcune prove di simulazione su Matlab. Di seguito, in figura 6.1 e 6.2 si riportano le immagini della rappresentazione del braccio in diverse configurazioni iniziali:

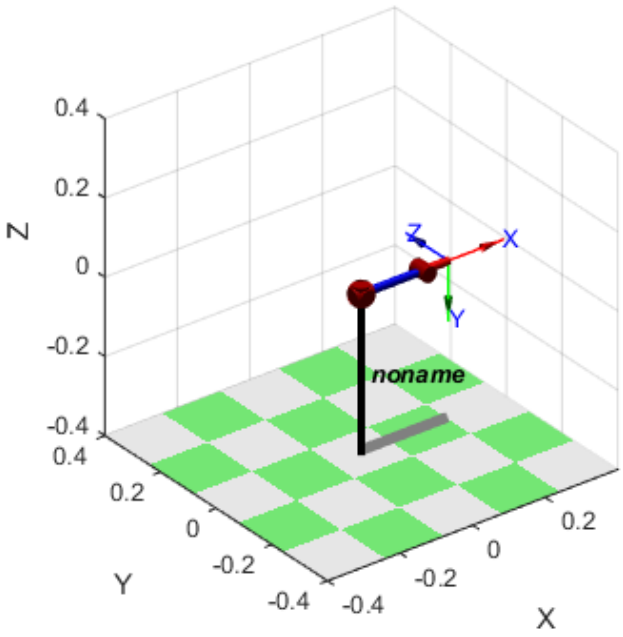


Figura 6.1.: *"Rappresentazione tridimensionale dello schema del braccio in configurazione [0, 0, 0, 0]"*

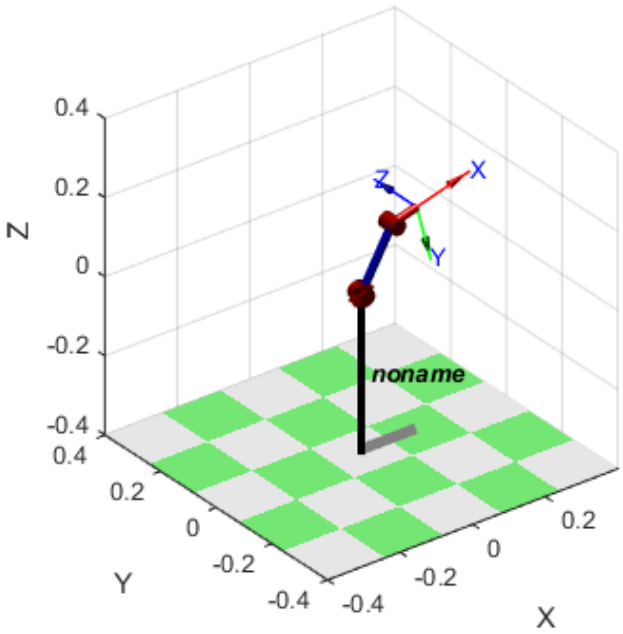


Figura 6.2.: *"Rappresentazione tridimensionale dello schema del braccio in configurazione [0, -pi/3, 0, pi/4]"*

6.1 Simulazioni

Come accennato, per le simulazioni è stato utilizzato il software Simulink, di cui in figura 6.3, si riporta lo schema utilizzato in fase di simulazione:

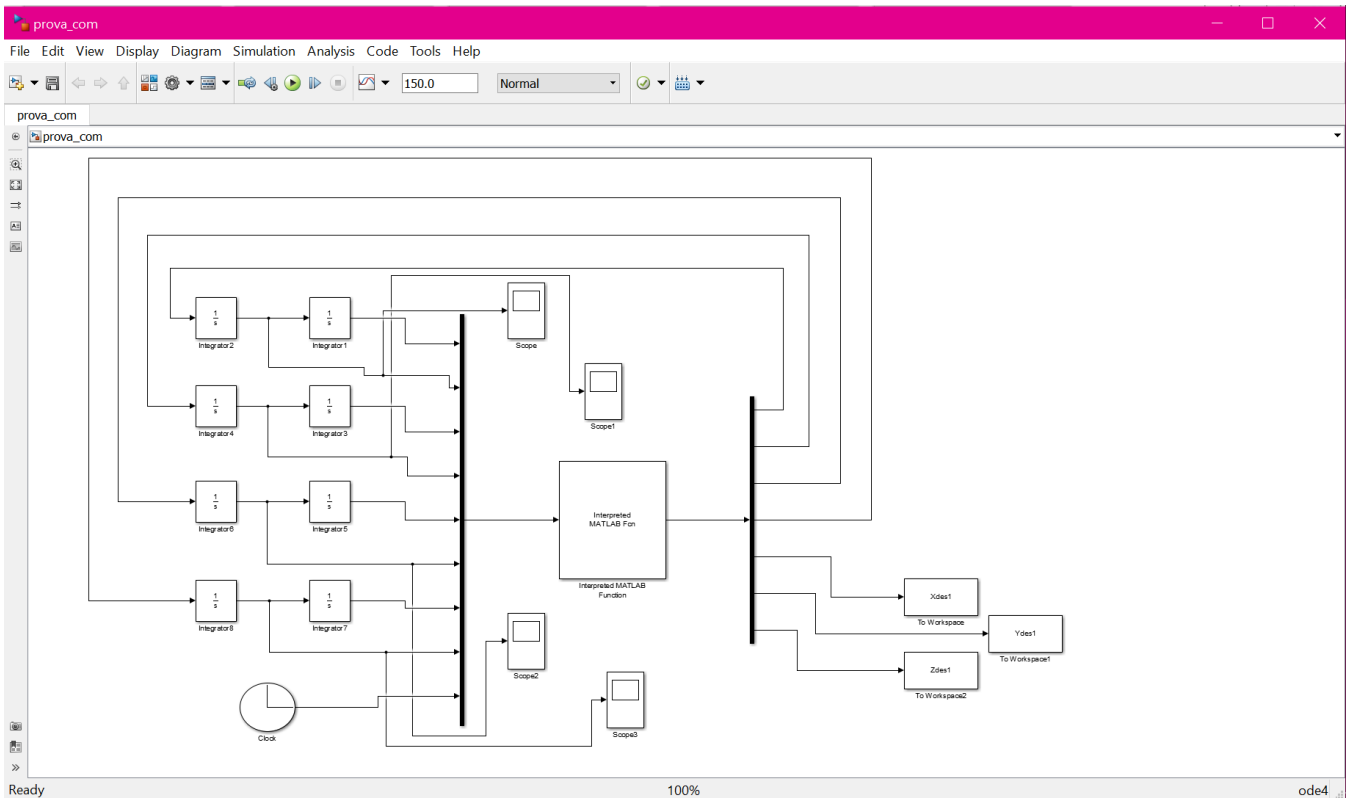


Figura 6.3.: *"Schema Simulink"*

In questo caso, si suppone che il braccio debba inseguire una traiettoria circolare di raggio 0,24 m nel piano x-y ad una quota z=0; si suppone inoltre, che traiettoria e braccio partano dallo stesso punto, quindi, nello stato iniziale il braccio si troverà in configurazione [0,0,0,0]. Come già accennato precedentemente, l'algoritmo di controllo utilizzato è quello Arimoto, ovvero un controllo PD con compensazione di gravità, e i cui coefficienti del controllo risultano essere:

$$kp = \begin{bmatrix} 95 & 0 & 0 \\ 0 & 95 & 0 \\ 0 & 0 & 95 \end{bmatrix}$$

$$kd = \begin{bmatrix} 25 & 0 & 0 \\ 0 & 25 & 0 \\ 0 & 0 & 25 \end{bmatrix}$$

Riportiamo di seguito i risultati ottenuti in fase di simulazione per l'inseguimento di una traiettoria circolare con Z=0:

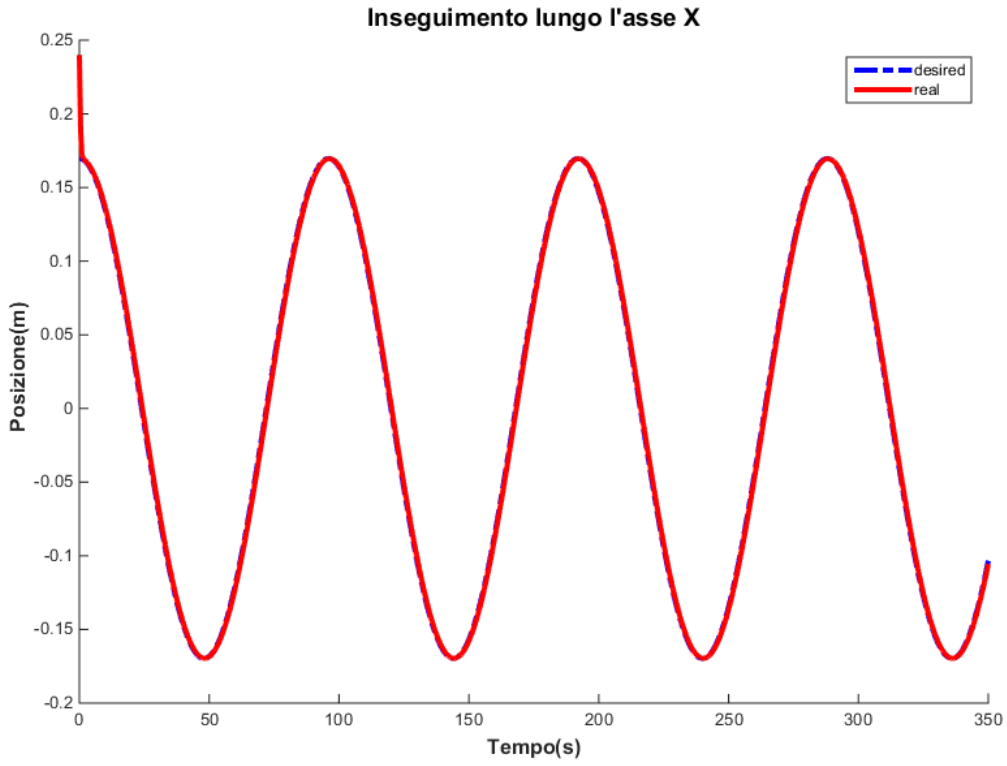


Figura 6.4.: *"Inseguimento della traiettoria lungo l'asse X ad una quota pari a 0 metri"*

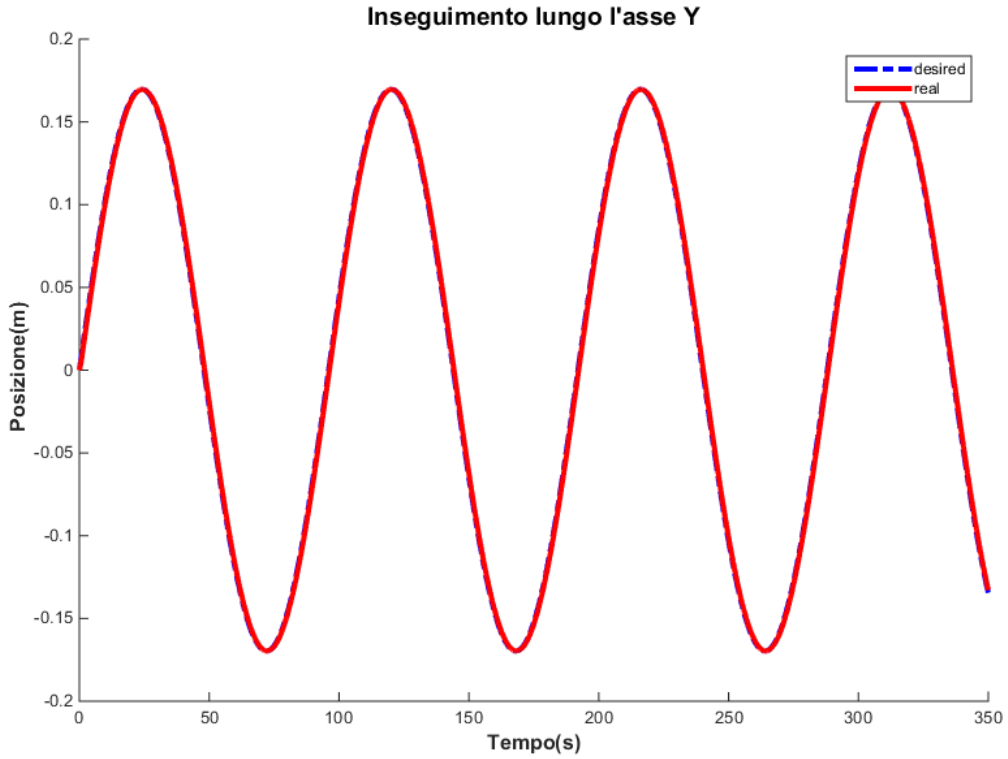


Figura 6.5.: *"Inseguimento della traiettoria lungo l'asse Y ad una quota pari a 0 metri"*

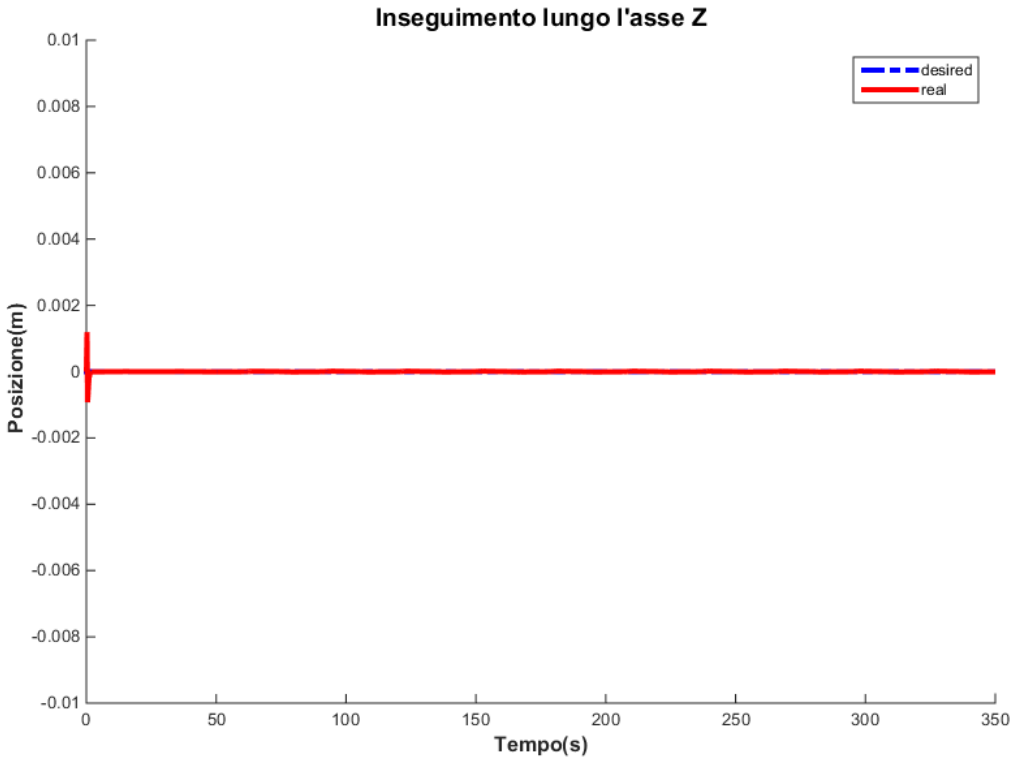


Figura 6.6.: *"Inseguimento della traiettoria lungo l'asse Z"*

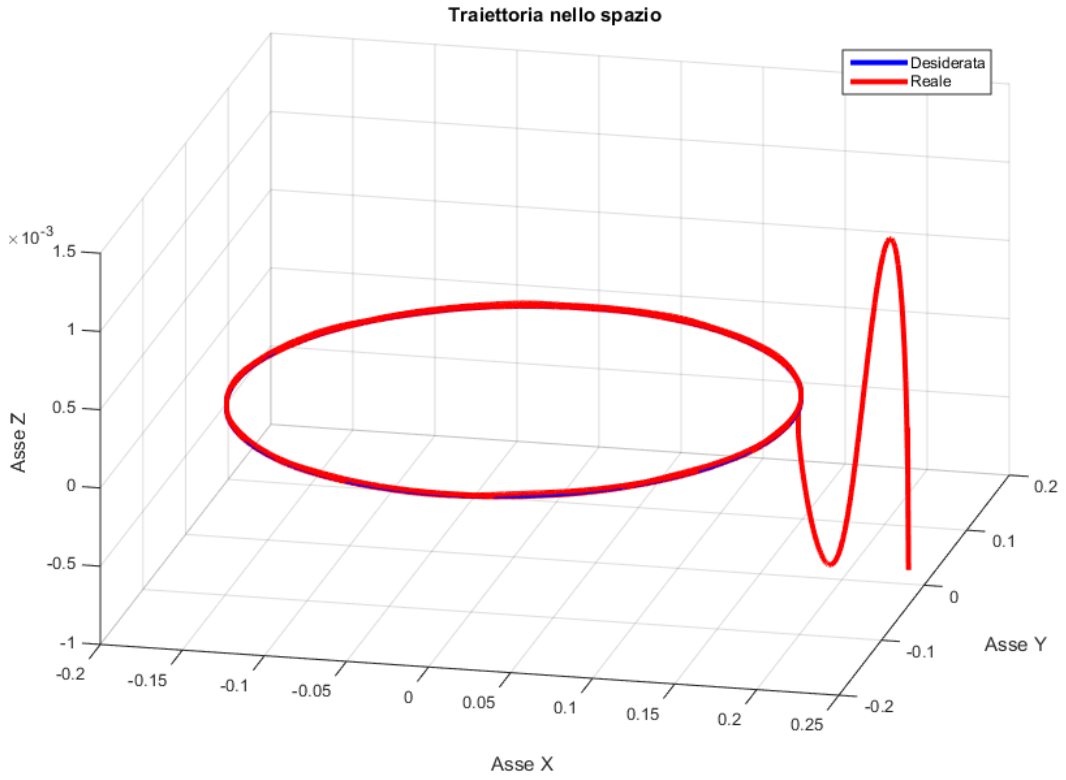


Figura 6.7.: *"Traiettoria nello spazio"*

Come si può osservare dalle immagini, in fase di simulazione si ottiene un buon inseguimento della traiettoria lungo tutti e tre gli assi, con traiettoria reale e desiderata che risultano quasi perfettamente sovrapposte. I relativi errori, mostrati di seguito, risultano, quindi, essere, come atteso, molto piccoli, dell'ordine di 10^{-3}-10^{-4}:

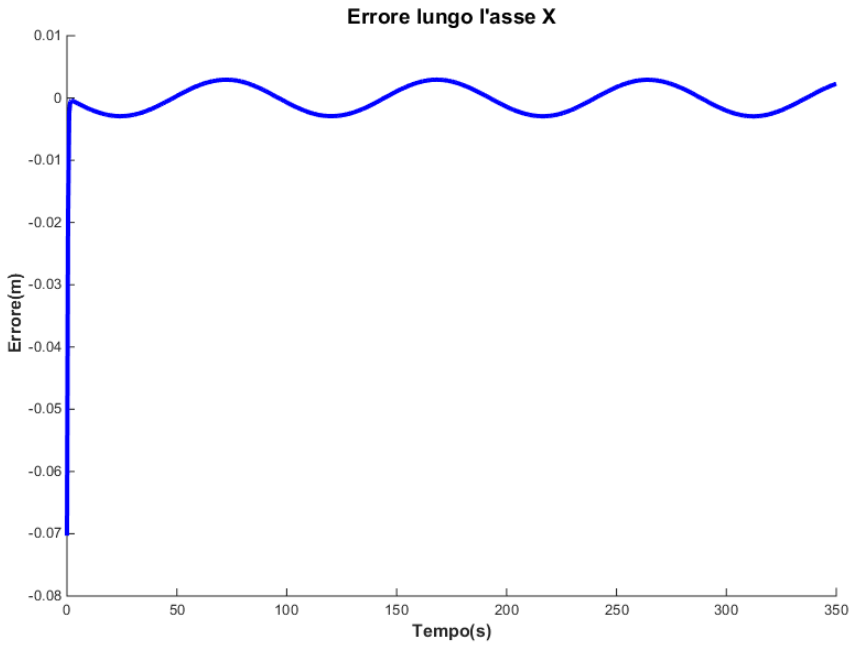


Figura 6.8.: *"Errore nell'inseguimento della traiettoria lungo l'asse X"*

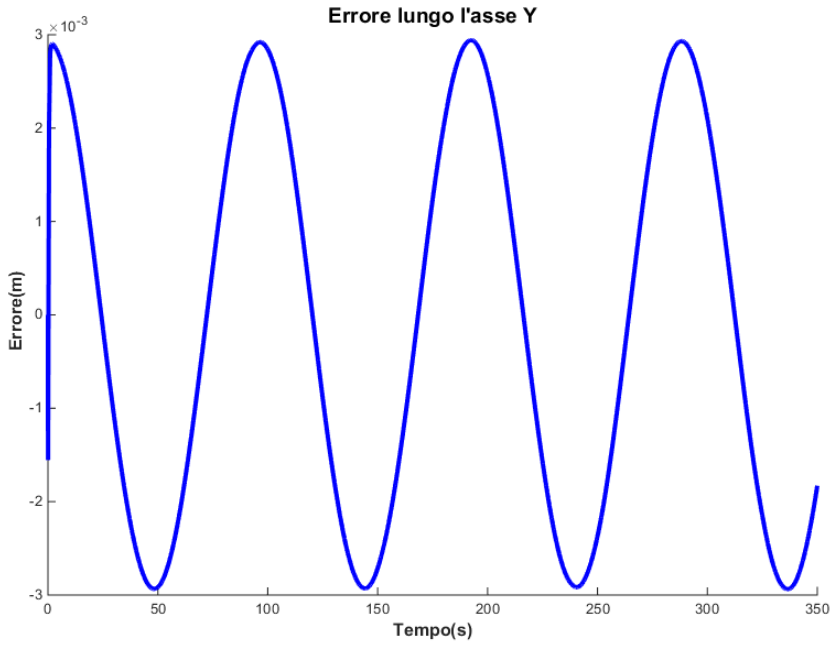


Figura 6.9.: *"Errore nell'inseguimento della traiettoria lungo l'asse Y"*

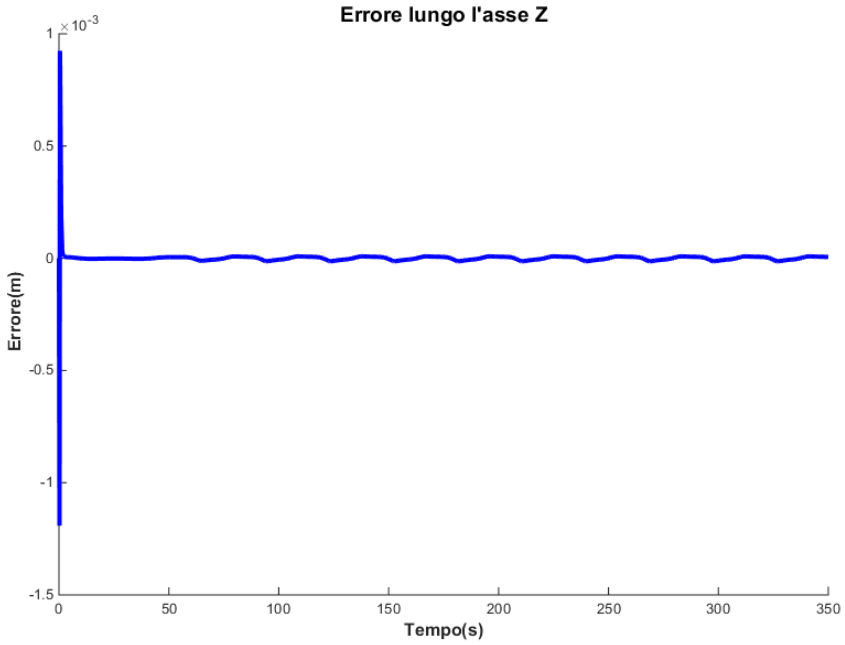


Figura 6.10.: *"Errore nell'inseguimento della traiettoria lungo l'asse Z"*

Le coppie esercitate da ogni singolo giunto, invece, risultano essere:

