UNIVERSITÀ DI PISA

FACOLTÀ DI INGEGNERIA

LAUREA SPECIALISTICA IN INGEGNERIA DELL'AUTOMAZIONE

Tesi di Laurea

Sviluppo di un software di analisi e sintesi per il controllo di impianti su rete ethernet industriale

Candidato:

Stefano MELANI

Relatore:

Prof. Ing. Antonio BICCHI
Dipartimento di Ingegneria dell'Informazione

Controrelatore:

Prof. Ing. Andrea CAITI
Dipartimento di Ingegneria dell'Informazione

Sessione di Laurea del 04/10/2013
Anno Accademico 2012/2013

Indice

Introduzione

Il collegamento di sensori e attuatori ai nodi controllori rappresenta spesso una fase molto delicata sia nella progettazione che nell'installazione di sistemi automatizzati. Tale aspetto si traduce infatti nella ricerca di un equilibrio tra vincoli strutturali (dettati dagli ingombri dell'impianto e dagli spazi fisicamente disponibili) e costi di posa in opera e manutenzione delle linee (per lo più dedicate). Tali difficoltà sono poi spesso amplificate da altri fattori, come il bisogno di mantenere dei setup precedentemente fissati (come accade in caso di installazioni su impianti preesistenti) o la necessità di controllare impianti particolarmente estesi o sottoposti ad ampliamenti o cambiamenti strutturali. Negli ultimi anni il mondo dell'automazione ha cercato di alleggerire tali problematiche tentando di abbandonare le connessioni point-to-point ed applicando le normali tecniche di controllo a sistemi interconnessi attraverso reti informatiche (da adesso indicati come NCS - Network Controlled System). Tali infrastrutture presenterebbero infatti una elevata flessibilità e scalabilità per impianti distribuiti su ampie superfici e risulterebbero molto più veloci da installare, ove non fossero addirittura già presenti per il normale scambio informativo aziendale. La condivisibilità di tali collegamenti (per scopi controllistici e informativi) rappresenta però anche il motivo per cui gli NCS non sono ancora riusciti a soppiantare le precedenti e più costose modalità di installazione. Nell'architettura NCS, infatti, le tradizionali teorie del controllo non sono sempre applicabili; i pacchetti in transito sulla rete subiscono dei ritardi di trasmissione collegati al traffico dati istantaneamente presente e quin-

di non sempre prevedibili o limitabili. La condivisione delle reti informatiche in modalità multiuser e multitasking impone inoltre che anche i pacchetti inerenti le informazioni di controllo siano sottoposti alle tecniche di schedulazione, rendendo pressoché impossibile il contemporaneo invio di pacchetti da diversi nodi. Supponiamo, ad esempio, di dover elaborare un segnale di controllo basato su un set di misure contemporaneamente effettuate da un certo numero di sensori dislocati nel sistema di riferimento. Le condizioni di lavoro precedentemente esposte faranno sì che, ad intervalli di tempo non del tutto prevedibili, solo alcuni sensori potranno inviare le proprie rilevazioni; il controllore riceverà quindi una sequenza di pacchetti ritardati rispetto alla reale misurazione che potrebbero anche non rappresentare il totale set di misure istantaneamente necessarie alla legge di controllo (rendendo praticamente inattendibile il segnale generato). Un problema similare sarà poi riscontrato, sempre a causa dei ritardi di trasmissione, anche nella ricezione dei pacchetti di controllo da parte dei nodi attuatori, per i quali non sarà predicibile a priori il tempo di risposta.

Nel Capitolo 1, dopo aver ripreso le problematiche sopra descritte, viene illustrata una tecnica di controllo basata sulla generazione di segnali di comando attraverso elaborazioni feedforward di un modello, più o meno accurato, dell'impianto da controllare (similarmente a quanto previsto in [6][8][7]). Tale tecnica, definita PBC (Packet Based Control) ed ampiamente descritta in [5][2][4], è capace di elaborare leggi di controllo (statiche o dinamiche) per sistemi non lineari, prevedendo inoltre l'utilizzo di osservatori di stato esponenzialmente convergenti.

Nel Capitolo 2 viene illustrata la struttura dell'ambiente software utilizzato e come questo, attraverso l'editazione di file testuali, sia capace di fornire supporto sia durante il dimensionamento della legge di controllo PBC, sia nella verifica delle performance raggiunte con la sua applicazione.

Il Capitolo 3, dopo una prima descrizione delle modalità di configurazione

del software elaborato, è stato dedicato alla presentazione dei due esempi applicativi realizzati attraverso il suo utilizzo: la stabilizzazione di un pendolo di Furuta ed il controllo di una macchina di taglio CNC.
Il Capitolo 4 viene infine dedicato al tuning di alcuni parametri funzionali e alla descrizione della loro interazione.

Capitolo 1

NCS e PBC

In questo capitolo verranno introdotte le difficoltà applicative delle normali tecniche di controllo ai sistemi NCS e come queste siano state risolte attraverso lo sviluppo della tecnica PBC ([5][2][4].

1.1 NCS - Networked Control System

Con l'acronimo NCS (Networked Controll System) vengono indicate quelle tipologie di sistemi per i quali l'anello di controllo è chiuso attraverso l'utilizzo di una rete informatica; per cui lo scambio di informazioni tra sensori, attuatori e centri di controllo avviene attraverso le strutture dati comunemente definite "pacchetti" ed il canale di collegamento tra i vari nodi è praticamente unico e condiviso.
Questa architettura risulta spesso preferibile alla canonica tipologia point-to-point cablata attraverso installazione di linee dedicate, proprio per la maggior facilità di installazione (dovuta all'utilizzo di un'unica linea condivisa) e la considerevole adattabilità ad impianti molto estesi; tuttavia, proprio a causa del contemporaneo utilizzo di un canale condiviso, presenta grosse difficoltà nell'utilizzo delle classiche tecniche di controllo.
Il flusso dei pacchetti su una rete è sottoposto a due tipologie di ritardi: il primo è quello imposto dalle così dette "tecniche di schedulazione" ed il

secondo è direttamente collegato al congestionamento del canale utilizzato.
Mentre il primo tipo di ritardo è spesso limitabile (o comunque modellabile) attraverso tecniche di schedulazione di tipo hard o soft real-time, il secondo risulta del tutto incontrollabile, data la sua istantanea casualità. Il ritardo globale prodotto sulla trasmissione è quindi fortemente aleatorio e non può essere efficacemente compensato con nessun tipo di tecnica normalmente utilizzata per il controllo di sistemi.
Un altro importante problema è poi rappresentato dalla possibilità di far trasmettere un solo nodo per volta; tale situazione obbliga infatti il controllore ad elaborare i segnali di controllo sulla base di informazioni istantaneamente incomplete o non sincronizzate.
Si pensi alla situazione nella quale n sensori debbano inviare contemporaneamente al controllore le misure effettuate sull'impianto al tempo τ. Dato che un solo sensore alla volta potrà accedere alla rete per la propria trasmissione avremo che il controllore dovrà attendere almeno n istanti prima di aver ottenuto un set completo di informazioni da elaborare; a questo punto si dovrà scegliere se attendere il set completo, ritardando ulteriormente l'azione di controllo o se elaborare quest'ultima su un set incompleto.
Riassumendo quanto sopra potremo allora sintetizzare i limiti del controllo su reti con i seguenti 5 punti:

(i) intervallo variabile tra i vari istanti di invio dati (conseguenza della schedulazione)

(ii) ritardi variabili di trasmissione (conseguenza della congestione della rete)

(iii) limiti comunicativi derivati dalla condivisione della rete da parte di più nodi e dal fatto che solo uno per volta può trasmettere

(iv) possibile perdita di dati derivata dalla spesso scarsa affidabilità della rete

(v) limiti indotti sulla quantizzazione dei segnali dal bilanciamento tra banda disponibile e velocità di comunicazione richiesta

A fronte di queste problematiche la tecnica di controllo PBC (Racket Based Control), sulla base della quale è stato realizzato questo software, si propone di gestire i primi tre punti del precedente elenco.

1.2 Tecniche di controllo

Come abbiamo precedentemente illustrato, l'introduzione di una rete di comunicazione nel loop di controllo di un sistema provoca un'alterazione sia dei segnali proveniente dal sistema (uscite y), sia di quelli ad essi diretto (controllo u). Con riferimento alla figura 1.1 accade infatti che, ad ogni istante, il segnale $\hat{u}(t)$ in ingresso al sistema non sia più identico al segnale $u(t)$ generato dal controllore, e dualmente che il segnale $y(t)$ in uscita dal sistema non sia più esattamente noto al controllore il quale potrà avere accesso solamente ad una versione $\hat{y}(t)$.

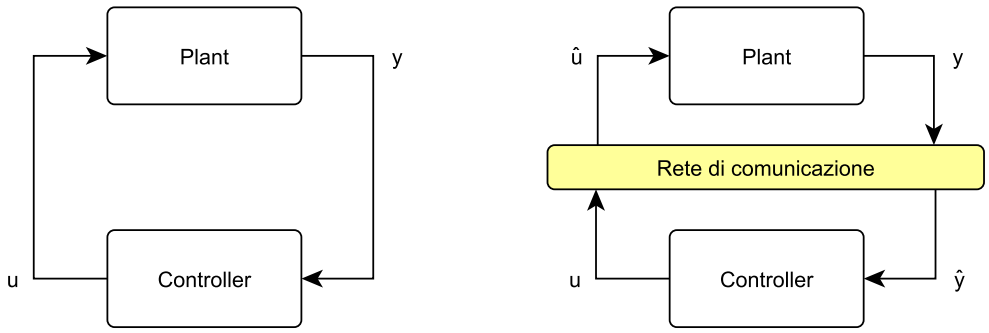

Figura 1.1: Introduzione della rete nel loop di controllo

Tali imperfezioni si ripercuoteranno sul sistema di controllo attraverso la generazione di due errori:

$$e_y \triangleq \hat{y} - y \tag{1.1}$$

$$e_u \triangleq \hat{u} - u \tag{1.2}$$

che possono spesso degradare le prestazioni del loop di controllo fino addirittura all'instabilità.
Dato che tali imperfezioni sono praticamente sempre presenti in un sistema di controllo su rete, è molto importante capire come queste dinamiche influenzino la stabilità e le performance del sistema in anello chiuso ed elaborare degli strumenti di controllo che le tengano costantemente in considerazione e che siano capaci di valutarne i vari effetti.
Nell'analisi di tali strumenti, come definito in [1], è possibile distinguere tre diversi metodi di approccio alla modellazione, analisi e sintesi di sistemi di controllo per architetture NCS:

1.2.1 Approccio tempo-discreto su dati campionati

Consiste nella realizzazione di un modello discreto del sistema NCS considerato e nell'introduzione dei ritardi di rete come incertezze esponenziali in forma discreta; la legge di controllo viene poi elaborata attraverso procedure LMI capaci di garantire la robusta stabilità del sistema in ciclo chiuso.
Questo metodo è di solito applicato ad NCS con sistemi e controllori lineari dato che in tali casi i modelli discreti possono essere esattamente costruiti sulla base dei dati campionati.

1.2.2 Approccio tempo-continuo su dati campionati

Similare al metodo precedente, questo approccio si basa su un modello a tempo continuo del sistema da controllare realizzato a partire dai dati campionati.
Le considerazioni effettuate per il calcolo di una legge di controllo, basate sull'analisi di funzioni di Lyapunov-Krasovskii o sul metodo di controllo

$H\infty$, sono adatte alla gestione di sistemi con ritardi e tempi di campionamento variabili ma non considerano direttamente la perdita di pacchetti durante la trasmissione di informazioni.

1.2.3 Approccio tempo-continuo emulativo

Si basa ancora su un modello tempo-continuo del sistema da controllare elaborato attraverso i dati campionati ma, diversamente dai precedenti due approcci, questo metodo è orientato, attraverso due fasi di progetto, alla realizzazione di un controllore tempo-continuo. In un primo momento viene disegnato un controllore capace di stabilizzare il sistema in assenza di effetti di rete e successivmante si analizza per quali di tali effetti (come ad esempio ritardi di trasmissione e tempi di campionamento) la struttura NCS realizzata con tale controllore continua a mantenere un comportamento stabile.
Questo approccio è applicabile ad una vasta classe di NCS sia lineari che non lineari ed è quella a cui appartiene la tecnica di controllo PBC.

1.3 PBC - Packet Based Control

Con l'acronimo PBC viene indicata una tecnica di controllo basata su strutture dati a pacchetto (Packet Based Control), appartenente all'approccio tempo-continuo emulativo, attraverso la quale si affrontano le prime tre tipologie di problemi relative ad architetture NCS.

1.3.1 Il sistema e il suo modello

Si consideri un sistema non lineare tempo continuo del tipo

$$\dot{x}_p = f_p(x_p, u) \tag{1.3}$$

$$y = g_p(x_p), \tag{1.4}$$

dove $x_p : \mathbb{R}_{\geq 0} \to \mathbb{R}^{n_p}$ è lo stato del sistema, $y : \mathbb{R}_{\geq 0} \to \mathbb{R}^{n_y}$ è l'output e $u : \mathbb{R}_{\geq 0} \to \mathbb{R}^{n_u}$ rappresenta l'input di controllo; $f_p : \mathbb{R}^{n_p} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_p}$ e $g_p : \mathbb{R}^{n_p} \to \mathbb{R}^{n_y}$ sono funzioni localmente Lipschitziane. Per tale sistema, si assume poi che esista una retroazione dinamica di controllo del tipo

$$\dot{x}_c = f_c(x_c, y) \tag{1.5}$$

$$u = g_c(x_c, x_p, y), \tag{1.6}$$

capace, in assenza di ritardi di comunicazione, di stabilizzare (in modo globalmente esponenziale) il sistema precedente; queste sono le condizioni necessarie affinché sia possibile applicare la tecnica PBC ad una data struttura NCS.

Assunzione 1. (Proprietà GES) *L'origine del sistema* (1.3) *in ciclo chiuso con* (1.5) *è globalmente esponenzialmente stabile (GES); esiste inoltre una funzione differenziabile* $V : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ *ed un insieme di costanti* $\underline{\alpha}, \overline{\alpha}, \alpha, d > 0$ *tali che valga quanto seguenti per ogni* $x \in \mathbb{R}^n$

$$\underline{\alpha} \|x\|^2 \leq V(x) \leq \overline{\alpha} \|x\|^2$$

$$\frac{\partial V}{\partial x}(x) f(x, g(x)) \leq -\alpha \|x\|^2$$

$$\left\| \frac{\partial V}{\partial x}(x) \right\| \leq d \|x\| .$$

Assunzione 2. (Lipschitzianità locale) *Date due costanti* $R_x, R_u > 0$*, esistono due costanti* $\lambda_f, \lambda_g > 0$ [1] *tali che per ogni* $x_1, x_2 \in B_{R_x}$ *e* $u_1, u_2 \in B_{R_u}$*, valga quanto segue*

$$\|f(x_1, u_1) - f(x_2, u_2)\| \leq \lambda_f \left(\|x_1 - x_2\| + \|u_1 - u_2\| \right) \tag{1.7}$$

$$\|g(x_1) - g(x_2)\| \leq \lambda_g \|x_1 - x_2\| . \tag{1.8}$$

[1] Dove λ_g può essere scelto indipendentemente da R_u.

La tecnica PBC compensa gli effetti indotti dai ritardi di rete attraverso una predizione del comportamento del sistema controllato. Si assume quindi che il modello per (1.3) (1.4) sia conosciuto con incertezza limitata

$$\dot{\hat{x}}_p = \hat{f}_p(\hat{x}_p, u) \tag{1.9}$$

$$\hat{y} = \hat{g}_p(\hat{x}_p) \tag{1.10}$$

e che tale modello, posto in anello chiuso con il controllore (1.5) (1.6), sia descrivibile attraverso la seguente forma compatta

$$\dot{\hat{x}} = \hat{f}(\hat{x}, \hat{u}) \tag{1.11}$$

$$\hat{u} = \hat{g}(\hat{x}) \tag{1.12}$$

Le dinamiche del controllore e del modello del sistema possono allora essere riassunte con le seguenti definizioni:

$$f(x,u) \triangleq \begin{bmatrix} f_p(x_p, u) \\ f_c(x_c, g_p(x_p)) \end{bmatrix} \tag{1.13}$$

$$\hat{f}(\hat{x},\hat{u}) \triangleq \begin{bmatrix} \hat{f}_p(\hat{x}_p, \hat{u}) \\ f_c(\hat{x}_c, \hat{g}_p(\hat{x}_p)) \end{bmatrix} \tag{1.14}$$

$$g(x) \triangleq g_c(x_c, x_p, g_p(x_p)) \tag{1.15}$$

$$\hat{g}(\hat{x}) \triangleq g_c(\hat{x}_c, \hat{x}_p, \hat{g}_p(\hat{x}_p)) \tag{1.16}$$

Assunzione 3. (Limitazione delle incertezze di modello) *Dati* $R_x, R_u > 0$, *esiste una costante* $\lambda_{f\hat{f}} \geq 0$ *tali che, per ogni* $x \in B_{R_x}$ *e* $u \in B_{R_u}$,

$$\left\| f(x,u) - \hat{f}(x,u) \right\| \leq \lambda_{f\hat{f}} \left(\|x\| + \|u\| \right). \tag{1.17}$$

La costante $\lambda_{f\hat{f}}$ misura quindi l'accuratezza del modello: più il modello $\hat{f}$ è vicino al sistema reale f, più $\lambda_{f\hat{f}}$ sarà piccola (nel caso ideale di modello perfetto, $\lambda_{f\hat{f}}$ sarebbe nulla).

1.3.2 La rete di controllo

Con il termine "rete" si indica genericamente un insieme di dispositivi, spesso geograficamente distribuiti, interconnessi attraverso un canale di comunicazione che permette la condivisione delle informazioni. Nella presente trattazione si considereranno solamente le reti di tipo "a commutazione di pacchetto" (packet-switched) capaci cioè di trasferire informazione di qualsiasi tipo e lunghezza incapsulando i dati in strutture omogenee dette "pacchetti".
I principali obiettivi delle reti a commutazione di pacchetto sono l'ottimizzazione dell'utilizzo delle risorse di rete, la minimizzazione dei tempi di ritardo e la massimizzazione della robustezza della comunicazione.
Le reti considerate presentano una struttura a blocchi similare a quella in figura 1.2, nella quale, oltre al sistema da controllare, possiamo distinguere cinque macroblocchi:

- il canale condiviso per la comunicazione
- i nodi sensori di stato (SSn): quando hanno accesso alla rete, rilevano gli stati del sistema reale, li codificano in un pacchetto dati e li inviano al nodo controllore attraverso il canale condiviso
- i nodi sensori di uscite (OSn): raccolgono continuamente gli output del sistema reale accumulandoli in un buffer, quando hanno accesso alla rete codificano l'intero buffer in un pacchetto dati e lo inviano al nodo controllore attraverso il canale condiviso
- il nodo controllore: riceve informazioni dai nodi sensori ed elabora i segnali di comando che invia ai nodi attuatori attraverso il canale condiviso; questo nodo è formato da un modello interno del sistema da controllare, un controllore dinamico ed un ulteriore sottoblocco (definito "Local Dynamics") capace di stimare, attraverso i dati ricevuti dai nodi sensori, gli stati interni del controllore e del modello locale

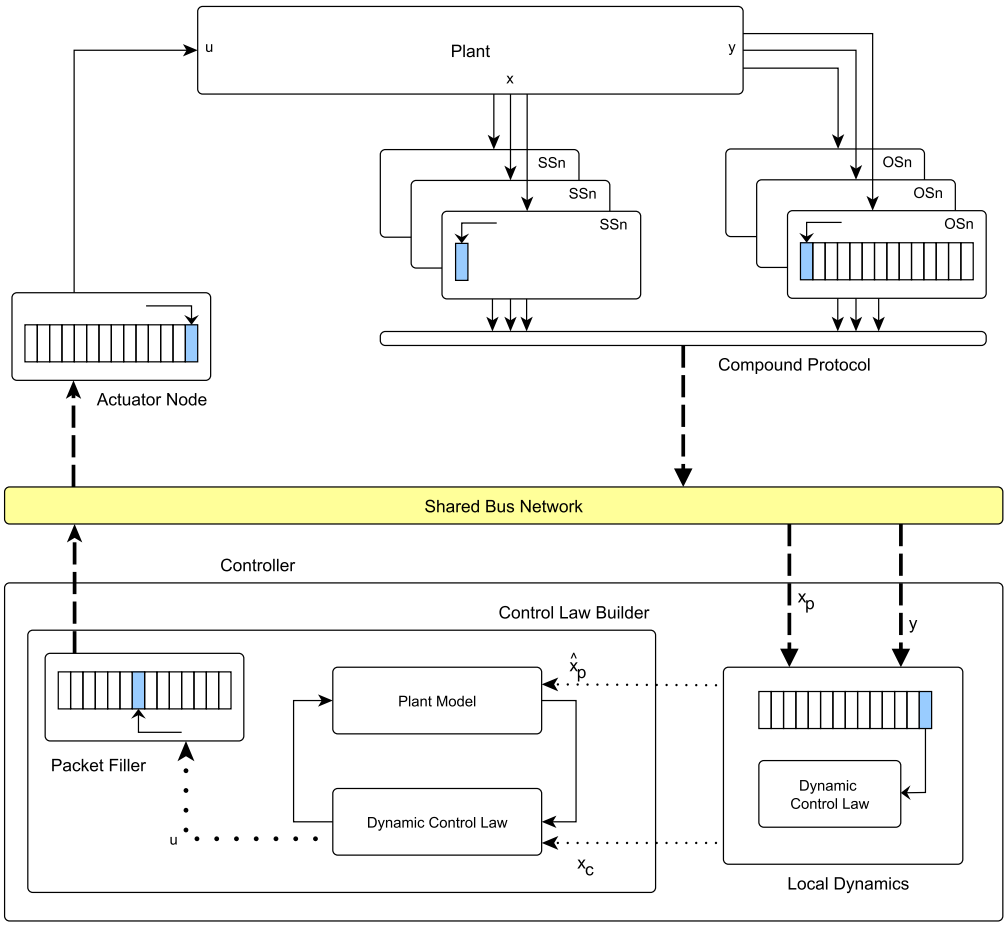

Figura 1.2: Struttura della rete

- i nodi attuatori, che ricevono, decodificano, sincronizzano ed applicano i comandi di controllo all'impianto

Assunzione 4. (Sincronizzazione) *Al fine di produrre un'unico riferimento temporale tutti i nodi sensori ed attuatori sono tra loro sincronizzati.*

Per la definizione dei tempi di invio e ricezione di ogni pacchetto dati è necessario prima di tutto fissare quanto segue:

1. $\{\tau_i^m\}$: sequenza temporale al quale un pacchetto viene inviato dal nodo sensore al nodo controllore

2. $\{\tau_i^c\}$: sequenza temporale al quale un pacchetto viene inviato dal nodo controllore al nodo attuatore

3. $\{T_i^m\}$: ritardo cumulativo nell'invio di un pacchetto dal nodo sensore al nodo controllore

4. $\{T_i^c\}$: ritardo cumulativo nell'invio di un pacchetto dal nodo controllore al nodo attuatore

Sulla base di questi valori è poi possibile definire le due seguenti sequenze temporali:

i $\{\tau_i^m + T_i^m\}$: sequenza di istanti temporali ai quali un pacchetto dati viene ricevuto dal nodo controllore

ii $\{\tau_i^c + T_i^c\}$: sequenza di istanti temporali ai quali un pacchetto dati viene ricevuto dal nodo attuatore

Assunzione 5 (Proprietà della rete)**.** *La rete di comunicazione utilizzata nel controllo soddisfa le seguenti proprietà:*

(i) ***(MATI - massimo tempo di trasferimento permesso)*** *Esistono due costanti* $\tau^m, \tau^c \in \mathbb{R}_{\geq 0}$ *tali che* $\tau_{i+1}^m - \tau_i^m \leq \tau^m$ *e* $\tau_{i+1}^c - \tau_i^c \leq \tau^c$, $\forall i \in \mathbb{N}$;

(ii) ***(mTI - minimo tempo di trasferimento)*** *Esistono le costanti* $\varepsilon^m, \varepsilon^c \in \mathbb{R}_{\geq 0}$ *tali che* $\varepsilon^m \leq \tau_{i+1}^m - \tau_i^m$ e $\varepsilon^c \leq \tau_{i+1}^c - \tau_i^c$ $\forall i \in \mathbb{N}$.

(iii) ***(MAD - massimo ritardo permesso)*** *Esistono due costanti* $T^m, T^c \in \mathbb{R}_{\geq 0}$ *tali che* $T_i^m \leq T^m$ *e* $T_i^c \leq T^c$, $\forall i \in \mathbb{N}$;

Secondo le proprietà (i) e (ii) i tempi intercorrenti tra due invii sono limitati superiormente e inferiormente; la proprietà (iii) garantisce invece la limitazione dei ritardi di trasmissione.

1.3.3 Il concetto di "protocollo"

Dato che lo scambio di dati tra i vari nodi della rete avviene attraverso il medesimo canale di comunicazione, è necessario definire una regola (protocollo) che descriva le modalità con le quali ogni nodo ne ottiene (e rilascia) l'accesso.
Il protocollo è la regola che i nodi devono rispettare per comunicare attraverso la rete condivisa, e può essere descritto attraverso gli aggiornamenti indotti sulle informazioni gestite dal nodo controllore dopo la ricezione di nuovi pacchetti. Fissando con x_p il generico stato del sistema controllato e con $\hat{x}_p$ una sua stima, possiamo definire un errore $e_p \triangleq \hat{x}_p - x_p$. Quando un pacchetto, spedito da un nodo sensore e contenente una misura relativa al tempo t, raggiungerà il nodo controllore, l'errore $e_p(t)$ decrescerà come conseguenza della ricezione di una misura reale di alcune variabili del sistema.
Normalmente, ma non necessariamente, se il numero di nodi sensori è ℓ, il vettore degl errori pu essere partizionato nel seguente modo:

$$e_p = [e_{p_1}^T e_{p_2}^T \dots e_{p_\ell}^T]^T \tag{1.18}$$

Per quanto sopra la variazione di errore indotta dalla ricezione di un pacchetto contenente misure di stato può essere espresso come [2]:

$$e_p\left(\tau_i^{m\,+}\right) = h_p\left(i, e_p(\tau_i^m)\right). \tag{1.19}$$

Quest'ultima forma è definita "protocollo" ed è praticamente una mappa tra l'errore al tempo τ_i^m e quello al tempo $\tau_i^{m\,+}$; una sua proprietà fondamentale è quella di garantire, dopo la ricezione di pacchetti sensore, la decrescita di una qualche funzione definita positiva dell'errore e_p.
Risulta talvolta utile definire anche la seguente dinamica a tempo discreto detta "sistema ausiliario indotto dal protocollo"

$$e_p\left(i+1\right) = h_p\left(i, e_p(i)\right). \tag{1.20}$$

[2]Viene qui utilizzata la notazione $f(t^+) := \lim_{s\to t, s>t} f(s)$

Si riporta a tal proposito una specializzazione di quanto indicato nelle Assunzioni 1 e 2 della sezione 1.3.1:

Definizione 1. *Una funzione $h_p : \mathbb{N} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_p}$ è detta "protocollo UGES" con costanti $\underline{a}, \overline{a}, \rho, c$ se esistono una funzione $W : \mathbb{N} \times \mathbb{R}^{n_p} \to \mathbb{R}_{\geq 0}$ (localmente Lipshitziana rispetto al suo secondo argomento) ed un set di valori $\underline{a}, \overline{a} > 0$, $c > \underline{a}$ e $\rho \in [0, 1)$ tali per cui le seguenti condizioni valgono per il sistema ausiliario tempo discreto indotto $\xi(i+1) = h(i, \xi(i))$ (vedi 1.20, sezione 1.3.3)*

$$\begin{aligned} \underline{a} \left\| \xi \right\| \leq W(i, \xi) \leq \overline{a} \left\| \xi \right\| \\ W(i+1, h(i, \xi)) \leq \rho W_p(i, \xi) \end{aligned} \tag{1.21}$$

per tutti gli $\xi \in \mathbb{R}^n$ e tutti gli $i \in \mathbb{N}$. Inoltre:

$$\left\| \frac{\partial W}{\partial \xi}(i, \xi) \right\| \leq c \tag{1.22}$$

per quasi tutti gli $\xi \in \mathbb{R}^n$ e tutti gli $i \in \mathbb{N}$.

