POLITECNICO DI MILANO
Dipartimento di Ingegneria dell'informazione

PROGETTO E SVILUPPO DI UNA LIBRERIA C DI REGOLATORI CON AUTOTUNING PER USO STAND-ALONE, IN MODELICA E IN LABVIEW

Relatore: Prof. Alberto Leva

Tesina di laurea di:

Antonio Congedi Mat. 736031

Anno accademico 2010/2011

Ai miei genitori,

per esser stati un costante sostegno

durante questo lungo e difficile percorso.

Alla mia dolce metà Valentina,

per aver camminato al mio fianco

dall'inizio alla fine.

Indice dei contenuti

Indice dei contenuti . . . . . **III**

Indice delle figure . . . . . **V**

Introduzione **1**

1 Metodi di autotuning PID **6**

1.1 Introduzione ai controllori PID **6**

1.1.1 Forma standard ISA-PID **12**

1.1.2 Regolatore PID a tempo discreto **14**

1.1.3 Internal model control (IMC) **18**

1.2 Politica di Autotuning PID **23**

2.2.1 Alcuni metodi in letteratura **26**

2 Architettura del sistema **30**

2.1 Introduzione a LabVIEW **30**

1.1.1 Interfacciamento tra codice C e LabVIEW **32**

2.2 Introduzione a Modelica **34**

1.2.1 Interfacciamento tra codice C e Modelica **37**

2.3 Struttura degli autotuner considerati **38**

2.3.1 Autotuner relay based **38**

2.3.2 Autotuner basato su IMC con tuning contestuale **42**

2.3.3 Autotuner basato su IMC e metodo delle aree **45**

3 Descrizione del codice 52

3.1 Strutture dati e funzioni fondamentali **52**

3.2 Utilizzo della libreria con LabVIEW **55**

3.3 Utilizzo della libreria con Modelica **61**

4 Esempi di applicazione 64

4.1 Uso stand-alone **64**

4.2 Uso in LabVIEW. **71**

4.3 Uso in Modelica **77**

Conclusioni 82

Bibliografia 84

Indice delle figure

1.1 Classico schema a blocchi per un sistema di controllo 8

1.2 Schema a blocchi di un controllore PID 9

1.3 Schema a blocchi alternativo tra set-point Ysp(s) ed uscita Y(s) 12

1.4 Schema a blocchi retroazionato per il controllo IMC 17

1.5 Schema a blocchi alternativo per il controllo IMC 18

1.6 Schema a blocchi per l'implementazione reale del controllo IMC 20

1.7 Schema a blocchi alternativo per l'implementazione reale del controllo IMC 21

1.8 Schema di controllo generale con sistema di autotuning 22

1.9.1 Caratteristiche rilevabili nel dominio del tempo 23

1.9.2 Caratteristiche rilevabili nel dominio della frequenza 23

2.1 Finestra del Front Panel e Block Diagram in LabVIEW 30

2.2 Rappresentazione in LabVIEW di una Flat Sequence Structure 32

2.3 Rappresentazione in LabVIEW di una Stacked Sequence Structure 33

2.4 Architettura generale del software Dymola 34

2.5 Schema a blocchi in Modelica dell'interpretazione del codice sino alla simulazione . 34

2.6 Schema a blocchi retroazionato con relè 38

2.7 Schema di controllo implementato "relay based" 39

2.8 Luogo delle radici con aggiunta di un integratore al processo 40

2.9 Tipiche risposte a gradino per un sistema di primo e secondo ordine 44

2.10 Rappresentazione delle aree S1ed S2 per il metodo delle aree 44

2.11 Rappresentazione dell'ingresso e dell'uscita al processo P(s) 46

2.12 Rappresentazione delle operazioni eseguite con la risposta in uscita 46

2.13 Rappresentazione del vettore dedicato alla raccolta dei dati campionati 47

2.14 Illustrazione del metodo della tangente 48

2.15 Illustrazione dei parametri per la determinazione del modello di secondo ordine . . . 49

3.1 Schema a blocchi realizzato per il controllo 51

3.2 Rappresentazione del "wirinig" di due blocchi in codice C 52

3.3 Blocchetto della Call Library Function in LabVIEW 53

3.4 Finestra di configurazione (scheda "Function") della Call Library Function in LabVIEW 54

3.5 Finestra di configurazione (scheda "Parameters") della Call Library Function in LabVIEW 54

3.6 Block Diagram del programma in LabVIEW ed indicazione dei parametri principali . 55

3.7 Pannello frontale del programma realizzato in LabVIEW 56

3.8 Pannello frontale del sub-VI RegistratoreSuDisco in LabVIEW 3.10 57

3.9 Schema a blocchi di ATPIdigital in Modelica 58

3.11 Blocchetto della FDT (funzione di trasferimento) in Dymola 60

4.1 Schema a blocchi realizzato per il controllo 61

4.2 Example1.1 stand-alone : pm=80 ; slope=0.3; Ts=0.3 63

4.3 Example1.2 stand-alone : pm=80 ; slope=0.3; Ts=0.3; lambda=0.1 63

4.4 Example1.3 stand-alone : Ts=0.3; lambda=0.005 64

4.5 Risposta a gradino acquisita per il petodo delle aree 65

4.6 Example1.4 stand-alone : pm=70 ; slope=0.3; Ts=0.2 66

4.7 Example1.4 stand-alone : pm=70 ; slope=0.3; Ts=0.2 66

4.8 Example1.6 stand-alone : Ts=0.005; lambda=0.005 67

4.9 Schema a blocchi realizzato per il controllo 69

4.10 Block Diagram del programma VI realizzato 70

4.11 Example2.1LabVIEW : pm=70; slope=0.3; Ts=0.2 71

4.12 Example2.2 LabVIEW : pm=70; slope=0.3; Ts=0.2 71

4.13 Example2.3LabVIEW : pm=80; slope=0.05; Ts=0.1; lambda=0.1 72

4.14 Example2.4LabVIEW : pm=80; slope=0.05; Ts=0.01; lambda=0.1 73

4.15 Example2.5 LabVIEW : Ts=0.005; lambda=0.1 74

4.16 Schema a blocchi del programma realizzato in Modelica 75

4.17 Example 3.1 Modelica: pm=70; slope=0.3; Ts=0.2 76

4.18 Example 3.2 Modelica: pm=60; slope=0.1; Ts=0.1 77

4.19 Example 3.3 Modelica: pm=60; slope=0.05; Ts=0.1 78

4.20 Example 3.4 Modelica: lambda=0.1; Ts=0.005 79

Introduzione

Fin dall'inizio della loro apparizione i controllori di tipo PID hanno ricoperto un ruolo chiave nel mondo dell'automazione, grazie alla loro versatilità e alla loro struttura semplice. Essi utilizzano in modo combinato tre leggi di controllo: azione proporzionale, azione integrale e azione derivativa . Da qui l'acronimo PID, che identifica questi particolari controllori. Benché utilizzino una tecnologia realizzativa antiquata per gli standard tecnologici attuali (possono infatti essere di tipo meccanico, idraulico o anche elettronico e digitale), questi controllori vengono ancora diffusamente utilizzati soprattutto per la relativa semplicità di taratura dei loro parametri di lavoro. E' infatti ragionevole ritenere che il controllore PID, nelle sue innumerevoli varianti, gestisca attualmente circa il 95% degli anelli di regolazione presenti negli impianti.

Il presente lavoro di tesina affronta il problema della progettazione di alcuni algoritmi di autotuning realizzati interamente in linguaggio di programmazione C, in grado di sintetizzare un controllore PID per processi solitamente equiparabili a sistemi del primo o secondo ordine con ritardo. Successivamente si è passati alla realizzazione delle corrispondenti librerie statiche e dinamiche, per utilizzo stand-alone in due diversi ambienti di sviluppo come LabVIEW e Modelica.

La sintesi dei controllori PID consiste nella scelta della configurazione più adatta all'applicazione dei parametri : K_p (costante di proporzionalità) ; T_i (tempo di integrazione) ; T_d (tempo derivativo) e T_s (periodo di campionamento per gli algoritmi digitali).

I regolatori PID commercialmente disponibili sono previsti per il controllo di variabili di processo relativamente lente, mentre per le variabili veloci si ricorre a routine di controllo ad hoc.

La procedura di tuning, solitamente fatta manualmente, non è banale e può richiedere molti tentativi che devono essere poi verificati, comportando un notevole dispendio di tempo. Le prestazioni del sistema dipenderanno, inoltre, dall'esperienza

dell'utente, che si occupa della sintonizzazione e dalla conoscenza che si ha del processo.

Normalmente, la procedura da effettuare è la seguente:

- l'operatore osserva il comportamento del sistema da controllare, anche stimolandolo con segnali noti, in modo da dedurre le informazioni necessarie;

- fissa il comportamento che ritiene di poter ottenere con il controllo a controreazione;

- calcola su base euristica il valore dei parametri che il regolatore deve avere per ottenere tale comportamento.

Va evidenziato il fatto che la procedura di tuning non va effettuata solo al momento della sua installazione, ma deve essere ripetuta tutte le volte che è necessario. In alcuni casi, infatti, sarà opportuno effettuarla nuovamente. Come, ad esempio, nei seguenti casi:

- quando viene modificato il punto di lavoro con conseguente variazione del guadagno e/o della dinamica dell'attuatore o del sistema da controllare;

- quando si verificano variazioni delle modalità operative in grado di alterarne il comportamento statico e/o dinamico;

- quando agiscono disturbi esterni in grado di alterare le caratteristiche del sistema da controllare;

- quando si riscontrano cambiamenti del comportamento del sistema dovuti, per esempio, ad invecchiamento.

Nasce, quindi, l'esigenza di realizzare un metodo il cui scopo è ottenere la regolazione automatica del controllore. Dalla formalizzazione di tali passi viene progettato il sistema di autotuning.

Demandare il calcolo dei parametri allo stesso sistema di gestione del controllo PID permette di scavalcare completamente tutte queste difficoltà e di poter ricalcolare automaticamente, in campo, i loro valori, ogni qualvolta lo si ritiene necessario.

Alcune indicazioni operative, che caratterizzano un buon metodo di autotuning, sono:

- il regolatore deve essere facile da sintonizzare;
- la procedura di auto-tuning non deve essere dispendiosa in termini di tempo e non deve disturbare significativamente il sistema;
- il controllo deve essere stabile, flessibile, robusto e adattarsi a diverse dinamiche (ritardi più o meno marcati, presenza di disturbi esterni, etc.);
- deve essere possibile velocizzare o rallentare la prontezza del sistema in modo semplice;
- le informazioni necessarie per il tuning devono essere direttamente disponibili;
- le perfomances del sistema devono migliorare.

Per realizzare lo scopo si può ricorrere a tecniche adattative di self-tuning, oppure di auto-tuning. Le prime prevedono la sintonizzazione del regolatore on-line ogni volta che le specifiche date non sono più soddisfatte; le seconde, sviluppate in questo lavoro, eseguono la taratura non di continuo, ma su richiesta di un operatore o di un segnale esterno. Si tratta, quindi, di tuning on-demand.

La letteratura è ricca di tecniche di autotuning, molte di esse come le "*model based approaches*", che permettono di ottenere in modo esplicito il modello del processo e,

di conseguenza, il tuning del regolatore è basato su tale modello; in altre, chiamate "*characteristics based approaches*", non vi è alcuna stima del sistema.

In questo lavoro si è focalizzata l'attenzione su 3 differenti approcci :

1. Characteristics based autotuner nel dominio della frequenza (Metodo Relay based);

2. Model based autotuner con contextual model parametrisation (Metodo IMC-λ);

3. Model based autotuner nel dominio del tempo (Metodo delle aree con sintesi IMC).

La scelta della realizzazione in C non è casuale. Oltre ai noti vantaggi che un linguaggio di programmazione come esso offre, tra cui efficienza, sinteticità e portabilità, l'obiettivo era rivolto alla realizzazione di appropriate librerie statiche e dinamiche per uso stand-alone in Modelica e LabVIEW. Infatti , è noto che la separazione del codice in librerie a collegamento dinamico e statico permette di suddividere il codice eseguibile in parti concettualmente separate, che verranno caricate solo se effettivamente necessarie. Inoltre, una singola libreria, caricata in memoria, può essere utilizzata da più programmi, senza la necessità di essere nuovamente caricata, il che permette di risparmiare le risorse del sistema.

Partendo dal codice C sviluppato, sono state realizzate delle simulazioni sia nello stesso ambiente, graficandone il risultato con l'ausilio del programma SciLab (noto pacchetto di programmi gratuiti per la computazione numerica sviluppati dallo INRIA e dallo ENPC in Francia), sia in ambiente LabVIEW e Modelica.

La caratteristica fondamentale di LabVIEW rispetto agli altri ambienti di sviluppo è l'uso di un linguaggio di programmazione grafico, detto linguaggio G, che permette di utilizzare schemi a blocchi invece che linee di codice. Inoltre, risulta adatto per implementazione su impianto; quindi, la libreria sviluppata consente di provare in simulazione esattamente il codice che verrà utilizzato per i calcoli.

D’altro canto Modelica, un linguaggio di modellazione a oggetti definito formalmente e sviluppato dalla Modelica Association, grazie alle vaste librerie di modelli, consente una modellazione fedele di sistemi integrati anche molto complessi. In entrambi, mediante l’editor grafico e alcune librerie multi-engineering, si semplifica la modellazione: le librerie contengono elementi corrispondenti a dispositivi fisici che devono essere semplicemente trascinati sul foglio di lavoro per costruire il modello, mentre le interazioni fra i componenti sono riprodotte e descritte dalle connessioni grafiche. Questo significa che i modelli possono essere organizzati in maniera intuitiva con le stesse modalità con cui si compone il sistema fisico reale.

La stesura del codice, le simulazioni e la creazione di librerie è stata realizzata e testata intermente in ambiente Windows. Con opportune modifiche sarà possibile, quindi, esportare il tutto in ambiente Linux. Inoltre, il codice realizzato sarà rilasciato sotto GNU GPL. Come ogni licenza di software libero, essa concede ai licenziatari il permesso di modificare il programma, di copiarlo e di ridistribuirlo con o senza modifiche, gratuitamente o a pagamento, secondo i termini della GPL stessa.

Capitolo 1

Metodi di Autotuning PID

Questo capitolo contiene nella prima parte delle nozioni generali sui regolatori PID, evidenziandone struttura, vantaggi e svantaggi. In particolare, ne verrà considerata una particolare forma chiamata ISA-PID utilizzata in questa tesina negli algoritmi di controllo degli autotuner. Saranno anche descritte le modalità di implementazione possibili per il calcolo del controllo a tempo discreto ed una particolare modalità di controllo chiamata con l'acronimo *IMC* utilizzata per uno dei metodi di autotuning realizzato. Successivamente, è presentata una panoramica sulle possibili politiche adottabili per realizzare un autotuning PID ed alcune delle diverse varianti che sono proposte in letteratura. Si potrà notare che l'argomento è molto vasto ed oggetto di continuo studio per ottenere risultati sempre migliori.

1.1 Introduzione ai controllori PID

Il successo dei regolatori PID è legato a diversi fattori:

- notevole efficacia nella regolazione di un'ampia gamma di processi industriali, anche in relazione alle specifiche di prestazione non sempre stringenti;
- relativa semplicità di taratura;
- importanza e convenienza economica della standardizzazione.

In molti sistemi di controllo non è difficile notare prestazioni scadenti che sono dovute a svariati motivi. I principali, sono dovuti a problemi nei sensori negli attuatori , rumore, filtri anti-aliasing inadeguati, errori di calibrazione dei sensori, eccessivo filtraggio nei sensori "smart", cattivo dimensionamento, isteresi o attriti statici nelle valvole di regolazione . In presenza di problemi di tale natura, il ruolo della legge di controllo può diventare modesto, per cui viene meno la motivazione di cercare leggi di controllo più sofisticate di quelle PID. Spesso, infatti, sono anche frequentemente usati come elementi di schemi di controllo più complessi e articolati, come il controllo in cascata che, sfruttando la conoscenza di specifiche proprietà dinamiche del processo, possono portare notevolissimi miglioramenti delle prestazioni a fronte di costi e sforzi realizzativi relativamente modesti.

Inoltre, per sfruttare pienamente algoritmi di controllo più complessi, è necessaria anche una conoscenza approfondita del processo da controllare e ciò richiede investimenti non trascurabili e con ritorni non quantificabili a priori per lo sviluppo, la messa a punto e la validazione di modelli matematici accurati.

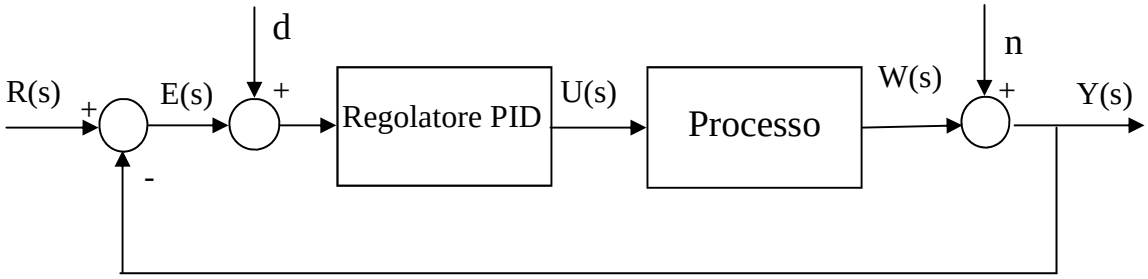
In sintesi, si può forse affermare che i PID hanno successo perché rappresentano una soluzione non facilmente superabile, in generale, nel rapporto efficacia/costo (globale), almeno nella maggior parte dei casi dove le specifiche non siano estremamente stringenti.

Il termine controllo definisce l'azione svolta per portare e mantenere ad un valore prefissato un parametro fisico di un impianto o di un processo (ad esempio: la temperatura di un forno, il livello di un fluido in un serbatoio, la posizione del braccio di un robot, la velocità di rotazione di un motore, ecc.). Indicando con $r(t)$ il valore che si vuole far assumere alla variabile controllata e con $y(t)$ il valore effettivamente assunto da tale grandezza, possiamo introdurre una funzione d'errore definita come: $e(t) = r(t) - y(t)$. Lo scopo dell' azione di controllo è quello di applicare la migliore scelta possibile della funzione $u(t)$, detta variabile di controllo, che :

- renda il sistema asintoticamente stabile;
- minimizzi il valor medio di $e(t)$;

- riduca al livello minore possibile il tempo di risposta e le fluttuazioni intorno al valore asintotico in concomitanza di transitori di $r(t)$.

Passando alla trasformata di Laplace, possiamo schematizzare il problema nel seguente modo:



[1.1] Classico schema a blocchi per un sistema di controllo

Le principali componenti del sistema sono: l'errore $E(s) = R(s) - Y(s)$, il controllore che ha il compito di trasformare il segnale d'errore in un segnale $U(s)$ che agisce sul processo sottoposto a controllo, un sensore posto all'interno del processo che misura la grandezza fisica da controllare fornendo il segnale $Y(s)$.

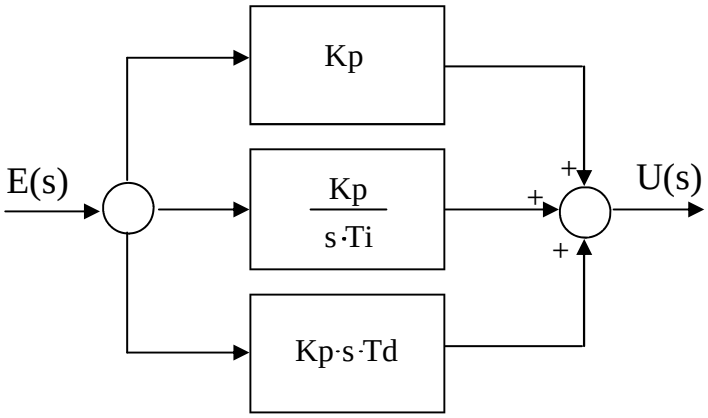
Lo schema mostra anche le sorgenti di rumore interne al sistema e le altre perturbazioni esterne che si sommano ai vari segnali. Una caratteristica essenziale di un buon controllore è quella di possedere una elevata "robustezza" rispetto alle possibili modifiche del sistema nel tempo, mantenendo la grandezza controllata al valore desiderato, anche in presenza di eventuali piccole variazioni della funzione di trasferimento che caratterizza il processo.

Supponendo che il controllore ed il processo possano essere schematizzati come sistemi lineari e stazionari caratterizzati da una funzione di trasferimento $G_c(s)$ e $G_p(s)$, utilizzando la ben nota relazione dei sistemi reazionati, si può scrivere la funzione di trasferimento del sistema ad anello chiuso:

$$T(s) = \frac{Y(s)}{R(s)} = \frac{G_c(s) \cdot G_p(s)}{1 + G_c(s) \cdot G_p(s)} \qquad [1]$$

Il problema generale del controllo si riduce, quindi, a determinare, per una certa funzione di trasferimento del processo $G_p(s)$, la migliore funzione di trasferimento del controllore $G_c(s)$, che ottiene il risultato su $T(s)$.