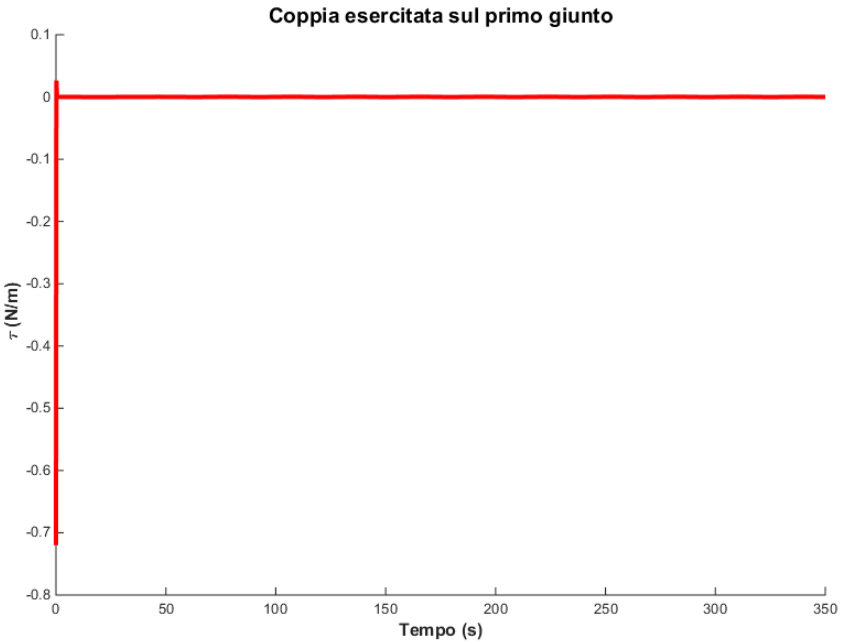


Figura 6.11.: *"Coppia esercitata dal primo giunto"*

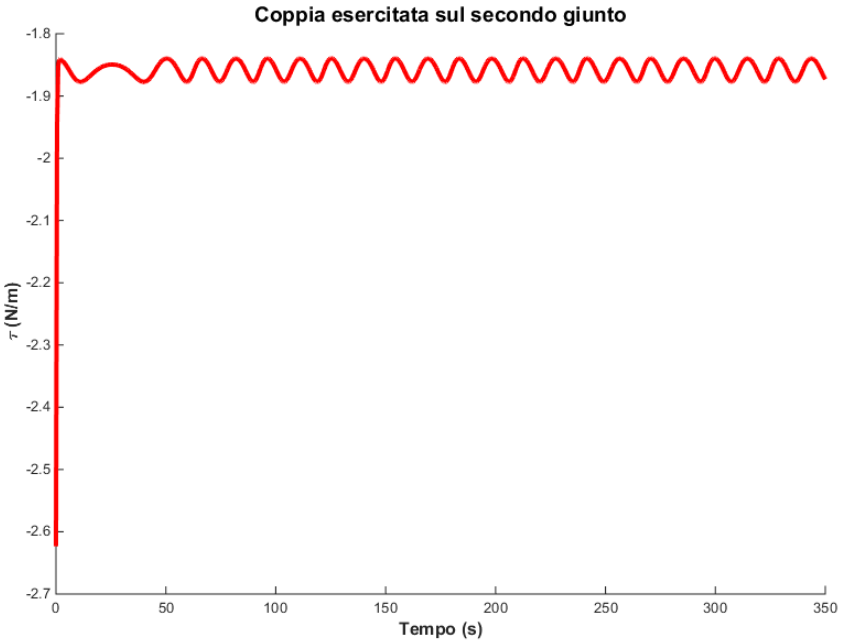


Figura 6.12.: *"Coppia esercitata dal secondo giunto"*

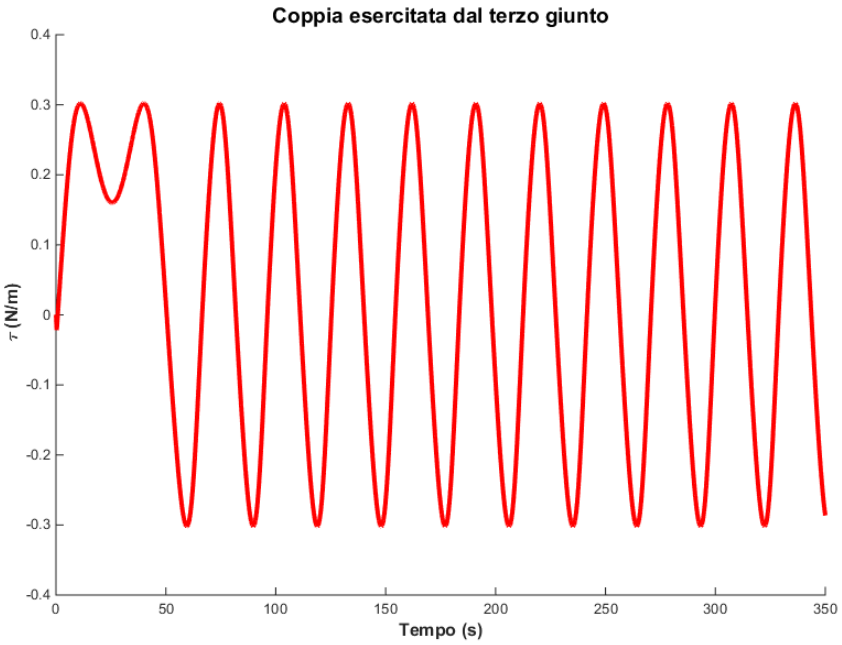


Figura 6.13.: *"Coppia esercitata dal terzo giunto"*

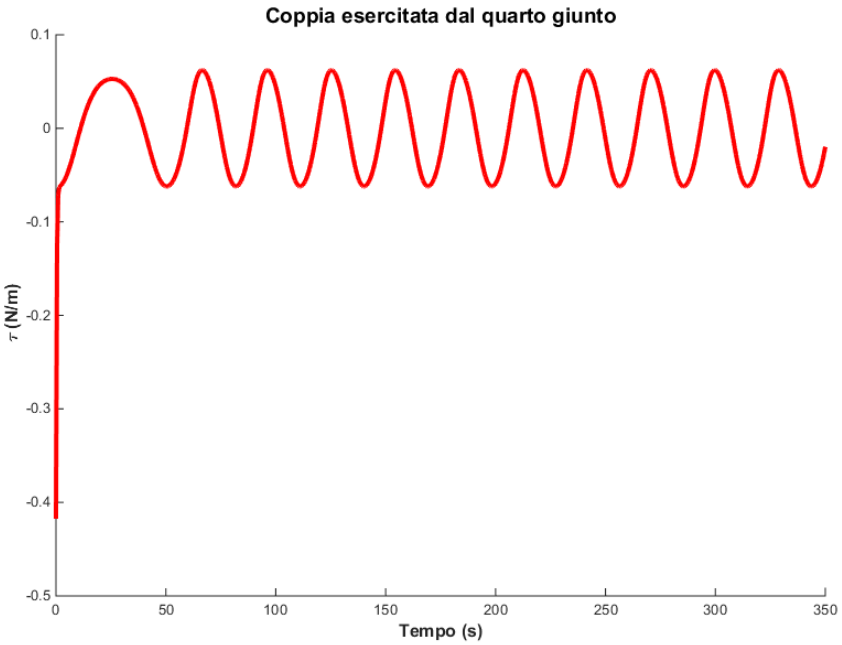


Figura 6.14.: *"Coppia esercitata dal quarto giunto"*

Alcune prove di simulazione sono state effettuate variando la quota in cui si ha l'inseguimento della traiettoria circolare nel piano x-y; in questo caso si riportano i grafici dei risultati ottenuti ad un'altezza pari a 0.169 m e in questo il punto di partenza della traiettoria di riferimento e del braccio risultano essere volutamente diverse:

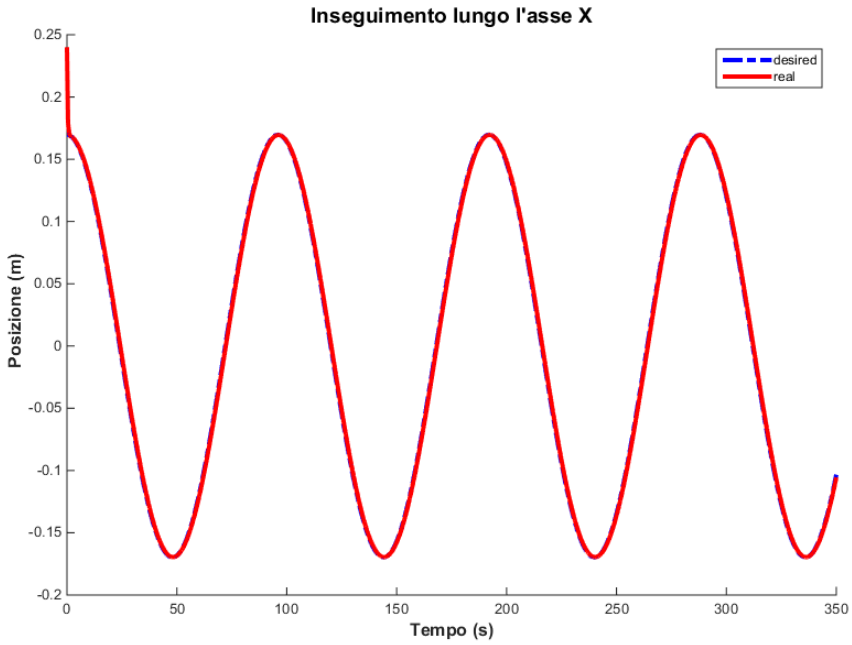


Figura 6.15.: *"Inseguimento della traiettoria lungo l'asse X ad una quota pari a 0.169 metri"*

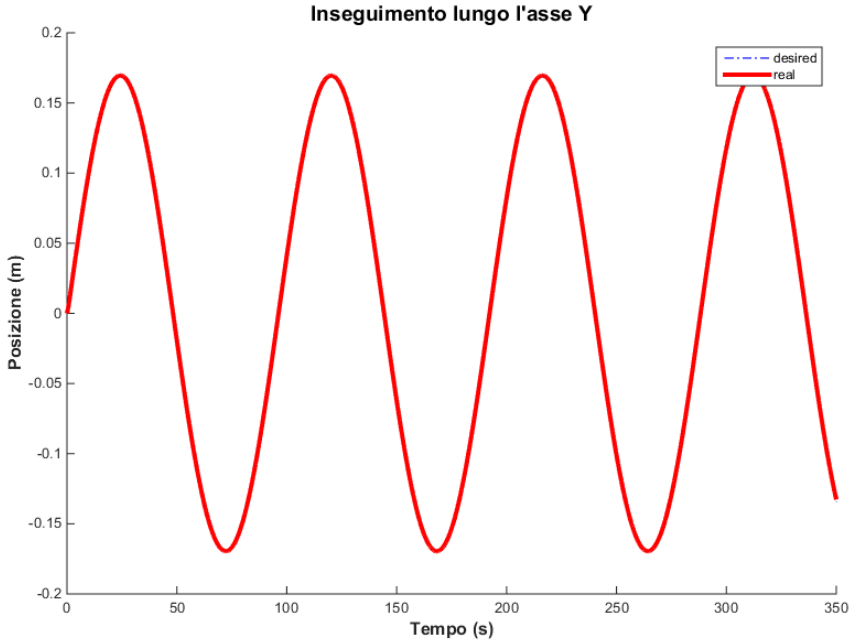


Figura 6.16.: *"Inseguimento della traiettoria lungo l'asse Y ad una quota pari a 0.169 metri"*

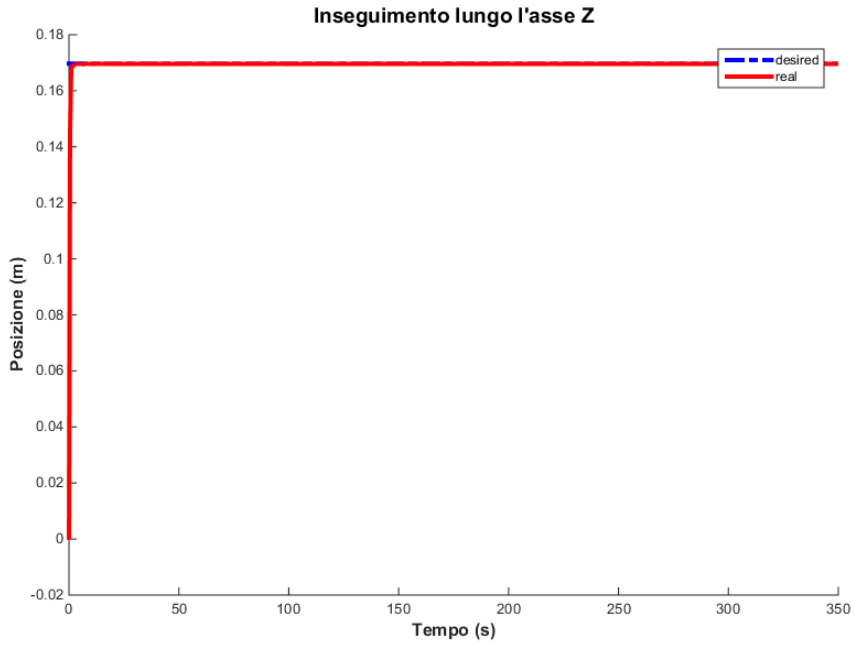


Figura 6.17.: *"Inseguimento della traiettoria lungo l'asse Z, il riferimento è a quota 0.169 metri"*

La traiettoria nello spazio tridimensionale risulta essere:

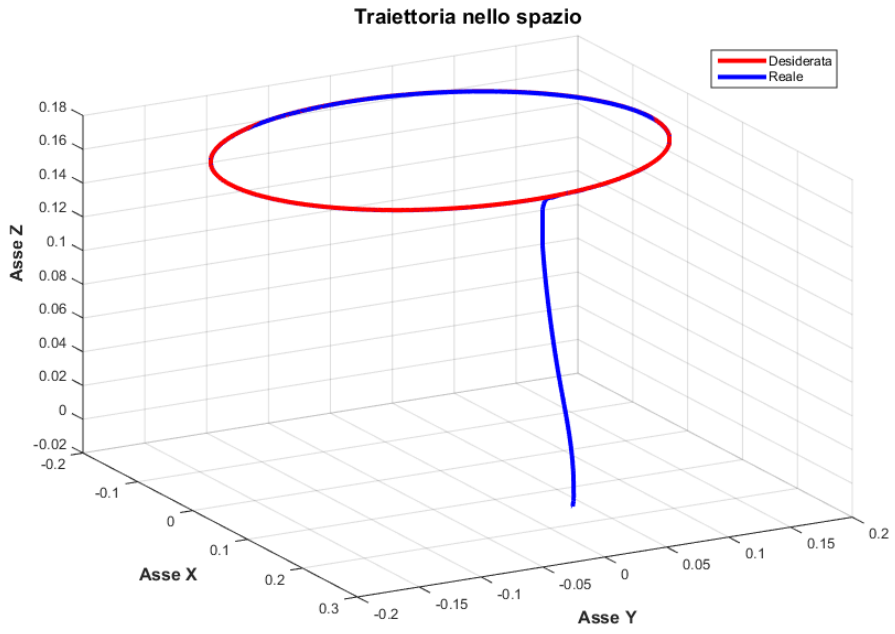


Figura 6.18.: *"Traiettoria nello spazio"*

Come si può osservare anche in questo caso si ottengono buoni risultati nell'inseguimento della traiettoria, con errori molto piccoli che risultano essere:

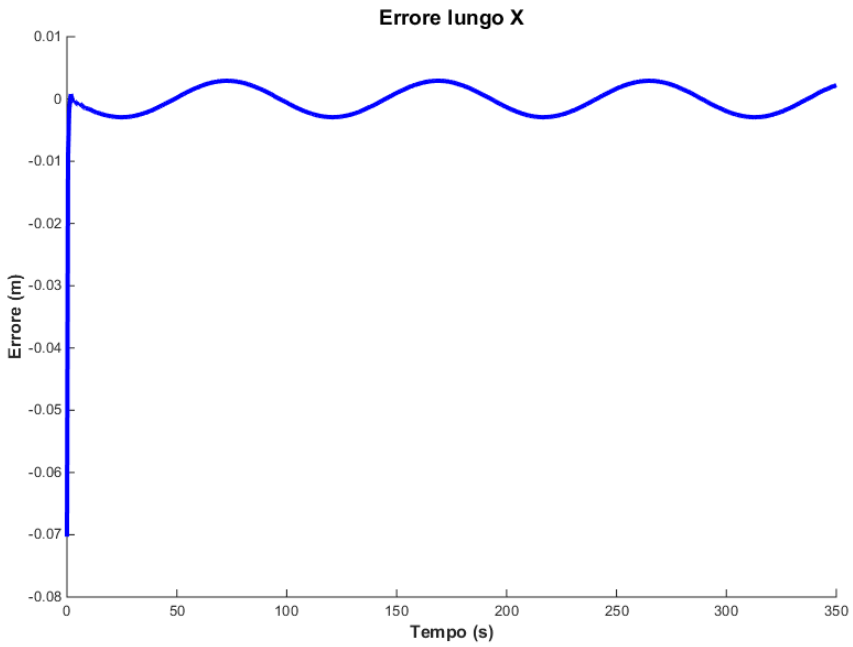


Figura 6.19.: *"Errore nell'inseguimento della traiettoria lungo l'asse X"*

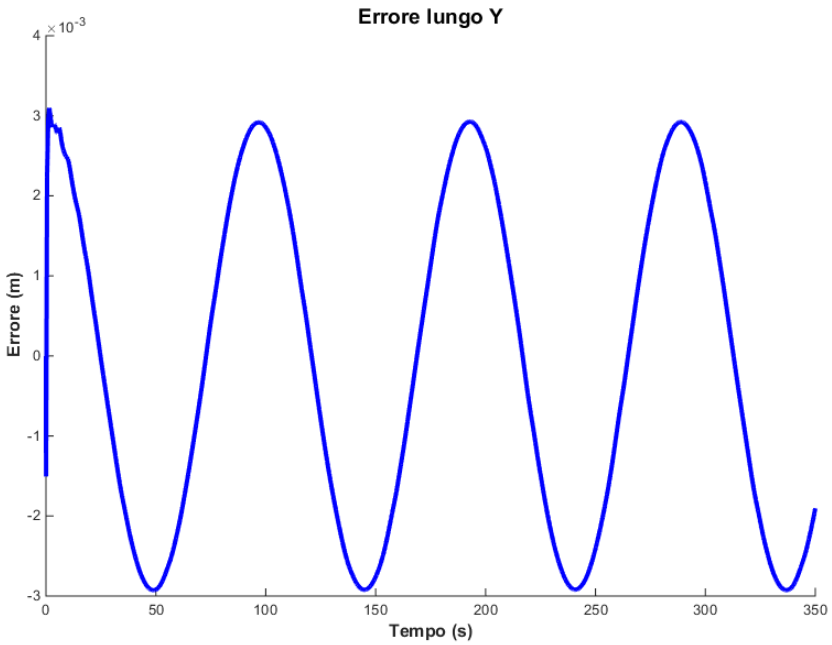


Figura 6.20.: *"Errore nell'inseguimento della traiettoria lungo l'asse Y"*

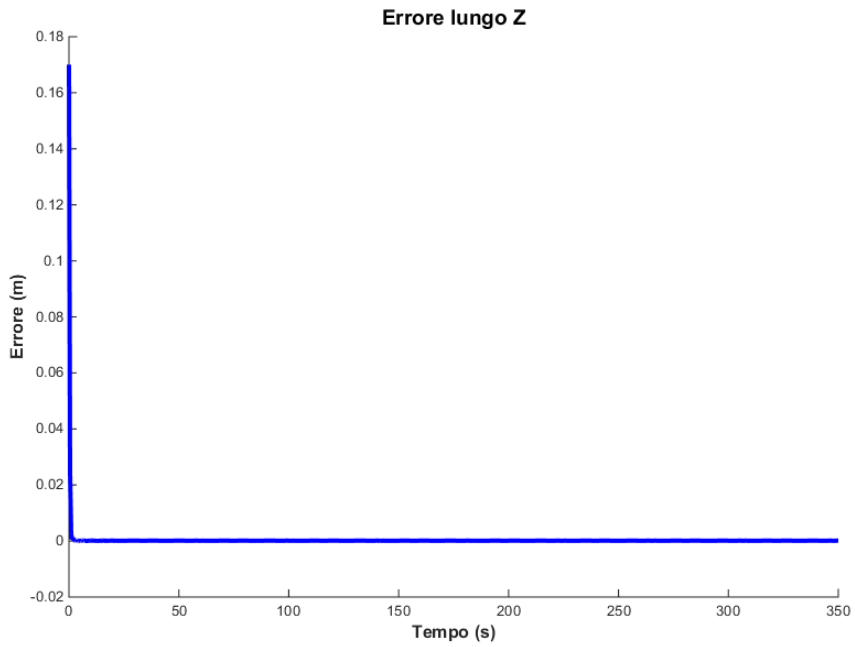


Figura 6.21.: *"Errore nell'inseguimento della traiettoria lungo l'asse Z"*

Le coppie esercitate da ogni giunto risultano essere:

