INTERNAL REPORT

MINOR SERVO SYSTEM

Software osservativo per il controllo dei servo sistemi minori del Sardinia Radio Telescope e del Radiotelescopio di Medicina

Marco Buttu

Report N. 14, released: 13/12/2011

Reviewers: Sergio Poppi, Carlo Migoni

OAC Osservatorio Astronomico di Cagliari

MINOR SERVO SYSTEM
Software osservativo per il controllo dei servo sistemi minori del Sardinia Radio Telescope e del Radiotelescopio di Medicina
Marco Buttu <mbuttu@oa-cagliari.inaf.it>

Questo documento descrive la modellazione e l'implementazione del software osservativo per il controllo dei *servo minori*[1]. È stato studiato un modello di servo minore generico, in modo da poterlo adottare sia per il Sardinia Radio Telescope che per il Radio Telescopio di Medicina. Il software è in fase avanzata di sviluppo ed al momento la sua implementazione è specifica per il SRT.

Il report è strutturato in modo da poter risultare utile sia come documento tecnico che come guida consultativa all'utilizzo del software. Per poter far ciò ciascun argomento è stato introdotto cercando di delineare il contesto in cui si opera; è stato quindi descritto il framework ACS (ALMA Common Software) che rappresenta una infrastruttura alla quale sarà legato il software sviluppato, gli strumenti di design utilizzati e quelli in fase di studio. Il primo capitolo descrive il modello adottato per il servo sistema minore, la cui implementazione è descritta nel capitolo 2, mentre nel capitolo 3 viene riporato il sistema di test. Nel capitolo 4 si illustra il sistema Nuraghe e il component di *alto livello* che si occuperà di controllare i component dei servo minori.

Contatti

Per quanto riguarda la parte **software**:
Marco Buttu <mbuttu@oa-cagliari.inaf.it>
Andrea Orlati <a.orlati@ira.inaf.it>
Franco Fiocchi <f.fiocchi@ira.inaf.it>
Giuseppe Maccaferri <g.maccaferri@ira.inaf.it>

Per quanto riguarda la parte **hardware**:
Franco Fiocchi <f.fiocchi@ira.inaf.it>
Marco Morsiani <m.morsiani@ira.inaf.it>

[1] Con il termine *servo minore* intendiamo uno dei vari elementi che, tramite il suo movimento, consente di indirizzare su di un particolare ricevitore il segnale catturato dallo specchio primario.

Indice

Capitolo 1

Definizione del modello di servo sistema minore

1.1 Panoramica del sistema

Con il termine *servo minore* intendiamo uno dei vari elementi che, tramite il suo movimento, consente di indirizzare su di un particolare ricevitore il segnale catturato dallo specchio primario[1]. I servo minori quindi, sostanzialmente, consentono tramite il loro movimento coordinato di scegliere il ricevitore da utilizzare.

Il servo minore, che chiameremo anche *servo sistema minore* (MSS, Minor Servo System) è composto principalmente dai seguenti elementi:

- una **drive cabinet** (DC), che consiste in un armadio per l'automazione all'interno del quale sono installati tutti i componenti elettronici ed elettromeccanici di controllo del servo sistema
- uno o più **driver**, detti anche *azionamenti*, che consistono nell'elettronica di controllo del motore
- uno o più **motori**, che trasformano l'energia elettrica in energia meccanica necessaria al movimento
- uno o più **assi** fisici che, tramite il loro movimento, determinano i gradi

[1] Nel caso del Sardinia Radio Telescope, per esempio, i servo minori sono: il posizionatore del subriflettore (SRP), il posizionatore del fuoco primario (PFP), il mirror 3 rotator (M3) ed il gregorian feed rotator (GFR).

di libertà del componente (subriflettore, ricevitori in fuoco primario, specchio M3, ecc.) al quale sono legati solidalmene[2].

1.2 Schematizzazione del sistema

Nel nostro modello il servo minore non renderà pubblica la posizione dei suoi assi fisici ma quella degli assi del sistema di riferimento della struttura alla quale è legato (subriflettore, M3, ecc.), che chiameremo *assi virtuali*. In generale assi reali e virtuali non coincidono, così nello schema di figura 1.1 riportiamo una rappresentazione semplificata del servo sistema dove è chiaramente indicato che il sistema esporterà gli assi virtuali ed implementerà al suo interno una eventuale logica di conversione tra posizioni degli assi fisici e posizioni degli assi virtuali.

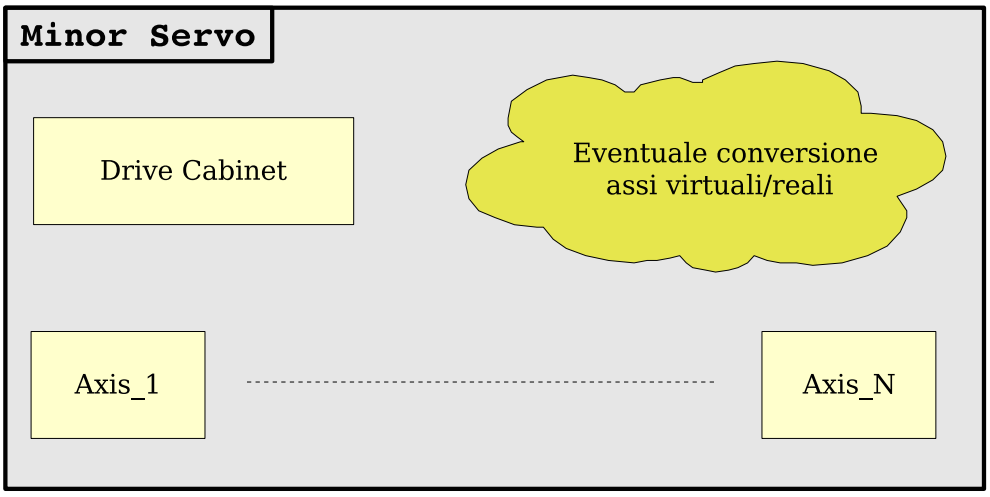

Figura 1.1: Generica rappresentazione del nostro modello di servo minore

1.3 Specifiche ACS

In questa sezione definiremo le specifiche implementative del servo minore al fine di realizzarne una sua integrazione in ACS. Chiameremo questo component ACS `WPServo` (Wave Path Servo).

[2]Per completezza chiariamo anche il significato di *attuatore*, il quale rappresenta il sistema meccanico che converte il movimento del motore nel movimento dell'asse (nel caso di assi con moto lineare).

1.3.1 Properties [5]

- `cmdPos`: RWdoubleSeq, posizione comandata dove ogni elemento della sequenza corrisponde ad una coordinata virtuale
- `actPos`: ROdoubleSeq, corrispondente alla posizione attuale dell'asse; ogni elemento della sequenza è una coordinata virtuale
- `posDiff`: ROdoubleSeq, differenza tra posizione comandata e attuale; ogni elemento rappresenta la differenza tra le corrispondenti coordinate
- `engTemperature`: ROdoubleSeq, temperatura dei motori, dove ogni elemento della sequenza è relativo ad un asse fisico
- `driTemperature`: ROdoubleSeq, temperatura dei driver, dove ogni elemento della sequenza è relativo
- `dcTemperature`: ROdouble, temperatura della drive cabinet
- `engVoltage`: ROdoubleSeq, tensione dei motori, dove ogni elemento della sequenza è relativo ad un asse fisico
- `driVoltage`: ROdoubleSeq, tensione dei driver, dove ogni elemento della sequenza è relativo ad un asse fisico
- `engCurrent`: ROdoubleSeq, corrente dei motori, dove ogni elemento della sequenza è relativo ad un asse fisico
- `driCurrent`: ROdoubleSeq, corrente dei driver, dove ogni elemento della sequenza è relativo ad un asse fisico
- `torquePerc`: ROdoubleSeq, percentuale di coppia dei motori
- `utilizationPerc`: ROdoubleSeq, percentuale di utilizzo dei motori (rapporto tra la potenza che si sta erogando e la massima erogabile)
- `counturingErr`: ROdouble, errore di inseguimento
- `driverStatus`: ROstringSeq, stringhe rappresentative dello stato dei driver
- `errorCode`: ROstringSeq, codici di errore dei driver
- `status`: ROpattern, status dell'hardware:
  - *ready*: 1 se il servo minore è pronto per essere posizionato;

- *warning*: se è a 1 allora si è verificata una condizione anomala cha però non dovrebbe condizionare l'osservazione;
- *failure*: 1 che c'è una condizione di failure e che l'osservazione dovrebbe essere compromessa.
- *setup*: 1 significa che il servo minore si trova in stato di startup;
- *park*: 1 significa che il servo minore è in stato di park.

La logica di comunicazione con l'hardware e l'eventuale conversione da assi virtuali a reali (e viceversa) verrà implementata a basso livello all'interno del DevIO; tutti i dettagli relativi al sistema hardware sottostante saranno così confinati ai livelli più bassi del nostro component `WPServo`.

