UNIVERSITA' DEGLI STUDI DI PAVIA

FACOLTA' DI INGEGNERIA

DIPARTIMENTO DI INFORMATICA E SISTEMISTICA

REALIZZAZIONE DI UN SISTEMA ROBOTICO PER IL TIRO A VOLO DI BERSAGLI MOBILI

Relatore:
Chiar.mo Prof. G. Buttazzo

Correlatore:
Chiar.mo Prof. G. De Nicolao

Tesi di Laurea di
Tullio Facchinetti

Anno Accademico 1999/2000

Indice

Capitolo 1

Presentazione

1.1 Obiettivo della tesi 1
1.2 Organizzazione della tesi 2
1.3 Premessa al visual tracking 3

Capitolo 2

Descrizione del sistema

2.1 DEA: un sistema di puntamento automatico 5
2.2 Soluzioni progettuali 7
2.3 Cos'è il visual tracking 8
2.4 Sistemi real-time 11
2.4.1 DEA e HARTIK 12
2.5 Previsione della traiettoria 13
2.5.1 Sistemi dinamici e sistemi di controllo 13
2.5.2 Sistemi e modelli 16
2.5.3 Sistemi lineari dinamici 17
2.5.4 Il modello del target 19
2.5.5 Il filtro di Kalman 22

2.6 Elaborazione dell'immagine 24
2.6.1 Informazioni sulle dimensioni degli oggetti 26
2.7 Tempo di volo e ritardo di attivazione della valvola 27
2.7.1 La regressione lineare 28
2.8 La rete neurale 31

Capitolo 3

L'hardware di DEA

3.1 L'apparato hardware 32
3.2 Il sistema di visione 33
3.2.1 La telecamera 34
3.2.2 Il frame-grabber 35
3.3 I servomotori 36
3.3.1 Protezione di un servocomando 38
3.3.2 Controllo di un servo 40
3.3.3 Imporre una curva di funzionamento al motore 42
3.4 Il puntatore laser 43
3.4.1 Il diodo laser 45
3.5 Dispositivo ettropneumatico 47
3.6 La scheda National Instruments PCI-6025E 48
3.6.1 Linee digitali di I/O 50
3.6.2 Timer general purpose 50
3.6.3 Misurazione dell'ampiezza di un impulso 52
3.6.4 Generazione di una onda quadra 54
3.7 Interfacciamento scheda-hardware 56
3.7.1 Alimentazione e massa 59
3.8 Il robot a 2 gradi di libertà 60

3.9 Compressore d'aria e proiettili 63
3.10 Stima dei ritardi 64
3.11 L'elaboratore 68

Capitolo 4

Reti neurali

4.1 Le reti neurali 70
4.2 Cos'è una rete neurale 70
4.3 Cosa può fare una rete neurale (e cosa no!) 72
4.4 Tipologie di reti neurali 73
4.5 Reti Error Backpopagation (EBP) 74
4.5.1 Funzione di trasferimento del neurone 76
4.5.2 Algoritmo di apprendimento 78
4.5.3 La generalized delta rule 79
4.5.4 Retropropagazione dell'errore 81
4.6 La rete EBP utilizzata in DEA 83
4.7 Test su reti backpropagation 86
4.7.1 Parametri testati durante le prove 88
4.8 Risultato dei test 88
4.8.1 Test sul β delle sigmoidi 89
4.8.2 Determinazione dei pesi iniziali 90

Capitolo 5

Architettura software

5.1 Idea di base 91

5.2 Sequenza esecutiva 93
5.3 I data files 95
5.3.1 Punti perimetrali dell'area di scansione 97
5.3.2 Area di scansione 98
5.3.3 Guadagni per il filtro di Kalman 99
5.3.4 Inizializzazione del filtro di Kalman 100
5.3.5 Caratteristiche del puntatore laser 101
5.3.6 Costanti di pilotaggio dei motori 101
5.3.7 Parametri per il training dell'apparato neurale 102
5.3.8 Caratteristiche del target 104
5.3.9 Inizializzazione del modulo di tracking 104
5.3.10 Training set 105
5.3.11 Pesi e soglie della rete neurale 106
5.4 Moduli esecutivi 107
5.4.1 CAMERA 108
5.4.2 Perimeter finding 111
5.4.3 Training set generation 114
5.4.4 Neural network training 115
5.4.5 Tracking and shooting 116
5.4.6 Real-time Kalman simulation 119
5.4.7 Velocità del proiettile e ritardo dell'elettrovalvola .. 122
5.5 Moduli esecutivi e data file 123
5.6 Implementazione delle funzioni 126
5.6.1 elabor 127
5.6.2 general 128
5.6.3 ioni 128
5.6.4 kc 131
5.6.5 neural 134
5.6.6 output 136

5.6.7 pci6025e 139
5.6.8 pxc 140
5.6.9 kalman 141
5.7 Software MATLAB 149

Capitolo 6

Risultati

6.1 Velocità del proiettile e ritardo della valvola 151
6.2 Calibrazione 154
6.3 Test di sparo 155
6.3.1 Test A 156
6.3.2 Test B 157
6.3.3 Test C 157
6.3.4 Test D1 159
6.3.5 Test D2 160
6.3.6 Test D3 160
6.3.7 Test E1 161
6.3.8 Test E2 162
6.3.9 Test F1 163
6.4 Conclusioni 164

Appendice A

HARTIK

A.1 Cos'è HARTIK 166
A.2 Il real-time clock 168

A.3 Avvio del sistema 168
A.4 Creazione dei task 169
A.5 Accesso ai file 170
A.6 Risorse condivise 171
A.6.1 Stack Resource Policy 172
A.6.2 Cyclic Asynchronous Buffer 173
A.7 Grafica 174

Appendice B

Librerie di funzioni

B.1 elabor 176
B.2 general 179
B.3 ioni 180
B.4 kc 182
B.5 neural 183
B.6 output 190
B.7 pci6025e 192
B.8 pxc 198
B.9 kalman 199

Appendice C

Codice MATLAB

C.1 Soluzione dell'equazione di Riccati 204
C.2 Simulazione del predittore di Kalman 206
C.3 Generazione dei guadagni 212

Bibliografia

..... 215

Capitolo 1

Presentazione

1.1 Obiettivo della tesi

Il seguente lavoro di tesi ha l'obiettivo di progettare e realizzare un sistema di puntamento in grado di agganciare la traiettoria di un oggetto in movimento, predirne il moto e colpirlo. Il sistema è stato progettato per funzionare autonomamente, senza l'intervento di un operatore, anche per quanto riguarda la fase di calibrazione dei parametri che ne determinano il funzionamento. Il software è stato organizzato in modo modulare come un insieme di librerie, così da permettere il riutilizzo futuro delle soluzioni proposte senza apportare modifiche sostanziali al codice già scritto.

In parallelo allo sviluppo del progetto sono stati eseguiti numerosi test su reti neurali di tipo *Backpropagation* (utilizzate per la calibrazione del sistema visivo), con lo scopo di caratterizzarne in modo qualitativo il comportamento in base alla variazione di alcuni parametri caratteristici.

1.2 Organizzazione della tesi

L’esposizione del lavoro di tesi è organizzata in capitoli e paragrafi che suddividono e raggruppano logicamente i diversi aspetti del progetto.

Di seguito viene elencato il contenuto di ciascun capitolo:

- ***Capitolo 1*:** introduzione, descrizione degli obiettivi della tesi e considerazioni generali sul progetto.
- ***Capitolo 2*:** descrizione dei diversi aspetti teorici che è stato necessario sviluppare per la realizzazione del progetto. In particolare si offre una panoramica su argomenti quali il visual tracking, l’elaborazione dell’immagine, la modellizzazione e l’osservazione di sistemi dinamici.
- ***Capitolo 3*:** presentazione delle soluzioni costruttive adottate, realizzazione del sistema elettromeccanico di puntamento e dei circuiti elettronici di interfacciamento e descrizione dell’elaboratore utilizzato e dei dispositivi progettati per effettuare i test sul tempo di volo della pallina e sul ritardo di attivazione dell’elettrovalvola.
- ***Capitolo 4:*** vengono trattate nel dettaglio le reti neurali di tipo error backpropagation, sulle quali sono stati condotti dei test allo scopo di determinarne alcuni comportamenti caratteristici.
- ***Capitolo 5*:** trattazione del progetto software, pilotaggio dell’hardware per mezzo di driver appositi e architettura della libreria di funzioni. In generale verrà descritta l’implementazione degli argomenti di teoria introdotti nel capitolo 2.
- ***Capitolo 6:*** risultato delle prove sul sistema completo, con svariati test di sparo a diverse distanze. Inoltre vengono discusse le conclusioni derivate dall’esperienza diretta di utilizzo del sistema.
- ***Appendice A*:** illustrazione delle caratteristiche del sistema operativo real-time HARTIK, limitatamente alle funzionalità utilizzate per lo sviluppo del progetto.

- ***Appendice B*:** elenco completo delle funzioni contenute nelle diverse librerie. Costituisce una sorta di manuale utente per un successivo utilizzo delle librerie stesse.
- ***Appendice C:*** listati e relative descrizioni dei programmi realizzati in ambiente MATLAB, per la simulazione del filtro di Kalman e la generazione dei guadagni precalcolati.

1.3 Premessa al visual tracking

Il tracking e l'intercettazione di un oggetto in movimento possono essere effettuati utilizzando le più svariate tecniche, da scegliersi in base ai requisiti specifici dell'applicazione da sviluppare.

In generale, per svolgere un compito particolare, sono spesso disponibili tecnologie e approcci diversi, talvolta equivalenti. Tuttavia, non è raro il caso in cui si è costretti ad utilizzarne uno e soltanto quello risolve correttamente e in modo semplice il problema. Spesso, quindi, si presenta il problema di dover scegliere tra metodi diversi, i quali garantiscono il medesimo risultato finale, ma che possono differenziarsi tra loro per semplicità, velocità, necessità di risorse oppure robustezza: è valutando questi parametri che, in relazione all'obiettivo del progetto, è possibile individuare la soluzione migliore.

Questo preambolo vuole spostare l'accento sul fatto che le soluzioni proposte e adottate in questo lavoro di tesi non mirano ad essere l'ultimo grido nei rispettivi campi di ricerca, quanto piuttosto delle soluzioni valide, robuste e possibilmente semplici, ai singoli problemi che devono essere affrontati per raggiungere l'obiettivo preposto.

Vorrei sottolineare l'importanza, per me fondamentale, della semplicità dei singoli mattoni che vanno a costituire un progetto complesso. Semplicità significa guadagno in termini di chiarezza, velocità di sviluppo, manutenibilità, modularità e facilità di utilizzo, tutti aspetti che diventano tanto più importanti quanto più

aumentano le dimensioni di un progetto. E' anche vero che il problema opposto, ovvero l'eccessiva semplicità, può, introdurre a sua volta dei problemi, legati principalmente all'inadeguatezza delle soluzioni adottate.

Nel presente progetto sono dunque state integrate metodologie proprie di svariati settori dell'ingegneria. Lo sviluppo del progetto ha comportato da un lato la costruzione del dispositivo meccanico di puntamento e dei circuiti elettronici ed elettromeccanici necessari a controllarlo, dall'altro la scrittura del software, per il quale sono state implementate tecniche di elaborazione dell'immagine, tecniche di controllo dei dispositivi hardware mediante driver appositamente progettati, algoritmi di intelligenza artificiale, in particolare nell'ambito delle reti neurali, e tecniche di controllo e monitoraggio di sistemi dinamici, il tutto sotto il controllo di un sistema operativo real-time in grado di garantire le specifiche temporali necessarie.

Capitolo 2

Descrizione del sistema

2.1 DEA: un sistema di puntamento automatico

La soluzione proposta in questa tesi consiste in un dispositivo elettro-meccanico a 2 gradi di libertà rotazionali attuato da due servomotori, controllati da una rete neurale multistrato, addestrata con il metodo di Error Backpropagation.

Il tracking dell'oggetto da colpire viene effettuato grazie ad un predittore di Kalman, che lavora sui dati forniti da un sotto-processo di elaborazione grafica, il quale individua la posizione dell’oggetto a partire dalle immagini catturate da una telecamera fissa. Il software è gestito da un sistema operativo real-time, che si occupa di garantire i vincoli temporali necessari alle diverse operazioni. Uno schema esemplificativo della struttura del sistema è riportato in figura 2.1.

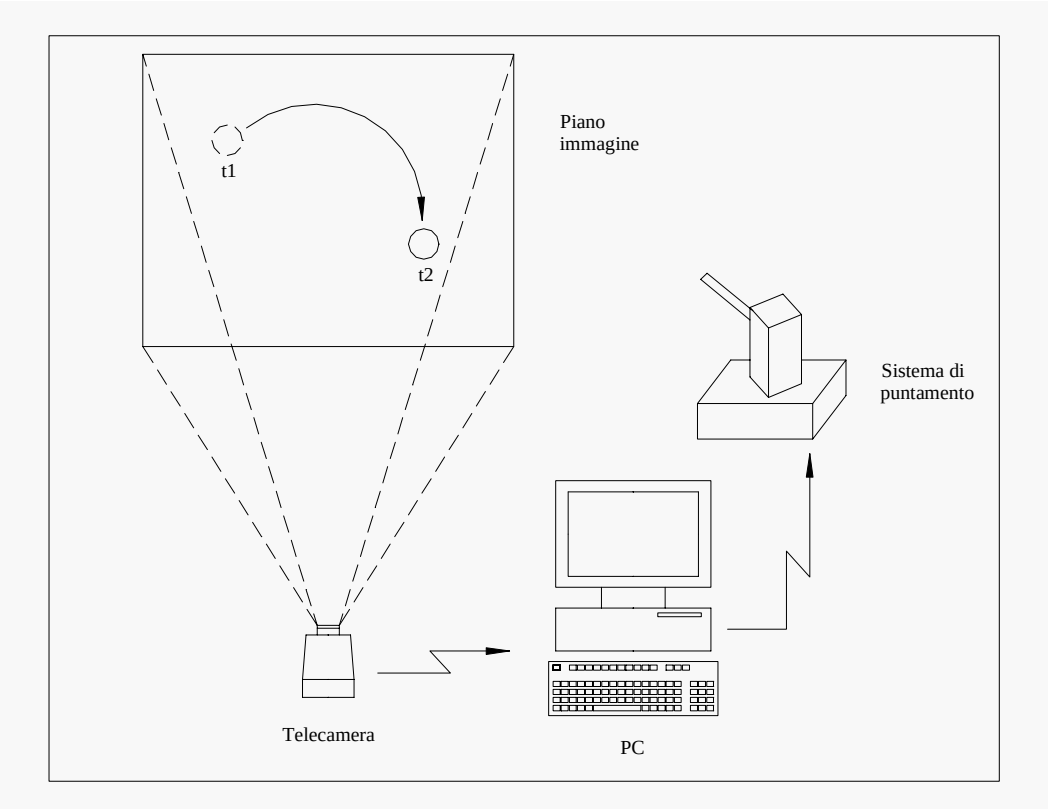


Figura 2.1. Componenti del sistema.

I sistemi di puntamento sono di due tipi: un puntatore laser utilizzato durante la fase di calibrazione automatica ed una “cerbottana” ad aria compressa, a colpo singolo e caricamento manuale, utilizzata per colpire il target.

La calibrazione è da effettuarsi ogni qualvolta avviene un cambiamento nelle condizioni operative del sistema (per esempio, quando la telecamera o il robot vengono spostati). Una volta effettuata la fase di taratura dei parametri è possibile concentrarsi esclusivamente sulla fase di tracking.

Il sistema è stato denominato DEA, ovvero Dexterous Eye-guided Aiming.

2.2 Soluzioni progettuali

Le scelte di progetto che maggiormente influiscono sul funzionamento e sulla struttura del sistema sono le seguenti:

- utilizzare dei servomotori per muovere il sistema di puntamento;
- sfruttare una rete neurale per controllare i motori stessi;
- riprendere le immagini con una sola telecamera fissa;
- lavorare con immagini a toni di grigio;
- utilizzare il predittore di Kalman per stimare e predire il moto del target.

L'utilizzo di servoattuatori per il movimento del robot ha il vantaggio di un pilotaggio molto semplice, come si avrà modo di illustrare al capitolo 3.3. La precisione del posizionamento è invece relativamente limitata dalle caratteristiche dei motori utilizzati, poiché si tratta di servocomandi di basso costo costruiti per applicazioni di modellismo.

La scelta di utilizzare una sola telecamera per l'osservazione dell'ambiente si traduce con l'impossibilità di stimare la distanza dell'oggetto da colpire e, quindi, è necessario assumere che l'oggetto si muova in un piano. Sostanzialmente, con una sola telecamera è impossibile ricostruire le caratteristiche tridimensionali di una scena, e quindi anche del moto dell'oggetto. Sempre riguardo all'apparato di visione, si è scelto di utilizzare una telecamera fissa. In questo modo l'area di movimento dell'oggetto è limitata dal campo visivo dell'obiettivo montato sulla telecamera: è ovvio che se il target non viene ripreso dalla telecamera, in quanto si muove al di fuori dell'area di visione, è impossibile predirne la traiettoria!

La soluzione di adottare delle reti neurali per immagazzinare la conoscenza necessaria al pilotaggio dei motori porta con sé delle interessanti conseguenze. Si può pensare che, modificando il sistema per migliorarne le prestazioni (ad esempio con la visione stereoscopica e il movimento delle telecamere), è possibile continuare ad utilizzare una rete neurale per muovere i motori. La nuova rete

neurale dovrebbe prevedere, come ingressi, la posizione angolare dei motori che movimentano le telecamere oltre a quella dei motori del dispositivo di puntamento: teoricamente il concetto di mappare lo spazio immagine (di una o più telecamere) nello spazio delle variabili di pilotaggio dei motori rimane invariato. Ciò che cambia è la complessità della rete, oltre all'algoritmo di generazione del training set.

L'aver optato per delle immagini a toni di grigio limita le possibilità di elaborazione dell'immagine necessaria per individuare il target nell'immagine stessa. Per trovare i pixel appartenenti al bersaglio viene fatta una semplice soglia sull'intensità dei punti, quindi è necessario che l'oggetto abbia un colore che lo separi nettamente dallo sfondo. Uno svantaggio di questo approccio è che la soglia dipende fortemente da diverse condizioni, come ad esempio dalle condizioni di illuminazione dell'ambiente, per cui va attentamente calibrata in fase di tuning del sistema. Questa scelta si è resa necessaria in quanto, tenuto conto delle caratteristiche dell'hardware a disposizione, una più complessa elaborazione dell'immagine sarebbe risultata troppo pesante dal punto di vista del tempo di calcolo richiesto.

Infine, per stimare la traiettoria dell'oggetto da colpire, si è scelto di utilizzare il filtro di Kalman, che rappresenta una soluzione robusta ed efficace al problema di stimare e prevedere lo stato di un oggetto in presenza di rumore sulle misurazioni. Questo fa del filtro di Kalman uno dei metodi più diffusamente usati per questo tipo di applicazioni.

2.3 Cos'è il visual tracking

Per visual tracking si intende, in generale, l'insieme di operazioni necessarie ad agganciare, seguire e prevedere la traiettoria (tracking) di un oggetto in movimento a partire dalle informazioni ricavate dall'elaborazione di immagini ricevute da uno o più sensori ottici.

Esistono numerosi sistemi per il tracking visivo, che si differenziano tra di loro principalmente in base alla componentistica hardware e alle strategie adottate per il riconoscimento del movimento e della traiettoria del target. Un esempio tipico di sistema atto a questo scopo è costituito da una o più telecamere, fisse o mobili, che forniscono una sequenza di immagini all'unità di elaborazione, la quale, eseguendo una serie di operazioni più o meno complesse, prima fra tutte l'elaborazione delle immagini per ricavarne una opportuna descrizione della scena, è in grado di riconoscere l'oggetto da seguire e computare i vari parametri relativi al movimento dell'oggetto stesso.