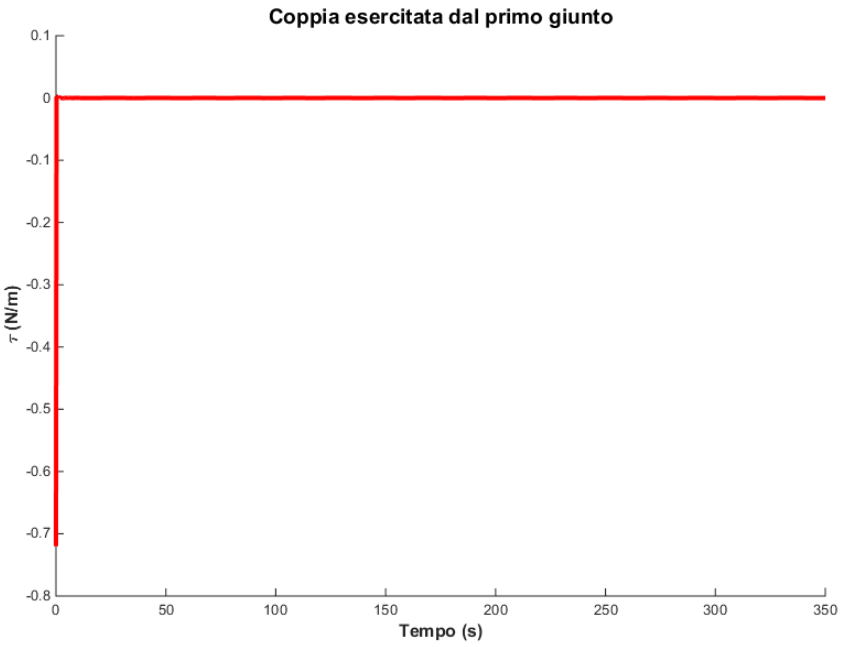


Figura 6.22.: *"Coppia esercitata dal primo giunto"*

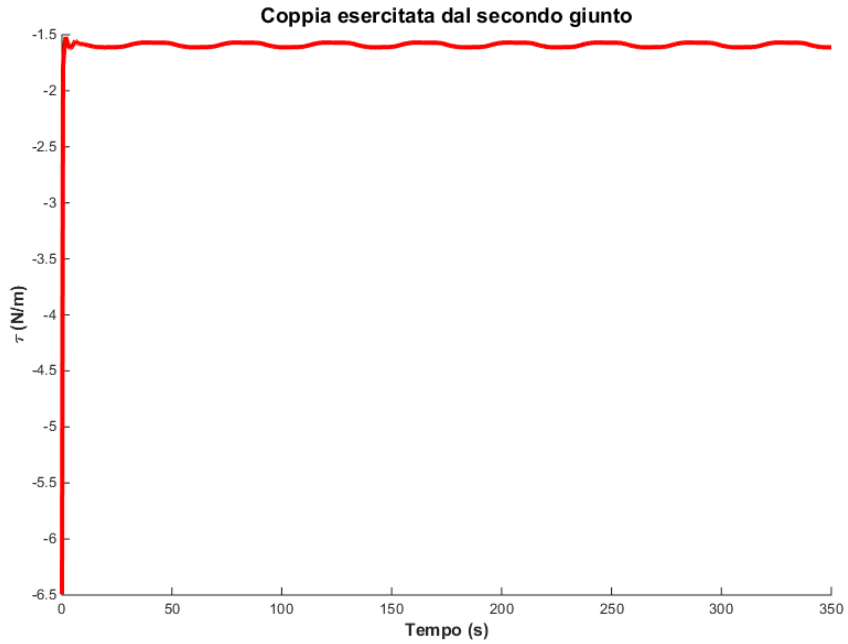


Figura 6.23.: *"Coppia esercitata dal secondo giunto"*

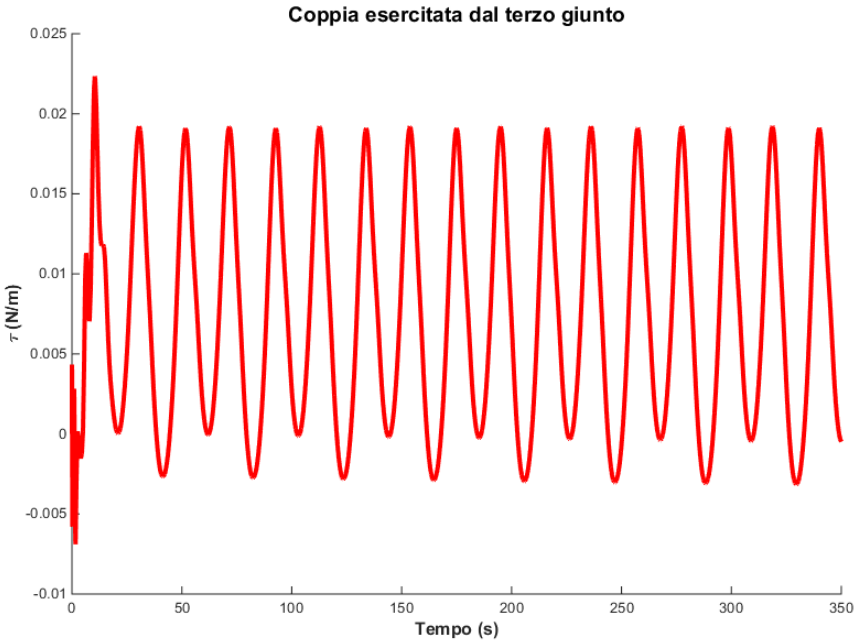


Figura 6.24.: *"Coppia esercitata dal terzo giunto"*

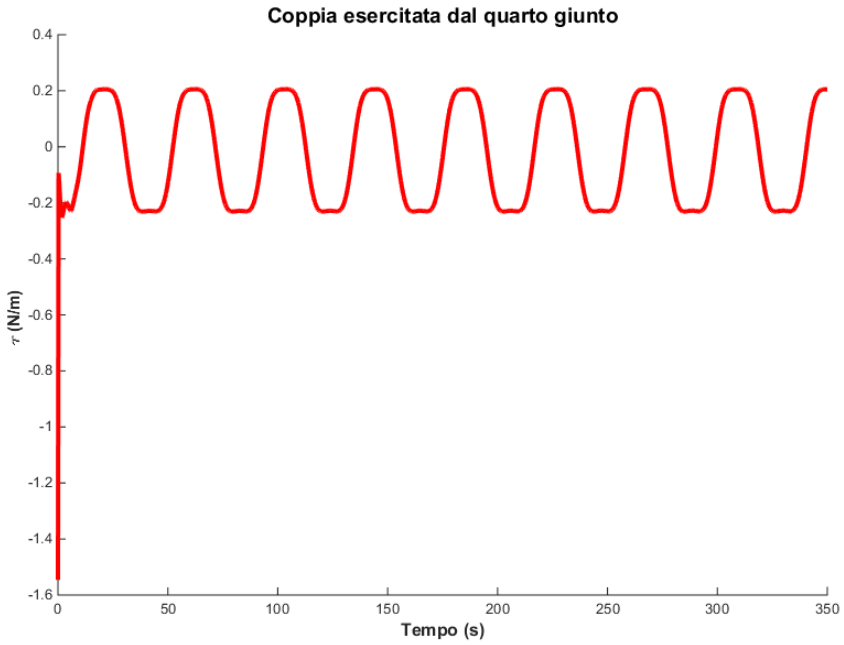


Figura 6.25.: *"Coppia esercitata dal quarto giunto"*

6.2 Prove reali

Infine, si riportano anche alcuni grafici ricavati da acquisizioni durante prove reali. In questo caso si riportano i grafici relati all'inseguimento di una traiettoria ad una quota di 0.169 metri, tuttavia in questo caso la traiettoria data come riferimento nel piano x-y non può essere una traiettoria circolare, in quanto il dispositivo non è in grado di effettuare un giro completo, ma partendo da quello che è il suo zero può effettuare rotazioni per un'ampiezza massima compresa tra $-2\pi/3$ e $2\pi/3$.
Dalle immagini riportate si può notare come nonostante la quota desiderata venga raggiunta con un errore dell'ordine del millimetro, nell'inseguimento della traiettoria nel piano x-y si ha invece un errore dell'ordine del centimetro, in quanto un errore di pochi gradi sull'angolo di giunto si traduce in un errore dell'ordine del centimetro nel piano cartesiano. In particolare, nelle immagini relative agli errori possiamo osservare come superato il transitorio l'errore lungo l'asse z si assesti nell'intorno dell'ordine del millimetro, mentre lungo gli assi x e y è dell'ordine del centimetro. Tali errori son principalmente dovuti ad una quantizzazione dell'angolo che riceviamo in ingresso e che si utilizza per la successiva elaborazione del controllo e ad una quantizzazione dell'angolo in uscita, ovvero di quello ottenuto in seguito all'elaborazione del controllo comportando così una inevitabile perdita di precisione. Oltre a questo un'altra possibile fonte di errore deriva da una non precisa modellizzazione dell'inerzia del sistema. Vediamo i grafici relativi all'inseguimento lungo i tre assi:

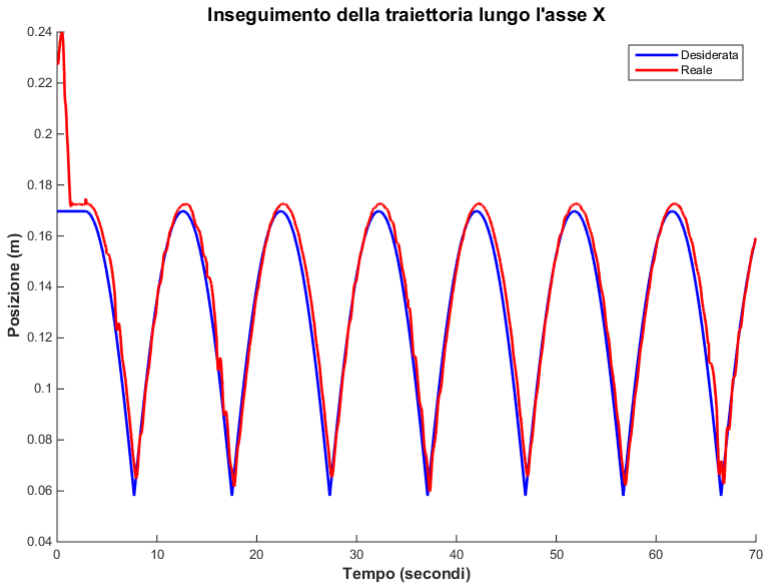


Figura 6.26.: *"Inseguimento reale della traiettoria lungo l'asse X ad una quota pari a 0.169 metri"*

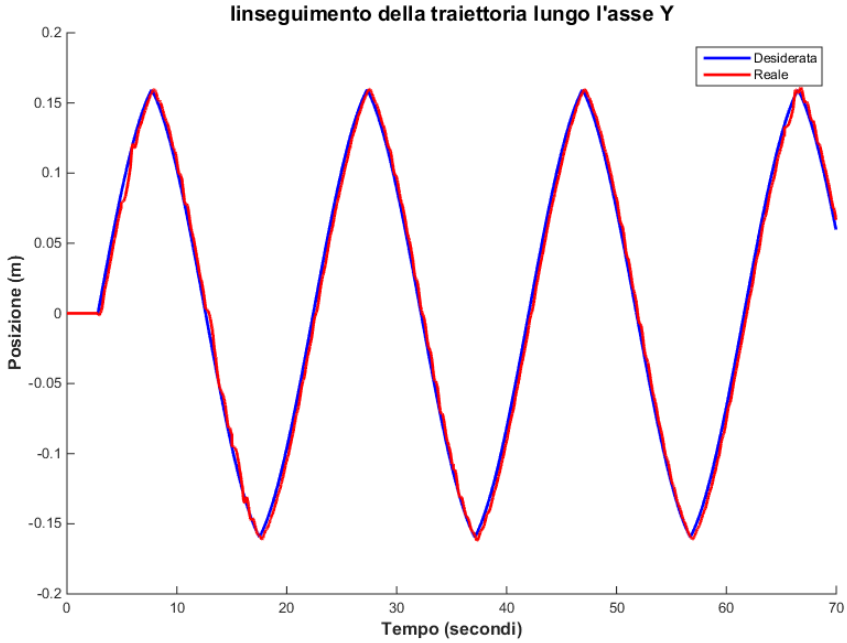


Figura 6.27.: *"Inseguimento reale della traiettoria lungo l'asse Y ad una quota pari a 0.169 metri"*

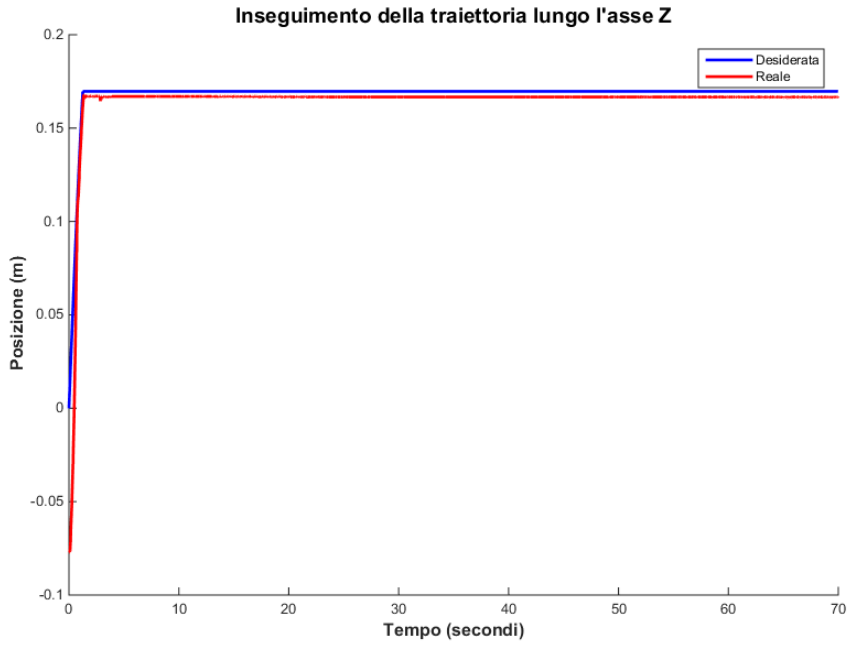


Figura 6.28.: *"Inseguimento reale della traiettoria lungo l'asse Z, il riferimento è a quota 0.169 metri"*

Vediamo quindi, i grafici relativi agli errori:

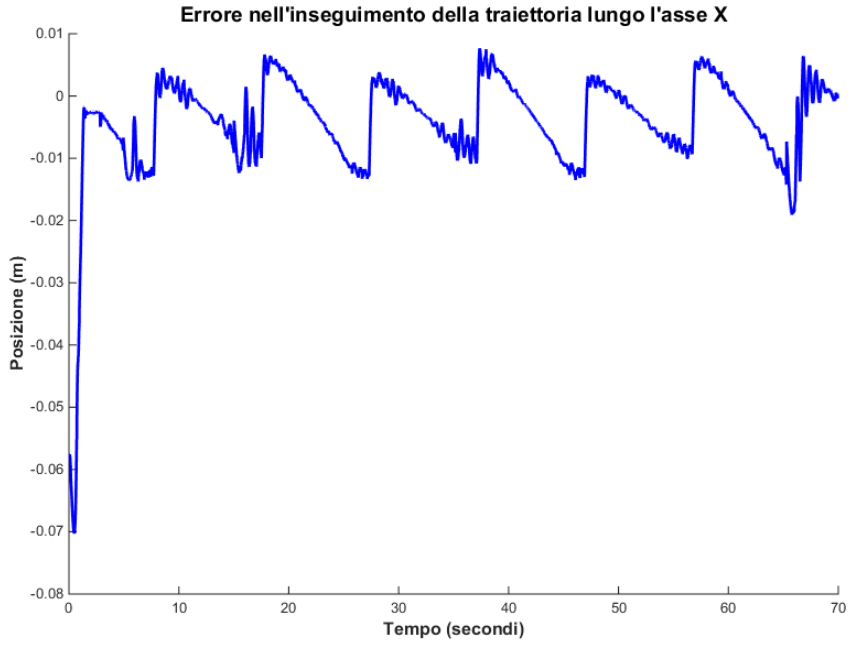


Figura 6.29.: *"Errore nell'inseguimento della traiettoria lungo l'asse X"*

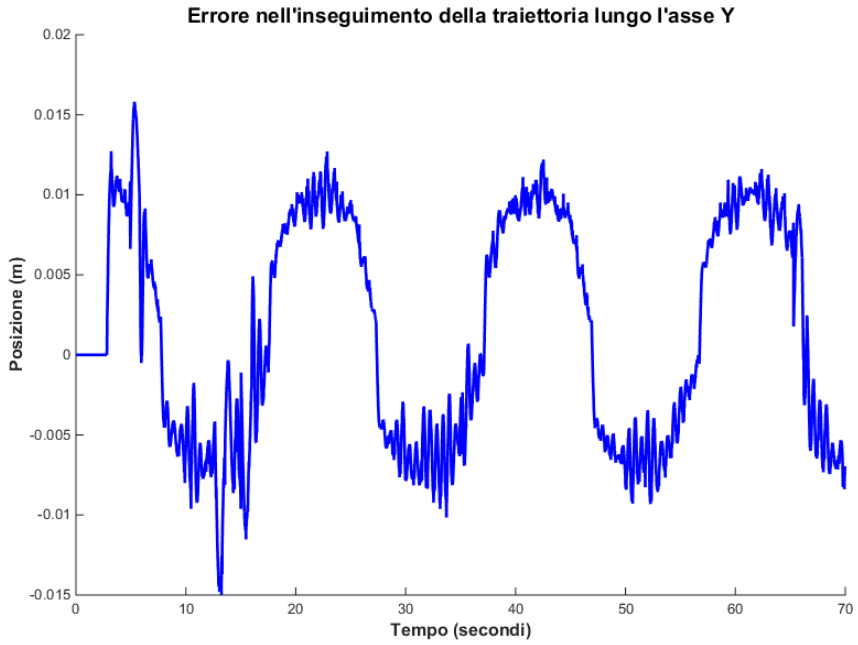


Figura 6.30.: *"Errore nell'inseguimento della traiettoria lungo l'asse Y"*

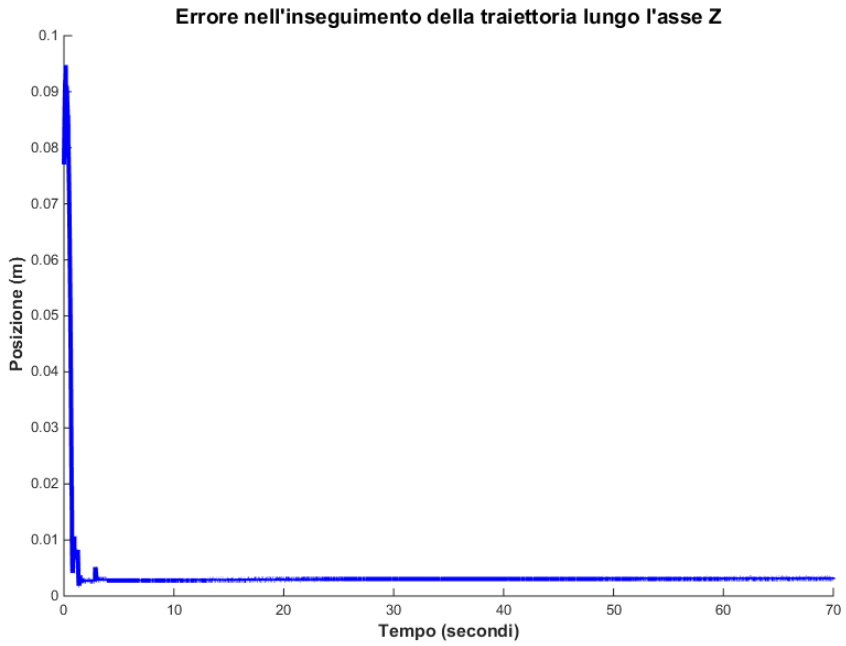


Figura 6.31.: *"Errore nell'inseguimento della traiettoria lungo l'asse Z"*

Infine riportiamo anche in questo caso i grafici relativi alle coppie esercitate da ogni singolo giunto:

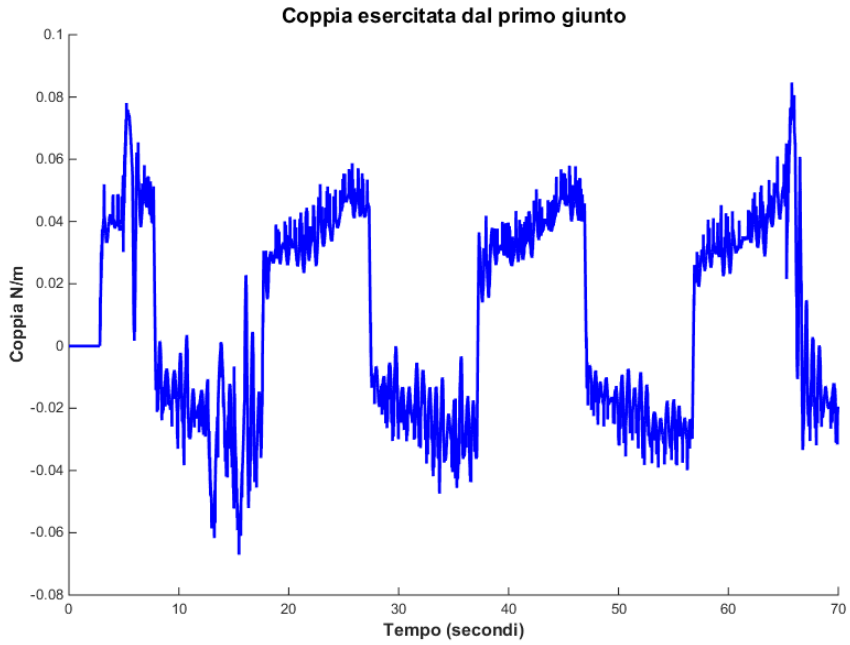


Figura 6.32.: *"Coppia esercitata dal primo giunto"*

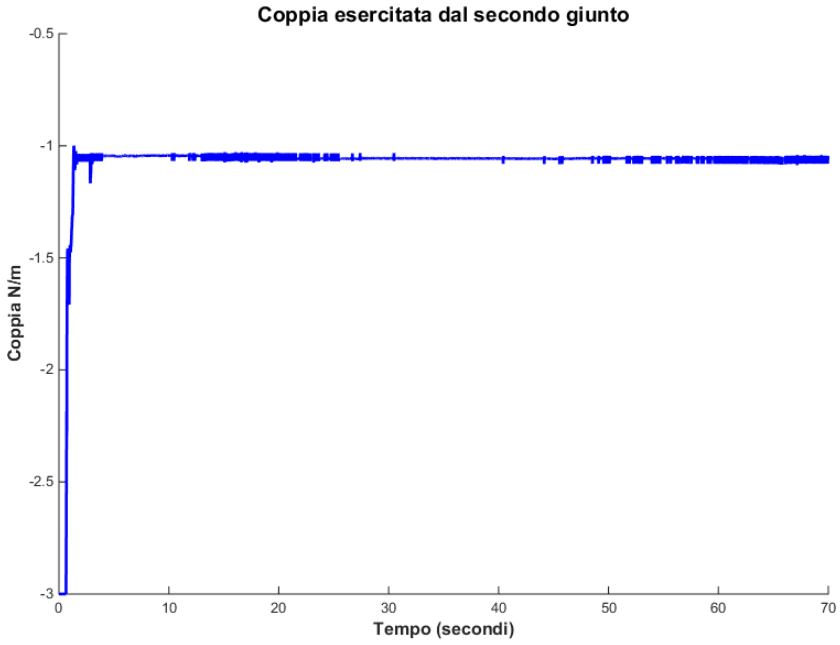


Figura 6.33.: *"Coppia esercitata dal secondo giunto"*

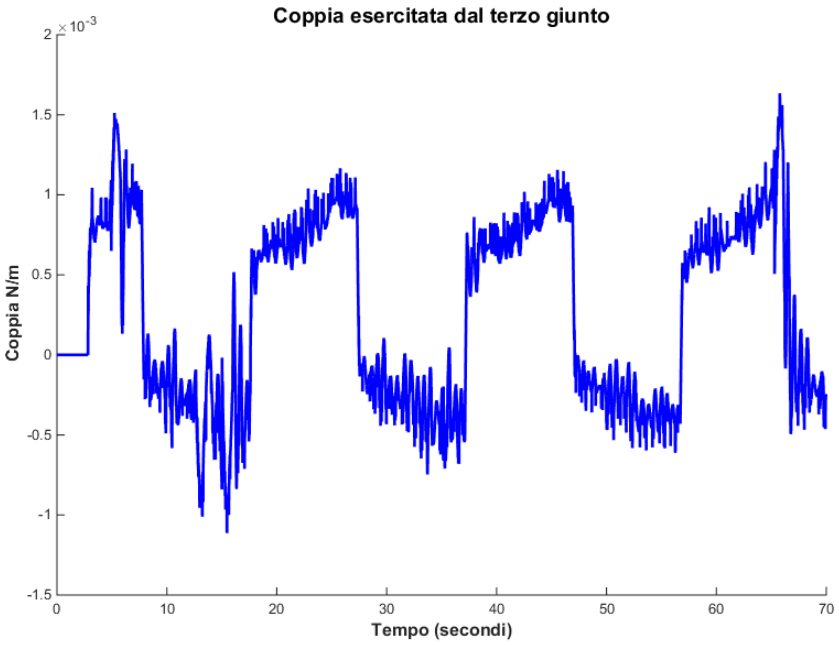


Figura 6.34.: *"Coppia esercitata dal terzo giunto"*

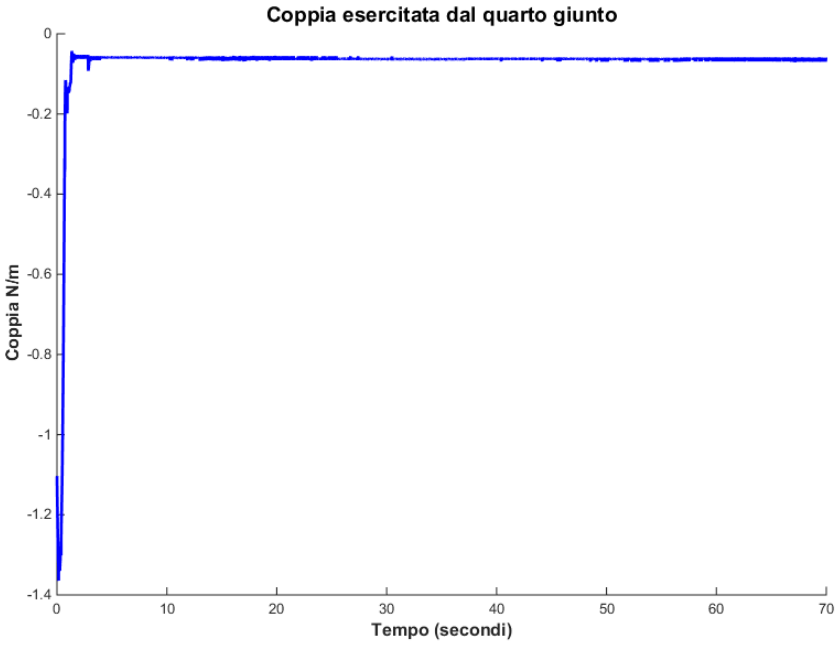


Figura 6.35.: *"Coppia esercitata dal quarto giunto"*

Infine, si riportano i grafici relativi al caso in cui oltre alla traiettoria eseguita nei casi precedenti si insegue una traiettoria sinusoidale anche lungo l'asse z, ricavando le stesse conclusioni dei casi sopra:

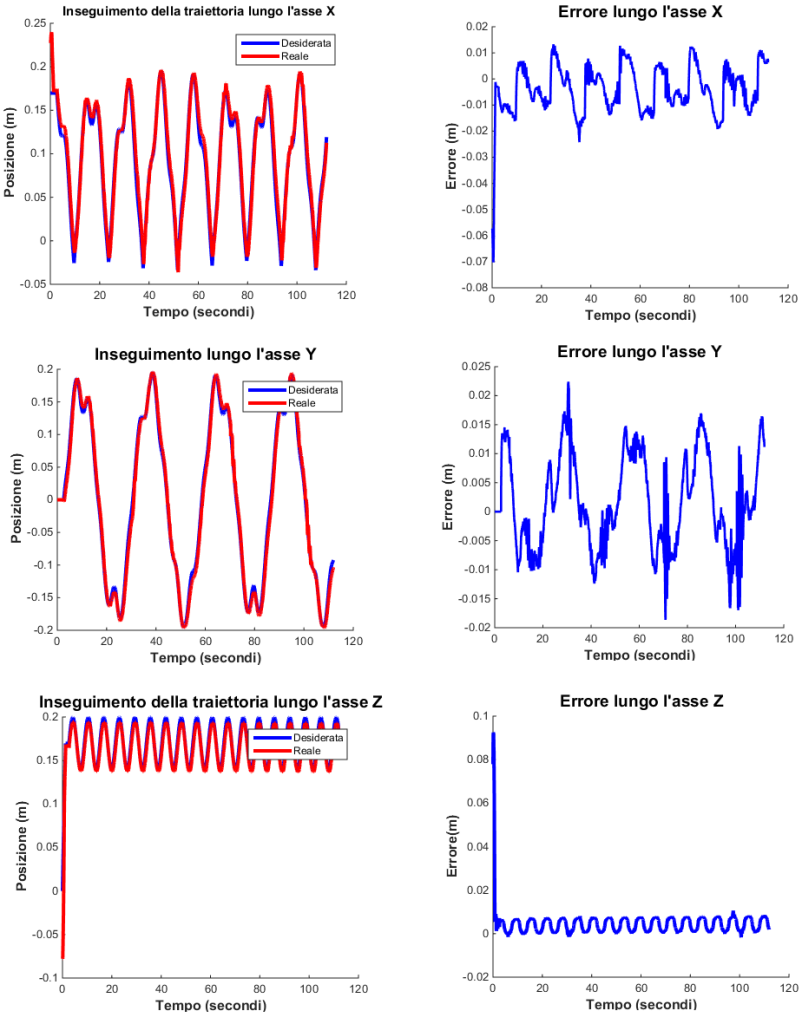


Figura 6.36.: *"Traiettorie e relativi errori lungo i tre assi"*

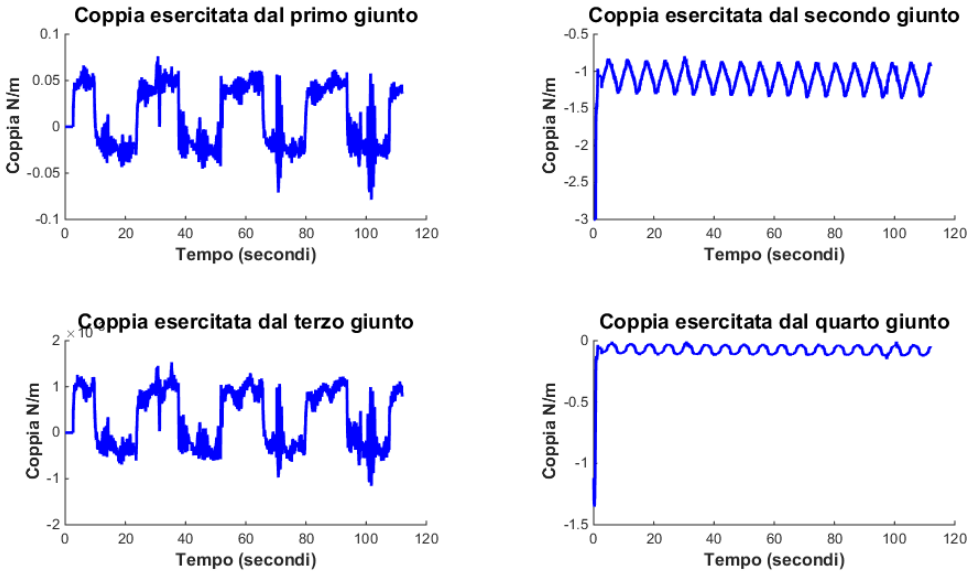


Figura 6.37.: *"Coppie esercitate sui quattro giunti"*

Capitolo 7

Conclusioni

Obiettivo del lavoro di tesi era quello di realizzare un controllore per un braccio a cedevolezza variabile con codice generato automaticamente da uno schema realizzato attraverso un approccio model based; questo è stato realizzato utilizzando il tool E4Coder.
Il vantaggio principale nell'utilizzare il tool E4Coder e il suo generatore di codice E4CoderCG, consistono nella possibilità di poter introdurre nuove board per cui si vuole generare codice e la possibilità di realizzare il modello non solo utilizzando i blocchi messi a disposizione dal tool ma aggiungendo e modellando ulteriori blocchi in base alle necessità dello sviluppatore, grazie alla presenza dei blocchi custom.
Nell'utilizzo di tale tool si può notare la comodità e l'efficienza dell'approccio "model-based", questo, infatti, consente a chi sviluppa il modello di concentrarsi più sulle funzionalità del sistema e non sulle funzioni da implementare, fornendo, inoltre, una visione più completa e generale dell'intero sistema. Questo tipo di approccio consente la realizzazione di un sistema che ci permette di interagire direttamente con il dispositivo, ma allo stesso tempo di visualizzarne e analizzarne il comportamento in fase di simulazione.
L'intero progetto si basa, quindi, sullo sviluppo e sulla progettazione dello schema del modello da controllare, partendo dall'inserimento di una nuova board, la UdooNeo, tra le possibili board per cui il tool supporta la generazione di codice e proseguendo con la creazione di alcuni blocchi realizzati appunto utilizzando i blocchi custom messi a disposizione dal tool; tali blocchi sono stati utilizzati per modellare la comunicazione tra il dispositivo e la board e per modellare il comportamento dei dispositivi qbmove utilizzati. In particolare il comportamento dei dispositivi viene modellato da due blocchi, uno per la parte di attuazione e l'altro per la parte di lettura dai sensori, che in base ad una opportuna scelta dei parametri possono essere utilizzati per ottenere le varie funzioni che possono essere eseguite da tali dispositivi. Allo stesso modo i blocchi custom sono stati utilizzati per l'implementazione dell'algoritmo di controllo e del modello, in questo modo modificando i file "linkati", o sostituendo tali blocchi con opportuni blocchi modellati è possibile riadattare lo stesso schema per la generazione di codice per il controllo di un braccio con un diverso algoritmo per il controllore o con una diversa configurazione dei giunti.
Una delle caratteristica principali della board UdooNeo è la presenza di un chip con due processori, un cortex-M4 su cui si trova un sistema operativo tempo reale e un cortex A9 su cui si trova un sistema general-purpose (LINUX), quindi, uno dei possibili sviluppi futuri può essere la generazione di codice per il sistema operativo tempo reale in modo da migliorare le prestazioni del controllo e utilizzare il processore su cui esegue il sistema operativo genaral pursuit per altre applicazioni come ad esempio realizzare grafici o per la realizzazione di un'interfaccia grafica per la gestione dell'applicazione, sfruttando in questo modo il potenziale offerto da una board Multi-OS.

Appendice A

Codice sorgente

"msg_rx.h"

```
#ifndef MSG_RX_H
#define MSG_RX_H

#include "data.h"

void msg_rx_OutputUpdate( msg_rx_Param *Param, real *In1, real *Out1, real *Out2);
void msg_rx_StateUpdate( msg_rx_Param *Param, real *In1, real *Out1, real *Out2);
void msg_rx_Init( msg_rx_Param *Param, real *In1, real *Out1, real *Out2);
void msg_rx_Terminate( msg_rx_Param *Param, real *In1, real *Out1, real *Out2);

#endif
```

"msg_rx.c"

```
#include "time.h"
#include <sys/time.h>
#include <pthread.h>
#include "msg_rx.h"
#include </home/udooer/Code_gen/provatot/linux-lib/mylib_qb/qbmove_com.h>
#include </home/udooer/Code_gen/provatot/linux-lib/mylib_qb/Gestori_Read_Write.h>


//real pck_rd[BUFFER_SIZE];

package_read pck_to_read;


void msg_rx_Init(msg_rx_Param *Param, real *In1, real *Out1, real *Out2){

        uint32_t ret;
        uint32_t baudrate = Param->BAUDRATE;    //il Baudrate mi viene dai parametri di ingresso al blocco
        baudrate=B460800;
        static int count_ser=0;
        static int ID_mem=0;

        if(count_ser==0){
                ID_mem=Param->ID;
        }
        //pck_to_read.id=Param->ID;
        pck_to_read.id=0;
        pck_to_read.command=0;
        pck_to_read.dim=0;

//----------------------------Apertura della porta seriale--------------------------//

        if(ID_mem==Param->ID && count_ser==0){
                //il Baudrate mi viene dai parametri di ingresso al blocco
                ret = open_serial((int) baudrate);
                if(ret==0){
                        printf("\n porta aperta!!\n ");
                }else{
                        printf("\n porta non aperta!!\n ");
                }
        }
        count_ser++;

//-----------------------------------------------------------------------------------//

//----------Creo il thread per la gestione della lettura scrittura-------------------//

        ret = create_thread_Read(Param->ID, Param->Command);
        if(ret==0){
                printf("Thread lettura creato!!\n");
        }else{
                if(ret>0){
                        printf("\nThread lettura gi  creato!!\n");
                }else{
                        printf("Thread lettura non creato!!\n");
                }
        }

//-----------------------------------------------------------------------------------//

}

void msg_rx_Terminate( msg_rx_Param *Param, real *In1, real *Out1, real *Out2){

        Close_Serial();
}

void msg_rx_StateUpdate( msg_rx_Param *Param, real *In1, real *Out1, real *Out2){
```

```
}

void msg_rx_OutputUpdate( msg_rx_Param *Param, real *In1, real *Out1,  real *Out2){

        real dim;
        uint32_t id=Param->ID;
        char *pun_aux;
        int32_t j;
        int32_t ret;

        //printf("\nSono in lettura!!\n");

        pck_to_read.id=Param->ID;
        pck_to_read.command=Param->Command;

        ret=(int32_t)extract_New_element_read(Param->ID,Param->Command ,&pck_to_read);
        if(ret==0){
                dim=(real)pck_to_read.dim;
                //printf("Dimensione di pck_rd %d \n",pck_to_read.dim);
                for(j=0;j<dim;j++){

                        Out1[j]=(real)pck_to_read.buffer_read[j];
                        //printf("Valori in Out1: %f \n",Out1[j]);
                }
                printf("\n QB %d, CMD %d ho letto pacchetto dati di dim %f!!\n", Param->ID, Param->Command, dim);
        }else{

                dim=0;
                //printf("QB %d,  CMD %d dati non letti!! \n",Param->ID, Param->Command);
        }

        *Out2=dim;
}
```

———————"qbmove_rx.h"———————

```
#ifndef QBMOVE_RX_H
#define QBMOVE_RX_H

#include "data.h"


void qbmove_rx_OutputUpdate( qbmove_rx_Param *Param, real *In1, real *In2, real *Out1, real *Out2);
void qbmove_rx_StateUpdate( qbmove_rx_Param *Param, real *In1, real *In2, real *Out1, real *Out2);
void qbmove_rx_Init( qbmove_rx_Param *Param, real *In1, real *In2, real *Out1, real *Out2);
void qbmove_rx_Terminate( qbmove_rx_Param *Param, real *In1, real *In2, real *Out1, real *Out2);

#endif
```

———————"qbmove_rx.c"———————

```
#include "qbmove_rx.h"
#include </home/udooer/Code_gen/provatot/linux-lib/mylib_qb/qb_select.h>
#include </home/udooer/Code_gen/provatot/linux-lib/mylib_qb/qb.h>
#include </home/udooer/Code_gen/provatot/linux-lib/mylib_qb/Gestori_Read_Write.h>

#define DIM_BUFFER 128
#define NUM_QB 4
#define NUM_CMD 4       //sono solo i comandi da cui mi aspetto una risposta
value_read value_rx[NUM_QB];


void qbmove_rx_Init(qbmove_rx_Param *Param, real *In1, real *In2, real *Out1, real *Out2){

        uint32_t l;
        static int c=0;
        if(c==0){
                for(l=0;l<NUM_CMD;l++){
                        value_rx[l].Id=0;
                        value_rx[l].dim=0;
                }
        }

        c++;

        for(l=0;l<NUM_CMD;l++){
                if(value_rx[l].Id==0){
                        value_rx[l].Id=Param->ID;
                        value_rx[l].CMD=Param->Command;
                        //printf("Indice l: %d value_rx.Id %d, value_rx.CMD: %d!!\n",l,
                        value_rx[l].Id,value_rx[l].CMD);
                        break;
                }

        }

        printf("Entro dentro la qbmove_rx_Init, ma non devo fare niente!!\n");
}

void qbmove_rx_Terminate(qbmove_rx_Param *Param, real *In1, real *In2, real *Out1, real *Out2){
```

```
}

void qbmove_rx_StateUpdate(qbmove_rx_Param *Param, real *In1, real *In2, real *Out1, real *Out2){

}

void qbmove_rx_OutputUpdate(qbmove_rx_Param *Param, real *In1, real *In2, real *Out1, real *Out2){

        uint32_t id= Param->ID;
        uint32_t package_int[25];
        real dim_package= *In2;
        uint32_t dim_pkcread;
        uint32_t l;
        int32_t pv_read[15];


        //printf("valore dimensione pck letto: %f\n",dim_package);
        if(dim_package!=0){
                //printf("valore dimensione pck letto: %f\n",dim_package);
                for(l=0;l<dim_package;l++){
                        package_int[l] = (uint32_t)In1[l];
                        //printf("valore contenuto in package %d\n", package_int[l]);
                }

                //printf("QB %d, CMD %d valore in dim_package %f\n",  id, Param->Command, dim_package);
                Select_commGet(id, Param->Command, (char*)package_int, dim_package, pv_read, &dim_pkcread);

                                for(l=0;l<dim_pkcread;l++){

                                        *(Out1+l)=(real)pv_read[l];
                                        printf("Valori pv_read %d!!\n",pv_read[l]);
                                }
                                *Out2=(real)dim_pkcread;
                                *(Out2+1)=Param->POS;
        }else{

                //printf("\n QB %d, CMD %d dati non aggiornati!!\n", Param->ID, Param->Command);
                *Out2=-1;
                *(Out2+1)=Param->POS;
        }
}
```

———————"control.h"———————

```
#ifndef CONTROL_H
#define CONTROL_H

#include "data.h"


void control_OutputUpdate(control_Param *Param, real *In1, real *Out1);
void control_StateUpdate(control_Param *Param, real *In1, real  *Out1);
void control_Init(control_Param *Param, real *In1, real *Out1) ;
void control_Terminate(control_Param *Param, real *In1, real *Out1);

#endif
```