Uno schema ampiamente utilizzato è quello PID, acronimo che indica l'utilizzo combinato di tre funzioni di controllo di tipo Proporzionale, Integrale e Differenziale. Lo schema generale di tale controllore è il seguente:



[1.2] Schema a blocchi di un controllore PID

In pratica, l'uscita di un controllore PID è costituita dalla somma di tre termini:

$$U(s) = K_p \cdot \left(1 + \frac{1}{sT_I} + sT_d\right) \cdot E(s) \qquad [2]$$

Il primo termine, K_p, è detto coefficiente proporzionale. Ponendo $T_d = 0$ e $T_i = \infty$ l'equazione precedente si riduce a: $U(s) = K_p \cdot E(s)$ e, in tal caso, si parla di controllore proporzionale o di tipo P. Il contributo dovuto a K_p è appunto proporzionale all'errore e diminuisce man mano che l'errore si avvicina a zero. Al crescere di K_p il guadagno in continua del sistema si avvicina all'unità e la costante tempo viene ridotta. In altre parole, aumentando K_p, il valore asintotico dell'uscita sarà

sempre più vicino al valore richiesto. Si noterà tuttavia che l'uscita non arriverà mai esattamente al set-point, a meno di non far tendere K_p all'infinito. Il valore asintotico dell'errore viene detto offset e può essere calcolato facilmente utilizzando il teorema del valore finale della trasformata di Laplace:

$$\lim_{t\to\infty} e(t) = \lim_{s\to 0} s \cdot E(s) = \frac{1}{1 + K_p \cdot \mu} \qquad [3]$$

da cui verifichiamo che si riduce a zero solo quando il coefficiente proporzionale tende ad infinito. Si dimostra, inoltre, che all'aumentare di K_p il transitorio che segue rapide variazioni di $r(t)$ è caratterizzato da oscillazioni di frequenza sempre più elevata e meno smorzate.

Riassumendo, utilizzando un controllore esclusivamente proporzionale si produce una differenza (offset) tra il valore richiesto e quello effettivamente ottenuto. Tale differenza può essere ridotta aumentando il guadagno del controllore. Tuttavia, se il processo da controllare possiede coppie di poli complessi e coniugati, l'aumento del coefficiente proporzionale è accompagnato da un corrispondente aumento delle oscillazioni generate a seguito di rapidi transitori.

Per porre rimedio a questo problema è necessario aggiungere al termine proporzionale un termine aggiuntivo che elimini a priori la presenza dell'offset. In un controllore PID tale funzione è svolta dal termine inversamente proporzionale a $s \cdot T_i$, detto anche contributo integrale (e un tempo chiamato anche "automatic reset"). È evidente che in presenza di un offset costante il contributo integrale è destinato a crescere indefinitamente nel tempo e questo ci permette di attivare una efficace azione correttiva. La costante T_i è detta tempo di reset.

L'effetto dell'integrazione è tanto più importante, quanto più T_i è piccolo ; riducendolo, il sistema risponde più velocemente al transitorio, ma si osservano anche delle forti oscillazioni.

Nella maggior parte delle applicazioni, la variabile di controllo di un anello di regolazione è limitata superiormente e inferiormente. Se il sistema è ben progettato,

essa assume valori abbastanza lontani dai limiti di saturazione, ma può capitare che li raggiunga in occasione di variazioni rilevanti del set-point o del disturbo di carico.

Quando la variabile di controllo è in saturazione, il processo evolve con ingresso costante, come se fosse in anello aperto. Anche il regolatore si trova ad evolvere in anello aperto; infatti, viene a mancare l'effetto della retroazione tra la sua uscita e l'errore. Poiché l'integratore è un sistema dinamico non asintoticamente stabile, in queste condizioni può allontanarsi di molto dal campo di valori utili per il controllo; se ciò accade, occorre tempo, dopo che l'errore è tornato a valori tali da non richiedere più la saturazione della variabile di controllo, prima che l'integratore, e con esso l'uscita del regolatore, ritorni a valori utili per il controllo. Tale fenomeno è conosciuto come *wind-up*. Un modo semplice e diffusamente utilizzato per evitarlo è quello di arrestare l'integrazione quando interviene la saturazione.

Per catene di regolazione complesse, a più livelli gerarchici o con anelli annidati, la saturazione di un attuatore o di un regolatore comporta di arrestare l'integrazione in tutti i regolatori della struttura a monte.

Il contributo derivativo al controllo T_d (costante tempo di derivazione), invece, tiene conto delle rapide variazioni dell'errore e cerca, in qualche modo, di anticipare la futura azione correttiva tenendo conto delle variazioni dell'errore nei tempi più recenti. In pratica, questo si concretizza in una riduzione delle oscillazioni.

Si nota che la presenza del termine derivativo introduce uno zero ed aumenta il coefficiente di s nel polinomio posto al denominatore.

Ambedue questi effetti producono una riduzione delle oscillazioni spurie che si verificano in occasione dei transitori, contribuendo a stabilizzare il sistema ed evitando che questo oscilli intorno al valore asintotico.

D'altro canto, in presenza di un forte rumore esterno, il contributo derivativo tende ad amplificare l'effetto del rumore, producendo una instabilità addizionale .

In conclusione, il problema del progetto di un controllore PID si riduce alla scelta dei valori più opportuni per i parametri K_p, T_i e T_d. Tale scelta non è banale, perché

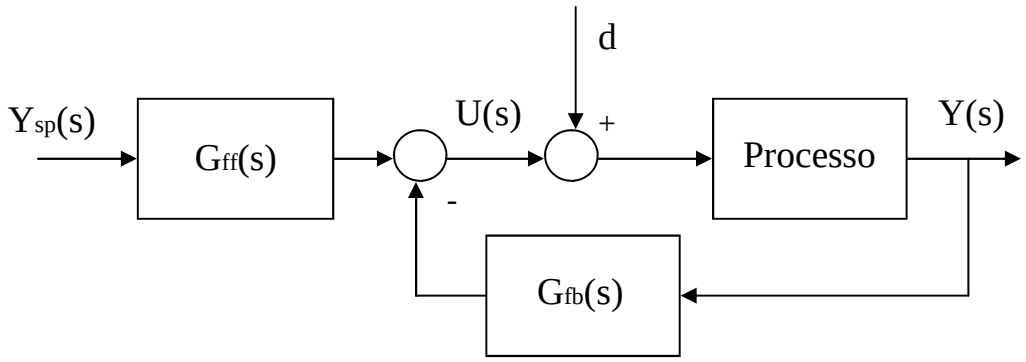
richiede la conoscenza dettagliata delle proprietà del processo che si vuole controllare. Esistono vari metodi per effettuare la scelta (tuning) dei coefficienti più opportuni.

1.1.1 Forma standard ISA-PID

L'implementazione delle legge di controllo nei regolatori commerciali, spesso, assume la seguente forma:

$$U(s) = K_p \cdot \left(bY_{sp}(s) - Y(s) + \frac{1}{sT_i} E(s) + \frac{sT_d}{a + sT_d / N} (cY_{sp}(s) - Y(s) \right) \qquad [4]$$

Con riferimento al seguente schema a blocchi:



[1.3] Schema a blocchi alternativo tra set-point Ysp(s) ed uscita Y(s)

i coefficienti b e c assumono, di regola, valori compresi fra 0 e 1. Un PID in forma standard ISA permette di elaborare in modo indipendente le risposte al set-point e al disturbo di carico . Considerando le seguenti funzioni di trasferimento:

$$G_{ff}(s) = \frac{U(s)}{Y_{sp}(s)} = K_p \cdot \left(b + \frac{1}{sT_i} + \frac{sT_d}{1 + sT_d / N} c \right) \quad [5]$$

$$G_{fb}(s) = \frac{U(s)}{Y(s)} = K_p \cdot \left(1 + \frac{1}{sT_i} + \frac{sT_d}{1 + sT_d / N} \right) \quad [6]$$

i parametri K_p T_i T_d oltre ad N, intervengono nell'anello di retroazione e consentono di aggiustarne la stabilità, la dinamica e quindi la risposta al disturbo di carico; i parametri b e c consentono di intervenire sulla risposta di $Y(s)$ a $Y_{sp}(s)$. Questa struttura del regolatore si dice "a due gradi di libertà", intendendo con questo che i cammini del segnale da $Y_{sp}(s)$ ad $U(s)$ e da $Y(s)$ ad $U(s)$ sono diversi; è possibile, pertanto, separare i problemi relativi al grado di stabilità e alla reiezione dei disturbi da quelli relativi all'inseguimento del set-point.

Osservando lo schema di riferimento presentato all'inizio di questo capitolo, risulta evidente che la variabile di controllo $U(s)$ risente subito della variazione del set-point $Y_{sp}(s)$; infatti, i due segnali sono separati dalla sola dinamica del PID. A meno che la risposta del processo sia praticamente istantanea a fronte di una variazione a scalino del riferimento, l'azione proporzionale subisce una brusca variazione a scalino che andrà a riflettersi sulla variabile di controllo.

Il peso sul set-point nell'azione proporzionale (parametro b nella forma standard ISA) viene introdotto proprio per ovviare a questo inconveniente. La possibilità di limitare gli sbalzi del controllo risulta particolarmente utile, ad esempio, negli anelli interni dei controlli in cascata, nei quali il set-point non è imposto direttamente da un operatore, ma fornito dal regolatore esterno. Nel caso in cui b sia diverso da 1, a regime l'azione proporzionale non sarà nulla; comunque, non influenza le fasi avanzate del transitorio , poiché in esse prevale nettamente l'azione integrale. E' bene notare come al crescere di b la risposta al set-point diventi più pronta, ma con possibilità di sovra elongazione. Scegliendo b nell'intervallo 0-1 si può trovare un buon compromesso tra velocità di risposta e sovra elongazione e contenere la sovraelongazione nella risposta al set-point, senza penalizzare la banda del sistema retroazionato.

Un altro possibile accorgimento implementativo riguarda l'introduzione del peso sul set-point nell'azione derivativa (corrispondente al parametro *c* nella forma standard ISA); anche in questo caso l'obiettivo è quello di evitare che il controllo subisca sbalzi troppo bruschi a fronte di variazioni repentine dell'errore. Il peso *c* ha influenza solo negli istanti in cui $Y_{sp}(s)$ varia (quando è costante la sua derivata è nulla); ne consegue che *c* gioca un ruolo fondamentale, ad esempio, negli anelli interni dei controlli in cascata, il cui set-point può variare di continuo.

1.1.2 Regolatore PID a tempo discreto

Per realizzare le azioni di un regolatore PID con un sistema di elaborazione digitale, è necessario ottenerne equivalenti discreti. Il problema della discretizzazione consiste, data una funzione di trasferimento a tempo continuo e dato un tempo di campionamento T_s, nel determinare i parametri di una funzione di trasferimento a tempo discreto che ne approssimi abbastanza bene il comportamento. La discretizzazione può essere compiuta secondo vari metodi :

- discretizzazione in avanti o formula esplicita di Eulero;
- discretizzazione all'indietro o formula implicita di Eulero;
- metodo del trapezio (o di Tustin).

Un altro metodo, che fornisce l'equivalente discreto di un sistema dinamico lineare, è quello dello Zero Order Hold (ZOH), che fornisce il valore esatto dell'uscita agli istanti di campionamento, nell'ipotesi in cui l'ingresso sia costante nel periodo di campionamento.

La scelta della frequenza deve scaturire da un buon compromesso tra esigenze diverse e talvolta contrapposte. Considerazioni relative al costo del sistema spingono a scegliere un valore basso, affinché la potenza di calcolo necessaria per eseguire gli algoritmi di controllo sia minima ed i convertitori A/D e D/A più economici. L'analisi

delle prestazioni del sistema di controllo impone, però, un limite inferiore assoluto. La condizione che il sistema di controllo in anello chiuso debba inseguire il riferimento fino alla banda passante

$$f_c = \frac{\omega_c}{2\pi} \ , \qquad [7]$$

impone un limite inferiore assoluto alla frequenza di campionamento: per il teorema di Shannon essa deve essere almeno doppia di f_c . Allo stesso risultato si giunge se si considera di dover acquisire senza perdita d'informazione le componenti di y in banda passante. Il limite imposto dal teorema del campionamento è, tuttavia, teorico e non adeguato alle applicazioni di controllo. In letteratura esistono varie indicazioni di massima che permettono di individuare la frequenza di campionamento ottimale per il singolo problema in esame.

Per quanto concerne la compensazione dei disturbi, il periodo di campionamento rappresenta il valore limite del ritardo di reazione di un sistema di controllo digitale a un disturbo di carico agente sul processo; pertanto, si tratta di un parametro critico ai fini della prontezza con cui il controllo reagisce al disturbo.

Con riferimento alla forma standard ISA, la discretizzazione delle tre azioni ha le seguenti forme:

Azione proporzionale:

$$P(s) = K_p \Leftrightarrow P^*(k) = K_p\left(by^{o*}(k) - y^*(k)\right)$$

Azione integrale :

$$I(s) = \frac{K_p}{sT_i} \Leftrightarrow I^*(k) = I^*(k-1) + \frac{K_p T_s}{T_i}\left(y^{o*}(k) - y^*(k)\right)$$

Azione derivativa :

$$D(s) = \frac{sK_p T_d}{1 + sT_d / N} \Leftrightarrow D^*(k) = \frac{T_d}{T_d + NT_s} D^*(k-1) + \frac{K_p N T_d}{T_d + NT_s}\left(y^{o*}(k) - y^*(k-1)\right)$$

L'equivalente discreto del PID, pertanto, sarà dato dalla somma delle tre azioni:

$$U^*(k) = P^*(k) + I^*(k) + D^*(k) \qquad [8]$$

In questo modo è possibile calcolare la legge di controllo in due diversi modi:

- *Posizionale* : in cui si calcola direttamente, ad ogni passo di campionamento, il nuovo valore del controllo;
- *Incrementale* : in cui si calcola, ad ogni passo di campionamento, la variazione del controllo.

Gli algoritmi PID assoluti (o di posizione) devono il loro nome al fatto che, ad ogni istante di campionamento kT , viene calcolato e fornito in uscita il nuovo valore del segnale di controllo $u(k)$, senza tener conto dello scostamento dal valore precedente $u(k-1)$.

Gli algoritmi incrementali, invece, devono il loro nome al fatto che viene calcolata la variazione $\Delta u(k)$ del segnale di controllo rispetto al valore assunto nell'istante precedente, cioè:

$$\Delta u(k) = u(k) - u(k-1) \qquad [9]$$

dove $u(k)$ e $u(k-1)$ possono essere calcolati mediante un algoritmo assoluto.

Quest'ultima forma è stata utilizzata in questo tesina negli algoritmi digitali. Essa è molto utilizzata nella pratica, in quanto semplifica notevolmente l'implementazione di funzionalità complementari alla mera legge di controllo (anti-windup, tracking di un segnale, commutazione manuale/automatico, logiche di blocco, etc…) nel software che rappresenta l'equivalente discreto del PID. Risulta immediato esprimere un PID in forma incrementale partendo dalla discretizzazione effettuata precedentemente.

Infatti, considerando la variazione del generico segnale a tempo discreto $v^*(k)$:

$$\Delta v^*(k) = v^*(k) - v^*(k-1) \qquad [10]$$

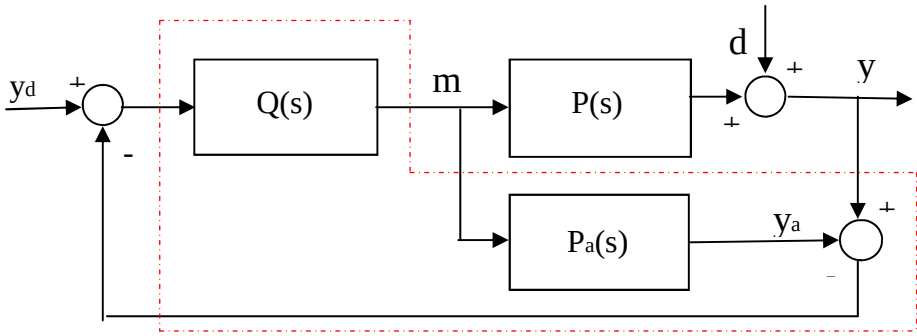
le tre azioni del regolatore del regolatore saranno:

$$\begin{cases} P^*(k) = K_p\left(b\Delta y^{o*}(k) - \Delta y^*(k)\right) \\ \Delta I^*(k) = \dfrac{K_p T_s}{T_i}\left(y^{o*}(k) - y^*(k)\right) \\ \Delta D^*(k) = \dfrac{T_d}{T_d + NT_s}\Delta D^*(k-1) - \dfrac{KNT_d}{T_d + NT_s}\left(\Delta y^{o*}(k) - \Delta y^*(k-1)\right) \end{cases}$$

Si può osservare come le variazioni delle azioni proporzionale e integrale non dipendano dai loro valori passati, mentre ciò si verifica nel caso dell'azione derivativa.

1.1.3 IMC : Internal Model Control

Questa tecnica si affida all'IMP (Internal Model Principle), che afferma che il controllo può essere raggiunto solo se il sistema di controllo incapsula, sia implicitamente sia esplicitamente, una qualche rappresentazione del processo da controllare. In particolare, se lo schema di controllo è stato sviluppato sulla base di un modello esatto del processo, allora si avrà un controllo perfetto teoricamente possibile. Il tipico schema IMC è rappresentato di seguito:



[1.4] Schema a blocchi retroazionato per il controllo IMC

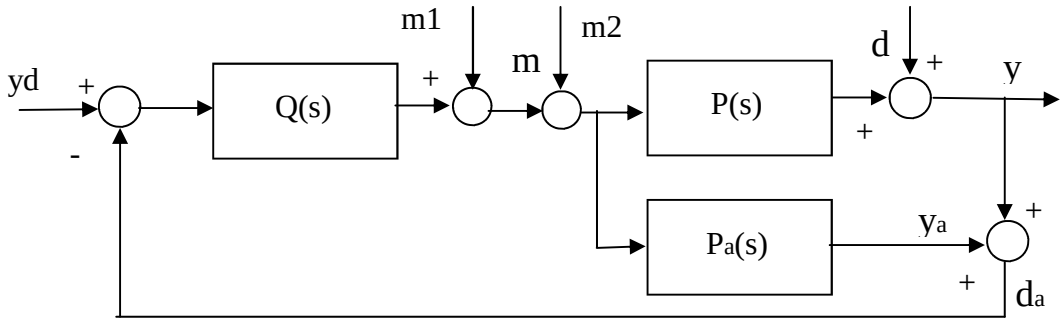
Alcune caratteristiche generali:

- Nel caso nominale , se il processo è stabile asintoticamente, la stabilità dello schema IMC è assicurata dalla stabilità di $Q(s)$;
- Le caratteristiche dello schema IMC (come ad esempio il tempo di assestamento) sono direttamente legate ai parametri di $Q(s)$;
- Il controllore complessivo è racchiuso nella parte tratteggiata di figura e contiene un modello del processo $P_a(s)$ (da cui il nome IMC);
- Il segnale in retroazione

$$d_a = \big(P(s) - P_a(s)\big)m + d \qquad [11]$$

qualora il modello fosse perfetto , $P_a(s) = P(s)$, e in assenza di disturbi in uscita, $d = 0$,è nullo. In tal caso si ha un controllo ad anello aperto feedforward. In altri termini, la necessità della retroazione deriva dalle incertezze (di modello o disturbi). d_a e può essere pensato come una misura della incertezza.

Lo studio della stabilità interna dell'IMC può essere effettuato considerando come 3 ingressi indipendenti y_d, m_1 e m_2 e come uscite indipendenti y, m e y_a



[1.5] Schema a blocchi alternativo per il controllo IMC

Il legame nel caso nominale $P_a(s) = P(s)$ è dato da

$$\begin{pmatrix} y \\ m \\ y_a \end{pmatrix} = \begin{pmatrix} PQ & (1-PQ)P & P \\ Q & -PQ & 0 \\ PQ & -P^2Q & P \end{pmatrix} \begin{pmatrix} y_d \\ m_1 \\ m_2 \end{pmatrix}$$

È pertanto immediato verificare che assumendo un modello perfetto del processo, $P_a(s) = P(s)$ e in assenza di dinamiche nascoste nelle singole funzioni di trasferimento, lo schema di controllo IMC è internamente stabile se il processo P e il controllore Q sono entrambi stabili asintoticamente.

Essendo $P(s)$ in generale noto solo attraverso un modello approssimato $P_a(s)$ e d non misurabile e dunque non noto, si può pensare di adottare la seguente strategia per implementare il controllo ideale appena ricavato:

- Ipotizzando che $P_a(s)$ sia la nostra migliore stima della dinamica del processo, la migliore stima del disturbo è ottenibile sottraendo la predizione fornita dal modello $P_a(s) \cdot m(s)$ all'uscita misurata $y(s)$ e cioè:

$$d_a = y(s) - y_a(s) = y(s) - P_a(s) \cdot m(s); \qquad [12]$$

- Ponendo $Q(s) = 1/P_a(s)$ e utilizzando la stima del disturbo al posto del disturbo vero, si ottiene un ingresso che coincide con quello ideale se $d_a(s) = d(s)$

$$m(s) = Q(s)\left[y_s(s) - d_a(s)\right] \qquad [13]$$

Lo schema a blocchi risultante è quello di partenza IMC.

In generale, però l'IMC è uno schema di controllo ideale. In particolare, si hanno i seguenti problemi :

- non si ha praticamente mai $P_a(s) = P(s)$;
- implementare $Q(s) = 1/P_a(s)$ non è sempre possibile, per la presenza di ritardi e/o zeri a parte reale positiva nel modello;
- anche in assenza di tali problemi, dall' espressione del controllo viene fuori che la sua implementazione richiederebbe uno sforzo di controllo infinito.