1.3.4 Strategia di gestione dei nodi sensore

La strategia di controllo è formata da due parti: l'acquisizione delle informazioni riguardanti stato ed uscite del sistema e l'invio dei comandi ai nodi attuatori.
In accordo con quanto definito nel capitolo 1.3.2, l'archietettura analizzata prevede l'utilizzo di l_y nodi sensori di tipo OSn ed l nodi sensori di tipo SSn; al fine di limitare l'accumulo di ritardo nella trasmissione dei dati da parte dei nodi sensore, consideriamo che tali nodi possano accedere alla rete secondo la seguente regola: dopo l'accesso alla rete (e la trasmissione) di ogni nodo sensore di stato (SSn), tutti i nodi sensori di output (OSn) dovranno trasmettere i propri buffer secondo un ordine preimpostato (protocollo Round Robin), dopo di che il successivo nodo senosre di stato potrà trasmettere il proprio pacchetto e così via.
Quanto fin qui descritto viene riassunto in figura 1.3, nella quale viene utilizzate la seguente simbologia:

- x_i: stato inviato dal sensore SS_i al tempo $\tau_{s_i}^m$
- y_i: buffer di uscite inviato dal sensore OS_i al tempo $\tau_{o_i}^m$

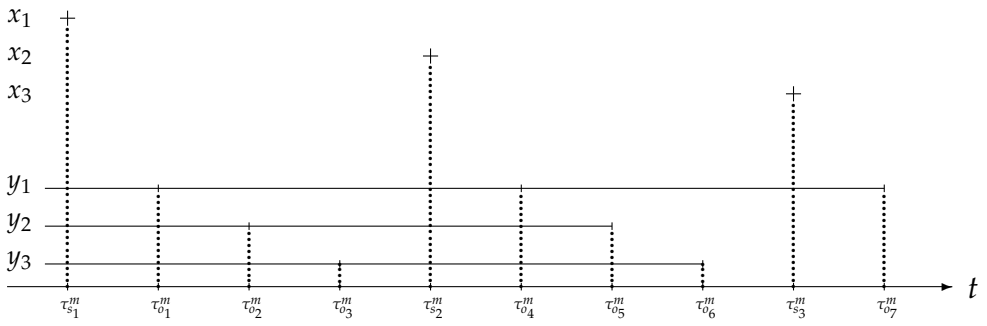

Figura 1.3: Esempio di sincronizzazione tra nodi sensore SS e OS

La regola precedentemente descritta può essere formalizzata estraendo dalla sequenza degli istanti di accesso alla rete $\{\tau_i^m\}$ due sottosequenze $\{\tau_{o_i}^m\}$ e $\{\tau_{s_i}^m\}$, associate a due sequenze $\{s_i\}$ e $\{o_i\}$ contenenti valori in $\mathbb{N}$; attraverso tali costrutti si può dire quanto segue:

- ad ogni istante τ_s^m, con $s \in \{s_i\}$, un sensore SSn ha accesso alla rete di comunicazione
- ad ogni istante τ_o^m, con $o \in \{o_i\}$, un sensore OSn ha la possibilità di trasmettere

La politica di autorizzazioni alla trasmissione è tale da garantire le seguenti prorpietà alle due sequenze:

a) $\{s_i\} \cup \{o_i\} = \mathbb{N} - \{0\}$
$\{s_i\} \cap \{o_i\} = \emptyset$;

b) $s_i \triangleq 1 + (\ell_y + 1)(i - 1)$

c) $o_{(i+1)} \triangleq \begin{cases} 1 + o_i & \text{if } 1 + o_i \notin \{s_i\} \\ 2 + o_i & \text{altrimenti} \end{cases}$.

Attraverso la definizione della seguente sequenza $\{\nu_i\}$ con valori in $[1, \ell_y] \subset \mathbb{N}$ e definita come

$$\nu_{i+1} \triangleq \begin{cases} 1 + \nu_i & \text{if } \nu_i < \ell_y \\ 1 & \text{altimenti} \end{cases} \tag{1.23}$$

sarà poi possibile tener traccia di quale sensore OSn ha avuto accesso alla rete in un dato istante. Considerando $\nu_0 = 1$, per esprimere il fatto che il sensore OSn numero 1 invia per primo, potremo dire che il sensore OSn indicato dall'elemento ν_i-esimo avrà accesso alla rete all'istante $\tau_{o_i}^m$. Oltre a quanto fin qui descritto, è necessario che l'accesso alla rete da parte nei nodi SSn sia garantito attraverso una regola conforme al seguente protocollo (specializzazione di quanto più genericamente indicato in 1.19)

$$e_p\left(\tau_{s_i}^{m\,+}\right) = h_p\left(i, e_p(\tau_{s_i}^m)\right), \forall i \in \mathbb{N} \tag{1.24}$$

con $h_p : \mathbb{N} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_p}$ e tale da rispettare la caratteristica UGES (secondo la definizione riportata nella precedente sezione 1.3.3). Nell'analisi fin qui condotta sul protocollo di gestione dei nodi sensore, è stata fatta la tacita supposizione che tutti gli stati necessari ai fini del controllo fossero disponibili per la misurazione da parte nei nodi sensori SSn. Nel caso in cui suddetta ipotesi non sia soddisfatta, risulterà necessario l'utilizzo di un osservatore di stato capace di sostituire gli stati mancanti con una loro stima; con riferimento a quanto illustrato nella sezione 1.3.2 tale funzione verrà espletata dal sottoblocco "Local Dynamics" del nodo controllore.
Per quanto sopra, consideriamo allora l'introduzione di un osservatore con proprietà GES e descrivibile attraverso la seguente dinamica

$$\dot{z} = \hat{f}_o(z, y) \tag{1.25}$$

$$\hat{x} = g_o(z, y) \tag{1.26}$$

dove $u \in \mathbb{R}^{n_u}$ sono gli input del sistema, $y \in \mathbb{R}^{n_y}$ sono gli output del sistema (che senza perdita di generalità potranno includere anche gli input u) e $z \in \mathbb{R}^{n_z}$, $\hat{x} \in \mathbb{R}^{n_x}$ sono gli stati e gli output dell'osservatore, e definendo con $\psi = x - \hat{x}$ l'errore di stima.

Definizione 2. (Osservatore GES) *L'osservatore definito attraverso la forma compatta di 1.25 e 1.26 è definito di classe GES se esiste una funzione differenziabile* $V : \mathbb{R}^{n_x} \to \mathbb{R}_{\geq 0}$ *ed un insieme di costanti* $\underline{\alpha}, \overline{\alpha}, \alpha, d > 0$ *tali che valga quanto seguenti per ogni* $x, \hat{x} \in \mathbb{R}^{n_x}, u \in \mathbb{R}^{n_u}$

$$\underline{\alpha} \left\| \psi \right\|^2 \leq V(\psi) \leq \overline{\alpha} \left\| \psi \right\|^2$$

$$\frac{\partial V}{\partial \psi}(\psi)\dot{\psi} \leq -\alpha \left\| \psi \right\|^2$$

$$\left\| \frac{\partial V}{\partial \psi}(\psi) \right\| \leq d \left\| \psi \right\|$$

Con tali supposizioni il protocollo inidcato in 1.24 sarà sostituito con

$$h_p\left(i, \bullet\right) = \psi\left(\tau_i^m\right) \tag{1.27}$$

1.3.5 Generazione ed invio degli orizzonti di controllo

Supponiamo che l'istante al quale è stato effettuata la misurazione sia τ_i^m; come anticipato nella sezione 1.3.2, tale pacchetto raggiungerà il nodo controllore al tempo $\tau_i^m + T_i^m$. All'arrivo del nuovo pacchetto il nodo controllore provvederà all'aggiornamento della stima dello stato $\hat{x}$ secondo quanto indicato in (1.28a) e (1.28b)(eventualmente attraverso il sottoblocco "Local Dynamics") e avvierà la simulazione del modello (1.5) (1.6).

$$\dot{\hat{x}}_i(t) = \hat{f}\left(\hat{x}_i(t), g\left(\hat{x}_i(t)\right)\right), \qquad \forall t \in [\tau_i^m, \tau_i^m + T_0^p] \tag{1.28a}$$

$$\hat{x}_i(\tau_i^{m\,+}) = x(\tau_i^m) + h\left(i, \hat{x}_{i-1}(\tau_i^m) - x(\tau_i^m)\right). \tag{1.28b}$$

Con riferimento alla seconda parte della strategia di controllo , e cioè l'invio di pacchetti di comando ai nodi attuatori, consideriamo che per ogni elemento nella sequenza $\{\tau_i^c\}$ sia generato un nuovo segnale di controllo $\hat{u}_i(t)$ il quale, dopo essere codificato in un pacchetto dati, raggiungerà il nodo attuatore al tempo $\tau_i^c + T_i^c$.

Il nuovo segnale di controllo viene generata come indicato da 1.30 e 1.31 sulla base del set di variabili stimate $\hat{x}_{\gamma(i)}$ [3]

$$\hat{u}_i(t) = \hat{g}\left(\hat{x}_{\gamma(i)}(t)\right) \qquad \forall t \in [\tau_i^c, \tau_i^c + T_0^c] \tag{1.30}$$

$$T_0^c \geq T^c + \tau^c, \tag{1.31}$$

La definizione di T_0^c risulta necessaria per assicurare che il nodo attuatore disponga sempre di un segnale di controllo valido e, per il medesimo motivo, la simulazione del modello interno al controllore deve essere elaborato per un periodo temporale T_0^p definito come segue

$$T_0^p \geq \tau^m + T^m + \tau^c + T^c. \tag{1.32}$$

All'istante τ^c, e cioè alla generazione del nuovo pacchetto di controllo, la variabile stimata $\hat{x}_i$ risulterà quindi elaborata fino all'istante $\tau_i^m + T_0^p$.
In figura 1.4 è riportata la sequenza temporale del caso peggiore, in cui il controllore invia un pacchetto di comando ai nodi attuatori nello stesso istante in cui riceve un pacchetto di misurazione dai nodi sensori (utilizzando quindi l'intero invervallo T_0^p precedentemente definito).
Dopo la ricezione del pacchetto di comando, il nodo attuatore provvederà

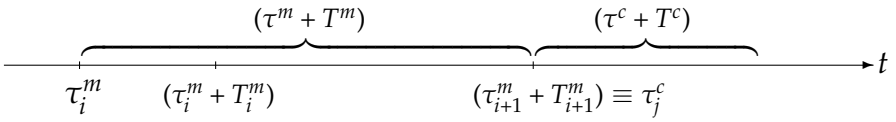

Figura 1.4: Esempio di temporizzazione

all'aggiornamento del suo buffer locale ed il segnale applicato in ingresso

[3] Dove con $\gamma(i)$ si vuole indicare l'indice dell'ultima misurazione ricevuta prima di τ_i^c, e quindi

$$\gamma(i) \mapsto \max\{j \in \mathbb{N} \mid \tau_j^m + T_j^m < \tau_i^c\}. \tag{1.29}$$

al sistema sarà definito come segue

$$\hat{u}(t) = \hat{u}_i(t), \qquad \forall t \in [\tau_i^c + T_i^c, \tau_{i+1}^c + T_{i+1}^c] \tag{1.33}$$

Capitolo 2

Ambiente software

L'ambiente software sviluppato ha lo scopo di supportare il controllista durante le fasi di analisi e sintesi della tecnica PBC. Procedendo con approccio top-down osserviamo che l'ambiente è composto da due moduli, uno per la realizzazione della logica di controllo vera e propria (che definiremo CONTROLLER) ed uno per la simulazione dell'impianto reale (che definiremo VIRTUALPLANT). L'interfaccia con l'utente è basata su file testuali attraverso i quali sarà possibile definire i vari componenti dell'archietettura ed in particolare, con riferimento alla figura 1.2 (Capitolo 1):

- Modello del sistema da controllare (Plant Model)
- Eventuale modello del sistema da simulare (Plant)
- Modello del controllore (Dynamic control low)
- Stati iniziali dei modelli
- Struttura degli stimatori di stato (Local Dynamics)
- Tipo di nodi sensori (OSn) e loro collegamento con gli output dei modelli

- Tipo di nodi attuatori e loro collegamento con gli input del sistema reale
- Configurazioni necessarie per l'accesso al canale di rete condiviso

Di particolare interesse per la verifica delle performance ottenute è la possibilità di settare due diversi modelli di impianto per il modulo di controllo e per quello di simulazione; tale distinzione permetterà infatti di verificare la robustezza del controllo PBC elaborato rispetto ad evenutali variazioni parametriche dell'impianto reale.

2.1 Tipi di file e loro organizzazione

I file utilizzati come interfaccia per la configurazione dell'ambiente di sviluppo possono essere suddivisi attraverso la seguente distinzione funzionale:

- File per la definizione dei modelli
  - File (".ncs") per la creazione dei singoli componenti del modello: ognuno di essi rappresenta un diverso blocco elementare [1] utilizzato nel modelo, contiene i parametri necessari alla propria configurazione ed il suo nome rappresenta l'etichetta assegnata al blocco associato.
  - File "conf_block.ncs": definisce la logica secondo la quale i vari blocchi sono interconnessi tra di loro all'interno del modello (o del sottomodello)
- File per la configurazione del modulo controller
  - File (".sbc") per la definizione dei nodi attuatori [2] e delle relative connessioni con i moduli precedentemente definiti

[1] Scelto tra quelli presenti in libreria, vedi appendice A.1
[2] Scelto tra quelli presenti in libreria, vedi appendice A.2

  - File (".oic") per la definizione dei nodi sensori [3], dei loro collegamenti con i canali di input del blocco Local Dynamics, e per la configurazione degli eventuali osservatori di stato
  - File "conf_sim.sbc" per il settaggio degli stati iniziali dei modelli

- File per la definizione di parametri (".par")
- File per la configurazione dell'interfaccia di rete (".con")

Con riferimento alla distinzione dei due moduli software precedentemente indicata, tali file devono essere organizzati in una struttura gerarchica contenuta in un'unica cartella e definita come segue:

- Sottocartella "CONTROLLER"
  - Tanti file ".ncs" quanti sono i blocchi elementari necessari (ad esempio "INT1.ncs" e "PID_A.ncs" potrebbero indicare un blocco integratore con etichetta INT1 ed un blocco pid con etichetta PID_A)
  - In caso di sistemi paritcolarmente complessi può risultare utile la suddivisione di un unico modello in più sottomodelli; in tal caso sarà sufficiente aggiungere una sottocartella (vista dal sistema come un ulteriore blocco complesso) per ogni sottomodello e racchiudere in essa tutti i file ".ncs" ad esso relativi
  - Un file "conf_block.ncs" nel quale verranno definiti i legami ingresso-uscita dei vari blocchi (elementari e complessi) presenti a questo livello gerarchico della struttura ed appartenenti ai modelli di sistema e controllore; in caso di definizione di modello attraverso scomposizione in sottomodelli sarà necessario editare un file "conf_block.ncs" per ogni sottocartella
  - Tanti file ".sbc" quanti sono i nodi attuatori previsti e contenenti il loro collegamento con gli output dei modelli

[3] Scelto tra quelli presenti in libreria, vedi appendice A.3

  - Un file "conf_sim.sbc" contenente gli stati iniziali dei modelli
  - Un file ".oic" per ogni struttura di ricostruzione degli stati; attraverso questi file è possibile indicare quali stati del modello aggiornare attraverso le stime effettuate
  - Un file ".oic" per ogni blocco (elementare o complesso) del modello che implementa la legge di controllo
  - Un file "conf_par.par" contenente un set prefissato di parametri necessari all'elaborazione della legge di controllo
  - Un file "conf_con.con" contenente le impostazioni IP per l'accesso alla rete condivisa

- Sottocartella "VIRTUALPLANT"

  - Tanti file ".ncs" quanti sono i blocchi elementari necessari (ad esempio "INT1.ncs" e "PID_A.ncs" potrebbero indicare un blocco integratore con etichetta INT1 ed un blocco pid con etichetta PID_A)
  - In caso di sistemi paritcolarmente complessi può risultare utile la suddivisione di un unico modello in più sottomodelli; in tal caso sarà sufficiente aggiungere una sottocartella (vista dal sistema come un ulteriore blocco complesso) per ogni sottomodello e racchiudere in essa tutti i file ".ncs" ad esso relativi
  - Un file "conf_block.ncs" nel quale verranno definiti i legami ingresso-uscita dei vari blocchi (elementari e complessi) presenti a questo livello gerarchico della struttura ed appartenenti ai modelli di sistema e controllore; in caso di definizione di modello attraverso scomposizione in sottomodelli sarà necessario editare un file "conf_block.ncs" per ogni sottocartella
  - Un file ".sbc" con indicazione dei nodi attuatori previsti (si ricordi che stiamo parlando del modulo simulatore di impianto, quindi tali nodi rappresenteranno gli ingressi del modello)

  - Un file “conf_sim.sbc” contenente gli stati iniziali dei modelli
  - Un file “.oic” con indicazione dei nodi sensori da utilizzare per l’invio delle uscite del sistema
  - Un file “conf_par.par” contenente un set prefissato di parametri necessari all’elaborazione della legge di controllo
  - Un file “conf_con.con” contenente le impostazioni IP per l’accesso alla rete condivisa

- Sottocartella “VIRTUALPLANT_UDP” (utilizzata per l’eventuale collegamento con un ambiente di simulazione esterno)
  - Stessi file definiti in VIRTUALPLANT
  - Un file “conf_con_ext.con” contenente le impostazioni IP per la comunicazione con il modello esterno

Per le regole sintattiche da rispettare durante l’editazione di ognuno dei precedenti file, fare riferimento all’appendice B.

2.2 Modulo Controller

Questo modulo provvede alla generazione ed alla gestione della logica di controllo PBC. É costituito principalmente da due componenti: il simulatore del ciclo chiuso tra modello di sistema e legge di controllo (Simulator) ed il blocco per la ricostruzione degli stati (Local Dynamics).

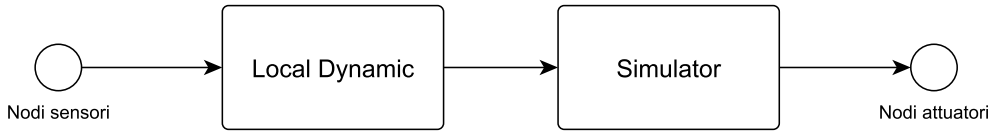

Figura 2.1: Struttura del modulo Controller

2.2.1 Blocco Simulator

Genera un pacchetto di comando a partire dalle informazioni sullo stato disponibili ad un certo istante; gli ingressi e le uscite di ogni passo di elaborazione sono quindi rispettivamente costituiti da una serie di pacchetti contenenti gli stati dei modelli (ricevuti dalla rete o generati all'interno del PC di controllo) ed un pacchetto contenente l'orizzonte di controllo da inviare ai nodi attuatori.
Come indicato in figura 1.2, le evoluzioni di questo blocco sono definite attraverso due tipi di dinamiche, quella relativa al modello di impianto da controllare (Plant Model) e quella che ne descrive la vera e propria legge di controllo (Dynamic Control Low). Dopo aver definito i vari modelli, come indicato nella sezione 2.1, è necessario effettuarne l'abbinamento con i nodi attuatori, in modo da definire la corretta mappatura tra output di controllo e loro registrazione nei relativi pacchetti dati.
Per quanto sopra la configurazione di questo blocco è riassumibile nei seguenti tre passi:

1. Definizione del sistema dinamico da usare per generare il controllo (costituito dall'interazione tra i modelli di sistema e controllore)

2. Abbinamento tra output della legge di controllo e pacchetti da inviare ai nodi attuatori

3. Settaggio degli stati iniziali dei modelli

L'evoluzione di questo modulo è di tipo ciclico e può essere riassunta con il seguente elenco di operazioni:

a. Attesa della ricezione di nuovi aggiornamenti di stato provenienti dal blocco Local Dynamics

b. Creazione di tutti i pacchetti attuatore da popolare (uno per ogni file ".sbc" definito nel precedente punto 2 della procedura di configurazione)

c. Aggiornamento dello stato interno dei modelli sulla base delle informazioni ricevute al punto (a)

d. Esecuzione di un passo di simulazione delle dinamiche dei modelli definiti in (1.11) e (1.12) della precedente sezione 1.3.1

e. Scansione dei vari file ".sbc" per la selezione dai modelli degli output necessari alla compilazione dei pacchetti da inviare ai nodi attuatore

f. Registrazione delle informazioni selezionate al passo (e) nei pacchetti creati al passo (b)

g. Iterazione dei passi (d, e, f) fino al completamento dell'orizzonte di controllo previsto

h. Invio ai nodi attuatori dei pacchetti appena compilati

Si noti per prima cosa che, al ricevimento di nuove informazioni provenienti dall'impianto, queste vengono utilizzate dal blocco Simulator per l'aggiornare dello stato interno dei modelli definiti[4]. A causa dei ritardi di trasmissione da parte dei nodi sensore però, ciò potrebbe avvenire anche in presenza di informazioni sullo stato istantaneamente non complete, richiedendo quindi una temporanea miscelazione tra i nuovi stati ricevuti e quelli già elaborati dalle precedenti iterazioni simulative. A tale scopo sono state definite due funzioni, getStartState e setState, rispettivamente capaci di raccogliere gli aggiornamenti di stato disponibili e di effettuare un parziale aggiornamenti degli stati fino a quel momento generati dal simulatore lasciando inalterati quelli relativi ad informazioni non ancora disponibili.
Durante la scelta dei nodi sensori ed attuatori, l'utente potrà attingere da una gamma di elementi organizzati in un'apposita libreria precompilata, oppure provvedere personalmente all'aggiunta di nuovi driver o alla

[4]Passo (c)

modifica di quelli già esistenti[5]. Dato che la struttura dati di ogni pacchetto dipende strettamente dalla logica di funzionamento dei nodi ad essi associati, si è reso necessario gestire questa situazione di polimorfismo con la definizione, per ognuno dei driver registrati in libreria, di apposite funzioni di interfaccia (PacketCreator e Set per i nodi attuatore, Get per i nodi sensore) capaci di gestire in modo autonomo la creazione, la compilazione e la lettura di tali strutture dati. Attraverso questa soluzione le routine dedicata all'elaborazione dei passi (e) ed (f) sono state tradotte in una sequenza di chiamate a dette funzioni.
Come indicato nella sezione 1.3.5 del precedente capitolo, la tecnica PBC prevede che ogni pacchetto inviato al nodo attuatore contenga una sequenza di comandi riferita (e ordinata) secondo un preciso orizzonte temporale, basato su un clock sincronizzato con quello di tutti i nodi della rete; la lunghezza di tale orizzonte deve essere indicata dall'utente tra i parametri del file "conf_par.par" e necessita di un'opportuna taratura basata su vari fattori, quali la banda disponibile nella rete di comunicazione, i ritardi di trasmissione e la capienza massima di ogni buffer di controllo.
Durante ogni evoluzione della precedente sequenza, le varie funzioni PacketCreator (passo (f)) rilevano l'istante di creazione del pacchetto gestito [6] identificandolo come istante iniziale per l'ordinamento dell'orizzonte. Per effettuare il corretto posizionamento dei nuovi dati, via via forniti del passo (e), all'interno della sequenza di controllo è quindi sufficiente analizzarne gli istanti di elaborazione ed ordinarli in senso crescente a partire dal suddetto istante iniziale.
Attraverso le funzioni PacketCreator[7], è inoltre possibile effettuare un precondizionamento dell'orizzonte di controllo per renderlo conforme a quanto atteso dal circuito hardware degli attuatori. Si supponga ad esempio che l'output del modello utilizzato nel blocco Simulator indichi la posizone (in

[5] Vedi libreria A.1

[6] E quindi dell'ultima esecuzione del passo (b)

[7] Che ricordiamo essere customizzabili per ognuno dei driver presenti in libreria

mm) di un'elettrovalvola ma che il relativo servomeccanismo necessiti di un controllo in tensione; tra le soluzioni possibili di tale problema si può scegliere di implementare un'opportuna mappatura dei segnali (da mm a V) all'interno della funzione PacketCreator assegnata al nodo attuatore utilizzato.

2.2.2 Blocco Local Dynamics

Come rappresentato in figura 1.2, questo blocco rappresenta il primo stadio di ingresso del modulo di controllo e provvede all'elaborazione dei dati contenuti in tutti i pacchetti provenienti dai nodi sensore; per quanto sopra questo blocco è anche implicitamente responsabile della gestione in feedback degli output dell'impianto reale.
Per semplicità gestionale tutti i nodi sensore utilizzati in questo software sono di tipo OSn [8] pertanto, anche gli stati direttamente misurabili sull'impianto, verranno trasmessi in forma di orizzonti temporali (come accade per i normali segnali di output); l'analisi dei dati contenuti nei vari pacchetti verrà poi diversificata in base alla loro provenienza: le strutture relative a segnali di output vengono interamente elaborate, mentre da quelle relative a segnali di stato viene estratto solamente l'ultimo elemento trasmesso (ricollegando così l'intera gestione a quella schematizzata in figura 1.3 del precedente capitolo).
La funzione di questo blocco, come previsto nella sezione 1.3.4, è quella di elaborare gli aggiornamenti di stato da applicare ai modelli del blocco Simulator e, dato che abbiamo precedentemente distinto il modello dell'impianto da quello del controllore, è utile descrivere come tale diversificazione sia stata utilizzata anche in questo blocco.
Tutte le stime vengono generate attraverso l'evoluzione di apposite dinamiche, alle quali vengono fornite le informazioni recepite dai nodi sensore come ingresso forzante. Per quanto riguarda gli stati dei modelli di

[8] Vedi sezione 1.3.4 del precedente capitolo

controllo, le dinamiche utilizzate sono esattamente identiche a quelle del blocco Simulator ed i loro stati rappresentano quindi direttamente le stime volute; per il modello di sistema verranno invece utilizzati degli appositi osservatori esponenzialmente convergenti , e le stime saranno rappresentate dai loro output.
Ogni sistema dinamico può quindi fornire due tipi di infomrazioni:

- I propri stati interni
- Le proprie uscite

e la sua funzione può essere abbinata ad uno dei i seguenti casi:

1. Parte dell'informazione ricevuta concorre direttamente (o tramite manipolazioni algebriche) all'elaborazione di parte dello stato interno
2. É disponibile un sistema dinamico le cui uscite concorrono a stimare parte dello stato interno del modello (per esempio un osservatore dello stato)
3. É disponibile un sistema dinamico il cui stato interno concorre a stimare parte dello stato interno del modello (sarà questo il caso del controllore dinamico)

Dopo la ricezione i pacchetti provenienti dai nodi sensore vengono organizzati in una struttura similare a quella della seguente tabella 2.1 e, prima

	τ_i^m	τ_{i+1}^m	τ_{i+2}^m	τ_{i+3}^m	τ_{i+4}^m	τ_{i+5}^m
Nodo sensore 1	x	x	x	x	x	-
Nodo sensore 2	x	x	-	-	-	-
Nodo sensore 3	x	x	x	x	-	-
Nodo sensore 4	x	x	x	-	-	-

Tabella 2.1: Organizzazione dei pacchetti sensore ricevuti

della generazione del successivo set di stime, la logica del blocco Local Dynamics dovrà verificare sia il tipo di pacchetto necessario per ogni dinamica utilizzata, sia fino a quale istante sono disponibili tutte le informazioni richieste.
Con riferimento alla precedente tabella 2.1 si supponga, ad esempio, di avere un osservatore convergente per la stima degli stati del modello, basato sulle informazioni provenienti dai nodi sensore 2 e 4, e due controllori pid collegati rispettivamente ai sengali provenienti dal nodo sensore 1 e dal nodo sensore 3. La stima degli stati del primo pid potrà essere effettuata facendone evolvere la dinamica fino all'istante τ^c_{i+4}, quella del secondo pid fino all'istante τ^c_{i+3} e quella dell'osservatore fino all'istante τ^c_{i+1}.