Gli algoritmi che sono stati sviluppati nel corso degli anni per effettuare il tracking di uno o più oggetti ricadono in una delle seguenti categorie:

- algoritmi basati su opportuni modelli della scena; essi riconoscono in quest'ultima delle caratteristiche note e le utilizzano per aggiornare la conoscenza sull'oggetto da seguire. Alcune tecniche sono basate su modelli geometrici prefissati oppure su "snake" deformabili.
- i "blob" trackers, che definiscono l'oggetto di interesse in modo da poterlo isolare all'interno della scena, considerando il resto come sfondo. I più semplici algoritmi di questa categoria, basati spesso sulla segmentazione, necessitano di uno sfondo immobile.
- algoritmi basati sulla misurazione diretta della velocità dell'oggetto nell'immagine; questa tecnica è basata, ad esempio, sulla valutazione del flusso ottico e della correlazione tra immagini successive.
- algoritmi basati sul tracciamento di caratteristiche comuni ad una sequenza di immagini, che, in genere, consistono in punti o linee; tali algoritmi si basano spesso su ipotesi di rigidità, al fine di garantire la consistenza globale del risultato.

I sistemi per il visual tracking possono essere divisi in base alle caratteristiche su cui basano l'individuazione dello stato proprio dell'ambiente circostante. In particolare si hanno due tipi di sistemi:

- basati sulla ricostruzione dell'ambiente tridimensionale nel quale si muove l'oggetto (o gli oggetti) a partire dai dati ricavabili dai sensori;
- basati sul mapping dello spazio reale nello spazio immagine. Questo tipo di sistemi, al quale appartiene anche DEA, non effettuano alcuna ricostruzione dell'ambiente fisico reale, ma rappresentano la propria conoscenza in termini di coordinate dei pixel delle immagini riprese da una o più telecamere.

Gli utilizzi del visual tracking, inteso come il tracciamento di una traiettoria di un oggetto, sono quanto mai vari: si può essere interessati a individuare il punto in cui il target si troverà in un certo istante per colpirlo o afferrarlo; può essere utile conoscere il momento esatto in cui il target si troverà in un determinato punto, nel caso si abbiano informazioni sulla traiettoria, come capita in svariate applicazioni in campo astronomico.

Esistono diversi sistemi che effettuano il visual tracking. In [1] viene descritto *Mousebuster*, un apparato che sfrutta il tracking dell'oggetto per comandare opportunamente il braccio di un robot, con lo scopo di afferrarlo. In [4] e [5] gli autori trattano sotto due diversi aspetti il problema del tracking in un ambiente dove ci siano variazioni nel livello illuminazione oppure nella geometria dell'oggetto da seguire, dovute a rotazioni o alla stessa variazione dell'illuminazione. In [6] viene descritto *X Vision*, un framework ideato per consentire di sviluppare in modo semplice e veloce complesse applicazioni basate sulla visione. In [7] gli autori tentano di individuare dei punti in comune a diversi ma correlati aspetti dell'elaborazione delle immagini: la *ricerca visuale*, definita come la ricerca di oggetti in un'immagine aventi determinate caratteristiche; la *segmentazione*, cioè il raggruppamento di pixel che, in una immagine, sono caratterizzati da aspetti comuni; il *tracking*, inteso come il problema di seguire il moto o le deformazioni di un oggetto in una sequenza di immagini. [8] rappresenta un'introduzione ai concetti di base del tracking. In [11] gli autori applicano una rete neurale auto-organizzante al problema di prevedere la

traiettoria di un oggetto. Infine, in [17], viene descritto un sistema di visual tracking basato sull'assunzione che, nell'unità di tempo, è meglio eseguire molte operazioni semplici piuttosto che poche e sofisticate funzioni.

2.4 Sistemi real-time

Nei sistemi di calcolo real-time la correttezza di una computazione non è dovuta soltanto al valore restituito, ma anche all'istante in cui tale valore è reso disponibile.

Talvolta, erroneamente, si identificano come sistemi real-time dei sistemi veloci nel compiere una determinata azione, come calcolare un valore o controllare un dispositivo; in questo caso l'ambiguità nasce dall'uso del termine "veloce", alquanto relativo.

Altri sistemi ritenuti, a torto, real-time, sono quelli che tendono a minimizzare il tempo medio di esecuzione dei vari processi di calcolo. Piuttosto, un sistema real-time deve essere in grado di garantire determinati vincoli temporali ai processi di calcolo in esso eseguiti (i *task*). I un sistema real-time, in particolare, a ciascun task sarà associata una *deadline*, cioè un vincolo sul tempo massimo di completamento dell'esecuzione del task stesso.

Questo differenzia i sistemi real-time da semplici sistemi timesharing, per quanto complessi possano essere. In questi ultimi, al massimo, si è in grado di gestire vincoli di priorità tra i task, ma non di garantire l'esecuzione di un processo rispetto ad altri tipi di vincoli temporali. La caratteristica fondamentale di un sistema real-time può essere considerata la *prevedibilità*, che può essere vista come la proprietà di poter stabilire, al momento dell'attivazione, se un processo sarà in grado di rispettare i vincoli temporali necessari.

Quando un sistema deve gestire l'esecuzione concorrente di diversi processi nascono una serie di problemi. I più importanti si possono riassumere in:

- decisione della strategia di assegnazione delle risorse di calcolo ai processi;

- scambio di dati tra i task;
- gestione opportuna delle situazioni di sovraccarico.

Assegnare le risorse di calcolo significa sostanzialmente eseguire il codice di un processo piuttosto che quello di un altro sul processore (o sui processori) a disposizione. Molto semplicisticamente, questo è ciò che si dice “schedulare” un insieme di processi.

Gli algoritmi di schedulazione più famosi ed utilizzati in ambienti real-time, utilizzati per trattare processi periodici, sono Rate Monotonic (RM) ed Earliest Deadline First (EDF), la cui differenza principale risiede nel fatto che, mentre il primo stabilisce la schedulazione in base al valore del periodo dei task, quindi genera una sequenza predeterminata di attivazioni, in quanto viene sempre eseguito il task con periodo più corto per primo, EDF privilegia i task con deadline più imminente, che sono sostanzialmente quelli per i quali il rischio di superare la deadline è maggiore.

2.4.1 DEA e HARTIK

DEA è stato sviluppato utilizzando il sistema operativo real-time HARTIK (HArd Real-TIme Kernel). Si tratta di un sistema basato sull’algoritmo di schedulazione EDF; supporta task con criticità soft, hard e non real-time, sia periodici che aperiodici; gestisce i vincoli su risorse mediante l’algoritmo Stack Resource Policy (SRP). Questi ed altri argomenti inerenti i sistemi real-time sono efficacemente illustrati in [2].

Il linguaggio di programmazione è il C classico, con il quale sono stati implementati il kernel e le librerie di funzioni. HARTIK mette a disposizione tutte le primitive fondamentali per lavorare in un vero ambiente di programmazione real-time.

La caratteristica più apprezzabile di HARTIK è la semplicità di utilizzo, rapportata al rigore degli algoritmi utilizzati nel kernel: in molte applicazioni è

possibile trascurare completamente i dettagli di tali algoritmi, a tutto vantaggio della rapidità di sviluppo.

Le funzioni di HARTIK utilizzate per lo sviluppo di DEA sono descritte in dettaglio in appendice A.

2.5 Previsione della traiettoria

Nei paragrafi seguenti si introdurranno brevemente i concetti fondamentali della teoria dei sistemi, con l'obiettivo di fornire le basi per la comprensione del funzionamento del predittore basato sulla teoria sviluppata dal dottor Rudolf E. Kalman intorno ai primi anni '60, e che può essere considerata una dei maggiori contributi del secolo nel campo della stima dei sistemi dinamici.

Il predittore di Kalman è, al giorno d'oggi, uno dei metodi più comunemente usati per stimare le variabili di stato di un sistema dinamico in presenza di errori statisticamente descrivibili sui valori delle variabili osservate. Questo poichè costituisce un metodo semplice e soprattutto robusto, in quanto poggia le proprie basi su una solida trattazione matematica.

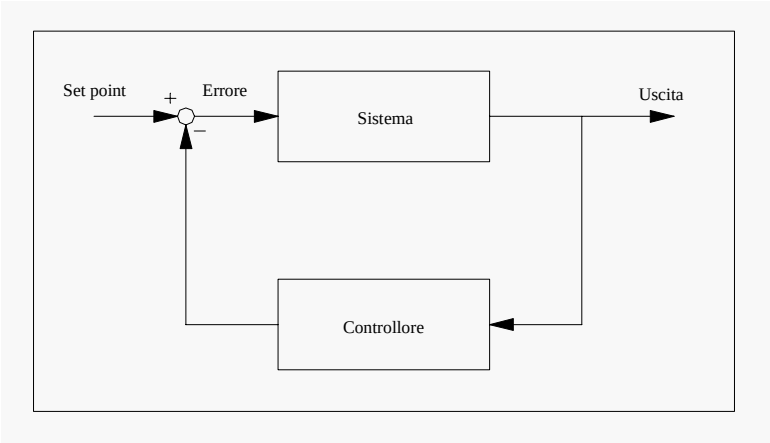
Il predittore di Kalman viene utilizzato da DEA per stimare la traiettoria dell'oggetto in movimento e calcolarne la posizione ad un determinato istante, in modo da posizionare correttamente il sistema di fuoco e sparare al momento opportuno.

2.5.1 Sistemi dinamici e sistemi di controllo

Per *sistema* si intende correntemente un complesso costituito da più elementi interagenti, e può descrivere un oggetto fisico di natura e di complessità qualsiasi: da un semplice interruttore, ad un serbatoio idraulico, al processo di crescita di una coltura vegetale, agli impianti di produzione, trasporto e distribuzione di energia elettrica, ad un qualsiasi oggetto in movimento.

In un sistema interagiscono fra di loro diverse grandezze fisiche (dette anche variabili o segnali) fra le quali è opportuno distinguere quelle di *ingresso* da quelle di *uscita*. Si parla in questo caso di *sistemi orientati*. In questi termini, realizzare un sistema di controllo significa predisporre una struttura (detta *controllore* o *regolatore*) in grado di generare automaticamente le opportune variabili di ingresso, in modo che le variabili di uscita di interesse si comportino nel modo desiderato.

Figura 2.2. Schema di un regolatore in anello chiuso.



I sistemi di controllo si dividono fondamentalmente in due categorie: sistemi ad *anello aperto* e sistemi ad *anello chiuso*, a seconda della diversa azione del controllo, cioè di quella entità che agisce sul sistema per produrre l'uscita desiderata. In particolare:

- si parla di sistema di controllo in anello aperto quando l'azione di controllo è indipendente dalle variabili di uscita;
- si parla di sistema di controllo in anello chiuso nel caso in cui l'azione di controllo dipende, in qualche modo, dall'uscita.

I controllori in anello chiuso, che assicurano in generale delle prestazioni complessivamente migliori rispetto a quelli in anello aperto, sono anche detti sistemi con retroazione (feedback), essendo questa intesa come l'azione con cui

dall'uscita, attraverso un confronto con una grandezza chiamata riferimento, si agisce sul controllo. Questa struttura, schematizzata in figura 2.2, caratterizza la maggior parte dei sistemi di controllo anche non automatici.

I sistemi di controllo, così come l'automazione in senso più lato, riguardano ormai i più svariati settori di attività, laddove è necessario sostituire l'intervento umano o migliorarne l'efficienza. Così la teoria dei Controlli Automatici trova applicazione non solo in tutte le aree classiche dell'ingegneria, ma anche in settori quali la biologia, l'ecologia, l'economia, le scienze sociali, ecc.

Risulta pertanto difficile una classificazione dei sistemi che faccia riferimento alla natura dei processi in gioco. Viceversa, considerando l'andamento nel tempo delle grandezze da controllare, si possono distinguere i seguenti casi:

1) regolare il sistema in modo da mantenere costante una determinata grandezza;
2) imporre un determinato andamento alle variabili di uscita del sistema;
3) stimare lo stato di un sistema a partire dall'osservazione delle variabili di ingresso e di uscita.

Lo svolgimento tipico di un problema concernente lo sviluppo di un sistema di controllo avviene dapprima attraverso una fase di *analisi*, nella quale si rappresenta il sistema fisico da controllare mediante un opportuno *modello matematico*, che serve a valutare le più importanti proprietà dinamiche dell'impianto in studio, seguita dalla realizzazione di un opportuno schema di regolazione.

2.5.2 Sistemi e modelli

Dato un sistema, il passo fondamentale da eseguire, sia per capirne il funzionamento che per progettare un eventuale sistema di controllo, consiste nella costruzione del suo *modello matematico*.

Per modello matematico di un sistema si intende una struttura astratta (in genere un insieme di equazioni) capace di descrivere con sufficiente accuratezza il comportamento del processo reale in studio.

Il modello matematico deve avere un opportuno *grado di complessità*, che risulta dal compromesso fra due esigenze:

1) riprodurre con sufficiente fedeltà i fenomeni essenziali cui dà luogo il sistema reale;
2) essere abbastanza semplice da permettere una sua efficace utilizzazione.

In genere, è impensabile la determinazione di un modello capace di descrivere esattamente il comportamento del sistema reale in esame, pertanto si dovrà stabilire un opportuno *livello di approssimazione*, in modo che il modello risponda alle esigenze dell'applicazione da sviluppare. Ad un modello non corretto (o non sufficientemente corretto) possono infatti fare seguito una errata comprensione del comportamento reale del sistema, e quindi il progetto di sistema di controllo che non soddisfa le specifiche. Il livello di approssimazione del modello condiziona i passi successivi di sviluppo del controllore, in quanto l'incertezza introdotta da un modello troppo semplificato rende inutile l'uso di tecniche sofisticate per il controllo o il monitoraggio del sistema.

Alcune possibili classificazioni delle principali caratteristiche dei modelli matematici utilizzabili per descrivere i sistemi reali sono le seguenti:

- *modelli lineari e non lineari*, a seconda che le variabili di ingresso e uscita siano legate tra loro da relazioni lineari o meno. Per i sistemi lineari è valido il noto principio di sovrapposizione degli effetti.
- *modelli statici o dinamici*, a seconda che si colleghino tra loro solo le variabili di ingresso e di uscita (relazioni algebriche) oppure anche le loro derivate (relazioni differenziali).
- *modelli continui o discreti*, a seconda che la variabile indipendente (che corrisponde al tempo) vari in maniera continua oppure discreta.

Tenendo presente che altri tipi di classificazioni sono possibili, nel seguito si tratteranno sistemi dinamici lineari tempo-invarianti con una sola variabile di

ingresso e una sola di uscita, in quanto sono sufficienti per la descrizione delle tecniche utilizzate nello sviluppo di DEA.

2.5.3 Sistemi lineari dinamici

I modelli dei sistemi lineari dinamici sono rappresentati in forma generale dall'*equazione ingresso-uscita*, che nella sua forma generale vale

$$\sum_{i=0}^{n} a_i \frac{d^i y}{dt^i} = \sum_{i=0}^{m} b_i \frac{d^i u}{dt^i}$$

dove a_i e b_i sono costanti, u e y sono le grandezze di ingresso e di uscita, e n è detto l'ordine del modello, cioè l'ordine massimo delle derivate dell'uscita presenti nell'equazione. E' richiesta la condizione

$$n \geq m$$

per garantire la realizzabilità fisica del sistema.

Un'altra forma di importanza fondamentale per la descrizione dei sistemi dinamici è quella basata sulle equazioni di stato. Vengono introdotte delle quantità denominate *variabili di stato*, che permettono di descrivere il sistema mediante le seguenti equazioni:

$$\frac{dx_i}{dt^i} = \sum_{j=1}^{n'} a_{ij}' x_j + b_i' u \qquad i = 1,2,\ldots,n'$$

$$y = \sum_{j=1}^{n'} c_j' x_j + du$$

dove x_i' sono le variabili di stato e a_{ij}', b_i', c_j' e d sono i parametri del modello.

Facendo uso delle notazioni vettoriali e matriciali è possibile scrivere le precedenti equazioni in una forma ancora più semplice e compatta:

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases}$$

essendo x un vettore $n \times 1$, e le matrici A ($n \times n$), B ($n \times 1$) e C ($1 \times n$) opportunamente formate dai coefficienti a'_{ij}, b'_i e c'_j, mentre per $\dot{x}$ si intende $\frac{dx}{dt}$.

Nel caso di sistemi fisici, i coefficienti dei vettori e delle matrici precedenti possono essere spesso ricavate scrivendo opportunamente le equazioni della fisica che descrivono il comportamento del sistema.

La precedente scrittura, però, riguarda sistemi lineari dinamici a tempo continuo. Esistono sistemi che è indispensabile descrivere discretizzando la variabile temporale, ed è possibile passare dal continuo al discreto con delle trasformazioni opportune. La scrittura di un sistema a tempo discreto non è molto diversa dalla precedente:

$$\begin{cases} x(k+1) = Ax(k) + Bu(k) \\ y(k) = Cx(k) + Du(k) \end{cases}$$

dove la variabile k = 0, 1, 2,... rappresenta l'istante in cui viene considerato il sistema.

Una volta ricavate le matrici è possibile affidarsi ad una base teorica molto solida che consente di sviluppare velocemente l'applicazione di controllo specifica.

2.5.4 Il modello del target

Al fine di sviluppare un sistema in grado di predire la posizione dell'obiettivo da colpire al momento desiderato è necessario modellizzare il sistema che costituisce l'obiettivo stesso.

La scelta del modello è stata effettuata in base al criterio di rendere l'algoritmo di tracking il più generale possibile. Conseguenza di ciò è il fatto che non è possibile costruire un modello che descriva in dettaglio un solo tipo di moto: se si fosse a conoscenza di qualche comportamento particolare del moto del target, cioè si sapesse, per esempio, che si muove di moto circolare uniforme oppure di moto rettilineo uniformemente accelerato, si potrebbero costruire dei predittori certamente più precisi e affidabili, ma meno generali e quindi meno efficaci nel caso di traiettorie diverse da quella ipotizzata.

Si sono dunque applicate le equazioni della dinamica del punto materiale, facendo opportune ipotesi sui valori di accelerazione nelle due direzioni del piano di moto e sui valori delle derivate di ordine superiore della accelerazione stessa.

Le equazioni del moto utilizzate sono dunque:

$$\frac{dz}{dt} = v$$

$$\frac{dv}{dt} = a$$

... derivate di ordine superiore.

dove z rappresenta la coordinata misurata e da stimare, v la velocità e a l'accelerazione del baricentro del target. Si è scelto di chiamare z la coordinata spaziale del punto per riservare la lettera x alle variabili di stato, secondo la consuetudine di rappresentazione introdotta nel campo della teoria dei controlli.

Effettuando la seguente ridenominazione delle variabili

$$x_1 = z$$

$$x_2 = v$$

$$x_3 = a$$

$$x_4 = \dot{a}$$

con la possibilità di procedere con l'ordine delle derivate, è possibile scrivere il modello del target come segue:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = x_3 \\ \dots \\ \dot{x}_{n-1} = x_n \\ z_x = x_1 \end{cases}$$

con *n* che rappresenta l'ordine del modello. Tale valore può essere pari a qualsiasi numero intero positivo, anche se, per considerazioni legate alle ipotesi su un qualche tipo di regolarità dell'andamento del moto, dei valori ragionevoli per *n* dovrebbero mantenersi nel range [2 ... 6]: come vedremo, più alto è il grado del modello e più il predittore sarà in grado di predire andamenti "bizzarri", adattandosi velocemente ai cambiamenti di direzione dell'oggetto, ma sarà, per contro, più sensibile agli errori di misura.