Per rendere implementabile l' I MC sono state proposte le seguenti modifiche :

- Si fattorizza il modello del processo in

$$P_a(s) = P_a^{+}(s)P_a^{-}(s)$$

dove $P_a^+(s)$ è a guadagno unitario e contiene tutti i termini non invertibili (ritardi, zeri a parte reale positiva) mentre in $P_a^-(s)$ rientrano tutti gli altri termini;

- Il controllore è scelto pari a

$$Q(s) = \frac{1}{P_a^-(s)} \qquad [14]$$

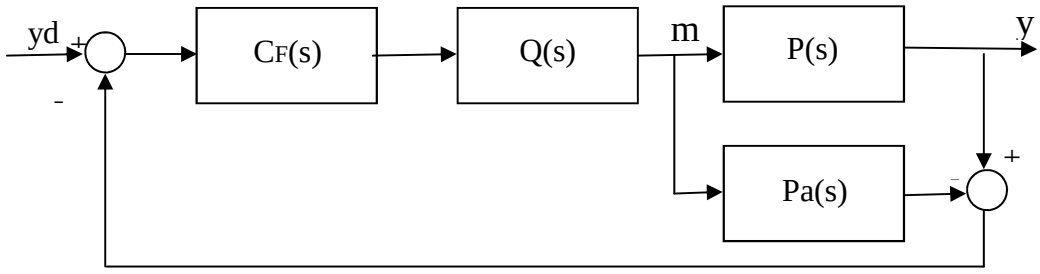
Si noti che può essere tipicamente improprio (grado del numeratore maggiore del grado del denominatore);

- Si aggiunge un filtro $C_F(s)$ della forma:

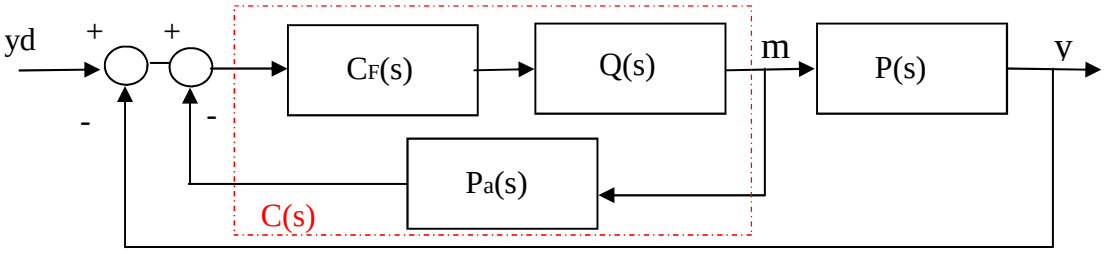
$$C_F(s) = \frac{1}{(1+\tau_F s)^m} \qquad [15]$$

con *m* tale da rendere $Q(s)$ almeno proprio e τ_F parametro libero.

Lo schema di controllo risultante può essere rappresentato in modo equivalente, secondo i due schemi a blocchi:



[1.6] Schema a blocchi per l'implementazione reale del controllo IMC



[1.7] Schema a blocchi alternativo per l'implementazione reale del controllo IMC

- Processo $P(s)$
- Filtro $C_F(s)$
- Modello interno del processo $P_a(s)$
- Compensatore $Q(s)$

Essendo l'ingresso di controllo $m(t)$ ottenibile dalla relazione

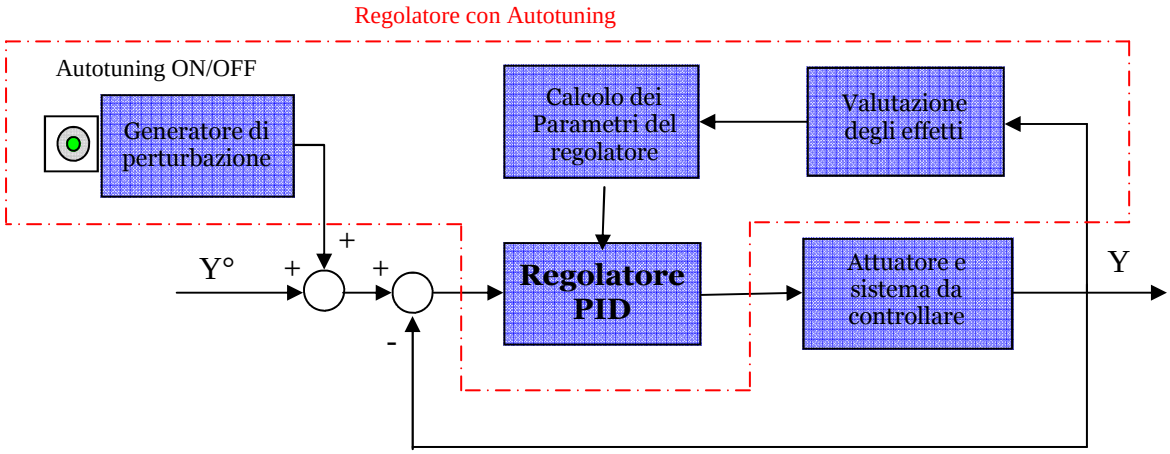
$$m(s) = Q(s)C_F(s) \cdot \left[y_d(s) - y(s) + P_a(s)m(s) \right] \qquad [16]$$

La funzione di trasferimento del controllore equivalente $C(s)$ è data da

$$C(s) = \frac{m(s)}{y_d(s) - y(s)} = \frac{Q(s)C_F(s)}{1 - Q(s)P_a(s)C_F(s)} \qquad [17]$$

1.2 Politica di autotuning

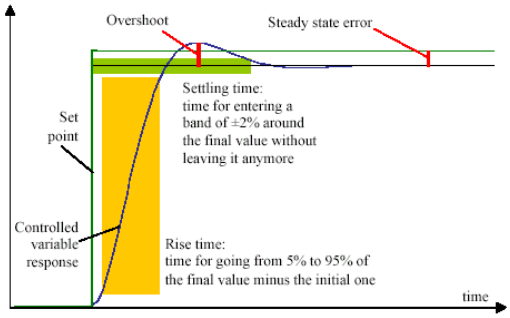
Implementare una procedura di autotuning significa determinare un algoritmo per la predisposizione automatica del valore dei parametri di un regolatore, quando esso è già collegato al sistema da controllare o l'operatore ha dato il consenso di inizializzazione.
Il tuning può essere avviato dall'operatore per sua decisione esplicita o come conseguenza di determinate manovre sul sistema da controllare. Il sistema di gestione può, inoltre, suggerire quando effettuarlo. Spesso è previsto anche l'avvio automatico in determinate situazioni, per esempio se l'errore è troppo grande. Per la determinazione di tali parametri, generalmente si stimola il sistema con segnali di vario tipo, come ad esempio segnali a gradino o sequenze di impulsi.

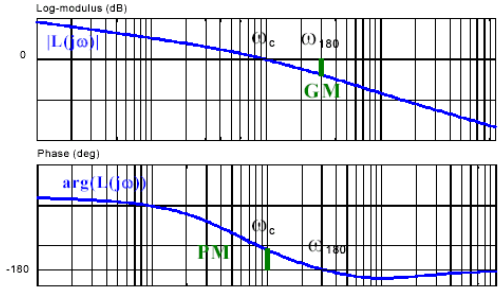


[1.8] Schema di controllo generale con sistema di autotuning

In questo caso si parla di "*experiment based autotuner*" ed in genere si ammette una perturbazione che provochi una modifica del valore della variabile controllata, mantenendola, però, entro la soglia d'accettabilità delle prestazioni. Alternativamente, un altro approccio è quello del "*non experiment based autotuner*", in cui si osserva il sistema nella sua naturale evoluzione.

In questo modo si cerca di dare una valutazione globale al comportamento del sistema secondo determinati criteri. La valutazione delle prestazioni di un processo può essere fatta nel dominio del tempo o nel domino della frequenza. La prima risulta intuitiva anche per i non esperti, ma può risultare difficile da automatizzare a causa del rumore. La seconda, d'altro canto, risulta facile da automatizzare, ma è comprensibile solo ad operatori con un minimo di conoscenze.





[1.9.1] Caratteristiche rilevabili nel dominio del tempo [1.9.2] Caratteristiche rilevabili nel dominio della frequenza

Nel domino del tempo è più facile tenere sotto controllo caratteristiche come overshoot, errore a regime, tempo di assestamento o di salita, ma nel domino della frequenza sarà più semplice imporre vincoli, ad esempio su margine di fase o di guadagno.

L'autotuner dovrà, poi, creare una descrizione del comportamento desiderato ad anello chiuso. Per fare ciò può:

- Procedere autonomamente basandosi su criteri pre-impostati: sono semplici da usare, non necessitano di un operatore esperto e sono date moltissime situazioni in cui portano a risultati soddisfacenti, ma non sono adatti a situazioni critiche in cui sia necessario un controllo sofisticato.
- Richiedere all'operatore le linee guida o in modo dettagliato le specifiche che si vogliono ottenere: modalità con prestazioni migliori ma con complessità maggiore. E' possibile richiedere specifiche lessicali che permettono di fornire

alcune linee guida, o specifiche numeriche che permettono l'operatore il massimo controllo. Per le prime, la richiesta di minimizzazione di un indice può comportare varianti impreviste sul funzionamento del resto della struttura.

- I requisiti richiesti possono essere di vario tipo: requisiti sul comportamento della variabile controllata o sulla variabile di controllo, requisiti sulle variabili di comando dell'attuatore allo scopo di tenere più basso possibile il picco di energia o potenza istantanea, oppure requisiti sulla stabilità di sicurezza sulla variabile di controllo.

Non tutte le specifiche possono essere realizzate contemporaneamente, ma bisognerà verificarne la validità di quelle introdotte, che potrebbero essere tra loro incompatibili e raggiungere il miglior controllo possibile decidendo quali aspetti preferire a discapito altri.

Le procedure di autotuning, quindi, non possono prescindere dall'esperienza. Infatti, se non esperto, un operatore, potrà fornire delle richieste che siano almeno coerenti, ma raggiungerà solo delle prestazioni sufficienti frutto di un compromesso.

Come già accennato, i metodi di autotuning sono molto numerosi e ricchi di varianti più o meno convenienti e valide. Principalmente, le politiche di ottimizzazione (Tuning rules) che permettono di ottenere i parametri del regolatore che deve avere prevedendo un particolare comportamento ad anello chiuso, si dividono in tre grandi categorie :

- *Model based , model following autotuner* .
  Si dispone di un modello del processo e lo si usa per prevedere il comportamento ad anello chiuso. Si cerca quindi di far assomigliare il comportamento ad anello chiuso a quello di un dato modello. Possiamo eseguire il tuning PID minimizzando una funzione costo che rappresenta la differenza tra la risposta ad anello e quella del modello da imitare.

- *Model based, characteristics following autotuner.*
  Si dispone di un modello, ma in anello chiuso si utilizzano solo alcuni valori caratteristici.

- *Characteristics based, characteristics following autotuner.*
  Si descrive il processo non costruendo il suo modello, ma con una serie di valori che caratterizzano il suo comportamento nel tempo o in frequenza.

- *Rule based methods.*
  Non c'è una descrizione del processo né come modello né come caratteristica di risposta; si cerca di imitare il ragionamento istintivo degli uomini, si usano reti neurali , logica fuzzy e algoritmi generici. I parametri vengono vagliati dopo l'osservazione di un transitorio generato da una variazione del set-point o del disturbo

1.2.1 Alcuni metodi in letteratura

Data una panoramica generale sui metodi di autotuning e le rispettive politiche, di seguito si citeranno solo alcune delle implementazioni presenti in letteratura.

Come già detto, per ottenere in modo esplicito il modello del processo, si può far riferimento alla risposta transitoria (Transient Response Methods) per mezzo della quale la dinamica del sistema è caratterizzata in termini della risposta a un segnale noto di ingresso. Seguendo tale principio, la tecnica più utilizzata è il I° metodo di Ziegler-Nichols (ZN Step Response Method), ripreso anche in Skogestad (2003) [36] , in cui si approssima il processo ad un sistema del primo ordine con ritardo e si studia la risposta al gradino ad anello aperto.

Alternativamente, è possibile avere una stima del processo studiando la risposta frequenziale (Frequency Response Methods). In tal senso, uno dei primi metodi che si

trovano in letteratura è quello proposto da Ziegler e Nichols (1942) [52], noto come II° metodo di Ziegler-Nichols (ZN Frequency Response Method) che ha, però, il limite di portare il sistema in condizione di oscillazioni permanenti ai limiti della stabilità, condizione spesso pericolosa per alcuni processi.

Per ovviare a tale problema, si ricorre alla tecnica del relay introdotta da Astrom e Hagglund (1984) [1], che consiste nell'applicare un relè nell'anello di retroazione durante la fase di tuning. Tale metodo è noto anche con l'acronimo ATV (Automate Tuning Variation) e porta il sistema ad un ciclo limite stabile. Tale tecnica è stata ripresa numerose volte come in Chang (1992) e Luyben (1987)[25].

Diverse sono, poi, le modifiche apportate per avere una stima più accurata del modello: il relay con isteresi bilanciato/sbilanciato in Loh (2004) [7] e Wang (1997)[49] , il SATV (Sinusoidal Automate Tuning Variation) ripresa in Girolami (2003) [16] Il metodo ATV+ proposto in Marchetti e Scali (1999)[26], [27] che permette di identificare tre punti della risposta frequenziale. Altre tecniche proposte in letteratura che si basano sul metodo del relay sono in A. Leva (1993) [1], (2005) [3] o in Friman e Waller (1997) [14], Park (1998) [29], Tan (2000) [42] ,Tan (2000) [42] e Peric (2000) [30]. In Tan (2000) [41] inoltre, viene affrontato il problema del controllo in cascata di anelli multipli. In presenza di processi asimmetrici, il metodo del relay può portare a delle stime completamente errate; sono quindi necessari degli accorgimenti come in Tan (1998) [43] , o in Lo e Kwon (2003) [23]

I metodi fin qui citati permettono di identificare solo uno, o al massimo tre punti della risposta frequenziale, così da non poter ottenere una buona sintonizzazione del regolatore PID. Tale problema può essere superato in parte utilizzando la FFT Fast Fourier Trasform come in Sung e Lee (2000) [37] , Tan (2000) [42] , Wang (1997) [47] e in Wang (2001) [50] oppure ricorrendo a metodi numerici di identificazione come in Poulin (1996) [32].

Una volta noto il modello del processo, i metodi per la sintonizzazione del regolatore sono molteplici.

Il più immediato è quello di utilizzare i valori presentati in Ziegler e Nichols (1942) [52], ottenuti col metodo della risposta frequenziale, che ha però lo svantaggio di offrire una regolazione piuttosto grossolana. Ricordando che le tecniche fin qui citate

permettono di identificare un punto della curva di Nyquist, con un controllore PID è possibile muovere tale punto nel piano complesso, in modo da soddisfare delle specifiche date.

Solitamente, i criteri a cui si fa riferimento sono il margine di guadagno e di fase ripresi in Lee (2004) [21], Park (1998) [29], in Loh (2004) [7] , Wang (2001) [51] e Ho (1995) [18]. Si può ricorrere a tecniche analitiche di tuning grazie alle quali, noto il modello del processo e del regolatore, si calcola direttamente la funzione di trasferimento ad anello chiuso.

In Junga (1999) [19] viene proposto di filtrare il comando di set-point con un filtro del primo ordine e di sintonizzare un controllore PI in modo da controllare l'overshoot della risposta ad anello chiuso. Shafiei e Shenton (1997) [34] fanno riferimento al metodo OLDP da loro stessi proposto in [33].

Liu e Daley (1999) [22] utilizzano la risposta frequenziale per ottimizzare i parametri del regolatore PID, in modo da soddisfare le specifiche desiderate.

Kozak e Hejdis (1995) [20] ricavano le regole di autotuning per un controllore discreto PID descritto nella forma di Takahashi.

Altre tecniche sono l' Haalman Method citati in Astrom e Hagglund (1995)[12] e l'IMC (Internal Model Control), ripreso in Skogestad (2003) [36] e Wang (2001) [48].

In letteratura si trovano, poi, metodi di ottimizzazione in cui la struttura del regolatore è funzione di alcuni parametri, che va ottimizzata secondo diversi criteri. Voda e Landau (1995) [46] propongono la regolazione di un PID ispirata al principio dell'ottimo simmetrico di Kessler, mentre in Loron (1997) [24]si fa riferimento all'ottimo non simmetrico.

In Toscano (2005) [45] e Wang (2001) [50] si propone un metodo numerico che massimizza la distanza della curva di Nyquist dal punto critico -1. In Sung (2002) [40] si propone una funzione di minimizzazione atta a trovare un compromesso tra le performances del sistema e la robustezza. Infine, è possibile posizionare i poli del sistema ad anello chiuso come in Fujinaka e Omatu (2001) [15].

Un approccio alternativo ai metodi sin qui elencati , implementato anche nel presente lavoro, è in Alberto Leva , Sara Negroe Alessandro Vittorio Papadopoulos (PI/PID Autotuning with contextual model parametrisation (2010)) in cui è proposto un "Model-based autotuning contextual model parametrisation" dove la parola

"contestuale" indica la fusione della fase di parametrizzazione del modello e la fase di tuning del regolatore. Altre nozioni interessanti possono essere apprese in A.Leva (1996) [4], (1993) [5] in cui è proposta una metodologia per la selezione entro una famiglia di possibili strutture di quella che risulta maggiormente adeguata alla dinamica dello specifico processo da controllare e tale classificazione viene effettuata in linea, a partire da dati ingresso/uscita, per mezzo di una rete neurale che, grazie ad un opportuno procedimento di normalizzazione dei dati, può essere addestrata fuori linea.

Capitolo 2

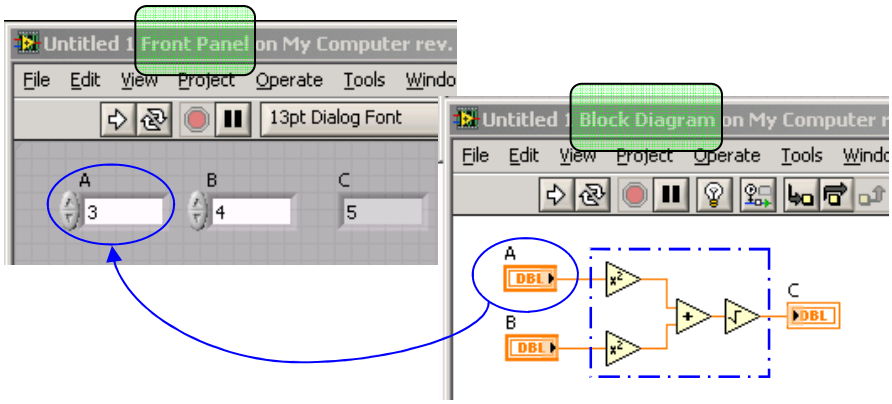
Architettura del sistema

In questo capitolo si riporta una breve descrizione dei due ambienti di sviluppo e sarà presa in esame una delle tante potenzialità che ambienti di sviluppo come LabVIEW e Modelica offrono, cioè la possibilità di integrare pezzi di codice in linguaggi come C , Fortran o Matlab, in un contesto più generale di simulazione. Essi vengono redatti ad hoc per svolgere determinate funzioni e richiamati per essere utilizzati nell'ambiente di sviluppo. Per fare ciò, entrambi gli ambienti dispongono di funzioni dedicate per l'importazione, l'utilizzo e scambio dati con librerie statiche e dinamiche esterne.

2.1 Introduzione a LabVIEW

LabVIEW (abbreviazione di Laboratory Virtual Instrumentation Engennering Workbench) è un ambiente di sviluppo integrato per il linguaggio di programmazione visuale di National Instruments. I programmi vengono chiamati virtual instruments, (VI) perché il loro aspetto e funzionamento è simile a quello di strumenti fisici; inoltre, non esistono sotto forma di testo, ma possono essere salvati solo come file binario, visualizzabile e compilabile solo nello stesso ambiente. LabVIEW possiede un insieme completo di strumenti per l'acquisizione, l'analisi, la visualizzazione e la memorizzazione dei dati, ed altri utili strumenti di debug per risolvere eventuali problemi di codice. La definizione di strutture, dati ed algoritmi avviene con icone e altri oggetti grafici, ognuno dei quali incapsula funzioni diverse, uniti da linee di collegamento (wire), in modo da formare una sorta di

diagramma di flusso. Tale linguaggio viene definito dataflow (flusso di dati), in quanto la sequenza di esecuzione è definita e rappresentata dal flusso dei dati stessi attraverso i fili monodirezionali che collegano i blocchi funzionali. Poiché i dati possono anche scorrere in parallelo attraverso blocchi e fili non consecutivi, il linguaggio realizza spontaneamente il multithreading senza bisogno di esplicita gestione da parte del programmatore. Quando si crea un nuovo VI compaiono due finestre: il pannello frontale, corrispondente all'interfaccia utente del VI e il diagramma a blocchi che include terminali, funzioni, costanti, strutture e collegamenti che trasferiscono dati tra un oggetto all'altro dello stesso. Esso contiene il codice sorgente grafico (blocchi e oggetti appaiono come terminali sotto forma di icona) grazie al quale il VI può essere eseguito.



[2.1] Finestra del Front Panel e Block Diagram in LabVIEW

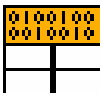
Dalla combinazione di questi elementi è possibile sviluppare programmi di notevole complessità, anche solo scrivendo poche righe di codice: bastano pochi click di mouse per avere sotto controllo tutto il programma.

I VI possono contenere al loro interno dei sub-VI, dei piccoli programmi a sè stanti, progettati anche dall'utente stesso, che permettono una maggiore astrazione nella programmazione.

2.1.1 Interfacciamento tra codice C e LabVIEW

In base all'architettura della propria applicazione, è facile impiegare la programmazione parallela di LabVIEW per eseguire due o più routine C in parallelo, senza la complessità della programmazione multithread dei linguaggi basati sul C. Per semplificare l'importazione di librerie esterne, LabVIEW include l' "Import Shared Library Wizard", il quale crea o aggiorna automaticamente una libreria wrapper di VI chiamata "VI project library" per file (.dll) in Windows, file (.framework) in Mac OS o file (.so) in Linux.

Particolarmente utili allo scopo della tesina sono state due funzioni che l'ambiente di programmazione LabVIEW mette a disposizione: Code Interface Node (CIN) e Call Library Function Node.