Attributi delle Property

Di seguito sono elencati gli **attributi** delle property del `WPServo`:

- `default_timer_trig`: (double) presente in Pdouble, ROpattern; intervallo di tempo di default tra due campionamenti consecutivi della grandezza (callbacks consecutivi sul monitor)
- `initialize_devio`: (boolean) presente in Pdouble, ROpattern; utilizzato per indicare se inizializzare o no il devIO. Se posto a 1 allora il devIO verrà inizializzato a `default_value`
- `min_timer_trig`: (double) presente in Pdouble, ROpattern; intervallo di tempo minimo tra due pubblicazioni consecutive. Se un timer viene richiesto con un intervallo minore di `min_timer_trig` verrà aggiornato ad intervalli di `min_timer_trig`
- `description`: (string) presente in Pdouble, ROpattern; descrizione della property
- `units`: (string) presente in Pdouble, ROpattern; unità di misura della property
- `resolution`: (int) presente in Pdouble, ROpattern; bit pattern rappresentativo dei bit significativi del valore della property. Ad esempio, se i bit significativi per il valore della property fossero 4 allora `resolution` sarebbe 15 (ovvero $2^4 - 1$)
- `default_value`: (double per le Pdouble, int per le ROpattern) presente in Pdouble, ROpattern; se *initialize_devio* = 1 allora questo valore viene utilizzato per inizializzare il devIO

- `graph_min`: (double) presente in Pdouble; valore minimo da utilizzare per rappresentare la property graficamente
- `graph_max`: (double) presente in Pdouble; valore massimo da utilizzare per rappresentare la property graficamente
- `min_step`: (double) presente in Pdouble; minimo step di incremento e decremento della property
- `min_value`: (double) presente in RWdouble; valore minimo che può essere importato. ACS lancia una eccezione se si tenta di impostare un valore inferiore a `min_value`
- `max_value`: (double) presente in RWdouble; valore massimo che può essere impostato. ACS lancia una eccezione se si tenta di impostare un valore superiore a `min_value`
- `alarm_low_on`: (double) presente in ROdouble; quando il valore della property scende al di sotto di questo valore viene attivato l'allarme
- `alarm_high_on`: (double) presente in ROdouble; quando il valore della property sale al di sopra di questo valore viene attivato l'allarme
- `alarm_low_off`: (double) presente in ROdouble; al di sopra di questo valore viene disattivato alarm low
- `alarm_high_off`: (double) presente in ROdouble; al di sotto di questo valore viene disattivato alarm high
- `alarm_mask`: (int) presente in ROpattern; l'allarme può essere attivato solo per i bit "non mascherati"
- `alarm_trigger`: (int) presente in ROpattern; valore che attiva l'allarme: se posto a 1 allora l'allarme viene attivato per i bit non mascherati che vanno ad 1
- `alarm_timer_trig`: (double) presente in ROdouble, ROpattern; intervallo di tempo in cui il sistema di allarme si vuole vada a controllare il valore della property
- `bitDescription`: (string) presente in ROpattern; descrizione dei bit
- `whenSet`: (string) presente in ROpattern; codice del colore da utilizzare per rappresentare il bit quando è alto
- `whenCleared`: (string) presente in ROpattern; codice del colore da utilizzare per rappresentare il bit quando è basso

1.3.2 Metodi e callbacks

Il component avrà i seguenti metodi pubblici:

- `setPosition`: è un metodo che riceve due parametri: la posizone comandata (una doubleSeq) e un tempo (l'istante in cui deve essere comandata la nuova posizione);
- `setup`: esegue il setup del servo minore[3]. Il `setup` dei servo minori di SRT ha un tempo intrinseco di esecuzione dovuto al fatto che lancia un allarme per 30 secondi, restituendo un ACK sia quando il setup viene avviato sia quando viene concluso. Come argomento prende il tempo in cui dovrà essere eseguito
- `stow`: porta il servo minore in posizione di pargheggio; come argomento prende il tempo in cui dovrà essere eseguito
- `calibrate`: esegue la calibrazione
- `isParked`: restituisce *true* se il servo minore è in posizione di park
- `isReady`: restituisce *true* se il servo minore è pronto per il posizionamento
- `isTracking`: restituisce *true* se il servo minore è in stato di *tracking*
- `isStarting`: restituisce *true* quando il servo minore si trova in uno stato di inizializzazione
- `setStatusUpdating`: prende un parametro booleano; quando questo parametro è *true* allora lo stato del servo minore viene aggiornato periodicamente, altrimenti no. Questo è utile nel caso in cui si abbia una configurazione nella quale il servo minore è in stato di park, e non si vuole evitare di comunicare con la MSCU per conoscerne lo status;
- `isStatusThreadEn`: restituisce *true* quando l'aggiornamento dello status è abilitato;
- `getData`: prende un paramentro string che può assumere uno sei seguenti valori:
  - `POS_LIMIT`: il massimo valore positivo che può assumere la posizione dell'asse

[3]Nel caso dei servo minori di SRT viene chiamato il metodo `setup` del MSCU, il quale provvede a ripristinare i comandi ad alto livello.

- `NEG_LIMIT`: il minimo valore negativo che può assumere la posizione dell'asse
- `ACCELERATION`: l'accelerazione dell'asse
- `MAX_SPEED`: la massima velocità dell'asse

e restituisce il dato corrispondente sotto forma di doubleSeq. Ogni elemento della doubleSeq è relativo al corrispondente asse.

1.3.3 Attributi del component

In tabella 1.1 sono riportati gli **attributi** del component `WPServo`.

Attributo	Tipo	Descrizione
`number_of_axis`	unsignedShort	Numero di assi
`number_of_slaves`	unsignedShort	Numero di slave
`scale_factor`	double	Fattore di scala
`scale_offset`	double	Offset di scala
`server_ip`	string	Indirizzo IP
`server_port`	unsignedShort	Porta
`timeout`	unsignedLong	Timeout della connessione
`servo_address`	unsignedShort	Indirizzo del servo minore
`zero`	double	Posizione di zero
`park_position`	double	Posizione di parcheggio
`max_speed`	unsignedLong	Velocità massima
`min_speed`	unsignedLong	Velocità minima
`driver_type`	string	Modello dell'azionamento
`virtual_rs`	boolean	Posizioni *virtuali*?
`require_calibration`	boolean	Richiede la calibrazione?
`expire_time`	double	Tempo di validità dei dati
`tracking_delta`	double	La massima differenza di posizione ammessa

Tabella 1.1: Attributi del component `WPServo`

1.3.4 Schema

In figura 1.2 è riportato uno schema del component `WPServo`. Il blocco chiamato server è il software di interfaccia tra i servo minori ed il software ACS utilizzato nel calcolatore di stazione[4]. Tutte le property utilizzano lo stesso socket[5].

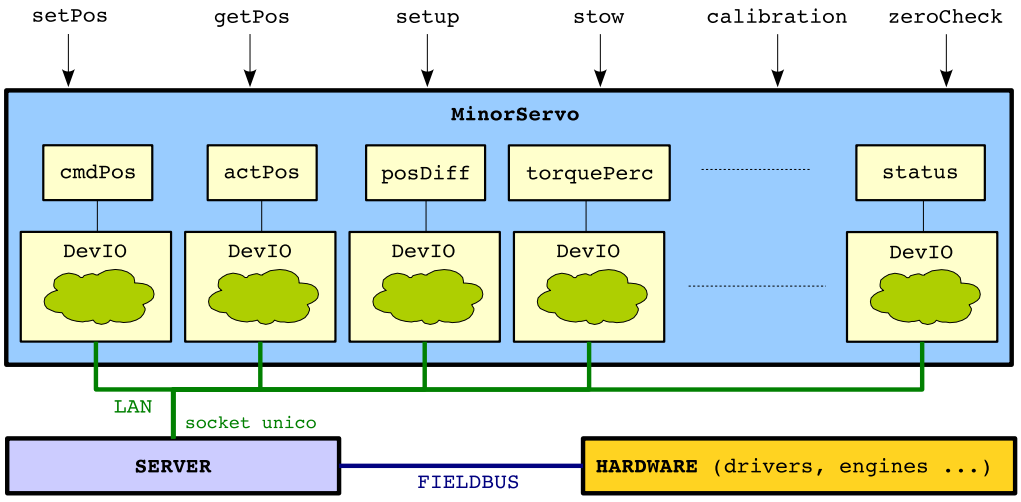

Figura 1.2: Schema ACS-like del component `WPServo`

Ciò che cambierà tra le varie implementazioni (ad esempio tra il subfiflettore di Medicina e quello di SRT) sarà il codice di basso livello confinato nelle DevIO delle property.

[4]Nel caso di SRT il server è chiamato MSCU (Minor Servo Control Unit) e provvede tramite interfaccia profibus a comandare gli assi nella posizone desiderata e a supervisionare e controllare le funzionalità del sistema.

[5]Il socket inoltre è condiviso con gli altri `WPServo`.

Capitolo 2

Implementazione ACS di un generico servo minore

2.1 Overview

Il codice sorgente del component, le librerie ed i file di test sono stati inseriti all'interno di una gerarchia di directory che rispecchia quella del repository SVN principale (repository contentente l'intero codice ACS sviluppato per Medicina ed SRT). All'interno di questa area di lavoro è contenuto solo il codice necessario per l'implementazione ACS dei servo minori. Nel seguito chiamerò *directory di lavoro* la root di tale progetto e tutti i path saranno relativi a tale directory.