Passando ad una rappresentazione discreta del modello, è possibile scrivere:

$$\begin{cases} x_1(k+1) = x_1(k) + x_2(k) \\ x_2(k+1) = x_2(k) + x_3(k) \\ \dots \\ x_{n-1}(k+1) = x_{n-1}(k) + x_n(k) + w(k) \\ z_x(k) = x_1(k) + v(k) \end{cases}$$

dove sono stati introdotti i termini *w(k)* e *v(k)*, che rappresentano dei rumori bianchi i quali permettono di introdurre l'errore di modellizzazione e l'errore di misura all'interno del modello stesso.

In termini di matrici, riprendendo la scrittura introdotta al paragrafo 2.5.4, si ha una descrizione del sistema del tipo:

- la $A \in M^{n \times n}$ matrice triangolare superiore e avente la diagonale principale e quella superiore composta da tutti 1, mentre il resto della matrice è composta da 0;
- il vettore colonna $B \in M^{n \times 1}$ composto da tutti 0 escluso l'ultimo valore, che è pari a 1;
- il vettore riga $C \in M^{1 \times n}$, composto anch'esso da tutti 0 escluso il primo valore che è pari a 1;
- le matrici *Q* ed *R*, appartenenti entrambi ad $M^{1 \times 1}$ (sono quindi degli scalari), pari alle varianze di *w* e *v* rispettivamente.

Un esempio di tali matrici, nel caso di n=4 è il seguente:

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$$

Queste matrici vengono utilizzate per calcolare la soluzione dell'equazione di Riccati e quindi calcolare il vettore colonna $(n \times 1)$ costituito dai guadagni da utilizzare nel filtro di Kalman (vedi [12]).

2.5.5 Il filtro di Kalman

Il predittore di Kalman è una tecnica sviluppata per ricostruire lo stato di un sistema a partire dall'osservazione dei suoi ingressi e delle uscite corrispondenti. In tal senso fa parte di una più ampia gamma di sistemi che sono denominati, per l'appunto, *ricostruttori dello stato*. Nel seguito, anche se un po' impropriamente, ai termini filtro e predittore verrà attribuito il medesimo significato.

Un ricostruttore (o *osservatore*) dello stato è un sistema dinamico che, essendo alimentato dai segnali di ingresso e di uscita, presenta in uscita la stima dello stato del sistema dinamico sotto controllo. Un osservatore dello stato è indispensabile nei casi in cui non è possibile o economicamente conveniente misurare direttamente le singole componenti del vettore di stato del sistema

Partendo dal modello introdotto al paragrafo precedente, definisco

$$Q = \mathrm{var}[w]$$
$$R = \mathrm{var}[v]$$

Q ed R sono dunque la varianza dei rispettivi errori. La teoria relativa al filtro di Kalman prevede di ricavare delle equazioni in funzione di questi parametri. In realtà, si scopre che, se Q ed R sono numeri reali, la trattazione dipende unicamente dal loro rapporto, che definisco come

$$\rho = \frac{R}{Q}$$

Tale parametro condiziona il comportamento del filtro, ed in particolare la sensibilità al rumore. La variazione di ρ agisce direttamente sul valore dei guadagni del filtro, che vengono ricavati dalla formula

$$K = A \cdot P \cdot C^T \cdot (C \cdot P \cdot C^T + R)^{-1}$$

K è dunque un vettore avente un numero di elementi pari alla dimensione della matrice *A* che modellizza il sistema, mentre *P* è un vettore che risolve la seguente equazione di Riccati:

$$P = A \cdot P \cdot A^T + B \cdot Q \cdot B^T - A \cdot P \cdot C^T \cdot (C \cdot P \cdot C^T + R)^{-1} \cdot C \cdot P \cdot A^T$$

la quale dipende dai parametri *Q* ed *R*. E' immediato verificare che, se invece di considerare *Q* ed *R* avessi

$$\tilde{Q} = \alpha Q$$
$$\tilde{R} = \alpha R$$

otterrei

$$\tilde{\rho} = \rho$$
$$\tilde{P} = \alpha P$$
$$\tilde{K} = K$$

il che dimostra che, nelle equazioni, è possibile semplicemente considerare il rapporto tra le varianze dei rumori, piuttosto che il loro valore assoluto. Questo ha l'importante conseguenza di far dipendere il comportamento del filtro da un solo parametro. In effetti è possibile porre

$$Q = 1$$
$$R = \rho$$

Così facendo, si ottiene che aumentando ρ il predittore tende a filtrare maggiormente il rumore, ma diventa più lento nel seguire i cambiamenti dello stato del sistema. Al contrario, diminuendo ρ, il filtro garantisce una risposta più

veloce, adattandosi più rapidamente alle variazioni dello stato, ma si ottiene una previsione più sensibile al rumore.

La soluzione dell'equazione di Riccati può essere ricavata iterativamente utilizzando la seguente formula:

$$P_{k+1} = A \cdot P_k \cdot A^T + B \cdot Q \cdot B^T - A \cdot P_k \cdot C^T \cdot (C \cdot P_k \cdot C^T + R)^{-1} \cdot C \cdot P_k \cdot A^T$$

$$P_0 = I$$

tenendo conto che tale formula converge a $\overline{P}$, ovvero che

$$\overline{P} = \lim_{k \to \infty} P_k$$

2.6 Elaborazione dell'immagine

Date le scelte progettuali descritte al paragrafo 2.2, l'elaborazione dell'immagine prevede delle semplici operazioni di soglia sui valori di luminosità dei pixel, per individuare sia i punti che appartengono al puntatore laser che al target, isolandoli dal background dell'ambiente nel quale si muovono. La funzione che realizza la soglia nei due casi è riportata in figura 2.3.

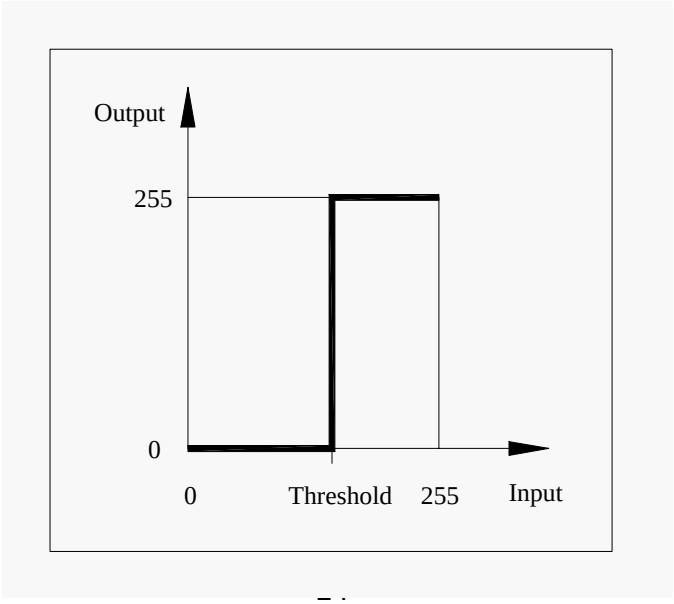


Figura 2.3. Soglia sul puntatore laser e sul target.

La differenza tra i due casi è che i pixel del target devono avere un livello di luminosità inferiore alla soglia, mentre i pixel del puntatore laser devono avere un livello di luminosità maggiore. Per garantire il funzionamento dell'apparato visivo è indispensabile che i due valori di soglia siano diversi tra loro: quello per il puntatore sarà molto alto (220-230 o più), mentre quello per il target deve essere molto basso (40-50 o meno).

Lo scopo dell'elaborazione dell'immagine è quello di trovare tutti i punti appartenenti al target o al puntatore e di calcolarne il baricentro, sempre espresso in pixel, quindi in coordinate immagine. Se (x_i, y_i) sono le coordinate degli n punti trovati, le due coordinate (x_c, y_c) del baricentro vengono calcolate con le formule:

$$x_c = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$y_c = \frac{\sum_{i=1}^{n} y_i}{n}$$

2.6.1 Informazioni sulle dimensioni degli oggetti

Le procedure che ricercano i punti appartenenti al target o al puntatore laser all'interno dell'area di scansione sono state sviluppate con dei piccoli accorgimenti per aumentarne l'efficienza in termini di velocità d'esecuzione. Esse utilizzano le informazioni sulle dimensioni degli oggetti, espresse in pixel e calcolate durante la primissima fase di inizializzazione, per scandire l'area di ricerca non punto per punto, bensì incrementando il valore della coordinata del

pixel di una quantità pari proprio alla dimensione dell’oggetto lungo l’asse corrispondente. Quando viene individuato un punto appartenente all’oggetto cercato viene effettuata una scansione pixel per pixel nel suo intorno, in una finestra di dimensioni adatte a contenere tutto l’oggetto.

Questa tecnica permette di risparmiare tanto più tempo quanto più gli oggetti cercati sono grandi.

Le funzioni gestiscono la possibilità che i valori di dimensione siano errati, sia a causa di una cattiva taratura che per motivi legati al moto dell’oggetto (rotazioni o allontanamenti ne possono ridurre l’effettiva dimensione sull’immagine). Nel caso in cui la prima scansione vada a vuoto, non venga trovato alcun punto appartenente all’oggetto, viene effettuata una seconda scansione su tutti i pixel, che sicuramente individua l’oggetto, se esso è presente nell’area di ricerca. Se anche questa scansione non ha esito positivo, significa che l’oggetto è uscito dal campo di visione del sistema, oppure che il valore di soglia non è corretto.

2.7 Tempo di volo e ritardo di attivazione della valvola

Per poter calcolare l'istante preciso in cui aprire la valvola per sparare il proiettile è necessario stimare due importanti parametri: il ritardo di attivazione dell'elettrovalvola e il tempo di volo del proiettile.

Il ritardo di attivazione è dovuto all'inerzia dei componenti in movimento dell'elettrovalvola, i quali fanno in modo che, una volta impartito il comando di apertura della valvola, debba trascorrere un certo lasso di tempo prima che il proiettile venga effettivamente lanciato. Dalle prove sperimentali descritte in questo stesso paragrafo, si è notato che il ritardo non è costante, come ci si potrebbe aspettare, ma varia leggermente di volta in volta. Questo fatto è altamente indesiderabile, in quanto si ripercuote con un inevitabile errore sulla precisione di tiro del sistema.

Il tempo di volo del proiettile è determinato da due parametri: dalla distanza tra punto di partenza e quello di impatto e dalla pressione dell'aria che viene usata per spararlo. Il primo aspetto, in prima approssimazione, può essere considerato costante, quindi si può ritenere che il tempo di volo dipenda unicamente dalla velocità che l'aria è in grado di imprimere alla pallina. In figura 2.4 è schematizzato il comportamento del moto proiettile dovuto al ritardo introdotto dall’elettrovalvola e dal tratto di volo.

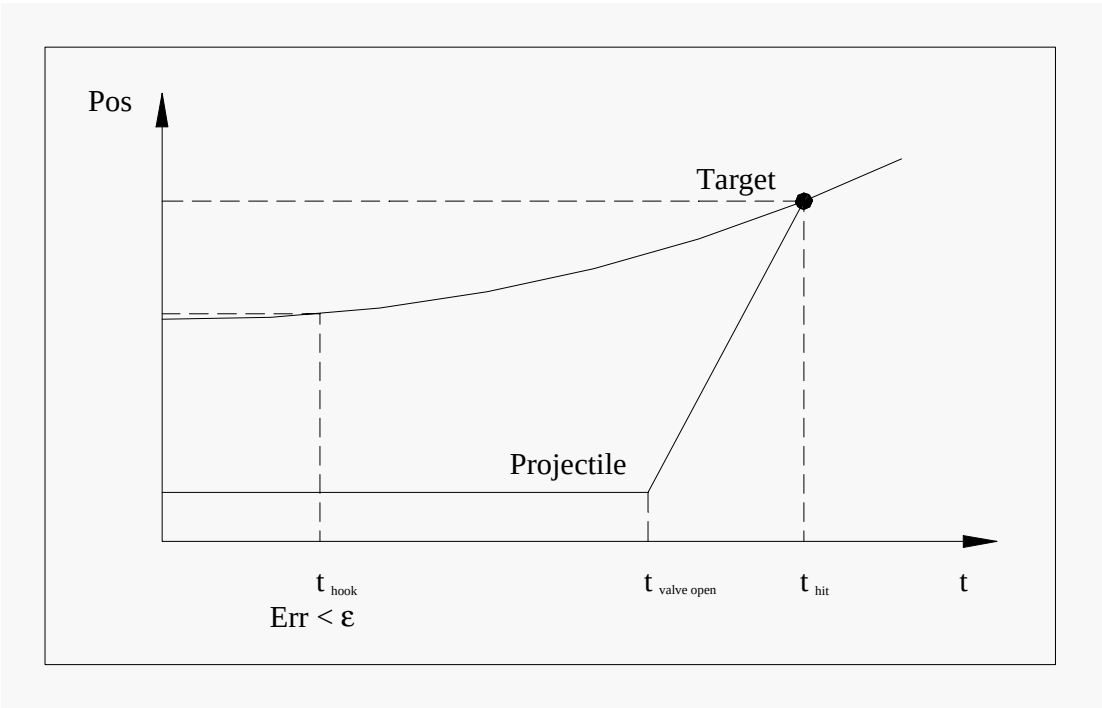


Figura 2.4. Ritardi dovuti all'elettrovalvola e al volo del proiettile.

2.7.1 La regressione lineare

Una regressione lineare viene calcolata sui risultati dei test a diverse distanze, con lo scopo di valutare la velocità della pallina e il ritardo di attivazione della valvola. Il risultato della regressione è una retta, identificata dal coefficiente angolare e dall'intercetta sull'asse delle ordinate, come schematizzato in figura 2.5. Questi due parametri, nel caso in esame, corrispondono rispettivamente all'inverso della velocità del proiettile e al tempo di ritardo dell'elettrovalvola. Effettuare una regressione lineare significa considerare sostanzialmente nullo l'attrito dell'aria sulla pallina, e quindi il moto come rettilineo uniforme.

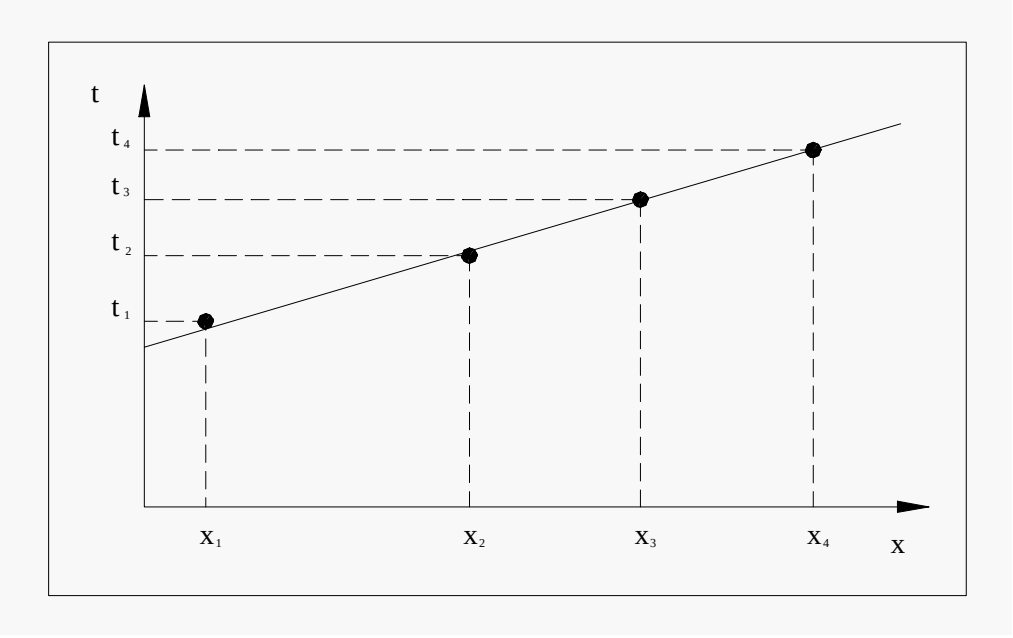


Figura 2.5. Risultato di una regressione lineare.

Una regressione lineare viene calcolata applicando le seguenti formule:

$$\begin{bmatrix} t_1 \\ t_2 \\ \dots \\ t_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \dots & \dots \\ 1 & x_n \end{bmatrix} \cdot \begin{bmatrix} q \\ m \end{bmatrix}$$

dove *n* è pari al numero di punti su cui viene effettuata la regressione. Pongo quindi

$$\begin{bmatrix} t_1 \\ t_2 \\ \dots \\ t_n \end{bmatrix} = t \quad \text{e} \quad \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \dots & \dots \\ 1 & x_n \end{bmatrix} = \Phi$$

Ora posso calcolare i valori che identificano la retta di regressione come

$$\begin{bmatrix} \hat{q} \\ \hat{m} \end{bmatrix} = (\Phi^T \cdot \Phi)^{-1} \cdot \Phi^T \cdot t$$

Per valutare la bontà dell'approssimazione dovuta alla regressione lineare è utile ricorrere all'*intervallo di confidenza*. Quest'ultimo è definito come l'intervallo nel quale cade una certa percentuale dei test. Nel nostro caso verrà calcolato un intervallo di confidenza al 95%. Per poter calcolare l'intervallo è necessario ricavare i seguenti opportuni valori:

$$\hat{\sigma}^2 = \frac{\left\| I - \Phi \cdot (\Phi^T \cdot \Phi)^{-1} \cdot \Phi^T \cdot t \right\|^2}{n-2}$$

mentre la matrice delle varianze è data da

$$\mathrm{var}\begin{bmatrix} \hat{q} \\ \hat{m} \end{bmatrix} = (\Phi^T \cdot \Phi)^{-1} \cdot \hat{\sigma}^2 = \begin{bmatrix} \mathrm{var}(\hat{q}) & \mathrm{cov}\,ar(\hat{q}, \hat{m}) \\ \mathrm{cov}\,ar(\hat{q}, \hat{m}) & \mathrm{var}(\hat{m}) \end{bmatrix}$$

Ricavo ora altri coefficienti

$$\sigma_{\hat{q}} = \sqrt{\mathrm{var}(\hat{q})}$$

e l'intervallo di confidenza è dato da

$$\left[\hat{q} - \sigma_{\hat{q}}, \hat{q} + \sigma_{\hat{q}}\right]$$

2.8 La rete neurale

DEA utilizza tecniche di intelligenza artificiale per convertire le coordinate di un punto sull'immagine nei valori di controllo da fornire ai servomotori. Nel caso specifico, per svolgere questo compito, viene addestrata una rete neurale opportunamente dimensionata.

Una panoramica sulle reti neurali in genere ed in particolare sulle reti error backpropagation viene rimandata al capitolo 4. Questa scelta è motivata dal fatto che sulle reti backpropagation sono stati condotti una serie di test per determinarne il comportamento al variare di opportuni parametri caratteristici.

Capitolo 3

L’hardware di DEA

3.1 L’apparato hardware

Per poter seguire e colpire un oggetto in movimento è ovviamente necessario poter mirare in una determinata direzione. Per fare questo è stato costruito un dispositivo meccanico motorizzato in grado di indirizzare la “cerbottana” ed il puntatore laser nella direzione desiderata.