Code Interface Node : permette di richiamare un codice scritto in un linguaggio di programmazione testuale, come il C, direttamente da un diagramma di flusso. Si può usare per accedere ad algoritmi scritti in altri linguaggi o in piattaforme con specifiche caratteristiche hardware che LabVIEW non supporta direttamente

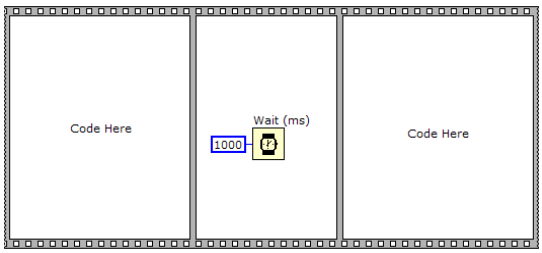
Call Library Function Node : chiama una DLL o una funzione della libreria condivisa direttamente senza usare un *Code Interface Node* (CIN). E' espandibile e supporta un grande numero di tipi di dato e chiamate convenzionali. Ci sono paia di terminali di input e output; si possono usare uno o entrambi i terminali. Ogni paio di terminali che viene aggiunto corrisponde ad uno della lista dei parametri della funzione che si vuole chiamare in ordine dall'alto in basso. Facendo click destro sul nodo e selezionando 'Configure' dal menu, compare la finestra 'Call Library Function', in cui si possono specificare il nome della libreria o il path, il nome della funzione , i parametri, le chiamate convenzionali ed il valore di ritorno del nodo.

Come già accennato, a differenza dei linguaggi di programmazione tradizionali in cui l'esecuzione è in modo sequenziale top-down, in questo caso il multithreading viene eseguito senza bisogno di esplicita gestione da parte del programmatore. In generale, un nodo viene eseguito solo quando tutti i suoi ingressi sono disponibili. Nasce, quindi, il problema della definizione di una sequenza ben precisa, quando necessario, delle chiamate a funzioni esterne. Ciò è possibile grazie ad uno specifico comando "functions >> structures >> sequence structure".

Le strutture Sequence servono per controllare il flusso di esecuzione di un programma e può contenere uno o più sub-diagrams (detti frame) che vengono eseguiti sequenzialmente. LabVIEW mette a disposizione 2 tipi di strutture:

- *Flat Sequence structure*.

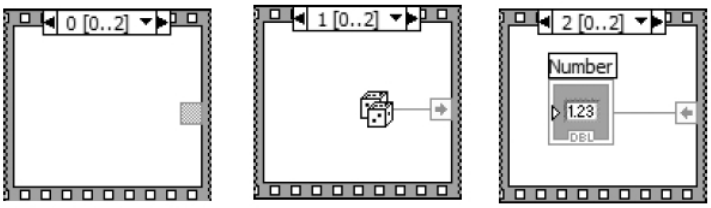
  Visualizza tutti i frames contemporaneamente ed esegue i frames in modo sequenziale, iniziando da quello più a sinistra fino a quello più a destra.



[2.2] Rappresentazione in LabVIEW di una Flat Sequence Structure

- *Stacked Sequence structure*.

  Visualizza un frame per volta. Il primo frame che viene eseguito è il frame 0; quindi, viene eseguito il frame 1 e così via, fino all'ultimo frame.



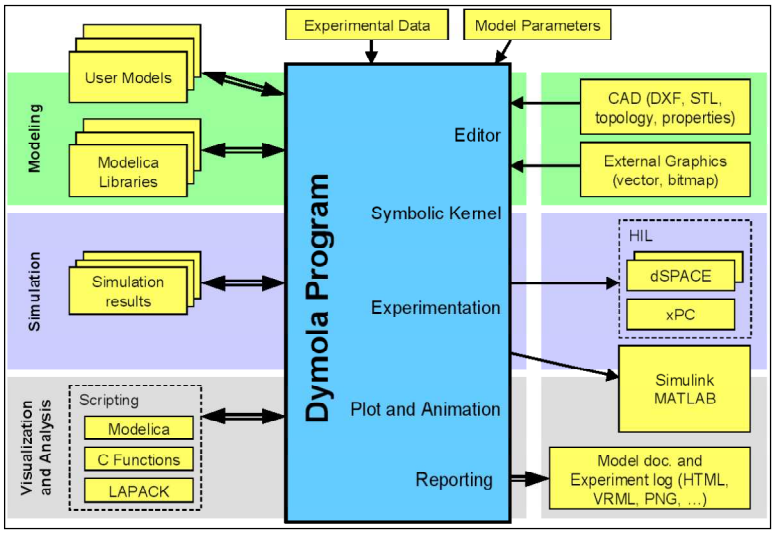
[2.3] Rappresentazione in LabVIEW di una Stacked Sequence Structure

Dopo aver visto in linea di principio quali sono i passi che Dymola esegue per realizzare la simulazione da codice Modelica, è possibile ,dunque, focalizzare l'attenzione sull'integrazione del codice C all'interno di Modelica. In particolar modo, sulla funzione 'external "C"'. Essa permette di richiamare una libreria statica (.a) ed eseguire istruzioni in C scritte al suo interno.

2.2 Introduzione a Modelica

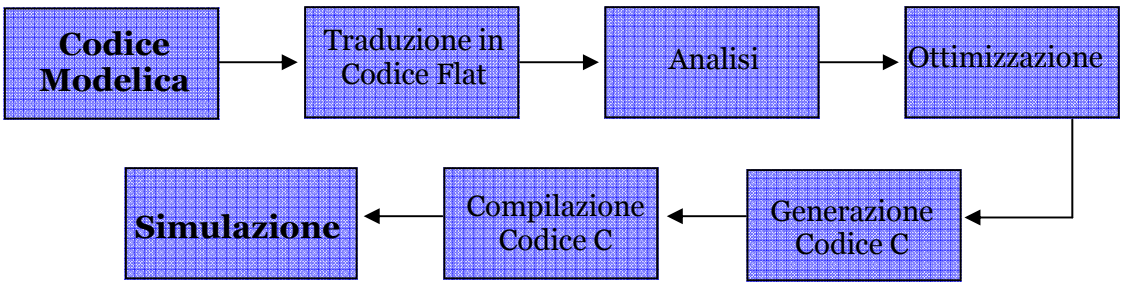
Modelica costituisce, di per sé, solo un linguaggio di descrizione dei sistemi: per poter effettivamente simulare un modello Modelica è necessario ricorrere a specifici strumenti software. Attualmente, esistono diversi strumenti che supportano Modelica: alcuni tra questi, Dymola, che verrà utilizzato come sistema di sviluppo di riferimento per questo lavoro.

Dymola (Dynamic Modeling Laboratory) è un ambiente di simulazione sviluppato per la modellizzazione di sistemi fisici eterogenei, supportando la composizione gerarchica di modelli e offrendo ampie librerie di componenti predefiniti e riutilizzabili. Tali librerie coprono svariati domini ingegneristici, come quello dei sistemi elettrici, fluidi e termici, dei sistemi multicorpo, ecc. Incorpora potenti funzioni automatiche di manipolazione simbolica delle equazioni del modello, in modo da produrre codice di simulazione altamente efficiente. L'architettura del programma è mostrata nella figura seguente:



[2.4] Architettura generale del software Dymola

Dymola contiene un traduttore complesso, basato su algoritmi per la manipolazione simbolica, che generano codice C efficiente ed ottimizzato per la simulazione. Il passaggio dal codice Modelica all'attuale codice di simulazione è complesso e composto da varie fasi. Esso è schematizzato nella figura seguente:



[2.5] Schema a blocchi in Modelica dell'interpretazione del codice sino alla simulazione

E' importante sottolineare che tale processo è completamente automatico e trasparente all'utente.

Il primo passo consiste nella traduzione nel cosiddetto "codice piatto" (per esempio un codice composto da sole equazioni, costanti, variabili e chiamate a funzioni) . Prima viene effettuato un controllo sintattico sul codice originale; successivamente, tutte le

classi sono espanse e, alla fine, tutte le equazioni esplicite del modello Modelica e dei suoi sottomodelli sono raggruppate insieme a tutte le equazioni derivate dalle dichiarazioni di connessione. Il risultato di questo primo passo è un sistema DAE (Differential Algebraic Equation). Questo può essere estremamente difficile da risolvere numericamente, specialmente in caso di indice elevato, richiedendo quindi una manipolazione simbolica ulteriore per trasformare il sistema in una forma che permetta un'integrazione facile ed efficiente.

In un secondo passo, il codice piatto è analizzato per verificare che il sistema DAE sia strutturalmente non singolare.

Il terzo passo, la fase di ottimizzazione, racchiude operazioni molto complicate: prima le equazioni corrispondenti a semplici assegnamenti, generalmente utilizzate in grande numero nelle dichiarazioni di connessione, sono semplificate ed eliminate; poi, vengono scelte le variabili di stato e, se necessario, viene ridotto l'indice del sistema, sfruttando l'algoritmo di Pantelides e il metodo delle derivate fittizie. Successivamente, viene estrapolata una matrice di adiacenza ridotta, dove le righe rappresentano tutte le equazioni e le colonne solamente le variabili algebriche. In seguito, viene effettuata l'operazione di tearing sulla matrice in questione: l'obiettivo è riformulare il sistema di equazioni acausale per trasformarlo in un insieme di assegnamenti causali ed equazioni implicite, dove le variabili che devono essere risolte sono inizializzate, sebbene non sia possibile trasformare le equazioni in assegnamenti solitamente a causa di forti non linearità. Alla fine di questa fase si ottiene un set minimale di equazioni differenziali che corrisponde agli stati selezionati ed equazioni algebriche.

Il quarto passo genera codice C, che verrà utilizzato per la simulazione attuale, collegando il sistema di equazioni al solutore numerico. In Dymola si possono scegliere molti solutori numerici, sia a passo variabile (per esempio DASSL, ODASSL, DOPRI, MEXX, LSODAR, DEABM, ecc..) che a passo fisso (per esempio Eulero implicito ed esplicito, Runge-Kutta di ordini differenti, ecc..).

Nell'ultima fase, il codice C generato precedentemente viene compilato per produrre un file eseguibile di simulazione. Sono supportati differenti compilatori C, sia open-source (come gcc), che proprietari (come Microsoft Visual C++) .

In generale, è opportuno chiedersi se la scelta di utilizzare un linguaggio di modellizzazione di alto livello come Modelica, che facilita molto il compito del

modellista, debba essere “pagata” in termini di efficienza del codice di simulazione e, quindi, in termini di complessità computazionale. La risposta, almeno per il caso di Modelica e Dymola, è negativa. Infatti, le tecniche di manipolazione simbolica delle equazioni permettono di ottenere automaticamente un codice di simulazione la cui efficienza è spesso comparabile, o addirittura migliore, rispetto a quella di codice Fortran, C o Matlab sviluppato laboriosamente “a mano”, con notevoli difficoltà di sviluppo, potenzialmente fonte di errori e difficile da documentare e da riusare.

2.2.1 Interfacciamento tra codice C e Modelica

Dopo aver introdotto le potenzialità che un linguaggio di programmazione come Modelica offre, si illustrerà come sia possibile integrare delle linee di codice scritte in C al suo interno. Sarà necessario una specifica sintassi come nel breve esempio illustrato qui di seguito:

```
function ExternalFunc
  input  Real  x;
  output Real y;
  external "C" annotation(Library="ExternalFunc.a",
                          Include="#include  \"ExternalFunc2.h\"");
end ExternalFunc1;
```

In particolare, è stata definita una funzione “ExternalFunc” con ingresso x ed uscita y, la quale implementazione è definita nella libreria ExternalFunc.a. Questa è richiamata specificandone semplicemente il nome file tra virgolette se posizionata nello stesso percorso del file padre Modelica, altrimenti se ne dovrà specificare la path.

Assieme alla libreria dovrà essere incluso il file header (.h), mediante il costrutto "Include", che conterrà i prototipi delle funzioni definite nella stessa.

Gli input definiranno i parametri che Modelica passerà alla funzione esterna e l'output il valore che gli sarà restituito. Sarà quindi necessario scrivere la funzione C nel file (.c), in modo che possa riconoscere tali parametri in ingresso, effettuare le opportune operazione e restituirne con un return il risultato.

Corredato di un particolare costrutto, il file (.c) sarà compilato per ottenerne un (.a). La funzione ExternalFunc(parameters) è così definita e pronta per l'utilizzo in un punto ben preciso del codice Modelica ogni qualvolta sarà necessario chiamarla a runtime.

2.3 Struttura degli autotuner considerati

Nel capitolo 1 è stato presentato un quadro generale dei vari metodi di implementazione per un autotuner, osservandone pregi e difetti; sono state, altresì, prese in esame le diverse politiche per il calcolo dei parametri del regolatore, necessari per ottenere un determinato comportamento desiderato ad anello chiuso. Sono stati scelti in particolare tre metodi ed in questo paragrafo se ne presenteranno struttura ed aspetti matematici.

2.3.1 Autotuner relay based

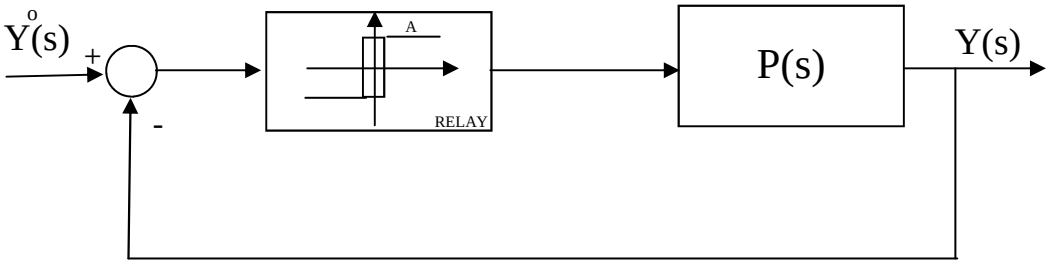
Il primo metodo implementato è del tipo experiment based, con caratteristiche determinate nel domino della frequenza. In tale dominio, esse vengono ottenute con esperimenti a relè.

Un relè è un elemento non lineare ideale, caratterizzato da un'ampiezza pari ad A. La sua caratteristica di uscita è di tipo discontinuo e può assumere solo due valori

distinti. I sistemi di controllo che fanno uso di elementi di questo tipo, come gli amplificatori a relè, vengono detti “sistemi di controllo a due posizioni” o “bang-bang”.

Si osservi, inoltre, che i sistemi in retroazione a due posizioni sono intrinsecamente auto-oscillanti : a seconda del segno dell’errore, la variabile manipolabile viene portata all’uno o all’altro degli estremi del suo campo di variazione, senza la possibilità di raggiungere una condizione di equilibrio.

Nello schema a blocchi della figura seguente, il relè innesca in controreazione delle oscillazioni permanenti di frequenza ω_{ox} .



[2.6] Schema a blocchi retroazionato con relè

L’idea con cui il metodo viene applicato, quindi, è basata sulla conoscenza del punto $\hat{P}(j\omega_{ox}) = P_{ox}e^{j\varphi_{ox}}$ sulla curva di Nyquist della funzione di trasferimento del processo $P(s)$, in cui la stessa curva interseca l’asse reale negativo.

Inserendo un ritardo variabile tra relè e processo, si possono ottenere facilmente altri punti, notando che un punto dell’anello con ritardo τ corrisponde ad un punto dell’anello senza ritardo, tramite la relazione $\frac{\hat{P}(j\omega)}{e^{-j\omega\tau}}$

Considerando una struttura di un regolatore PID ad un grado di libertà del tipo :

$$R_{PI}(s) = \left(1 + \frac{1}{sTi}\right) \qquad [18]$$

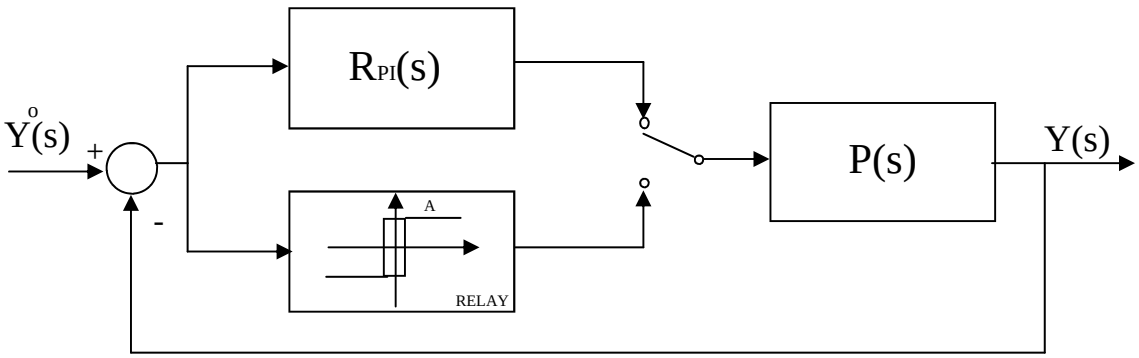
il metodo prevede tre fasi, che vengono elencate di seguito:

1) Con la richiesta di autotuning, il regolatore PI collegato al sistema viene sostituito con un blocco di isteresi .
2) Innescatesi le auto oscillazioni si calcolano:
   - frequenza di oscillazione permanente ω_{ox} con T periodo d'oscillazione:

$$\omega_{ox} = \frac{2\pi}{T}; \qquad [19]$$

   - guadagno di $P(j\omega)$, con *A* ampiezza della variabile di controllo e *D* relay swing:

$$P_{ox} = \frac{\pi^2 A}{8D} \qquad [20]$$



[2.7] Schema di controllo implementato "relay based"

3) Per determinare i parametri di tuner del regolatore PI si è scelto un punto $\bar{L}$ della risposta in frequenza in anello aperto:

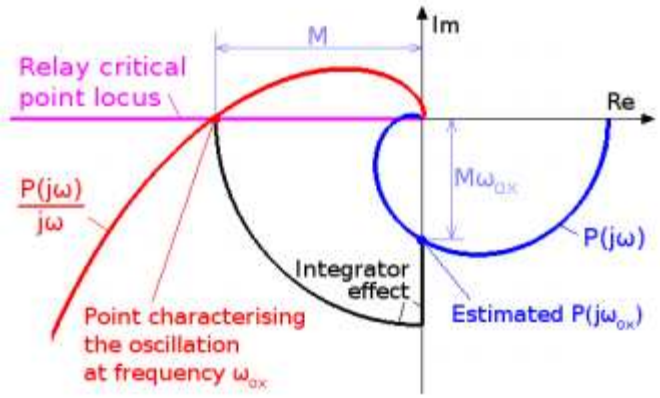
$$L(j\omega) = R(j\omega)P(j\omega) \qquad [21]$$

E, successivamente , si risolve l'equazione complessa :

$$R(j\omega_o x)P_{ox}e^{j\varphi_{ox}} = \bar{L} \qquad [22]$$

Se si sceglie il margine di fase ad anello chiuso φ_m , $L(j\omega)$ di conseguenza, interseca il cerchio unitario nella frequenza ω_{ox} , nel punto $\overline{L} = e^{j(\varphi_m - \pi)}$.

Introducendo un integratore in serie al processo, come mostrato nella figura seguente, si fa in modo che l'oscillazione sia alla frequenza in cui la fase di $\frac{P(j\omega)}{j\omega}$ è $(-\pi)$; quindi, quella di $P(j\omega)$ è $\left(-\frac{\pi}{2}\right)$.



[2.8] Luogo delle radici con aggiunta di un integratore al processo

Nella figura, la variabile M denota il guadagno della risposta in frequenza di $\frac{P(j\omega)}{j\omega}$.

Avendo imposto un vincolo sul margine di fase e, scegliendo il punto della risposta in frequenza con fase $\left(-\frac{\pi}{2}\right)$, è possibile calcolare la costante di integrazione e la costante proporzionale del regolatore PI, come segue:

$$K = \frac{\tan(\varphi_m)}{P_{ox}\sqrt{1+\tan^2(\varphi_m)}} \qquad [23]$$

$$T_i = \frac{\tan(\varphi_m)}{\omega_{ox}} \qquad [24]$$

Si fa notare che in letteratura esistono altre varianti per il calcolo di K e T_i.

2.3.2 Autotuner basato su IMC con tuning contestuale

Il metodo utilizzato fa parte della famiglia dei "model-based approaches" e permette di ottenere in modo esplicito il modello del processo che potrà essere utilizzato per il tuning del regolatore.

L'approccio in questo lavoro è di tipo "contestuale". Con tale aggettivo si indica la fusione dei due passi da seguire per ottenere il risultato cercato in un unico sistema di equazioni da risolvere. Il primo riguarda la ricerca dei parametri per determinare una funzione di trasferimento di una struttura FOPDT e il secondo la ricerca dei parametri necessari per effettuare il tuning.

In particolare, per il secondo passo è stato implementato un metodo della famiglia Internal Model Control IMC-λ, dove con λ si intende la costante di tempo desiderata in anello chiuso.