Prima di procedere è necessario configurare correttamente ACS. Per far ciò modificare la variabile `ACS_PATH` del file `acsconf` in maniera tale che punti alla directory di lavoro; fatto ciò digitare da shell:

```
$ source acsconf
```

2.2 Interfaccia e Schema

L'interfaccia e lo schema degli attributi del component e delle sue property sono definiti in `Common/Interfaces/MinorServoInterface`, inquanto comuni ai servo minori di Medicina ed SRT. All'interno di questa directory l'interfaccia è descritta in `idl/MinorServo.idl`, mentre lo schema è definito in `Servers/SRTMinorServo/config/CDB/schemas/MinorServo.xsd`; in `src` è presente il Makefile.

Per compilare ed installare l'interfaccia andare in `src` e da shell (dopo aver configurato ACS come indicato nella sezione 5.1):

```
$ make
$ make install
```

Listato 2.1: Compilazione ed installazione dell'interfaccia

2.3 CDB

Nella directory `SRT/CDB/alma/MINORSERVO` troviamo 4 sotto-directory, una per ciascun servo minore: `GFR`, `SRP`, `PFP` e `M3R`; ciascuna di queste contiene un proprio file di configurazione. Questa gerarchia di directory la ritroviamo in `SRT/CDB/MACI/Components/MINORSERVO`, dove sono presenti le informazioni utilizzate dal manager per caricare i component. Il cuore di questi file sono le sequenti righe:

```
Code="WPServoImpl"
ImplLang="cpp"
Type="IDL:alma/MinorServo/WPServo:1.0"
Container="MinorServoContainer"
```

La prima riga indica che ciascun component `WPServo` è una istanza della classe `WPServoImpl`, nella seconda riga è indicato il lunguaggio nel quale è stata codificata la classe WPServo, nella terza riga troviamo le informazioni relative alla interfaccia (il nome del modulo e quello dell'interfaccia) definiti nella IDL, ed infine la quarta riga informa il manager che il container a cui è legato il component è il `MinorServoContainer`.

La variabile `ACS_CDB` punta al CDB che si vuole utilizzare; questa variabile può essere impostata nel file di configurazione `acsconf`. Le configurazioni possibili attualmente sono due, una per la parabola di Medicina ed una per SRT. Per quanto riguarda il component `WPServo` durante la fase di sviluppo è possibile utilizzare entrambe le configurazioni: il CDB di SRT verrà utilizzato in concomitanza con il server di test, mentre il CDB di medicina verrà utilizzato se il component dovrà dialogare con il MSCU. Nel capitolo 3 verranno discussi in dettaglio questi concetti.

2.4 Librerie

Il component `WPServo` utilizza la libreria `hexlib` [1] (realizzata da Franco Buffa) per effettuare una trasformazione del sistema di riferimento del SRP di SRT. La libreria `hexlib` a sua volta si appoggia alla GNU Scientific Library

la quale dovrà essere installata a parte. Per compilare ed installare la libreria `hexlib` si vada nelle directory `SRT/Libraries/MinorServoLibrary/src` e da shell:

```
$ make
$ make install
```

2.5 Il Component WPServo

I sorgenti del `WPServo` si trovano in `SRT/Servers/SRTMinorServo`; qua abbiamo come principali sotto-directory:

- `src`, contenente (oltre al Makefile) i seguenti sorgenti:
  - `WPServoImpl.cpp`: definisce la classe `WPServoImpl` dalla quale verranno istanziati i component `MinorServo`;
  - `WPServoSocket.cpp`: classe `minorServoSocket` utilizzata per connettersi all'hardware tramite socket. Questa classe può essere istanziata una sola volta (*singleton* pattern) per cui tutti i `MinorServo` utilizzeranno lo stesso socket;
  - `libCom.cpp`: libreria per la comunicazione con il MSCU;
  - `RequestScheduler.cpp`: thread che si occupa della schedulazione delle richieste provenienti dai vari servo minori;
  - `SocketListener.cpp`: thread che si occupa di smistare le risposte provenienti dal MSCU;
  - `WPServoTalker.cpp`: file per la definizione delle stringhe di comando da inviare al MSCU;
  - `WPStatusUpdater.cpp`: thread che si occupa di aggiornare lo stato dei servo minori e le loro posizioni attuali e comandate;
  - `utils.cpp`: definisce alcune funzioni utili per il parsing;
  - `WPUtils.cpp`: definisce alcune funzioni di utilità specifiche dei component di basso livello.
- `include`, contentente (oltre agli *header* dei file sorgenti di `src`) i device di I/O delle property rappresentative dell'hardware.
- `test`, contentente i test (descritti in dettaglio nel capitolo 3);
- `bin`, contenente alcuni eseguibili utilizzati durante i test;

- `doc`, contenete la documentazione.

Per compilare ed installare il component `WPServo` si vada in `src` e da shell si diano i seguenti comandi:

```
$ make
$ make install
```

Capitolo 3

Sistema di Test

3.1 Overview

I test rivestono un ruolo chiave nello sviluppo del component `WPServo`, ed ogni singola funzionalità viene implementata solo dopo aver scritto il corrispondente test automatico.

Per facilitare questo tipo di approccio *agile*, che comporta una frequente e veloce esecuzione dei test, si è deciso di sviluppare un simulatore del MSCU che faciliti questo scopo. Il simulatore inoltre, poichè consente di simulare vari tipi di risposta (inaspettata, ACK, NACK, ecc.), permette di effettuare delle validazioni relative a dei casi che non sarebbe possibile ottenere in modo ripetibile ed autometico interrogando direttamente il MSCU[1].

La suite di test, prevalentemente scritta con il linguaggio di programmazione Python, si trova nella directory `SRT/Servers/SRTMinorServo/test` e contiene i seguenti file:

- `mscu.py`: modulo che implementa un simulatore del MSCU;
- `parameters.py`: modulo utilizzato per configurare i test;
- `wpservo_test.py`: modulo contenente i test unitari automatici;
- `real2virtual.c`: file utilizzato per generare un eseguibile (utilizzato nei test unitari) che realizza una conversione da assi reali a virtuali del subreflector positioiner di SRT;
- `virtual2real.c`: analogo a `real2virtual.c`, ma da luogo ad un eseguibile che converte da assi virtuali ad assi reali.

[1] La verifica di tutti questi casi comporta che il codice sia più *robusto*, e questo evidentemente a lungo termine è l'approccio migliore anche dal punto di vista della ottimizzazione dei tempi di sviluppo e di debug.

I test possono anche essere eseguiti direttamente sul MSCU reale, passando la chiave "MSCU" al dizionario `servers` del modulo `wpservo_test.py` (come discusso in dettaglio nella sezione 3.3) e ponendo in `acsconf`:

```
ACS_CDB = /full_path/SRT
```

In seguito verrà descritto in dettaglio come configurare la suit di test a seconda che si voglia usare il server reale o il suo simulatore, e a seconda di come si voglia configurare il tipo di risposta di quest'ultimo.

3.2 MSCU Simulator

Il simulatore del MSCU è implementato nel modulo `mscu.py`. Consiste sostanzialmente in un server che riceve delle richieste di esecuzione di alcuni comandi[2] e risponde più o meno correttamente a seconda di come viene configurato.

Il MSCU simulator viene istanziato passando al costruttore i seguenti parametri (tutti opzionali):

- `host`: l'indirizzo IP dal quale sono accettate delle connessioni (di default tutti gli IP). Il server accetta un numero arbitrario di connessioni;
- `port`: la porta nella quale si mette in ascolto (di default la porta 8000);
- `response_type`: il tipo di risposta che dovrà generare:
  1. `expected_ack`: risposta corretta dal punto di vista della sintassi e utile come risultato (risposta di default);
  2. `expected_nak`: risposta corretta dal punto di vista sintattico ma priva dell'informazione richiesta (ad esempio quando il server ci informa che il servo minore non può essere interrogato);
  3. `without_closer`: genera risposte prive di carattere terminatore;
  4. `unexpected`: genera risposte inaspettate, con header scorretto, con un numero errato di parametri, ecc.

Il simulatore si comporta esattamente come il MSCU per cui tutte le risposte a richieste di posizione sono relative agli assi fisici del servo minore.

Per lanciare il server di sviluppo (dopo aver configurato ACS come indicato nella sezione 5.1) si esegue da shell il comando `run_server`. Si aprirà

[2] Il simulatore risponde a tutti i comandi della MSCU detti di *alto livello*. Può essere inoltre configurato in modo che risponda in modo errato, permettendo così di verificare anche le condizioni di errore.

una nuova finestra nella quale verranno visualizzati i vari messaggi di logging del server; questo ad esempio è quanto appare nel terminale del server non appena viene avviato:

```
****************************************
MSCU_simulator - Waiting for connections...
****************************************
Response type: expected_ack
```

3.3 Test Unitari Automatici

Il test unitari si trovano nel modulo `wpservo_test.py` e sono sono realizzati utilizzando PyUnit (modulo `unittest`). Sostanzialmente i controlli effettuati nel test verificano che gli ouput generati dal component siano quelli attesi. Per far ciò le classi di test interrogano sia il component che il server (direttamente) per poi confrontare i risultati sulla base di quanto ci si attende. La variable `server` definisce il server che la classe di test va ad interrogare direttamente e viene impostata come mostrato di seguito:

```
    # Set server = servers['MSCU'] to connect directly to MSCU
    server = servers['MSCUSimulator']
```

La validazione delle posizioni dei servo minori con assi *virtuali* viene fatta previa conversione delle posizioni fisiche ottenute dal server. Nel caso del posizionatore del subriflettore di SRT, questa conversione è realizzata dai due eseguibili `real2virtual` e `virtual2real`, i quali utilizzando la libreria `hexlib`.