———————"control.c"———————

```
#include "control.h"
#include </home/udooer/Code_gen/provatot/linux-lib/mylib_qb/Control_alg2.h>
#include </home/udooer/Code_gen/provatot/linux-lib/mylib_qb/Gestori_Read_Write.h>


#define NUM_QB 4
#define NUM_OF_SENSORS 3
#define BUFFER_SIZE 32
#define SINGLE_BUFFERIN 12

real old_time;
real pos_q[NUM_QB][NUM_OF_SENSORS];
real value_control1[BUFFER_SIZE];
real value_control2[BUFFER_SIZE];

extern value_read value_rx[NUM_QB*2];

void control_Init(control_Param *Param, real *In1, real *Out1){

}

void control_Terminate(control_Param *Param, real *In1, real *Out1){


}

void control_StateUpdate(control_Param *Param, real *In1, real *Out1){


}

void control_OutputUpdate(control_Param *Param, real *In1, real *Out1){

        real tmp=0;
        real *value_p;
```

```
        real coef_kp[3];
        real coef_kd[3];
        uint32_t l,k,j,x;
        real dim_p=0;
        real dim_v=0;
        uint32_t pos_cat;
        uint32_t ord[NUM_QB];
        int32_t agg_v[NUM_QB]={0,0,0,0};
        int32_t val_ap[NUM_QB];

        value_p=In1;

        coef_kp[0]=Param->kp1;
        coef_kp[1]=Param->kp2;
        coef_kp[2]=Param->kp3;
        coef_kd[0]=Param->kd1;
        coef_kd[1]=Param->kd2;
        coef_kd[2]=Param->kd3;

        for(l=0;l<NUM_QB;l++){
                j=l*SINGLE_BUFFERIN;
                dim_p=*(value_p+10+j);
                pos_cat=*(value_p+11+j);
                if(dim_p!=-1){
                        for(k=0;k<dim_p;k++){
                                pos_q[pos_cat-1][k]=*(value_p+k+j);
                        }
                        agg_v[pos_cat-1]=1;

                }else{
                        for(k=0;k<NUM_OF_SENSORS;k++){
                                if(pos_q[pos_cat-1][k]==0){
                                        agg_v[pos_cat-1]=agg_v[pos_cat-1]-1;
                                }
                        }

                }
                ord[l]=pos_cat;

        }

        for(l=0;l<NUM_QB;l++){
                printf("\nValori di posizione ricevuti: %f %f %f", pos_q[l][0],pos_q[l][1], pos_q[l][2]);
                //printf("\nValori di vel ricevuti: %f %f %f", vel_q[l][0],vel_q[l][1], vel_q[l][2]);
                printf("\nOrdine di ricezione delle posizioni: %d", ord[l]);
        }


        tmp = old_time + 0.005; //secondi

        control_qb(value_control1, value_control2, coef_kp, coef_kd, pos_q, NUM_QB,  NUM_OF_SENSORS, agg_v);
        //printf("\n Valori ottenuti da control_qb: %f %f %f\n", value_control[0], value_control[1],
        value_control[2]);

        old_time=tmp;

        for(l=0;l<NUM_QB;l++){
                pos_cat=ord[l];
                val_ap[l]=(int32_t)value_control1[pos_cat-1];
                Out1[l]=(real)val_ap[l];
                val_ap[l]=(int32_t)value_control2[pos_cat-1];
                Out1[l+NUM_QB]=(real)val_ap[l];
        }


}
```

"qbmove_com.h"

```
#ifndef QBMOVE_SERIALPORT_H_INCLUDED
#define QBMOVE_SERIALPORT_H_INCLUDED


#if (defined(_WIN32) || defined(_WIN64))
    #include <windows.h>
#else
    #define HANDLE  int
    #define INVALID_HANDLE_VALUE    -1
#endif

#if !(defined(_WIN32) || defined(_WIN64)) && !(defined(__APPLE__)) //only for linux
    #include <termios.h>
#endif
#include <pthread.h>


#include "commands.h"


//typedef struct comm_settings comm_settings;
//typedef struct package_received package_received;

#define BUFFER_SIZE 128

typedef struct{

        HANDLE file_handle;
```

```
        pthread_mutex_t mut_ser;
}comm_settings;


int RS485listPorts( char list_of_ports[10][255]);
int open_serial(int baudrate);
int RS485read(int ID, int *pck_l, int *dim);
int inner_read(int nb, unsigned char *data_in);
int RS485read_2( int ID, int *pck_l, int *dim);
void Write_RS485(char *buffer_info, int d_pck);
char checksum ( char * data_buffer, int data_length);
int insert_value(float *value_new, int num_value, int CMD);
long timevaldiff(struct timeval *starttime, struct timeval *finishtime);
void Close_Serial(void);



#endif
```

"qbmove_com.c"

```
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include <ctype.h>
#include <time.h>
#include <semaphore.h>

#if (defined(_WIN32) || defined(_WIN64))
    #include <windows.h>
#endif

#if !(defined(_WIN32) || defined(_WIN64))
    #include <unistd.h>  /* UNIX standard function definitions */
    #include <fcntl.h>   /* File control definitions */
    #include <errno.h>   /* Error number definitions */
    #include <termios.h> /* POSIX terminal control definitions */
    #include <sys/ioctl.h>
    #include <dirent.h>
    #include <sys/time.h>
    #include <stdlib.h>
#endif

#if !(defined(_WIN32) || defined(_WIN64)) && !(defined(__APPLE__))
    #include <linux/serial.h>
#endif


#include "qbmove_com.h"
#include "qb_select.h"
#include "Gestori_Read_Write.h"


#if (defined(_WIN32) || defined(_WIN64))
    // windows stuff

    #define usleep(X) Sleep((X) / 1000)
#elif (defined(__APPLE__))
    // apple stuff
#else
    // linux stuff
#endif



#define BUFFER_SIZE 128
#define NUM_OF_SENSORS 3
#define NUM_OF_MOTORS 2


comm_settings comm_settings_t;

short int inputs[NUM_OF_MOTORS];
short int measurements[NUM_OF_SENSORS];
short int measurements_offset[NUM_OF_SENSORS];
short int currents[NUM_OF_MOTORS];
short int info_type[NUM_OF_MOTORS];

void packet_to_send(uint32_t *buffer_info, package_to_send *p_send, int *get_reply);



int RS485listPorts( char list_of_ports[10][255] ){

    ////////////////////////////////   WINDOWS   ////////////////////////////////
#if (defined(_WIN32) || defined(_WIN64))

    HANDLE port;
    int i, h;
    char aux_string[255];

    h = 0;

    for(i = 1; i < 10; ++i) {
        strcpy(list_of_ports[i], "");
        sprintf(aux_string, "COM%d", i);
        port = CreateFile(aux_string, GENERIC_WRITE|GENERIC_READ,
```

```
                0, NULL, OPEN_EXISTING, 0, NULL);

        if( port != INVALID_HANDLE_VALUE) {
            strcpy(list_of_ports[h], aux_string);
            CloseHandle( port );
            h++;
        }
    }

    return h;

    ////////////////////////////   UNIX   ////////////////////////////
#else

    DIR     *directory;
    struct  dirent *directory_p;
    int i = 0;

    directory = opendir("/dev");

    while ( ( directory_p = readdir(directory) ) && i < 10 ) {
        if (strstr(directory_p->d_name, "tty.usbserial") || strstr(directory_p->d_name, "ttyUSB")) {
            strcpy(list_of_ports[i], "/dev/" );
            strcat(list_of_ports[i], directory_p->d_name);
            i++;
        }
    }

    (void)closedir(directory);

    return i;

#endif

    return 0;
}


int open_serial(int baudrate){

        #define BAUD_RATE baudrate
        int l;
        char lista[10][255];
        char my_port[255];
        char port_s[255];
        FILE *file;

        int g=pthread_mutex_init(&(comm_settings_t.mut_ser),0);
        if(g==0)
                printf("Mutex seriale inizializzato!! \n");
        else
                printf("Mutex seriale non inizializzato!! \n");
        ////////////////////////////   WINDOWS CODE   ////////////////////////////
#if (defined(_WIN32) || defined(_WIN64))

        l = RS485listPorts(lista);

        if(l){
                strcpy(my_port, lista[0]);
                file = fopen("QBMOVE_FILE", "w+");
            if (file == NULL) {
                printf("Cannot open qbmove.conf\n");
            }
            fprintf(file,"serialport %s\n", my_port);
            fclose(file);
        }else{
                 puts("No serial port available.");
         return 0;
        }

        file = fopen("QBMOVE_FILE", "r");

    if (file == NULL) {
        //printf("Error opening file %s\n", QBMOVE_FILE);
        return 0;
    }

    fscanf(file, "serialport %s\n", port_s);

    fclose(file);


    DCB  dcb;                   // for serial port configuration
    COMMTIMEOUTS cts;           // for serial port configuration

//=========================================================     opening serial port

    comm_settings_t.file_handle =
        CreateFile( port_s,
                    GENERIC_WRITE|GENERIC_READ,
                    0, NULL, OPEN_EXISTING, FILE_FLAG_OPEN_REPARSE_POINT, NULL);

    if (comm_settings_t.file_handle == INVALID_HANDLE_VALUE) {
        goto error;
    }

//=========================================     serial communication properties

    dcb.DCBlength = sizeof (DCB);
```

```
    GetCommState(comm_settings_t.file_handle, &dcb);
    dcb.BaudRate  = BAUD_RATE;
    dcb.Parity    = NOPARITY;
    dcb.StopBits  = ONESTOPBIT;

    dcb.fOutxCtsFlow = FALSE;         // No CTS output flow control
    dcb.fOutxDsrFlow = FALSE;         // No DSR output flow control
    dcb.fDtrControl = FALSE;          // DTR flow control type

    dcb.fDsrSensitivity = FALSE;      // DSR sensitivity
    dcb.fTXContinueOnXoff = FALSE;     // XOFF continues Tx
    dcb.fOutX = FALSE;                // No XON/XOFF out flow control
    dcb.fInX = FALSE;                 // No XON/XOFF in flow control
    dcb.fErrorChar = FALSE;           // Disable error replacement
    dcb.fNull = FALSE;                // Disable null stripping
    dcb.fRtsControl = RTS_CONTROL_DISABLE;          // RTS flow control
    dcb.fAbortOnError = FALSE;        // Do not abort reads/writes on
                                      // error
    dcb.ByteSize = 8;                 // Number of bits/byte, 4-8

    dcb.DCBlength = sizeof(DCB);
    SetCommState(comm_settings_t.file_handle, &dcb);

    //Set up Input/Output buffer size
    SetupComm(comm_settings_t.file_handle, 100, 100);

    // timeouts
    GetCommTimeouts(comm_settings_t.file_handle, &cts);
    cts.ReadIntervalTimeout         = 0;                    // msec
    // ReadTimeout = Constant + Multiplier * Nwritten
    cts.ReadTotalTimeoutMultiplier  = 0;                    // msec
    cts.ReadTotalTimeoutConstant    = 100;                  // msec
    // WriteTimeout = Constant + Multiplier * Nwritten
    cts.WriteTotalTimeoutConstant   = 100;                  // msec
    cts.WriteTotalTimeoutMultiplier = 0;                    // msec
    SetCommTimeouts(comm_settings_t.file_handle, &cts);


    return 0;
error:

    if (comm_settings_t.file_handle != INVALID_HANDLE_VALUE){
        CloseHandle(comm_settings_t.file_handle);
                return -1;
    }

//////////////////////////////   UNIX CODE   ////////////////////////////////
#else

        //const char *port_s = "/dev/ttyUSB0";

        l = RS485listPorts(lista);

        if(l){
                strcpy(my_port, lista[0]);
                file = fopen("QBMOVE_FILE", "w+");
            if (file == NULL) {
                printf("Cannot open qbmove.conf\n");
            }
            fprintf(file,"serialport %s\n", my_port);
            fclose(file);
        }else{
                 puts("No serial port available.");
         return 0;
        }

        file = fopen("QBMOVE_FILE", "r");

    if (file == NULL) {
        //printf("Error opening file %s\n", QBMOVE_FILE);
        return 0;
    }

    fscanf(file, "serialport %s\n", port_s);

    fclose(file);


        struct termios options;
        int parity=0;

    comm_settings_t.file_handle =
        open(port_s, O_RDWR | O_NOCTTY | O_NONBLOCK);

    if(comm_settings_t.file_handle == -1) {
        goto error;
    }

    // prevent multiple openings
    if (ioctl(comm_settings_t.file_handle, TIOCEXCL) == -1) {
        goto error;
    }

    // set communication as BLOCKING

    if(fcntl(comm_settings_t.file_handle, F_SETFL, 0) == -1) {
        goto error;
    }
```

```
    if (tcgetattr(comm_settings_t.file_handle, &options) == -1) {
        goto error;
    }

    // set baud rate
    cfsetispeed(&options, BAUD_RATE);
    cfsetospeed(&options, BAUD_RATE);

#if (defined __APPLE__)

    // enable the receiver and set local mode
    options.c_cflag |= (CLOCAL | CREAD);

    // enable flags
    options.c_cflag &= ~PARENB;
    //options.c_cflag &= ~CSTOPB;
    options.c_cflag &= ~CSIZE;
    options.c_cflag |= CS8;

    //disable flags
    options.c_cflag &= ~CRTSCTS;
    options.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
    options.c_oflag &= ~OPOST;
    options.c_iflag &= ~(IXON | IXOFF | IXANY | INLCR);

    options.c_cc[VMIN] = 0;
    options.c_cc[VTIME] = 0;

#else
    cfmakeraw(&options);

    options.c_cc[VMIN] = 0;
    options.c_cc[VTIME] = 0;

    struct serial_struct serinfo;

    ioctl(comm_settings_t.file_handle, TIOCGSERIAL, &serinfo);
    serinfo.flags |= ASYNC_LOW_LATENCY;
    ioctl(comm_settings_t.file_handle, TIOCSSERIAL, &serinfo);
#endif

    // save changes
    if (tcsetattr(comm_settings_t.file_handle, TCSANOW, &options) == -1) {
        goto error;
    }

        printf("\n Porta seriale aperta!!\n");
        printf(" Handle: %d\n", comm_settings_t.file_handle);
    return 0;


error:
    if (comm_settings_t.file_handle != -1) {
        close(comm_settings_t.file_handle);
    }

    comm_settings_t.file_handle = INVALID_HANDLE_VALUE;
    return -1;
#endif
}

int RS485read( int ID, int *pck_l, int *dim){

        char *package;
        unsigned char data_in[BUFFER_SIZE] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};     // output data
        buffe
        unsigned int package_size = 6;
        int id=ID;
        package=(char *)pck_l;

        memcpy(package, data_in, package_size);

        // WINDOWS
        #if (defined(_WIN32) || defined(_WIN64))
        DWORD data_in_bytes = 0;

        if (!ReadFile(comm_settings_t.file_handle, data_in, 4, &data_in_bytes, NULL))
                return -1;

        // Control ID
        if ((id != 0) && (data_in[2] != id)) {
                return -1;
        }

                package_size = data_in[3];

        if (!ReadFile(comm_settings_t.file_handle, data_in, package_size, &data_in_bytes, NULL))
                return -1;

        // UNIX
        #else
        int n_bytes;
        struct timeval start, now;

        gettimeofday(&start, NULL);
        gettimeofday(&now, NULL);

        ioctl(comm_settings_t.file_handle, FIONREAD, &n_bytes);
```

```
        while((n_bytes < 4) && ( timevaldiff(&start, &now) < 3000) ) {
                gettimeofday(&now, NULL);
        ioctl(comm_settings_t.file_handle, FIONREAD, &n_bytes);
    }

        if (!read(comm_settings_t.file_handle, data_in, 4)) {
                    printf("punto 1\n");
                    return -1;
            }
int l;
        // Control ID
        if ((id != 0) && (data_in[2] != id)) {
                //for(l=0;l<4;l++){
                //printf("Valori %d\n",data_in[l]);
                //}
                printf("ID ricevuto %d\n Id mio: %d\n",data_in[2],id);
                printf("punto 2\n");
                return -1;
        }


        package_size = data_in[3];

        gettimeofday(&start, NULL);
        gettimeofday(&now, NULL);

        ioctl(comm_settings_t.file_handle, FIONREAD, &n_bytes);

        while((n_bytes < package_size) && ( timevaldiff(&start, &now) < 3000)) {

                gettimeofday(&now, NULL);
                ioctl(comm_settings_t.file_handle, FIONREAD, &n_bytes);
        }

        if (!read(comm_settings_t.file_handle, data_in, package_size)) {
                                printf("punto 3\n");
                return -1;
        }

        #endif

        // Control checksum
        if (checksum ( (char *) data_in, package_size - 1) != (char) data_in[package_size - 1]) {
                //return -1;
                printf("Errore nel checksum!! \n");
                return -1;
        }

        #ifdef VERBOSE
        printf("Received package size: %d \n", package_size);
        #endif

        memcpy(package, data_in, package_size);
        /*for(l=0;l<package_size;l++){
                printf("Valori letti: %d\n",data_in[l]);
        }*/

        *dim = package_size;
        pck_l =(int*)package;
        return package_size;
        //return;
}



int inner_read( int nb, unsigned char *data_in)
{
    struct timeval start, now, timeout;
    int err;
    int n_bytes;
    int to_read = nb;
    int yet_read = 0;

    gettimeofday(&start, NULL);
    now = start;

    fd_set r;


    timeout.tv_sec = 0;
    timeout.tv_usec = 600; //4000
    start = timeout;
    while(to_read > 0) {
        FD_ZERO(&r);
        FD_SET(comm_settings_t.file_handle, &r);
#if 0
    err = ioctl(comm_settings_t->file_handle, FIONREAD, &n_bytes);
    if(err == -1)
        perror("ioctl1");

    while((n_bytes < nb) && (timevaldiff(&start, &now) < 40000 * 100)) {
        gettimeofday(&now, NULL);
        err = ioctl(comm_settings_t->file_handle, FIONREAD, &n_bytes);
        if(err == -1)
            perror("ioctl2");
    }
#else
        err = select(comm_settings_t.file_handle + 1, &r, NULL, NULL, &timeout);
```

```
        if (err == -1) {
            perror("select");
            return -1;
        } else if (err == 0) {
            printf("TIMEOUT  (%07ld)\n", timevaldiff(&timeout, &start));
            return -1;
        } else {
            printf("elapsed: %07ld\n", timevaldiff(&timeout, &start));
        }

#endif
        err = read(comm_settings_t.file_handle, data_in + yet_read, to_read);
        if (err > 0) {
            to_read -= err;
            yet_read += err;
            printf("%s]bytes read: %d\n",__func__, err);
        } else {
            perror("read");
            printf("%s](err)bytes reading: %d \n",__func__, err);
            return -1;
        }
    }
    printf("%s](out)bytes reading: %d \n",__func__, yet_read);
    printf("TOTelapsed: %07ld\n", timevaldiff(&timeout, &start));
    return 0;
}

int RS485read_2( int ID, int *pck_l, int *dim)
{
    char *package;
    unsigned char data_in[BUFFER_SIZE] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};     // output data
    buffer
    unsigned int package_size = 6;
    int err;
    int id=ID;
    package=(char *)pck_l;

    memcpy(package, data_in, package_size);
#if 0
    int n_bytes;
    struct timeval start, now;

    gettimeofday(&start, NULL);
    gettimeofday(&now, NULL);

    err = ioctl(comm_settings_t->file_handle, FIONREAD, &n_bytes);
    if(err == -1)
        perror("ioctl1");

    while((n_bytes < 4) && ( timevaldiff(&start, &now) < 4000)) {
        gettimeofday(&now, NULL);
        err = ioctl(comm_settings_t->file_handle, FIONREAD, &n_bytes);
        if(err == -1)
            perror("ioctl2");
    }

    err = read(comm_settings_t->file_handle, data_in, 4);
    if (!err) {
            perror("read");
            printf("bytes reading: %d \n",err);
        return -1;
    }
#else
    err = inner_read( 4, data_in);
    if (err)
        return -1;
#endif
    // Control ID
    if ((id != 0) && (data_in[2] != id)) {
        printf("invalid ID \n");
        return -1;
    }


    package_size = data_in[3];
#if 0
    gettimeofday(&start, NULL);
    gettimeofday(&now, NULL);

    ioctl(comm_settings_t->file_handle, FIONREAD, &n_bytes);

    while((n_bytes < package_size) && ( timevaldiff(&start, &now) < 4000)) {
        gettimeofday(&now, NULL);
        ioctl(comm_settings_t->file_handle, FIONREAD, &n_bytes);
    }

    if (!read(comm_settings_t->file_handle, data_in, package_size)) {
        return -1;
    }
#else
    err = inner_read(package_size, data_in);
    if (err)
        return -1;
#endif
    // Control checksum
    if (checksum ( (char *) data_in, package_size - 1) != (char) data_in[package_size - 1]) {
        printf("crc error\n");
       return -1;
    }
```

```
#ifdef VERBOSE
    printf("Received package size: %d \n", package_size);
#endif

    memcpy(package, data_in, package_size);

    *dim = package_size;
    pck_l =(int*)package;
    return package_size;
}



void Write_RS485(char *buffer_info, int d_pck){
        int n_bytes;
        char package_in[BUFFER_SIZE];
        char package_out[BUFFER_SIZE];
        int ll,k;
        int err;
        //int *pck_prova=(int *)buffer_info;

        //printf("Sto per scrivere sulla seriale!! \n");

        //printf("lunghezza pacchetto: %d \n", d_pck);
        /*for(ll=0;ll<d_pck;ll++){
                printf("Valori stringa: %c \n",buffer_info[ll]);
        }*/

        #if(defined(_WIN32) || defined(_WIN64))
        DWORD package_size_out;

        WriteFile(comm_settings_t.file_handle, buffer_info, d_pck, &package_size_out, NULL);


        #else
        //ioctl(comm_settings_t.file_handle, FIONREAD, &n_bytes);
        //if(n_bytes)
                //read(comm_settings_t.file_handle, package_in, n_bytes);
    err=tcflush(comm_settings_t.file_handle, TCIFLUSH);

            if(err!=0)
            perror("tcflush");
            /*for(k=0;k<d_pck;k++){
                printf("Sto spedendo: %d \n",buffer_info[k]);
        }*/
        write(comm_settings_t.file_handle,buffer_info, d_pck);
        //printf("Ho scritto i valori nella seriale!!\n");

        #endif
}


void packet_to_send( uint32_t *buffer_info, package_to_send *p_send, int *get_reply){

        //printf("VALORE INDIRIZZO: %d \n", buffer_info);
        int id = buffer_info[0];
        int cmd = buffer_info[1];
        int dim_data= buffer_info[2];
        int h;
        int dim_b=0;
        char buffer_data[BUFFER_SIZE];
        char activate;


        if(cmd==0)
                activate=buffer_info[2];

        //printf("\n valore ingresso: %d %d %d \n", id, cmd, dim_data);

        buffer_data[0]= ':';
        buffer_data[1]= ':';
        buffer_data[2] = (unsigned char)id;


        switch(cmd){
                case 0:
                        buffer_data[3] = 3;
                        buffer_data[4] = CMD_ACTIVATE;
                        buffer_data[5] = activate ? 3 : 0;
                        buffer_data[6] = checksum(buffer_data+4,2);
                        dim_b = 7;
                        *get_reply=0;

                        break;
                case 1:
                        buffer_data[3] = 2;
                        buffer_data[4] = CMD_GET_ACTIVATE;              // command
                        buffer_data[5] = CMD_GET_ACTIVATE;
                        dim_b= 6;
                        *get_reply=1;
                        break;
                case 2:
                        buffer_data[3] = 2;
                        buffer_data[4] = CMD_PING;
                        buffer_data[5] = CMD_PING;
                        dim_b = 6;
                        *get_reply=1;
                        break;
```

```
case 3:
        buffer_data[3] = 6;
        buffer_data[4] = CMD_SET_INPUTS;
        buffer_data[5] = ((char *) &inputs[0])[1];
        buffer_data[6] = ((char *) &inputs[0])[0];
        buffer_data[7] = ((char *) &inputs[1])[1];
        buffer_data[8] = ((char *) &inputs[1])[0];
        buffer_data[9] = checksum(buffer_data + 4, 5);   // checksum
        dim_b= 10;
        *get_reply=0;
        break;
case 4:
        buffer_data[3] = 6;
        buffer_data[4] = CMD_SET_POS_STIFF;
        buffer_data[5] = ((char *) &inputs[0])[1];
        buffer_data[6] = ((char *) &inputs[0])[0];
        buffer_data[7] = ((char *) &inputs[1])[1];
        buffer_data[8] = ((char *) &inputs[1])[0];
        buffer_data[9] = checksum(buffer_data + 4, 5);        // checksum
        dim_b = 10;
        *get_reply=0;
        break;
case 5:
        buffer_data[3] = 2;
        buffer_data[4] = CMD_GET_INPUTS;             // command
        buffer_data[5] = CMD_GET_INPUTS;
        dim_b = 6;
        *get_reply=1;
        break;
case 6:
        buffer_data[3] = 2;
        buffer_data[4] = CMD_GET_MEASUREMENTS;             // command
        buffer_data[5] = CMD_GET_MEASUREMENTS;
        dim_b = 6;
        *get_reply=1;
        break;
case 7:
        buffer_data[3] = 2;
        buffer_data[4] = CMD_GET_CURRENTS;             // command
        buffer_data[5] = CMD_GET_CURRENTS;
        dim_b = 6;
        *get_reply=1;
        break;
case 8:
        buffer_data[3] = 2;
        buffer_data[4] = CMD_GET_CURR_AND_MEAS;             // command
        buffer_data[5] = CMD_GET_CURR_AND_MEAS;
        dim_b = 6;
        *get_reply=1;
        break;
case 9:
        buffer_data[3] = 2;
        buffer_data[4] = CMD_GET_VELOCITIES;             // command
        buffer_data[5] = CMD_GET_VELOCITIES;
        dim_b = 6;
        *get_reply=1;
        break;
case 10:
        buffer_data[3] = 4;
        buffer_data[4] = CMD_GET_INFO;
        buffer_data[5] = ((unsigned char *) &info_type)[1];   // parameter type
        buffer_data[6] = ((unsigned char *) &info_type)[0];   // parameter type
        buffer_data[7] = checksum(buffer_data + 4, 3);           // checksum
        dim_b = 8;
        *get_reply=1;
        break;
case 11:
        buffer_data[3] = 2;
        buffer_data[4] = CMD_BOOTLOADER;           // command
        buffer_data[5] = CMD_BOOTLOADER;
        dim_b= 6;
        *get_reply=0;
        break;
case 12:
        buffer_data[3] = 2;
        buffer_data[4] = CMD_CALIBRATE;            // command
        buffer_data[5] = CMD_CALIBRATE;            // checksum
        dim_b = 6;
        *get_reply=0;
        break;
case 13:
        /*enum qbmove_parameter type;
        void *values;
        unsigned short int value_size;
        void *value;

        Select_SetParam(type, values, &value_size, value);
        buffer_data[3] = 4 + num_of_values * value_size;
        buffer_data[4] = CMD_SET_PARAM;                // command
        buffer_data[5] = ((char *) &type)[1];          // parameter type
        buffer_data[6] = ((char *) &type)[0];          // parameter type

        for(h = 0; h < num_of_values; ++h) {
                for(i = 0; i < value_size; ++i) {
                        buffer_data[ h * value_size +  7 + i ] =
                                ((char *) value)[ h * value_size + value_size - i - 1 ];
                }
        }
        buffer_data[ 7 + num_of_values * value_size ] =
```

```
                                checksum( buffer_data + 4, 3 + num_of_values * value_size );
                        *dim =  7 + num_of_values * value_size+1;*/
                        break;
                case 14:
                        /*buffer_data[3] = 4;
                        buffer_data[4] = CMD_GET_PARAM;                 // command
                        buffer_data[5] = ((char *) &type)[1];           // parameter type
                        buffer_data[6] = ((char *) &type)[0];           // parameter type
                        buffer_data[7] = checksum (buffer_data + 4, 3);   // checksum
                        *dim = 8;*/
                        break;
                case 15:
                        buffer_data[3] = 2;
                        buffer_data[4] = CMD_STORE_PARAMS;          // command
                        buffer_data[5] = CMD_STORE_PARAMS;
                        *get_reply=0;
                        dim_b = 6;
                        break;
                case 16:
                        buffer_data[3] = 2;
                        buffer_data[4] = CMD_STORE_DEFAULT_PARAMS;          // command
                        buffer_data[5] = CMD_STORE_DEFAULT_PARAMS;
                        dim_b = 6;
                        *get_reply=0;
                        break;
                case 17:
                        buffer_data[3] = 2;
                        buffer_data[4] = CMD_RESTORE_PARAMS;            // command
                        buffer_data[5] = CMD_RESTORE_PARAMS;
                        dim_b = 6;
                        *get_reply=0;
                        break;
                case 18:
                        buffer_data[3] = 2;
                        buffer_data[4] = CMD_INIT_MEM;              // command
                        buffer_data[5] = CMD_INIT_MEM;
                        dim_b = 6;
                        *get_reply=0;
                        break;
                default:
                        break;
        }

        for(h=0;h<dim_b;h++){
                p_send->buffer_aux[h]=buffer_data[h];
                //printf("Valori nella struct: %d !!\n",p_send->buffer_aux[h]);
        }
        p_send->dim_pck=dim_b;
        p_send->id=id;

}

int insert_value(float *value_new, int num_value, int CMD){

        switch(CMD){
                case 3:
                        inputs[0]=value_new[0];
                        inputs[1]=value_new[1];
                        //printf("\n Valori passati al cubotto in posizione 1: q11 %d, q12 %d\n", inputs[0],
                        inputs[1]);
                        break;
                case 4:
                        inputs[0]=value_new[0];
                        inputs[1]=value_new[1];
                        break;
                default:
                        break;
        }

        return 0;

}



/*char checksum ( char * data_buffer, int data_length ) {

    int i;
    char checksum = 0x00;

    for(i = 0; i < data_length; ++i) {
       checksum = checksum ^ data_buffer[i];
    }

    return checksum;
}*/

long timevaldiff(struct timeval *starttime, struct timeval *finishtime)
{
    long usec;
    usec=(finishtime->tv_sec-starttime->tv_sec)*1000000;
    usec+=(finishtime->tv_usec-starttime->tv_usec);
    return usec;
}

void Close_Serial(void){
        close(comm_settings_t.file_handle);
}
```