Il metodo fa riferimento ad una struttura ISA- PID con $b = 1$ e $c = 0$:

$$R_{PID}(s) = K\left(1 + \frac{1}{sT_i} + \frac{sT_d}{1 + sT_d / N}\right) \qquad [25]$$

ed un modello avente struttura FOPDT :

$$M(s) = \frac{\mu}{1 + sT} e^{-sL} \qquad [26]$$

in cui :

$$T_i = T + \frac{L^2}{2(L + \lambda)} \qquad [27]$$

$$K = \frac{T_i}{\mu(L+\lambda)} \qquad [28]$$

$$N = \frac{T(L+\lambda)}{\lambda T_i} - 1 \qquad [29]$$

$$T_d = \frac{\lambda L N}{2(L+\lambda)} \qquad [30]$$

Mediante relay experiment si è determinato un punto della curva di Nyquist $P(\overline{\omega}) = Ae^{j\varphi_p}$ e si sono poste le seguenti condizioni :

- il punto sia anche appartenente alla risposta in frequenza del modello $M(\overline{\omega}) = P(\overline{\omega})$;

- la frequenza di taglio nominale ω_{cn} sia uguale a $\overline{\omega}$, con :

$$\omega_{cn} = \frac{1}{(L+\lambda)} = \overline{\omega} \qquad [31]$$

Tali condizioni portano a risolvere il seguente sistema di sette equazioni a otto incognite:

$$\begin{cases} T_i = T + \dfrac{L^2}{2(L+\lambda)} \\ K = \dfrac{T_i}{\mu(L+\lambda)} \\ N = \dfrac{T(L+\lambda)}{\lambda T_i} - 1 \\ T_d = \dfrac{\lambda LN}{2(L+\lambda)} \\ A_p = \dfrac{\mu}{\sqrt{1+(\overline{\omega}L)^2}} \\ \varphi_p = -arctg\ (\overline{\omega}L) - \overline{\omega}L \\ \overline{\omega} = \dfrac{1}{L+\lambda} \end{cases}$$

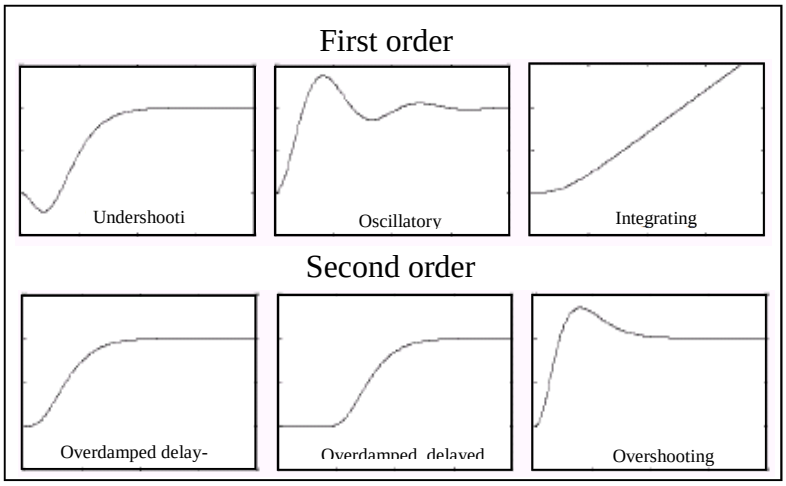
Fissando il parametro λ (interpretabile come costante di tempo ad anello chiuso) è possibile risolvere l'intero sistema nella forma:

$$\begin{cases} T_i = T_i = T + \dfrac{L^2}{2(L+\lambda)} \\ K = \dfrac{T_i}{\mu(L+\lambda)} \\ \mu = A_p\sqrt{1+(\overline{\omega}L)^2} \\ L = \dfrac{1}{\omega} - \lambda \\ N = \dfrac{T(L+\lambda)}{\lambda T_i} - 1 \\ T_d = \dfrac{\lambda LN}{2(L+\lambda)} \end{cases}$$

avente sette equazioni e sette incognite. Si determinano , quindi, n_R variabili per il regolatore (T_i, T_d, K, N) ed n_M variabili per il modello (μ, T, L) .

2.3.3 Autotuner basato su IMC e metodo delle aree

Il terzo metodo implementato è ancora una volta un model based, questa volta con un approccio nel dominio del tempo. Per identificare il modello del processo con i relativi parametri, è stato utilizzato il metodo delle aree. Si tratta di una tecnica di identificazione in anello aperto che permette di ricavare, tramite semplici prove sul processo, dei modelli approssimati di semplice struttura, utili ai fini della sintesi del controllore. La procedura consiste nel portare il sistema da controllare in uno stato di equilibrio e, successivamente, valutare la sua risposta ad uno scalino in ingresso. Dalla forma di questa risposta se ne può dedurre un modello matematico.
Molti processi sono caratterizzati da una risposta allo scalino stabile e non oscillante ad eccezione, eventualmente, nella fase iniziale del transitorio.



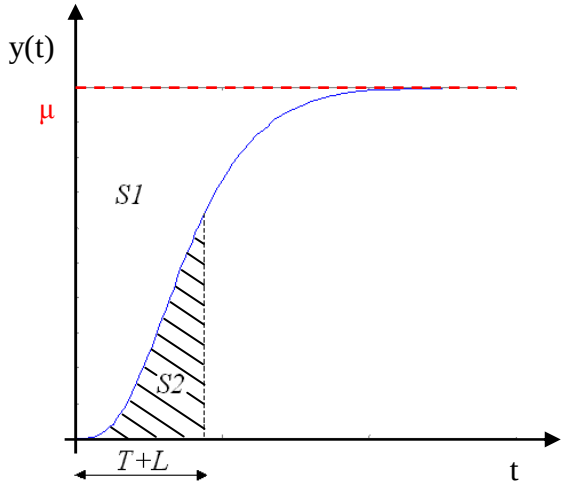
[2.9] Tipiche risposte a gradino per un sistema di primo e secondo ordine

Per tali sistemi è possibile ricavare un modello approssimato, valido in bassa frequenza, descritto da una funzione di trasferimento caratterizzata da un solo polo reale ed un ritardo, cioè :

$$M(s) = \frac{\mu}{1+sT} e^{-\tau s} \qquad [32]$$

Il parametro τ è comunemente chiamato “ritardo equivalente”, mentre T è definita “costante di tempo equivalente”.

Il metodo delle aree è un metodo per la determinazione dei parametri μ, τ e T a partire dalla risposta sperimentale del processo .



[2.10] Rappresentazione delle aree S1ed S2 per il metodo delle aree

Considerato in ingresso al sistema da stimare uno scalino di ampiezza $\bar{\lambda}$, il guadagno del sistema si ricava direttamente nel modo che segue :

$$\mu = \frac{\bar{y}}{\bar{\lambda}} \qquad [33]$$

dove $\bar{y}$ è il valore di regime dell'uscita. Successivamente è necessario valutare l'area S1, compresa fra la risposta del sistema e l'asintoto della curva, che è data da

$$S_1 = \mu\bar{\lambda}\tau + \int_0^\infty \mu\bar{\lambda}e^{-t/T}dt = \mu\bar{\lambda}(\tau + T) = \bar{y}(\tau + T) \qquad [34]$$

Si determina quindi il valore di $\tau + T$. Considerando ora l'area S2, compresa tra la curva, l'asse dei tempi e una retta verticale passante per il punto di ascissa $\tau + T$, si ha:

$$S_2 = \int_0^T \mu\bar{\lambda}(1 - e^{-t/T})dt = \frac{\mu\bar{\lambda}T}{e} = \frac{\bar{y}T}{e} \qquad [35]$$

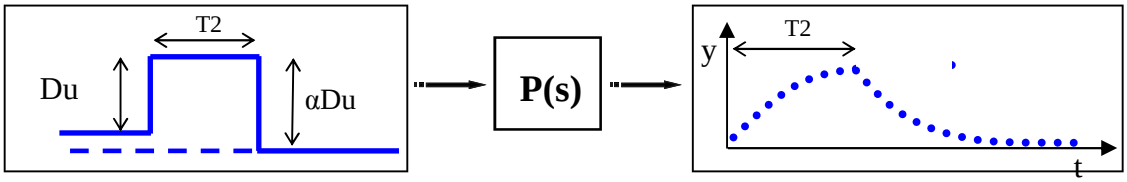
da cui si ricava :

$$T = \frac{eS_2}{\bar{y}} \qquad [36]$$

E, quindi, τ si calcolerà come :

$$\tau = \frac{S_1 - \bar{y}T}{\bar{y}} \qquad [37]$$

Il metodo sin qui esaminato è come si presenta in letteratura. Basandosi su tali equazioni, è stata implementata la programmazione in codice C. In particolare, però, è stata utilizzata un'accortezza realizzativa. Infatti, in realtà, se il processo viene stimolato in anello aperto (con ingresso a gradino), difficilmente si è a conoscenza della durata della risposta. E' per questo motivo che il metodo delle aree è stato applicato con una piccola variante: l'esperimento è stato eseguito con un doppio gradino in ingresso.

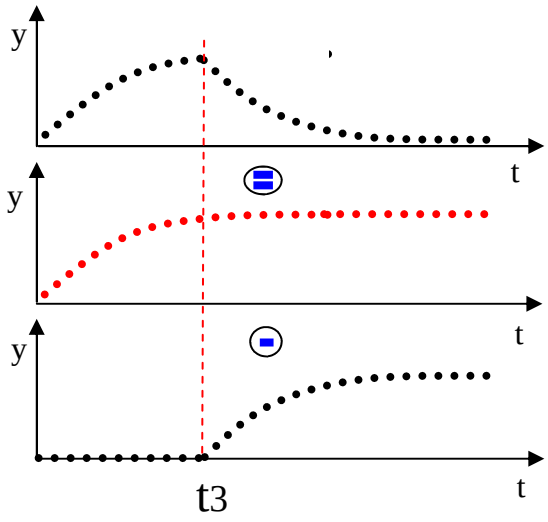


[2.11] Rappresentazione dell'ingresso e dell'uscita al processo P(s)

Come si vede nella figura [2.11], al comando di autotuning, il processo viene stimolato con un gradino di ampiezza positiva Du per un tempo T_2. Successivamente, con un secondo gradino di ampiezza negativa αDu . Il parametro α , in percentuale, deve assicurare che l'ampiezza del primo gradino sia superiore a quella del secondo.

Così facendo, la risposta registrata sarà la risultante della differenza di due risposte.

$$y_{reg} = \begin{cases} rsu(t).Du & t < t_3 \\ rsu(t).Du - \alpha Du \cdot rsu(t - t_3) & t > t_3 \end{cases}$$

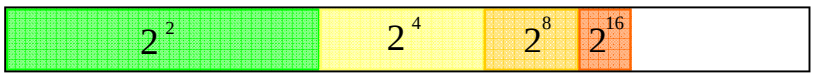


[2.12] Rappresentazione delle operazioni eseguite con la risposta in uscita

Effettuando una semplice operazione con i dati acquisiti, si ricava la risposta a gradino su cui applicare il metodo sopracitato.

Particolare attenzione si è data all'acquisizione dei punti appartenenti alla curva in risposta al sistema. Si sono utilizzati due vettori, destinati a contenere tutti i dati in ascissa (tempo t) e ordinata (uscite y) appartenenti alla curva. Ipotizzando uno spazio di archiviazione limitato, si è adottato un espediente che consente di registrare i dati più significativi in un vettore.

Iniziato l'autotuning, il primo passo consiste nel registrare i dati nei due vettori ed una volta riempiti, si manterrà in memoria solo un dato ogni due, liberando la metà dello spazio restante in essi. Nel secondo passo, quando il vettore sarà nuovamente pieno, manterrà in memoria solo un dato ogni quattro. E così via, con passo di registrazione 2^k.



[2.13] Rappresentazione del vettore dedicato alla raccolta dei dati campionati

La sub-rutine terminerà quando la risposta a gradino negativo supererà il set-point di partenza.

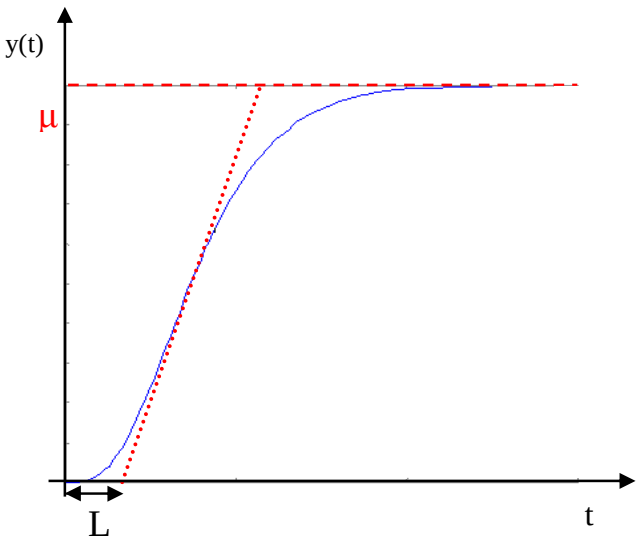
Si fa notare che i parametri Du e T_2 sono fondamentali per un corretto funzionamento del metodo. Un ulteriore approfondimento potrebbe renderli dinamici al variare della risposta del processo, il che tuttavia esula da questo lavoro.

Come si è visto, questa tecnica consente di determinare i parametri della funzione di trasferimento di un sistema FOPDT (First Order Plus Delay Time); inoltre, risulta essere anche piuttosto robusto rispetto al rumore in quanto, lavorando con gli integrali, il rumore a media nulla scompare.

Per quanto riguarda invece l'identificazione di un modello del tipo SOPDT (Second Order Plus Delay Time), il procedimento è differente : si utilizza il metodo della tangente. Il modello da identificare, in questo caso, ha la forma:

$$M(s) = \mu \frac{e^{-sL}}{(1+sT_1)(1+sT_2)} \qquad [38]$$

Il guadagno μ si ricava come nel caso precedente e L può essere calcolato osservando la forma della risposta a scalino.



[2.14] Illustrazione del metodo della tangente

La tangente alla curva è tracciata passante per il punto di massima pendenza. I parametri T_1 e T_2, vengono calcolati adattando un modello del tipo

$$M(s) = \mu\left[1 + \frac{T_2 e^{\frac{t-L}{T_2}} - T_1 e^{\frac{t-L}{T_1}}}{T_1 - T_2}\right] \qquad \text{con} \quad T_1 > T_2 \qquad [39]$$

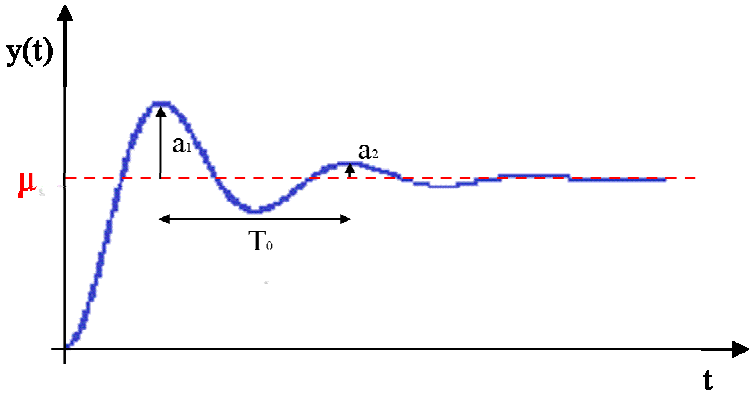
alla curva della nostra risposta a gradino. Generalmente, i punti sono al 33% ed al 67% del valore finale della risposta.

Facendo un po' d'attenzione è possibile identificare anche modelli del secondo ordine con risposta a gradino oscillatoria. In questo caso, scriviamo il modello come segue:

$$M(s) = \frac{\mu}{1 + 2\frac{\zeta}{\omega_n} + \frac{s^2}{\omega_n^{\,2}}} \qquad [40]$$

Anche in questo caso il guadagno μ si calcola come visto nel primo caso (sistemi FOPDT).

Gli altri parametri si ricavano osservando la risposta a gradino oscillante :



[2.15] Illustrazione dei parametri per la determinazione del modello di secondo ordine

Dove T_o è il periodo di oscillazione, a_1 e a_2 l'ampiezza dei primi due picchi della risposta $y(t)$. Infine, è possibile calcolare i parametri rimanenti in questo modo:

$$\xi = \frac{1}{\sqrt{1+\left[\frac{2\pi}{\log(a_2/a_1)}\right]^2}} \quad [41]$$

$$\omega_n = \frac{2\pi}{T_o\sqrt{1-\xi^2}} \quad [42]$$

In conclusione, modificando ed ampliando leggermente il codice già realizzato, sarà possibile estendere a più processi le potenzialità dell'autotuner.

Capitolo 3

Descrizione del codice

Scopo di questo capitolo è descrivere dapprima le strutture dati e le funzioni implementate nei file sorgenti C; successivamente, si riporterà una breve guida sull'utilizzo delle librerie rilasciate.

Essa riguarderà i due ambienti di sviluppo LabVIEW e Modelica. In particolar modo per Modelica, la guida è riferita all'utilizzo di Dymola versione 6.1 del 2007 e per labVIEW è riferita alla versione 11.0 del 2011. Saranno elencati tutti i file ed il loro utilizzo.

3.1 Le strutture dati e le funzioni fondamentali

Nel presente lavoro di tesina tutti i files sorgenti sono stati scritti in C e compilati successivamente con Dev-C++ per uso stand-alone, in Dymola e LabVIEW, per l'utilizzo delle librerie create. E' stato utilizzato, inoltre, il programma SciLab da sostegno a Dev-C++ per poterne graficare il risultato della simulazione con dati raccolti in un file.

L'approccio utilizzato è stato quello proposto in M. Maggio, A. Leva [6]. Sono state create delle funzioni e strutture per realizzare il sistema dinamico in esame e, successivamente, un piccolo script in Scilab per graficarne i risultati.

In particolare, si sono create due strutture dati. La prima, chiamata "ISAPIDinc_DATA", che contiene tutte le variabili e costanti di tipo int , float, o char, necessarie ad implementare un controllore ISA-PID (Astrom et al.2006) nella forma :

$$CS(s) = K\left(bSP(s) - PV(s) + \frac{1}{sT_i}\left(SP(s) - PV(s)\right) + \frac{sT_d}{1 + sT_d / N}\left(cSP(s) - PV(s)\right)\right) \quad [43]$$

Dove SP,PV e CS sono rispettivamente il set point, la variabile controllata del processo ed il segnale di controllo. Si fa notare che l'acronimo "inc" è significativo del fatto che la legge di controllo è stata calcolata nel discreto nella forma incrementale; ovvero si calcola, ad ogni passo di campionamento, la variazione del controllo.

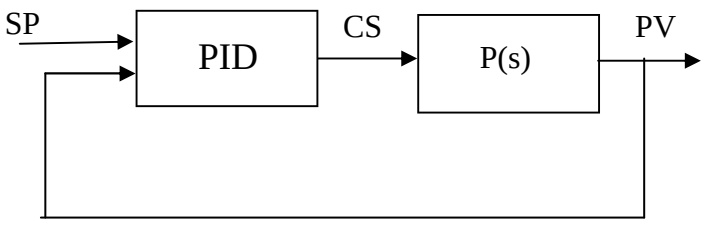
La seconda, chiamata "TF1P0Z_DATA" contiene, invece, variabili di tipo float, necessarie ad implementare una tipica funzione di trasferimento nella forma :

$$P(s) = \frac{\mu}{1 + sT} \quad [44]$$

I parametri di entrambe le strutture saranno inizializzati da due apposite funzioni chiamate "isapidinc_coldinit" e "tf1p0z_coldinit", rispettivamente per il controllore ISA-PID e per la funzione di trasferimento P(s).

Tali funzioni richiedono in ingresso alcuni parametri corrispondenti ad alcune variabili all'interno della struttura dati corrispondente. (Si veda il paragrafo successivo per studiarne l'utilizzo).

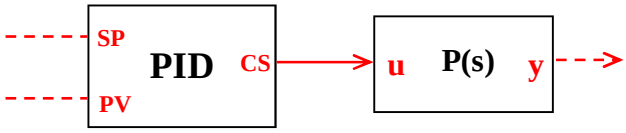
Scopo dell'organizzazione del codice C nella maniera presentata è quello di richiamare e creare il concetto del classico schema a blocchi.



[3.1] Schema a blocchi realizzato per il controllo

Sarà così facile effettuare il collegamento tra un blocco ed un altro, mediante l'imposizione di uguaglianza delle variabili input e output dei due blocchi da collegare, accessibili tramite puntatore. Ad esempio, per collegare il blocco PID ed il blocco P(s), sarà necessario solo imporre la seguente uguaglianza:

$$P(s).u = PID.CS$$



[3.2] Rappresentazione del "wirinig" di due blocchi in codice C

nella quale cui PID e P(s) sono strutture del tipo "ISAPIDinc_DATA" e "TF1P0Z_DATA", inizializzate con funzioni "isapidinc_coldinit" e "tf1p0z_coldinit" ed aventi come input SP,PV, u ed output CS e y.

La funzione che calcolerà il valore dell'uscita y dato l'ingresso u, è stata implementata e chiamata "tf1p0z".

Essa riceve in ingresso il puntatore alla struttura di tipo TF1P0Z_DATA, da cui preleva i valori necessari ad eseguire i calcoli (compreso l'ingresso u) e ne restituisce y.

Il core principale del regolatore , con autotuning implementato con tre diversi metodi, risiede nella funzione denominata "isapidinc".

Essa riceve in ingresso una serie di parametri illustrati nel capitolo successivo (compresi SP e PV) e ne restituisce con un return il frutto dell'elaborazione "CS" , variabile di controllo.

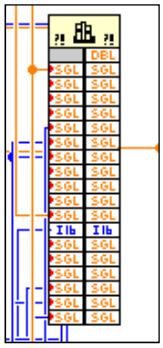
All'interno di tale codice è previsto l'utilizzo in automatico di un anti windup , necessario a causa di sovraelongazioni prodotte dalla concomitanza di una saturazione della variabile di controllo e di un'eccessiva azione integrale. Infatti, quando l'attuatore satura, limita l'entità dell'azione di controllo con il risultato che l'errore continua ad essere consistente. La presenza dell'errore per lungo tempo finisce con il favorire la saturazione dell'attuatore, perché più passa il tempo e più l'azione integrale si fa energica, ma l'attuatore è già saturato. Si verifica, in altre parole, una forte tendenza alla

saturazione degli attuatori in saturazione e di conseguenza, un significativo calo delle prestazioni del sistema. In presenza di saturazione della variabile di controllo, il termine integrale può raggiungere valori molto elevati.

3.2 Utilizzo della libreria con LabVIEW

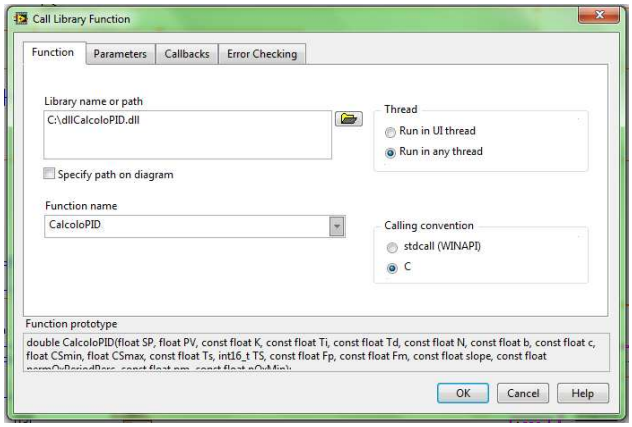
I files rilasciati con il presente lavoro, che potranno essere utilizzati per eseguire le simulazione sono:

dllCalcoloPID.dll : file libreria dinamica da incorporare nel programma Autotuning_PID_1 .vi di labVIEW. Nello schema a blocchi è presente la parte relativa al "Call Library Function" (si rimanda al capitolo 2) che dovrà essere configurata per un corretto utilizzo della libreria.