Per generare gli eseguibili si vada nella directory `test` e da shell si esegua `make`; a questo punto nella directory `bin` vi saranno `real2virtual` e `virtual2real`.

3.4 Utilizzo del component ed esecuzione dei test step-by-step

In questa sezione riassumiamo i vari passi necessari per configurare ed avviare ACS, i component dei servo minori ed i test.

3.4.1 Configurazione di ACS

Per configurare ACS si vada nella directory di lavoro e si editi il file `acsconf`, le cui variabili principali sono riportate nel listato 3.1.

```
1  #!/usr/bin/env bash
2
3  source ~/.acs/.bash_profile.acs
4
5  ACS_PATH=/home/marco/minor_servo
6  ACS_CDB=$ACS_PATH/SRT
7
8  # Library path for GSL
9  LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
10 export LD_LIBRARY_PATH
11 ...
```

Listato 3.1: Principali variabli definite in `acsconf`

Si modifichi il path che verrà assegnato alla variabile `ACS_PATH` in modo che punti alla directory di lavoro. Se si vuole utilizzare il MSCU e non il simulatore si ponga `ACS_CDB=$ACS_PATH/SRT`. Se durante l'installazione delle GSL si è scelto di utilizzare le librerie dinamiche, qualora il path fosse diverso da quello assegnato a `LD_LIBRARY_PATH`, si modifichi anche il valore da assegnare a questa variabile. Fatto ciò si può configurare ACS dando da shell il comando:

```
$ source acsconf
```

Se si utilizza il simulatore è necessario modificare l'attributo `server_ip` nei file `SRT/CDB/alma/MINORSERVO/*/*.xml`, assegnandogli l'indirizzo IP della macchina sulla quale si sta lavorando.

3.4.2 Building

A questo punto possiamo compilare ed installare l'interfaccia IDL. Si vada nella directory `Common/Interfaces/MinorServoInterface/src` e da shell si diano i seguenti comandi:

```
$ make
$ make install
```

Per compilare ed installare le librerie `hexlib` si proceda come indicato nella sezione 2.4.

Si proceda allo stesso modo (`make` e `make install`) anche per compilare ed installare il component `WPServo`, questa volta dando i comandi all'interno della directory `SRT/Servers/SRTMinorServo/src`.

Per generare gli eseguibili necessari per il test si dia il comando `make` all'interno della directory `SRT/Servers/SRTMinorServo/test`.

3.4.3 Avvio di ACS, del server e del component

Dopo aver completato gli step precedenti si può avviare ACS, il simulatore ed il container:

```
$ acsStart
$ run_server
$ getup_container
```

Durante la fase di avvio del container vengono attivati automaticamente anche i servo minori e, come mostra la finestra di logging del server:

```
****************************************
MSCU_simulator - Waiting for connections...
****************************************
Response type: expected_ack


1. Got connection from ('192.167.8.78', 40140)
```

viene utilizzato un unico socket per la comunicazione. A questo punto siamo in grado di eseguire i test automatici.

3.4.4 Running dei test automatici

Per lanciare i test si va nella directory `SRT/Servers/SRTMinorServo/test` e da shell si da il comando:

```
$ ./wpservo_test.py
```

Ad esempio, nel caso di un test che va a buon fine, si avrà un output simile a questo:

```
Test the cmdPos property of every minor servo. ... ok
Test the actPos property of every minor servo. ... ok
Test the counturingErr property of every minor servo. ... ok
Test the dcTemperature property of minor servo. ... ok
Test the driTemperature property of every minor servo. ... ok
```

```
Test the driverStatus property of every minor servo. ... ok
Test the engCurrent property of every minor servo. ... ok
Test the engTemperature property of every minor servo. ... ok
Test the engVoltage property of every minor servo. ... ok
Test the errorCode property of every minor servo. ... ok
Test the status property of every minor servo. ... ok
Test the torquePerc property of every minor servo. ... ok
Test the utilizationPerc property of every minor servo. ... ok
Test all properties concurrently. ... ok

----------------------------------------------------------------------
Ran 14 tests in 143.438s

OK
```

Il seguente invece è un esempio di output di un test di `actPos` che fallisce (il test rileva che una coordinata della posizione restituita dal component ACS è diversa da quella letta direttamente dal server):

```
Test the actPos property of every minor servo. ... FAIL

======================================================================
FAIL: Test the actPos property of every minor servo.
----------------------------------------------------------------------
Traceback (most recent call last):
  File "./test.py", line 105, in test_getActPos
    self.assertAlmostEqual(acs_positions[idx], positions[idx])
    AssertionError: -0.0026168428301381152 != -0.002617

----------------------------------------------------------------------
Ran 1 test in 0.109s

FAILED (failures=1)
```

Capitolo 4

Il MinorServoBoss

4.1 Il sistema Nuraghe

Il sistema software per la gestione delle osservazioni con il SRT è stato chiamato *Nuraghe* ed è mostrato in figura 4.1.

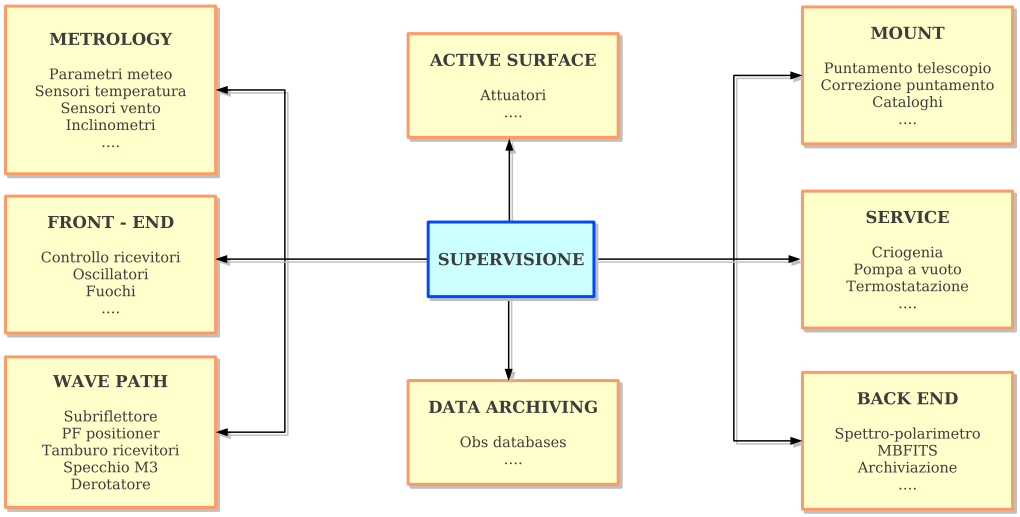

Figura 4.1: Schema del sistema *Nuraghe*

Il sistema Nuraghe è strutturato su una gerarchia di tre livelli:

1. component di basso livello, che tipicamente si interfacciano con l'hardware, come ad esempio i component `WPServo`
2. sottosistemi che supervisionano i component di basso livello;
3. un supervisore che coordina i sottosistemi.

I *sottosistemi* sono raggruppati per omogeneità funzionale. Ognuno di loro contiene i component necessari per il controllo di determinati hardware. Il sottosistema che si occupa del controllo dei servo minori è indicato in figura 4.1 con il nome di *Wave Path* e si occupa di:

- avviare e supervisionare i servo minori;
- selezionare il ricevitore in funzione della configurazione richiesta;
- effettuare le correzioni di posizione degli assi dei servo minori in funzione dell'elevazione;
- abilitare o disabilitare per ciascun servo minore le correzioni in funzione dell'elevazione;
- effettuare una curva di fuoco sia programmata che in *real time*;
- eseguire comandi impartiti da client esterni.

Il `MinorServoBoss` è il component che si occura della supervisione dei servo minori e si trova nella directory `Common/Servers/MinorServo`. I sorgenti relativi al sottositema Wave Path sono i seguenti:

- `MinorServoBossImpl.cpp`: definisce la classe `MinorServoBossImpl` dalla quale verrà istanziato il component `MinorServoBoss`;
- `MSBossPublisher.cpp`: si occupa della pubblicazione dello status del sottosistema su di un *Notification Channel*;
- `TrackingThread.cpp`: thread che gestisce le correzioni delle posizioni degli assi dei servo minori in funzione dell'elevazione;
- `SetupThread.cpp`: thread che si occupa dell'inizializzazione del sistema.

4.2 Interfaccia del MinorServoBoss

Descriveremo in questa sezione l'interfaccia del component `MinorServoBoss`.