2.3 Modulo Virtualplant

Questo modulo provvede alla simulazione di un modello dell'impianto, con i relativi nodi sensori ed attuatori, e può essere schematizzato con la seguente figura 2.2.
Le dinamiche dell'impianto possono essere descritte sia attraverso la li-

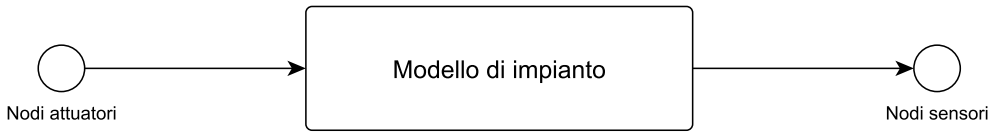

Figura 2.2: Struttura del modulo Virtualplant

breria networkatom, sia attraverso qualsiasi altro software di simulazione (ad esempio Matlab/Simulink), purché esso sia opportunamente interfacciabile con questo modulo.
Il modulo risulta particolarmente utile nella fase di analisi delle prestazioni ottenute dalla tecncia PBC precedetnemente elaborata, e le sue caratteristiche possono essere riassunte come segue:

- Verifica del corretto funzionamento della tecnica di controllo prima dell'applicazione sull'impianto reale (che può anche non essere nell'immediata disponibilità del controllista)
- Verifica della robustezza della tecnica PBC elaborata attraverso la simulazione di modelli di impaianto diversi (e non necessariamente conformi a quelli utilizzati nel modulo Controller)
- Analisi dei risultati ottenuti attraverso loro visualizzazione grafica o testuale

2.3.1 Funzionamento e configurazione

Come precedentemente illustrato, questo modulo esegue la simulazione sia del modello di impianto che dei nodi attuatori e sensori ad esso realmente collegati; gli input attesi sono quindi i pacchetti elaborati dal nodo controllore (contenenti gli orizzonti di controllo), mentre gli output restituiti sono i pacchetti generati dai nodi sensore e contenenti gli output misurabili dell'impianto.
In fase di avvio il modulo provvede all'elaborazione della sottocartella VIRTUALPLANT, definita nella sezione 2.1, ed esegue le seguenti operazioni:

1. Scansione dei file ".oic" e creazione della lista dei nodi sensore utilizzati
2. Scansione dei file ".sbc" e creazione della lista dei nodi attuatori utilizzati
3. Definizione del modello di impianto attraverso scansione dei file ".ncs" e loro composizione attraverso il file "conf_block.ncs"
4. Settaggio degli stati iniziali del modello secondo quanto descritto nel file "conf_sim.sbc"

5. Rilevamento del tempo di simulazione attraverso il parametro *simulation_time* definito nel file "conf_par.par"

Al termine di tale inizializzazione, viene avviato un ciclo di simulazioni che terminerà automaticamente al raggiungimento del tempo rilevato nel precedente punto (5). La sequenza eseguita ad ogni passo di simulazione è riassumibile nei seguenti punti:

a. Creazione di tutti i pacchetti sensore da inviare (definiti al precedente punto 1 della procedura di configurazione)

b. Attesa della ricezione di nuove informazioni dal nodo controllore

c. Aggiornamento degli input dei modelli sulla base delle informazioni ricevute

d. Rilevazione degli output generati dal modello simulato

e. Esecuzione di un passo di simulazione delle dinamiche dei modelli definiti in (1.11) e (1.12) della precedente sezione 1.3.1

f. Registrazione delle informazioni selezionate al passo (d) nei pacchetti creati al passo (a)

g. Iterazione dei passi (b, c, d, e, f) fino al completamento dell'orizzonte di previsto per i pacchetti sensori

h. Invio al nodo controllore, attraverso i nodi sensori, dei pacchetti appena compilati

Durante la scelta dei nodi sensori ed attuatori, l'utente potrà ancora attingere da una gamma di elementi organizzati in un'apposita libreria precompilata, oppure provvedere personalmente all'aggiunta di nuovi driver o alla modifica di quelli già esistenti[9]. Dato che la struttura dati di ogni pacchetto dipende strettamente dalla logica di funzionamento del nodo ad

[9] Vedi librerie A.2 e A.3

esso associato, è stato necessario gestire questa situazione di polimorfismo con la definizione, per ognuno dei driver registrati in libreria, di apposite funzioni di interfaccia (inputGenerator e notifyPacket per i nodi attuatore, outputNotifier per i nodi sensore) capaci di gestire in modo autonomo la lettura dei nuovi pacchetti provenienti dal nodo controllore e di compilare opportunamente quelli ad esso destinati. Attraverso questa soluzione le routine dedicata all'elaborazione dei passi (a), (c), (f) ed (h) sono state tradotte in una sequenza di chiamate a dette funzioni.
Come indicato nella sezione 1.3.4 del precedente capitolo, la tecnica PBC prevede che ogni pacchetto ricevuto dai nodi sensori di output[10] contenga una sequenza di informazioni riferita (e ordinata) secondo un preciso orizzonte temporale, basato su un clock sincronizzato con quello di tutti i nodi della rete; la lunghezza di tale orizzonte deve essere indicata dall'utente tra i parametri del file "conf_par.par" e necessita di un'opportuna taratura basata su vari fattori, quali la banda passante disponibile nella rete di comunicazione, i ritardi di trasmissione e la capienza massima di ogni buffer dei sensori.
Durante ogni evoluzione della precedente sequenza, le varie funzioni outputNotifier (passo (f)) rilevano l'istante di creazione del pacchetto gestito [11] identificandolo come istante iniziale per l'ordinamento dell'orizzonte. Per effettuare il corretto posizionamento dei nuovi dati, via via forniti del passo (e), all'interno della sequenza delle misurazioni è quindi sufficiente analizzarne gli istanti di elaborazione ed ordinarli in senso crescente a partire dal suddetto istante iniziale.

2.3.2 Interfaccia con altri software di simulazione

Questa modalità di funzionamento può essere riassunta con la seguente figura 2.3, ed è stata implementata come ulteriore metodo di verifica delle

[10] Come descritto in 2.2.2, tutti i nodi sensore utilizzati in questo ambiente di sviluppo sono di tipo OSn

[11] E quindi dell'ultima esecuzione del passo (a)

performance raggiungibili con la tecnica PBC elaborata.
Dopo un'adeguata configurazione degli indirizzamenti di rete, il pc dedicato alla gestione del modulo Virtualplant viene avviato come "bridge" ed utilizzato per mettere in comunicazione il nodo controllore con un modello esterno attraverso l'elaborazione ciclica delle seguenti operazioni:

1. Ricezione orizzonte di controllo da nodo controller (attraverso la scansione del nodo attuatore)
2. Invio al modello esterno (attraverso datagram UDP) ed al modello interno di ogni singolo step di controllo
3. Ricezione dal modello esterno di un datagram UDP contenente l'output generato dall'ultimo step di simulazione (eseguito utilizzando come input il segnale inviato in 2) [12]
4. Estrazione dell'output generato dal modello interno dopo l'ultimo step di simulazione
5. Confronto degli output rilevati in 3 e 4
6. Compilazione dei pacchetti sensore attraverso gli output (esterni) rilevati in 3

Come indicato nella sezione 2.1, i file dedicati alla configurazione di questa modalità operativa dovranno essere contenuti in un'unica sottocartella nominata VIRTUALPLANT_UDP e, oltre a quelli precedentemente editati per la normale esecuzione di questo modulo, ne dovrà essere aggiunto uno dedicato all'indirizzamento di rete. Quest'ultimo, nominato "conf_con_ext.con", dovrà contenere nell'ordine

- Inirizzo IP dell'elaboratore esterno (sul quale viene simulato il modello esterno)

[12] Al fine di mantenere la corretta sequenza di comando è necessario che lo scambio di datagram tra modello esterno e modulo Virtualplant avvenga attraverso connessioni UDP di tipo "bloccante"

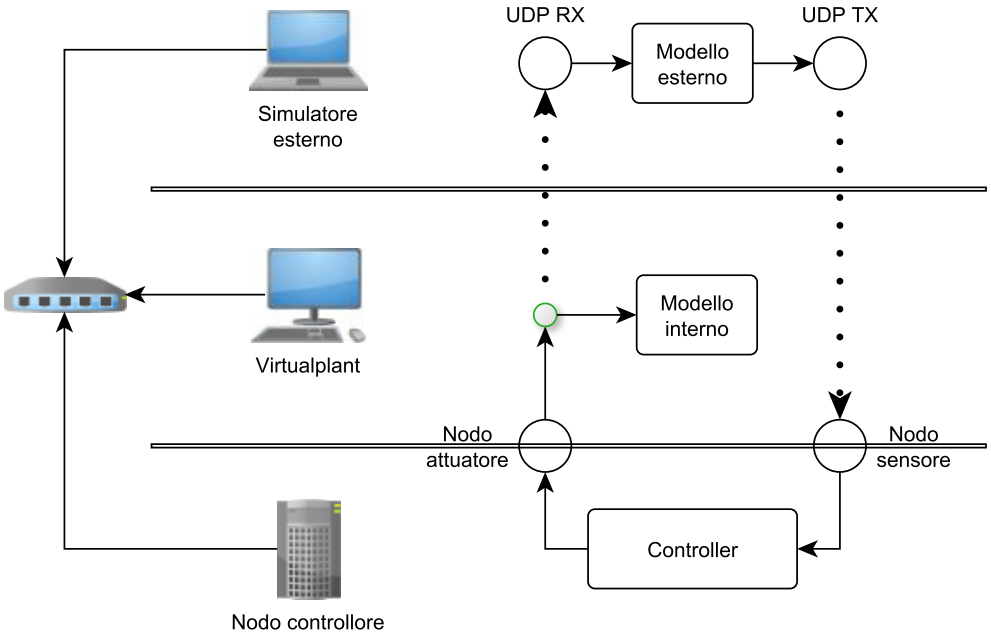

Figura 2.3: Metodo di interfaccia con software di simulazione esterno

- Porta dalla quale l'elaboratore esterno invia informazioni
- Porta dalla quale l'elaboratore esterno riceve informazioni
- Inirizzo IP dell'elaboratore locale (sul quale viene avviato il modulo Virtualplant)
- Porta dalla quale l'elaboratore locale invia informazioni
- Porta dalla quale l'elaboratore locale riceve informazioni

Oltre all'elaborazione del suddetto file, sarà necessario ampliare il modello esterno di impianto affinché diventi capace di scambiare informazioni attraverso comunicazioni UDP e, a tale scopo, si dovrà verificare che l'ambiente desiderato preveda l'utilizzo (in modalità bloccante) di blocchi "UDP sender" ed "UDP receiver". Per eseguire tale modifica sarà poi sufficiente seguire la seguente procedura:

i Aprire il modello esterno attraverso l'ambiente di simulazione desiderato

ii Sostituire gli input del modello con un unico blocco di tipo "UDP receiver" in modalità bloccante [13]

iii Sostituire gli output del modello con un unico blocco di tipo "UDP sender" [14]

iv Impostare in modo opportuno le priorità di esecuzione dei blocchi affinchè, durante ogni passo di simulazione, il blocco UDP receiver venga fatto eseguire prima del blocco UDP sender

v Settare gli indirizzi richiesti dai due blocchi inseriti in modo conforme a quanto indicato nel file "conf_con_ext.con"

Al termiine della sopraindicata configurazione, e dopo aver verificato che il pc utilizzato per la gestione del modello esterno sia raggiungibile dal modulo Virtualplant, sarà possibile lanciare la simulazione avviando nell'ordine il nodo controllore, il modulo esterno ed il modulo Virtualplant[15]. In fase di avvio il modulo provvede all'elaborazione della sottocartella VIRTUALPLANT_UDP, definita nella sezione 2.1, ed esegue le seguenti operazioni:

1. Scansione dei file ".oic" e creazione della lista dei nodi sensore utilizzati

2. Scansione dei file ".sbc" e creazione della lista dei nodi attuatori utilizzati

3. Definizione del modello di impianto attraverso scansione dei file ".ncs" e loro composizione attraverso il file "conf_block.ncs"

[13] In caso di più input attesi sarà necessario posizionare a valle di tale blocco un demodulatore capace di estrarre dall'unico pacchetto inviato i vari segnali necessari

[14] In caso di più output generati sarà necessario posizionare a monte di tale blocco un modulatore capace di convogliare i vari segnali all'interno di un unico pacchetto dati

[15] Vedi esempi per i comandi da utilizzare

4. Settaggio degli stati iniziali del modello secondo quanto descritto nel file "conf_sim.sbc"

5. Creazione del socket UDP dedicato alla comunicazione con il modello esterno

6. Rilevamento del tempo di simulazione attraverso il parametro *simulation_time* definito nel file "conf_par.par"

Al termine di tale inizializzazione, viene avviato un ciclo di simulazioni che verrà automaticamente arrestato al raggiungimento del tempo rilevato nel precedente punto (6). La sequenza eseguita ad ogni passo di simulazione è riassumibile, con riferimento alla figura 2.4, nei seguenti punti:

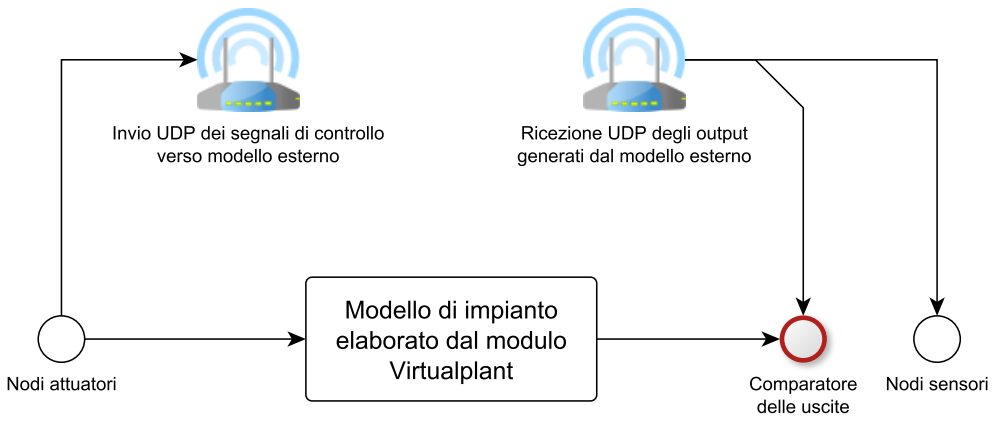

Figura 2.4: Principio di connessione tra modulo interno ed esterno

a. Creazione di tutti i pacchetti sensore da inviare (definiti al precedente punto 1 della procedura di configurazione)

b. Attesa della ricezione di nuove informazioni dal nodo controllore

c. Aggiornamento degli input del modello interno sulla base delle informazioni ricevute

d. Codifica delle informazioni ricevuta in un unico pacchetto e suo invio al bloco "UDP receiver" del modello esterno (attraverso la connessione definita al punto (5) della procedura di avvio)

e. Attesa della ricezione di nuove informazioni dal modello esterno (sintetizzate in un unico pacchetto dati ed inviate dal blocco "UDP sender")

f. Verifica delle informazioni ricevute (output del modello esterno)

g. Rilevazione degli output generati dal modello interno

h. Confronto comparativo tra gli output rilevati nei due precedenti punti e conseguente sengalazione di errore in caso di scarto superiore ad una soglia preimpostata [16]

i. Esecuzione di un passo di simulazione delle dinamiche dei modelli definiti in (1.11) e (1.12) della precedente sezione 1.3.1

j. Registrazione delle informazioni verificate al passo (f) nei pacchetti creati al passo (a)

k. Iterazione dei passi (b, c, d, e, f, g, h, i, j) fino al completamento dell'orizzonte di previsto per i pacchetti sensori

l. Invio al nodo controllore, attraverso i nodi sensori, dei pacchetti appena compilati

Per la scelta e la configurazione dei nodi sensori ed attuatori si rimanda interamente a quanto descritto nella precedente sezione 2.3.1.

[16] Attraverso questo controllo è possibile verificare quanto il modello adottato sia conforme a quello esterno

Capitolo 3

Esempi applicativi

Dopo l'introduzione fatta in merito alla configurazione ed al funzionamento dei vari moduli software implementati, in questo capitolo verrà analizzata la loro applicazione a casi reali.
Nella prima parte verranno elencate, attraverso un breve riassunto di quanto visto precedentemente, tutte le operazioni necessarie per l'applicazione del software ad un generico sistema di controllo NCS/PBC; successivamente, attraverso l'analisi di due casi reali, verranno illustrati i risultati ottenibili dall'interazione dei vari moduli.

3.1 Procedura operativa

Per prima cosa è necessario definire, a partire dallo schema di controllo, i contenuti delle tre sottocartelle indicate nella sezione 2.1 e cioè:

- CONTROLLER
- VIRTUALPLANT
- VIRTUALPLANT_UDP

A tal fine, si prenda come esempio la semplice struttura rappresentata in figura 3.1; costituito da un sistema LTI (qui rappresentato in forma di sta-

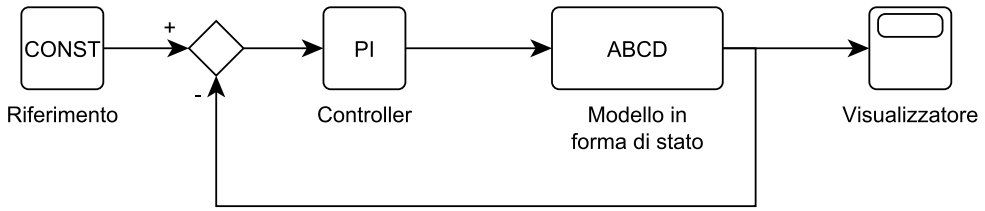

Figura 3.1: Struttura del modulo Controller

to) e sottoposto ad un ingresso costante, l'impianto è controllato attraverso un blocco PI ed analizzato con il supporto di un visualizzatore collegato alla sua unica uscita.
Per un corretto approccio al problema si consideri adesso la seguente sequenza di passi:

1. Effettuare una prima distinzione, come rappresentato nella seguente figura 3.2, tra ambiente di controllo [1] ed ambiente di simulazione [2].

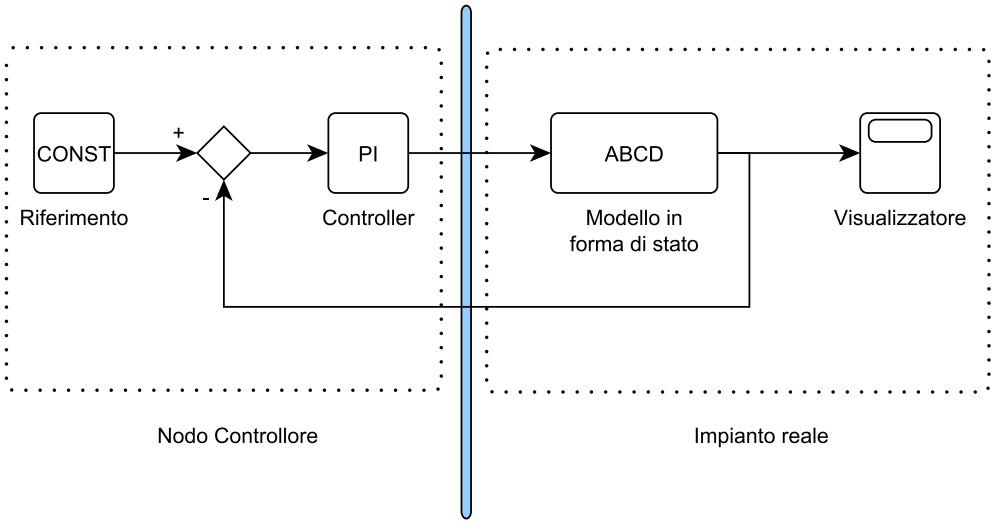

Figura 3.2: Limitazione degli ambienti di controllo e simulazione

[1] Collegato ai contenuti della cartella CONTROLLER
[2] Collegato ai contenuti della cartella VIRTUALPLANT/VIRTUALPLANT_UDP

2. Definire le caratteristiche dei nodi attuatori e sensori necessari per lo scambio di dati tra i due ambienti precedentemente definiti; supponiamo quindi di utilizzare

   - Un unico nodo attuatore capace di gestire 1 canale (orizzonte di controllo)
   - Un unico nodo sensore capace di gestire n canali (output del modello)

   come rappresentato in figura 3.3

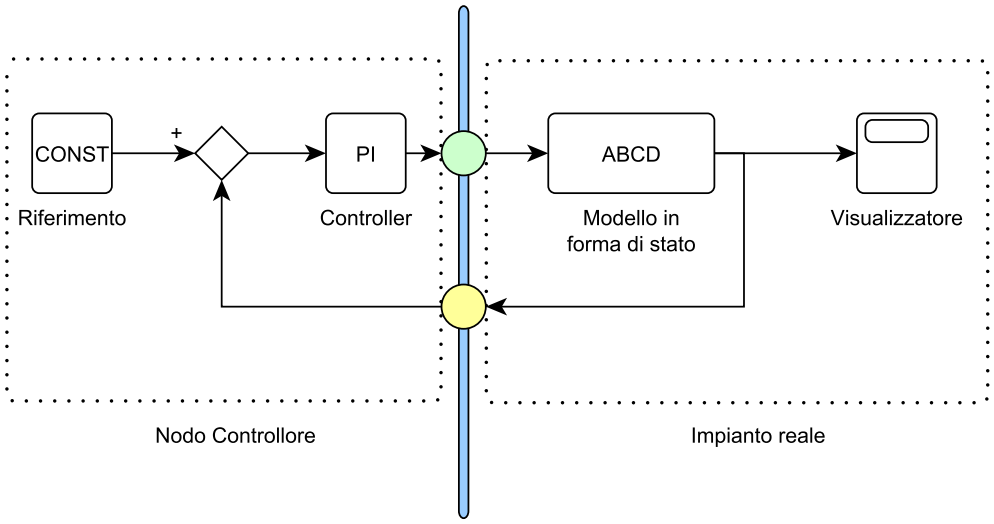

Figura 3.3: Introduzione di nodi sensori ed attuatori

3. Controllare quali modelli (ed eventuali osservatori di stato) è necessario definire per permettere la continuità del loop secondo l'interazione Simulator/Local Dynamics prevista nella precdente sezione 2.2. In questo caso è immediato verificare la necessità di:

   - Un modello dell'impianto (da introdurre nel sottoblocco Simulator del modulo Controller) per il ripristino della catena di feedback

- Un osservatore di stato (da introdurre nel sottoblocco Local Dynamics del modulo Controller) per l'aggiornamento degli stati del precedente modello attraverso l'elaborazione dei pacchetti dei nodi sensori [3]

Con l'introduzione di quanto sopra il modello di figura 3.3 si traduce in quello di figura 3.4.

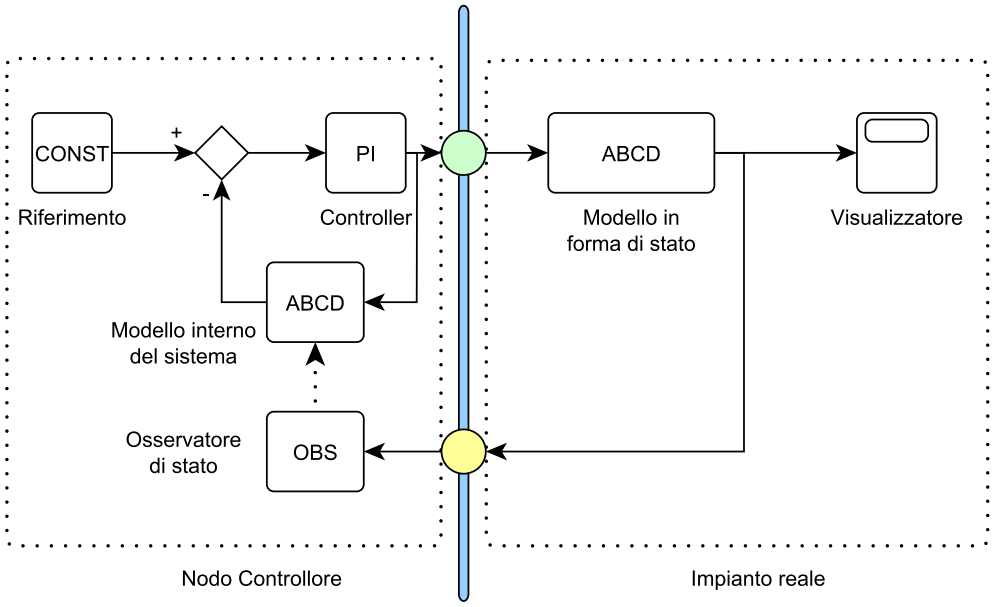

Figura 3.4: Ripristino dei collegamenti di feedback

4. Si ricordi che i modelli elaborati dal sottoblocco Simulator possono recepire le informazioni provenienti dai nodi sensori solo attraverso l'intervento del blocco Local Dynamics; con riferimento a quanto definito nella sezione 2.2.2, si ricordi inoltre che il blocco Local Dynamics richiede una strutturazione dell'ambiente Controller attraverso due sole tipologie di modelli:

[3] Vedi sezione 2.2.2

- modelli d'impianto (aggiornabili solo attraverso un proprio osservatore di stato)
- modelli di controllo (direttamente aggiornabili)

Controllare che la struttura fin qui ottenuta rispetti questa architettura, ed eventualmente raggruppare in modo opportuno elementi riconducibili ad un medesimo modello. Nel caso in esame si può rilevare quanto segue:

- il modello di definito al passo (3) è l'unico modello di impianto presente
- per una più agevole gestione è utile inglobare i blocchi "riferimento", "sommatore" e "pid" all'interno di un unico sottomodello di controllo (ottenendo così lo schema definito nella seguente figura 3.5

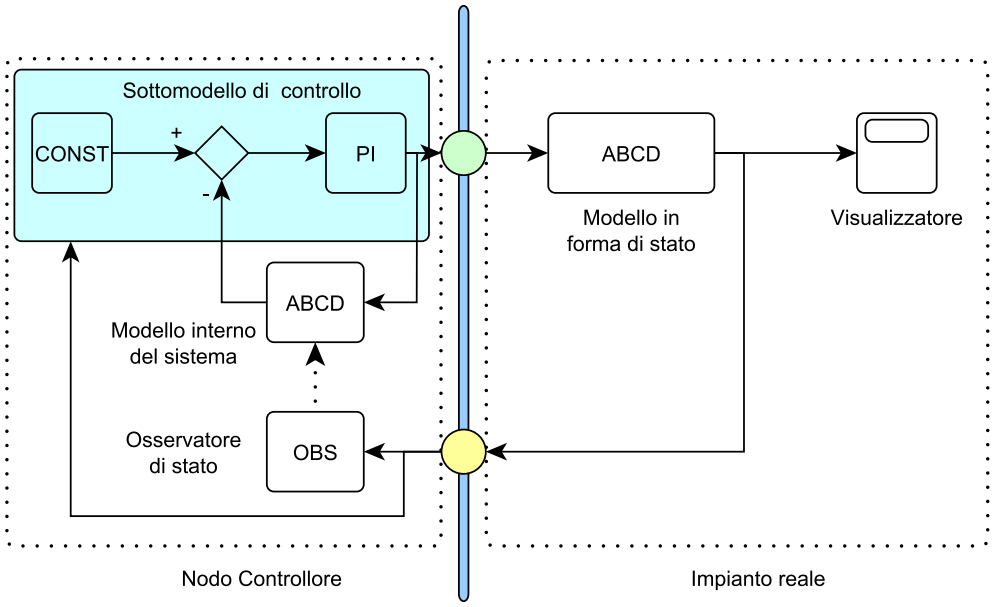

Figura 3.5: Schema finale dell'ambiente di controllo

5. Verificare l'eventuale necessità di osservatori di stato da associare a modelli finora non discussi. Nel caso in esame, oltre al modello di impianto definito al passo (3) (insieme al relativo osservatore), non sono presenti ulteriori modelli di questo tipo e non sono quindi necessari ulteriori osservatori.