Per controllare i servomotori utilizzati e per fornire i segnali di attivazione al laser ed all’elettrovalvola, che controlla l’afflusso di aria compressa alla bocca di sparo, si sono utilizzate le uscite della scheda National Instruments PCI-6025E opportunamente programmate via software.

I dati in input vengono invece catturati da una telecamera abbinata ad un frame-grabber Imagenation PXC200L che digitalizza l’immagine. Lo schema di interfaccia tra i diversi dispositivi hardware è riportato in figura 3.1. Nei paragrafi a seguire, invece, vengono descritti nel dettaglio i diversi aspetti costruttivi e di utilizzo dei componenti hardware.

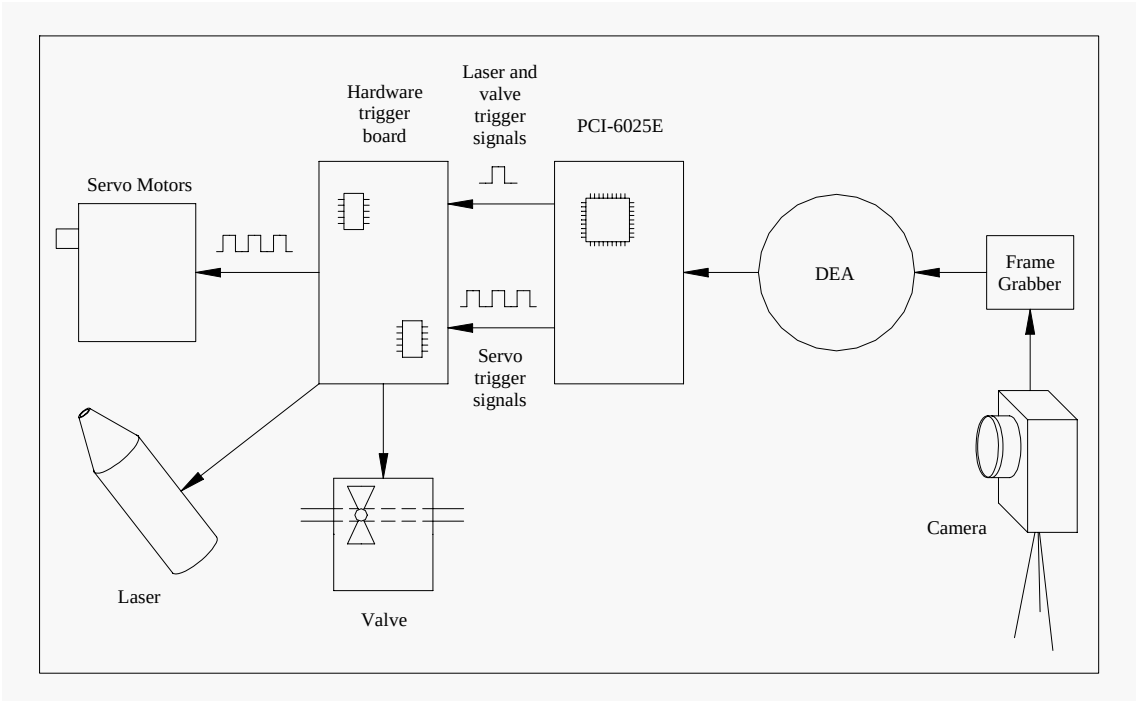


Figura 3.1. Dispositivi hardware usati da DEA.

3.2 Il sistema di visione

Per sistema di visione si intende l'apparato hardware che consente a DEA di ottenere una sequenza di immagini le quali, opportunamente elaborate, permettono di effettuare la calibrazione automatica iniziale dell'intero sistema, necessaria quando avviene una modifica nella disposizione o nella taratura di uno qualsiasi dei componenti del sistema stesso. Le immagini vengono elaborate per eseguire il tracking dell'oggetto da colpire, una volta calibrato il sistema.

Il sistema di visione, dunque, è costituito da una telecamera e da una scheda di acquisizione, il *frame-grabber*, che, montato sull'elaboratore sul quale viene eseguito il software di DEA, digitalizza le immagini che provengono dalla telecamera.

3.2.1 La telecamera

La telecamera costituisce l'unico sensore presente nell'intero sistema. Qualsiasi operazione deve quindi basarsi sui dati visivi ottenuti proprio dalla telecamera.

Figura 3.2. La telecamera.

La telecamera utilizzata in DEA è una IVC500, con obiettivo da 50 millimetri. Per dare un'idea dell'ampiezza della visuale ottenibile con questo obiettivo si tenga presente che, alla distanza di circa 530 centimetri, viene ripresa un'immagine le cui dimensioni reali sono di circa 70x60 centimetri.

La telecamera è corredata di un apposito dispositivo che funge sia da alimentatore che da interfaccia tra il frame-grabber e la telecamera stessa. A questo dispositivo viene infatti connesso l'unico cavo che arriva alla telecamera, e sempre da questo stesso dispositivo esce il cavo di collegamento col grabber.

3.2.2 Il frame-grabber

Il frame-grabber non è altro che una scheda di acquisizione in grado di digitalizzare un segnale video e fornirlo in uscita ad una determinata frequenza. La scheda utilizzata nel progetto è una Imagenation PXC200L, installabile su un qualsiasi elaboratore dotato di uno slot PCI libero. Le caratteristiche tecniche della scheda PXC200 sono le seguenti:

- scheda PCI corta e compatta;
- master design del bus PCI che garantisce la cattura real-time delle immagini;
- supporto per i formati di output YCrCb, RGB e Y8 (livelli di grigio);
- elevata accuratezza nella cattura del colore, con un basso jitter sui pixel;
- supporto per gli standard di cattura delle immagini di 640 x 480 pixel (NTSC) e di 768 x 576 pixel (PAL);
- 4 ingressi video multiplexati (NTSC/PAL/S-video);
- operazioni di time image scaling con interpolazione, cropping orizzontale e verticale in real-time;
- possibilità di cattura dei frame continua, pilotata da software e controllata da segnale esterno;
- segnale di pilotaggio esterno in standard TLL;
- alimentatore +12 VDC per la videocamera;
- monta un micro-controllore single-chip compatibile con il Conexant Bt848;
    - periodo di campionamento pari a 20 millisecondi, che corrisponde a un frame-rate massimo di 50 immagini al secondo.

La caratteristica più importante di questa scheda è la possibilità di abilitare il mastering del bus PCI e, attraverso il DMA, di scrivere direttamente in memoria l'immagine alla massima frequenza possibile senza bisogno di utilizzare la CPU, che quindi è libera di eseguire altre operazioni.

La scheda della Imagenation si colloca nella fascia di prezzi medio-bassa per questo tipo di dispositivi, garantendo comunque delle prestazioni di tutto rispetto.

3.3 I servomotori

I servomotori, detti brevemente “servo”, sono dei dispositivi utilizzati molto comunemente nel campo del modellismo e della robotica. Essi sono costituiti da un motore in corrente continua, dall’elettronica necessaria a pilotare il motore stesso e da una serie di ingranaggi di riduzione della velocità, il tutto montato all’interno di uno chassis rigido. La parte elettronica è costituita, in particolare, da un trasduttore di posizione angolare, in genere un potenziometro, montato sull’albero d’uscita, in grado di restituire appunto la posizione angolare dell’albero. Sono infine presenti due finecorsa che limitano l’angolo massimo di movimento dell’albero.

I servo si caratterizzano in base a diversi aspetti: la coppia disponibile all’albero, la velocità di rotazione, l’angolo massimo di rotazione, il materiale con cui sono costruiti i riduttori, la presenza o meno di cuscinetti a sfera, le dimensioni e il peso.

I due servo utilizzati per il movimento di DEA, dei quali uno è raffigurato in figura 3.3, sono degli HITEC, modello HS-805BB, le cui caratteristiche principali sono le seguenti:

- coppia di 19.8 kg cm;
- velocità pari a 0.19 sec per compiere 60 gradi, con tensione di alimentazione pari a 4.8V;
- angolo di rotazione massimo corrispondente a circa 180 gradi (mezzo giro);
- assorbimento di corrente pari 8 mA in stato di riposo, e 800 mA in condizioni di carico massimo;
- ingranaggi in plastica dura;
- 2 cuscinetti a sfera sull’albero di uscita;
- dimensione 66x30x57.6 mm,
- peso pari a 152 grammi.

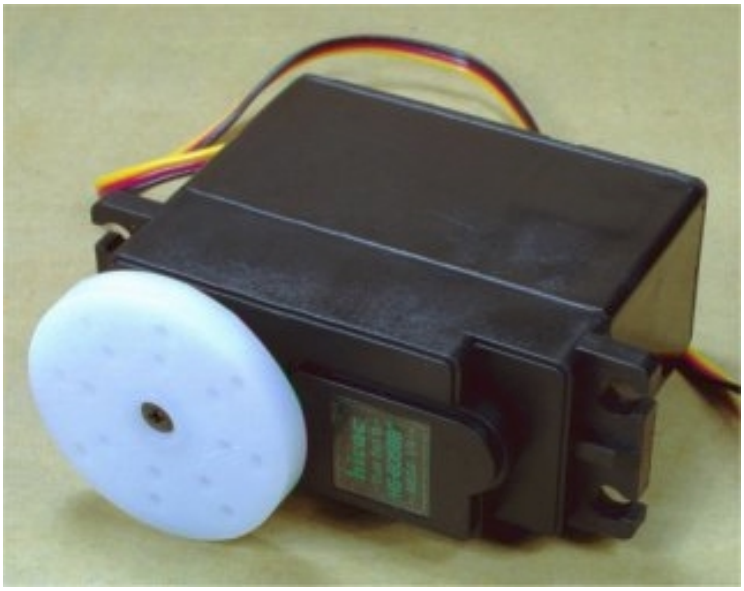


Figura 3.1. Servocomandi utilizzati per muovere DEA

Nel corso del progetto ho avuto modo di utilizzare dei servo in grado non solo di compiere rotazioni maggiori dei canonici 180 gradi assicurati da quasi tutti i servo, ma addirittura di ruotare indefinitamente nell'una o nell'altra direzione.

Questi servocomandi vengono ottenuti modificando opportunamente i normali servo. In genere è necessario rimuovere i finecorsa interni allo chassis, che è il solo responsabile delle limitazioni sull'ampiezza di rotazione. Inoltre, il reostato va sostituito con una resistenza di valore fisso pari a quella del reostato in posizione centrale.

Effettuando tali modifiche si ha che, fornendo un valore di pilotaggio che porterebbe il servo in una posizione a sinistra della posizione centrale, il motore ruota verso sinistra a velocità costante; il caso della rotazione a destra è duale. Fondamentalmente si tratta di "far credere" all'elettronica di bordo del servo che questo si trovi sempre nella posizione centrale, cosicchè quando lo si comanda di portarsi in una posizione diversa, il motore continua a girare indefinitamente nel tentativo di ottenere dal reostato (sostituito dalla resistenza) il valore

corrispondente. Per arrestare il servo è necessario fornire il valore di pilotaggio che corrisponde alla posizione centrale, oppure uno abbastanza vicino allo stesso, in quanto, a causa degli attriti, il motore tende naturalmente a fermarsi, quando non è sufficientemente "stimolato".

Appare chiaro che, così facendo, non si controlla il motore in velocità, ma si ottiene solo una rotazione dello stesso in una direzione o in quella opposta *alla massima velocità raggiungibile dal servo*.

In realtà è possibile modulare la velocità del servo calibrando attentamente l'ampiezza dell'onda quadra in ingresso, col problema che il dispositivo fornisce in uscita una coppia molto minore di quella nominale.

I servo che muovono il dispositivo di puntamento di DEA, comunque, non hanno subito alcuna modifica, quindi vengono pilotati normalmente in posizione.

3.3.1 Protezione di un servocomando

Per protezione di un servo si intende un meccanismo atto ad evitare che il motore raggiunga il finecorsa in entrambe le direzioni e che, nel tentativo di portarsi in una posizione non consentita, oltre i limiti imposti dai finecorsa stessi, causi il precoce deterioramento del motore in continua, oppure quello dell'elettronica di bordo.

Proteggere un servo significa quindi, sostanzialmente, limitare il range di variazione del segnale modulato in PWM entro un intervallo adeguato. Questo tipo di protezione è indispensabile intrinsecamente alle caratteristiche costruttive e di funzionamento del servo, anche se fatto funzionare a vuoto. In effetti una tale protezione non è prevista dall'elettronica di bordo, ed è quindi necessario realizzarla a monte dell'elettronica stessa.

Una seconda ragione per limitare ulteriormente il range di variazione del PWM è data, in genere, dal sistema entro il quale il servo viene montato, come accade anche per tutti gli altri tipi di azionamento: il servo, non potrà quindi

spazzare l'intero campo di movimento. Questo è necessario, in genere, per non danneggiare sia il sistema portato in movimento dal servo che il servo stesso. Una schematizzazione dei vincoli che nascono dalle caratteristiche del servo e del sistema su cui è montato è riportata in figura 3.4.

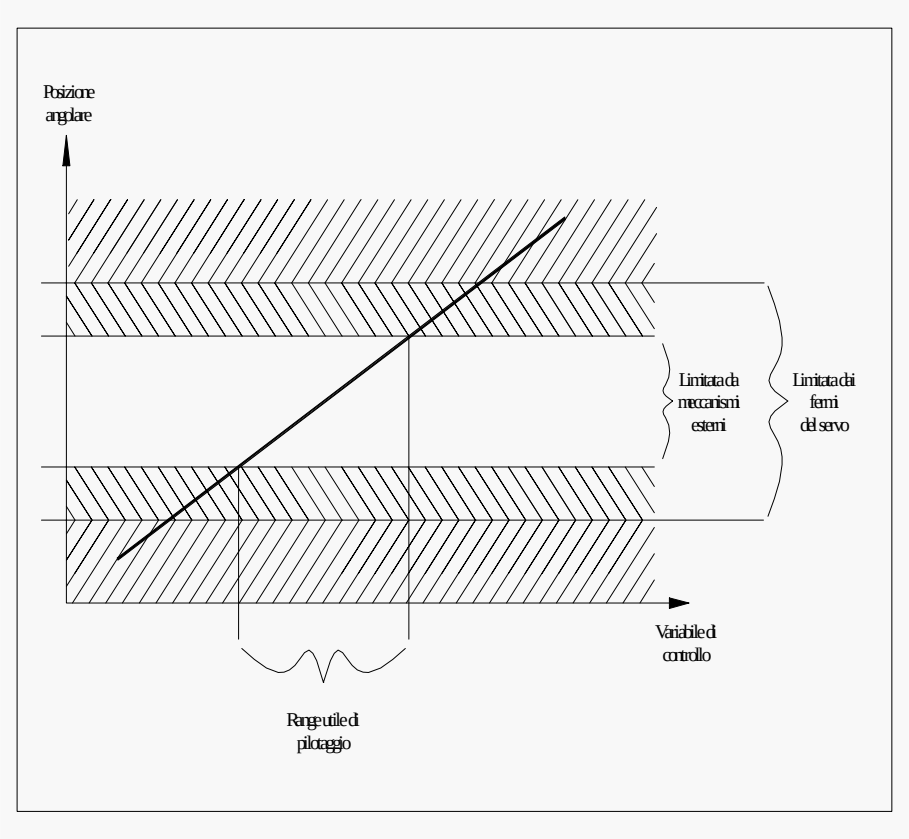


Figura 3.4. Limiti di pilotaggio dei servocomandi.

Il corretto range di funzionamento, tenuto conto dei vincoli precedentemente esposti, va stimato con opportuni test sul servocomando, in quanto cambiano caso per caso. I limiti, infine, devono essere fatti rispettare dal sistema che genera i valori di pilotaggio del motore, che può essere sia un dispositivo elettronico, un processo software che controlla il cambiamenti di stato dell'onda quadra generata (anche se è spesso difficile gestire tali tipi di processo, in quanto i vincoli temporali da rispettare sono strettissimi, per un buon

funzionamento del pilotaggio), oppure, ancora, una soluzione ibrida, con una routine che comanda un dispositivo elettronico, imposta il duty-cycle dell'onda da generare e poi demanda la generazione del treno d'onda all'elettronica.

Quest'ultima soluzione è quella adottata in DEA per una sola, semplice ragione: è l'unica applicabile in questo caso.

La soluzione basata sull'elettronica pura va scartata in quanto il pilotaggio di DEA avviene da elaboratore.

La soluzione che prevede la generazione dell'onda quadra da parte di un task software è inaccettabile per problemi di precisione ed efficienza. Infatti, per realizzare una temporizzazione dell'ordine dei microsecondi, sarebbe necessario dedicare un processo periodico ad elevatissima frequenza, che provocherebbe un carico eccessivo sul processore.

3.3.2 Controllo di un servo

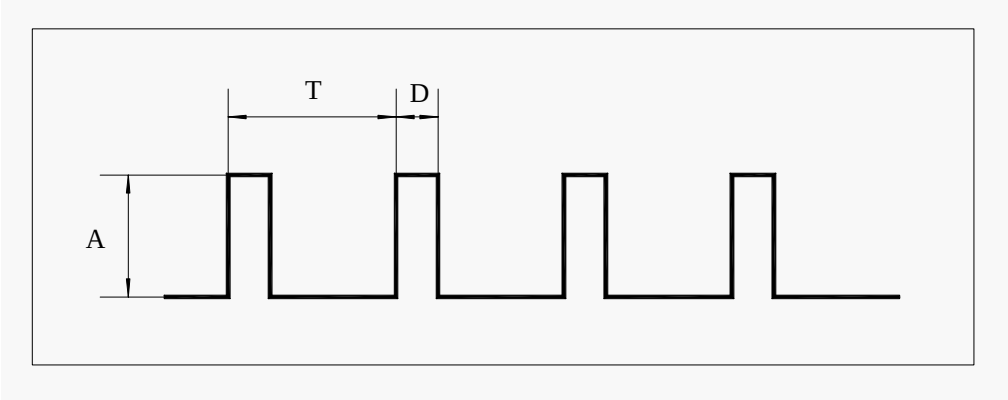
Un servomotore ha fisicamente in ingresso tre fili: l'alimentazione (rosso), il segnale di controllo (giallo) e la massa (nero). Il primo fornisce potenza al motore e all'elettronica di bordo, mentre il secondo serve a definire la posizione angolare alla quale si desidera posizionare l'albero del servo.

Il controllo vero e proprio avviene con un segnale modulato PWM (Pulse Width Modulation). Un segnale PWM è, nel caso dei servo, una onda quadra di frequenza opportuna e ampiezza pari a 5 volt (vedi figura 3.5), il cui impulso viene modulato in ampiezza per definire la posizione angolare dell'albero: si ha che tale posizione varia linearmente con la durata dell'impulso. E' importante notare che il comportamento del servo non dipende tanto dal duty-cycle del segnale, quanto piuttosto dalla durata dell'impulso. Infatti i servo sono progettati in modo da lavorare correttamente in un certo range di frequenze del segnale di controllo, all'interno del quale sono insensibili a variazioni di frequenza;

ovviamente, a parità di frequenza, parlare di durata dell'impulso e di duty-cycle è indifferente.

Figura 3.5. Segnale PWM per pilotare i servo motori

I servo utilizzati per muovere DEA vengono controllati con una onda quadra



di periodo T pari a 15 millisecondi, con un impulso di durata D, variabile da circa 700 microsecondi (albero a finecorsa a sinistra) a circa 2250 microsecondi (albero a finecorsa a destra).