[3.3]Blocchetto della Call Library Function in LabVIEW

Con un doppio click sul blocchetto è possibile accedere alla finestra di configurazione. Nella prima scheda bisognerà specificare la Path, in cui risiede il file dllCalcoloPID.dll e il nome della funzione da utilizzare all'interno della libreria (in questo caso "CalcoloPID"):



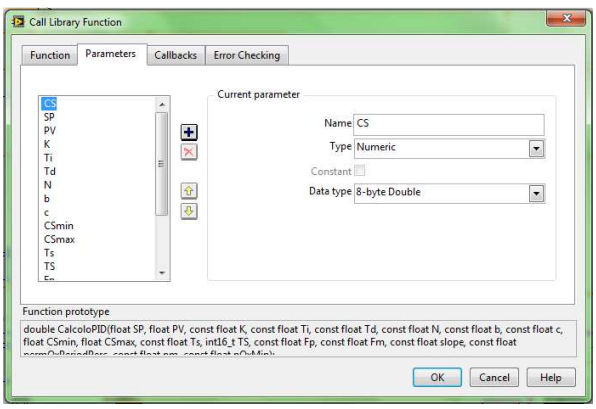
[3.4] Finestra di configurazione (scheda "Function") della Call Library Function in LabVIEW

Nella seconda scheda dovranno essere indicati i parametri da passare alla funzione. Essendo strettamente legati alla struttura "CalcoloPID", dovranno essere inseriti nell'ordine seguente:

Per la libreria *dllCalcoloPID.dll* : CS , SP, PV, K, Ti, Td, N, b, c, CSmin, CSmax, Ts, TS, slope, permOxPeriodPerc, pm, nOxMin

Per la libreria *dllCalcoloPID2.dll* : CS , SP, PV, K, Ti, Td, N, b, c, CSmin, CSmax, Ts, TS, lambda, slope, permOxPeriodPerc, pm, nOxMin

Per la libreria *dllCalcoloPID3.dll* : CS , SP, PV, K, Ti, Td, N, b, c, CSmin, CSmax, Ts, TS, lambda



[3.5] Finestra di configurazione (scheda "Parameters") della Call Library Function in LabVIEW

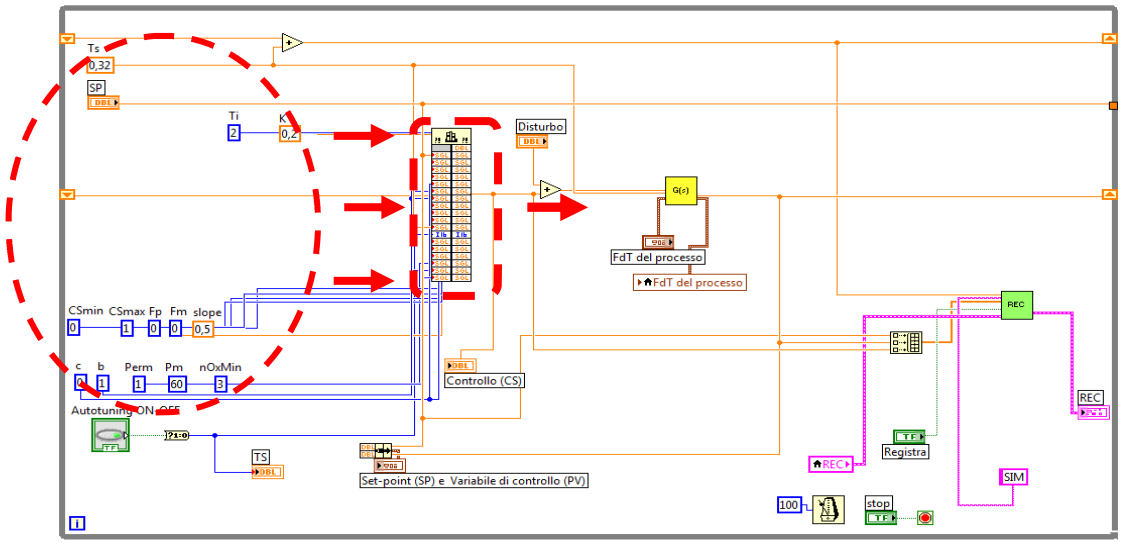
Tutti i parametri potranno essere dichiarati :

Type : Numeric

Data type : - 8 byte-double (CS, PV)

- 4 byte-single (restanti)

Autotuning_PID_1 .vi : Programma virtual instruments, che contiene lo schema a blocchi e consolle con interfaccia grafica per l'utilizzatore. Il primo si presenta come nella figura seguente:



[3.6] Block Diagram del programma in LabVIEW ed indicazione dei parametri principali

Nella parte evidenziata sono presenti i parametri necessari alla simulazione, che saranno utilizzati come ingresso alla funzione "CalcoloPID" ,richiamata dalla libreria dinamica "dllCalcoloPID.dll".

In particolare, in tutti e tre metodi implementati verrà richiesto di impostare :

- Ko : Valore iniziale della costante proporzionale ;
- Ti : Valore iniziale della costante di integrazione ;
- Td : Valore iniziale della costante di derivazione ;
- N , b,c : coefficiente della funzione di trasferimento del regolatore ;
- Csmin : Valore minimo che può assumere la variabile di controllo ;
- Csmax : Valore massimo che può assumere la variabile di controllo ;

- Ts : Periodo di campionamento.

Per il primo e secondo metodo verranno richiesti inoltre :

- Slope : percentuale inclinazione delle oscillazioni ;
- permOxPeriodPerc : percentuale sul controllo del periodo delle oscillazioni ;
- pm : margine di fase richiesto per l'autotuning ;
- nOxMi : numero minimo oscillazioni per il calcolo dei parametri.

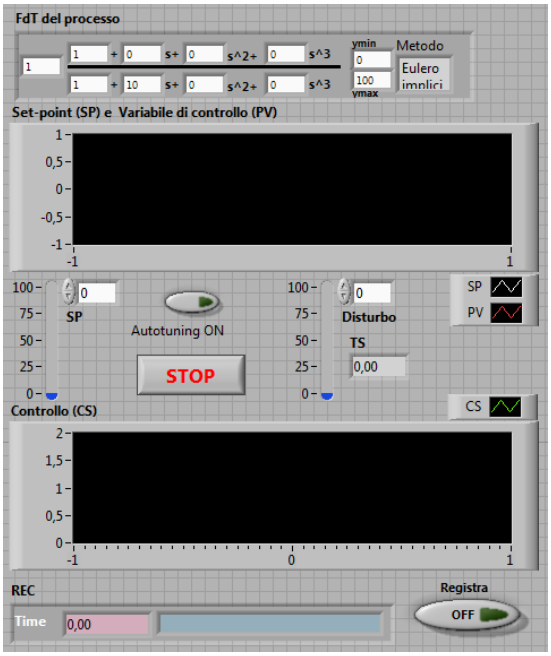
Nel secondo e nel terzo, invece :

- lambda : costante di tempo ad anello chiuso desiderata.

Il comando TS, che avvia l'autotuning , può assumere solo due valori 1/0. Esso è stato implementato mediante uno switch presente sul pannello frontale.

Una particolare attenzione va fatta alla scelta del periodo di campionamento. Nelle applicazioni di controllo di processi industriali, nei quali le variabili da controllare sono portata, temperatura, pressione, livello, ecc., sono sufficienti periodi di campionamento dell'ordine di frazioni di secondo o secondi (temperatura), o decine di millisecondi (pressione); mentre, per il controllo di una tensione o corrente elettrica (ad esempio negli azionamenti elettrici), sono necessari tempi di campionamento inferiori al millisecondo.

Il Pannello frontale, nel suo complesso, appare come nella figura seguente:



[3.7] Pannello frontale del programma realizzato in LabVIEW

Nella parte superiore è presente il pannello che permette di impostare la funzione di trasferimento P(s) del tipo :

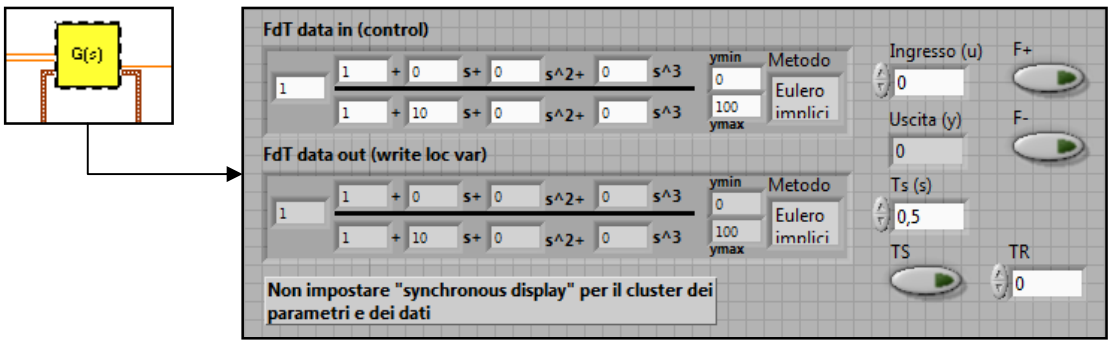
$$P(s) = \mu \frac{a + bs + cs^2 + ds^3}{a' + b's + c's^2 + d's^3} \qquad [45]$$

Si sceglieranno, quindi, i coefficienti del numeratore, denominatore e μ. E' data, inoltre, la possibilità di scegliere il metodo di integrazione.

Nella parte centrale è possibile impostare sia il set-point SP, sia il disturbo sulla variabile controllata CS, inserendone il valore numerico nel campo o scorrendo l'indicatore verso l'alto.

Nella parte inferiore è possibile avviare la registrazione su file mediante uno switch. Saranno visualizzati il tempo di registrazione ed il nome del file *.txt*, creato nella stessa cartella di esecuzione del programma.

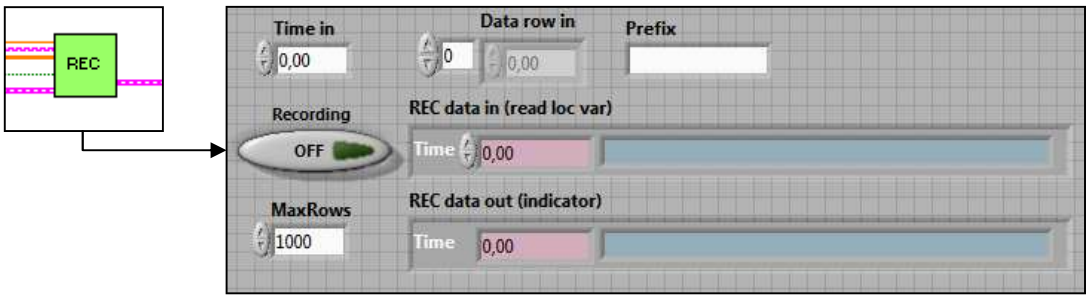
FunzioneDiTrasferimento.vi : Programma virtual instruments utilizzato come sub-VI e richiamato dal file principale Autotuning_PID_1.vi. Implementa un blocchetto SISO contenente un applicazione scritta in codice C che fornisce l'uscita di una funzione di trasferimento di ordine generico. In questo caso, riceve in ingresso la variabile di controllo CS ed tempo di campionamento Ts e produce come output la variabile controllata PV.



[3.8] Pannello frontale del sub-VI FunzioneDiTrasferimento in LabVIEW

Per conoscere nel dettaglio le potenzialità di tale VI con i suoi numerosi I/O, si faccia riferimento alla documentazione in AutomationLabVIsuite [6] .

RegistratoreSuDisco.vi : Come il precedente, è richiamato dal VI principale. Esso è stato utilizzato per raccogliere alcuni dati elaborati. In particolar modo, il tempo t, SP set point, PV variabile controllata e CS variabile di controllo.



[3.9] Pannello frontale del sub-VI RegistratoreSuDisco in LabVIEW

Essi verranno memorizzati in un file .txt nella stessa cartella, in cui verrà eseguito il programma Autotuning_PID_1 .vi. Anche tale sub-VI è presente in AutomationLabVIsuite [6]; si rimanda alla documentazione allegata per maggiori informazioni.

Plot_from_LabVIEW.sce : Programma realizzato in Scilab versione 5.3.3 del 2011. Contiene uno script utile a graficare i dati raccolti da RegistratoreSuDisco.vi in un file di testo .txt, creato in automatico e denominato di volta in volta (es. SIM_d09m11y2011_h15m45s00c93.txt). In particolare, conterrà quattro colonne di dati corrispondenti da sinistra verso destra al tempo t, SP set point, PV variabile controllata e CS variabile di controllo.
La seconda riga è nella forma :

```
M  = fscanfMat("SIM.txt","%lg");
```

In cui la stringa Sim.txt è il prototipo del nome file appena citato, che dovrà essere sostituito con il corrispondente nome esatto contenente i dati di cui si vorrà produrre un plot.

3.3 Utilizzo della libreria con Modelica

I file rilasciati, riguardanti il lavoro in ambiente Modelica eseguito, sono essenzialmente quattro:

CalcoloPID.h : File header che contiene i prototipi delle funzioni implementate nella libreria libCalcoloPID.a

libCalcoloPID.a : File libreria da richiamare nel file del progetto di modelica salvato come Modelica.mo

Plot_from_Modelica.sce : Come nel caso di LabVIEW, il file contiene un piccolo script (si veda paragrafo precedente) per plottare i dati raccolti dalla funzione “print” del package di Modelica, nel file “data.txt”. Qui sarà necessario un accorgimento in più; infatti, il file .txt conterrà la virgola, come separatore delle cifre intere da quelle decimali, che non è purtroppo accettata dalla funzione “fscanfMat” di Scilab. Tutte le virgole dovranno essere sostituite da un punto in un qualsiasi editor di testo.

Modelica.mo : E’ il file principale che contiene il package completo per la simulazione. In esso sono presenti tre funzioni e due modelli.
Le funzioni sono:

- *CalcoloPID* : E’ la funzione che chiama a runtime la funzione “CalcoloPID” presente nella libreria statica libCalcoloPID.a definendo i parametri di input che gli saranno passati. Contiene la sintassi che ne permette il collegamento con tale

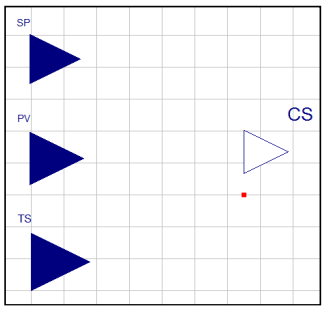
libreria (Vedi capitolo 2). L'output sarà il valore della variabile di controllo CS restituito dalla libreria.

- *Create file* : E' la funzione che richiama a runtime la funzione "create file", presente nella libreria statica libCalcoloPID.a. Il suo utilizzo ha scopo di creare e inizializzare i file che saranno utilizzati per salvare il valore delle variabili del passo precedente di calcolo.

- *Removefile* : E' la funzione che richiama a runtime la funzione "remove file" presente nella libreria statica libCalcoloPID.a. Il suo utilizzo ha scopo di eliminare i file creati dalla funzione precedente.

I modelli :

- *ATPIdigital* : E' il core del regolatore. Strutturalmente ha tre ingressi Set-point SP, variabile controllata PV e segnale di abilitazione Autotuning. A tali ingressi dovranno essere collegati opportuni segnali e l'ingresso PV sarà utilizzato per realizzare la retroazione.



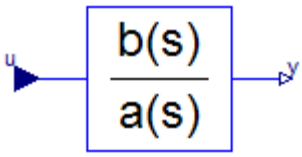
[3.10] Schema a blocchi di ATPIdigital in Modelica

Al suo interno è richiamata la funzione "print" (contenuta nel package "Utilities" di modelica) utilizzata per catturare nel file "data.txt", ad ogni passo di campionamento,

le variabili time, SP, PV, CS. Tali dati saranno raccolti nel file, organizzati in quattro colonne.

```
Modelica.Utilities.Streams.print(String(time)+" "+String(SP)
                                 +" "+String(PV)+" "+String(CS),"data.txt");
```

- *Example* : Contiene un piccolo esempio per la simulazione e la prova della libreria. Oltre che al set-point, dovrà essere impostata la funzione di trasferimento del processo:



[3.11] Blocchetto della FDT (funzione di trasferimento) in Dymola

$$y(s) = \frac{b[1]*s^{[nb-1]} + b[2]*s^{[nb-2]} + \ldots + b[nb]}{a[1]*s^{[na-1]} + a[2]*s^{[na-2]} + \ldots + a[na]} * u(s)$$

Tale modello è presente nella libreria di modelica ed è denominato “TransferFunction”.

Capitolo 4

Esempi di applicazione

Dopo aver presentato l'argomento dal punto di vista teorico e da quello matematico, evidenziando le formule utilizzate per implementare i metodi scelti di autotuning, si procede nel presentare alcune simulazioni effettuate.

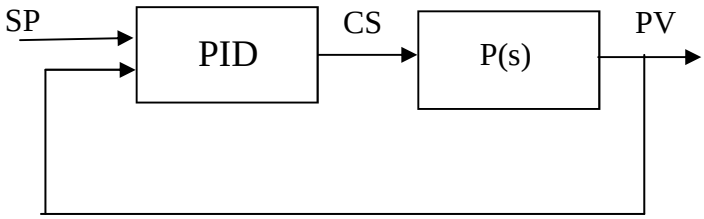
E' stato possibile testare il codice implementato per individuarne criticità e robustezza. Nel primo paragrafo, sono state effettuate alcune prove con il file sorgente ed il corrispettivo header in ambiente Dev-C++. Nel secondo, in ambiente LabVIEW e nel terzo, in ambiente Modelica.

4.1 Uso stand-alone

La prima prova presentata considera un semplice processo avente funzione di trasferimento:

$$P(s) = \frac{1}{(1+s)^2}$$

Utilizzando i files messi a disposizione per il primo metodo (Characteristics based time frequency : Relay based), è stato possibile simulare lo schema reatrozionato come nella figura :

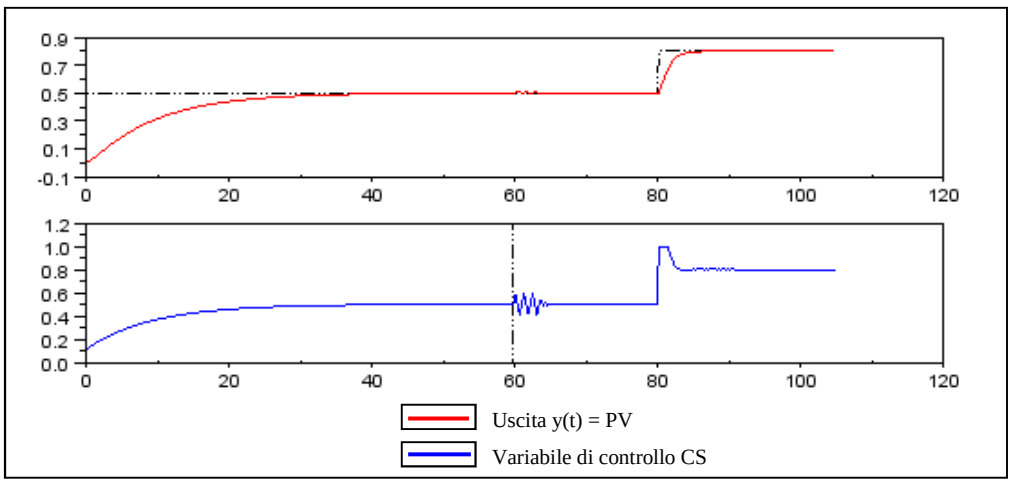


[4.1] Schema a blocchi realizzato per il controllo

Impostando i valori come segue :

- margine di fase richiesto pm : 80
- slope : 0.3
- periodo di campionamento Ts : 0.3

i risultati della simulazione risultano essere :



[4.2] Example1.1 stand-alone : pm=80 ; slope=0.3; Ts=0.3

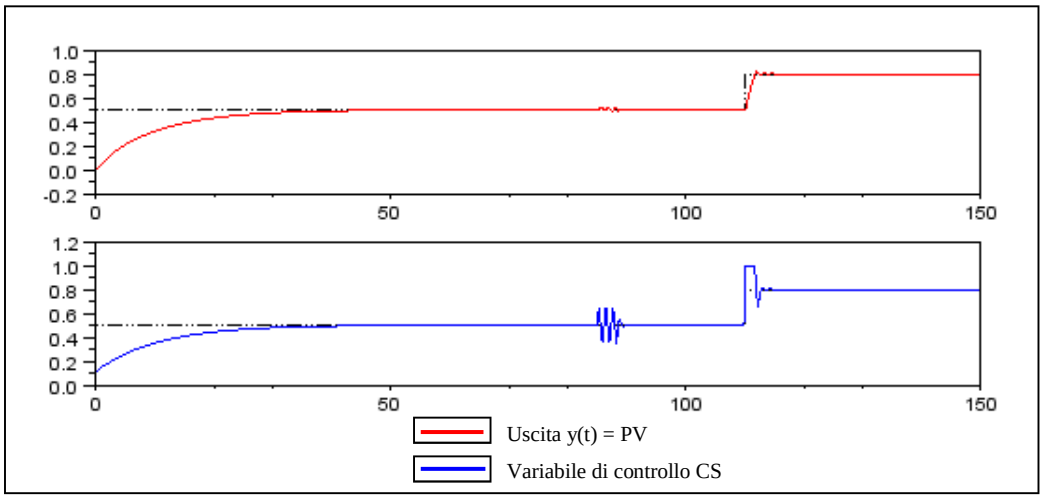
Si nota che partendo da una configurazione non adeguata dei valori K, T_i e T_d, dopo il comando di tuning (che avviene al 60° secondo), il sistema reagisce in maniera più veloce alla variazione del set-point (avvenuta all' 80° secondo). Infatti, nei primi istanti sono necessari almeno 35 secondi perché la variabile controllata PV vada a regime, successivamente ne basteranno solo 7.