6. Il sistema di controllo è adesso conforme all'archietuttura prevista dall'ambiente software ed è necessario editare i file introdotti nella sezione 2.1 e riassumibili con la seguente gerarchia:

Cartella CONTROLLER		
Sottocartella	**File**	**Tipo**
MOD_CONTR	PID.ncs	discrete_pid
MOD_CONTR	RIFERIMENTO.ncs	constant
MOD_CONTR	SOMMATORE.ncs	sum
MOD_CONTR	conf_block.ncs	-
OBS_MOD_IMP	OBS_DYNAMIC.ncs	abcd
OBS_MOD_IMP	MUX.ncs	mux
OBS_MOD_IMP	conf_block.ncs	-
-	SCOPE.ncs	scope
-	MOD_IMP.ncs	abcd
-	conf_block.ncs	-
-	ACTUATOR.sbc	-
-	MOD_CONTR.oic	-
-	OBS_MOD_IMP.oic	-
-	conf_sim.sbc	-
-	conf_con.con	-
-	conf_par.par	-

Cartella VIRTUALPLANT		
Sottocartella	**File**	**Tipo**
-	SCOPE.ncs	scope
-	MOD_IMP.ncs	abcd
-	conf_block.ncs	-
-	FLEXACTUATOR.sbc	-
-	FLEXSENSOR.oic	-
-	conf_sim.sbc	-
-	conf_con.con	-
-	conf_par.par	-

Cartella VIRTUALPLANT_UDP		
Sottocartella	**File**	**Tipo**
-	SCOPE.ncs	scope
-	MOD_IMP.ncs	abcd
-	conf_block.ncs	-
-	FLEXACTUATOR.sbc	-
-	FLEXSENSOR.oic	-
-	conf_sim.sbc	-
-	conf_con.con	-
-	conf_con_ext.con	-
-	conf_par.par	-

Per una completa analisi dei contenuti di tali file si faccia riferimento alle appendici A e B.

7 A queto punto è necessario scegliere la modalità di lavoro con la quale si vuole avviare l'ambiente software e procedere con le seguenti operazioni:

- Modalità di solo controllo:
  - Dal pc di controllo, entrare nella cartella di installazione del software e digitare la seguente istruzione

./controller {path relativo della cartella CONTROLLER}

- Modalità di controllo e simulazione (diretta):
  - Dal pc di simulazione, entrare nella cartella di installazione del software e digitare la seguente istruzione
    ./virtual_plant {path relativo della cartella VIRTUALPLANT}
  - Dal pc di controllo, entrare nella cartella di installazione del software e digitare la seguente istruzione
    ./controller {path relativo della cartella CONTROLLER}
- Modalità di controllo e simulazione (attraverso software di simulazione esterno):
  - Seguire le indicazioni riportate nella sezione 2.3.2
  - Avviare la simulazione del modello esterno
  - Dal pc di simulazione, entrare nella cartella di installazione del software e digitare la seguente istruzione
    ./virtual_plant {path relativo della cartella VIRTUALPLANT_UDP}
  - Dal pc di controllo, entrare nella cartella di installazione del software e digitare la seguente istruzione
    ./controller {path relativo della cartella CONTROLLER}

3.2 Pendolo di Furuta

In questa sezione verrà presentata, come primo esempio applicativo del software realizzato, il controllo NCS/PBC di un "Pendolo di Furuta". Questo sistema rappresenta un particolare tipo di pendolo inverso ed essendo basato su una dinamica molto veloce, fortemente non lineare e con un punto di equilibrio instabile, si presta molto bene alla verifica delle performance raggiungibili attraverso questa architettura di controllo. Come presentato in [3], questo esempio è stato precedentemente trattato con la definizione di un controllore ad-hoc ed il candidato ha partecipato a tale

sviluppo attraverso l'elaborazione del software utilizzato nei nodi sensori ed attuatori.

3.2.1 Modellazione non lineare

Il Pendolo di Furuta, schematicamente rappresentato in figura 3.6, è formato dai seguenti componenti:

- Colonna centrale ancorata alla base
- Braccio orizzontale accoppiato alla colonna attraveso un albero motore
- Asta del pendolo, incernierata al braccio orizzontale attraverso un giunto rotoidale e libera di ruotare

La dinamica del sistema è quindi rappresentabile attraverso la generica formulazione Lagrangiana

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau \tag{3.1}$$

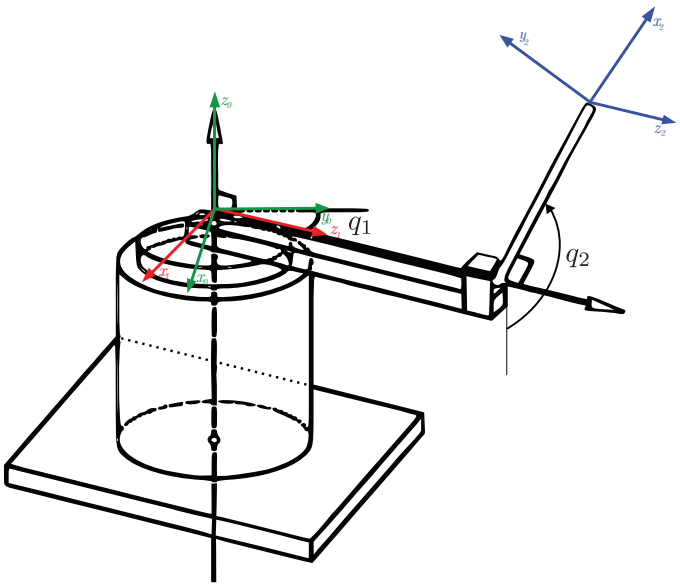

Figura 3.6: Schematica rappresentazione del Pendolo di Furuta

con $\boldsymbol{q}$ vettore delle coordinate Lagrangiane, $B(\boldsymbol{q})$ matrice di inerzia, $C(\boldsymbol{q}, \dot{\boldsymbol{q}})$ matrice delle forze di Coriolis, $G(\boldsymbol{q})$ matrice delle forze gravitazionali e $\boldsymbol{\tau}$ insieme delle coppie applicate a ciascun giunto.
Il vettore delle variabili di stato è definito da $\boldsymbol{q} = [q_1, q_2]^T$, con q_1 posizione angolare del braccio e q_2 posizione angolare dell'asta del pendolo ed il sistema considerato è di tipo "sottoattuato", dato che il motore riesce a controllare solo la variabile q_1. Le equazioni dinamiche sono quindi riassumibili con (3.1)

$$\begin{aligned}
&\begin{bmatrix} \pi_1 + \pi_2 \sin^2 q_2 + \pi_3 \cos^2 q_2 & \pi_4 \cos q_2 \\ \pi_4 \cos q_2 & \pi_7 \end{bmatrix} \ddot{\boldsymbol{q}} + \\
&\begin{bmatrix} \pi_6 + \pi_5 \sin(2q_2)\dot{q}_2 & -\pi_4 \sin q_2 \dot{q}_2 + \pi_5 \sin(2q_2)\dot{q}_1 \\ -\pi_5 \sin(2q_2)\dot{q}_1 & \pi_8 \end{bmatrix} \dot{\boldsymbol{q}} + \\
&\begin{bmatrix} 0 \\ \pi_9 \sin q_2 \end{bmatrix} = \begin{bmatrix} \tau \\ 0 \end{bmatrix}
\end{aligned} \tag{3.2}$$

dove le quantità π_i, sulla base dei parametri meccanici elencati in tabella 3.1, sono definiti come segue:

$$\begin{aligned}
\pi_1 &= J_{z_0} + m_1 l_1^2 + m_2 L_1^2 & \pi_2 &= J_{y_2} + m_2 l_2^2 \\
\pi_3 &= J_{x_2} & \pi_4 &= m_2 L_1 l_2 \\
\pi_5 &= \frac{1}{2}\left(J_{y_2} - J_{x_2} + m_2 l_2^2\right) & \pi_6 &= c_1 \\
\pi_7 &= J_{z_2} + m_2 l_2^2 & \pi_8 &= c_2 \\
\pi_9 &= m_2 l_2 g
\end{aligned}$$

con g accelerazione gravitazionale.

Al fine di cambiare la posizione verticale del pendolo da $q_2 = \pi$ a $q_2 = 0$ si provveda alla seguente trasformazione di coordinate:

$$\boldsymbol{\theta} = \begin{bmatrix} q_1 \\ q_2 - \pi \end{bmatrix}$$

Attraverso tale cambiamento di variabili, e scegliendo come vettore di stato $\boldsymbol{x} = \left[\boldsymbol{\theta}^T, \dot{\boldsymbol{\theta}}^T\right]^T$, è allora possibile riscrivere la (3.2) attraverso il seguente modello con equilibrio nell'origine:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\boldsymbol{\theta}} \\ -B(\boldsymbol{\theta})^{-1}\left[C(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})\dot{\boldsymbol{\theta}} + G(\boldsymbol{\theta})\right] \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ B(\boldsymbol{\theta})^{-1} \end{bmatrix} \boldsymbol{\tau} \tag{3.3}$$

La coppia τ è generata da un motore in corrente continua, modellato con la seguente dinamica lineare del primo ordine:

$$L_a\dot{\tau} = KV - R_a\tau - K^2\dot{\theta}_1 \tag{3.4}$$

dove V è la tensione applicata al motore e K, L_a, R_a sono i suoi parametri (descritti in tabella 3.1).

Grandezza fisica	Simbolo	Unità di misura	
Massa del braccio	m_1	200×10^{-3}	[kg]
Massa dell'asta (pendolo)	m_2	72×10^{-3}	[kg]
Lunghezza del braccio	L_1	224×10^{-3}	[m]
Centro di massa del braccio	l_1	144×10^{-3}	[m]
Centro di massa dell'asta	l_2	106×10^{-3}	[m]
Inerzia del braccio rispetto a z_0	J_{z_0}	0.9×10^{-3}	[kg m^2]
Inerzia del pendolo rispetto a x_2	J_{x_2}	1.65×10^{-6}	[kg m^2]
Inerzia del pendolo rispetto a y_2	J_{y_2}	2.7×10^{-4}	[kg m^2]
Inerzia del pendolo rispetto a z_2	J_{z_2}	2.71×10^{-4}	[kg m^2]
Attrito del braccio	c_1	0.9×10^{-2}	[N m s]
Attrito del pendolo	c_2	2.71×10^{-7}	[N m s]
Costante di coppia del motore	K	2.2274	[N m A^{-1}]
Induttanza del motore	L_a	0.044	[H]
Resistenza del motore	R_a	1.9	[Ω]

Tabella 3.1: Parametri del Pendolo di Furuta

3.2.2 Modello linearizzato

Per l'elaborazione del modello linearizzato dell'impianto meccanico si è proceduto ad una prima linearizzazione dell dinamica (3.3) nell'intorno del punto di equilibrio superiore, ottendo la seguente equazione (3.5):

$$\dot{x} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{\pi_4\pi_9}{d} & \frac{-\pi_6\pi_7}{d} & \frac{-\pi_4\pi_8}{d} \\ 0 & \frac{\pi_9(\pi_1+\pi_3)}{d} & \frac{-\pi_4\pi_6}{d} & \frac{-\pi_8(\pi_1+\pi_3)}{d} \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ \frac{\pi_7}{d} \\ \frac{\pi_4}{d} \end{bmatrix} \tau \tag{3.5}$$

con $d = -\pi_4^2 + \pi_7(\pi_1 + \pi_3)$. Dopo tale elaborazione è stato necessario combinare le dinamiche (3.4) e (3.5), considerando la coppia τ come uno stato aggiuntivo della dinamica e la tensione V come un ingresso, ottenendo così il modello linearizzato dell'intero sistema (pendolo + motore).

3.2.3 Sistema reale

Per le prove effettuate è stato utilizzato il Pendolo di Furuta rappresentato nella seguente figura 3.7.

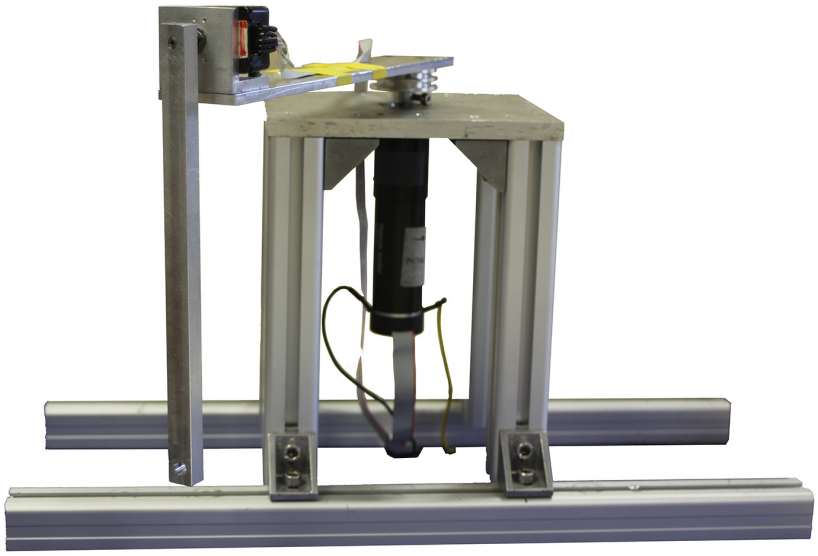

Figura 3.7: Pendolo di Furuta utilizzato per il test

Il sistema meccanico è equipaggiato con due encoder: uno capace di misurare l'angolo θ_1 con una risoluzione di $2\pi/176128$ rad ed uno associato all'angolo θ_2 con risoluzione $2\pi/2000$ rad; il braccio orizzontale è attuato attraverso un motore DC a sua volta comandato attraverso una tensione compresa tra -24V e +24V.
Oltre alle dinamiche precedente elaborate, come spesso accade l'impianto reale è affetto da alcuni fenomeni fisici non modellati, come l'attrito di Coulomb tra i materiali a contatto ed il gioco esistente tra braccio orizzontale e motore (dovuto alla presenza di un riduttore meccanico di velocità).

3.2.4 Logica di controllo

Il metodo di controllo utilizzato per questo sistema è stato diviso nei seguenti due step:

- *Swing-Up*
  In questa fase il pendolo parte dalla posizione di equilibrio stabile (rivolto verso il basso) e viene portato nella posizione di equilibrio instabile (rivolto verso l'alto). Per l'evoluzione di questa manovra viene inizialmente applicata una coppia costante al braccio orizzontale (scegliendone arbitrariamente la direzione), per poi invertirla non appena la velocità del pendolo si annulla; attraverso qusta procedura l'angolo descritto dal pendolo raddoppia ad ogni inversione. Data la difficoltà nella precisa rilevazione della velocità del pendolo, la tecnica appena descritta è stata semplificata come segue:
  - si applica una coppia positiva quando il pendolo ha velocità positiva e $mod(\theta_2, 2\pi)$ è minore di π
  - si applica una coppia negativa quando il pendolo ha velocità negativa e $mod(\theta_2, 2\pi)$ è maggiore di π
  - si applica una coppia nulla altrimenti

Con riferimento all'equazione (3.4), e trascurando i transitori della dinamica del motore ($\dot{\tau} = 0$), la tensione da applicare al motore per ottenere la coppia $\bar{\tau}$ è pari a:

$$V = \frac{R_a \bar{\tau} + K^2 \dot{\theta}_1}{K}$$

- *Stabilizzazione*
  In questa fase il controllore deve mantenere il pendolo nella posizione superiore e, a tale scopo, vengono utilizzati un osservatore di stato (di tipo Luenberger) ed un feedback dello stato del precedente modello linaearizzato 3.2.2. Al fine di stabilizzare il sistema nella posizione superiore [4], l'osservaore ha in input
  - la posizione angolare del braccio orizzontale
  - la tensione di alimentazione del motore
  - lo scarto tra la posizione angolare del pendolo ed il relativo valore di riferimento [5]

Lo scambio tra la prima e la seconda fase del controllo è effettuata automaticamente non appena il pendolo entra in un determinato intorno della posizione superiore.

3.2.5 Definizione dell'ambiente software

In questo primo esempio, data la complessità dei vari elementi utilizzati nella modellazione (ed organizzati schematicamente secondo la figura 3.8), si è scelto di ampliare le librerie con i blocchi riportati nelle seguenti tabelle 3.2, 3.3 e 3.4[6]:

[4] Definita dal vettore di stato $\boldsymbol{x} = [0 \quad 2k\pi \quad 0 \quad 0]^T, \tau = 0$ con $k \in \mathbb{Z}$

[5] Per θ_2 compreso tra $-\pi$ e π il valore di riferimento è pari a 0

[6] Per la loro definizione si rimanda alle sezioni dell'Appendice A

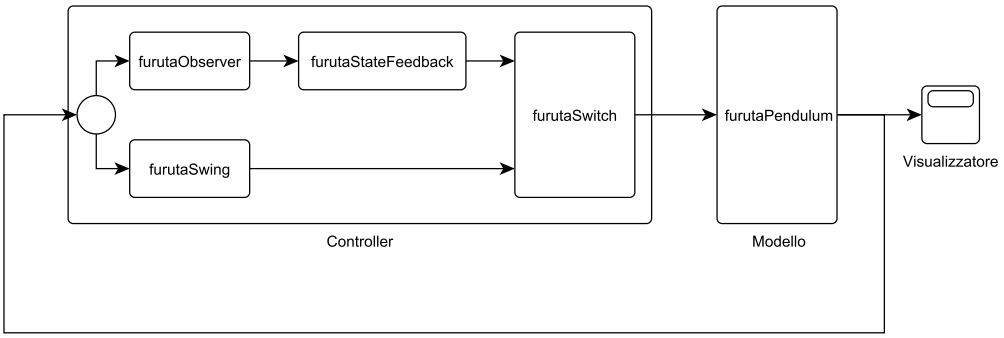

Figura 3.8: Schema a blocchi del modello simulato

Nome	#In	#Out	Funzione
furutaObserver	3	1	Luenberger del sistema
furutaStateFeedback	1	1	Retroazione dello stato stimato
furutaSwing	1	2	Esecuzione prima fase di controllo
furutaSwitch	2	2	Cambiamento fase di controllo
furutaPendulum	1	2	Elabora il modello

Tabella 3.2: Blocchi definiti per la modellazione

Nome	Applicazione	#Canali	Varianti	Note
SerialCS_FP	Controller	1	-	Conversione da forza a tensione e gestione comando PWM
FlexActuator	Virtualplant	1	-	-

Tabella 3.3: Nodi attuatori definiti per la modellazione

Seguendo la procedura definita nella precedente sezione 3.1 sono stati quindi elaborati i seguenti passi:

1. Distinzione tra ambiente di controllo ed ambiente di simulazione, come rappresentato nella seguente figura 3.9

2. Scelta dei seguenti nodi esterni:

Nome	Applicazione	#Canali	Varianti	Note
Encoder	Controller	2	r-u	Conversione da PWM a tensione e decodifica impulsi encoder
FlexSensor	Virtual	2	r-u	-

Tabella 3.4: Nodi sensori definiti per la modellazione

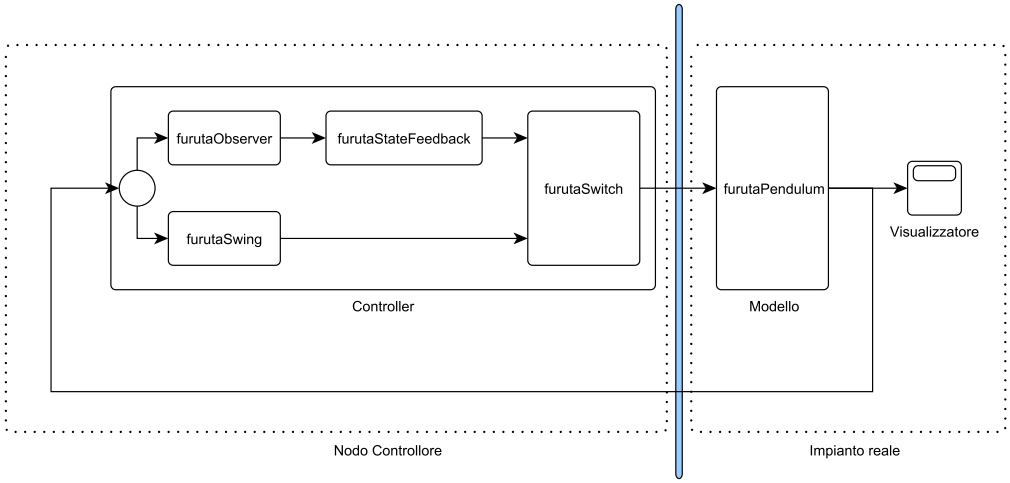

Figura 3.9: Limitazione degli ambienti di controllo e simulazione

- Un nodo attuatore di tipo SerialCS_FP (orizzonte di controllo costituito dalla tensione del motore)
- Due nodi sensori di tipo Encoder
  - Uno associato alla tensione applicata al motore DC e alla relativa posizione (θ_1) e definito attraverso la variante U
  - Uno associato alla sola posizione angolare del pendolo (θ_2) e definito attraverso la variante R

da collegare come rappresentato in figura 3.10

3. Introduzione dei seguenti modelli per la continuità del loop, secondo l'interazione Simulator/Local Dynamics prevista nella sezione 2.2:

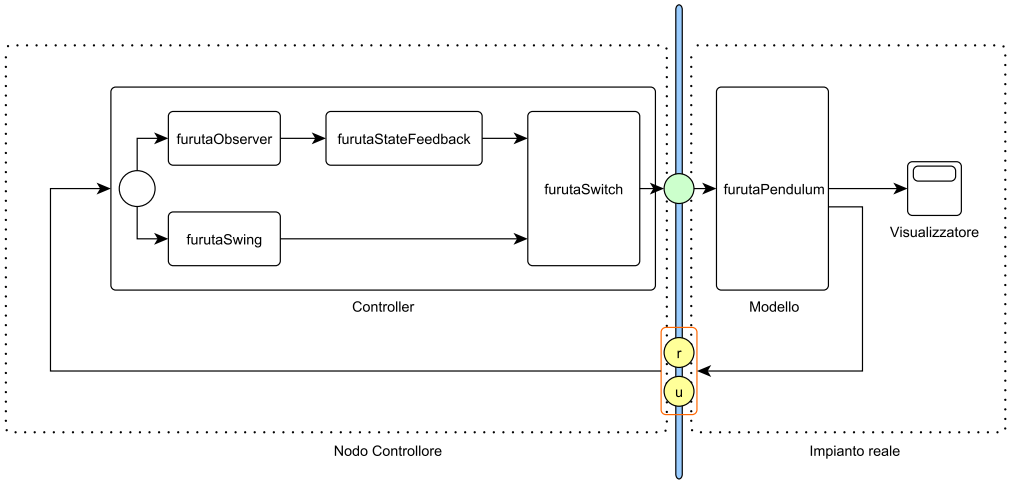

Figura 3.10: Introduzione di nodi sensori ed attuatori

- Un modello dell'impianto (da introdurre nel sottoblocco Simulator del modulo Controller) per il ripristino della catena di feedback
- Un osservatore di stato (da introdurre nel sottoblocco Local Dynamics del modulo Controller) per l'aggiornamento degli stati del precedente modello attraverso l'elaborazione dei pacchetti dei nodi sensori [7]

Con l'introduzione di quanto sopra il modello di figura 3.10 si traduce in quello di figura 3.11.

4. Si verifica che la struttura fin qui ottenuta rispetta l'architettura Simulator/Local Dynamics prevista nella sezione 2.2.2 e che non sono necessari ulteriori semplificazioni

5. Nel caso in esame, oltre al modello di impianto definito al passo (3) e al relativo osservatore), non sono presenti ulteriori modelli di questo tipo e non sono quindi necessari ulteriori osservatori.

[7] Vedi sezione 2.2.2

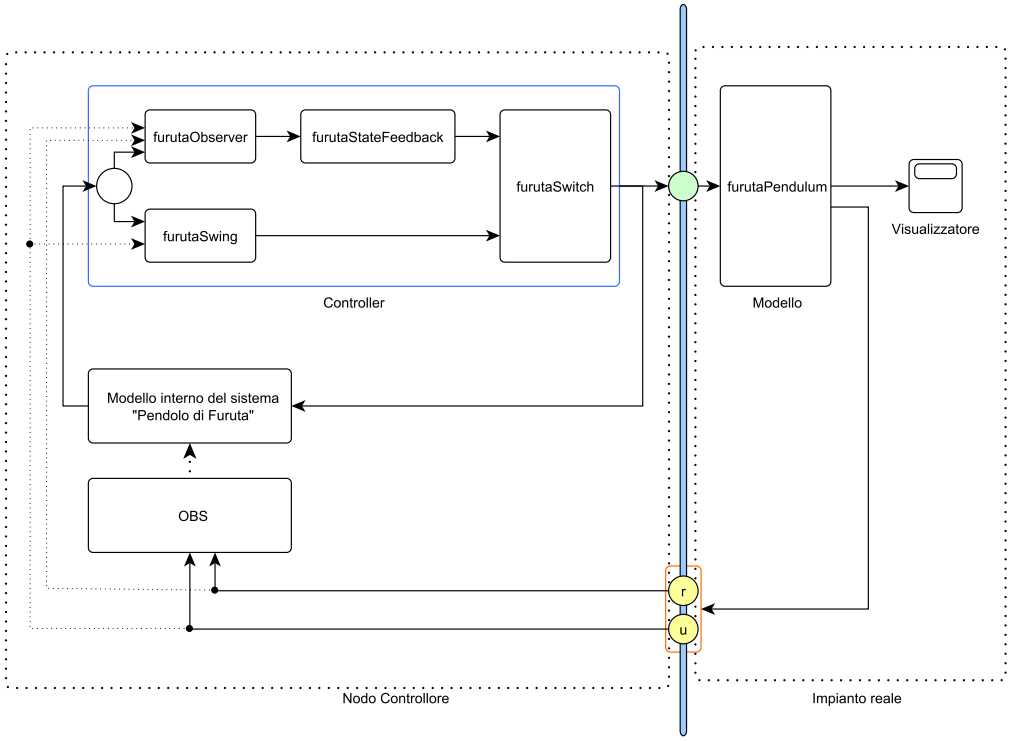

Figura 3.11: Ripristino dei collegamenti di feedback

6. Il sistema di controllo è adesso conforme all'archietuttura prevista dall'ambiente software ed è necessario editare i file introdotti nella sezione 2.1 e riassumibili con la seguente gerarchia:

Cartella CONTROLLER		
Sottocartella	**File**	**Tipo**
OBSERVER	DELAY_R.ncs	delay
OBSERVER	DELAY_U.ncs	delay
OBSERVER	GAIN_R.ncs	proportional
OBSERVER	GAIN_U.ncs	proportional
OBSERVER	SUM_R.ncs	sum
OBSERVER	SUM_U.ncs	sum
OBSERVER	MUX_OUT.ncs	mux

Continua nella pagina successiva

Cartella CONTROLLER		
Sottocartella	**File**	**Tipo**
OBSERVER	conf_block.ncs	-
-	furutaOBSERVER.ncs	furutaOBSERVER
-	furutaPENDULUM.ncs	furutaPENDULUM
-	furutaSTATEFEEDBACK.ncs	furutaSTATEFEEDBACK
-	furutaSWING.ncs	furutaSWING
-	furutaSWITCH.ncs	furutaSWITCH
-	conf_block.ncs	-
-	ACTUATOR.sbc	-
-	OBSERVER.oic	-
-	furutaOBSERVER.oic	-
-	furutaSWING.oic	-
-	conf_sim.sbc	-
-	onf_con.con	-
-	conf_par.par	-

Cartella VIRTUALPLANT		
Sottocartella	**File**	**Tipo**
-	furutaPENDULUM.ncs	furutaPENDULUM
-	conf_block.ncs	-
-	FLEXACTUATOR.sbc	-
-	FLEXSENSOR.oic	-
-	conf_sim.sbc	-
-	conf_con.con	-
-	conf_par.par	-

Cartella VIRTUALPLANT_UDP		
Sottocartella	**File**	**Tipo**
-	furutaPENDULUM.ncs	furutaPENDULUM
-	conf_block.ncs	-
-	FLEXACTUATOR.sbc	-
-	FLEXSENSOR.oic	-
-	conf_sim.sbc	-
-	conf_con.con	-
-	conf_con_ext.con	-
-	conf_par.par	-

Per una completa analisi dei contenuti di tali file si faccia riferimento alle appendici A e B.