4.2.1 Il metodo command

```
/**
 * This method implements the command line interpreter.
 * The interpreter allows to ask for services or to issue commands
 * to the sub-system by human readable command lines.
 * @param cmd string that contains the command line
 * @return the string that contains the answer to the command if
 * succesful. It must be freed by the caller.
 * @throw CORBA::SystemException
 * @throw ManagementErrors::CommandLineErrorEx Thrown when the
 * command execution or parsing result in an error.
 */
virtual char * command(const char *cmd) throw (
    CORBA::SystemException,
    ManagementErrors::CommandLineErrorEx
);
```

Listato 4.1: Dichiarazione del metodo `command`

4.2.2 Il metodo park

```
/** This method is used to stow the minor servos.
 *
 * @throw CORBA::SystemExcpetion
 * @throw ManagementErrors::ParkingErrorEx
 */
void park() throw (
    CORBA::SystemException,
    ManagementErrors::ParkingErrorEx
);
```

Listato 4.2: Dichiarazione del metodo `park`

4.2.3 Il metodo isTrackingEn

```
/** Return true when the elevation tracking is enabled */
bool isTrackingEn();
```

Listato 4.3: Dichiarazione del metodo `isTrackingEn`

4.2.4 Il metodo `setup`

```
/** This method will be used to configure the MinorServoBoss
 * before starting an observation.
 * @param config mnemonic code of the required configuration
 * @throw CORBA::SystemException
 * @throw ManagementErrors::ConfigurationErrorEx
 */
void setup(const char *config) throw (
    CORBA::SystemException,
    ManagementErrors::ConfigurationErrorEx
);
```

Listato 4.4: Dichiarazione del metodo `setup`

4.2.5 Il metodo `turnTrackingOn`

```
/** Turn the elevation tracking of minor servos on
 * @throw ManagementErrors::ConfigurationErrorEx
 */
void turnTrackingOn() throw(ManagementErrors::ConfigurationErrorEx);
```

Listato 4.5: Dichiarazione del metodo `turnTrackingOn`

4.2.6 Il metodo `turnTrackingOff`

```
/** Turn the elevation tracking of minor servos off
 * @throw ManagementErrors::ConfigurationErrorEx
 */
void turnTrackingOff() throw(ManagementErrors::ConfigurationErrorEx);
```

Listato 4.6: Dichiarazione del metodo `turnTrackingOff`

4.2.7 Il metodo `getActualSetup`

```
/** Return the actual configuration */
char * getActualSetup();
```

Listato 4.7: Dichiarazione del metodo `getActualSetup`

4.2.8 Il metodo `getCommandedSetup`

```
/** Return the commanded configuration */
char * getCommandedSetup();
```

Listato 4.8: Dichiarazione del metodo `getCommandedSetup`

4.2.9 Il metodo `checkScan`

Nel capitolo 5 ci occuperemo in dettaglio di un possibile algoritmo da utilizzare per effettuare lo scan.

```
/** Check if the scan is achievable
 * @param starting_time the time the scan will start
 * @param range the total axis movement in mm (centered in the
 * actual position)
 * @param total_time the duration of axis movement
 * @param axis the identification code of the axis
 * @param servo the servo name
 * @return true if the scan is achievable
 * @throw ManagementErrors::ConfigurationErrorEx
 * @throw ManagementErrors::SubscanErrorEx
 */
bool checkScan(
       const ACS::Time starting_time,
       double range,
       const ACS::Time total_time,
       const unsigned short axis,
       const char *servo
) throw (
     ManagementErrors::ConfigurationErrorEx,
     ManagementErrors::SubscanErrorEx
);
```

Listato 4.9: Dichiarazione del metodo `checkScan`

4.2.10 Il metodo `startScan`

```
/** Start the scan of one axis of a MinorServo target.
 * @param starting_time the time the scan will start
 * @param range the total axis movement in mm (centered in the
```

```
 * actual position)
 * @param total_time the duration of axis movement
 * @param axis the identification code of the axis
 * @param servo the servo name
 * @return pointer to ROpattern verbose status property
 * @throw ManagementErrors::ConfigurationErrorEx
 * @throw ManagementErrors::SubscanErrorEx
 */
void startScan(
       const ACS::Time starting_time,
       const double range,
       const ACS::Time total_time,
       const unsigned short axis,
       const char *servo
) throw (
     ManagementErrors::ConfigurationErrorEx,
     ManagementErrors::SubscanErrorEx
);
```

Listato 4.10: Dichiarazione del metodo `startScan`

4.2.11 Il metodo `stopScan`

```
/** Stop the scan */
void stopScan() throw (ManagementErrors::SubscanErrorEx);
```

Listato 4.11: Dichiarazione del metodo `stopScan`

Capitolo 5

Traiettorie controllate

5.1 Premessa

In questo capitolo descriveremo un possibile algoritmo da utilizzare per effettuare uno *scan* (sottosezioni 4.2.9, 4.2.10, 4.2.11), ovvero uno spostamento da un punto di partenza x_0 ad un punto di arrivo x_F in un tempo stabilito. Indichiamo con $x_T = x_F - x_0$ l'elongazione totale richiesta e con $t_R = t_F - t_0$ il tempo richiesto per compiere la movimentazione (il tempo da impiegare per la movimentazione deve essere esattamente t_R). La velocità iniziale nel punto x_0 sarà nulla mentre l'accelerazione a e la velocità massima v_{max} dell'asse sono dei parametri definiti in fase di configurazione e saranno per noi delle costanti.

Come prima cosa dobbiamo stabilire se nel tempo t_R è possibile compiere una movimentazione dell'asse pari a x_T, in caso contrario non sarà possibile realizzare la traiettoria nel tempo stabilito. Per poter calcolare il tempo impiegato dobbiamo capire se l'andamento della velocità nel tempo è trapezoidale o triangolare. Per far ciò calcoliamo lo spazio minimo necessario per raggiungere la v_{max} partendo da una velocità iniziale nulla; indichiamo con $x_{v_{max}}$ tale spazio e con $t_{v_{max}}$ il tempo necessario per raggiungerlo.

Dal sequente sistema:

$$\begin{cases} v_{max} = a\, t_{v_{max}} \\ \\ x_{v_{max}} = \dfrac{1}{2}\, a\, t_{v_{max}}^2 \end{cases}$$

otteniamo:

$$x_{v_{max}} = \frac{1}{2}\, \frac{v_{max}^2}{a} \tag{5.1}$$

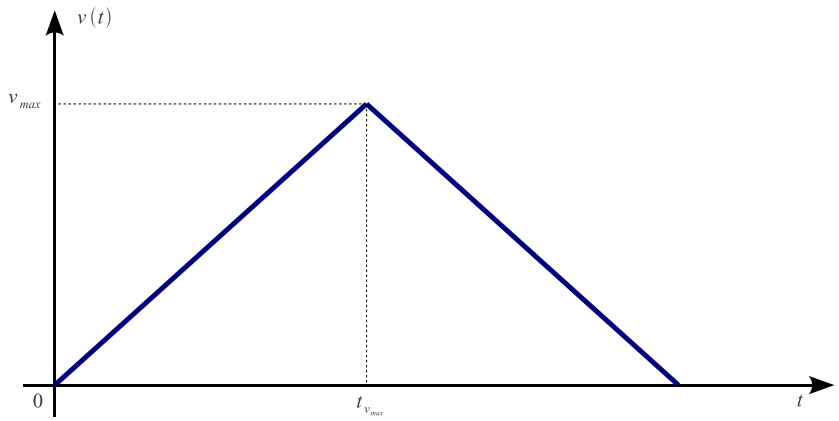

Figura 5.1: Andamento triangolare della velocità con $v_0 = 0$ e massimo in v_{max}

Poichè l'accelerazione e la decelerazione hanno uguale valore, lo spazio minimo da percorrere per avere un andamento trapezoidale della velocità in funzione del tempo sarà pari a $2x_{v_{max}}$, quindi:

$$\begin{cases} x_T \leq \dfrac{v_{max}^2}{a} & \rightarrow \quad \text{andamento triangolare} \\ x_T > \dfrac{v_{max}^2}{a} & \rightarrow \quad \text{andamento trapezoidale} \end{cases} \tag{5.2}$$

Il primo passo consiste quindi nell'utilizzare le 5.2 per capire quale formula applicare per il calcolo del tempo t_d richiesto per la movimentazione diretta.

5.2 Tempo necessario per la movimentazione

Noto l'andamento della velocità (ricavato dalle 5.2), vediamo come calcolare il tempo t_d necessario per compiere l'elongazione x_T dell'asse partendo da velocità nulla.

5.2.1 Caso triangolare

Indichiamo con v_p la velocità di picco ($v_p \leq v_{max}$) e con t_d il tempo necessario per compiere l'elongazione x_T dell'asse partendo da una velocità nulla. L'andamento della velocità in funzione del tempo sarà quello di figura 5.3. Lo spazio percorso in accelerazione sarà uguale a quello percorso in

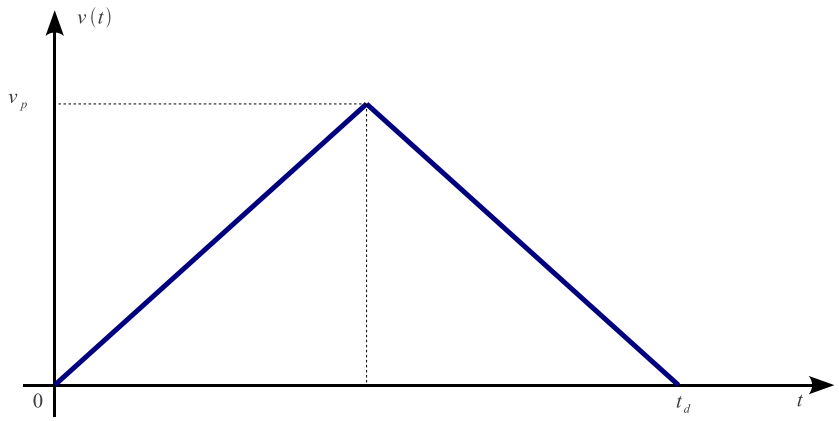

Figura 5.2: Andamento triangolare della velocità con massimo in v_p

decelerazione, pari a:

$$x = \frac{1}{2}\,a\,t^2 \qquad \Leftrightarrow \qquad \frac{x_T}{2} = \frac{1}{2}\,a\left(\frac{t_d}{2}\right)^2$$

Nel caso triangolare si avrà quindi:

$$t_d = 2\sqrt{\frac{x_T}{a}} \tag{5.3}$$

5.2.2 Caso trapezoidale

Indichiamo con ΔT_c il tempo percorso alla velocità costante v_{max}.