La stessa simulazione, considerando lo stesso processo P(s) è stata eseguita con gli altri due metodi implementati.

In particolare, per il secondo (Model based contextual model parametrisation IMC) una volta scelto il valore dei parametri :

- margine di fase richiesto pm : 80
- slope : 0.3
- periodo di campionamento Ts : 0.3
- costante ad anello chiuso λ : 0.1

il risultato della simulazione è il seguente :



[4.3] Example1.2 stand-alone : pm=80 ; slope=0.3; Ts=0.3; lambda=0.1

Anche qui si nota il netto miglioramento implicito dei parametri del regolatore PID, dopo aver effettuato l'autotuning all' 80° secondo. Si nota un leggero miglioramento rispetto al caso precedente. Qui il tempo di assestamento alla variazione del set-point dopo l'autotuning è appena di 5 secondi. D'altro canto, il leggero miglioramento era prevedibile, in quanto qui è utilizzata una tecnica differente, basata sul model based. Infatti, si aggiunge una richiesta in più, che è quella di settare il valore di λ. In questo caso, quindi, utilizzare la stima del modello per il calcolo dei parametri del regolatore, ha portato ad un leggero miglioramento del risultato.

Nel terzo metodo (Model based time domain : Metodo delle aree con sintesi IMC), non si considererà lo stesso processo, in quanto è del secondo ordine (vedi paragrafo 2.3.3) , ma un processo :

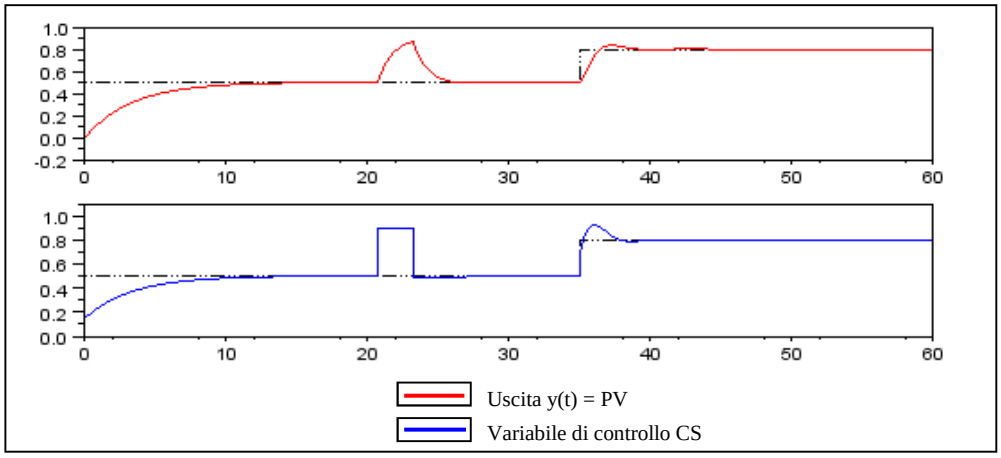
$$P(s) = \frac{1}{1+s}$$

I parametri da settare per la simulazione sono essenzialmente due :

- costante di tempo ad anello chiuso λ : 0.3
- periodo di campionamento Ts : 0.005 s

In questo caso, si rende necessario diminuire notevolmente il periodo di campionamento Ts, in quanto una sub-routine dovrà acquisire correttamente la risposta a scalino, per poter su di essa effettuare tutti i calcoli necessari a determinate i parametri del regolatore.

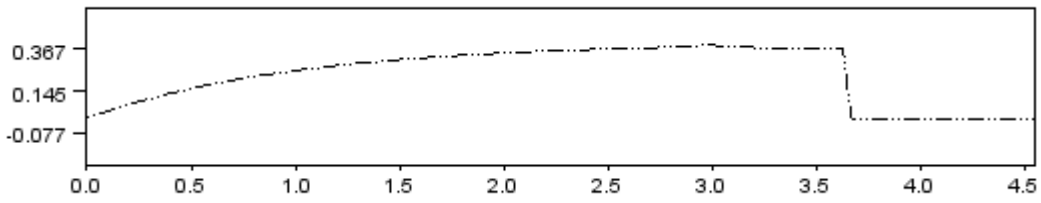
Il risultato della simulazione è il seguente :



[4.4] Example1.3 stand-alone : Ts=0.3; lambda=0.005

Il tempo si assestamento al cambio del set-point è di circa 5 secondi, ma si nota un leggero overshoot.

In questo caso, avendo utilizzato il metodo delle aree, si riporta in basso la figura della risposta a scalino acquisita :



[4.5] Risposta a gradino acquisita per il metodo delle aree

Essa è la stessa utilizzata per il calcolo dei parametri del regolatore PID, come descritto nel paragrafo 2.3.3.

Di seguito, si riportano svariate simulazioni eseguite per tutti e tre i metodi.

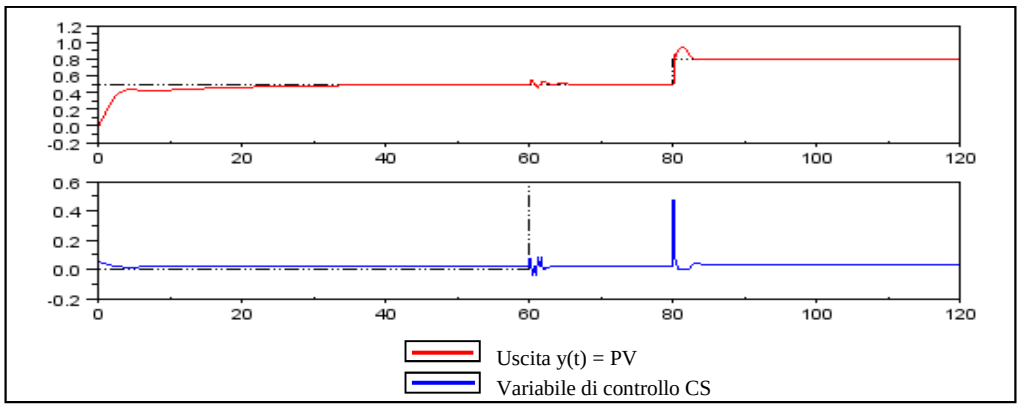
Per il primo , considerando un processo:

$$P(s) = \frac{5}{(1+2s)^2}$$

ed inserendo dei parametri :

- margine di fase richiesto pm : 70
- slope : 0.3
- periodo di campionamento Ts : 0.2
-

si ottengono risultati riportati in figura:



[4.6] Example1.4 stand-alone : pm=70 ; slope=0.3; Ts=0.2

Partendo da valori (palesemente errati) dei coefficienti K, T_i e T_d, si nota nel complesso, dopo l'operazione di autotuning, un netto miglioramento delle prestazioni del sistema.

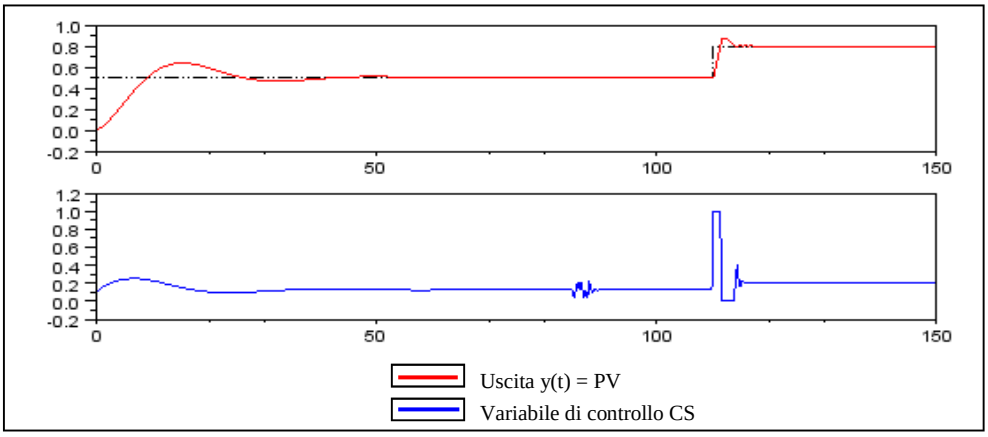
Per il secondo metodo è stato considerato un processo P(s), avente funzione di trasferimento :

$$P(s) = \frac{4}{(1+10s)\cdot(1+s)}$$

Con parametri :

- margine di fase richiesto pm : 80
- slope : 0.3
- periodo di campionamento Ts : 0.2
- costante ad anello chiuso λ : 0.1

Il risultato della simulazione appare in figura:



[4.7] Example1.4 stand-alone : pm=70 ; slope=0.3; Ts=0.2

Come si nota, al cambio del set-point si raggiunge la saturazione del controllo, sia sul limite superiore che inferiore, ma ciò non determina un problema.

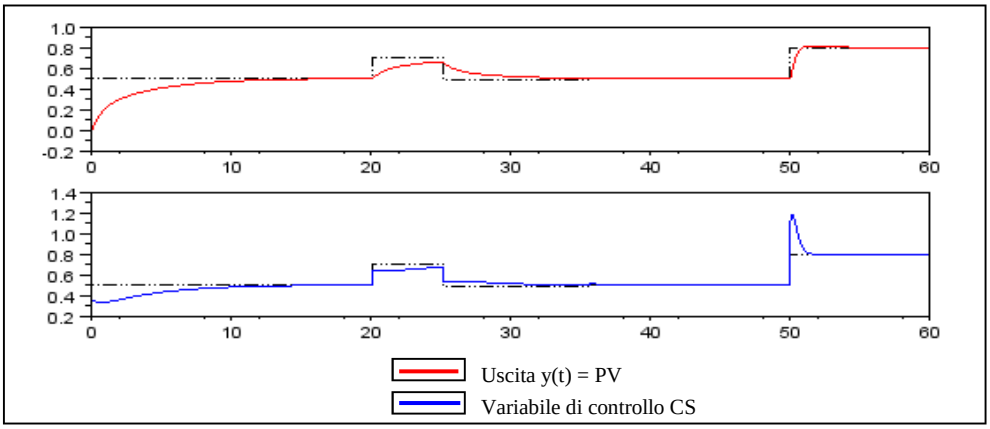
Per il terzo metodo è stato scelto un processo con ritardo :

$$P(s) = \frac{1}{1+s} e^{-1.5s}$$

Imponendo richieste del tipo :

- costante di tempo ad anello chiuso λ : 0.3
- periodo di campionamento Ts : 0.005 s

si è ottenuto il seguente risultato:

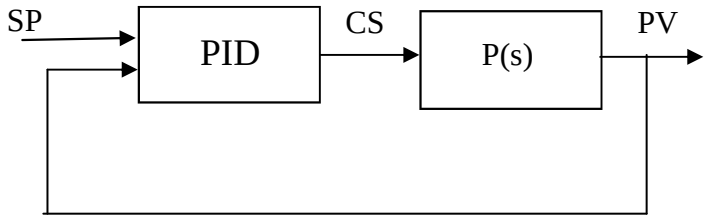


[4.8] Example1.6 stand-alone : Ts=0.005; lambda=0.005

4.2 Uso in LabVIEW

Per eseguire le simulazioni in LabVIEW e testare la funzionalità delle librerie realizzate, sono stati utilizzati i programmi Autotuning_PID_1, Autotuning_PID_2 ed Autotuning_PID_3, rispettivamente per il primo, secondo e terzo metodo di autotuning implementato. Per la scelta e configurazione dei parametri richiesti, si veda il paragrafo 3.2.

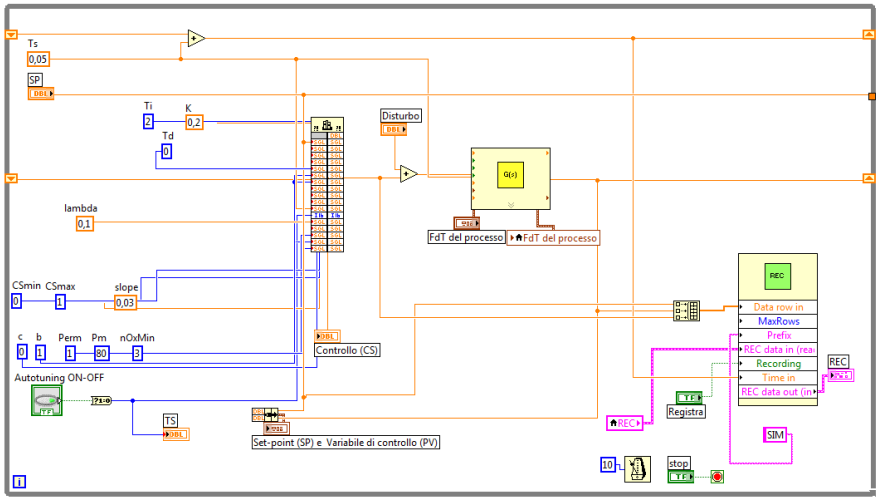
Anche qui, in maniera più semplice grazie all'ambiente di sviluppo G, è stato implementato il classico schema a blocchi retroazionato, del tipo :



[4.9] Schema a blocchi realizzato per il controllo

Questa volta, il set-point SP dovrà essere impostato in tempo reale durante l'esecuzione del programma, attraverso il pannello frontale del file VI.

Nell'immagine seguente, si riporta il corrispettivo schema a blocchi realizzato in labVIEW :



[4.10] Block Diagram del programma VI realizzato

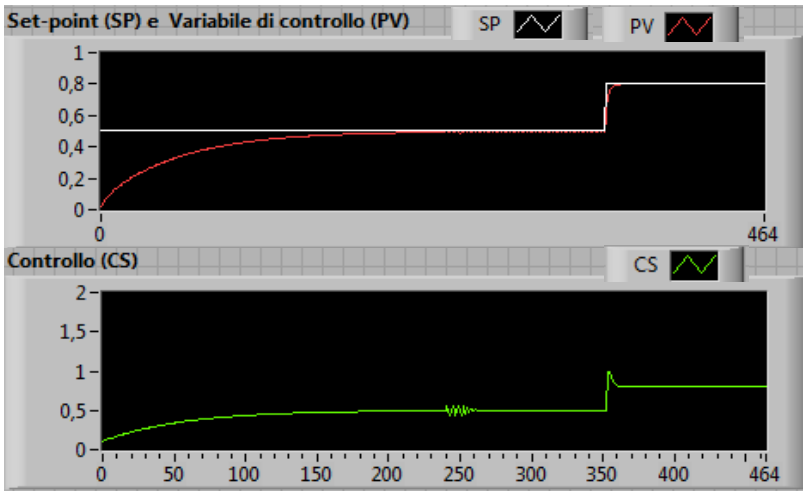
Nella simulazione seguente presentata con il primo metodo (Characteristics based time frequency : Relay based) è stato considerato un processo:

$$P(s) = \frac{1}{1+s}$$

I parametri impostati sono i seguenti:

- margine di fase richiesto pm : 70 ;
- slope : 0.3 ;
- periodo di campionamento Ts : 0.2.

Il risultato della simulazione nella finestra plot di LabVIEW è mostrato nella figura seguente:



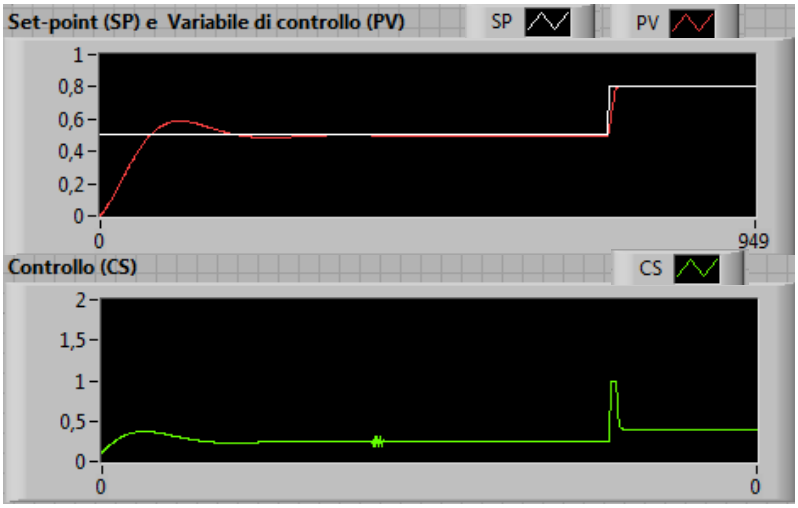
[4.11] Example2.1LabVIEW : pm=70; slope=0.3; Ts=0.2

E' possibile notare il netto miglioramento delle prestazioni del sistema all'inseguimento del set-point, dopo il comando di autotuning.

La seconda simulazione (sempre con il primo metodo) considera una funzione di trasferimento :

$$P(s) = \frac{2}{1+10s}$$

Il risultato, soddisfacente come prima, è il seguente:



[4.12] Example2.2 LabVIEW : pm=70; slope=0.3; Ts=0.2

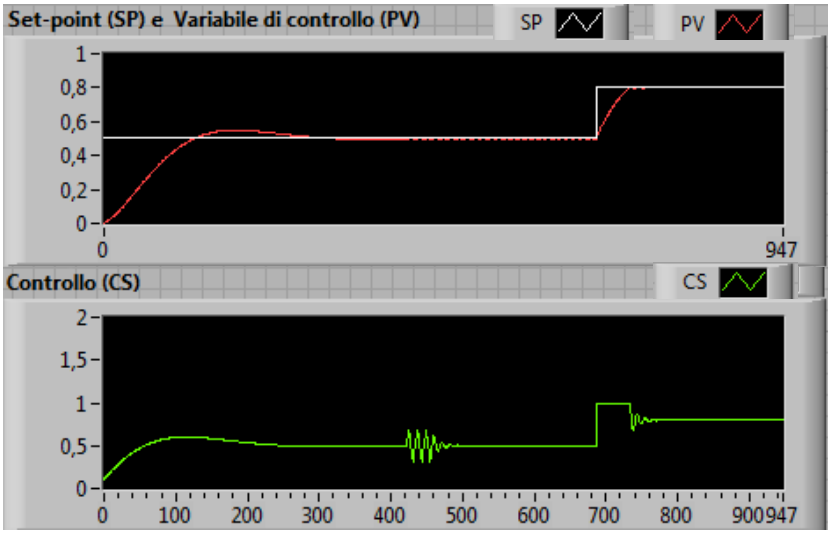
Per il secondo metodo (Model based contextual model parametrisation IMC) si presenta una simulazione avente processo:

$$P(s) = \frac{1}{1+10s}$$

con richieste:

- margine di fase richiesto pm : 80 ;
- slope : 0.05 ;
- periodo di campionamento Ts : 0.1 ;
- costante ad anello chiuso λ : 0.1 .

Nella figura seguente, il risultato:



[4.13] Example2.3LabVIEW : pm=80; slope=0.05; Ts=0.1; lambda=0.1

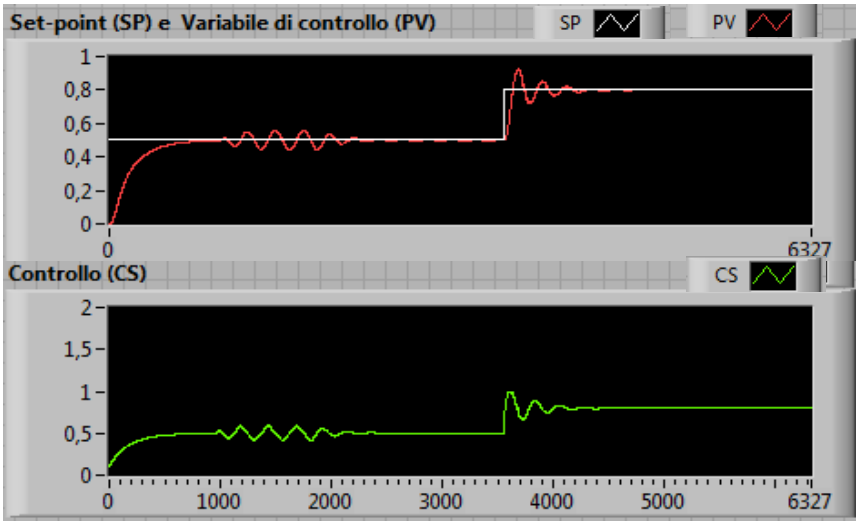
Si considera ora il processo P(s) :

$$P(s) = \frac{1}{1 + 3s + 4s^2 + 3s^3}$$

con le seguenti caratteristiche:

- margine di fase richiesto pm : 80 ;
- slope : 0.05 ;
- periodo di campionamento Ts : 0.01 ;
- costante ad anello chiuso λ : 0.1 ;

Il risultato è rappresentato nella figura :



[4.14] Example2.4LabVIEW : pm=80; slope=0.05; Ts=0.01; lambda=0.1

Si nota che le oscillazioni diventano più marcate, sia in fase di tuning che durante l'assestamento al valore di regime nella seconda fase. La situazione dopo il tuning sicuramente migliora, ma non è ottima.

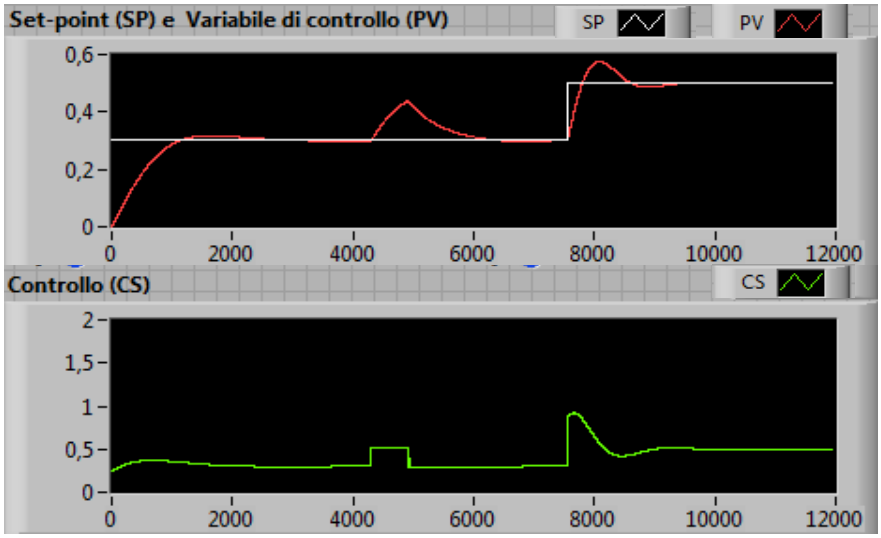
Per il terzo ed ultimo metodo (Model based time domain : Metodo delle aree con sintesi IMC), si considera un processo :

$$P(s) = \frac{1}{1+3s}$$

I valori impostati questa volta saranno:

- costante di tempo ad anello chiuso λ : 0.1 ;
- periodo di campionamento Ts : 0.005 s .