7. Il test di controllo effettuato sul modello sopra descritto è stato realizzato attraverso l'interazione dei moduli VirtualPlant e Controller; i parametri settati per tale simulazione sono riassumibili come segue:

   - Orizzonte di controllo: 40ms
   - Orizzonte sensori: 10ms
   - Ritardo di trasmissione su rete: 10ms con distribuzione normale e varianza di 2ms[8]

   I risultati ottenuti da tale simulazione di controllo sono rappresentati nella seguente figura 3.12.

[8]Tale ritardo è stato introdotto attraverso il comando "tc" utilizzato su piattaforma Unix per la gestione del traffico dati su rete; questo particolare impostazione è stata effettuata attraverso le seguenti stringhe di comando:

```
sudo tc qdisc add dev eth0 root netem delay 10ms 2ms distribution normal
sudo tc qdisc add dev eth0 parent 1:1 pfifo limit 1000
```

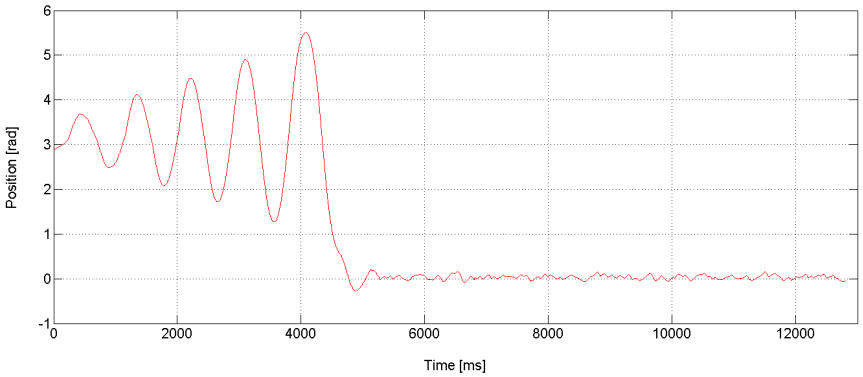

Figura 3.12: Evoluzione temporale dell'angolo θ_2

3.3 Cutting a Circle

In questa sezione verrà presentata, come secondo esempio applicativo del software realizzato, l'approccio ad un test di interesse industriale generalmente indicato con l'espressione "Cutting a circle" ed utilizzato all'interno dei metodi di valutazione dello standard ISO 25.040.20 (macchine a controllo numerico).

3.3.1 Modellazione non lineare

Il test viene normalmente effettuato per la valutazione delle performance di macchine CNC similari a quella rappresentat in figura 3.13 impegnate nell'inseguimento di una traiettoria circolare con raggio e velocità di rotazione prefissati.

Il modello di impianto utilizzato in questa analisi è stato estratto dal progetto CHAT ed implementa un movimento bidimensionale lungo gli assi X e Z (vedi figura 3.13).

Il punto di partenza di questo esempio è un modello elaborato su piattaforma Matlab® /Simulink® (rappresentato in figura 3.14) sufficientemente complesso da rappresentare tutte le dinamiche in gioco ma al contempo abbastanza semplificato da non rendere troppo onerosa la fase com-

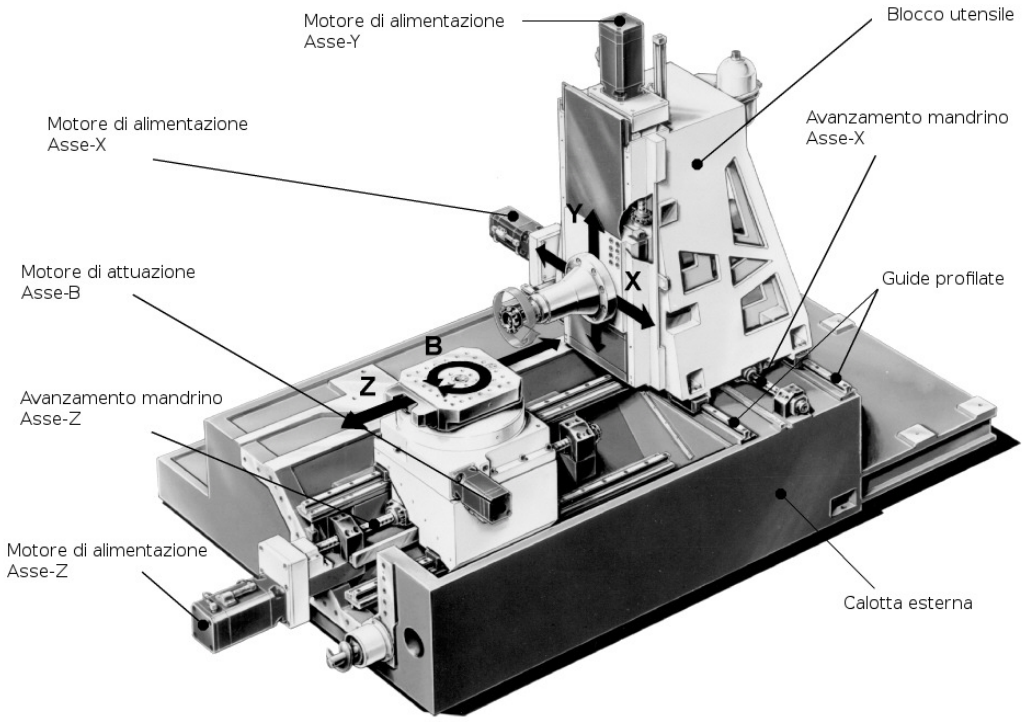

Figura 3.13: Esempio di macchina CNC utilizzata nel test

putazionale. Le dinamiche in esso elaborate possono essere schematizzate come segue:

- Dinamica asse X: Sottosistema non lineare con 3 input e 4 output composto da un sistema LTI (10 input, 20 output e 16 stati) ed un blocco non lineare per la modellazione dell'attrito statico nell'intorno dei punti di arresto dell'utensile
- Dinamica asse Z: Sottosistema non lineare con 3 input e 4 output composto da un sistema LTI (6 input, 12 output e 10 stati) ed un blocco non lineare per la modellazione dell'attrito statico nell'intorno dei punti di arresto dell'utensile
- Dinamica controllo asse X: Sottosistema per l'implementazione, lungo l'asse X, della logica di controllo descritta nella sezione 3.3.2

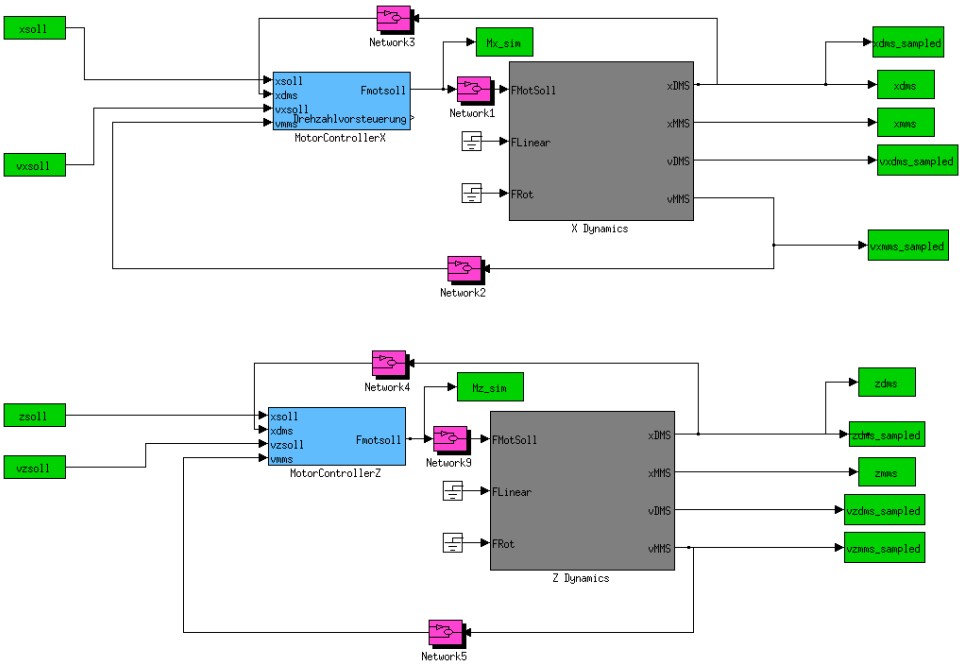

Figura 3.14: Modello Simulink® iniziale

- Dinamica controllo asse X: Sottosistema per l'implementazione, lungo l'asse Z, della logica di controllo descritta nella sezione 3.3.2

Per l'analisi della struttura di ciascun sottosistema si rimanda all' appendice B.

3.3.2 Logica di controllo

Come indicato nella precedente sezione (vedi figura 3.14), entrambi i sottosistemi relativi alla dinamica dei due assi sono rappresentati attraverso un blocco capace di generare, a partire da una certa sequenza di comandi in forza, la serie temporale di posizioni e velocità ad esso corrispondenti; definita una certa traiettoria di riferimento i due sottosistemi di controllo dovranno quindi elaborare la corretta sequenza di forze affinchè l'utensile la insegua.
Il sistema di controllo utilizzato in questo modello prevede che la traietto-

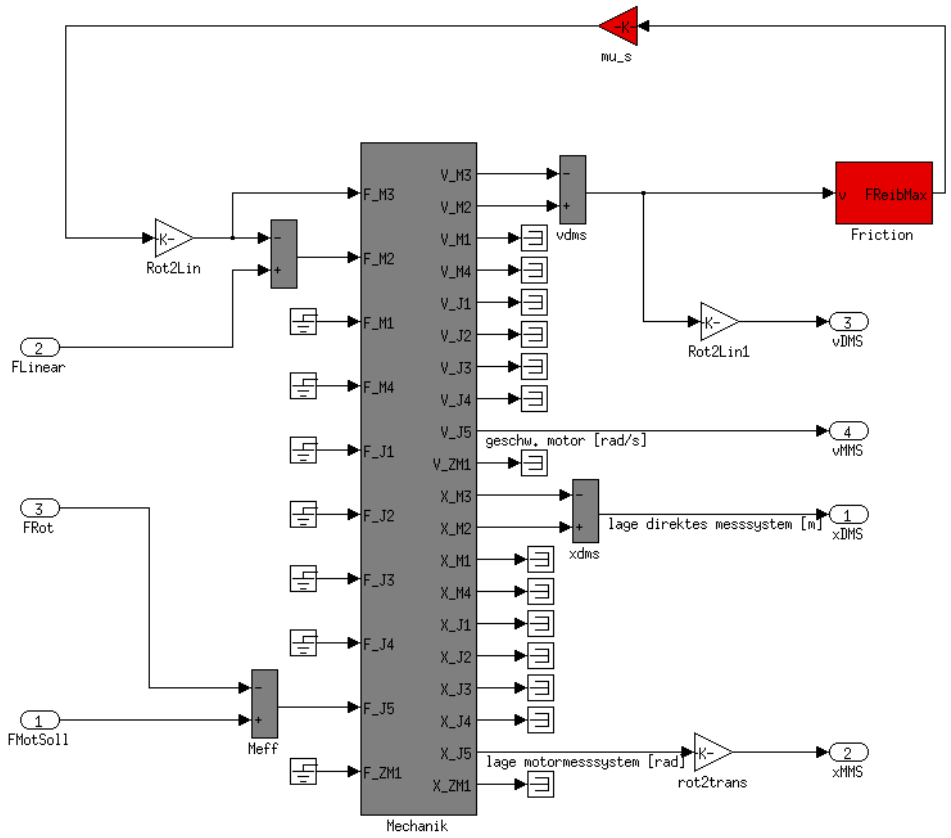

Figura 3.15: Modello della dinamica dell'asse X

ria di riferimento sia definita attraverso i quattro seguenti profili temporali:

- $(t[k],\ p_x[k])$: sequenza temporale della posizione dell'utensile lungo l'asse X
- $(t[k],\ v_x[k])$: sequenza temporale della velocità dell'utensile lungo l'asse X
- $(t[k],\ p_z[k])$: sequenza temporale della posizione dell'utensile lungo l'asse Z

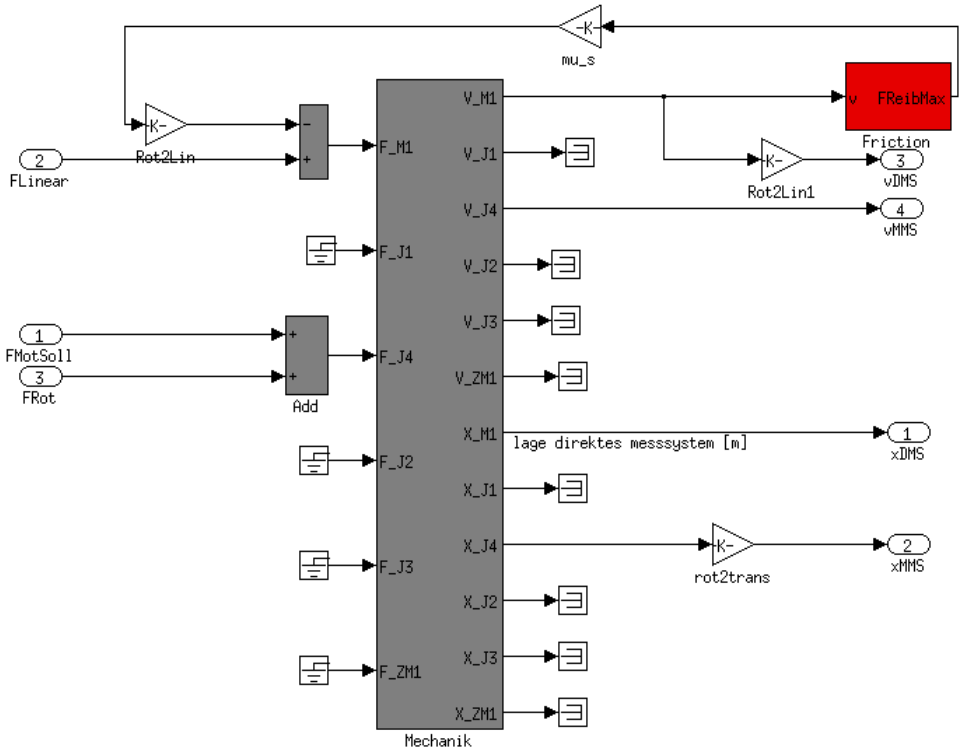

Figura 3.16: Modello della dinamica dell'asse Z

- $(t[k],\ v_z[k])$: sequenza temporale della velocità dell'utensile lungo l'asse Z

e ciascuno dei due sottosistemi di controllo elabora la propria sequenza di forze attraverso le strutture in feedback presentate in figura 3.17 e 3.18.

Per essere in grado di distinguere le prestazioni di controllo reali dagli effetti indotti dalle fasi di avvio ed arresto, la traiettoria di riferimento viene estesa (all'inizio e alla fine) con fasi di accelerazione e decelerazione che rendono nulla la velocità angolare iniziale e finale della lavorazioe.

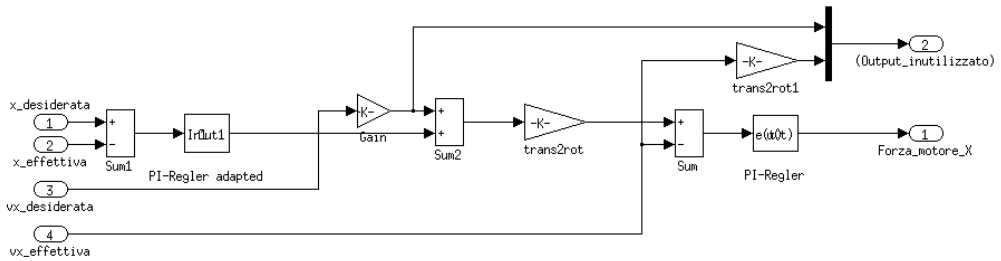

Figura 3.17: Struttura feedback del controllo lungo l'asse X

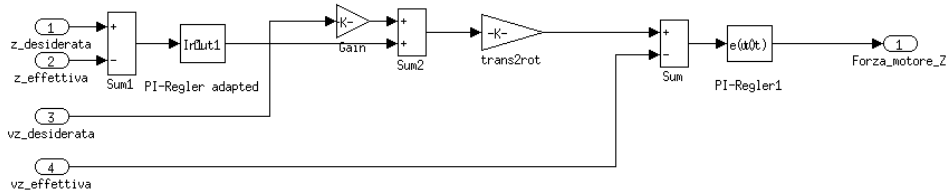

Figura 3.18: Struttura feedback del controllo lungo l'asse Z

3.3.3 Definizione dell'ambiente software

Il modello simulato rispecchia fedelmente quello fornito su piattaforma Matlab® /Simulink® (rappresentato in figura 3.14), quindi sia i parametri dei blocchi funzionali sia i profili di posizione e velocità desiderati sono esattamente congruenti con quelli originali.

Seguendo la procedura definita nella precedente sezione 3.1 sono stati quindi elaborati i seguenti passi:

1. Distinzione tra ambiente di controllo ed ambiente di simulazione, come rappresentato nella seguente figura 3.20

2. Scelta dei seguenti nodi esterni:

   - Un nodo attuatore di tipo GenAct2_32 associato ad entrambi gli orizzonti di controllo (forze da applicare rispettivamente all'asse X e all'asse Z)

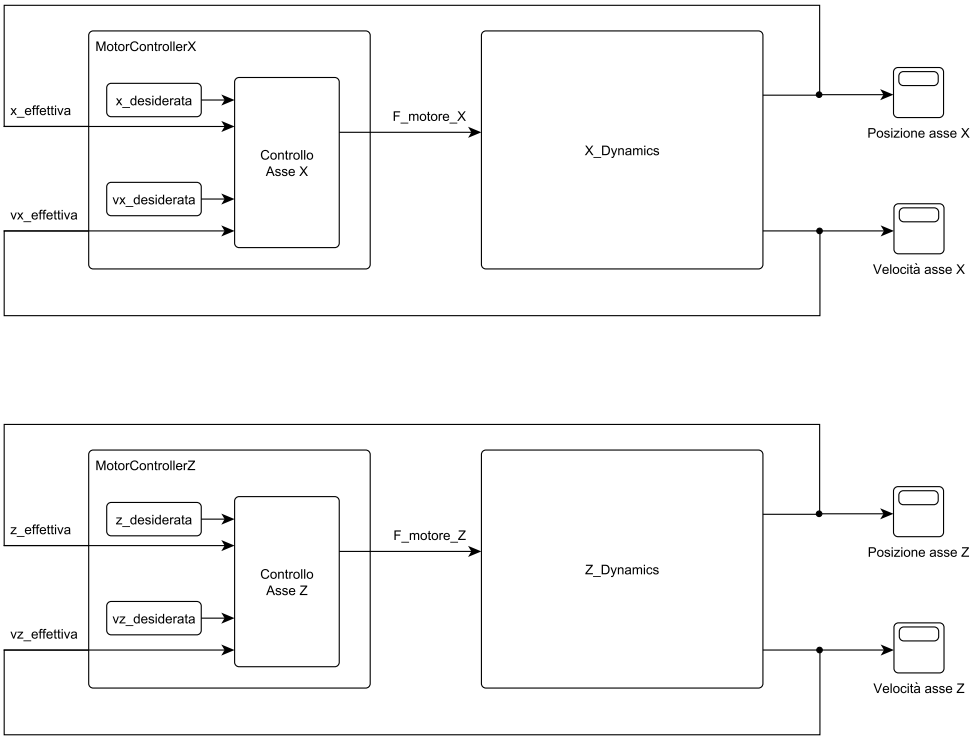

Figura 3.19: Schema a blocchi del modello simulato

- Quattro nodi sensore come segue (per l'analisi dei canali collegati si faccia riferimento alla successiva tabella 3.8):
  - Un GensSens2_32 in variante "a", utilizzato come input per il MotorControllerX e collegato ai canali 21 e 24
  - Un GensSens2_32 in variante "b", utilizzato come input per il MotorControllerZ e collegato ai canali 38 e 41
  - Un GensSens21_32 in variante "c", utilizzato come input per il successivo Osservatore di stato della XDynamic e collegato ai canali 0/20
  - Un GensSens13_32 in variante "d", utilizzato come input per il successivo Osservatore di stato della ZDynamic e collegato ai canali 25/37

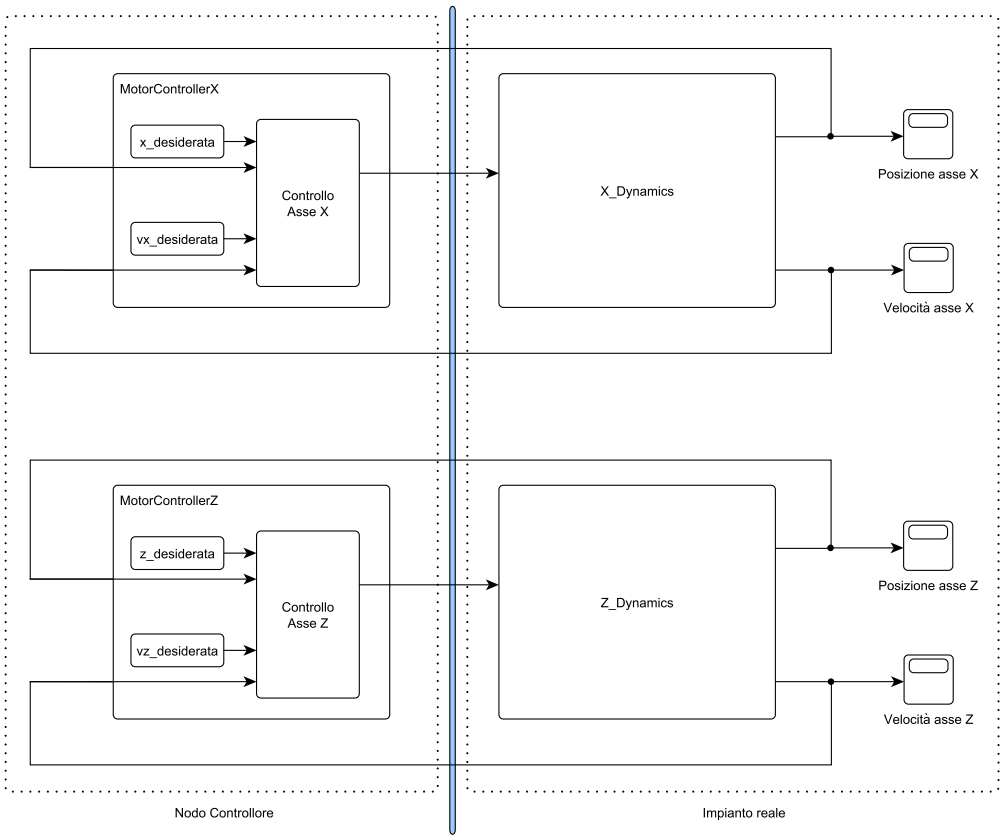

Figura 3.20: Limitazione degli ambienti di controllo e simulazione

da collegare come rappresentato in figura 3.21

Tabella 3.8: Mappa dei canali di output

N	Origine	Note
0	X_Dynamic/N(0)	Segnale di controllo asse X
1	X_Dynamic/Mechanik(0)	-
2	X_Dynamic/Mechanik(1)	-
3	X_Dynamic/Mechanik(2)	-
4	X_Dynamic/Mechanik(3)	-
5	X_Dynamic/Mechanik(4)	-

Continua nella pagina successiva

N	Origine	Note
6	X_Dynamic/Mechanik(5)	-
7	X_Dynamic/Mechanik(6)	-
8	X_Dynamic/Mechanik(7)	-
9	X_Dynamic/Mechanik(8)	-
10	X_Dynamic/Mechanik(9)	-
11	X_Dynamic/Mechanik(10)	-
12	X_Dynamic/Mechanik(11)	-
13	X_Dynamic/Mechanik(12)	-
14	X_Dynamic/Mechanik(13)	-
15	X_Dynamic/Mechanik(14)	-
16	X_Dynamic/Mechanik(15)	-
17	X_Dynamic/Mechanik(16)	-
18	X_Dynamic/Mechanik(17)	-
19	X_Dynamic/Mechanik(18)	-
20	X_Dynamic/Mechanik(19)	-
21	X_Dynamic/XDMS	Posizione asse X
22	X_Dynamic/ROT2TRANS	-
23	X_Dynamic/ROT2LIN1	-
24	X_Dynamic/Mechanik(8)	Velocità asse X
25	Z_Dynamic/N(0)	Segnale di controllo asse Z
26	Z_Dynamic/Mechanik(0)	-
27	Z_Dynamic/Mechanik(1)	-
28	Z_Dynamic/Mechanik(2)	-
29	Z_Dynamic/Mechanik(3)	-
30	Z_Dynamic/Mechanik(4)	-
31	Z_Dynamic/Mechanik(5)	-
32	Z_Dynamic/Mechanik(6)	-
33	Z_Dynamic/Mechanik(7)	-
34	Z_Dynamic/Mechanik(8)	-

Continua nella pagina successiva

N	Origine	Note
35	Z_Dynamic/Mechanik(9)	-
36	Z_Dynamic/Mechanik(10)	-
37	Z_Dynamic/Mechanik(11)	-
38	Z_Dynamic/Mechanik(6)	Posizione asse Z
39	Z_Dynamic/ROT2TRANS	-
40	Z_Dynamic/ROT2LIN1	-
41	Z_Dynamic/Mechanik(0)	Velocità asse Z

3. Introduzione dei seguenti modelli per la continuità del loop, secondo l'interazione Simulator/Local Dynamics prevista nella sezione 2.2:

   - Un modello della XDynamics (da introdurre nel sottoblocco Simulator del modulo Controller) per il ripristino della catena di feedback
   - Un osservatore di stato (da introdurre nel sottoblocco Local Dynamics del modulo Controller) per l'aggiornamento degli stati del precedente modello attraverso l'elaborazione dei pacchetti dei nodi sensori GenSens21_32 (variante "c")[9]
   - Un modello della ZDynamics (da introdurre nel sottoblocco Simulator del modulo Controller) per il ripristino della catena di feedback
   - Un osservatore di stato (da introdurre nel sottoblocco Local Dynamics del modulo Controller) per l'aggiornamento degli stati del precedente modello attraverso l'elaborazione dei pacchetti dei nodi sensori GenSens13_32 (variante "d")[10]

   Con l'introduzione di quanto sopra il modello di figura 3.21 si traduce in quello di figura 3.22.

[9] Vedi sezione 2.2.2

[10] Vedi sezione 2.2.2

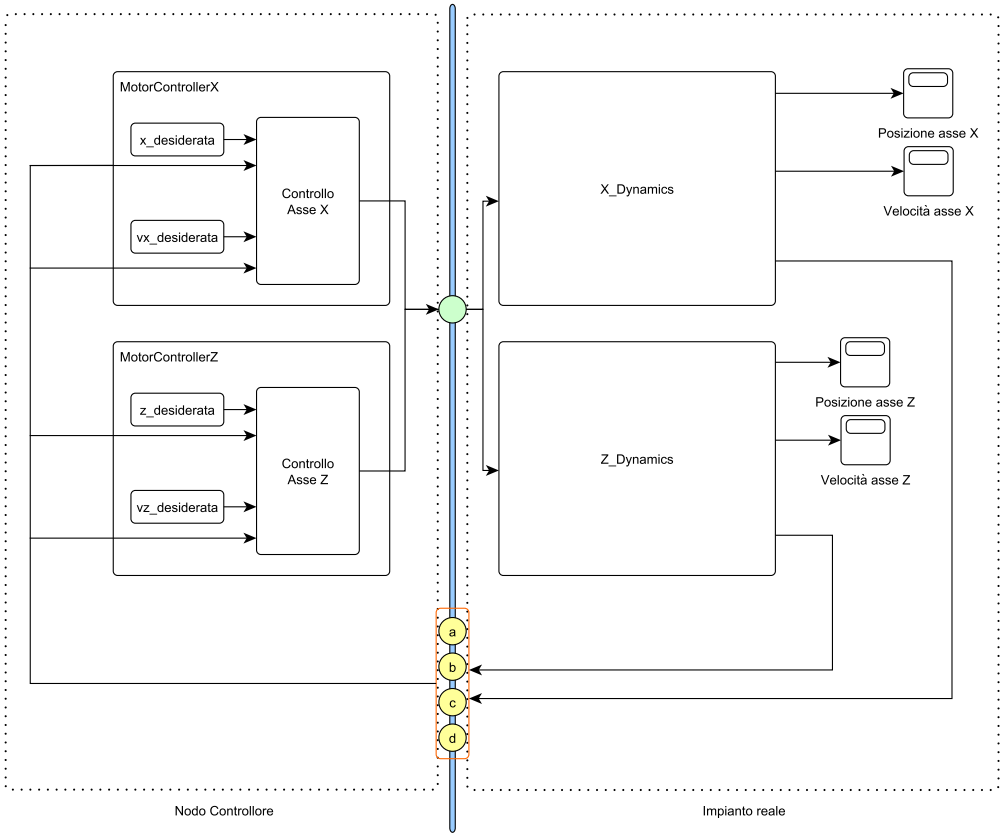

Figura 3.21: Introduzione di nodi sensori ed attuatori

4. Si verifica che la struttura fin qui ottenuta rispetta l'architettura Simulator/Local Dynamics prevista nella sezione 2.2.2 e che non sono necessari ulteriori semplificazioni

5. Nel caso in esame, oltre ai modelli delle dinamiche definiti al passo (3) e ai relativi osservatori), non sono presenti ulteriori modelli di questo tipo e non sono quindi necessari ulteriori osservatori.