Si è verificato sperimentalmente che i servomotori in dotazione funzionano correttamente fornendo un segnale di controllo con periodo variabile tra i 10 e i 70 millisecondi circa. Si è però notato che con un periodo troppo breve, intorno ai 10 ms, il servo mostra dei problemi di stabilità in uscita, mentre con un periodo maggiore di 30-40 ms la coppia in uscita diminuisce quanto più il periodo aumenta. Valori accettabili per il periodo sono stati riscontrati tra i 12 e i 18 millisecondi, da cui la scelta di 15 ms. I valori ottenuti sono relativi ai motori utilizzati nel progetto di DEA: motori di altre marche o modelli hanno, in genere, valori tipici diversi. E' anche vero che, in diversi tipi di servomotori testati, l'ordine di grandezza è lo stesso, tant'è vero che, fornendo in ingresso l'onda modulata con frequenza pari a 15-20-30 millisecondi, si è quasi certi, in ogni circostanza, di riuscire a pilotare correttamente il servo.

3.3.3 Imporre una curva di funzionamento al motore

Uno dei problemi che possono presentarsi nel controllo di un servomotore è quello di voler imporre una determinata curva di funzionamento al motore. La dinamica del suo moto, per raggiungere la posizione desiderata, si basa su quella del motore in continua presente al suo interno, quindi prima accelera fino a raggiungere la velocità massima, poi decelera fino a portarsi nel punto specificato. Sarebbe auspicabile poter controllare in particolare il valore della velocità massima, per garantire movimenti più fluidi ed evitare sollecitazioni eccessive ai dispositivi meccanici.

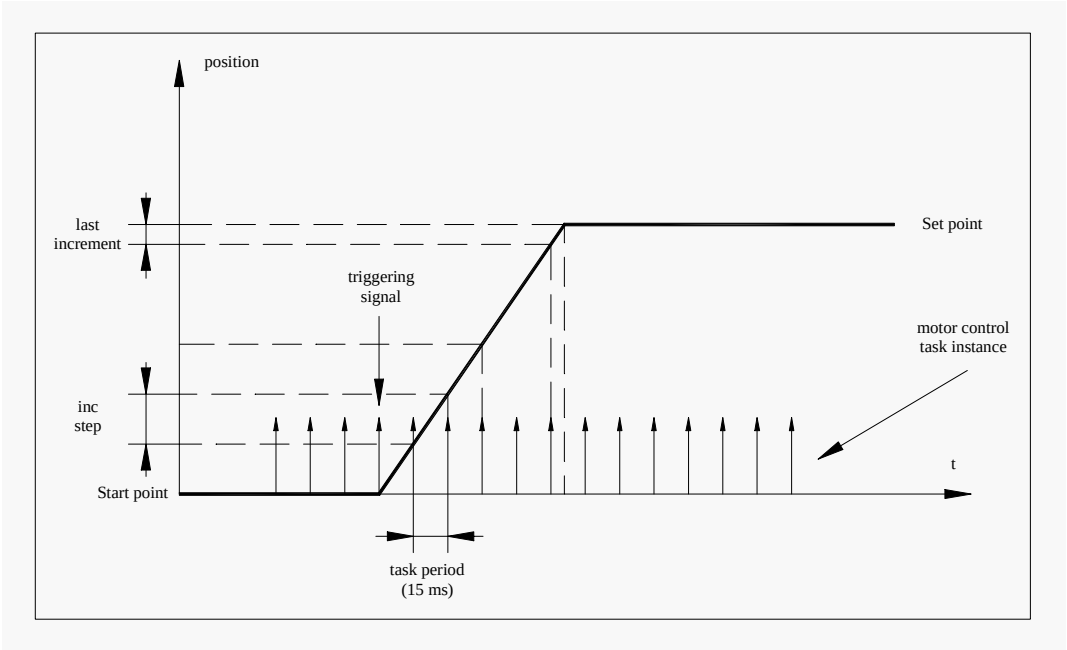


Figura 3.6. Curva di funzionamento imposta ai servomotori.

In figura 3.6 è riportato il grafico di una tipica curva con andamento a rampa, la cui pendenza è proporzionale alla velocità massima dell'azionamento.

Un vantaggioso effetto collaterale del poter imporre una curva di funzionamento predeterminata al motore risiede nel fatto che, in tal modo, è

possibile conoscere istante per istante la posizione del motore, quindi, in particolare, l'istante nel quale il motore si arresta nella posizione finale. Da notare che, in assenza di una tale imposizione della curva di funzionamento, tali informazioni si potrebbero comunque calcolare analizzando la dinamica del motore ed effettuando i calcoli opportuni sui risultati di tale analisi, ma questa si dovrebbe ripetere in caso, per esempio, di sostituzione dei motori con altri non identici. Col modello adottato, invece, si eliminano intrinsecamente tali possibili problemi.

Inoltre è possibile rendere parametrica la velocità massima di funzionamento, in modo da poterla adattare alle diverse condizioni di funzionamento del sistema.

3.4 Il puntatore laser

Il puntatore laser, raffigurato in figura 3.7, è un comune puntatore che si trova spesso abbinato ai portachiavi. Questi gadget possono essere acquistati presso qualsiasi cartoleria, anche se ultimamente i prodotti di questo tipo sono stati ritirati dal commercio perché possono causare danni alla retina se vengono puntati sugli occhi. Le caratteristiche del puntatore sono le seguenti:

- potenza massima del fascio laser pari a 5 milliwatt;
- alimentato con tensione di 3 volt;
- proietta un punto di colore rosso;
- le dimensioni del punto luminoso sono di circa 4 millimetri alla distanza di 5 metri.

Figura 3.7. Puntatore laser

Le dimensioni del punto e il grado di nitidezza con cui viene proiettato dipendono fortemente dalla qualità della lente che viene utilizzata per correggere la direzione dei fasci laser, come spiegato nel paragrafo 3.4.1 dedicato ai diodi laser.

Il puntatore è stato modificato per poterne controllare l'accensione e lo spegnimento da PC. Sono state rimosse le batterie di alimentazione e sono stati opportunamente collegati i morsetti del puntatore all'alimentazione pilotata dall'elaboratore. L'interruttore presente sul puntatore è stato manipolato in modo che fosse sempre chiuso. Così facendo si è realizzato il dispositivo il cui schema è riportato in figura 3.8, che viene acceso fornendo la giusta tensione dall'esterno.

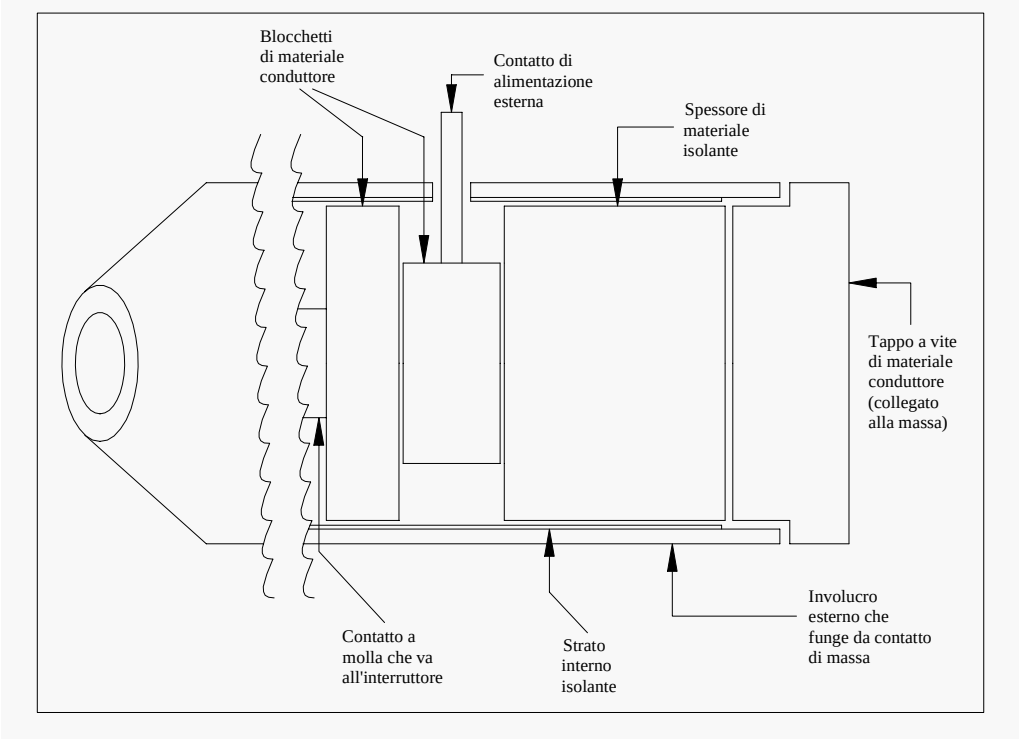


Figura 3.8. Schema del puntatore laser modificato.

3.4.1 Il diodo laser

I diodi laser sono i dispositivi che vengono comunemente utilizzati per produrre il fascio laser in tutti i puntatori usati come gadget per portachiavi e simili. Il diodo laser, al pari di un led, è formato da una giunzione PN che, percorsa da corrente elettrica, emette una radiazione luminosa. Un comune diodo led produce delle radiazioni spontanee, cioè non coerenti, mentre un diodo laser emette una luce coerente.

Il diodo produce un fascio luminoso di colore rosso, infatti la lunghezza d'onda della radiazione luminosa emessa si aggira intorno ai 670-680 nanometri. Ultimamente sono entrati in commercio dei diodi capaci di produrre una

radiazione di colore verde, avente lunghezza d'onda di circa 520-530 nanometri (ricordo che lo spettro della luce visibile all'occhio umano varia tra 400 e 700 nanometri circa).

Tutti i diodi laser producono un fascio luminoso ovalizzato e con due differenti divergenze, l'una variabile tra 6 e 12 gradi e l'altra tra 20 e 40 gradi. In pratica il fascio emesso dal diodo laser si allarga a ventaglio tanto che alla distanza di pochi centimetri questo coprirà un'area di diversi centimetri. Quindi un diodo laser, se non completato da un apposito obiettivo, non sarà mai in grado di fornire il classico puntino luminoso, la cui definizione e dimensione in rapporto alla distanza è tanto migliore quanto maggiore è la qualità dell'obiettivo. In figura 3.9 e 3.10 viene schematizzata la forma del fascio laser con e senza obiettivo.

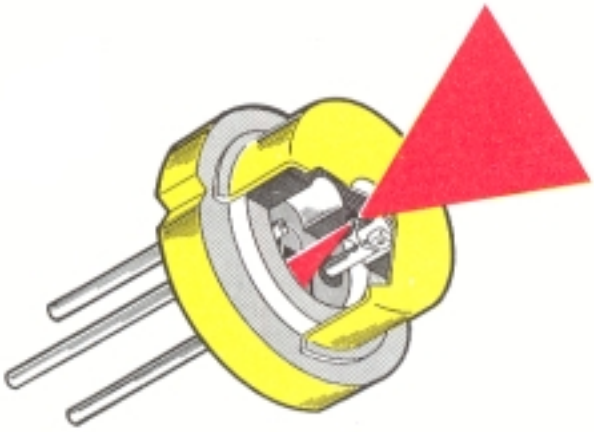


Figura 3.9. Fascio luminoso generato da un diodo laser.

Figura 3.10. Fascio luminoso di un diodo laser corretto dalle lenti.

La caratteristica più importante di un diodo laser è, ad ogni modo, la potenza del fascio emesso, i cui valori tipici sono di 1, 3, 5 e 15 milliwatt. La potenza di un diodo laser non va comunque confusa con la brillantezza del punto luminoso prodotto: quest'ultima dipende dalla lunghezza d'onda della radiazione prodotta, che è indipendente dalla potenza del diodo.

3.5 Dispositivo elettropneumatico

DEA utilizza come proiettili delle sfere di plastica dura, del diametro di circa 6 millimetri, utilizzati nelle più comuni pistole ad aria compressa. Similmente, il "propellente" che consente di scagliare il dardo contro l'obiettivo è proprio l'aria compressa. E' stato quindi progettato un circuito in grado di controllare l'afflusso di aria ad una canna che costituisce la "bocca di fuoco" del sistema. Questa è costituita da una coppia di tubi infilati l'uno dentro l'altro, come in una normale pistola ad aria: in effetti sono proprio gli stessi tubi utilizzati in tali pistole. Il tubo interno è in stagno piuttosto malleabile, mentre quello esterno è in ottone molto rigido.

L'aria proviene da un comune compressore, in grado di comprimere l'aria fino a raggiungere una pressione massima di 8 atmosfere. L'aria viene portata alla canna per mezzo di un tubo flessibile per aria compressa del diametro di 6 millimetri. Quest'ultimo viene connesso alla canna grazie ad un adattatore per circuiti pneumatici del tipo "a sgancio rapido".

L'elettrovalvola, riportata in figura 3.11, cioè il dispositivo che permette di aprire o chiudere il passaggio dell'aria dal compressore verso la canna, è costituita da una valvola pneumatica comandata da un relè, che può essere a sua volta controllato fornendogli o meno potenza. Il relè, per poter funzionare, richiede un'alimentazione con tensione a +12 volt in corrente continua e consuma una potenza pari a 5 watt. Esso è costituito da un solenoide che, se alimentato, fa scorrere un cilindro in materiale ferromagnetico il quale agisce sulla valvola

pneumatica aprendola. Togliendo alimentazione il cilindro ritorna nella posizione di riposo grazie a un meccanismo a molla; in questa posizione la valvola pneumatica è chiusa.

Figura 3.11. L'elettrovalvola.

Da notare il fatto che è possibile aprire manualmente la valvola pneumatica con un cacciavite, andando ad agire su un apposito comando a vite.

3.6 La scheda National Instruments PCI-6025E

La PCI-6025E è una scheda multi-funzione per PC di costo relativamente basso prodotta dalla National Instruments, che può essere installata su un qualsiasi computer che disponga di uno slot PCI libero.

Le principali funzionalità messe a disposizione dalla PCI-6025E sono le seguenti:

- 8 linee ADC in ingresso a 24 bit;
- 2 linee DAC in uscita a 24 bit;

- 8 linee di I/O digitali (DIO) programmabili singolarmente in ingresso oppure in uscita;
- 2 contatori multifunzione a 24 bit (GPCT, acronimo di General Purpose Counters).

Grazie all'utilizzo del bus PCI, la scheda è completamente programmabile via software, compresa, per esempio, l'impostazione dell'indirizzo base, che in schede per bus ISA, ad esempio, deve invece essere effettuata manualmente con l'ausilio di appositi dip-switch.

Il cuore della PCI-6025E è il microchip DAQ-STC, progettato dalla National Instruments per fornire i servizi di ADC, DAC, DIO e GPCT. In effetti questo stesso chip viene montato anche su altre schede della stessa casa (ve ne sono con interfaccia ISA e seriale), oltre che su componenti di terze parti, e per accedere alle funzionalità della scheda è necessario programmare proprio il DAQ-STC. La National Instruments mette a disposizione un potente ambiente di sviluppo, il LAB-VIEW, che, grazie a un linguaggio di programmazione proprietario, è appositamente studiato per sfruttare appieno le potenzialità delle schede National senza sforzo eccessivo. La National Instruments fornisce anche delle librerie di funzioni e driver per i più diffusi ambienti di sviluppo (ve ne sono anche per il Visual C++ della Microsoft e per Delphi di Borland), che semplificano lo sviluppo di applicazioni che utilizzano tali schede. Per potere invece utilizzare le stesse schede con linguaggi di programmazione diversi dai precedenti, per esempio il C, come nel caso del presente progetto, è necessario scrivere una apposita libreria di funzioni per accedere al bus PCI, ai registri della scheda e a quelli del DAQ-STC. Fortunatamente la documentazione relativa a questi tre aspetti, fornita nel CD-ROM allegato alla scheda oppure scaricabile gratuitamente dal sito della National Instruments, è decisamente completa, cosa che semplifica notevolmente la codifica delle funzioni necessarie.

Di seguito verrà tralasciata la trattazione di DAC e ADC per concentrarla su linee digitali e contatori, che sono le uniche funzionalità della scheda utilizzate nel progetto.

3.6.1 Linee digitali di I/O

La scheda National Instruments PCI-6025E dispone di 8 linee digitali (DIO0.. DIO7) per un utilizzo generico. Ciascuna di esse può venire impostate via software in input oppure in output secondo le necessità. All'avvio dell'elaboratore oppure al reset della scheda ogni linea si trova in stato di alta impedenza.

Il funzionamento di DEA richiede l'utilizzo di 4 linee in uscita. Le funzioni dei 4 bit in output sono:

- controllare l'apertura dell'elettrovalvola;
- pilotare l'interruttore del puntatore laser;
- abilitare o meno il conteggio del contatore che pilota il servo orizzontale;
- abilitare o meno il conteggio del contatore che pilota il servo verticale.

Le ultime due uscite elencate sono, di fatto, collegate a due ingressi della scheda stessa, impostati per controllare il conteggio dei contatori general-purpose messi a disposizione sempre dalla scheda. Si tenga presente che i valori di tensione in uscita dalle linee digitali sono in standard TTL, quindi 0 volt oppure +5 volt, mentre elettrovalvola e laser funzionano con tensioni di alimentazione di +12 volt e 3 volt rispettivamente; per questo motivo non è possibile utilizzare direttamente le uscite digitali per pilotare i dispositivi, pratica da evitare comunque e in qualsiasi caso, onde evitare possibili danni alla scheda; le uscite vengono connesse in ingresso al circuito elettronico descritto nel paragrafo 3.7, che fornisce a sua volta in uscita dei segnali di controllo di opportuna ampiezza e potenza.

3.6.2 Timer general-purpose

Di più complessa utilizzazione sono i contatori. Questi possono essere utilizzati in svariati modi:

- in input per contare gli eventi che si presentano su un pin appositamente designato, sotto forma di cambiamento di stato di un segnale in tensione;
- sempre in input, può essere misurato il periodo di un segnale periodico o la durata dell'impulso di un segnale periodico o meno;
- possono essere utilizzati in output per la generazione di impulsi singoli della durata desiderata;
- per la generazione di N impulsi con frequenza e duty-cycle desiderati;
- per la creazione di un treno continuo di impulsi.

Tutte queste operazioni hanno in genere una o più varianti: ad esempio, il conteggo di eventi può essere bufferizzato o meno, oppure il treno di impulsi può essere modulato automaticamente in ampiezza in base ad opportuni parametri.

Lo schema di ciascun contatore montato sulla scheda e dei registri di interfacciamento è riportato in figura 3.12.

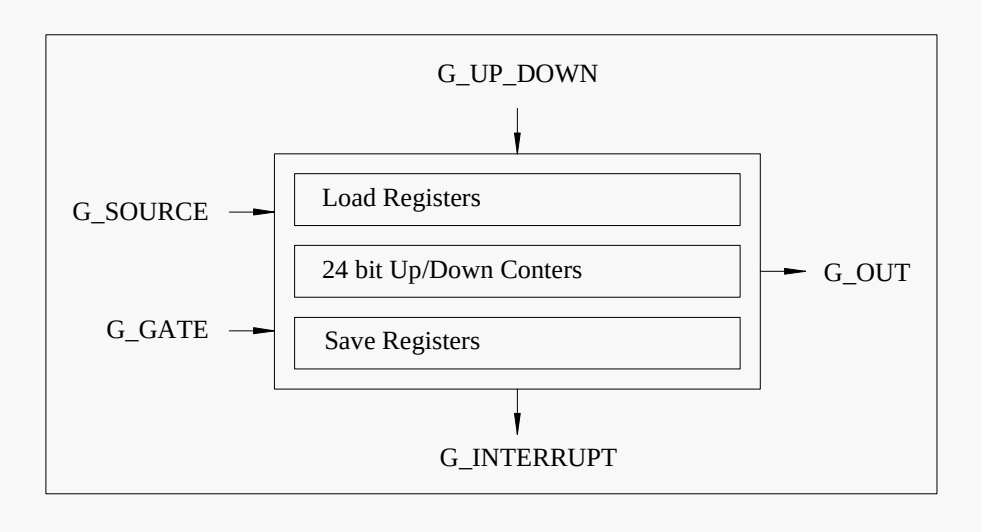


Figura 3.12. Schema di un timer general purpose.