"Control_alg2.h"

```
int Traiettoria(float *Error, float *pos, float *vel);
int Dinamic_Model(float *v_q, float *v_dq);
void control_qb(float *value_new_con1, float *value_new_con2, float *Kp, float *Kd, float *pos_act, int num_qb,
int num_sens, int *val_agg);
```

"Control_alg2.c"

```
#include "math.h"
#include <stdio.h>
#include <time.h>
#include <sys/time.h>
#include "definitions.h"
#include "Gestori_Read_Write.h"
#include "qbmove_com.h"
#include "Control_alg2.h"

#define DEFAULT_RESOLUTION 1
#define DEG_TICK_MULTIPLIER (65536.0 / (360.0 * (pow(2, DEFAULT_RESOLUTION))))

#define NUM_OF_SENSORS 3
#define NUM_QB 4
#define Massa 0.450
#define test_sinusoide= 1;
#define test_fisso= 0;

FILE *file_error_X;
FILE *file_error_Y;
FILE *file_error_Z;
FILE *file_coppie;
FILE *file_vettoreG;
FILE *file_posizioneletta;
FILE *file_comandiqb1;
FILE *file_lettiqb1;

float q1=0;
float q2=0;
float q3=0;
float q4=0;
float dot_q10=0.9;
float dot_q20=0.9;
float dot_q30=0.7;
float dot_q40=0.8;
float dot_q1=0;
float dot_q2=0;
float dot_q3=0;
float dot_q4=0;
float dotdot_q1=0;
float dotdot_q2=0;
float dotdot_q3=0;
float dotdot_q4=0;
float q1_0=0;
float q2_0=-(3.1415/4);
float q3_0=0;
float q4_0=0;

float pos_des[3];
float dotdot_q[NUM_QB];
float dot_q[NUM_QB];
float q[NUM_QB];

float Bm_t[NUM_QB][NUM_QB];
float C[NUM_QB][NUM_QB];
float PsdJacob_rid[NUM_QB][3];
float Jacob_rid[3][NUM_QB];
float Jacobtrp_rid[NUM_QB][3];
float g = 9.81;
float G[NUM_QB];
float B_inv[NUM_QB][NUM_QB];
float d1=0.072;
float ln2=0;
float ln3=0;
float d3=0.17;
float ln4=0.07;

float EE[3];
float dotEE[3];

float ref_trk[3];
float t_oldTr;
float x_old;
int counter_request;
extern struct timeval start_newcmd, now_cmd, now_read, start_now_read, newcmd_old, oldnow_read, new_cmdf,
now_readf;
float dt_old=0.002;
float quota=0;
float time_old;
struct timeval start_time, actual_time;

int Traiettoria(float *Error, float *pos, float *vel){
        static float X,Y,Z;
        float dotX, dotY, dotZ;
        static float quota_des;
        int ll=0;
```

```
float t, w, w_h;
static float verso=1;
static float l=0;
static float l_g=0;
static float l_r=0;
static float l_h=0;
static float l_hr=0;
static float w_hd=0;
float time_s;

float raggio=0.24*cos(-(3.1415/4));
float omega=1.5090;
float Z_quota;

//quota des con offset
quota_des=-d3*sin(q2_0) - ln4*cos(q4_0)*sin(q2_0) - ln4*cos(q2_0)*cos(q3_0)*sin(q4_0);
printf("\nValore quota des %f\n", quota_des);


if(quota==0){


                w=-(((l_g*0.5)/180)*3.1415);
                w_p=-((((l_g*0.5)+1)/180)*3.1415);
                w_m=-((((l_g*0.5)-1)/180)*3.1415);
                //printf("\nValore w %f\n", w);
                if(fabs(w) > fabs(q2_0)){
                        //printf("\nValore q2_0 %f\n", q2_0);
                        w=q2_0;
                        w_p=-((((-3.1415/4)+1)/180)*3.1415);
                        w_m=-((((-3.1415/4)-1)/180)*3.1415);
                }
                //con offset
                Z_quota=-d3*sin(w) - ln4*cos(q4_0)*sin(w) - ln4*cos(w)*cos(q3_0)*sin(q4_0);
                Z_quota_p=-d3*sin(w_p) - ln4*cos(q4_0)*sin(w_p) - ln4*cos(w_p)*cos(q3_0)*sin(q4_0);
                Z_quota_m=-d3*sin(w_m) - ln4*cos(q4_0)*sin(w_m) - ln4*cos(w_m)*cos(q3_0)*sin(q4_0);

                Z=Z_quota;
                X=raggio;
                l_g++;


        if(Error[2]<=0.01 && Error[2]>=0){
                t=t_oldTr+0.015;
                if(t>=2){
                        printf("\nValore errore %f\n", Error[2]);
                        quota=1;
                        w_h=q2_0;
                }
        }
        if(Error[2]<0 && Error[2]>=-0.01){
                t=t_oldTr+0.015;
                if(t>=2){
                        printf("\nValore errore caso 2 %f\n", Error[2]);
                        quota=1;
                        w_h=q2_0;

                }
        }

        if(Error[2]>0.01 || Error[2]<-0.01){
                t_oldTr=0;
        }else{
                t_oldTr=t;
        }
}
if(test_fisso){
if(quota==1){
        if(l<500){
                l++;
                if(verso==1){
                        w=((l*0.20)/180)*3.1415;
                        //printf("\nAngolo: %f\n",w);
                        X=raggio*cos(w);
                        Y=raggio*sin(w);
                        //con offset
                        Z=-d3*sin(q2_0) - ln4*cos(q4_0)*sin(q2_0) - ln4*cos(q2_0)*cos(q3_0)*sin(q4_0);
                }else{
                        w=((l*0.20)/180)*3.1415;
                        //printf("\nAngolo: %f\n",w);
        X=raggio*cos(-w);
                        Y=raggio*sin(-w);
                        //con offset
                        Z=-d3*sin(q2_0) - ln4*cos(q4_0)*sin(q2_0) - ln4*cos(q2_0)*cos(q3_0)*sin(q4_0);
                }
                if(l==500)
                        l_r=l;
        }else{
                l_r--;
                if(verso==1){
                        w=((l_r*0.20)/180)*3.1415;
                        //printf("\nAngolo: %f\n",w);
                        X=raggio*cos(w);
                        Y=raggio*sin(w);
                        //con offset
                        Z=-d3*sin(q2_0) - ln4*cos(q4_0)*sin(q2_0) - ln4*cos(q2_0)*cos(q3_0)*sin(q4_0);
                }else{
                        w=((l_r*0.20)/180)*3.1415;
                        //printf("\nAngolo: %f\n",w);
```

```
                                X=raggio*cos(-w);
                                Y=raggio*sin(-w);
                                //con offset
                                Z=-d3*sin(q2_0) - ln4*cos(q4_0)*sin(q2_0) - ln4*cos(q2_0)*cos(q3_0)*sin(q4_0);
                        }
                        if(l_r==0){
                                l=l_r;
                                if(verso==1)
                                        verso=-1;
                                else
                                        verso=1;
                        }
                }

        }
        }


        dotX=-omega*raggio*sin(omega*t);
        dotY=omega*raggio*cos(omega*t);
        dotZ=0;

        if(test_sinusoide){
                if(quota==1){
                l_h=l_h+0.014;
                float an=1*l_h;
                printf("tempo Z: %f", l_h);
                printf("angolo Z: %f", 0.03*sin(an));
                Z=-d3*sin(q2_0)-ln4*cos(q4_0)*sin(q2_0)-ln4*cos(q2_0)*cos(q3_0)*sin(q4_0)+(0.03*sin(an));
                w_h=-asin(Z/0.24);
                if(l<500){
                        l++;
                        if(verso==1){
                                w=((l*0.20)/180)*3.1415;
                                //printf("\nAngolo: %f\n",w);
                                X=(0.24*cos(w_h))*cos(w);
                                Y=(0.24*cos(w_h))*sin(w);
                        }else{
                                w=((l*0.20)/180)*3.1415;
                                X=(0.24*cos(w_h))*cos(-w);
                                Y=(0.24*cos(w_h))*sin(-w);
                        }
                        if(l==500)
                                l_r=l;
                }else{
                        l_r--;
                        if(verso==1){
                                w=((l_r*0.20)/180)*3.1415;
                                //printf("\nAngolo: %f\n",w);
                                X=(0.24*cos(w_h))*cos(w);
                                Y=(0.24*cos(w_h))*sin(w);

                        }else{
                                w=((l_r*0.20)/180)*3.1415;
                                X=(0.24*cos(w_h))*cos(-w);
                                Y=(0.24*cos(w_h))*sin(-w);

                        }
                        if(l_r==0){
                                l=l_r;
                                if(verso==1)
                                        verso=-1;
                                else
                                        verso=1;
                        }
                }

             }
        }


        EE[0]=X;
        EE[1]=Y;
        EE[2]=Z;
        dotEE[0]=dotX;
        dotEE[1]=dotY;
        dotEE[2]=dotZ;

        for(ll=0;ll<3;ll++){
                pos[ll]=EE[ll];
                vel[ll]=dotEE[ll];
        }

        time_s_old=time_s;
    x_old=X;

        return 0;
}


void control_qb(float *value_new_con1, float *value_new_con2, float *Kp, float *Kd, float *pos_act, int num_qb,
int num_sens, int *val_agg){

        float m1=Massa;
        float m2=Massa;
        float m3=Massa;
        float m4=0.050;
        float tau_cntrl[NUM_QB];
```

```
float value_stiff[NUM_QB];
float pos_int[3];
float vel_des[3];
float kp[3][3];
float kd[3][3];
int j,h,k;
float kp_1, kp_2, kp_3;
float kd_1, kd_2, kd_3;
float q1_eq,q2_eq,q3_eq,q4_eq;
float preset1,preset2, preset3, preset4;
float X_ee, Y_ee, Z_ee;
static float er_x=1000;
static float er_y=1000;
static float er_z=1000;
static float error_pos[3]={-1, -1, -1};
float dotEE_act[NUM_QB];
float comp_prop[3];
float comp_der[3];
float comp_PD[NUM_QB];
static float time_old=0;
float delta_time=0;
static float q_old[NUM_QB]={ 0, 0, 0, 0};
int count_not_ag=0;
static int v_a[NUM_QB];
float delta_timer;
float p_qb1[NUM_OF_SENSORS], p_qb2[NUM_OF_SENSORS], p_qb3[NUM_OF_SENSORS], p_qb4[NUM_OF_SENSORS]; //in
ogni vettore, per ogni cubotto si trovano nell'ordine la posizione angolare del primo motore, del secondo
motore e dell'albero motore

// qi sono le posizioni degli alberi d'uscita di ogni cubo, qi_1 e qi_2 sono le posizioni angolari dei
due servomotori
float tau_t[NUM_QB];
float k1 = 0.0227;      //Nm
float a1 = 6.7328;      //1/rad
float k2 = 0.0216;      //Nm
float a2 = 6.9602;      //1/rad
float k_m=(k1+k2)/2;
float a=(a1+a2)/2;
static float l=0;
static count_init=0;
static float time=0;
float final_time= 0.048;
float delta_t;

if(count_init == 0)
        gettimeofday(&start_time, NULL);
count_init++;

gettimeofday(&actual_time, NULL);

for(h=0;h<NUM_QB;h++){
        if(val_agg[h]==-3){
                value_new_con1[h]=0;
                value_new_con2[h]=0;
                v_a[h]=0;
        }
        if(val_agg[h]==1){
                v_a[h]=val_agg[h];
        }

}

for(h=0;h<NUM_QB;h++){
        count_not_ag=count_not_ag+v_a[h];
}

if(count_not_ag<NUM_QB){
        file_error_X = fopen("ERRORE_X", "w+");
        file_error_Y = fopen("ERRORE_Y", "w+");
        file_error_Z = fopen("ERRORE_Z", "w+");
        file_coppie = fopen("COPPIE", "w+");
        file_vettoreG = fopen("VETTORE_G", "w+");
        file_posizioneletta = fopen("POS_LETTA", "w+");
        file_comandiqb1 = fopen("POS_QB1","w+");
        file_lettiqb1 = fopen("LETTURA_QB1","w+");
}

if(count_not_ag==NUM_QB){

        for(h=0;h<NUM_OF_SENSORS;h++){
                p_qb1[h]=*(pos_act+0*NUM_OF_SENSORS+h);
                p_qb2[h]=*(pos_act+1*NUM_OF_SENSORS+h);
                p_qb3[h]=*(pos_act+2*NUM_OF_SENSORS+h);
                p_qb4[h]=*(pos_act+3*NUM_OF_SENSORS+h);
        }
        /*printf("\nValori in ingresso all'algoritmo di controllo: q11 %f q12 %f q %f\n",p_qb1[0],
        p_qb1[1], p_qb1[2]);
        printf("\nValori in ingresso all'algoritmo di controllo: q21 %f q22 %f q %f\n",p_qb2[0],
        p_qb2[1], p_qb2[2]);
        printf("\nValori in ingresso all'algoritmo di controllo: q31 %f q32 %f q %f\n",p_qb3[0],
        p_qb3[1], p_qb3[2]);
        printf("\nValori in ingresso all'algoritmo di controllo: q41 %f q42 %f q %f\n",p_qb4[0],
        p_qb4[1], p_qb4[2]);*/
        fprintf(file_lettiqb1, "\n q111 %f q121 %f ql %f ",p_qb1[0], p_qb1[1], p_qb1[2]);

        q1=p_qb1[2]/DEG_TICK_MULTIPLIER;        //gradi
        q2=(p_qb2[2]-380)/DEG_TICK_MULTIPLIER; //+250
        q3=p_qb3[2]/DEG_TICK_MULTIPLIER;
        q4=p_qb4[2]/DEG_TICK_MULTIPLIER;
```

```
                q1=q1*(3.1415/180);
                q2=q2*(3.1415/180);
                q3=q3*(3.1415/180);
                q4=q4*(3.1415/180);

                if(quota==0){
                        q1=0;
                        q3=0;
                        q4=0;
                }

                q1_eq=(p_qb1[0]+p_qb1[1])/(2*DEG_TICK_MULTIPLIER);      //gradi
                q2_eq=(p_qb2[0]+p_qb2[1])/(2*DEG_TICK_MULTIPLIER);
                q3_eq=(p_qb3[0]+p_qb3[1])/(2*DEG_TICK_MULTIPLIER);
                q4_eq=(p_qb4[0]+p_qb4[1])/(2*DEG_TICK_MULTIPLIER);

                //qi_eq espresse in radianti
                q1_eq=q1_eq*(3.1415/180);
                q2_eq=q2_eq*(3.1415/180);
                q3_eq=q3_eq*(3.1415/180);
                q4_eq=q4_eq*(3.1415/180);

                //printf("\n Valori di q1: %f e di q1_eq: %f\nValori di q2: %f e di q2_eq: %f\nValori di q3: %f e
                di q3_eq: %f\nValori di q4: %f e di q4_eq: %f\n", q1, q1_eq, q2, q2_eq,q3, q3_eq,q4, q4_eq);
                preset1=0;      //preset1=0.03; //0
                preset2=0;
                preset3=0;
                preset4=0;

                //if(time==0)
                Traiettoria( error_pos, pos_des, vel_des);

        //---------------------------------Calcolo della posizione corrente dell'EE e dell'errore di
        posizione---------------------------------//

                //Vettore Posizione con offset
                X_ee=d3*cos(q1)*cos(q2) + ln4*cos(q1)*cos(q2)*cos(q4) + ln4*((-cos(q1))*cos(q3)*sin(q2) -
                sin(q1)*sin(q3))*sin(q4);
                Y_ee=d3*cos(q2)*sin(q1) + ln4*cos(q2)*cos(q4)*sin(q1) + ln4*((-cos(q3))*sin(q1)*sin(q2) +
                cos(q1)*sin(q3))*sin(q4);
                Z_ee=-d3*sin(q2) - ln4*cos(q4)*sin(q2) - ln4*cos(q2)*cos(q3)*sin(q4);



            //er_x=pos_des[0]-q1;
                er_x=pos_des[0]-X_ee;
                er_x_m=X_m-X_ee;
                er_x_p=X_p-X_ee;
                fprintf(file_error_X,"\n Valore della posizione desiderata lungo X: %f, posizione attuale: %f,
                Errore nom: %f\n", pos_des[0], X_ee, er_x);
                er_y=pos_des[1]-Y_ee;
                er_y_m=Y_m-Y_ee;
                er_y_p=Y_p-Y_ee;
                fprintf(file_error_Y, "\n Valore della posizione desiderata lungo Y: %f, posizione attuale: %f,
                Errore nom: %f\n", pos_des[1], Y_ee, er_y);
                er_z=pos_des[2]-Z_ee;
                er_z_p=Z_quota_p-Z_ee;
                er_z_m=Z_quota_m-Z_ee;
                fprintf(file_error_Z, "\n Valore della posizione desiderata lungo Z: %f, posizione attuale: %f,
                Errore nom: %f\n", pos_des[2], Z_ee, er_z);
            //printf("\n Valore della posione desiderata : %f, posizione attuale: %f, Errore: %f\n", pos_des[0],
            q1, er_x);
                //printf("\n Valore della posione desiderata : %f, posizione attuale: %f, Errore: %f\n",