6. Il sistema di controllo è adesso conforme all'archietuttura prevista dall'ambiente software ed è necessario editare i file introdotti nella sezione 2.1 e riassumibili con la seguente gerarchia:

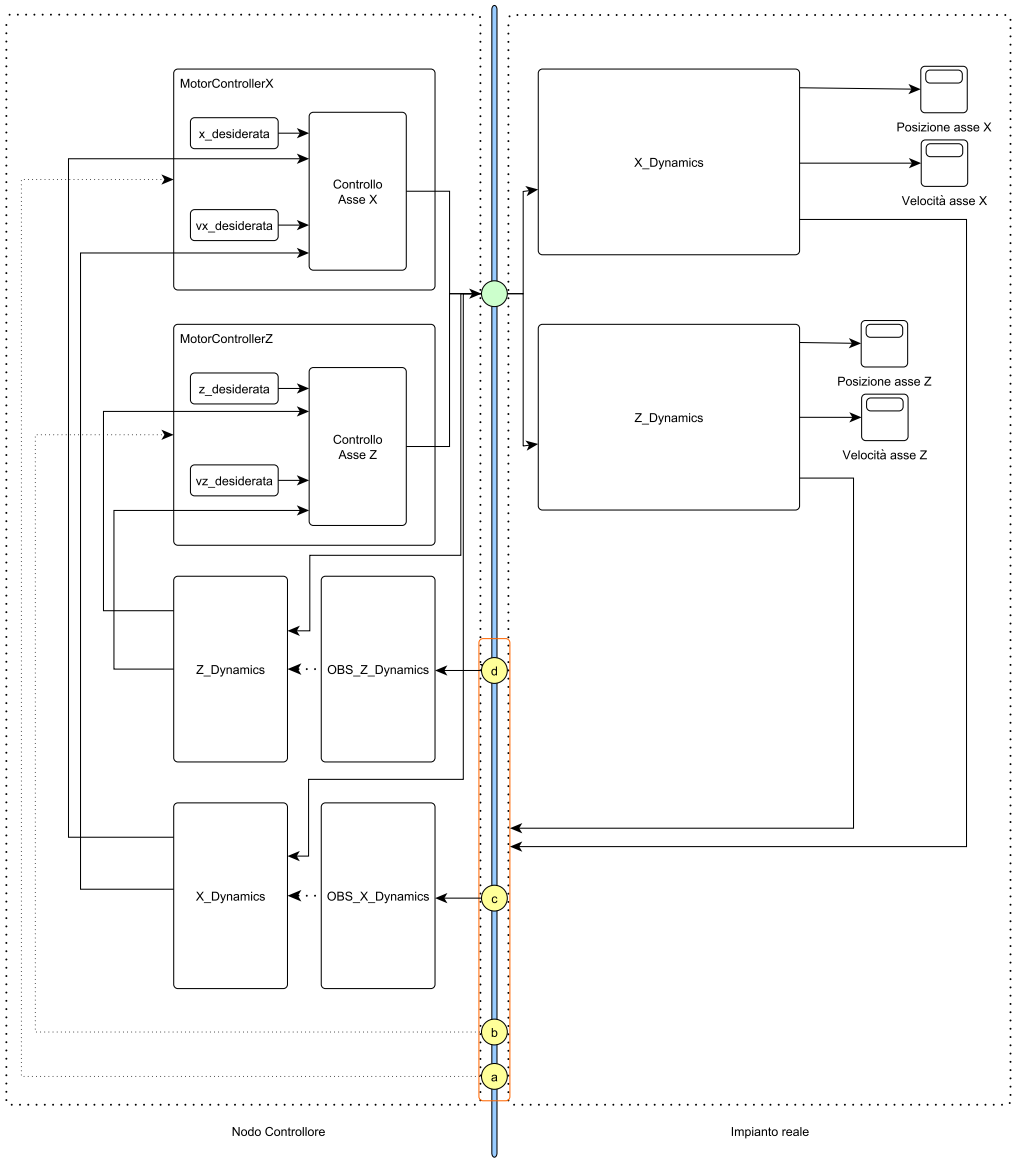

Figura 3.22: Ripristino dei collegamenti di feedback

Cartella CONTROLLER		
Sottocartella	**File**	**Tipo**
MotorControllerX	GAIN.ncs	proportional
		Continua nella pagina successiva

Cartella CONTROLLER		
Sottocartella	**File**	**Tipo**
MotorControllerX	PIREGLER.ncs	discrete_pid
MotorControllerX	PIREGLERA.ncs	discrete_pid
MotorControllerX	SUM.ncs	sum
MotorControllerX	SUM1.ncs	sum
MotorControllerX	SUM2.ncs	sum
MotorControllerX	TRANS2ROT.ncs	proportional
MotorControllerX	TRANS2ROT1.ncs	proportional
MotorControllerX	XSOLL.ncsx	source
MotorControllerX	XSOLL.txt	-
MotorControllerX	VXSOLL.ncsx	source
MotorControllerX	VXSOLL.txt	-
MotorControllerX	conf_block.ncs	-
MotorControllerZ	GAIN.ncs	proportional
MotorControllerZ	PIREGLER.ncs	discrete_pid
MotorControllerZ	PIREGLERA.ncs	discrete_pid
MotorControllerZ	SUM.ncs	sum
MotorControllerZ	SUM1.ncs	sum
MotorControllerZ	SUM2.ncs	sum
MotorControllerZ	TRANS2ROT.ncs	proportional
MotorControllerZ	ZSOLL.ncsx	source
MotorControllerZ	ZSOLL.txt	-
MotorControllerZ	VZSOLL.ncsx	source
MotorControllerZ	VZSOLL.txt	-
MotorControllerZ	conf_block.ncs	-
OBS_X_Dynamics	LUENBERGER.ncsx	abcd_MTL4
OBS_X_Dynamics	MUX.ncs	mux
OBS_X_Dynamics	conf_block.ncs	-

Continua nella pagina successiva

Cartella CONTROLLER		
Sottocartella	**File**	**Tipo**
OBS_Z_Dynamics	LUENBERGER.ncsx	abcd_MTL4
OBS_Z_Dynamics	MUX.ncs	mux
OBS_Z_Dynamics	conf_block.ncs	-
X_DYN/REIBUNG/REG	C1.ncs	constant
X_DYN/REIBUNG/REG	C2.ncs	constant
X_DYN/REIBUNG/REG	C3.ncs	constant
X_DYN/REIBUNG/REG	C5.ncs	constant
X_DYN/REIBUNG/REG	C6.ncs	constant
X_DYN/REIBUNG/REG	C7.ncs	constant
X_DYN/REIBUNG/REG	D1.ncsx	division
X_DYN/REIBUNG/REG	F1.ncs	product
X_DYN/REIBUNG/REG	F2.ncsx	scalar_function
X_DYN/REIBUNG/REG	F3.ncsx	scalar_function
X_DYN/REIBUNG/REG	G1.ncs	proportional
X_DYN/REIBUNG/REG	G2.ncs	proportional
X_DYN/REIBUNG/REG	P1.ncs	product
X_DYN/REIBUNG/REG	P2.ncs	product
X_DYN/REIBUNG/REG	P3.ncs	product
X_DYN/REIBUNG/REG	P4.ncs	product
X_DYN/REIBUNG/REG	P5.ncs	product
X_DYN/REIBUNG/REG	S1.ncs	sum
X_DYN/REIBUNG/REG	S2.ncs	sum
X_DYN/REIBUNG/REG	conf_block.ncs	-
X_DYN/REIBUNG	SMOOTHD.ncs	discrete_pid
X_DYN/REIBUNG	conf_block.ncs	-
X_DYN	DEMUX.ncs	demux
X_DYN	GROUNG.ncs	constant

Continua nella pagina successiva

Cartella CONTROLLER		
Sottocartella	**File**	**Tipo**
X_DYN	MECHANIK.ncsx	ABCD_MTL4
X_DYN	MEFF.ncs	sum
X_DYN	MEFF1.ncs	sum
X_DYN	MU_S.ncs	proportional
X_DYN	MUX.ncs	mux
X_DYN	ROT2LIN.ncs	proportional
X_DYN	ROT2LIN1.ncs	proportional
X_DYN	ROT2TRANS.ncs	proportional
X_DYN	VDMS.ncs	sum
X_DYN	XDMS.ncs	sum
X_DYN	conf_block.ncs	-
Z_DYN/REIBUNG/REG	C1.ncs	constant
Z_DYN/REIBUNG/REG	C2.ncs	constant
Z_DYN/REIBUNG/REG	C3.ncs	constant
Z_DYN/REIBUNG/REG	C5.ncs	constant
Z_DYN/REIBUNG/REG	C6.ncs	constant
Z_DYN/REIBUNG/REG	C7.ncs	constant
Z_DYN/REIBUNG/REG	D1.ncsx	division
Z_DYN/REIBUNG/REG	F1.ncs	product
Z_DYN/REIBUNG/REG	F2.ncsx	scalar_function
Z_DYN/REIBUNG/REG	F3.ncsx	scalar_function
Z_DYN/REIBUNG/REG	G1.ncs	proportional
Z_DYN/REIBUNG/REG	G2.ncs	proportional
Z_DYN/REIBUNG/REG	P1.ncs	product
Z_DYN/REIBUNG/REG	P2.ncs	product
Z_DYN/REIBUNG/REG	P3.ncs	product
Z_DYN/REIBUNG/REG	P4.ncs	product
Z_DYN/REIBUNG/REG	P5.ncs	product

Continua nella pagina successiva

Cartella CONTROLLER		
Sottocartella	**File**	**Tipo**
Z_DYN/REIBUNG/REG	S1.ncs	sum
Z_DYN/REIBUNG/REG	S2.ncs	sum
Z_DYN/REIBUNG/REG	conf_block.ncs	-
Z_DYN_REIBUNG	SMOOTHD.ncs	discrete_pid
Z_DYN_REIBUNG	conf_block.ncs	-
Z_DYN	ADD.ncs	sum
Z_DYN	DEMUX.ncs	demux
Z_DYN	GROUNG.ncs	constant
Z_DYN	MECHANIK.ncsx	ABCD_MTL4
Z_DYN	MEFF1.ncs	sum
Z_DYN	MU_S.ncs	proportional
Z_DYN	MUX.ncs	mux
Z_DYN	ROT2LIN.ncs	proportional
Z_DYN	ROT2LIN1.ncs	proportional
Z_DYN	ROT2TRANS.ncs	proportional
Z_DYN	conf_block.ncs	-
-	GROUND.ncs	constant
-	X_SCOPE.ncs	scope
-	Z_SCOPE.ncs	scope
-	conf_block.ncs	-
-	MotorControllerX.oic	-
-	MotorControllerZ.oic	-
-	OBS_X_Dynamics.oic	-
-	OBS_Z_Dynamics.oic	-
-	Actuator.sbc	-
-	conf_con.con	-
-	conf_par.par	-

Continua nella pagina successiva

Cartella CONTROLLER		
Sottocartella	**File**	**Tipo**
-	conf_sim.sbc	-

Cartella VIRTUALPLANT		
Sottocartella	**File**	**Tipo**
X_DYN/REIBUNG/REG	C1.ncs	constant
X_DYN/REIBUNG/REG	C2.ncs	constant
X_DYN/REIBUNG/REG	C3.ncs	constant
X_DYN/REIBUNG/REG	C5.ncs	constant
X_DYN/REIBUNG/REG	C6.ncs	constant
X_DYN/REIBUNG/REG	C7.ncs	constant
X_DYN/REIBUNG/REG	D1.ncsx	division
X_DYN/REIBUNG/REG	F1.ncs	product
X_DYN/REIBUNG/REG	F2.ncsx	scalar_function
X_DYN/REIBUNG/REG	F3.ncsx	scalar_function
X_DYN/REIBUNG/REG	G1.ncs	proportional
X_DYN/REIBUNG/REG	G2.ncs	proportional
X_DYN/REIBUNG/REG	P1.ncs	product
X_DYN/REIBUNG/REG	P2.ncs	product
X_DYN/REIBUNG/REG	P3.ncs	product
X_DYN/REIBUNG/REG	P4.ncs	product
X_DYN/REIBUNG/REG	P5.ncs	product
X_DYN/REIBUNG/REG	S1.ncs	sum
X_DYN/REIBUNG/REG	S2.ncs	sum
X_DYN/REIBUNG/REG	conf_block.ncs	-
X_DYN/REIBUNG	SMOOTHD.ncs	discrete_pid
X_DYN/REIBUNG	conf_block.ncs	-
X_DYN	DEMUX.ncs	demux

Continua nella pagina successiva

Cartella VIRTUALPLANT		
Sottocartella	**File**	**Tipo**
X_DYN	GROUNG.ncs	constant
X_DYN	MECHANIK.ncsx	ABCD_MTL4
X_DYN	MEFF.ncs	sum
X_DYN	MEFF1.ncs	sum
X_DYN	MU_S.ncs	proportional
X_DYN	MUX.ncs	mux
X_DYN	ROT2LIN.ncs	proportional
X_DYN	ROT2LIN1.ncs	proportional
X_DYN	ROT2TRANS.ncs	proportional
X_DYN	VDMS.ncs	sum
X_DYN	XDMS.ncs	sum
X_DYN	conf_block.ncs	-
Z_DYN/REIBUNG/REG	C1.ncs	constant
Z_DYN/REIBUNG/REG	C2.ncs	constant
Z_DYN/REIBUNG/REG	C3.ncs	constant
Z_DYN/REIBUNG/REG	C5.ncs	constant
Z_DYN/REIBUNG/REG	C6.ncs	constant
Z_DYN/REIBUNG/REG	C7.ncs	constant
Z_DYN/REIBUNG/REG	D1.ncsx	division
Z_DYN/REIBUNG/REG	F1.ncs	product
Z_DYN/REIBUNG/REG	F2.ncsx	scalar_function
Z_DYN/REIBUNG/REG	F3.ncsx	scalar_function
Z_DYN/REIBUNG/REG	G1.ncs	proportional
Z_DYN/REIBUNG/REG	G2.ncs	proportional
Z_DYN/REIBUNG/REG	P1.ncs	product
Z_DYN/REIBUNG/REG	P2.ncs	product
Z_DYN/REIBUNG/REG	P3.ncs	product
Z_DYN/REIBUNG/REG	P4.ncs	product

Continua nella pagina successiva

Cartella VIRTUALPLANT		
Sottocartella	**File**	**Tipo**
Z_DYN/REIBUNG/REG	P5.ncs	product
Z_DYN/REIBUNG/REG	S1.ncs	sum
Z_DYN/REIBUNG/REG	S2.ncs	sum
Z_DYN/REIBUNG/REG	conf_block.ncs	-
Z_DYN_REIBUNG	SMOOTHD.ncs	discrete_pid
Z_DYN_REIBUNG	conf_block.ncs	-
Z_DYN	ADD.ncs	sum
Z_DYN	DEMUX.ncs	demux
Z_DYN	GROUNG.ncs	constant
Z_DYN	MECHANIK.ncsx	ABCD_MTL4
Z_DYN	MEFF1.ncs	sum
Z_DYN	MU_S.ncs	proportional
Z_DYN	MUX.ncs	mux
Z_DYN	ROT2LIN.ncs	proportional
Z_DYN	ROT2LIN1.ncs	proportional
Z_DYN	ROT2TRANS.ncs	proportional
Z_DYN	conf_block.ncs	-
-	DEMUX.ncs	demux
-	GROUND.ncs	constant
-	X_SCOPE.ncs	scope
-	Z_SCOPE.ncs	scope
-	conf_block.ncs	-
-	FlexSensor.oic	-
-	FlexActuator.sbc	-
-	conf_con.con	-
-	conf_par.par	-
-	conf_sim.sbc	-

Cartella VIRTUALPLANT_UDP		
Sottocartella	**File**	**Tipo**
X_DYN/REIBUNG/REG	C1.ncs	constant
X_DYN/REIBUNG/REG	C2.ncs	constant
X_DYN/REIBUNG/REG	C3.ncs	constant
X_DYN/REIBUNG/REG	C5.ncs	constant
X_DYN/REIBUNG/REG	C6.ncs	constant
X_DYN/REIBUNG/REG	C7.ncs	constant
X_DYN/REIBUNG/REG	D1.ncsx	division
X_DYN/REIBUNG/REG	F1.ncs	product
X_DYN/REIBUNG/REG	F2.ncsx	scalar_function
X_DYN/REIBUNG/REG	F3.ncsx	scalar_function
X_DYN/REIBUNG/REG	G1.ncs	proportional
X_DYN/REIBUNG/REG	G2.ncs	proportional
X_DYN/REIBUNG/REG	P1.ncs	product
X_DYN/REIBUNG/REG	P2.ncs	product
X_DYN/REIBUNG/REG	P3.ncs	product
X_DYN/REIBUNG/REG	P4.ncs	product
X_DYN/REIBUNG/REG	P5.ncs	product
X_DYN/REIBUNG/REG	S1.ncs	sum
X_DYN/REIBUNG/REG	S2.ncs	sum
X_DYN/REIBUNG/REG	conf_block.ncs	-
X_DYN/REIBUNG	SMOOTHD.ncs	discrete_pid
X_DYN/REIBUNG	conf_block.ncs	-
X_DYN	DEMUX.ncs	demux
X_DYN	GROUNG.ncs	constant
X_DYN	MECHANIK.ncsx	ABCD_MTL4
X_DYN	MEFF.ncs	sum
X_DYN	MEFF1.ncs	sum

Continua nella pagina successiva

Cartella VIRTUALPLANT_UDP		
Sottocartella	**File**	**Tipo**
X_DYN	MU_S.ncs	proportional
X_DYN	MUX.ncs	mux
X_DYN	ROT2LIN.ncs	proportional
X_DYN	ROT2LIN1.ncs	proportional
X_DYN	ROT2TRANS.ncs	proportional
X_DYN	VDMS.ncs	sum
X_DYN	XDMS.ncs	sum
X_DYN	conf_block.ncs	-
Z_DYN/REIBUNG/REG	C1.ncs	constant
Z_DYN/REIBUNG/REG	C2.ncs	constant
Z_DYN/REIBUNG/REG	C3.ncs	constant
Z_DYN/REIBUNG/REG	C5.ncs	constant
Z_DYN/REIBUNG/REG	C6.ncs	constant
Z_DYN/REIBUNG/REG	C7.ncs	constant
Z_DYN/REIBUNG/REG	D1.ncsx	division
Z_DYN/REIBUNG/REG	F1.ncs	product
Z_DYN/REIBUNG/REG	F2.ncsx	scalar_function
Z_DYN/REIBUNG/REG	F3.ncsx	scalar_function
Z_DYN/REIBUNG/REG	G1.ncs	proportional
Z_DYN/REIBUNG/REG	G2.ncs	proportional
Z_DYN/REIBUNG/REG	P1.ncs	product
Z_DYN/REIBUNG/REG	P2.ncs	product
Z_DYN/REIBUNG/REG	P3.ncs	product
Z_DYN/REIBUNG/REG	P4.ncs	product
Z_DYN/REIBUNG/REG	P5.ncs	product
Z_DYN/REIBUNG/REG	S1.ncs	sum
Z_DYN/REIBUNG/REG	S2.ncs	sum
Z_DYN/REIBUNG/REG	conf_block.ncs	-

Continua nella pagina successiva

Cartella VIRTUALPLANT_UDP		
Sottocartella	**File**	**Tipo**
Z_DYN_REIBUNG	SMOOTHD.ncs	discrete_pid
Z_DYN_REIBUNG	conf_block.ncs	-
Z_DYN	ADD.ncs	sum
Z_DYN	DEMUX.ncs	demux
Z_DYN	GROUNG.ncs	constant
Z_DYN	MECHANIK.ncsx	ABCD_MTL4
Z_DYN	MEFF1.ncs	sum
Z_DYN	MU_S.ncs	proportional
Z_DYN	MUX.ncs	mux
Z_DYN	ROT2LIN.ncs	proportional
Z_DYN	ROT2LIN1.ncs	proportional
Z_DYN	ROT2TRANS.ncs	proportional
Z_DYN	conf_block.ncs	-
-	DEMUX.ncs	demux
-	GROUND.ncs	constant
-	X_SCOPE.ncs	scope
-	Z_SCOPE.ncs	scope
-	conf_block.ncs	-
-	FlexSensor.oic	-
-	FlexActuator.sbc	-
-	conf_con.con	-
-	conf_con_ext.con	-
-	conf_par.par	-
-	conf_sim.sbc	-

Per una completa analisi dei contenuti di tali file si faccia riferimento all'appendice B.

7. Il test di controllo effettuato sul modello sopra descritto è stato re-

alizzato attraverso l'interazione dei moduli VirtualPlant e Controller con il seguente set di parametri:

- Orizzonte di controllo: 60ms
- Orizzonte sensori: 5ms
- MS_PER_TICK: 2
- Ritardo di trasmissione su rete: 10ms con distribuzione normale e varianza di 2ms[11]

ed i tracciati di posizione e velocità desiderati son riportati nelle figure 3.23 - 3.26.

Figura 3.23: Andamento temporale posizione asse X

Il modulo Controller è stato fatto evolvere sia con il modello nominale sia con un modello alterato (come indicato nella seguente tabella) nella definizione dei parametri relativi alla "Frizione statica" di

[11]Tale ritardo è stato introdotto attraverso il comando "tc" utilizzato su piattaforma Unix per la gestione del traffico dati su rete; questo particolare impostazione è stata effettuata attraverso le seguenti stringhe di comando:

```
sudo tc qdisc add dev eth0 root netem delay 10ms 2ms distribution normal
sudo tc qdisc add dev eth0 parent 1:1 pfifo limit 1000
```

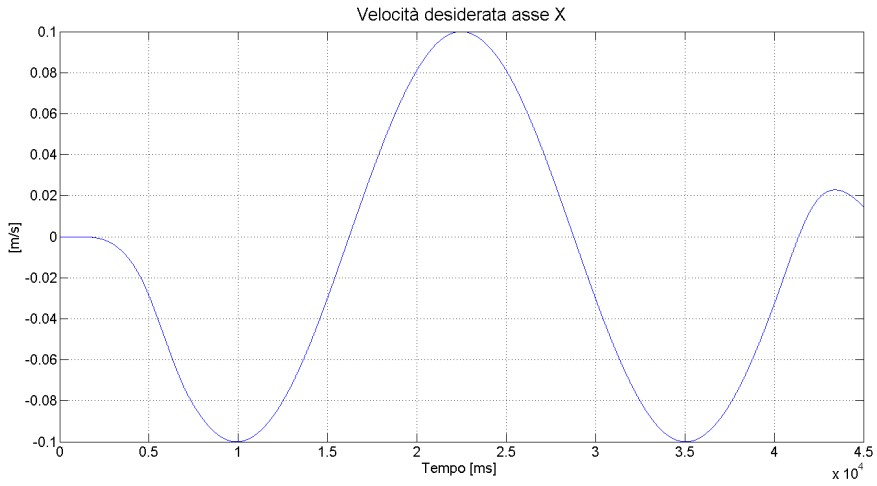

Figura 3.24: Andamento temporale velocità asse X

Figura 3.25: Andamento temporale posizione asse Z

entrambi gli assi ed i risultati ottenuti sono rappresentati nelle seguenti figure 3.27-3.28 (nella quale sono stati tracciati sia la traiettoria eseguita sia l'andamento dell'errore radiale aumentato di un fattore 1000).

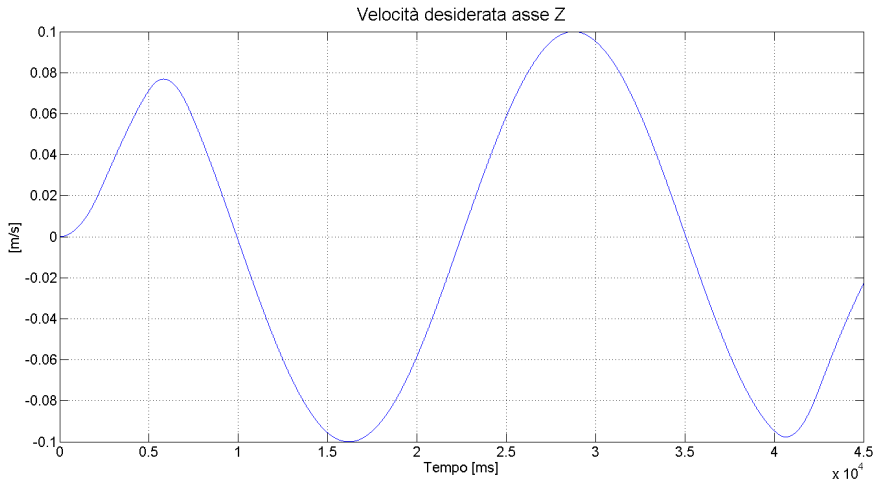

Figura 3.26: Andamento temporale velocità asse Z

Modifiche parametriche			
Sottocartella	**File**	**Nominale**	**Modificato**
X_DYN/REIBUNG/REG	C1.ncs	7.17057	15
X_DYN/REIBUNG/REG	C2.ncs	1.49378	5
X_DYN/REIBUNG/REG	C3.ncs	0.00980583	1
X_DYN/REIBUNG/REG	C5.ncs	25	5
X_DYN/REIBUNG/REG	C6.ncs	0.1	0.5
X_DYN/REIBUNG/REG	C7.ncs	1.7107	2.5
Z_DYN/REIBUNG/REG	C1.ncs	6.73134	3
Z_DYN/REIBUNG/REG	C2.ncs	-0.322606	1
Z_DYN/REIBUNG/REG	C3.ncs	0.0206522	-0.05
Z_DYN/REIBUNG/REG	C5.ncs	25	1
Z_DYN/REIBUNG/REG	C6.ncs	0.1	1
Z_DYN/REIBUNG/REG	C7.ncs	2.0304	4

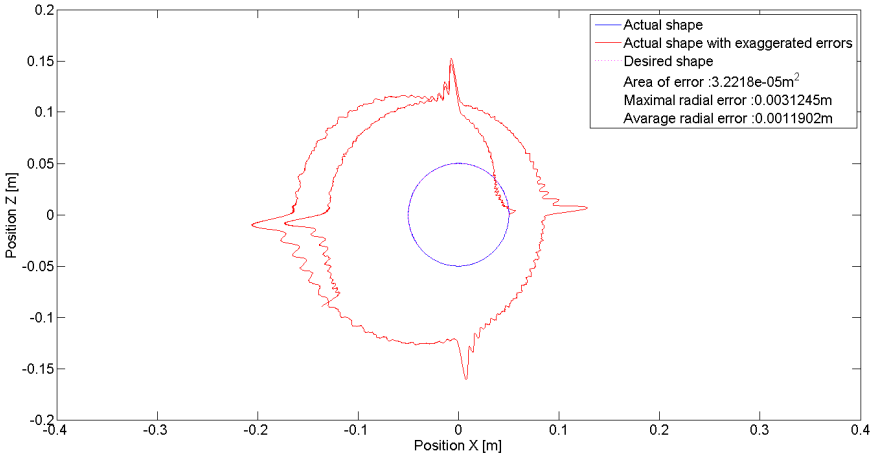

Figura 3.27: Modello nominale

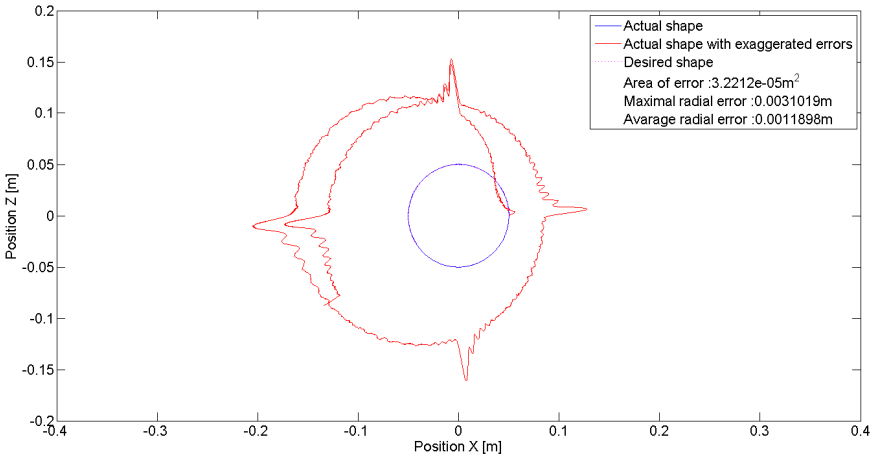

Figura 3.28: Modello con variazioni parametriche

8 Dato che questo esempio è associato ad un modello Simulink®, vengono di seguito indicati i passi necessari per la sua interazione con il modulo CONTROLLER:

- Editare il file conf_con_ext.con contenuto nella cartella VIRTUAL PLANT_UDP; in conformità con quanto indicato nell'appendice B,

e con riferimento alla figura 2.3, supponiamo di eseguire l'ambiente Simulink sul medesimo elaboratore del modulo Virtualplant utilizzando i seguenti indirizzamenti di rete:

- IP simulatore esterno: 127.0.0.1
- VirtualPlant - Porta TX: 3496
- VirtualPlant - Porta RX: 3497
- Simulink - Porta TX: 3494
- Simulink - Porta RX: 3495

- Elaborare il modello fornito sostituendo i sottosistemi di controllo con un blocco UDP receiver ed inviando tutti gli output verso un blocco UDP sender, come mostrato nella seguente figura 3.29.