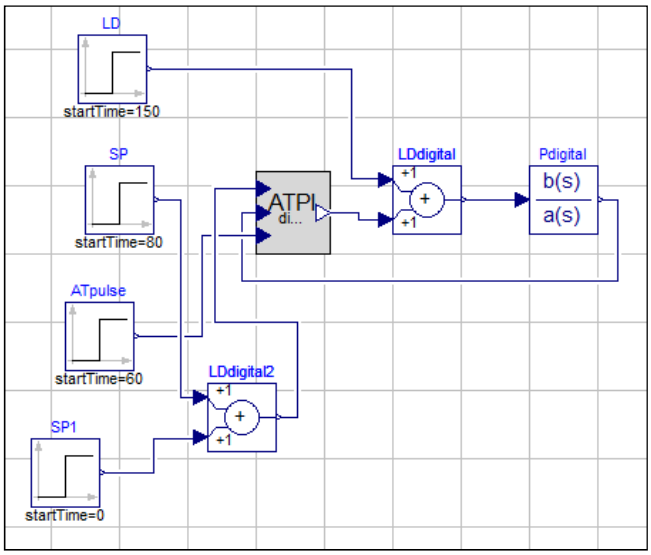
Il risultato è il seguente:



[4.15] Example2.5 LabVIEW : Ts=0.005; lambda=0.1

4.3 Uso in Modelica

In questo paragrafo si procede presentando alcuni risultati ottenuti in Dymola, mediante l'uso delle librerie realizzate. Anche qui, utilizzando l'interfaccia grafica messa a disposizione dal software, è stato creato uno schema a blocchi con retroazione. La figura seguente illustra lo schema realizzato :



[4.16] Schema a blocchi del programma realizzato in Modelica

La simulazione è stata effettuata utilizzando tutte e tre le librerie a disposizione, considerando tre diversi processi.

Per il primo metodo (Characteristics based time frequency : Relay based), si è scelto il seguente processo:

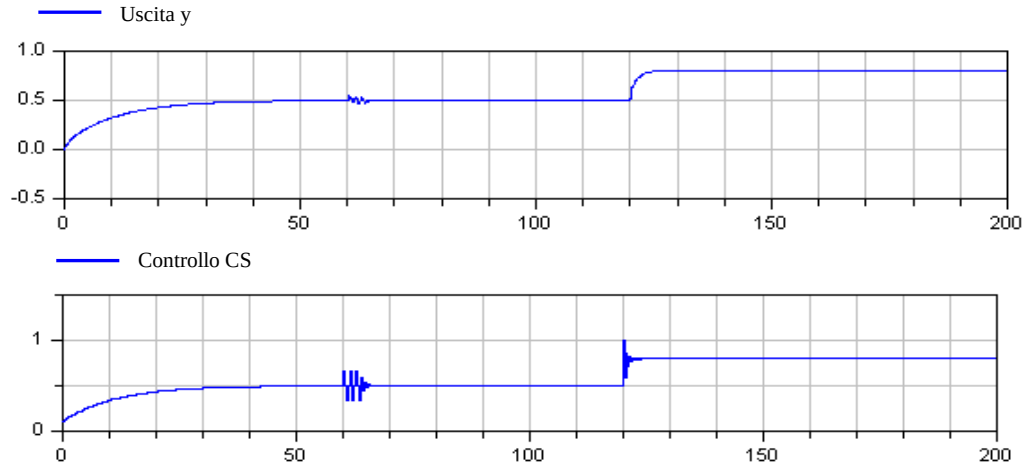
$$P(s) = \frac{1}{1+s}$$

In questo modo è possibile compararlo al risultato che labVIEW ci ha fornito nel paragrafo precedente (Paragrafo 4.2).

I parametri settati sono :

- margine di fase richiesto pm : 70 ;
- slope : 0.3 ;
- periodo di campionamento Ts : 0.2.

La figura seguente rende chiaro il risultato della simulazione :



[4.17] Example 3.1 Modelica: pm=70; slope=0.3; Ts=0.2

Il risultato ottenuto è molto simile a quello ottenuto da labVIEW. La risposta y è quasi sovrapponibile e la variabile di controllo CS differisce di poco.

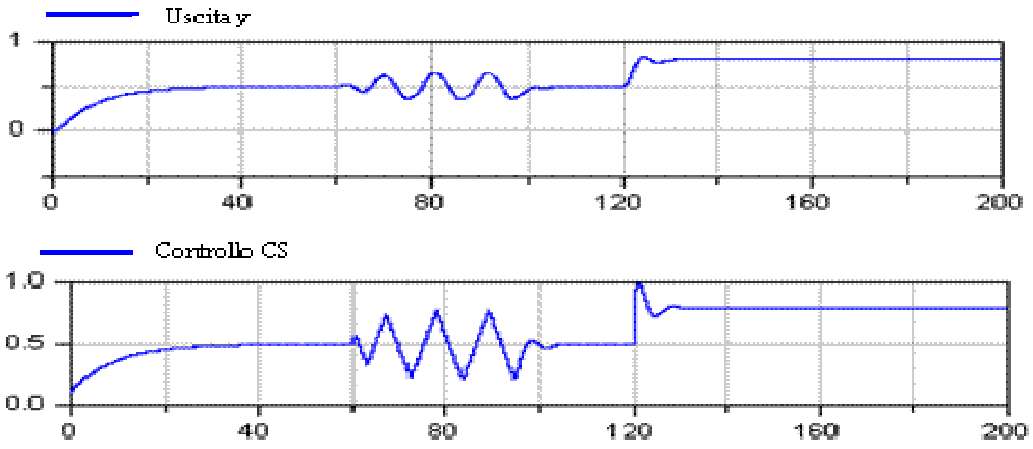
Nel secondo esempio, si considera un processo:

$$P(s) = \frac{1}{(1+s)^3}$$

Con parametri settati su:

- margine di fase richiesto pm : 60 ;
- slope : 0.1 ;
- periodo di campionamento Ts : 0.1 .

Il risultato è il seguente:



[4.18] Example 3.2 Modelica: pm=60; slope=0.1; Ts=0.1

Come è evidente in figura, l'autotuinig avviene al cinquantesimo secondo. Successivamente, al cambio di set-point, il sistema reagisce più velocemente.

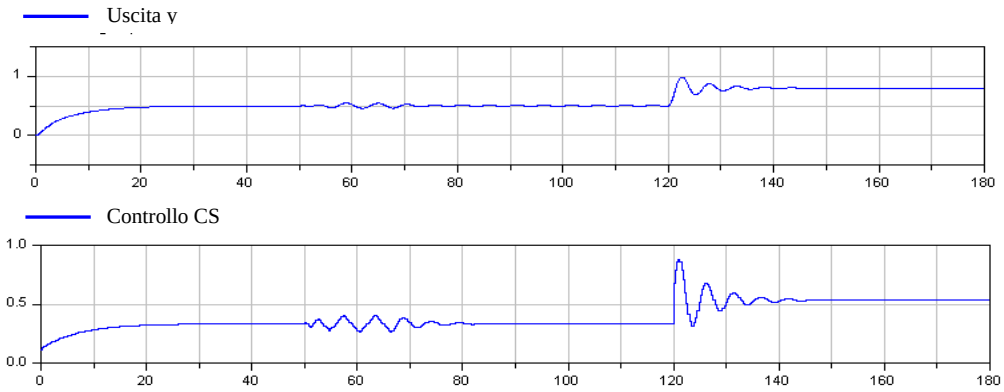
Si presenta ora una simulazione, utilizzando il secondo metodo di autotuning (Model based contextual model parametrisation IMC). Il processo considerato è in questo caso:

$$P(s) = \frac{1.5}{(1+s)^2}$$

con parametri :

- margine di fase richiesto pm : 60 ;
- slope : 0.05 ;
- periodo di campionamento Ts : 0.1 .

Di seguito, i risultati:



[4.19] Example 3.3 Modelica: pm=60; slope=0.05; Ts=0.1

In questo esempio si è impostata la variabile slope a 0.05 per avere meno influenza sul sistema durante la fase di tuning. Infatti le oscillazioni, durante l'esperimento sul processo, risultano molto meno marcate. Si nota, inoltre, che al crescere del guadagno μ di P(s) , le oscillazioni durante l'assestamento a regime aumentano.

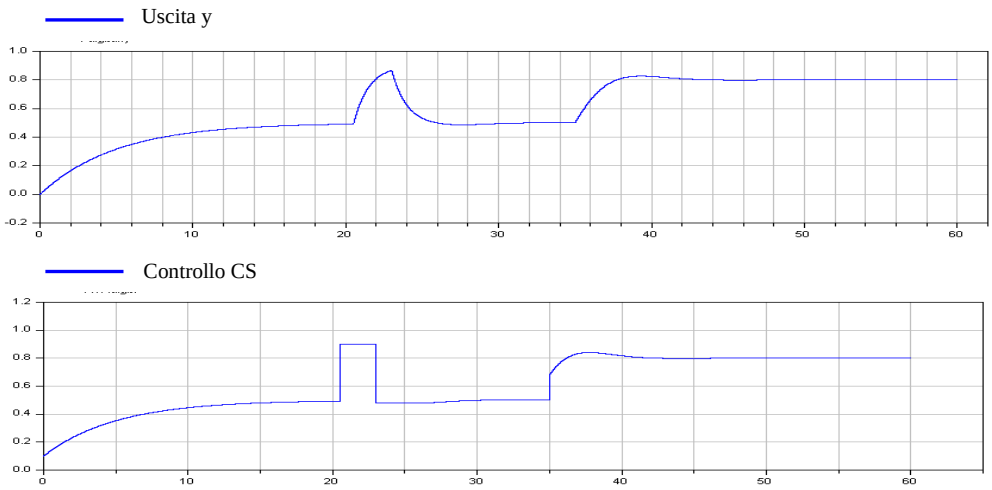
Per simulare la fase di autotunig, adoperando il terzo metodo (Model based time domain : Metodo delle aree con sintesi IMC), si è considerato il semplice processo del primo ordine :

$$P(s) = \frac{1}{1+s}$$

Richiedendo le seguenti caratteristiche:

- costante di tempo ad anello chiuso λ : 0.1 ;
- periodo di campionamento Ts : 0.005 s .

Il risultato si può notare nella seguente figura :



[4.20] Example 3.4 Modelica: lambda=0.1; Ts=0.005

Conclusioni

Lo scopo principale del lavoro era quello della realizzazione di alcune librerie C di regolatori con autotuning per uso stand-alone, in Modelica e LabVIEW.

Alla luce degli esiti dei test e di ulteriori prove, svolte al di fuori della campagna di verifica e di convalida, gli obiettivi iniziali del progetto possono considerarsi raggiunti.

Il primo passo è stato quello di scegliere tre metodi di autotuning differenti e di studiarne i diversi metodi realizzativi presenti in letteratura. Una volta che se ne è appreso il funzionamento, è stato cercato un approccio di realizzazione per l'algoritmo in codice C. Per far ciò si è partiti dalla lettura dell'articolo di A. Leva ed M. Maggio [13] ed, una volta apprese le nozioni fondamentali, si è realizzato l'algoritmo per ogni metodo di autotuning considerato.

Il secondo passo è stato quello di realizzare le corrispondenti librerie. Studiando un'opportuna organizzazione del codice C , il tutto è stato compilato sia come progetto di libreria statica (file.a) , per l'utilizzo in Modelica, sia come progetto di libreria dinamica (file.dll), per l'utilizzo in LabVIEW.

In seguito, si è passati alla valutazione vera e propria, effettuando alcune simulazioni al calcolatore. Il sistema realizzato ha dimostrato di poter regolare in maniera soddisfacente (se pur solo in simulazione) un semplice processo del primo o del secondo ordine. Inoltre, gli accorgimenti implementativi, adottati nella realizzazione dell'applicazione, consentono alla stessa di poter essere sviluppata ulteriormente attraverso l'aggiunta di nuove funzionalità e di poter essere inserita all'interno di un sistema di controllo più vasto.

Il prodotto realizzato, ovviamente, non può e non deve essere posto a diretto confronto con prodotti "professionali" analoghi, a causa delle limitazioni introdotte dalla struttura del codice; ma, soprattutto, perché non è questa la finalità per cui è stato

realizzato. Infatti, abbiamo visto come alcuni parametri implementativi dei metodi siano molto influenti sul risultato.

In realtà, più che il prodotto stesso, sono le attività di progettazione, di realizzazione del regolatore e delle rispettive librerie ad assumere importanza, in quanto hanno costituito per l'autore una valida opportunità per mettere in pratica le conoscenze acquisite nel proprio percorso formativo.

Concludendo, ci si può ritenere soddisfatti dei risultati conseguiti, come dai grafici riportati è stato possibile constatare.

Bibliografia

[1] A. Leva. PID autotuning algorithm based on relay feedback.
IEE Proceedings-D, 140(5):328–338, 1993.

[2] A. Leva, S. Negro, and A.V. Papadopoulos. PI/PID autotuning with contextual model parametrisation.
Journal of Process Control, 20(4):452–463, 2010.

[3] A. Leva. Simple model-based PID autotuners with rapid relay identification.
In Proc. 16th IFAC World Congress, Praha, Czech Republic,2005.

[4] A. Leva and L. Piroddi. Model-specific autotuning of classical regulator: a neural approach to structural identification.
Control Engineering and Practice, 4(10):1381–1391, 1996.

[5] A. Leva, L. Piroddi, "A Neural Network-Based Technique for Structural Identification of SISO Systems".
IMTC'94, Hamamatsu, 1994.

[6] A. Leva AutomationLabVIsuite (versione 1.0) Software libero distribuito nei termini della licenza GNU.
http://home.dei.polimi.it/leva/Projects/AutomationLabVIsuite/AutomationLabVIsuite-1.0.zip

[7] A. P. Loh, X. Cai e W. W. Tan. Auto-tuning of phase lead-lag compensators.
Automatica, 40:423–429, 2004.

[8] K. J. Astrom e T. Hagglund. Automatic tuning of simple regulators with specifications on phase and amplitude margins.
Automatica, 20:645–651, 1984.

[9] M. Mojiri e A. R. Bakhshai. An Adaptive Notch Filter for Frequency Estimation of a Periodic Signal.

[10] M. Riccato. Identificazione di Assi Meccanici Movimentati da Motori Lineari. Master's thesis, Dipartimento di Elettronica, Università di Padova, 2001.

[11] S. M. Savaresi, S. Bittanti e H. C. So. Closed-Form Unbiased FrequencyEstimation of a Noisy Sinusoid Using Notch Filters.
IEEE Transactions on Automatic Control, 48(7):1285–1293, July 2003.

[12] K. J. Astrom e T. Hagglund. PID Controllers: Theory, Design, and Tuning.
Instrument Society of America, 1995.

[13] M. Maggio A. Leva. Teaching to write control code.
18th IFAC World Congress Milano (Italy), 2011

[14] M. Friman e K. V. Waller. A Two-Channel Relay for Autotuning.
Ind. Eng. Chem. Res., 36:2662–2671, 1997.

[15] T. Fujinaka e S. Omatu. Pole Placement Using Optimal Regulators.
T.IEE Japan, 121(1), 2001.

[16] A. Girolami. Identificazione simultanea di parametri in sistemi lineari.
Master's thesis, Università di Pisa, 2003.

[17] M. Gregroire, A. Desbiens e E. Richard.
Development of an Auto Tuning PID.

[18] W. K. Ho, C. C. Hang e L. S. Cao. Tuning of PID Controllers Based on Gain and Phase Margin Specifications.
Automatica, 31(3):497–502, 1995.

[19] C. S. Junga, H. K. Song e J. C. Hyun. A direct synthesis tuning method of unstable first-order-plus-time-delay processes.
Journal of Process Control, 9: 265–269, 1999.

[20] S. Kozak e J. Hejdis. Simple, Robust, Selftuning Controller.
Applied Mathematics and Computation, 70:203–214, 1995.

[21] C. H. Lee. A survey of PID controller design based on Gain and Phase Margins (invited paper). International Journal of Computational Cognition, 2:63–100, 2004.

[22] G. P. Liu e S. Daley. Optimal-tuning PID controller design in the frequency domain with application to a rotary hydraulic system.
Control Engineering Practice, 7:821–830, 1999.

[23] K. Lo e W. H. Kwon. New identification approaches for disturbed models.
Automatica, 39:1627–1634, 2003

[24] L. Loron. Tuning of pid controllers by the non-symmetrical optimum method.
Automatica, 3(1):103–107, 1997.

[25] W. L. Luyben. Derivation of Transfer Functions for Highly Nonlinear.
Distillation Columns. Ind. Eng. Chem. Res., 26:2490–2495, 1987.

[26] G. Marchetti e C. Scali. Use of Modified Relay Techniques for the Design of Model Based Controllers for Chemical Processes.
Relazione tecnica, Dipartimento di Ingegneria Chimica, Università di Pisa, 1999.

[27] G. Marchetti, C. Scali e D. R. Lewin. Identification and Control of Open-Loop Unstable Processes by Relay Methods. Relazione tecnica,
Dept. of Chemical Engineering, University of Pisa.

[28] J. H. Park, S. W. Sung e I. B. Lee. Improved Relay Auto-tuning with Static Load Disturbance. Automatica, 33:711–715, 1997.

[29] J. H. Park, S. W. Sung e I. B. Lee. An Enhanced PID Control Strategy for Unstable Processes. Automatica, 34(6):751–756, 1998.

[30] N. Peric, I. Branica e I. Petrovic. Modification and application of autotuning PID controller.
In Proceedings of the 8th IEEE Mediterranean Conference on Control and Automation, 2000.

[31] R. Pintelon. Frequency-domain system identification using non-parametric noise models estimated from a small number of data sets.
Automatica, 38: 1295–1311, 2002.

[32] E. Poulin, A. Pomerleau, A. Desbiens e D. Hodouin. Development and Evaluation of an Auto-tuning and Adaptive PID Controller.
Automatica, 32:71–82,1996

[33] Z. Shafiei e A. T. Shenton. Tuning of PID-type controllers for stable and unstable systems with time delay.
Automatica, 30:1609–1615, 1994.

[34] Z. Shafiei e A. T. Shenton. Frequency-domain Design of PID Controllers for Stable and Unstable Systems with Time Delay.
Automatica, 33(12):2223–2232, 1997.

[35] S. Skogestad. Probably the best simple PID tuning rules in the world. Relazione tecnica, Department of Chemical Engineering National Chung Cheng University, TAIWAN.

[36] S. Skogestad. Simple analytic rules for model reduction and PID controller tuning. Journal of Process Control, 13:291–309, 2003.

[37] S. W. Sung e I. B. Lee. An improved algorithm for automatic tuning of PID controllers. Chemical Engineering Science, 55:1883–1891, 2000.

[38] S. W. Sung, I. B. Lee e B. K. Lee. On-line process identification and automatic tuning method for pid controllers.
Chemical Engineering Science, 53:1847–1859, 1998.

[39] S. W. Sung, I. B. Lee e J. Lee. New Process Identification Method for Automatic Design of PID Controllers. Automatica, 34(4):513–520, 1998.

[40] S. W. Sung, T. Y. Lee e S. Park. Optimal PID Controller Tuning Method for Single-Input-Single-Output Processes.
AIChE Journal, 48(6):1358–1361,June 2002.

[41] K. K. Tan, T. H. Lee e R. Ferdous. Simultaneous online automatic tuning of cascade control for open loop stable processes.
ISA Transactions, 39:233–242,2000.

[42] K. K. Tan, T. H. Lee e X. Jiang. Robust on-line relay automatic tuning of PID control systems. ISA Transactions, 39:219–232, 2000.

[43] K. K. Tan, Q. G. Wang, T. H. Lee e C. H. Gan. Automatic tuning of gainscheduled control for asymmetrical processes.
Control Engineering Practice,6:1353–1363, 1998

[45] R. Toscano. A simple robust PI/PID controller design via numerical optimization approach. Journal of Process Control, 15:81–88, 2005.

[44] K.K. Tan, Q. G. Wang e T.H. Lee. Relay-tuned finite spectrum assignment control of time delay processes.
ISA Transactions, 37:123–131, 1998.

[46] A. A. Voda e I. D. Landau. A method for the auto-calibration of pid controllers. Automatica, 31:41–53, 1995.

[47] Q. G. Wang, C. C. Hang e Q. Bi. Process Frequency Response Estimation from Relay Feedback. Control Eng. Practice, 5(9):1293–1302, 1997.

[48] Q. G. Wang, C. C. Hang e X. P. Yang. Single-loop controller design via IMC principles. Automatica, 37:2041–2048, 2001.

[49] Q. G. Wang, C. C. Hang e B Zou. Low-Order Modeling from Relay Feedback Ind. Eng. Chem. Res., 36:375–381, 1997.

[50] Q. G. Wang, Y. Zhang e X. Guo. Robust closed-loop identification with application to auto-tuning. Journal of Process Control, 11:519–530, 2001.

[51] Y. G. Wang, W. J. Cai e Z. G. Shi. PID autotuning for integrating processes with specifications on gain and phase margins.
In Proceedings of the American Control Conference, 2001.

[52] J. G. Ziegler e N. B. Nichols. Optimum Settings for Automatic Controllers. Trans. ASME, 65:433–444, 1942.
IEEE Transactions on Automatic Control, 49(2):314–318, February 2004.