In figura sono mostrati i diversi componenti del timer/counter, i quali hanno il seguente significato:

- i *Load Registers* sono una coppia di registri a 16 bit che permettono di caricare nel contatore/timer un qualsiasi valore a 24 bit;
- il *24 bit Up/Down Counter*, ovvero il contatore vero e proprio, ha al suo interno 2 banchi di registri indipendenti nei quali memorizzare il valore del conteggio, e che permettono, ad esempio, di cambiare il duty cycle di un'onda quadra senza interrompere in treno di impulsi;
- i *Save Registers* che sono disponibili per la lettura del valore di conteggio, in quanto sono in latch con registri di conteggio;
- la linea *G_SOURCE* in ingresso, costituito in genere dall'output di un'oscillatore, che costituisce il clock sul quale si basa il conteggio;
- l'ingresso *G_GATE*, che può essere un segnale del quale si desidera misurare la durata dell'impulso o del periodo, oppure, nel caso della generazione di impulsi, determina se abilitare il conteggio o meno;
- *G_OUT* costituisce il segnale di uscita quando il timer viene usato per generare degli impulsi;
- la linea di *INTERRUPT* viene invece abilitata al termine del conteggio, in genere utilizzata per segnalare al software o a dell'hardware esterno, l'avvenuto completamento di un'operazione.

3.6.3 Misurazione dell'ampiezza di un impulso

Questa feature, utilizzabile attraverso l'opportuna programmazione dei timer della scheda PCI-6025E, è stata sfruttata nel progetto per la misurazione del tempo di volo del proiettile e del ritardo di attivazione dell'elettrovalvola che ne controlla lo sparo.

La misurazione della durata di un impulso avviene contando il numero di periodi del clock in ingresso sul pin G_SOURCE che intercorrono tra un cambiamento di livello del segnale G_GATE e il corrispondente ritorno al livello di partenza. La figura 3.13 permette di capire meglio il meccanismo.

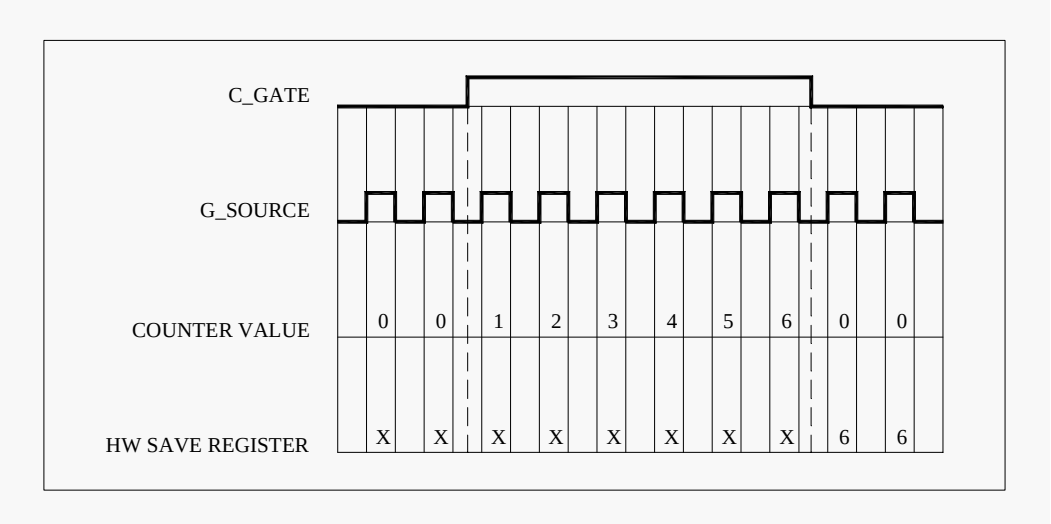


Figura 3.13. Misurazione della durata di un impulso.

In figura 3.13 viene mostrato il diagramma di funzionamento del timer/contantore che misura la durata dell'impulso sull'ingresso C_GATE col passo temporale fornito dall'impulso di clock sull'ingresso G_SOURCE. Quando il segnale di gate effettua una transizione che lo porta dal livello basso a quello alto, il contatore viene incrementato di 1 ad ogni rise del clock. Nel Save Register è contenuto il valore del conteggio in corso, mentre l'HW Save Register conterrà, al termine dell'impulso, il numero di periodi di clock di durata del segnale. Conoscendo la frequenza del clock è possibile conoscere in modo esatto la durata del segnale.

L'opportuna programmazione dei registri della scheda permette, se necessario, di far partire il conteggio da un valore eventualmente diverso da zero.

3.6.4 Generazione di una onda quadra

Per il controllo dei servomotori è necessaria un segnale modulato in PWM, come illustrato nel paragrafo 3.3.2, quindi una onda quadra di frequenza e duty-cycle impostabili via software.

Per poter generare tale forma d'onda è necessario, innanzitutto, selezionare un clock in ingresso al contatore: il contatore effettua un passo del conteggio in occasione del rise (transizione 0-1) o del fall (transizione 1-0) del segnale di clock, a seconda delle impostazioni.

Il clock può essere selezionato tra due diversi input: generato on-board dalla scheda stessa, con la possibilità di scegliere tra la frequenze di 20 MHz oppure di 100 KHz, eventualmente scalabili, oppure può essere fornito dall'esterno mediante un pin apposito nel connettore della scheda stessa. Viste le specifiche temporali richieste nel pilotaggio dei servomotori è stato impostato un clock con periodo di 10 microsecondi.

Il periodo e la durata dell'impulso dell'onda quadra generata vengono espressi come multipli interi del periodo di clock, e tali valori di moltiplicazione vengono memorizzati in appositi registri in fase di programmazione dei timer.

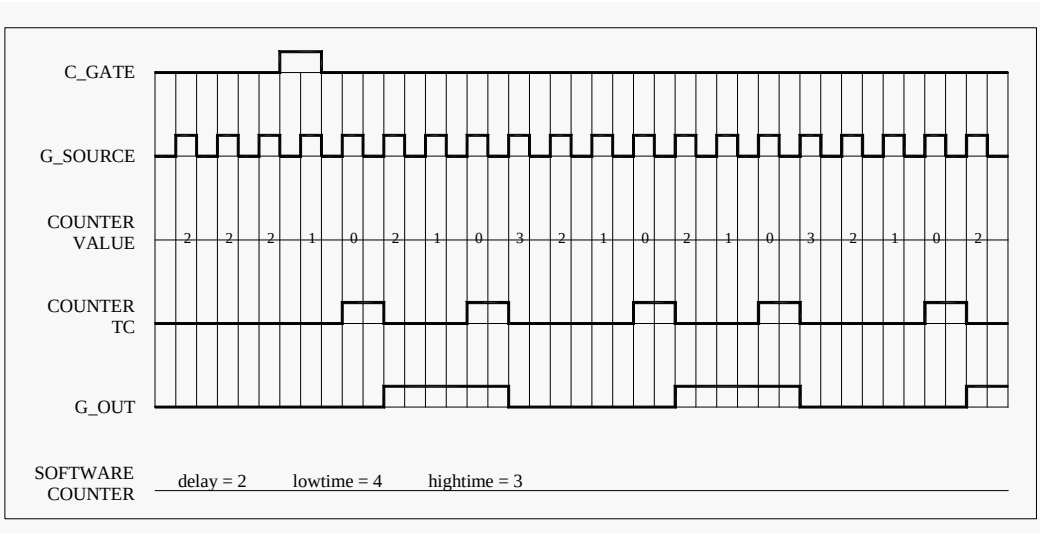


Figura 3.14. Generazione di un treno continuo di impulsi.

Il funzionamento dei contatori come generatori di onda quadra è esemplificato in figura 3.14, nel quale viene viene generata una onda quadra con un high-time pari a 3 periodi di clock e un low-time pari a 4 periodi di clock.

In effetti, al momento di specificare i parametri che servono a generare la forma d'onda, non è possibile indicare direttamente il periodo dell'onda stessa, ma va impostato il suo low-time, sempre relativo al periodo del clock utilizzato; dovendo variare la durata dell'impulso bisogna ricalcolare il valore del low-time, se si desidera mantenere costante la frequenza del segnale in uscita!

Si noti la presenza di un segnale di gate, che abilita o inibisce il conteggio: questo segnale ha una sorgente che può essere impostata via software. Il relativo morsetto è collegato ad una delle uscite digitali della scheda stessa, in modo che tale uscita funga di fatto da interruttore per il funzionamento del servomotore controllato dal timer. E' bene sottolineare che tale segnale non agisce da interruttore sull'alimentazione del servo, quindi, nonostante il servo non applichi più alcuna coppia al proprio carico, esso è comunque alimentato!

Un cenno particolare deve essere tributato all'operazione di cambiamento della durata dell'impulso. Questo può avvenire senza interrompere il treno di impulsi in uscita grazie alla presenza di due banchi di registri per ciascun contatore. Il contatore utilizza un solo banco di registri per volta, mentre l'altro viene usato per caricare i valori corrispondenti alla nuova forma d'onda desiderata: una volta caricati i nuovi valori, si effettua lo switch tra un banco di registri e quello aggiornato; la successiva modifica ai parametri viene effettuata scrivendo nel primo banco ed effettuando nuovamente lo switch.

Queste operazioni di accesso alternato ai banchi di registri richiedono una attenta programmazione dei registri in fase di setup della scheda.

3.7 Interfacciamento scheda-hardware

I segnali che provengono dalla scheda NI PCI-6025E e che vengono utilizzati per pilotare la valvola elettro-pneumatica, il puntatore laser e i due servomotori hanno precise caratteristiche elettriche. La tensione, in standard TTL, può assumere i due valori discreti di 0 e +5 volt, e la corrente erogabile non può superare i 5 milliampere: oltre questo valore si rischia di danneggiare la scheda stessa. Inoltre, come già esposto nei paragrafi relativi, il puntatore laser e l'elettro-valvola vanno alimentate rispettivamente a +3V e a +12V. Si è reso così necessario fabbricare un circuito che, a partire dai segnali ricevuti dalla PCI-6025E, fornisse ai dispositivi dei segnali con adeguate caratteristiche di tensione e potenza.

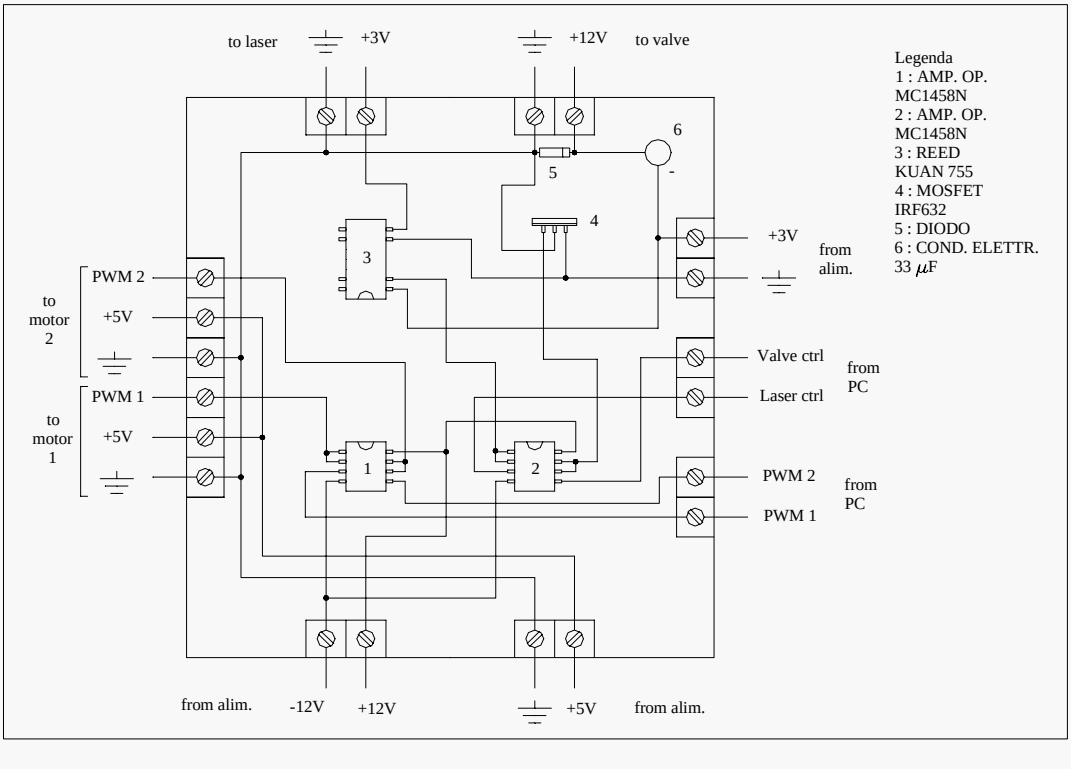


Figura 3.15. Schema di montaggio del circuito elettronico di potenza.

Il circuito realizzato è schematizzato in figura 3.15, in una rappresentazione che privilegia l'identificazione corretta dei pin di input e di output, una sorta di schematizzazione di alto livello, ovvero uno schema di montaggio, mentre in figura 3.16 si ha uno schema di livello più basso.

Il circuito si avvale di 4 amplificatori operazionali utilizzati come inseguitori di tensione, ciascuno dei quali riceve in ingresso uno dei segnali provenienti dalla scheda National Instruments, in modo da isolare la scheda dai carichi presenti a valle dei buffer. Gli inseguitori di tensione provvedono, inoltre, a fornire la potenza necessaria a tali carichi.

L'adeguato livello di tensione da fornire al laser è garantito da un interruttore REED comandato in tensione, il cui segnale di controllo proviene dalla scheda National passando per i buffer. Tale interruttore consente o meno il passaggio di corrente verso il carico ad una tensione adatta al dispositivo. Il REED è stato utilizzato in quanto può fornire correnti elevate, provenienti dall'alimentazione, pur essendo comandato a tensioni in standard TTL ed a bassa potenza. Per quanto riguarda l'azionamento dell'elettrovalvola, il REED è una soluzione inaccettabile: esso si è rivelato fonte di disturbi enormi, che si sono manifestati con bruschi movimenti del robot. Evidentemente veniva disturbata il treno di impulsi che controlla i servo. Per ovviare a questo inconveniente si è preferito sfruttare un MOSFET di potenza, utilizzato come interruttore sull'alimentazione della valvola.

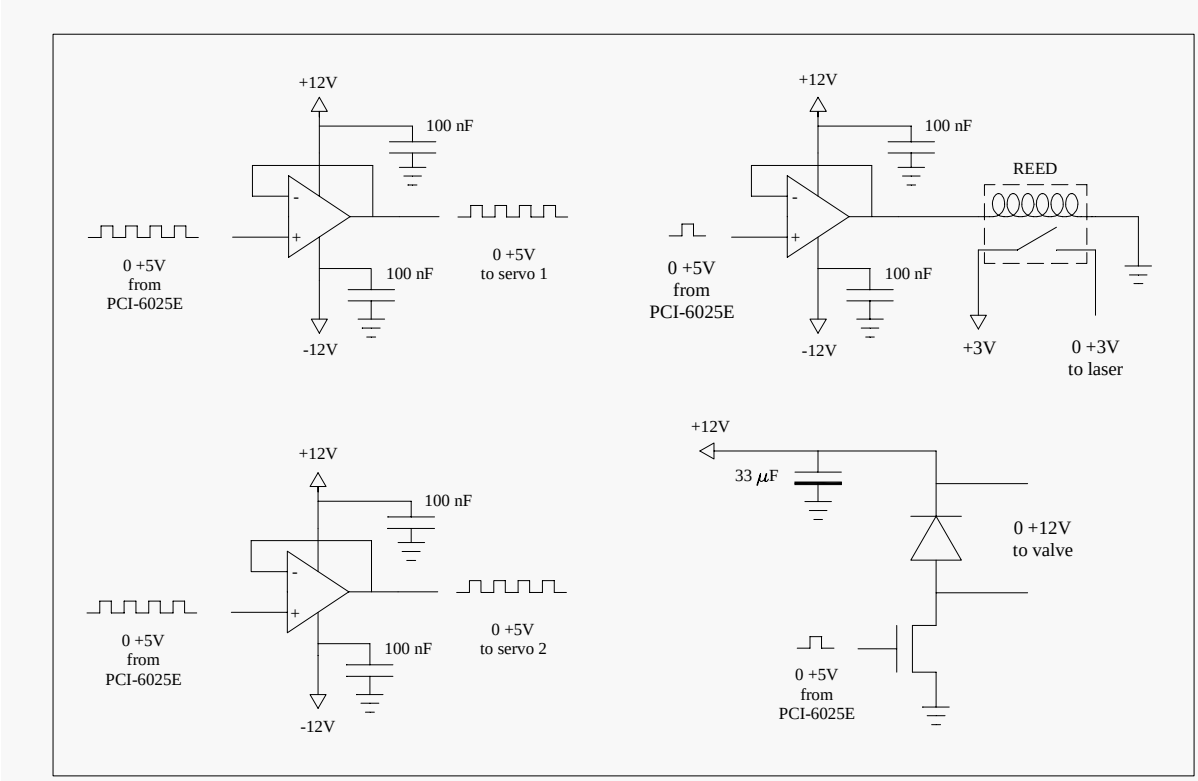


Figura 3.16. Schema elettrico del circuito elettronico di potenza.

Il circuito riceve in ingresso diverse alimentazioni: relativamente al segnale di massa si hanno –12V, +3V, +5V e +12V. Le tensioni di +12V e –12V servono ad alimentare i circuiti integrati su cui sono presenti gli amplificatori operazionali; la tensione a +12V, inoltre, è applicata in ingresso ad uno degli interruttori comandati in tensione; i +3V sono in ingresso all'altro interruttore, mentre i +5V vengono passati direttamente all'uscita per alimentare i servomotori. Altri 4 ingressi sono costituiti dai segnali provenienti dalla scheda PCI-6025E. Le uscite sono logicamente raggruppate nel modo seguente: massa, +5V e segnale di controllo per ciascun servo; massa e alimentazione a +3V per il laser, massa e alimentazione a +12V per l'elettrovalvola.

3.7.1 Alimentazione e massa

L’alimentazione dei circuiti elettronici è garantita da un alimentatore per PC di circa 75 watt di potenza, che fornisce tensioni di +5V, -12V e +12V, che alimentano i servomotori, gli amplificatori operazionali e l’elettrovalvola. Un secondo alimentatore, di potenza massima erogabile pari a 1 watt, fornisce la tensione di +3V necessaria ad alimentare il diodo laser. Il motivo per cui viene utilizzato un secondo alimentatore per fornire la tensione necessaria al funzionamento del laser invece di ricavarla dai +5V piuttosto che dai +12V è che, nel corso dello sviluppo, si sono utilizzati altri puntatori laser, aventi tensioni di alimentazione differenti. Con la soluzione adottata è sufficiente cambiare l’alimentatore esterno, senza dover intervenire sul circuito. In più, l’alimentatore utilizzato permette di impostare diversi livelli di tensione (3, 4.5, 6, 9 e 12 volt), cosa che ha consentito di utilizzare lo stesso alimentatore anche cambiando la tensione richiesta dal puntatore laser.

La massa dei due alimentatori viene posta in comune all’interno del circuito elettronico di potenza, ed un filo di massa viene portato all’opportuno piedino della scheda PCI-6025E, in modo da fornire il riferimento per i segnali digitali in ingresso e in uscita.