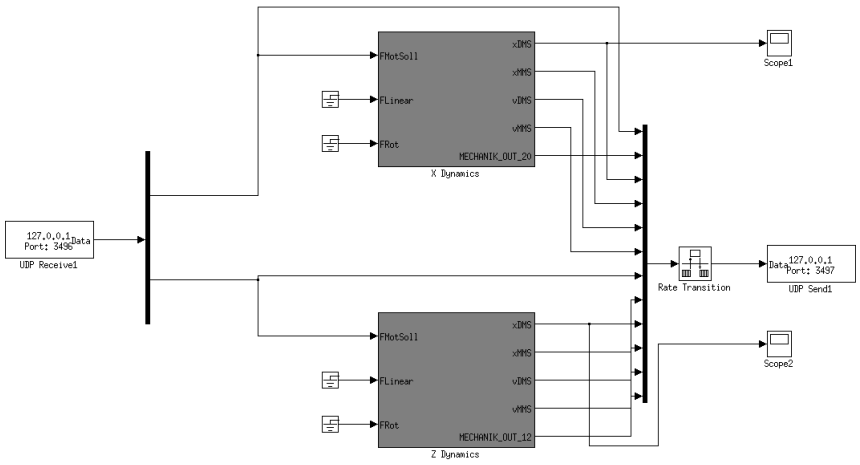

Figura 3.29: Elaborazione modello Simulink®

- Settare in modo opportuno i parametri dei due blocchi UDP receiver e UDP sender (vedi figura 3.30).
- Il test di controllo effettuato sul modello sopra descritto è stato realizzato attraverso l'interazione dei moduli VirtualPlant_UDP e Controller con il seguente set di parametri:
  - Orizzonte di controllo: 60ms

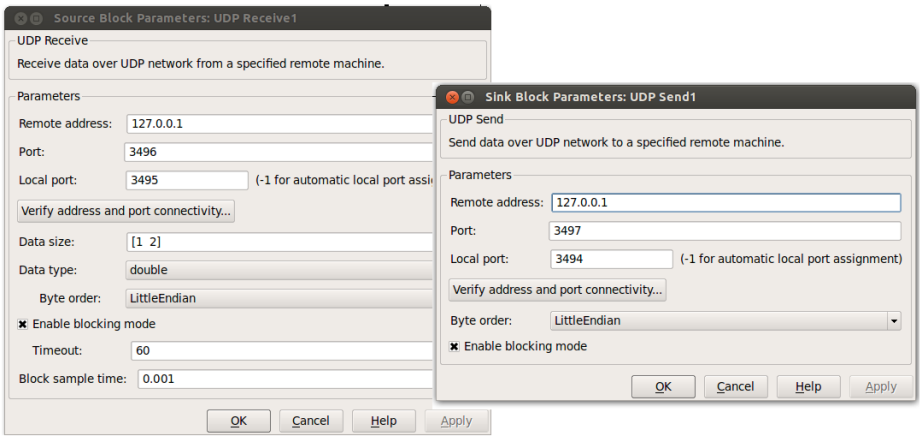

Figura 3.30: Settaggio parametri UDP sender e UDP receiver

- Orizzonte sensori: 5ms
- MS_PER_TICK: 2
- Ritardo di trasmissione su rete: 10ms con distribuzione normale e varianza di 2ms[12]

I risultati ottenuti sono rappresentati nella seguente figura 3.31 (nella quale sono stati tracciati sia la traiettoria eseguita sia l'andamento dell'errore radiale aumentato di un fattore 1000).

Al fine di poter effettuare una comparazione tra le prestazioni ottenute con la tecnica NCS/PBC ed il controllo diretto, è stata effettuato un test di controllo chiudendo direttamente in loop i modelli utilizzati nei moduli Controller e VirtualPlant; nella seguente figura 3.32 sono quindi riportati i risultati ottenuti:

[12]Tale ritardo è stato introdotto attraverso il comando "tc" utilizzato su piattaforma Unix per la gestione del traffico dati su rete; questo particolare impostazione è stata effettuata attraverso le seguenti stringhe di comando:

```
sudo tc qdisc add dev eth0 root netem delay 10ms 2ms distribution normal
sudo tc qdisc add dev eth0 parent 1:1 pfifo limit 1000
```

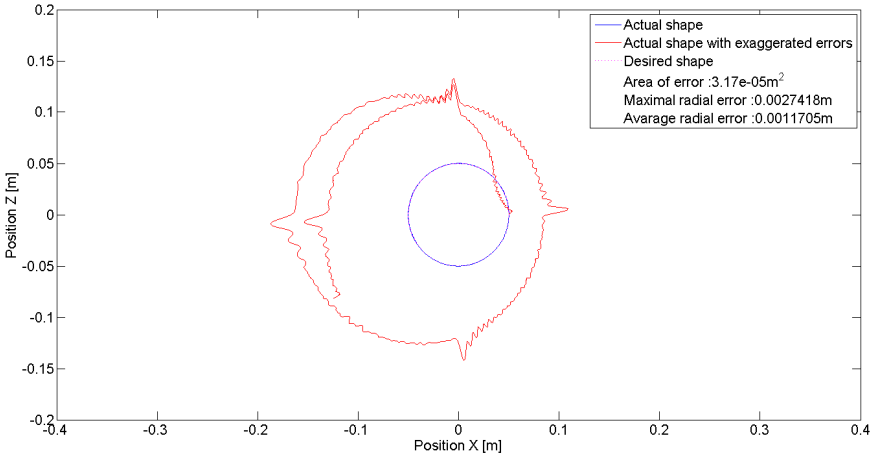

Figura 3.31: Modello esterno in ambiente Simulink

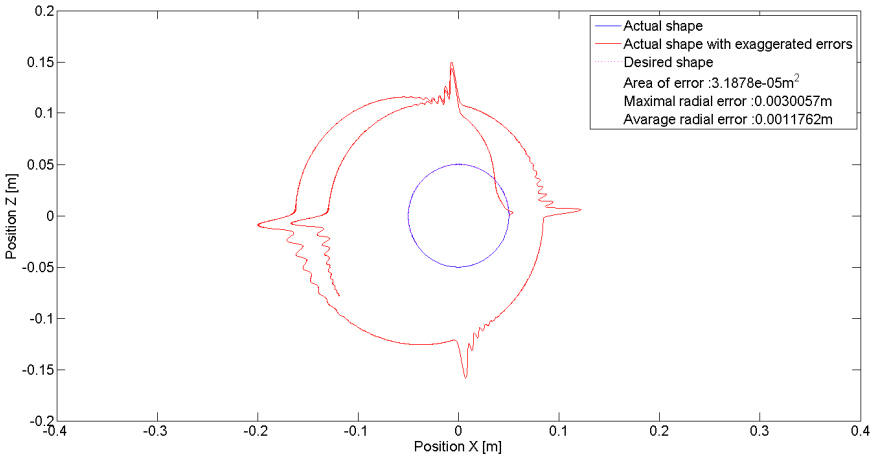

Figura 3.32: Controllo diretto del modello di impianto senza rete

Capitolo 4

Tuning dei parametri funzionali

In questo capitolo verranno date alcune indicazioni circa la taratura dei seguenti parametri:

1. Velocità di simulazione (MS_PER_TICK)
2. Tolleranza agli errori di simulazione (FAILURE_TOL)
3. Lunghezza orizzonte di controllo (HORIZON)
4. Lunghezza orizzonte sensore (N_SENS_HORIZON)
5. Ritardo di trasmissione

I test effettuati sono basati sulla struttura NCS descritta nella precedente sezione 3.3 (Cutting a Circle).

4.1 Velocità di simulazione

Attraverso il parametro MS_PER_TICK, settato nel file *conf_par.par* presente nelle cartelle *VIRTUALPLANT* e *VIRTUALPLANT_UDP*, è possibile variare l'intervallo temporale (in millisecondi) tra ognuno dei passi di simulazione effettuati dal modulo VirtualPlant. Tale adattamento si rende necessario nel caso in cui il pc su cui viene fatto evolvere il modulo Virtualplant non riesca ad eseguire un passo di simulazione entro il tempo di

campionamento fissato per il modello; al crescere di questo parametro si assisterà infatti, come riassunto in 4.1 ad un proporzionale rallentamento della simulazione o, dualmente, sarà possibile assistere ai benefici offerti da un controllore proporzionalmente più veloce.

$$\frac{T_c + R_{sim}}{m} = R_{real} + T_c' \tag{4.1}$$

con

- T_c: tempo di elaborazione del nodo controller per la generazione di un orizzonte di controllo
- R_{sim}: ritardo di rete impostato durante la simulazione
- m: valore impostato per il parametro MS_PER_TICK
- T_c': tempo di elaborazione del nodo controller scelto per l'installazione in impianto
- R_{real}: ritardo di rete equivalente nell'evoluzione a tempo reale

Affinché i risultati ottenuti dalla simulazione siano attendibili è necessario che il valore assegnato a questo parametro verifichi quanto segue:

$$\underline{m} \leq m \tag{4.2}$$

dove

- m è il valore assegnato al parametro MS_PER_TICK
- $\underline{m} = \frac{T_{sim}}{T}$
  - T_{sim} = Tempo di esecuzione di un passo di simulazione [ms]
  - T = Tempo di campionamento del modello [ms]

Nel caso in cui si voglia imporre l'uguaglianza $R_{sim} = R_{real}$ (garantendo quindi una reale tolleranza della legge di controllo all'esatto ritardo di

rete impostato), è necessario modificare la precedente relazione 4.3 come segue:

$$\underline{m} \leq m \leq \overline{m} \tag{4.3}$$

dove il limite superiore $\overline{m}$, definito come

$$\overline{m} = \frac{T_c}{R_{sim}} - 1 \tag{4.4}$$

ha lo scopo di garantire che i risultati ottenuti dalla simulazione siano traducibili con il solo potenziamento del nodo controllore[1].

4.2 Tolleranza agli errori di simulazione

Attraverso il parametro FAILURE_TOL, settato nel file *conf_par.par* presente nelle cartelle *VIRTUALPLANT* e *VIRTUALPLANT_UDP*, è possibile verificare la correttezza del valore MS_PER_TICK precedentemente impostato. L'esecuzione del modulo VirtualPlant comporta la ripetizione ciclica delle seguenti operazioni:

1. Rilevazione dell'istante temporale di avvio del ciclo
2. Esecuzione di un passo di simulazione del modello
3. Verifica di non aver sforato lo slot temporale definito attraverso il parametro MS_PER_TICK

[1] A tal fine deve verificarsi quanto segue (con x fattore di potenziamento del nodo controllore):

$$\frac{T_c + R_{sim}}{m} = R_{sim} + \frac{T_c}{x} \tag{4.5}$$

$$x = \frac{m}{\frac{R_{sim}}{T_c}(1-m)+1} \tag{4.6}$$

$$\frac{R_{sim}}{T_c}(1-m)+1 > 0 \tag{4.7}$$

$$\overline{m} = \frac{T_c}{R_{sim}} - 1 \tag{4.8}$$

4. Se la precedente condizione è verificata allora procede alla ricezione di pacchetti dati provenienti dal nodo controllore, altrimenti salta questo step ed avvia il ciclo successivo

Come si può notare quindi, un settaggio non corretto del parametro MS_PER_TICK potrebbe addirittura imporre al modulo VirtualPlant di non ricevere alcun pacchetto dal mondo esterno invalidando così ogni tipo di attendibilità della simulazione effettuata.
Attraverso il parametro FAILURE_TOL è possibile imporre il numero massimo di cicli per i quali il modulo VirtualPlant può rinunciare alla ricezione di pacchetti dal nodo controllore, imponendo un'immediata interruzione della simulazione nel caso che tale limite venga superato.

4.3 Lunghezza dell'orizzonte di controllo

Attraverso il parametro HORIZON, settato nel file *conf_par.par* presente nelle cartelle *CONTROLLER*, *VIRTUALPLANT* e *VIRTUALPLANT_UDP*, è possibile indicare la lunghezza della sequenza di controllo elaborata da ciascun nodo attuatore presente nella struttura NCS.
Per la verifica della correttezza del valore assegnato a questo parametro è necessario garantire il rispetto della relazione 1.32 (qui riportata per semplicità):

$$T_0^p \geq \tau^m + T^m + \tau^c + T^c. \tag{4.9}$$

con

- T_0^p = lunghezza dell'orizzonte di controllo da elaborare
- $T^m + T^c$ = ritardo massimo nelle comunicazioni di rete
- τ^c = intervallo massimo tra l'invio di due orizzonti di controllo successivi

- τ^m = tempo necessario per la raccolta dei dati necessari alla definizione di tutti i pacchetti sensori; questo valore è definibile come

$$\tau^m = \ell_y(lunghezza\ pacchetti\ sensori) \tag{4.10}$$

$$\ell_y = numero\ nodi\ sensori \tag{4.11}$$

dove tutte le misure temporali devono essere espresse rispetto al tempo di campionamento previsto per la simulazione del modello di impianto.
È necessario ricordare che, dato il forte legame tra il tempo di elaborazione dell'orizzonte (T_c)[2] ed il parametro MS_PER_TICK, è necessario considerare quanto segue:

- Se il valore di MS_PER_TICK può essere impostato ad 1, allora sarà possibile modificare la lunghezza dell'orizzonte di controllo senza ulteriori accorgimenti
- Se il valore di MS_PER_TICK deve essere superiore ad 1 (a causa di una capacitá di calcolo non sufficiente del modulo VirtualPlant), allora sarà possibile variare la lunghezza dell'orizzonte di controllo solamente dopo aver verificato che il T_c associato al suo nuovo valore soddisfi i requisiti previsti nella precedente sezione 4.1

4.4 Lunghezza dell'orizzonte dei sensori

Attraverso il parametro N_SENS_HORIZON, settato nel file *conf_par.par* presente nelle cartelle *VIRTUALPLANT*, *VIRTUALPLANT_UDP* e *CONTROLLER*, è possibile indicare la lunghezza della sequenza di output inviata da ciascun nodo sensore presente nella struttura NCS. Per questo parametro non è possibile indicare a priori un valore ottimale; la lunghezza degli orizzonti dei nodi sensori rappresenta infatti un trade-off da valutare di volta in volta secondo quanto segue:

[2]Variabile con il variare della lunghezza dell'orizzonte di controllo

- Larghezza di banda disponibile nelle comunicazioni di rete
- Necessità della legge di controllo (elaborata dal nodo controllore) di feedback più o meno frequenti dall'impianto controllato

Diversamente da quanto illustrato per la scelta della lunghezza dell'orizzonte di controllo, la variazione di questo parametro può essere effettuata senza ulteriori verifiche (data la sua estraneità con il parametro MS_PER_TICK). A titolo di esempio vengono riportati i risultati ottenuti nell'esempio "Cutting a Cirlce" (analizzato nel precedente capitolo 3.3) utilizzando come indicatori di prestazione i seguenti valori numerici:

- Errore radiale massimo (ERM): indica il massimo errore radiale raggiunto durante l'intera fase di taglio
- Errore radiale medio (mER): indica la media degli errori radiali rilevati lungo la fase di taglio

e riportando per ciascuna soluzione le dimensioni (in Byte) dei vari pacchetti sensori generati (di tipo GenSens_x).

Orizzonte	ERM [m]	mER [m]	2_32	13_32	21_32
5	0.0031245	0.0011902	130	570	890
10	0.0032871	0.0012090	210	1090	1696
20	0.0039599	0.0013026	370	2096	3376
40	0.0073255	0.0016399	690	4176	6736

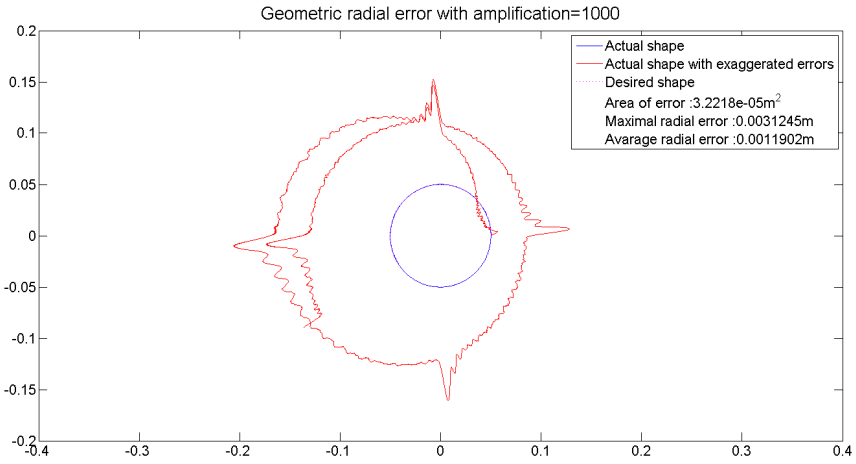

Figura 4.1: Orizzonte sensore = 5

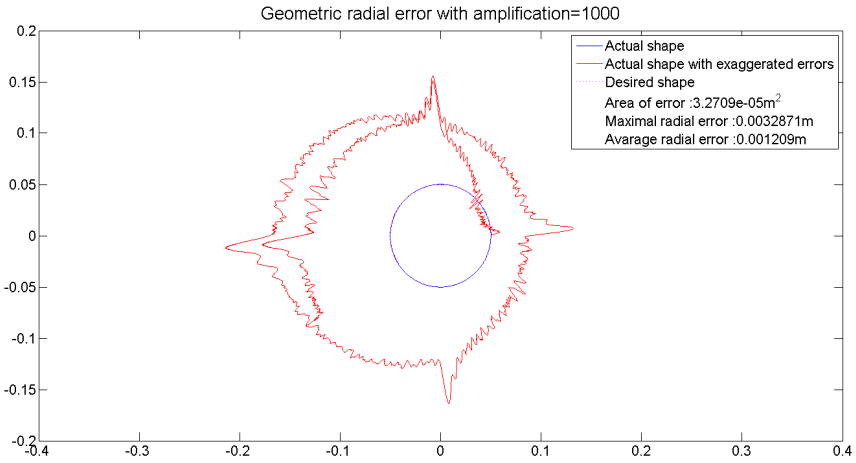

Figura 4.2: Orizzonte sensore = 10

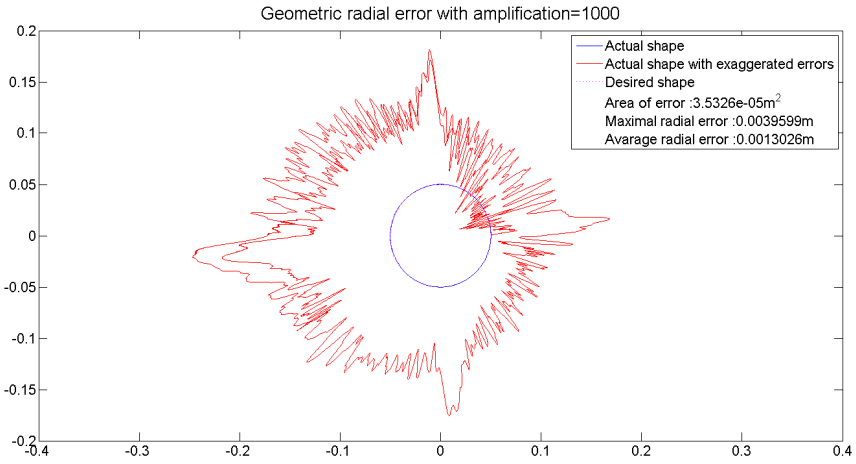

Figura 4.3: Orizzonte sensore = 20

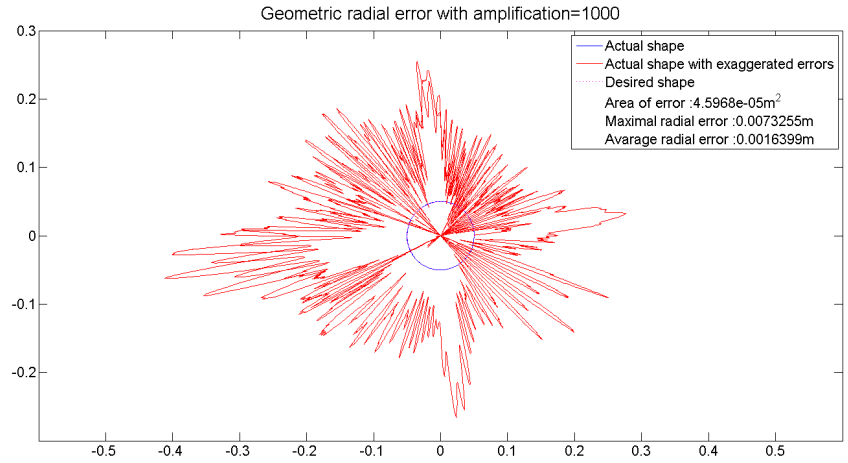

Figura 4.4: Orizzonte sensore = 40

4.5 Ritardo di trasmissione

Questo parametro serve per simulare il ritardo di trasmissione presente sulla linea di comunicazione tra i nodi ed il controllore ed attraverso la sua variazione è possibile analizzare il comportamento del sistema NCS elaborato. In fase di simulazione viene settato attraverso il comando "tc", utilizzato su piattaforma Unix per la gestione del traffico dati su rete, con le due seguenti stringhe di comando:

- tc qdisc add dev γ root netem delay αms βms distribution normal
- tc qdisc add dev γ parent 1:1 pfifo limit 1000

Attraverso la prima stringa è possibile aggiungere sul dispositivo di comunicazione γ (normalmente una scheda di rete) un ritardo con distribuzione normale, media α e varianza β; con la seconda si impone invece che il suddetto ritardo non alteri la sequenza dei pacchetti inviati.
Come per la taratura dell'orizzonte di controllo è anche qui necessario ricordare che, dato il forte legame tra il ritardo di rete (R) ed il parametro MS_PER_TICK, è necessario considerare quanto segue:

- Se il valore di MS_PER_TICK può essere impostato ad 1, allora sarà possibile modificare il ritardo di rete senza ulteriori accorgimenti
- Se il valore di MS_PER_TICK deve essere superiore ad 1 (a causa di una capacitá di calcolo non sufficiente del modulo VirtualPlant), allora sarà possibile variare il ritardo di rete solamente dopo aver verificato che il nuovo valore soddisfi i requisiti previsti nella precedente sezione 4.1

Capitolo 5

Conclusioni

A fronte dei risultati ottenuti, è possibile concludere che il software elaborato costituisce un attendibile supporto sia alla fase di sintesi della tecnica di controllo, sia durante l'analisi delle performance con essa raggiunte; tra i suoi maggiori vantaggi possono essere menzionati: facilità di configurazione attraverso l'editazione di file, estendibilità delle librerie (con blocchi customizzati realizzabili dall'utente), totale capacità di interazione con modelli sviluppati in ambienti di simulazione esterni. Durante i test effettuati sono inoltre emersi alcuni spunti utili per successivi sviluppi; in particolare è stata rilevata l'utilità di una interfaccia grafica per la compilazione dei file testuali, dell'introduzione di modelli con struttura parametrizzata (e periodicamente aggiornabile) e del possibile interfacciamento con routine di ambienti software esterni capaci di tradurre i propri modelli in codice C/C++ (come ad esempio l'ambiente Matlab® Coder).

Appendice A

Librerie

A.1 Blocchi funzionali

Nella seguente lista sono elencati i blocchi funzionali predefiniti durante l'elaborazione del software; in caso di necessità l'utente potrà comunque definirne di nuovi seguendo la procedura indicata al termine di questa sezione.

Nome	Tipo	#In	#Out	Funzione
abcd	ncs	1 (muxed)	1 (muxed)	Forma di stato ABCD
abcd_MTL4	ncsx	1 (muxed)	1 (muxed)	Forma di stato ABCD
constant	ncs	1	1	Costante
delay	ncs	1	1	Elemento di ritardo
demux	ncs	1 (muxed)	n	Demodulatore
discrete_pid	ncs	1	1	Pid discreto
division	ncsx	n	1	Divisione
integrator	ncs	1	1	Integratore
integrator	ncsx	1	1	Integratore
mux	ncs	n	1 (muxed)	Modulatore
product	ncs	n	1	Prodotto

Tabella A.1: Blocchi funzionali predefiniti

Nome	Tipo	#In	#Out	Funzione
proportional	ncs	n	n	Guadagno
scalar_fun	ncsx	n	n	Funzione scalare
scope	ncs	1	1	Visualizzatore
source	ncs	1	1	Riferimento
source	ncsx	1	1	Riferimento
sum	ncs	n	1	Sommatore
switch	ncs	n	1	Switch

Tabella A.1: Blocchi funzionali predefiniti

Gli elementi di tipo "ncs" possono essere editati solamente in modalità testuale, mentre quelli di tipo "ncsx" sono configurabili attraverso appositi file con struttura ".xml".
Per l'editazione di tali file si faccia riferimento alla seguente Appendice B.