                pos_des[0], q2, er_y);
                //printf("\n Valore della posione desiderata : %f, posizione attuale: %f, Errore: %f\n",
                pos_des[0], q3, er_z);
        //printf("\n Valore della posione desiderata : %f, posizione attuale: %f, Errore: %f\n", pos_des[0], q4,
        pos_des[0]-q4);
                error_pos[0] = er_x;
                error_pos[1] = er_y;
                error_pos[2] = er_z;


//-------------------------------------------------------------------------------------------------------------------------------------------------------
                //Espressione della dinamica
                //tau_t[0] = k1*sinh(a1*(q1-q1_1))+k2*sinh(a2*(q1-q1_2));
                //tau_t[1] = k1*sinh(a1*(q2-q2_1))+k2*sinh(a2*(q2-q2_2));
                //tau_t[2] = k1*sinh(a1*(q3-q3_1))+k2*sinh(a2*(q3-q3_2));
                //tau_t[3] = k1*sinh(a1*(q4-q4_1))+k2*sinh(a2*(q4-q4_2));

                kp_1=Kp[0];
                kp_2=Kp[1];
                kp_3=Kp[2];

                kp[0][0] = kp_1;
                kp[0][1] = 0;
                kp[0][2] = 0;
                kp[1][0] = 0;
                kp[1][1] = kp_2;
                kp[1][2] = 0;
                kp[2][0] = 0;
                kp[2][1] = 0;
                kp[2][2] = kp_3;

                kd_1=Kd[0];
                kd_2=Kd[1];
                kd_3=Kd[2];
```

```
        kd[0][0] = kd_1;
        kd[0][1] = 0;
        kd[0][2] = 0;
        kd[1][0] = 0;
        kd[1][1] = kd_2;
        kd[1][2] = 0;
        kd[2][0] = 0;
        kd[2][1] = 0;
        kd[2][2] = kd_3;

        float Jac_pos[3][NUM_QB];

        //Jacobiano con offset
        Jac_pos[0][0] = -d3*cos(q2)*sin(q1) - ln4*cos(q2)*cos(q4)*sin(q1) +
        ln4*cos(q3)*sin(q1)*sin(q2)*sin(q4) - ln4*cos(q1)*sin(q3)*sin(q4);
        Jac_pos[0][1] = -d3*cos(q1)*sin(q2) - ln4*cos(q1)*cos(q4)*sin(q2) -
        ln4*cos(q1)*cos(q2)*cos(q3)*sin(q4);
        Jac_pos[0][2] = -ln4*cos(q3)*sin(q1)*sin(q4) + ln4*cos(q1)*sin(q2)*sin(q3)*sin(q4);
        Jac_pos[0][3] = -ln4*cos(q1)*cos(q3)*cos(q4)*sin(q2) - ln4*cos(q4)*sin(q1)*sin(q3) -
        ln4*cos(q1)*cos(q2)*sin(q4);
        Jac_pos[1][0] = d3*cos(q1)*cos(q2) + ln4*cos(q1)*cos(q2)*cos(q4) -
        ln4*cos(q1)*cos(q3)*sin(q2)*sin(q4) - ln4*sin(q1)*sin(q3)*sin(q4);
        Jac_pos[1][1] = -d3*sin(q1)*sin(q2) - ln4*cos(q4)*sin(q1)*sin(q2) -
        ln4*cos(q2)*cos(q3)*sin(q1)*sin(q4);
        Jac_pos[1][2] = ln4*cos(q1)*cos(q3)*sin(q4) + ln4*sin(q1)*sin(q2)*sin(q3)*sin(q4);
        Jac_pos[1][3] = -ln4*cos(q3)*cos(q4)*sin(q1)*sin(q2) + ln4*cos(q1)*cos(q4)*sin(q3) -
        ln4*cos(q2)*sin(q1)*sin(q4);
        Jac_pos[2][0] = 0;
        Jac_pos[2][1] = -d3*cos(q2) - ln4*cos(q2)*cos(q4) + ln4*cos(q3)*sin(q2)*sin(q4);
        Jac_pos[2][2] = ln4*cos(q2)*sin(q3)*sin(q4);
        Jac_pos[2][3] = -ln4*cos(q2)*cos(q3)*cos(q4) + ln4*sin(q2)*sin(q4);



        //Vettore gravitazionale con offset
        G[0] = 0;
        fprintf(file_vettoreG, "\n Valore G[0]: %f\n", G[0]);
        G[1] = g*(cos(q2)*(-0.1*m2-d3*(m4+m3)-ln4*m4*cos(q4))+ln4*m4*cos(q3)*sin(q2)*sin(q4));
        //G[1] =
        -g*(m4*(0.066*cos(q4)+0.17)*cos(q2)+m3*0.17*cos(q2)+m2*0.1*cos(q2)-0.066*sin(q2)*cos(q3)*sin(q4)*m4);
        fprintf(file_vettoreG, "\n Valore G[1]: %f\n", G[1]);
        G[2] = g*ln4*m4*cos(q2)*sin(q3)*sin(q4);
        fprintf(file_vettoreG, "\n Valore G[2]: %f\n", G[2]);
        G[3] = g*ln4*m4*(-cos(q2)*cos(q3)*cos(q4)+sin(q2)*sin(q4));
        //G[3] = g*0.066*m4*(cos(q4)*cos(q2)*cos(q3)+sin(q2)*sin(q4));
        fprintf(file_vettoreG, "\n Valore G[3]: %f\n", G[3]);


        delta_time = timevaldiff(&newcmd_old, &start_newcmd);
        delta_time = delta_time/1000000;
        printf("\n delta_time tra chiamata Gestore Write in sec: %f\n",delta_time);
        delta_timer = timevaldiff(&oldnow_read, &start_now_read);
        delta_timer = delta_timer/1000000;
        printf("\n delta_time tra chiamata Gestore Read in sec: %f\n",delta_timer);
        if(delta_timer<=0 || delta_timer>0.015){
                delta_timer=0.015;
        }else{
                //delta_time=0.008;
                dt_old=delta_timer;
        }

        dot_q[0] = (q1 - q_old[0])/delta_timer;
        dot_q[1] = (q2 - q_old[1])/delta_timer;
        dot_q[2] = (q3 - q_old[2])/delta_timer;
        dot_q[3] = (q4 - q_old[3])/delta_timer;
        for(j=0;j<NUM_QB;j++){
                if(dot_q[j]>0.01){
                        dot_q[j]=0.01;
                }
                if(dot_q[j]<-0.01){
                        dot_q[j]=-0.01;
                }
                printf("\nValore velocit : %f\n", dot_q[j]);
        }

        //printf("\n Value q: %f, e q_old: %f\n",q1, q_old[0]);
        //printf("\n Value q: %f, e q_old: %f\n",q2, q_old[1]);
        //printf("\n Value q: %f, e q_old: %f\n",q3, q_old[2]);
        //printf("\n Value q: %f, e q_old: %f\n",q4, q_old[3]);

//---------------Calcolo delle tau di controllo------------------//
        for(k=0;k<NUM_QB;k++){
                tau_cntrl[k] = 0;
        }

//----------------Calcolo della velocit  corrente dell'EE------------------//
        for(k=0;k<3;k++){
                dotEE_act[k]=0;
        }
        for(k=0;k<3;k++){
                for(h=0;h<NUM_QB;h++){
                        //printf("\n Value dot_q: %f\n",dot_q[h]);
                        dotEE_act[k]=dotEE_act[k]+Jac_pos[k][h]*dot_q[h];
                }
        }
//-------------------------------------------------------------------------//

        float Jac_posTrp[NUM_QB][3];
```

```
                //JAcobTrp con offset
                Jac_posTrp[0][0] = -d3*cos(q2)*sin(q1) - ln4*cos(q2)*cos(q4)*sin(q1) +
                ln4*cos(q3)*sin(q1)*sin(q2)*sin(q4) - ln4*cos(q1)*sin(q3)*sin(q4);
                Jac_posTrp[0][1] = d3*cos(q1)*cos(q2) + ln4*cos(q1)*cos(q2)*cos(q4) -
                ln4*cos(q1)*cos(q3)*sin(q2)*sin(q4) - ln4*sin(q1)*sin(q3)*sin(q4);
                Jac_posTrp[0][2] = 0;
                Jac_posTrp[1][0] = -d3*cos(q1)*sin(q2) - ln4*cos(q1)*cos(q4)*sin(q2) -
                ln4*cos(q1)*cos(q2)*cos(q3)*sin(q4);
                Jac_posTrp[1][1] = -d3*sin(q1)*sin(q2) - ln4*cos(q4)*sin(q1)*sin(q2) -
                ln4*cos(q2)*cos(q3)*sin(q1)*sin(q4);
                Jac_posTrp[1][2] = - d3*cos(q2) - ln4*cos(q2)*cos(q4) + ln4*cos(q3)*sin(q2)*sin(q4);
                Jac_posTrp[2][0] = -ln4*cos(q3)*sin(q1)*sin(q4) + ln4*cos(q1)*sin(q2)*sin(q3)*sin(q4);
                Jac_posTrp[2][1] = ln4*cos(q1)*cos(q3)*sin(q4) + ln4*sin(q1)*sin(q2)*sin(q3)*sin(q4);
                Jac_posTrp[2][2] = ln4*cos(q2)*sin(q3)*sin(q4);
                Jac_posTrp[3][0] = -ln4*cos(q1)*cos(q3)*cos(q4)*sin(q2) - ln4*cos(q4)*sin(q1)*sin(q3) -
                ln4*cos(q1)*cos(q2)*sin(q4);
                Jac_posTrp[3][1] = -ln4*cos(q3)*cos(q4)*sin(q1)*sin(q2) + ln4*cos(q1)*cos(q4)*sin(q3) -
                ln4*cos(q2)*sin(q1)*sin(q4);
                Jac_posTrp[3][2] = -ln4*cos(q2)*cos(q3)*cos(q4) + ln4*sin(q2)*sin(q4);


                //Calcolo della componente proporzionale e della componente derivativa del controllore PD
                        for(k=0;k<3;k++){
                                comp_prop[k]=0;
                                comp_der[k]=0;
                                for(h=0;h<3;h++){
                                        comp_prop[k]=comp_prop[k]+kp[k][h]*error_pos[h];
                                        comp_der[k]=comp_der[k]+kd[k][h]*dotEE_act[h];
                                }
                        }

                        fprintf(file_coppie,"\n\n\n");
                        for(k=0;k<NUM_QB;k++){
                                comp_PD[k]=0;
                                tau_cntrl[k]=0;
                                fprintf(file_coppie,"\n");
                                for(h=0;h<3;h++){
                                        comp_PD[k]=comp_PD[k]+(Jac_posTrp[k][h]*comp_prop[h]-Jac_posTrp[k][h]*comp_der[h]);
                                }
                                tau_cntrl[k]=tau_cntrl[k]+comp_PD[k]+G[k];

                                if(tau_cntrl[k]>3){
                                        tau_cntrl[k]=3;
                                        printf("\nValore delle coppie di controllo: %f\n ", tau_cntrl[k]);
                                        fprintf(file_coppie, " Valore delle coppie di controllo: %f ",
                                        tau_cntrl[k]);
                                }else if(tau_cntrl[k]<-3){
                                        tau_cntrl[k]=-3;
                                        printf("\n Valore delle coppie di controllo: %f\n", tau_cntrl[k]);
                                        fprintf(file_coppie, " Valore delle coppie di controllo: %f",
                                        tau_cntrl[k]);
                                }else{
                                        printf("\n Valore delle coppie di controllo: %f\n", tau_cntrl[k]);
                                        fprintf(file_coppie, " Valore delle coppie di controllo: %f",
                                        tau_cntrl[k]);
                                }
                        }

        //------------------------------------------------------------------------------------------------------------------//

                //Da qui dovrei ricavare q2, q3, q4
                q1_eq=(1/a)*asinh(tau_cntrl[0]/(2*k_m*cosh(a*preset1)))+q1 ;
                //printf("\n Value nuovo q: %f\n",q1);
                q2_eq=(1/a)*asinh(tau_cntrl[1]/(2*k_m*cosh(a*preset2)))+q2;
                printf("\n Value nuovo q2_eq: %f\n",q2_eq);
                q3_eq=(1/a)*asinh(tau_cntrl[2]/(2*k_m*cosh(a*preset3)))+q3;
                //printf("\n Value nuovo q: %f\n",q3);
                q4_eq=(1/a)*asinh(tau_cntrl[3]/(2*k_m*cosh(a*preset4)))+q4;
                //printf("\n Value nuovo q: %f\n",q4);


                q_old[0]=q1;
                q_old[1]=q2;
                q_old[2]=q3;
                q_old[3]=q4;


                //sono i valori di q1
                value_new_con1[0]=(q1_eq+preset1)*(180/3.1415)*DEG_TICK_MULTIPLIER;
                value_new_con1[1]=(q2_eq+preset2)*(180/3.1415)*DEG_TICK_MULTIPLIER;
                value_new_con1[2]=(q3_eq+preset3)*(180/3.1415)*DEG_TICK_MULTIPLIER;
                value_new_con1[3]=(q4_eq+preset4)*(180/3.1415)*DEG_TICK_MULTIPLIER;

                //sono i valori delle q2
                value_new_con2[0]=(q1_eq-preset1)*(180/3.1415)*DEG_TICK_MULTIPLIER;
                value_new_con2[1]=(q2_eq-preset2)*(180/3.1415)*DEG_TICK_MULTIPLIER;
                value_new_con2[2]=(q3_eq-preset3)*(180/3.1415)*DEG_TICK_MULTIPLIER;
                value_new_con2[3]=(q4_eq-preset4)*(180/3.1415)*DEG_TICK_MULTIPLIER;



                for(h=2;h<NUM_QB;h++){

                        if( value_new_con1[h]>(DEFAULT_SUP_LIMIT/pow(2, DEFAULT_RESOLUTION) -
                        DEFAULT_STIFFNESS*DEG_TICK_MULTIPLIER)){
                                value_new_con1[h]=(DEFAULT_SUP_LIMIT/pow(2, DEFAULT_RESOLUTION) -
                                DEFAULT_STIFFNESS*DEG_TICK_MULTIPLIER);
                printf("\nValori fuorisoglia corretti!!\n");
```

```
                        }else if(value_new_con1[h]< (DEFAULT_INF_LIMIT/pow(2, DEFAULT_RESOLUTION) +
                        DEFAULT_STIFFNESS*DEG_TICK_MULTIPLIER)){
                                value_new_con1[h]= (DEFAULT_INF_LIMIT/pow(2, DEFAULT_RESOLUTION) +
                                DEFAULT_STIFFNESS*DEG_TICK_MULTIPLIER);
                printf("\nValori fuorisoglia corretti!!\n");
                        }

                        if( value_new_con2[h]>(DEFAULT_SUP_LIMIT/pow(2, DEFAULT_RESOLUTION) -
                        DEFAULT_STIFFNESS*DEG_TICK_MULTIPLIER)){
                                value_new_con2[h]=(DEFAULT_SUP_LIMIT/pow(2, DEFAULT_RESOLUTION) -
                                DEFAULT_STIFFNESS*DEG_TICK_MULTIPLIER);
                printf("\nValori fuorisoglia corretti!!\n");
                        }else if(value_new_con2[h]< (DEFAULT_INF_LIMIT/pow(2, DEFAULT_RESOLUTION) +
                        DEFAULT_STIFFNESS*DEG_TICK_MULTIPLIER)){
                printf("\nValori fuorisoglia corretti!!\n");
                                value_new_con2[h]= (DEFAULT_INF_LIMIT/pow(2, DEFAULT_RESOLUTION) +
                                DEFAULT_STIFFNESS*DEG_TICK_MULTIPLIER);
                        }

                }
                if( value_new_con1[3]>100)
                        value_new_con1[3]=100;
                if( value_new_con2[3]> 100)
                        value_new_con2[3]=100;
                if( value_new_con1[3]<-100)
                        value_new_con1[3]=-100;
                if( value_new_con2[3]< -100)
                        value_new_con2[3]=-100;


                value_new_con1[2]=0;

                value_new_con2[2]=0;
        counter_request++;


        }
}
```

———————"Gestori_Read_Write.h"———————

```
#ifndef QBMOVE_GESTORI_H_INCLUDED
#define QBMOVE_GESTORI_H_INCLUDED

#include <pthread.h>

#define NUM_ELEM 4
#define BUFFER_SIZE 32
#define NUM_QB 4



typedef struct{
        int id;
        int dim;
        int command;
        char buffer_read[BUFFER_SIZE];
}package_read;

typedef struct{

        int id; //dopo l'inizializzazione della struttura non viene pi  modificato.
        int buffer_aux[BUFFER_SIZE];
        int dim_pck;
        int command;
}package_to_send;

typedef struct{
        int ident;
        int command;
        package_read array_read[NUM_ELEM];      //   un vettore di strutture i cui campi sono ancora un vettore,
        la sua dimensione e                                                             l'id del cubo
        int tail;       //indica il nuovo elemento inserito
        int head;       //indica l'elemento da estrarre
        pthread_mutex_t mutex;
}buffer_read;

typedef struct{

        package_to_send arraystr[NUM_ELEM];     //   un vettore di strutture i cui campi sono un vettore e la sua
        dimensione
        int tail;       //indica il nuovo elemento da inserire
        int head;       //indica l'alemento da estrarre
        pthread_mutex_t mutex;
}buffer_write;

typedef struct{
        int Id;
        int CMD;
        float value[BUFFER_SIZE];
        float dim;
}value_read;


int create_thread_Read(int id, int cmd);
int create_thread_Write(int id);
void GestoreRead(void *arg);
```

```
void GestoreWrite(void *l);
int insert_New_element_write(package_to_send *New_pck_to_send, int reply);
int extract_New_element_read(int id, int cmd, package_read *New_package);

#endif
```