3.8 Il robot a 2 gradi di libertà

DEA utilizza una torretta costruita in alluminio per limitare i pesi e i momenti di inerzia delle parti rotanti. Alla torretta vengono imbullonati i servo, il puntatore laser e la canna nella quale viene inserita la pallina da sparare. Il tutto è rappresentato in figura 3.17.

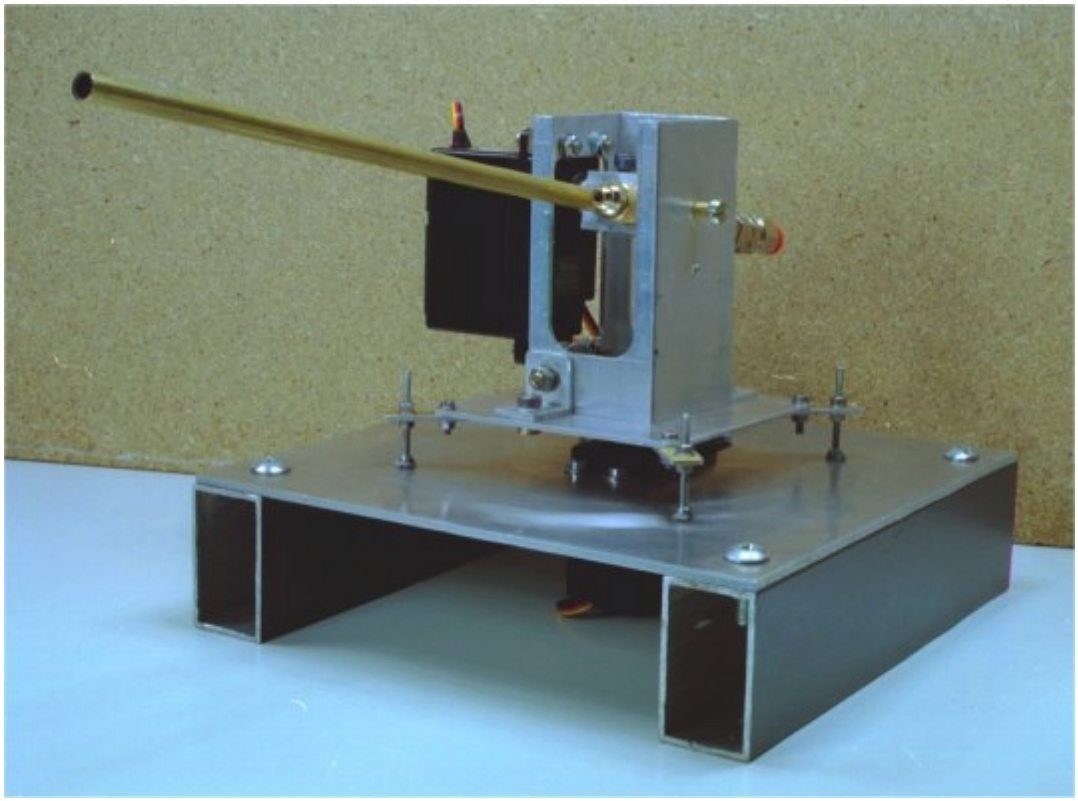


Figura 3.17. Il robot a 2 gradi di libertà.

I servomotori sono necessari a garantire la mobilità alle strutture meccaniche. Tali strutture sono direttamente fissate all'albero dei servo, senza la necessità di riduzioni a cinghia o ad ingranaggi, in quanto tali riduzioni sono già incluse nello chassis dei servo. Questo fatto può provocare delle sollecitazioni laterali indesiderate all'albero dei servomotori, in particolare al servo che deve portare in rotazione la torretta, sulla quale sono montati l'altro servo e i dispositivi

di puntamento, e che determinano dei momenti di inerzia e di flessione relativamente elevati da caricare su un solo albero. Per scongiurare possibili danni al sistema dovuti a questo inconveniente è stata predisposta la piattaforma sulla quale viene imbullonata la torretta.

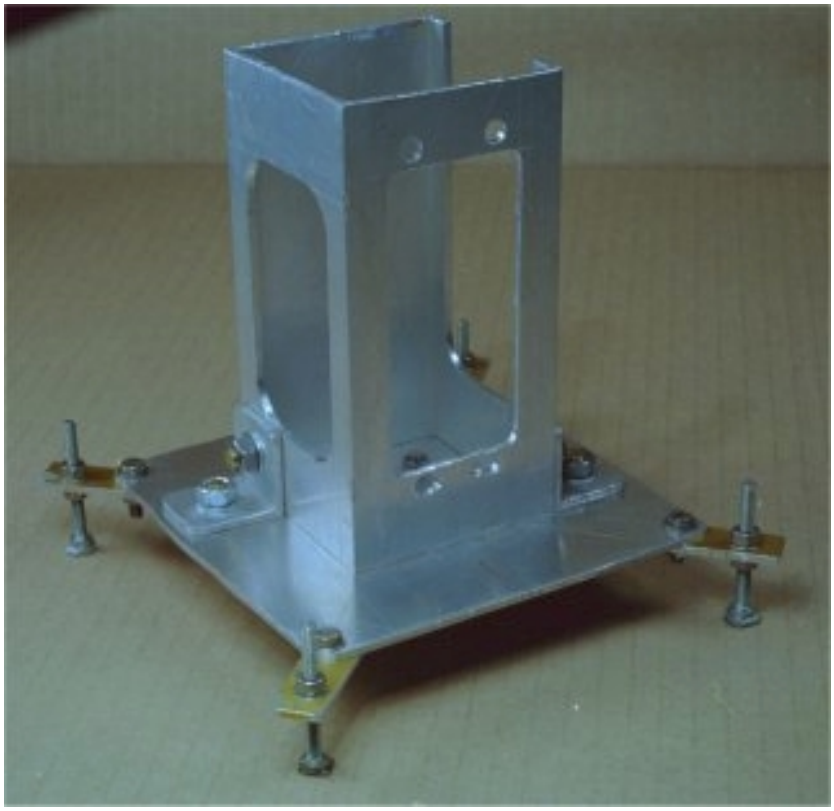


Figura 3.18. Piattaforma rotante del sistema di puntamento.

La piattaforma, rappresentata in figura 3.18, viene fissata all'albero, e dispone di 4 piedini regolabili a vite indipendentemente l'uno dall'altro, in modo da appoggiare alla superficie inferiore e tali da limitare le oscillazioni della torretta stessa.

La calibrazione della "altezza dal suolo" dei piedini viene calibrata in fase di montaggio della struttura: dopo aver avvitato il servomotore orizzontale nella propria sede, prima si fissa la piattaforma all'albero del servo, successivamente si

regolano le singole viti in modo che il piedino sfiori la superficie di appoggio, in modo che, facendo ruotare la piattaforma, si crei il minor attrito radente possibile. Per limitare l'attrito viene utilizzato anche dell'olio lubrificante, e la base di appoggio dei piedini è costituita da materiale atto anch'esso a ridurre l'attrito.

Particolare cura, infine, è stata riservata al progetto e alla costruzione del blocchetto sul quale vanno montati il laser e la canna di sparo, e che va poi fissato all'albero del servomotore dislocato sulla torretta.

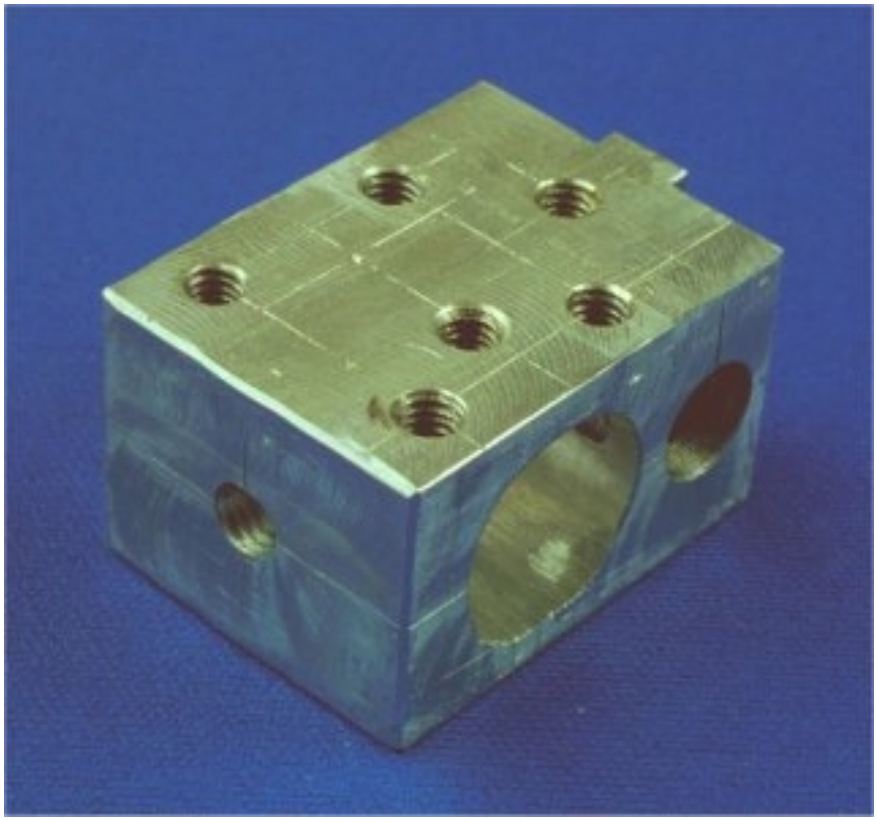


Figura 3.19. Blocco sul quale vengono montato laser e canna.

Come si vede in figura 3.19, il blocchetto è forato longitudinalmente, con fori di dimensione opportuna per potervi innestare il puntatore laser e la canna. Superiormente sono stati praticati e filettati dei fori nei quali vanno inseriti i grani e le viti che mantengono in posizione i due dispositivi.

La vite della canna ha soltanto la funzione di bloccarla in posizione, mentre il puntatore viene fissato da 4 viti, in modo da consentire un minimo di regolazione della direzione del fascio: una volta inserito il puntatore nella posizione desiderata (operazione che è possibile in quanto il foro è leggermente asolato), si stringono le 4 viti in modo da fissarlo in tale posizione. La vite centrale, rispetto all'asse del puntatore, infine, serve a tenere premuto l'interruttore dell'alimentazione del laser: questa non va stretta oltremodo, in quanto si potrebbe danneggiare l'interruttore stesso; va semplicemente stretta fino a quando, fornita l'alimentazione al puntatore, questo si accende.

3.9 Compressore d'aria e proiettili

DEA, per colpire il bersaglio, spara delle palline identiche a quelle utilizzate nelle comuni pistole ad aria compressa. Si tratta di palline di plastica dura del diametro di circa 6 millimetri e del peso di circa 10 grammi. Si è verificato che questi proiettili non sono in grado di nuocere alle persone, anche se sparati direttamente. L'unica situazione di pericolosi può avere nel caso in cui una pallina colpisse un essere vivente agli occhi, ed è per questo motivo che gli esperimenti che hanno riguardato i test di sparo sono stati effettuati durante le ore serali e notturne, in modo da non costituire pericolo per i frequentatori del laboratorio di robotica. Io stesso ho preso delle opportune precauzioni a tal proposito.

Per sparare la pallina è stato utilizzato un compressore d'aria della Fini con le seguenti caratteristiche:

- pressione massima pari a 8 atmosfere;
- motorino con lubrificazione a secco (senza olio);
- capacità serbatoio di 5 litri.

3.10 Stima dei ritardi

Come illustrato al paragrafo 3.10, non è possibile sparare con precisione se non si tiene conto di due importanti parametri: il ritardo di attivazione dell'elettrovalvola e il tempo di volo del proiettile. Per stimare i due valori è stato predisposto un apposito sistema hardware-software costituito da un sensore di urto e da un circuito elettronico di interfaccia tra questo e la scheda NI PCI-6025E. Il software è invece un programma eseguito sotto HARTIK che gestisce l'attivazione della valvola e il conteggio del tempo da parte dei timer messi a disposizione dalla scheda della National.

Durante i primi test che sono stati affrontati per stimare le variabili in gioco, si è riscontrato che il tempo di volo, a parità di distanza, variava fino al 20-30% a seconda che il compressore si fosse appena caricato, quindi ci si trovasse in condizioni di massima pressione, oppure nel caso in cui fossero già stati sparati diversi colpi, con una conseguente perdita di pressione. Per ovviare a questo inconveniente si è pensato di effettuare tutte le prove con il compressore alla massima pressione. Per fare ciò, ad ogni colpo sparato, dopo aver registrato i dati necessari, il software comanda la valvola perchè rimanga aperta fino a portare la pressione del compressore al livello di ricarica. In questo modo è possibile registrare dei dati sul delay-time e sul fly-time coerenti tra loro, poichè rilevati in condizioni uniformi. Questo, comunque, è vero solo in parte, poichè si suppone che il compressore non rispetti una soglia molto precisa per terminare la ricarica, quindi è possibile che ci siano lievi differenze di pressione fra uno sparo e l'altro. In più, si è notato che, effettuando cicli di carica e sparo molto serrati, il compressore tende a surriscaldarsi, con la conseguenza che il fenomeno appena citato si accentua notevolmente. Per evitare l'eccessivo riscaldamento del compressore e la perdita di precisione nella ricarica si è dovuto aumentare la distanza temporale tra uno sparo e l'altro, introducendo anche delle pause relativamente lunghe per permettere il raffreddamento del dispositivo. Tutti questi

tempi morti hanno allungato notevolmente la durata dei test, ma si sono dimostrati indispensabili per ottenere dei risultati significativi.

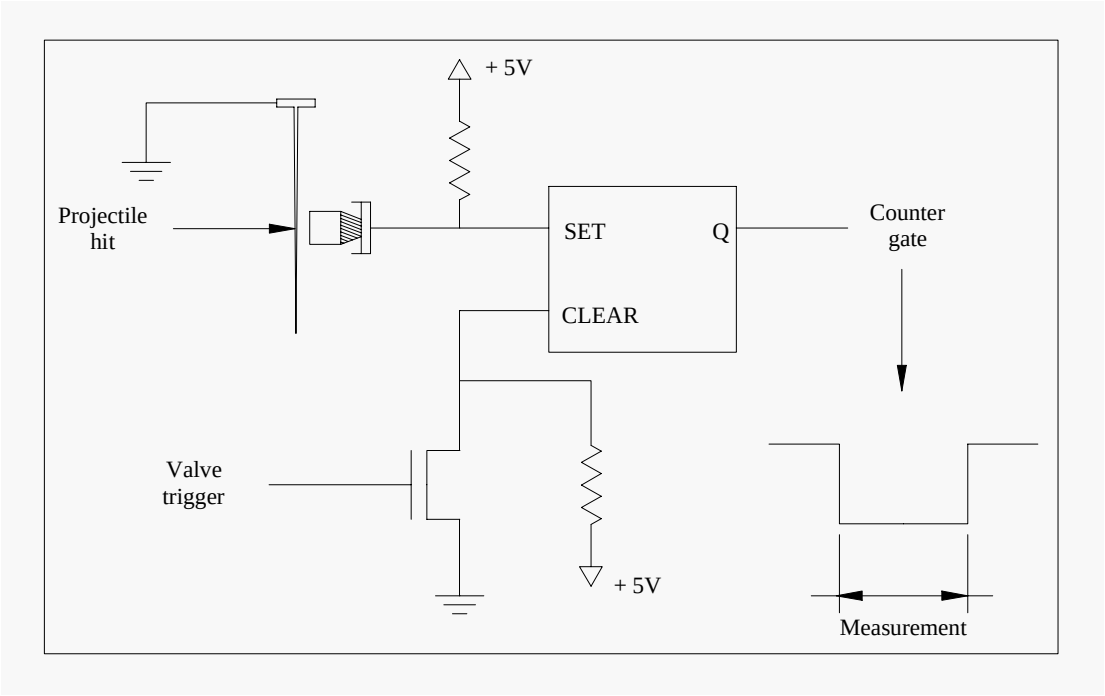


Figura 3.20. Dispositivo per i test di velocità proiettile e ritardo valvola.

Per misurare il tempo intercorso tra l'istante in cui viene fornito l'impulso di apertura dell'elettrovalvola e il momento in cui il pallino colpisce il bersaglio è stato fabbricato un dispositivo (schematizzato in figura 3.20) che è stato utilizzato unitamente ai contatori/timer della scheda NI PCI-6025E opportunamente programmati. Inoltre si è dovuto fabbricare un semplice circuito elettronico (schematizzato in figura 3.21) con un flip-flop set-reset in grado di mantenere il segnale all'ingresso della scheda della National.

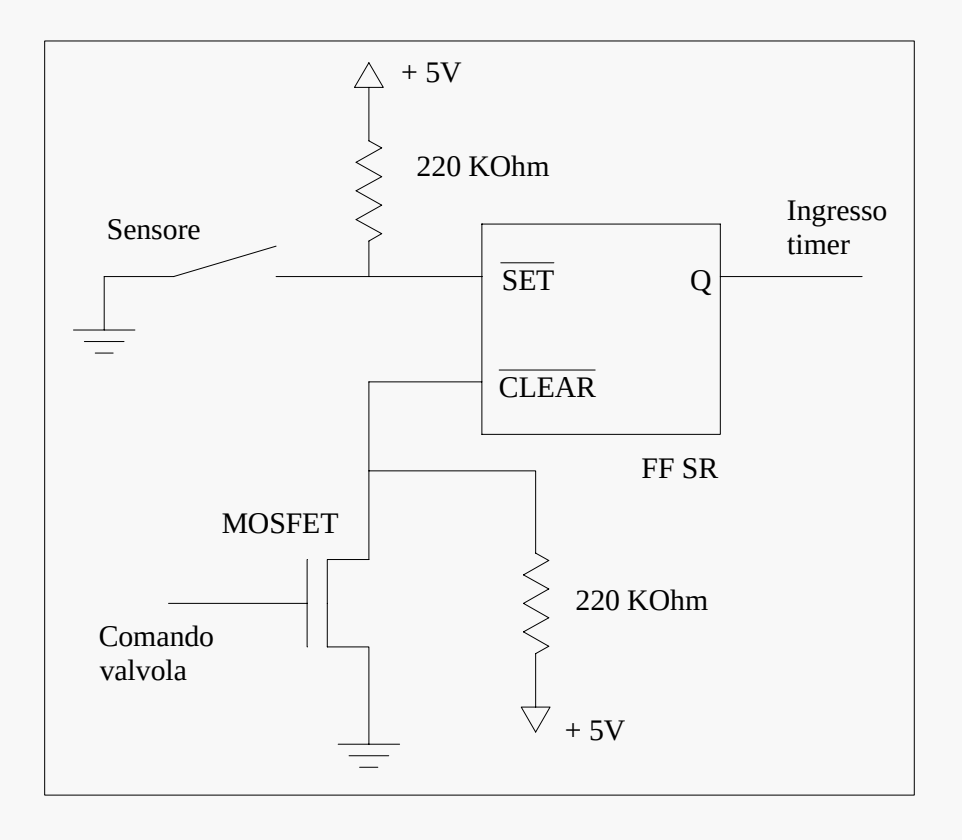


Figura 3.21. Circuito elettronico di servizio.

Ciò che si doveva costruire era un sistema che generasse un impulso il cui inizio coincidesse col segnale di sparo e la cui fine corrispondesse all'istante di impatto del proiettile col bersaglio. Il diagramma di funzionamento del circuito elettronico è riportato in figura 3.22.