A.1.1 Come aggiungere nuovi blocchi

In caso di necessità il suddetto elenco può essere ampliato attraverso la definizione di nuovi blocchi funzionali customizzati; tale procedura si basa sulle tre seguenti modifiche della libreria **networkatom**:

1. Creazione del file contenente le definizioni del nuvo blocco
   Per prima cosa è necessario ricordare che tutti i file sorgente dedicati alla definizione di blocchi devono essere raggruppati nella medesima cartella del sistema; si consideri tale cartella raggiungibile al path *../networkatom/blockLibrary* (dove .. indica il path della cartella nella quale sono stati copiati i file sorgente).
   La struttura di ciascun file di definizione deve comprendere almeno le seguenti quattro funzioni:

- *vpData outputFunction(...)*
  Attraverso questa funzione il sistema, ad ogni passo di elaborazione, estrapola dal blocco gli opportuni output e li fornisce ai vari moduli utilizzatori; mentre i parametri da fornire in input sono customizzabili in base alle esigenze, l'output deve obbligatoriamente essere di tipo vpData.
- *bool isFeedThrough()*
  In caso di output positivo il sistema riconosce a tale blocco una diretta dipendenza tra output rilevati ed input settati nel medesimo istante; tale funzione, particolarmente rilevante in caso di loop algebrici, non ha parametri in ingresso e deve obbligatoriamente fornire un output di tipo booleano.
- *pDynamic factory(...)*
  Attraverso questa funzione il sistema genera la struttura elementare utilizzata dal sistema per l'elaborazione di ogni passo di simulazione; mentre i parametri da fornire in input sono customizzabili in base alle esigenze, l'output deve obbligatoriamente essere di tipo pDynamic.
- una o più funzioni per la lettura dei file ".ncs" editati dall'utente

2. Registrazione del nuvo blocco all'interno degli archivi della libreria
   Per effettuare questa operazione è necessario modificare tre file, contenuti nella cartella *../networkatom* (dove .. indica il path della cartella nella quale sono stati copiati i file sorgente)

   - *blockLibrary.hpp*
     Questo file rappresenta l'indice dei blocchi disponibili e contiene tutte le indicazione relative alle funzioni definite al precedente passo
   - *FFFManager.hpp*
     Questo file permette la configurazione del sottomodulo soft-

ware dedicato alla lettura dei file editati dall'utente durante la definizione dei modelli

- *Makefile.am*
  Questo fie è di fondamentale importanza per la successiva ricompilazione della libreria, dato che raccoglie tutti i file da elaborare in tale fase

3. Ricompilazione della libreria
   Attraverso quest'ultima operazione il sistema analizza le modifiche fatte, compila i nuovi blocchi creati ed aggiorna la libreria del sistema; per la sua esecuzione è sufficeinte digitare i seguenti comandi

   - *../configure*
   - *../make*
   - *sudo ../make install*

   ed attendere il termine dell'elaborazione.

Si riportano, a titolo di esempio, i file necessari per la definizione dei blocchi utilizzati nella precedente sezione 3.3.3.

A.2 Nodi attuatori

I nodi attuatori sono fisicamente costituiti da schede hardware collegate sia all'impianto reale che alla rete NCS ed il loro compito è quello di trasformare gli orizzonti di controllo, inviati dal nodo controllore, in segnali elettrici da applicare all'impianto reale. Per ogni nodo hardware risultano quindi necessari due tipi di driver software:

- Uno capace di organizzare i pacchetti generati dal modulo Controller secondo la struttura dati prevista dal firmware del nodo hardware
- Uno capace di emulare la funzione del nodo reale all'interno del modulo Virtualplant (utilizzato per simulare l'impianto)

Nella seguente lista sono elencati i driver predefiniti durante l'elaborazione del software; in caso di necessità l'utente potrà comunque definirne di nuovi seguendo la procedura indicata al termine di questa sezione.

Nome	Applicazione	#Canali	Varianti	Note
GenAct1_32	Controller	1	-	-
FlexGenAct1_32	Virtualplant	1	-	-
GenAct2_32	Controller	2	-	-
FlexGenAct2_32	Virtualplant	2	-	-

Tabella A.2: Driver definiti per nodi attuatore

Il campo "Varietà" assume il seguente significato:

- (-) all'interno del medesimo impianto può esistere un solo elemento di questo tipo
- (x,y,z) all'interno del medesimo impianto possono coesistere più elementi di questo tipo (rispettivamente identificati dal carattere x,y e z)

A.2.1 Come aggiungere nuovi nodi attuatori

In caso di necessità il suddetto elenco può essere ampliato attraverso la definizione di nuovi nodi attuatori customizzati; tale procedura si basa sulle tre seguenti modifiche della libreria **ncs/Actuators**:

1. Creazione del file contenente le definizioni del nuvo nodo
   Per prima cosa è necessario ricordare che tutti i file sorgente dedicati alla definizione di blocchi devono essere raggruppati nella medesima cartella del sistema; si consideri tale cartella raggiungibile al path *../ncs/Actuators* (dove *..* indica il path della cartella nella quale sono stati copiati i file sorgente).
   La struttura di ciascun driver dedicato al modulo Controller deve comprendere almeno le seguenti funzioni:

   - *static *** factory(...)*
     Funzione statica attraverso la quale il sistema provvede alla creazione del pacchetto da inviare
   - *pSerial alloc() constant*
     Funzione utilizzata per la gestione della memoria dinamica del pacchetto dati
   - *void set(pcSerial)*
     Funzione utilizzata per copiare nel pacchetto gestito un dato pcSerial, generato dal modulo Controller, secondo la struttura dati prevista dal firmware del nodo hardware
   - *void set(char*)*
     Funzione utilizzata per copiare nel pacchetto gestito una stringa di caratteri, generata dal modulo Controller, secondo la struttura dati prevista dal firmware del nodo hardware
   - *char* get() const*
     Funzione utilizzata per estrapolare tutte le informazioni contenute nel pacchetto dati gestito

- *char id() const*
  Funzione utilizzata per estrapolare il carattere identificatore del pacchetto dati gestito; tale carattere indica il tipo di nodo attuatore al quale inviare il pacchetto ricevuto, è pertanto necessario che nel medesimo impianto non coesistano più di due nodi attuatori con il medesimo id
- *timer::type getTimestamp() const*
  Funzione utilizzata per estrapolare l'istante di generazione del pacchetto dati gestito
- *void setTimestamp(timer::type) const*
  Funzione utilizzata per impostare l'istante di generazione del pacchetto dati gestito
- *unsigned int getPacketSize() const*
  Funzione utilizzata per rilevare la lunghezza (in byte) del pacchetto dati gestito
- *void setCmd(int,pData)*
  Funzione utilizzata per l'inserimento dell'orizzonte di controllo all'interno del pacchetto dati gestito
- *pData getCmd(timer::type) const*
  Funzione utilizzata per l'estrpolazione, dall'orizzonte di controllo presente nel pacchetto dati gestito, del segnale di comando associato ad un preciso istante temporale
- *void print() const*
  Funzione utilizzata per la stampa a video dell'orizzonte di controllo contenuto nel pacchetto gestito
- *vpData outputSelector(vpData,PVector*
  Funzione utilizzata per la selezione degli output del modulo Controller da inviare al nodo attuatore destinatario del pacchetto gestito

- *pPacket packetCreator(std::map¡timer::type, vpData¿)*
  Funzione utilizzata dal modulo Simulator per la creazione e compilazione del pacchetto dati
- *Simulator::Initializer getInizializerFun(std::string)*
  Funzione utilizzata per la configurazione del nodo attuatore attraverso i file ".sbc" editati dall'utente
- $\sim$***
  Funzione utilizzata per la deallocazione della memoria dinamica occupata dal pacchetto gestito

La struttura di ciascun driver dedicato al modulo Virtualplant deve comprendere almeno le seguenti funzioni:

- *static *** factory(...)*
  Funzione statica attraverso la quale il sistema provvede alla creazione del pacchetto da inviare
- *pData inputGenerator(timer::type)*
  Funzione utilizzata per estrarre dal pacchetto dati ricevuto il segnale di controllo relativo ad un preciso istante temporale
- *void notifyPacket(pPacket)*
  Funzione utilizzata per copiare il contenuto del pacchetto dati ricevuto all'interno della struttura dati (di tipo pPacket) indicata nel parametro

2. Registrazione del nuvo nodo all'interno degli archivi della libreria
   Per effettuare questa operazione è necessario modificare due file, entrambi contenuti nella cartella *../ncs/Actuators* (dove .. indica il path della cartella nella quale sono stati copiati i file sorgente)

   - *Actuator_Manager.hpp*
     Questo file rappresenta l'indice dei nodi attuatori disponibili e contiene alcune indicazione relative alle funzioni definite al precedente passo

- *Makefile.am*
  Questo fie è di fondamentale importanza per la successiva ricompilazione della libreria, dato che raccoglie tutti i file da elaborare in tale fase

3. Ricompilazione della libreria
   Attraverso quest'ultima operazione il sistema analizza le modifiche fatte, compila i nuovi nodi creati ed aggiorna la libreria del sistema; per la sua esecuzione è sufficeinte digitare i seguenti comandi

   - *../configure*
   - *../make*
   - *sudo ../make install*

   ed attendere il termine dell'elaborazione.

A.3 Nodi sensori

I nodi sensori sono fisicamente costituiti da schede hardware collegate sia all'impianto reale che alla rete NCS ed il loro compito è quello di trasformare i segnali rilevati sul sistema, in pacchetti dati da inviare al nodo controllore attraverso la rete. Per ogni nodo hardware risultano quindi necessari due tipi di driver software:

- Uno capace di estrapolare i dati contenuti nei pacchetti ricevuti, da utilizzare per la gestione degli ingressi del blocco Local Dynamics
- Uno capace di emulare la funzione del nodo reale all'interno del modulo Virtualplant (utilizzato per simulare l'impianto)

Nella seguente lista sono elencati i driver predefiniti durante l'elaborazione del software; in caso di necessità l'utente potrà comunque definirne di nuovi seguendo la procedura indicata al termine di questa sezione.

Nome	Applicazione	#Canali	Varianti	Note
GenSens2_32	Controller	2	a-b-c-d	-
FlexGenSens2_32	Virtual	2	a-b-c-d	-
GenSens4_32	Controller	4	a-b-c-d	-
FlexGenSens4_32	Virtual	4	a-b-c-d	-
GenSens13_32	Controller	13	a-b-c-d	-
FlexGenSens13_32	Virtual	13	a-b-c-d	-
GenSens20_32	Controller	20	a-b-c-d	-
FlexGenSens20_32	Virtual	20	a-b-c-d	-
GenSens21_32	Controller	21	a-b-c-d	-
FlexGenSens21_32	Virtual	21	a-b-c-d	-

Tabella A.3: Driver definiti per nodi attuatore

Il campo "Varietà" assume il seguente significato:

- (-) all'interno del medesimo impianto può esistere un solo elemento di questo tipo
- (x,y,z) all'interno del medesimo impianto possono coesistere più elementi di questo tipo (rispettivamente identificati dal carattere x,y e z)

A.3.1 Come aggiungere nuovi nodi sensori

In caso di necessità il suddetto elenco può essere ampliato attraverso la definizione di nuovi nodi sensori customizzati; tale procedura si basa sulle tre seguenti modifiche della libreria **ncs/Sensors**:

1. Creazione del file contenente le definizioni del nuvo nodo
   Per prima cosa è necessario ricordare che tutti i file sorgente dedicati alla definizione di blocchi devono essere raggruppati nella medesima cartella del sistema; si consideri tale cartella raggiungibile al path *../ncs/Sensors* (dove .. indica il path della cartella nella quale sono stati copiati i file sorgente).
   La struttura di ciascun driver dedicato al modulo Controller deve comprendere almeno le seguenti funzioni:
   - *static *** factory(...)*
     Funzione statica attraverso la quale il sistema provvede alla creazione della struttura necessaria all'utilizzo del pacchetto ricevuto
   - *pSerial alloc() constant*
     Funzione utilizzata per la gestione della memoria dinamica del pacchetto dati
   - *void set(pcSerial)*
     Funzione utilizzata per copiare il contenuto del pacchetto ricevuto nella struttura pcSerial fornita come parametro

- *void set(char*)*
  Funzione utilizzata per copiare il contenuto del pacchetto ricevuto all'interno di una stringa di caratteri fornita come parametro
- *char* get() const*
  Funzione utilizzata per estrapolare tutte le informazioni contenute nel pacchetto dati ricevuto
- *pData get(timer::type) const*
  Funzione utilizzata per estrapolare dal pacchetto ricevuto l'informazione associata ad un preciso istante temporale
- *char id() const* e *char getId() const*
  Funzioni utilizzata per estrapolare il carattere identificatore del pacchetto dati ricevuto; tale carattere rappresenta il tipo di nodo sensore dal quale proviene il pacchetto ricevuto, è pertanto necessario che nel medesimo impianto non coesistano più di due nodi sensori con il medesimo id (a patto che per essi non sia stata definita una varietà superiore a 1)
- *timer::type getTimestamp() const*
  Funzione utilizzata per estrapolare l'istante di generazione del pacchetto dati ricevuto
- *void setTimestamp(timer::type) const*
  Funzione utilizzata per impostare l'istante di generazione del pacchetto dati ricevuto
- *unsigned int getPacketSize() const*
  Funzione utilizzata per rilevare la lunghezza (in byte) del pacchetto dati ricevuto
- *char getSensorId() const*
  Funzione utilizzata per rilevare, in caso di varietà multipla del nodo sensore utilizzato, l'istanza che ha generato il pacchetto ricevuto

- *char setSensorId(char)*
  Funzione utilizzata per impostare in caso di varietà multipla del nodo sensore utilizzato, l'istanza che ha generato il pacchetto ricevuto
- *void print() const*
  Funzione utilizzata per la stampa a video dell'orizzonte di output contenuto nel pacchetto gestito
- *Data::dataType getCh_n() const*
  Funzione utilizzata per copiare, all'interno di una struttura Data::dataType, l'intero orizzonte di output contenuto nel pacchetto gestito ed associato all'n-esimo canale
- *Data::dataType getCh_n(timer::type) const*
  Funzione utilizzata per copiare, all'interno di una struttura Data::dataType, l'intero orizzonte di output contenuto nel pacchetto gestito, associato all'n-esimo canale e riferito ad un preciso istante temporale
- *void setCh_n(timer::type, Data::dataType)*
  Funzione utilizzata per settare, all'interno dell'orizzonte di output associato all'n-esimo canale, il valore riferito ad un preciso istante temporale
- $\sim$***
  Funzione utilizzata per la deallocazione della memoria dinamica occupata dal pacchetto gestito

La struttura di ciascun driver dedicato al modulo Virtualplant deve comprendere almeno le seguenti funzioni:

- *static *** factory(...)*
  Funzione statica attraverso la quale il sistema provvede alla creazione della struttura necessaria all'utilizzo del pacchetto ricevuto

- *void outputNotifier(vpData)*
  Funzione utilizzata per creare, attraverso la copia di una struttura vpData elaborata dal Virtualplant, il pacchetto sensore da inviare al nodo controllore

2. Registrazione del nuvo nodo all'interno degli archivi della libreria
   Per effettuare questa operazione è necessario modificare due file, entrambi contenuti nella cartella *../ncs/Sensors* (dove *..* indica il path della cartella nella quale sono stati copiati i file sorgente)

   - *Sensor_Manager.hpp*
     Questo file rappresenta l'indice dei nodi attuatori disponibili e contiene alcune indicazione relative alle funzioni definite al precedente passo
   - *Makefile.am*
     Questo fie è di fondamentale importanza per la successiva ricompilazione della libreria, dato che raccoglie tutti i file da elaborare in tale fase

3. Ricompilazione della libreria
   Attraverso quest'ultima operazione il sistema analizza le modifiche fatte, compila i nuovi nodi creati ed aggiorna la libreria del sistema; per la sua esecuzione è sufficeinte digitare i seguenti comandi

   - *../configure*
   - *../make*
   - *sudo ../make install*

   ed attendere il termine dell'elaborazione.

Appendice B

Compilazione file

B.1 File .ncs/.ncsx

In base alla funzione svolta, come illustrato nella sezione 2.1, questi file sono distinguibili in due tipologie:

- Definizione di blocchi
- Definizione delle interconnessioni

B.1.1 Definizione di blocchi

- *abcd (ncs)*

```
abcd
#Matrice A
[1,2,3,4;4,5,6,7;7,8,9,10;10,11,12,13]
#Matrice B
[14;15;16;17]
#Matrice C
[18,19,20,21]
```

```
#Matrice D
[22]
```

- *abcd (ncsx)*

```
<?xml version="1.0"?>
<atom>
  <type>abcd_MTL4</type>
  <specific_parameters>
    <A>[1,2,3,4;4,5,6,7;7,8,9,10;10,11,12,13]</A>
    <B>[14;15;16;17]</B>
    <C>[18,19,20,21]</C>
    <D>[22]</D>
    <discretize>
    →<method>forwardEuler</method>
    →<T>0.001</T>
    </discretize>
  </specific_parameters>
  <multirate>1</multirate>
</atom>
```

Attraverso questo blocco è possibile effettuare la discretizzazione di un sistema abcd indicando il passo di campionamento (T) ed il metodo di discretizzazione (forwardEuler o RungeSecondOrder)

- *constant (ncs)*

```
constant
# numero uscite
3
# valore delle uscite
2
35
3
```

- *delay (ncs)*

```
delay
# tempo di ritardo dell'output rispetto all'input
1
```

- *demux (ncs)*

```
demux
# numero ingressi
3
# elenco dimensioni ingressi
2
1
9
```

- *discrete_pid (ncs)*

```
discrete_pid
# valore di p
5
# valore di i
4
# valore di d
3
# valore di n
2
# valore di T
1
```

- *division (ncsx)*

```
<?xml version="1.0"?>
<atom>
  <type>division</type>
  <specific_parameters>
```

```
5       →  <inputSize>1</inputSize>
6     </specific_parameters>
7   </atom>
```

- *integrator (ncs)*

```
1   integrator
2   #dimensione ingresso (d) e costante temporale (T)
3   1    →0.001
```

  Attraverso questo blocco è possibile definire un integratore attraverso la seguete struttura ABCD

  - A = 1;
  - B = 1;
  - C = T;
  - D = T/2;

- *integrator (ncsx)*

```
1   <?xml version="1.0"?>
2   <atom>
3     <type>integrator</type>
4     <specific_parameters>
5       <integratorType>tustin</integratorType>
6       <inputSize>2</inputSize>
7       <T>0.001</T>
8     </specific_parameters>
9     <multirate>1</multirate>
10  </atom>
```

  Attraverso questo blocco è possibile definire un integratore attraverso due diverse struttura ABCD (in base al valore assunto dal parametro "integratorType"

  - tustin
    - A = 1;
    - B = 1;
    - C = T;
    - D = T/2;
  - forwardEuler
    - A = 1;
    - B = T;
    - C = 1;
    - D = 0;

- *mux (ncs)*

```
mux
# numero ingressi
4
# elenco dimensioni ingressi
1
2
1
1
```

- *product (ncs)*

```
product
# tipo prodotto (s= scalare; m= matriciale)
m
# in caso di prodotto scalare inserire
# il numero degli ingressi
# in caso di prodotto matriciale inserire
# in sequenza altezza input1, larghezza input1,
# altezza input2 e larghezza input2
1
```

```
2
2
3
```

- *proportional (ncs)*

```
proportional
# matrice
[1,2,3;4,5,6;7,8,9;10,11,13]
```

```
proportional
# scalare
[14]
```

- *scalar_fun (ncsx)*

```
<?xml version="1.0"?>
<atom>
  <type>scalar_function</type>
  <specific_parameters>
    <function_name>sin</function_name>
  </specific_parameters>
</atom>
```

La funzione definita in "function_name" deve essere scelta tra quelle presenti nella libreria *math.h*

- *scope (ncs)*

```
scope
#Numero input
3
#Decimazione
1
#Nome file
/home/batch.dat
```

```
#Active (0 = non attivo, 1 = attivo)
1
```

- *source (ncs)*

```
source
# Tipo di segnale
sin
# Ampiezza
3.4
# Bias
2.1
# Periodo
1
# Fase
1
# T
2.4
```

```
source
# Tipo di segnale
pulse
# Ampiezza
3.4
# Periodo
2.1
# Larghezza
1
# Ritardo di fase
1
```

- *source (ncsx)*

```
<?xml version="1.0"?>
<atom>
```

```
<type>source</type>
<specific_parameters>
  <sourceType>refFromFile</sourceType>
  <sourceFile>/home/file.txt</sourceFile>
</specific_parameters>
</atom>
```

- *sum (ncs)*

```
sum
# Dimensione ingressi
1
# Sequenza di addizione algebrica; es. ++ +- -- -+
+-
```

- *switch (ncs)*

```
switch
# Dimensione ingressi/uscite
4
# Soglia di switch
0.345
```

B.1.2 Definizione delle interconnessioni

Per ciascun modello (o sottomodello) questa funzione viene realizzata attraverso l'editazione di un unico file (conf_block.ncs) basato sulla seguente struttura sintattica:

```
# N is the network being built.
# No blocks named N are allowed into the network
#n_inputs
#sizes
```

```
..elenco dimensioni input ..
#outputs
#n_outputs
#sizes
..elenco dimensioni output ..

#initializers for inputs
#the ordering will be reflected in the state ordering
#BLOCK →       →n_in   →BLOCK  →INDEX  →BLOCK  →INDEX
..     →       →..     →..     →..     →..     →..
..     →       →..     →..     →..     →..     →..

#outputs
#BLOCK →       →INDEX
..     →       →..
..     →       →..
```

B.2 File .sbc

Per ciascuna delle tre cartelle di questo software (CONTROLLER, VIRTUALPLANT e VIRTUALPLANT_UDP) devono essere editati i due seguenti file:

- uno per l'inizializzazione degli stati iniziali dei modelli (conf_sim.sbc)

```
# conf_sim.sbc
#State initializer
[s1 s2 s3 s4 .. ]
```

- uno per l'interazione dei modelli con i nodi attuatori (Actuator.sbc nella cartella CONTROLLER e FlexActuator.sbc nelle cartelle VIRTUALPLANT e VIRTUALPLANT_UDP)

```
# Actuator.sbc
Tipo_attuatore (es. GenAct2_32)
# Vettore delle uscite della network
# da collegare, in sequenza, ai canali
# dell'attutore selezionato
[ch_output1 ch_output2]
```

```
# FlexActuator.sbc
Tipo_attuatore (es. GenAct2_32)
```

B.3 File .oic

Per ciascuna delle tre cartelle principali di questo software (CONTROLLER, VIRTUALPLANT e VIRTUALPLANT_UDP) devono essere editati i seguenti file

- CONTROLLER
  - Tanti file ".oic" quanti sono i controllori da collegare ai segnali ricevuti dall'impianto (o dal VIRTUALPLANT)

```
    # NomeControllore.oic
    # {Tipo sensore,Variante,(canali)}
    {GenSens2_32,a,(0-1)}
    # [NOME,N]
    # NOME: nome del blocco i cui primi
    # N stati devono essere aggiornati
    # N: 1= agg. primo stato
    # N: 2= agg. primi due stati
    # N: *= agg. tutti gli stati
    []
```

  - Tanti file ".oic" quanti sono gli osservatori di stato da collegare ai segnali ricevuti dall'impianto (o dal VIRTUALPLANT)

```
# NomeOsservatore.oic
# {Tipo sensore,Variante,(canali)}
{GenSens2_32,a,(0-1)}
# [NOME,N]
# NOME: nome del blocco i cui primi
# N stati devono essere aggiornati
# N: 1= agg. primo stato
# N: 2= agg. primi due stati
# N: *= agg. tutti gli stati
[SISTEMA,*]
```

- VIRTUALPLANT e VIRTUALPLANT_UDP: un solo file (FlexSensor.oic) per la connessione degli output dell'impianto ai canali dei nodi sensori

```
# FlexSensor.oic
# {Tipo sensore 1,Variante,(canali);
# Tipo sensore 2,Variante,(canali);
# ...}
{GenSens2_32,a,(21-24);GenSens2_32,b,(7-8)}
```

B.4 File .con

Per ciascuna delle tre cartelle principali di questo software (CONTROLLER, VIRTUALPLANT e VIRTUALPLANT_UDP) devono essere editati i seguenti file

- CONTROLLER: un solo file (conf_con.con) per il settaggio della porta di comunicazione e dell'indirizzo IP broadcast della sottorete utilizzata per il collegamento della struttura NCS

```
# conf_con.con
# IP broadcast
193.168.0.255
# Porta
3491
```

- VIRTUALPLANT un solo file (conf_con.con) per il settaggio della porta di comunicazione (la medesima indicata nel precedente file) e dell'indirizzo IP del nodo controllore

```
# conf_con.con
# IP nodo controller
193.168.0.46
# Porta
3491
```

- VIRTUALPLANT_UDP due file come segue:
  - conf_con.con per il settaggio della porta di comunicazione (la medesima indicata nel precedente file) e dell'indirizzo IP del nodo controllore

```
# conf_con.con
# IP nodo controller
193.168.0.46
# Porta
3491
```

  - conf_con_ext.con per il settaggio dell'indirizzo IP del simulatore esterno e delle porte di comunicazione tra quest'ultimo e nodo VIRTUALPLANT

```
# conf_con_ext.con
# Indirizzo del simulatore
```

```
3     127.0.0.1
4     # Porta di invio del simulatore
5     3494
6     # Porta di ricezione del simulatore
7     3495
8     # Indirizzo della macchina locale
9     # su cui    in esecuzione il VIRTUALPLANT
10    127.0.0.1
11    # Porta di invio del VIRTUALPLANT
12    # verso il simulatore esterno
13    3496
14    # Porta di ricezione del VIRTUALPLANT
15    # dal simulatore esterno
16    3497
```

B.5 File .par

Per ciascuna delle tre cartelle principali di questo software (CONTROLLER, VIRTUALPLANT e VIRTUALPLANT_UDP) deve essere editata una copia del seguente file (conf_par.par)

```
1     # conf_par.par
2     SIMULATION_TIME    →61548
3     T →        →        →0.0001
4     HORIZON    →        →50
5     N_SENSOR_HORIZON   →10
6     N_STATE    →        →30
7     N_CMD      →        →1
8     MS_PER_TICK→        →10
9     FAILURE_TOL→        →100
```

Qui di seguito sono brevemente riassunte le funzioni di ciascun parametro:

- SIMULATION_TIME: tempo di simulazione del modulo VIRTUALPLANT (o VIRTUALPLANT_UDP) espresso in numero di passi di simulazione
- T: tempo di discretizzazione utilizzato per l'esecuzione dei modelli definiti
- HORIZON: lunghezza dell'orizzonte di controllo che il nodo controller deve inviare ai nodi attuatori
- N_SENSOR_HORIZON: lunghezza dell'orizzonte di output dei pacchetti inviati dai nodi sensore al nodo controller
- N_STATE: numero totale di stati del modello utilizzato
- N_CMD: numero di nodi attuatori utilizzati
- MS_PER_TICK: intervallo temporale (in millisecondi) tra l'esecuzione di due passi di simulazione; in caso di modelli particolarmente onerosi dal punto di vista computativo, tale parametro permette di rallentare l'evoluzione del modulo VIRTUALPLANT
- FAILURE_TOL: permette di inserire un limite massimo al numero di mancate ricezioni di pacchetti UDP avvenute durante la simulazione del modulo VIRTUALPLANT; al suo raggiungimento la simulazione viene interrotta segnalando il superamento della soglia impostata.

Bibliografia

[1] Bemporad A., Heemels W.P.M.H., and M. Johansson (Eds.). *Networked Control Systems*, volume 406 of *Lecture Notes in Control and Information Sciences*. Springer, 2010. 11

[2] A. Chaillet and A. Bicchi. Delay compensation in packet-switching networked controlled systems. In *Proc. IEEE Int. Conf. on Decision and Control*, pages 3620–3625, 2008. 5, 8

[3] T. Fabbri, D. Fenucci, S. Falasca, M. Gamba, and A. Bicchi. Packet-based dynamic control of a furuta pendulum over ethernet. In *Mediterranean Conference on Control and Automation (MED '13)*, Platanias-Chania, Crete, Greece, 2013. 51

[4] S. Falasca, M. Gamba, and A. Bicchi. A strategy for dynamic controller emulation in packet-based networked control. In *International Conference on Informatics in Control (ICINCO '13)*, Reykjavík, Iceland, 2013. 5, 8

[5] L. Greco, A. Chaillet, and A. Bicchi. Exploiting packet size in uncertain nonlinear networked control systems. *Automatica*, 48:2801–2811, 2012. 5, 8

[6] L. A. Montestruque and P. Antsaklis. Stability of model-based networked control systems with time-varying transmission times. *IEEE Trans. on Automat. Contr.*, 49(9):1562–1572, 2004. 5

[7] I. G. Polushin, P. X. Liu, and C.-H. Lung. On the model-based approach to nonlinear networked control systems. *Automatica*, 44(9):2409–2414, 2008. 5

[8] D. E. Quevedo, E. I. Silva, and G. C. Goodwin. Packetized predictive control over erasure channels. In *Proc. American Control Conf.*, pages 1003–1008, NY City, USA, July 2007. 5