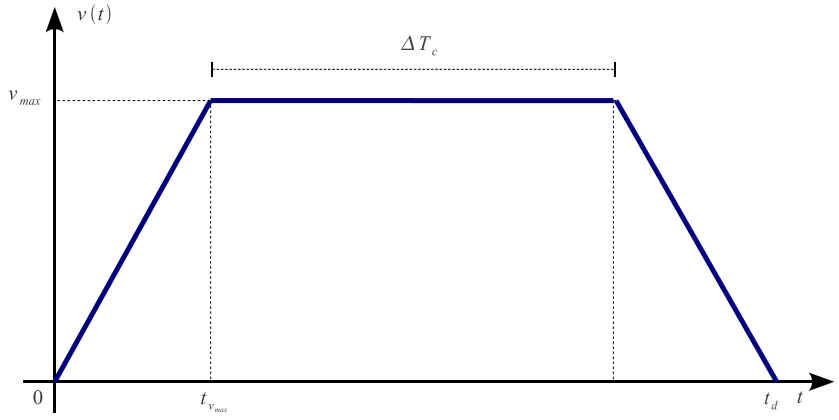

Figura 5.3: Andamento trapezoidale della velocità

Lo spazio x_T percorso è dato da:

$$x_T = x_{v_{max}} + v_{max}\,\Delta T_c + x_{v_{max}}$$

Sostituendo a $x_{v_{max}}$ l'espressione 5.1, si ottiene:

$$x_T = \frac{1}{2}\,\frac{v_{max}^2}{a} + v_{max}\,\Delta T_c + \frac{1}{2}\,\frac{v_{max}^2}{a} = \frac{v_{max}^2}{a} + v_{max}\,\Delta T_c$$

e da quest'ultima ricaviamo ΔT_c:

$$\Delta T_c = \frac{x_T - v_{max}^2/a}{v_{max}}$$

Quindi nel caso trapezoidale $t_d = 2\,t_{v_{max}} + \Delta T_c$, ovvero:

$$t_d = 2\,\frac{v_{max}}{a} + \sqrt{\frac{x_T - v_{max}^2/a}{v_{max}}} \tag{5.4}$$

Le 5.3 e 5.4 ci permettono pertanto di calcolare il tempo t_d necessario per compiere una elongazione x_T, rispettivamente nel caso di andamento triangolare e trapezoidale della velocità. A questo punto, se $t_d \leq t_R$ possiamo proseguire con l'algoritmo, mentre se $t_d > t_R$ non sarà possibile completare la richiesta con i parametri x_T e t_R dati.

5.3 Calcolo dei punti intermedi

Poichè dobbiamo compiere una movimentazione di un asse pari a x_T in un tempo t_R, e proprio in quel tempo, in generale non possiamo farlo con un movimento diretto, poichè l'azionamento arriverebbe in un tempo $t_d \leq t_R$ (arriverebbe in anticipo, a meno che non siamo nel caso $t_d = t_R$). Dobbiamo quindi capire come compiere la movimentazione esattamente nel tempo t_R. La soluzione che si adotterà qua consiste nel calcolare un insieme di punti intermedi, tali che l'azionamento non sosti in alcun punto e che il tempo totale di percorrenza sia t_R. Con riferimento alla figura 5.4, suddividiamo x_T in $N + 2$ intervalli tali che:

$$t_R = \Delta t_0 + N\Delta t + \Delta t_0 \tag{5.5}$$

Per ottenere l'andamento della velocità mostrato in figura 5.4 dobbiamo procedere come segue:

- all'istante t_0 viene comandato il posizionamento:

$$t_1 = t_0 + \Delta t + 2\Delta t_0 \quad \rightarrow \quad P_1 = x_0 + (\Delta x + 2\Delta x_0)$$

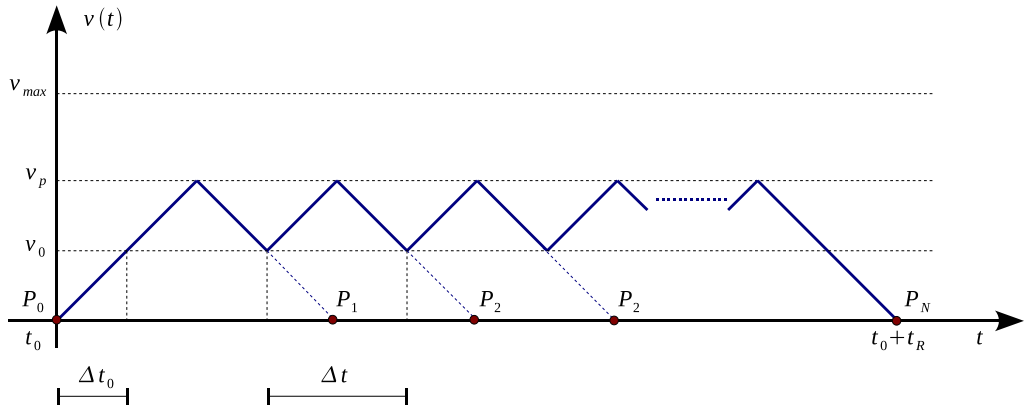

Figura 5.4: Suddivisione dell'intervallo temporale di elongazione

- all'istante $t_1 - \Delta t_0$ viene comandato il posizionamento:

$$t_2 = t_1 + \Delta t \quad \rightarrow \quad P_2 = P_1 + \Delta x$$

  e così via per il resto dei punti intermedi.

In questo modo è come se l'asse percorra lo spazio $N\Delta x$ alla velocità costante $v_0 + (v_p - v_0)/2$; l'andamento sarà tanto più uniforme quanto più vicine saranno v_p e v_0.

Cerchiamo adesso di capire come determinare v_0, Δx e quindi N. Poichè $v_0 = a\Delta t_0$, si avrà:

$$\Delta t_0 = \frac{v_0}{a}$$

e sostituendo Δt_0 nella 5.5 si ottiene:

$$\Delta t = \frac{t_R - 2v_0/a}{N} \tag{5.6}$$

Lo spazio Δx percorso nel tempo Δt, partendo da una velocità iniziale v_0 è dato da:

$$\Delta x = v_0\Delta t + \frac{1}{4}a\Delta t^2 \tag{5.7}$$

Poichè Δt deve essere positivo, dalla 5.6 si ottiene:

$$v_0 \leq \frac{at_R}{2} \tag{5.8}$$

Per quanto riguarda lo spazio complessivo x_T da percorrere, questo è dato da:

$$x_T = 2\Delta x_0 + N\Delta x$$

ed essendo:

$$\Delta x_0 = \frac{1}{2} a \Delta t_0^2 = \frac{1}{2} a \left(\frac{v_0^2}{a} \right)$$

otteniamo:

$$x_T = \frac{v_0^2}{a} + N \Delta x \tag{5.9}$$

Il sistema da risolvere è quindi il seguente:

$$\begin{cases} \Delta t = \dfrac{t_R - 2v_0/a}{N} \\ \Delta x = \dfrac{x_T - v_0^2/a}{N} \\ \Delta x = v_0 \Delta t + \dfrac{1}{4} a \Delta t^2 \end{cases} \tag{5.10}$$

Affinchè il modello funzioni, è necessario che la movimentazione che permette all'azionamento di trovarsi al tempo $t_0 + (m+1)\Delta t$ in posizione $x_0 + (m+1)\Delta x$ venga comunicata un pò prima del tempo $t_0 + m\Delta t$. L'intervallo temporale Δt vorremmo che fosse un parametro impostabile; cerchiamo quindi di capire queli siano i suoi estremi massimo e minimo in funzione della velocità v_0. Partiamo quindi dal sistema 5.10, sostituendo l'espressione di Δx della seconda equazione nella terza equazione:

$$\frac{x_T - v_0^2/a}{N} = v_0 \Delta t + \frac{1}{4} a \Delta t^2 \tag{5.11}$$

mentre dalla prima equazione si ottiene:

$$N = \frac{t_R - 2v_0/a}{\Delta t}$$

Sostituendo quest'ultima nella 5.11 si ottiene:

$$\frac{x_T - v_0^2/a}{t_R - 2v_0/a} \Delta t = v_0 \Delta t + \frac{1}{4} a \Delta t^2 \tag{5.12}$$

e sviluppando i calcoli otteniamo $\Delta t(v_0)$:

$$\Delta t = \frac{4}{a} \left(\frac{-v_0^2 + a t_R v_0 - a x_T}{2v_0 - t_R a} \right) \tag{5.13}$$