"Gestori_Read_Write.c"

```
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include <ctype.h>
#include <time.h>
#include <semaphore.h>


#if (defined(_WIN32) || defined(_WIN64))
    #include <windows.h>
#endif

#if !(defined(_WIN32) || defined(_WIN64))
    #include <unistd.h>  /* UNIX standard function definitions */
    #include <fcntl.h>   /* File control definitions */
    #include <errno.h>   /* Error number definitions */
    #include <termios.h> /* POSIX terminal control definitions */
    #include <sys/ioctl.h>
    #include <dirent.h>
    #include <sys/time.h>
    #include <stdlib.h>
#endif

#if !(defined(_WIN32) || defined(_WIN64)) && !(defined(__APPLE__))
    #include <linux/serial.h>
#endif


#include "Gestori_Read_Write.h"
#include "qbmove_com.h"


#if (defined(_WIN32) || defined(_WIN64))
    // windows stuff

    #define usleep(X) Sleep((X) / 1000)
#elif (defined(__APPLE__))
    // apple stuff
#else
    // linux stuff
#endif



#define NUM_OF_SENSORS 3
#define NUM_OF_MOTORS 2
#define NUM_CMD 4
#define NUM_QB 4

buffer_write buffer_write_t;
buffer_write buffer_write_r;

sem_t semGW, semGR;
pthread_attr_t attr_read, attr_write;
struct sched_param param_read, param_write;

int cnt;        //mi serve per calcolare l'intervallo temporale
int cntr;
buffer_read buffer_read_reply[NUM_CMD];
extern comm_settings comm_settings_t;

extern long timevaldiff(struct timeval *starttime, struct timeval *finishtime);

struct timeval start_newcmd, now_cmd, now_read, start_now_read, newcmd_old, oldnow_read, new_cmdf, now_readf;
//struct timeval, start_RW, finish_RW
int create_thread_Read(int id, int cmd){

        int r, arg;
        int g;
        pthread_t th1;
        static int count_r=0;

        if(count_r==0){
                if(id==1){
                        for(r=0;r<NUM_CMD;r++){
                                buffer_read_reply[r].ident=0;
                                buffer_read_reply[r].command=0;
                        }
                }

                buffer_write_r.tail=-1;
                buffer_write_r.head=-1;
                g=pthread_mutex_init(&(buffer_write_r.mutex),0);

                for(r=0;r<NUM_CMD;r++){

                        if(buffer_read_reply[r].ident==0 && buffer_read_reply[r].command==0){
                                buffer_read_reply[r].ident=id;
                                buffer_read_reply[r].command=cmd;
```

```
                                //printf("buffer_read_reply.ident== %d !! \n",buffer_read_reply[r].ident);
                                buffer_read_reply[r].tail=-1;
                                buffer_read_reply[r].head=-1;
                                g=pthread_mutex_init(&(buffer_read_reply[r].mutex),0);
                                if(g==0)
                                        printf("Mutex lettura  inizializzato!! \n");
                                else
                                        printf("Mutex lettura  non inizializzato!! \n");
                                break;
                        }
                }
        pthread_attr_init(&attr_read);
                param_read.sched_priority = 4;
                if(g==0)
                        printf("Mutex scrittura/lettura inizializzato!! \n");
                else
                        printf("Mutex scrittura/lettura non inizializzato!! \n");
                if(id==1){
            pthread_attr_setschedparam(&attr_read, &param_read);
                        r=pthread_create(&th1, &attr_read, GestoreRead,(void*)arg);
                        if(r != 0){
                                printf("Errore nella creazione del secondo thread\n");
                                return -1;
                        }else{
                                g=sem_init(&semGR,1,0);
                                if(g==0)
                                        printf("Semaforo inizializzato!! \n");
                                else
                                        printf("Semaforo non inizializzato!! \n");

                        }
                }
        }else{

                for(r=0;r<NUM_CMD;r++){

                        if(buffer_read_reply[r].ident==0 && buffer_read_reply[r].command==0){
                                buffer_read_reply[r].ident=id;
                                buffer_read_reply[r].command=cmd;
                                //printf("buffer_read_reply.ident== %d !! \n",buffer_read_reply[r].ident);
                                buffer_read_reply[r].tail=-1;
                                buffer_read_reply[r].head=-1;
                                g=pthread_mutex_init(&(buffer_read_reply[r].mutex),0);
                                if(g==0)
                                        printf("Mutex lettura  inizializzato!! \n");
                                else
                                        printf("Mutex lettura  non inizializzato!! \n");
                                break;
                        }
                }

                return count_r;
        }


        count_r++;
        return 0;
}

int create_thread_Write(int id){

        int r, l;
        pthread_t th2;
        static int count_w=0;

        if(count_w==0){
                if(id==1){
                        buffer_write_t.tail=-1;
                        buffer_write_t.head=-1;
                        int g=pthread_mutex_init(&(buffer_write_t.mutex),0);
                        if(g==0)
                                printf("Mutex scrittura inizializzato!! \n");
                        else
                                printf("Mutex scrittura non inizializzato!! \n");
                pthread_attr_init(&attr_write);
                    param_write.sched_priority = 4;
            pthread_attr_setschedparam(&attr_write, &param_write);
                        r=pthread_create(&th2, &attr_write, GestoreWrite,(void*)l);
                        if(r != 0){
                                printf("Errore nella creazione del secondo thread\n");
                                return -1;
                        }else{
                                g=sem_init(&semGW,1,0);
                                if(g==0)
                                        printf("Semaforo inizializzato!! \n");
                                else
                                        printf("Semaforo non inizializzato!! \n");

                        }
                }
        }else{
                return count_w;
        }
        count_w++;
        return 0;
}

void GestoreRead(void *arg){
```

```
    package_to_send pck_r;
    int hh,k;
    char pck_a[BUFFER_SIZE];
    char pck[BUFFER_SIZE];
    int dim;
    int ret, m;
    static float usecr;
static int count_post=4;


    while(1){
            //printf("Gestore per le letture partito e messo in attesa!!\n");
            sem_wait(&semGR);
            gettimeofday(&now_read, NULL);
            if(cntr==0){
                    oldnow_read=start_now_read;
                    gettimeofday(&start_now_read, NULL);
            }
            printf("\nPartenza thread Read/Write: secondi:%d , microsecondi: %d\n", now_read.tv_sec,
            now_read.tv_usec);

                //printf("Gestore della lettura sveglio!!\n");
                pthread_mutex_lock(&buffer_write_r.mutex);
                if(buffer_write_r.tail!=-1 && buffer_write_r.head!=-1){
                        //printf("Valore indice head: %d \n",buffer_write_r.head);
                        pck_r.id=buffer_write_r.arraystr[buffer_write_r.head].id;
                        pck_r.command=buffer_write_r.arraystr[buffer_write_r.head].command;
                        pck_r.dim_pck=buffer_write_r.arraystr[buffer_write_r.head].dim_pck;
                        for(hh=0;hh<pck_r.dim_pck;hh++){
                                pck_r.buffer_aux[hh]=buffer_write_r.arraystr[buffer_write_r.head].buffer_aux[hh];
                                pck_a[hh]=pck_r.buffer_aux[hh];
                                //printf("Valori in pck_r.buffer: %d \n",pck_r.buffer_aux[hh]);
                        }
                        //printf("Dimensione pacchetto: %d \n", pck_r.dim_pck);
            if(buffer_write_r.head==buffer_write_r.tail){
                buffer_write_r.head=-1;
                buffer_write_r.tail=-1;
            }else{
                buffer_write_r.head=(buffer_write_r.head+1)%NUM_ELEM;
            }
                        //printf("Valore indice head dopo l'estrazione: %d \n",buffer_write_r.head);

                        pthread_mutex_lock(&comm_settings_t.mut_ser);
                        //printf("Scrittura del gestore per le letture!! \n");
                        //printf("Dimensione pacchetto da scrivere: %d \n", pck_r.dim_pck);
                        //printf("Sto spedendo ID: %d \n", pck_r.id);
                        //gettimeofday(&start_RW, NULL);
                        Write_RS485(pck_a,pck_r.dim_pck);
                        usleep(1000);
                        //printf("Chiamo la funzione RS485!!\n");
                        //printf("Sto leggendoID: %d \n", pck_r.id);
                        //ret=RS485read_2(pck_r.id, pck, &dim);
                        ret=RS485read( pck_r.id, pck, &dim);
                        //gettimeofday(&finish_RW, NULL);
                        //usecr=(finish_RW.tv_sec-start_RW.tv_sec)*1000000;
                            //usecr+=(finish_RW.tv_usec-start_RW.tv_usec);
                        //usecr=usecr/1000000;

                        //printf("\nesecuzione thread Read/Write solo funzioni: %f\n", usecr);
                        //printf("Valore ret_Read: %d \n",ret);
                        pthread_mutex_unlock(&comm_settings_t.mut_ser);

                        /*for(hh=0;hh<dim;hh++){
                                printf("Valori Letti dalla read del Gestore_Read: %d\n",pck[hh]);

                        }*/

                        if(ret!=-1){
                                //printf("Inserisco i dati nel buffer delle letture!! \n");
                                //-----------Inserisco i dati letti nel buffer--------------//
                                for(k=0;k<NUM_CMD;k++){
                                        pthread_mutex_lock(&buffer_read_reply[k].mutex);
                                        //printf("valori di ident: %d e id:
                                        %d\n",buffer_read_reply[k].ident,pck_r.id);
                                        if(buffer_read_reply[k].ident==pck_r.id
                                            && buffer_read_reply[k].command==pck_r.command){
                                                /*printf("valori di k: %d, ident: %d e id: %d\n",k,
                                                    buffer_read_reply[k].ident,pck_r.id);*/
                                                buffer_read_reply[k].array_read[
                                                    (buffer_read_reply[k].tail+1)%NUM_ELEM].dim=dim;
                                                for(hh=0;hh<dim;hh++){
                                                        buffer_read_reply[k].array_read[
                                                            (buffer_read_reply[k].tail+1)%NUM_ELEM].buffer_read[hh]=pck[hh];
                                                }
                                                buffer_read_reply[k].tail=(buffer_read_reply[k].tail+1)%NUM_ELEM;
                                                if(buffer_read_reply[k].head==-1){
                                                        buffer_read_reply[k].head=buffer_read_reply[k].tail;
                                                }
                                                pthread_mutex_unlock(&buffer_read_reply[k].mutex);
                                                break;
                                        }else{
                                                pthread_mutex_unlock(&buffer_read_reply[k].mutex);
                                        }
                                }
                                //-----------------------------------------------------------//
                        }else{
                                printf("Lettura non effettuata!! \n");
                        }
                    cntr++;
```

```
                }

            //printf("\nValore contatore: %d\n", cntr);
            if((usecr*1000000)>2000){
                //printf("\nValore durata lettura: %f\n", usecr);
                while(cntr!=4){
                    /*printf("\nValore tail: %d e head: %d\n",
                                    buffer_write_r.tail, buffer_write_r.head );*/
                    if(buffer_write_r.head==buffer_write_r.tail){
                        buffer_write_r.head=-1;
                        buffer_write_r.tail=-1;
                    }else{
                        buffer_write_r.head=(buffer_write_r.head+1)%NUM_ELEM;
                    }
                    cntr++;
                }

            }
            count_post--;
                    gettimeofday(&now_readf, NULL);
                    usecr=(now_readf.tv_sec-now_read.tv_sec)*1000000;
                usecr+=(now_readf.tv_usec-now_read.tv_usec);
                    usecr=usecr/1000000;

                    printf("\nfine esecuzione thread Read/Write: secondi:%d,
                        microsecondi: %d\n", now_readf.tv_sec, now_readf.tv_usec);
                    printf("\nesecuzione thread Read/Write: %f\n", usecr);
            pthread_mutex_unlock(&buffer_write_r.mutex);
            //printf("\nvalore count_post: %d\n", count_post);
            if(cntr==NUM_QB && count_post==0){
                m=sem_post(&semGW);
                count_post=4;
            }

        }

}


void GestoreWrite(void *l){

        package_to_send pck;
        int h,i;
        char pck_a[BUFFER_SIZE];
        float usect;


        while(1){
                //printf("Gestore per le scritture partito!!\n");
                sem_wait(&semGW);
                //printf("Scrittore: sono stato svegliato!! \n");
        gettimeofday(&now_cmd, NULL);
                printf("\nInizio esecuzione thread Write: secondi:%d ,
                microsecondi: %d\n", now_cmd.tv_sec, now_cmd.tv_usec);
        pthread_mutex_lock(&buffer_write_t.mutex);
        for(i=0;i<NUM_QB;i++){
                    if(cnt==0){
                            newcmd_old=start_newcmd;
                            gettimeofday(&start_newcmd, NULL);
                usect=(start_newcmd.tv_sec-newcmd_old.tv_sec)*1000000;
                    usect+=(start_newcmd.tv_usec-newcmd_old.tv_usec);
                //if(usect<7000){
                    //while(timevaldiff(&newcmd_old, &start_newcmd)<7000){
                            //gettimeofday(&start_newcmd, NULL);
                    //}
                //}

                    }

                    //printf("!!!!!!!!!!!!!!Scrittore: buffer_write_t.mutex!!!!!!!!!!!!!!!!\n");

                    if(buffer_write_t.tail!=-1 && buffer_write_t.head!=-1){
                            pck.dim_pck=buffer_write_t.arraystr[buffer_write_t.head].dim_pck;
                            pck.command=buffer_write_t.arraystr[buffer_write_t.head].command;
                            pck.id=buffer_write_t.arraystr[buffer_write_t.head].id;
                            for(h=0;h<pck.dim_pck;h++){
                                    pck.buffer_aux[h]=buffer_write_t.arraystr[buffer_write_t.head].
                                        buffer_aux[h];
                                    pck_a[h]=pck.buffer_aux[h];
                            }
                    if(buffer_write_t.head==buffer_write_t.tail){
                            buffer_write_t.head=-1;
                            buffer_write_t.tail=-1;
                        }else{
                            buffer_write_t.head=(buffer_write_t.head+1)%NUM_ELEM;
                        }

                                //-------------Invio i dati nella seriali-------------------------//
                                /*printf("Valori tail: %d and head %d!! \n",buffer_write_t.tail,
                                    buffer_write_t.head);*/
                                //printf("Scrittore: buffer_write_t.mutex fi!! \n");
                                //printf("****************Valore id %d********************** \n",pck.id);
                                pthread_mutex_lock(&comm_settings_t.mut_ser);
                                //printf("Sto spedendo ID: %d \n", pck.id);
                                Write_RS485(pck_a,pck.dim_pck);
                                usleep(100);
                                //printf("Sto uscendo dal Gestore_Write!!\n");
                                pthread_mutex_unlock(&comm_settings_t.mut_ser);
                                //------------------------------------------------------------//
```

```
                }

                cnt++;
                //printf("!!!!!!!!!!!!!!Scrittore: sbloccata la seriale!!!!!!!!!!!!!!! \n");
        }
        pthread_mutex_unlock(&buffer_write_t.mutex);
        gettimeofday(&new_cmdf, NULL);
                usect=(new_cmdf.tv_sec-now_cmd.tv_sec)*1000000;
        usect+=(new_cmdf.tv_usec-now_cmd.tv_usec);
                usect=usect/1000000;
                printf("\nfine esecuzione thread Write: secondi:%d ,
                microsecondi: %d\n", new_cmdf.tv_sec, new_cmdf.tv_usec);
                printf("\nesecuzione thread Write: %f\n", usect);

        }
}


int insert_New_element_write(package_to_send *New_pck_to_send, int reply){

        int dim;
        int h;

        dim=New_pck_to_send->dim_pck;
        if(reply==0){
                pthread_mutex_lock(&buffer_write_t.mutex);
                //if(buffer_write_t.tail<NUM_ELEM){
        if(buffer_write_t.head!=(buffer_write_t.tail+1)%NUM_ELEM){
                        for(h=0;h<dim;h++){
                                buffer_write_t.arraystr[(buffer_write_t.tail+1)%NUM_ELEM].
                                buffer_aux[h]=New_pck_to_send->buffer_aux[h];
                                /*printf("Valori inseriti in coda per la scrittura: %d \n",
                                buffer_write_t.arraystr[(buffer_write_t.tail+1)%NUM_ELEM].buffer_aux[h]);*/
                        }
                        buffer_write_t.arraystr[(buffer_write_t.tail+1)%NUM_ELEM].id=New_pck_to_send->id;
                        buffer_write_t.arraystr[(buffer_write_t.tail+1)%NUM_ELEM].command=New_pck_to_send->command;
                        buffer_write_t.arraystr[(buffer_write_t.tail+1)%NUM_ELEM].dim_pck=dim;
                        //printf("Dimensione pacchetto in coda per la scrittura: %d
                        \n",buffer_write_t.arraystr[(buffer_write_t.tail+1)%NUM_ELEM].dim_pck);
                        buffer_write_t.tail=(buffer_write_t.tail+1)%NUM_ELEM;
                        if(buffer_write_t.head==-1){
                                buffer_write_t.head=buffer_write_t.tail;
                        }
                        //printf("Valori indici coda per la scrittura: head=%d    tail=%d
                        \n",buffer_write_t.head,buffer_write_t.tail);
                        pthread_mutex_unlock(&buffer_write_t.mutex);
                }else{
                        pthread_mutex_unlock(&buffer_write_t.mutex);
                        return -1;
                }

        }else{
                pthread_mutex_lock(&buffer_write_r.mutex);
                //if(buffer_write_r.tail<NUM_ELEM){
        if(buffer_write_r.head!=(buffer_write_r.tail+1)%NUM_ELEM){
                        for(h=0;h<dim;h++){
                                buffer_write_r.arraystr[(buffer_write_r.tail+1)%NUM_ELEM].buffer_aux[h]=
                                New_pck_to_send->buffer_aux[h];
                                /*printf("Valori inseriti in coda per la scrittura/lettura: %d \n",
                                buffer_write_r.arraystr[(buffer_write_r.tail+1)%NUM_ELEM].buffer_aux[h]);*/
                        }
                        buffer_write_r.arraystr[buffer_write_r.tail+1].id=New_pck_to_send->id;
                        buffer_write_r.arraystr[buffer_write_r.tail+1].command=New_pck_to_send->command;
                        buffer_write_r.arraystr[buffer_write_r.tail+1].dim_pck=dim;
                        /*printf("Dimensione pacchetto in coda per la scrittura/lettura: %d \n",
                        buffer_write_r.arraystr[(buffer_write_r.tail+1)%NUM_ELEM].dim_pck);*/
                        buffer_write_r.tail=(buffer_write_r.tail+1)%NUM_ELEM;
                        if(buffer_write_r.head==-1){
                                buffer_write_r.head=buffer_write_r.tail;
                        }
                        /*printf("Valori indici coda per la scrittura/lettura: head=%d    tail=%d \n",
                        buffer_write_r.head,buffer_write_r.tail);*/
                        pthread_mutex_unlock(&buffer_write_r.mutex);
                }else{
                        pthread_mutex_unlock(&buffer_write_r.mutex);
                        return -1;
                }
        }

        return 0;

}

int extract_New_element_read(int id, int cmd, package_read *New_package){

        int k, i;

        for(i=0;i<NUM_CMD;i++){
                pthread_mutex_lock(&buffer_read_reply[i].mutex);
                if(buffer_read_reply[i].ident==id && buffer_read_reply[i].command==cmd){

                        if(buffer_read_reply[i].head!=-1){
                                New_package->id=buffer_read_reply[i].array_read[buffer_read_reply[i].head].id;
                                New_package->command=
                                buffer_read_reply[i].array_read[buffer_read_reply[i].head].command;
                                New_package->dim=buffer_read_reply[i].array_read[buffer_read_reply[i].head].dim;
                                for(k=0;k<New_package->dim;k++){
                                        /*printf("valori nella FiFo lettura: %d \n",
                                        buffer_read_reply.array_read[buffer_read_reply.head].buffer_read[k]);*/
```

```
                                        New_package->buffer_read[k]=
                                        buffer_read_reply[i].array_read[buffer_read_reply[i].head].buffer_read[k];
                                        //printf("valori nella struct lettura: %d
                                        \n",New_package->buffer_read[k]);
                                }
                if(buffer_read_reply[i].head==buffer_read_reply[i].tail){
                    buffer_read_reply[i].head=-1;
                    buffer_read_reply[i].tail=-1;
                }else{
                    buffer_read_reply[i].head=(buffer_read_reply[i].head+1)%NUM_ELEM;
                }

                                pthread_mutex_unlock(&buffer_read_reply[i].mutex);
                                break;
                        }else{
                                pthread_mutex_unlock(&buffer_read_reply[i].mutex);
                                return -1;
                        }
                }else{
                        pthread_mutex_unlock(&buffer_read_reply[i].mutex);
                }
        }

        return 0;
}
```

Bibliografia

[1] A. Bicchi and G. Tonietti, *"Fast and soft arm tactics: Dealing with the safety–performance tradeoff in robot arms design and control"*, IEEE Robotics and Automation Magazine, Vol. 11, No. 2, June, 2004.

[2] G. Tonietti and R. Schiavi and A. Bicchi, *"Design and Control of a Variable Stiffness Actuator for Safe and Fast Physical Human/Robot Interaction"*, In Proc. IEEE Int. Conf. on Rob. & Aut., ICRA, 2005, pp.528-533.

[3] R. Schiavi, G. Grioli, S. Sen, and A. Bicchi, *"Vsa-II: A novel prototype of variable stiffness actuator for safe and performing robots interacting with humans"*, in Proc. IEEE Int. Conf. on Robotics and Automation, 2008, pp. 2171 – 2176.

[4] M. Catalano, G. Grioli, M. Garabini, F. Bonomo,M. Mancini, N. Tsagarakis and A. Bicchi, *"VSA-CubeBot: a modular variable stiffness platform for multiple degrees of freedom robots."*.

[5] *"http://www.udoo.org/udoo-neo/"*

[6] B. Morelli, R. Schiavi, C. Scordino, P. Gai,M. Di Natale,*"Automatic generation of controls code from models for real-time Linux platforms."*.

[7] *"E4Coder Reference Manual"*, Evidence Srl.

[8] T.G. Moreira, M. Wehrmeister, C. Pereira, J. Pétin and E. Levrat, *"Automatic Code Generation for Embedded Systems From UML Specifications to VHDL code"*,IEEE Conference Publications, 2010.

[9] H. Sporer, G. Macher, E. Armengaud and C. Kreiner, *"Incorporation of Model-based System and Software Development Environments"*, IEEE Conference Publications, 2015.

[10] M. Rashid, M.W. Anwar and A.M. Khan, *"Identification of Trends for Model Based Development of Embedded Systems"*, IEEE Conference Publications, 2015.

[11] L. Tan, J. Kim, O. Sokolsky and I. Lee, *"Model-based Testing and Monitoring for Hybrid Embedded Systems"*, IEEE Conference Publications.

[12] Qiang. Yu, H. Wei, M. Liu and T wang, *"A Novel Multi-OS Architecture for Robot Application"*, IEEE Conference Publications, 2011.

[13] G. Heiser, *"The Role of Virtualization in Embedded Systems"*.

[14] P. Schnarz, J. Wietzke, I. Stengel, *"Co-Processor Aided Attack on Embedded Multi-OS Environments"*, IEEE Conference Publications.