L'equazione 5.13 ci permette di individuare il range di valori ammissibili per Δt. Noto tale range e noto il valore da utilizzare per Δt, la 5.13 ammette come possibili valori per v_0:

$$v_{0_{12}} = \frac{a}{4}\left(2t_R - \Delta t \pm \sqrt{\frac{a\Delta t^2 + 4at_R^2 - 16x_T}{a}}\right) \tag{5.14}$$

È inoltre importante notare che la seconda equazione del sistema 5.10 deve rispettare il vincolo:

$$\Delta x \geq Risoluzione$$

e la velocità di picco ovviamente non deve superare la v_{max}

$$v_0 + a\frac{\Delta t}{2} \leq v_{max}$$

5.3.1 Dominio e segno della funzione

La 5.13 è definita per $v_0 \neq \frac{at_R}{2}$, ma questo punto di discontinuità l'avevamo già escluso in 5.8. La nostra funzione è allora definita nel seguente dominio:

$$\mathcal{D} = \left\{v_0 \in \mathbb{R} \mid 0 \leq v_0 < \frac{at_R}{2}\right\} \tag{5.15}$$

con l'ulteriore vincolo $v_0 < v_{max}$. Studiamo adesso il segno della funzione partendo dal segno del suo numeratore:

$$v_{0_{12}} = \frac{at_R \pm \sqrt{a^2t_R^2 - 4ax_T}}{2} \tag{5.16}$$

La seconda soluzione non può essere presa in considerazione perchè per la 5.8 siamo fuori dal dominio. Se $4ax_T > a^2t_R^2$ allora il numeratore è sempre negativo (la parabola non interseca l'asse v_0), mentre se $4ax_T < a^2t_R^2$ allora il numeratore sarà non negativo solo per:

$$\frac{at_R - \sqrt{a^2t_R^2 - 4ax_T}}{2} \leq v_0 < \frac{at_R}{2}$$

Il denominatore nel dominio è sempre negativo, per cui il segno della funzione per $4ax_T < t_R^2a^2$ è riportato in figura 5.5. Nel caso in cui $4ax_T \geq t_R^2a^2$ la funzione sarà sempre positiva.

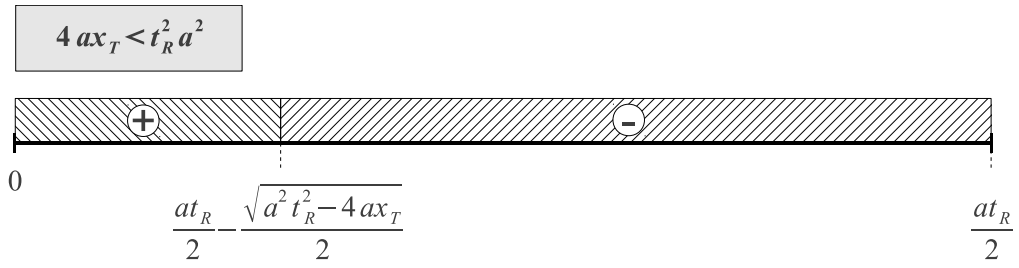

Figura 5.5: Segno della 5.13 per $4ax_T < t_R^2 a^2$

5.3.2 Punti di massimo e minimo

La derivata prima della 5.13 è data da:

$$\frac{d\,\Delta t}{d\,v_0} = \frac{-2\,v_0^2 + 2at_R\,v_0 + (2ax_T - a^2 t_R^2)}{(2v_0 - at_R)^2} \tag{5.17}$$

ed essendo il denominatore sempre positivo, studieremo il segno del solo numeratore. Gli zeri del numeratore sono:

$$\left(\frac{d\,\Delta t}{d\,v_0}\right)_{12} = \frac{at_R}{2} \pm \frac{\sqrt{4ax_T - a^2 t_R^2}}{2}$$

e l'unico ammissibile nel nostro dominio è il secondo. Inoltre tale punto è di minimo. Possiamo dire quindi che per $4ax_T > a^2 t_R^2$ la funzione ha un minimo relativo in corrispondenza di:

$$v_0 = \frac{at_R}{2} - \frac{\sqrt{4ax_T - a^2 t_R^2}}{2} \tag{5.18}$$

mentre per $4ax_T \leq a^2 t_R^2$ la funzione non ha punti di minimo.

5.3.3 Asintoti

Vediamo quali sono i valori assunti dalla funzione agli estremi del nostro dominio. Nell'origine si ha:

$$\Delta t(0) = \frac{4x_T}{at_R}$$

mentre al limite per $v_0 \to (at_R/2)^-$ si ha:

$$\lim_{v_0 \to \left(\frac{at_R}{2}\right)^-} \Delta t(v_0) = +\infty \qquad \text{per} \qquad 4ax_T > a^2 t_R^2$$

$$\lim_{v_0 \to \left(\frac{at_R}{2}\right)^-} \Delta t(v_0) = -\infty \qquad \text{per} \qquad 4ax_T < a^2 t_R^2$$

5.3.4 Valori massimi e minimi di Δt

Occupiamoci adesso di individuare i valori massimi e minimi di Δt in funzione di v_0, a seconda del segno assunto da $4ax_T - a^2t_R^2$. I grafici sono mostrati in figura 5.6. Il valore di v_{0_M} lo si ottiene eguagliando la 5.13 a $\Delta t(0)$:

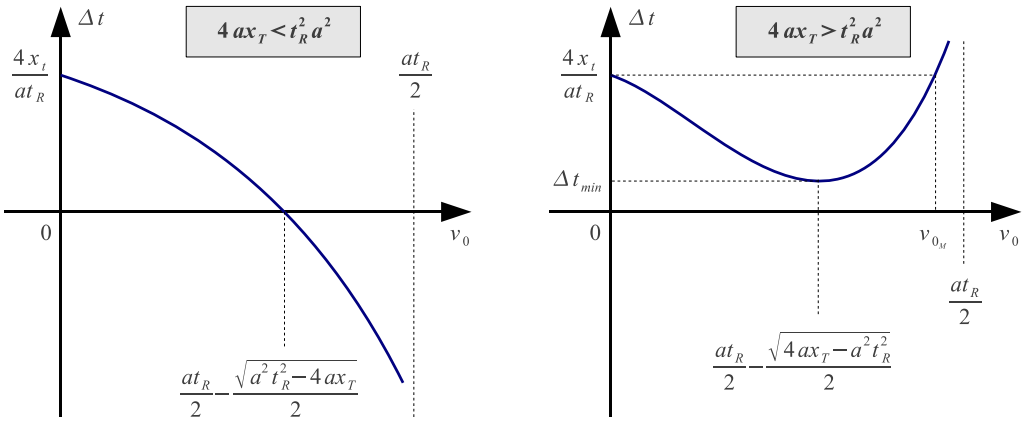

Figura 5.6: Grafici della 5.13

$$\frac{4x_T}{at_R} = \frac{4}{a}\left(\frac{-v_0^2 + at_R v_0 - ax_T}{2v_0 - t_R a}\right) \quad \Rightarrow \quad v_{0_M} = at_R - 2\frac{x_T}{t_R}$$

Il valore di Δt_{min} nel caso in cui $4ax_T > a^2t_R^2$ è dato da:

$$\Delta t_{min} = \frac{4}{a}\left[\frac{-\left(\frac{at_R}{2} - \frac{\sqrt{4ax_T - a^2t_R^2}}{2}\right)^2 + at_R\left(\frac{at_R}{2} - \frac{\sqrt{4ax_T - a^2t_R^2}}{2}\right) - ax_T}{2\left(\frac{at_R}{2} - \frac{\sqrt{4ax_T - a^2t_R^2}}{2}\right) - at_R}\right]$$

Sviluppando i calcoli si ottiene:

$$\Delta t_{min} = \frac{2}{a}\sqrt{4ax_T - a^2t_R^2} \qquad (4ax_T > a^2t_R^2) \tag{5.19}$$

Caso in cui $4ax_T < t_T^2a^2$

Le soluzioni possibili variano a seconda del valore di v_{max}. Per:

$$0 < v_{max} < \frac{at_R}{2} - \frac{\sqrt{a^2t_R^2 - 4ax_T}}{2}$$

otteniamo un range continuo di valori per Δt, compresi tra un valore minimo:

$$\Delta t_{min} = \frac{4}{a}\left(\frac{-(v_{max}^{-})^2 + at_R v_{max}^{-} - ax_T}{2v_{max}^{-} - at_R}\right) \tag{5.20}$$

ed uno massimo:

$$\Delta t_{max} = \frac{4x_T}{at_R}$$

Nella 5.20 si si è posto $v_0 = v_{max}^{-}$ per indicare un valore di v_0 prossimo a v_{max} ma inferiore, poichè dobbiamo rispettare il vincolo $v_0 < v_{max}$.

Nel caso in cui invece:

$$\frac{at_R}{2} - \frac{\sqrt{a^2t_R^2 - 4ax_T}}{2} \leq v_{max} < \frac{at_R}{2}$$

il massimo sarà sempre $\frac{4x_T}{at_R}$, mentre per il minimo avremo $\Delta t_{min} \to 0$.

Caso in cui $4ax_T > t_T^2 a^2$

Anche in questo caso le soluzioni possibili sono legate a v_{max}. Per:

$$0 < v_{max} < \frac{at_R}{2} - \frac{\sqrt{4ax_T - a^2t_R^2}}{2}$$

il minimo sarà:

$$\Delta t_{min} = \frac{4}{a}\left(\frac{-(v_{max}^{-})^2 + at_R v_{max}^{-} - ax_T}{2v_{max}^{-} - at_R}\right) \tag{5.21}$$

mentre il massimo:

$$\Delta t_{max} = \frac{4x_T}{at_R}$$

Per:

$$\frac{at_R}{2} - \frac{\sqrt{4ax_T - a^2t_R^2}}{2} \leq v_{max} \leq at_R - 2\frac{x_T}{t_R}$$

il massimo non cambia mentre il minimo sarà dato dalla 5.19.

Infine per $v_{max} > v_{0_M}$ il minimo sarà ancora dato dalla 5.19, mentre il massimo risulta:

$$\Delta t_{max} = \frac{4}{a}\left(\frac{-(v_{max}^{-})^2 + at_R v_{max}^{-} - ax_T}{2v_{max}^{-} - at_R}\right) \tag{5.22}$$

5.3.5 Algoritmo per la suddivisione dell'intervallo

Riassumiamo quanto visto in questa sezione per quanto riguarda i passi da compiere per suddividere l'intervallo x_T in sotto-intervalli:

1. la prima cosa da fare è individuare l'intervallo di valori ammissibili per Δt, seguendo quanto visto nella sezione 5.3. I casi sono due:
   - se il Δt impostato rientra all'interno del range ammissibile, allora si calcola v_0 funzione di questo Δt;
   - se Δt è fuori dal range, allora si scegli il valore di Δt più vicino a quello desiderato e si calcola il v_0 in funzione di questo Δt;
2. a questo punto si calcola in valore di N con la prima equazione del sistema 5.10 utilizzando il Δt desiderato o quello più prossimo a questo, e se ne prende la parte intera:
$$\lfloor N \rfloor = \left\lfloor \frac{t_R - 2v_0/a}{\Delta t} \right\rfloor$$
3. si calcolano i Δt e Δx effettivi, che saranno tipicamente più grandi (a causa del troncamento) rispetto a quelli ideali:
$$\Delta t = \frac{t_R - 2v_0/a}{\lfloor N \rfloor} \quad ; \quad \Delta x = \frac{x_T - v_0^2/a}{\lfloor N \rfloor}$$

Si potrebbe procedere in modo iterativo calcolando il v_0 in funzione del nuovo Δt, ma poichè il posizionamento sul punto $x_0 + (M+1)\Delta x$ viene comandato un pò prima che si arrivi al punto $x_0 + M\Delta x$, questa differenza dovuta al fatto che N deve essere intero viene compensata. A tal proposito si potrebbe far dipendere questo *anticipo di posizionamento* dalla differenza $N - \lfloor N \rfloor$.

5.4 Esempio: asse z del PFP

Si vuol muovere l'asse z del PFP di $z_T = 10\,mm$ in $t_R = 15\,sec$. I parametri dell'asse sono i seguenti:

- velocità massima $v_{max} = 5\,mm/sec$
- accelerazione $a = 5\,mm/sec^2$
- risoluzione $R = 1.6\,nm$.

Vogliamo avere inoltre un $\Delta t = 100\,ms$.

5.4.1 Tempo minimo impiegato per la movimentazione

Per poter individuare il tempo minimo dobbiamo stabilire se l'andamento della velocità nel tempo è triangolare o trapezoidale. Per far ciò calcoliamo v_{max}^2/a:

$$z_T = 10\,mm < \frac{v_{max}^2}{a} = 5\,mm$$

Abbiamo quindi un andamento trapezoidale ed il tempo impiegato per compiere la movimentazione risulta:

$$t_d = 2\,\frac{v_{max}}{a} + \sqrt{\frac{x_T - v_{max}^2/a}{v_{max}}} = 3\,sec < t_R$$

Possiamo quindi procedere nella ricerca dei punti intermedi. Come prima cosa valutiamo il segno di $4ax_T - t_R^2a^2$ per capire in quale dei due casi ci troviamo:

$$4ax_T = 200\,mm^2/sec^2 < t_T^2a^2 = 5625\,mm^2/sec^2$$

per cui siamo nel caso di sinistra di figura 5.6. Calcoliamo quindi lo zero della funzione Δt per capire se v_{max} si colloca alla sua sinistra o alla sua destra:

$$\frac{at_R}{2} - \frac{\sqrt{a^2t_R^2 - 4ax_T}}{2} = 0.673 < v_{max}$$

Ci troviamo quindi nel caso in cui sicuramente esiste un v_0 in corrispondenza del quale $\Delta t = 100\,ms$. Calcoliamo questo valore di v_0 utilizzando la 5.14:

$$v_{0_{12}} = \frac{a}{4}\left(2t_R - \Delta t \pm \sqrt{\frac{a\Delta t^2 + 4at_R^2 - 16x_T}{a}}\right) = \frac{5}{4}\,(29.9 \pm 29.462) \quad (5.23)$$

La nostra soluzione sarà quindi:

$$v_0 = 0.547\,mm/sec$$

Tenuto fisso questo valore di v_0 otteniamo:

$$\lfloor N \rfloor = \left\lfloor \frac{t_R - 2v_0/a}{\Delta t} \right\rfloor = \left\lfloor \frac{15 - 2\cdot 0.547/5}{0.1} \right\rfloor = 147$$

I valori effettivi di Δt e Δz saranno quindi:

$$\Delta t = \frac{t_R - 2v_0/a}{\lfloor N \rfloor} = 0.1005\,sec \quad ; \quad \Delta z = \frac{z_T - v_0^2/a}{\lfloor N \rfloor} = 0.0676\,mm$$

È inoltre verificata sia la condizione $\Delta z \geq Risoluzione$ che quella sulla velocità di picco:

$$v_p = v_0 + a\frac{\Delta t}{2} = 0.547 + 5\cdot\frac{0.1005}{2} = 0.798\,mm/sec < v_{max} = 5\,mm/sec$$

5.5 Calcolo dei punti intermedi in coordinate virtuali

Vediamo come calcolare i punti intermedi quando viene comandata una posizione virtuale, ovvero un'elongazione di un asse che non coincide con un asse fisico. Supponiamo che ci venga richiesta una movimentazione dell'asse virtuale x di una quantità x_T, dal punto x_0 al punto $x_0 + x_T$. Questa movimentazione coinvolgerà più assi fisici, per cui ciò che possiamo fare è applicare l'algoritmo discusso nella sezione 5.3 all'asse reale più critico, ovvero quello che impiega più tempo a compiere l'elongazione che gli compete. Ciò che dovremmo fare quindi sarebbe trasformare la movimentazione dell'asse virtuale x nelle corrispondenti elongazioni degli M assi reali, per poi calcolare il tempo impiegato da ogni asse reale per compiere la propria elongazione e applicare infine l'algoritmo all'asse che impiegherà più tempo. Se N è il numero di intervalli in cui verrà suddiviso tale asse reale, allora anche gli altri assi reali verranno suddivisi in N intervalli e le posizioni verranno comandatate ad istanti di tempo distanti Δt (i Δt calcolati per l'asse critico). In questo modo individueremo un insieme di $N + 1$ posizioni intermedie per gli assi fisici:

$$\begin{gathered} P_{10}(t_0 + \Delta t_0), P_{11}(t_0 + \Delta t_0 + \Delta t), \ldots, P_{1N}(t_0 + \Delta t_0 + N\Delta t) \\ \vdots \\ P_{M0}(t_0 + \Delta t_0), P_{M1}(t_0 + \Delta t_0 + \Delta t), \ldots, P_{MN}(t_0 + \Delta t_0 + N\Delta t) \end{gathered}$$

Ogni posizione va convertita nella corrispondente coordinata virtuale. Ad esempio, una volta calcolata la coordinata virtuale relativa al punto k, tale posizione virtuale andrà comandata al tempo $t_0 + \Delta t_0 + k\Delta t$.

Il grosso problema di questo metodo è che i punti virtuali intermedi non è detto che coinvolgano un solo asse, anzi generalmente coinvolgeranno più assi virtuali. Possiamo limitare il problema suddividendo l'asse virtuale in intervalli più piccoli per poi applicare il metodo sopra descritto ad ognuno di questi intervalli; in questo modo siamo sicuri che in tutti i punti in cui abbiamo suddiviso l'asse virtuale la posizione (rispetto a quella di partenza) cambierà solo per una elongazione dell'asse virtuale interessato.

Tanto più piccoli saranno gli intervalli in cui viene suddiviso l'asse virtuale e tanto più piccolo sarà l'errore commesso nel considerere la movimentazione a singolo asse.

Un altro vincolo potrebbe essere quello di richiedere che N sia un intero pari, in modo tale che nella curva di fuoco il punto intermedio centrale coincida con il punto rispetto quale vogliamo valutare gli scostamenti.

Bibliografia

[1] Franco Buffa. Cinematica del Subriflettore di SRT. Comunicazione privata.

[2] Franco Fiocchi. SMCU - Servo Minor Control Unit. Comunicazione privata.

[3] Marco Morsiani e GAI05. Prime Focus Positioner Control System (PFP-CS). GAI05 Memo Series.

[4] Marco Morsiani e GAI05. Sub Reflector Positioner Control System (SRP-CS). GAI05 Memo Series.

[5] M. Plesko, G. Chiozzi, e M. Sekoranja. ACS Supported BACI Types.

Elenco dei listati

Elenco delle figure

Indice analitico