Per realizzare quanto illustrato si è connessa l'uscita della scheda che comanda l'elettrovalvola con il piedino di reset del flip-flop, passando attraverso il MOSFET che funge da inverter, in quanto il comando della valvola funziona in logica positiva, al contrario del flip-flop che lavora in logica negativa. Il dispositivo meccanico, invece, è sostanzialmente un interruttore normalmente aperto, che, quando viene colpito dal proiettile, si chiude verso massa, attivando il segnale si set, che fa cambiare stato al bistabile.

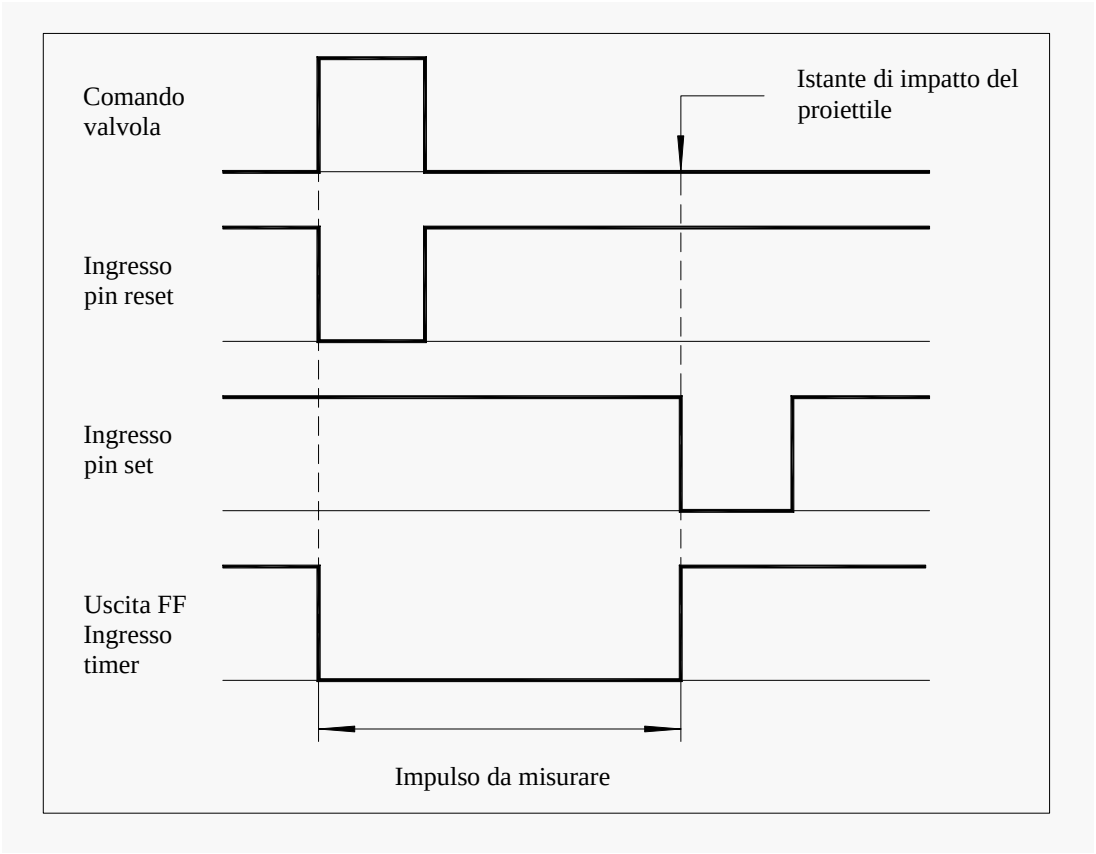


Figura 3.22. Diagramma di funzionamento del circuito elettronico.

Come noto, quando in un flip-flop SR gli ingressi sono entrambi attivi l'uscita è errata, in quanto non viene rispettata la consistenza tra il livello dell'uscita e del suo negato. Il bistabile utilizzato, in questa particolare condizione, presenta i due valori al livello alto. In linea di principio, potrebbe nascere un problema se il proiettile colpisse il bersaglio prima che il segnale che comanda la valvola torni al livello basso (ricordo che funziona in logica positiva): entrambi gli ingressi del flip-flop si troverebbero al livello basso, cioè attivi. Tenendo conto del comportamento del bistabile in questa situazione, il circuito elettronico è appositamente studiato per ovviare a questo inconveniente: lavorando in logica negata, il segnale di apertura della valvola causa un cambiamento di stato del

segnale di uscita, portandolo al livello basso; quando l'interruttore viene chiuso dall'urto della pallina, indipendentemente dal livello del segnale della valvola, l'uscita del set-reset torna al valore alto, fermando il contatore. Ora, se il segnale che controlla la valvola è basso l'uscita rimane alta, mentre se è basso (cioè la valvola è ancora aperta) l'uscita torna bassa. Questo fatto non crea però alcun problema, in quanto il contatore è programmato per misurare la durata del primo impulso in logica negata che si presenta al suo ingresso.

3.11 L'elaboratore

Il computer utilizzato per sviluppare il progetto è un Hewlett Packard con le seguenti caratteristiche:

- processore Intel Pentium con clock a 133 MHz;
- memoria di 16 MB;
- hard disk da 4 GB;
- 2 slot EISA e 2 slot PCI.

Sul PC, dotato di sistema MS Windows 95, è stato montato l'hardware consistente nella scheda National Instruments PCI 6025E e nel frame-grabber PXC200L della Imagenation. Sullo stesso PC gira, in modalità MS-DOS, il sistema real-time HARTIK.

Le caratteristiche dell'elaboratore, in particolare la non eccelsa potenza di calcolo, se paragonata a macchine dell'ultima generazione, ha influenzato in diversi aspetti le decisioni circa lo sviluppo del progetto. Soprattutto l'elaborazione dell'immagine per la ricerca del target è stata penalizzata dalle prestazioni del PC: ci si è dovuti limitare ad una soglia per rendere sufficientemente veloce l'operazione che, pur con tutti gli espedienti descritti nel capitolo 2, paragrafo 6, ha un tempo di esecuzione variabile tra i 30 e i 50 millisecondi, quindi circa il doppio del frame-rate garantito dal grabber.

Altro aspetto che ha risentito della scarsità di MIPS a disposizione è l’elaborazione grafica relativa all’interfaccia utente.

In sostanza si sono dovuti trascurare in qualche modo i due precedenti aspetti per privilegiare il corretto funzionamento del sistema di tracking e calibrazione, a discapito quindi dello styling visivo dell'applicazione.

Capitolo 4

Reti neurali

4.1 Le reti neurali

In questo capitolo, dopo una breve introduzione generale sulle reti neurali, verrà descritta nel dettaglio la rete error backpropagation, in modo da permettere una migliore comprensione dei risultati ottenuti nei test effettuati su tali reti.

Innanzitutto, quando si parla di reti neurali, si dovrebbe più propriamente utilizzare il termine “reti neurali artificiali” (ANN – Artificial Neural Net): le reti neurali biologiche sono molto più complesse dei modelli matematici che vengono utilizzati nelle ANN; nel seguito utilizzeremo comunque il termine rete neurale (NN – Neural Net), senza pericolo di ambiguità.

4.2 Cos’è una rete neurale

Non esiste una definizione universalmente accettata di rete neurale. In genere può essere definita tale una rete composta da semplici unità operative tra loro interconnesse, ciascuna possibilmente dotata di una certo quantitativo di

memoria locale. Le unità sono collegate tra loro tramite canali di comunicazione che veicolano dati in genere numerici oppure simbolici opportunamente codificati. La singola unità opera a livello locale sui suoi soli dati e sugli input che riceve dalle altre unità. Talune reti si ispirano a modelli biologici, con l'obiettivo produrre un sistema "intelligente", tentando di simulare il funzionamento del cervello animale. Molte reti neurali dispongono di una sorta di regola di apprendimento, mediante la quale vengono aggiustati i pesi delle interconnessioni a partire dai dati in ingresso. Le reti neurali hanno in genere un elevato grado di parallelismo, dal momento che ciascuna unità è largamente indipendente dalle altre. Questo non è però un requisito fondamentale, visto che un semplice interpolatore lineare formato da due unità è a tutti gli effetti una rete neurale. Di seguito vengono riportate alcune definizioni di rete neurale:

Secondo il Neural Network Study del DARPA (AFCEA International Press, 1988):

> ... una rete neurale è un sistema composto da più elementi di processo semplici posti in parallelo la cui funzione è determinata dalla struttura della rete, dalla forza delle connessioni e dall'elaborazione locale dei singoli elementi.

Secondo Neural Networks: A Comprehensive Foundation, di S. Haykin (Macmillan, 1994):

> Una rete neurale è assimilabile a un processore distribuito in parallelo che ha una naturale propensione all'immagazzinamento di conoscenza e al rendela disponibile per l'uso. Simula il cervello sotto due aspetti:
>
> 1) La conoscenza è acquisita mediante un processo di addestramento;
> 2) Il peso delle connessioni, come le sinapsi per il cervello, sono utilizzate per memorizzare le informazioni;

Infine, secondo Neural Network For Pattern Recognition, di A. Nigrin (The MIT Press, 1993):

> Una rete neurale è un circuito composto da un numero molto elevato di semplici unità di elaborazione. Ogni elemento opera solo su informazioni

locali. In più, ciascuna unità funziona in modo asincrono dalle altre: non esite un clock di sistema.

4.3 Cosa può fare una rete neurale (e cosa no!)

In linea di principio, una rete neurale può computare ogni funzione computabile, quindi può fare ogni cosa che può essere fatta con un normale computer (Valiant, 1988; Siegelmann e Sontag, 1999). Nella pratica le reti neurali sono particolarmente adatte a problemi quali la classificazione, l'approssimazione e il mapping di funzioni in modo che siano tolleranti a qualche forma di imprecisione, quando si disponga di un elevato numero di dati relativi al problema e, magari, non è possibile applicare algoritmi diretti di analisi dei dati. Ciò è possibile, in genere, sotto una ipotesi non sempre verificabile: lo stato della la rete evolve grazie a tecniche di minimizzazione di un errore che devono essere in grado di avvicinarsi arbitrariamente al minimo globale dell'errore stesso.

Il funzionamento della rete è legato alla bontà dell'algoritmo di minimizzazione adottato, che talvolta si può rivelare computazionalmente intrattabile se non in casi semplificati, oppure, in altri casi, può essere ingannato dalla presenza di minimi locali. Il problema dei minimi locali diventa tanto più rilevante quanto più la funzione da approssimare è complessa, oppure l'imprecisione nei dati è elevata. Come vedremo, sono stati studiati diversi metodi per fuggire dai minimi locali.

Tipici problemi che una rete neurale non può risolvere, se non memorizzando l'intero set di esempi sono, a grandi linee, i seguenti:

- fattorizzazione di interi con elevato numero di cifre;
- determinazione della primalità di un numero intero con elevato numero di cifre;
- decrittazione di dati che vengano criptati con un qualsiasi "buon algoritmo".

L’aspetto più importante da tenere presente è che non esistono metodi per addestrare una rete neurale in modo da “creare” informazioni che non siano contenute nel training set: nessun tipo di rete neurale può imparare ad estrapolare dei dati se gli esempi che le vengono forniti non sono sufficientemente "esplicativi".

4.4 Tipologie di reti neurali

Partendo dalla considerazione che esiste un numero elevatissimo di tipi di rete neurale, di seguito sono indicati i principali parametri che classificano le reti.

I principali tipi di tecniche di apprendimento sono il supervised e l’unsupervised learning.

- Nel supervised learning il risultato desiderato (target) è noto e viene fornito alla rete in fase di apprendimento in modo che i pesi sinaptici possano essere modificati per approssimarlo. Terminato l’addestramento vengono forniti alla rete i soli input, e questa fornirà uscite in accordo con la qualità dell’apprendimento realizzato.
- Nell’unsupervised learning non viene presentato alla rete alcun target, e i pesi vengono modificati solo in base ai dati in input. Questo tipo di rete, in genere, effettua un qualche tipo di compressione dei dati, come la riduzione di dimensione o il clustering.

Le principali topologie di rete neurale sono la feedforward e la feedback:

- In una feedforward NN le connessioni tra le unità non formano cicli. Questo tipo di rete in genere risponde velocemente agli stimoli di ingresso. La maggior parte delle reti feedforward possono essere addestrate usando una ampia varietà di metodi numerici (gradiente coniugato, metodi trust-region, metodi quasi-Newton), oltre a metodi specifici sviluppati dai ricercatori del settore;

- Nelle feedback NN (dette anche ricorrenti) esistono dei cicli nelle connesioni. In teoria, una rete a feedback può iterare anche per molto tempo su ciascun input. Solitamente tale tipo di rete è più difficile da addestrare che non una di tipo feedforward.

Le reti neurali si distinguono in base al tipo di dato che elaborano, che può essere qualitativo o quantitativo:

- Le variabili qualitative assumono un numero finito (numerabile) di valori, per esempio “maschio” o “femmina”, oppure “rosso”, “blu” o “giallo”. Tali valori devono essere opportunamente codificati sotto forma di numeri per poter essere elaborati dalle NN.
- Le variabili qualitative sono, in genere, misurazioni numeriche di qualche attributo, come una distanza in metri. Le misurazioni devono essere effettuate in modo che le loro relazioni aritmetiche rispecchino una corrispondente relazione tra gli attributi misurati. Un supervised learning con un target di variabili quantitative viene chiamato *regressione*.

4.5 Reti Error Backpropagation (EBP)

Di seguito si concentrerà l'attenzione su reti neurali di tipo particolare: le reti Error Backpropagation. Questo per il semplice motivo che DEA utilizza esclusivamente questo tipo di rete neurale, ed inoltre perchè i test, i cui risultati vengono commentati in questo stesso capitolo, sono stati condotti proprio su reti error backpropagation. Nel seguito della trattazione, con il termine Error Backpropagation, o, più semplicemente, backpropagation, ci si riferirà ad una rete feedforward istruita con un algoritmo Backpropagation.

Una rete backpropagation prevede sempre uno strato di neuroni di ingresso, ai quali vengono applicati i segnali di input. Similmente esiste sempre uno strato di neuroni di uscita, dai quali si preleva l’output. Tra questi due strati sono posti, in genere, degli strati di neuroni intermedi, detti *nascosti*. Il caso limite prevede

l’assenza di strati nascosti, e la rete che ne deriva, che quindi ha i soli strati di input e di output, è detta *perceptron*. E’ stato dimostrato che lo strato di ingresso non ha alcuna funzione relativa all’apprendimento: solo gli strati successivi sono responsabili dell’immagazzinamento di un’informazione da parte della rete; sono cioè detti *strati decisionali*. Spesso nascono delle incomprensioni nel definire il numero di strati di una rete neurale: in ogni caso andrebbe specificata la natura dei layer (decisionali, nascosti, etc.). Nel seguito, se non altrimenti specificato, una rete con n strati viene considerata avente 1 strato di ingresso e n-1 strati decisionali.

La schematizzazione di una rete backpropagation a 3 strati e con funzione di trasferimento a sigmoide è riportata in figura 4.1.

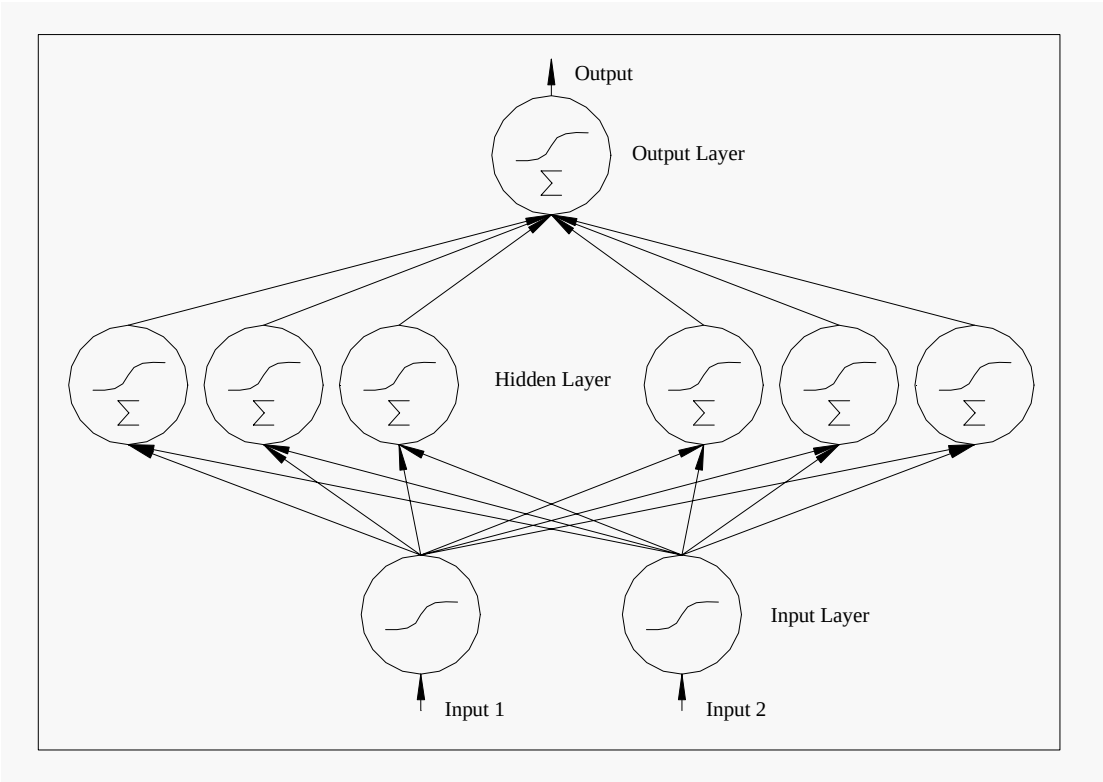


Figura 4.1. Rete neurale a 3 strati, con 2 ingressi e un’uscita.

4.5.1 Funzione di trasferimento del neurone

Lo stato di un neurone viene calcolato effettuando una sommatoria pesata dei suoi ingressi. L'uscita viene determinata a partire dallo stato interno, valutando una opportuna funzione di trasferimento (f.d.t.). Spesso, per lo più in applicazioni di riconoscimento di forme, si usa una f.d.t. a soglia: il neurone restituisce un certo valore se lo stato è maggiore della soglia, mentre se è minore restituisce, in genere, 0. Questo corrisponde ad una f.d.t. a gradino, con la transizione in corrispondenza del valore di soglia e l'ampiezza corrispondente al valore restituito (figura 4.2).

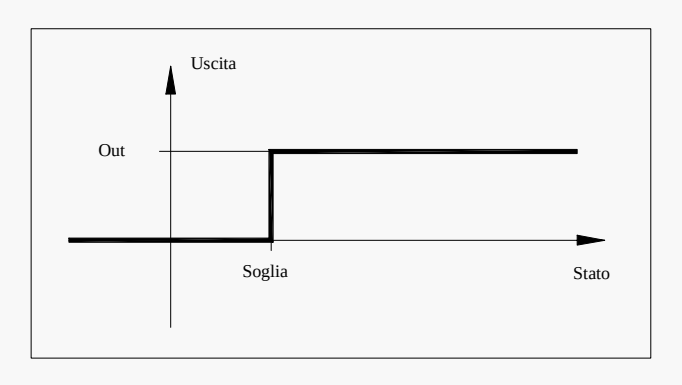


Figura 4.2. Funzione di trasferimento a soglia.

Altre volte viene usata una f.d.t. lineare (con pendenza opportuna): allora l'uscita è equivalente allo stato del neurone opportunamente scalato per tener conto della pendenza della f.d.t. (figura 4.3). Una funzione di trasferimento di questo tipo, è dimostrato, presenta un problema fondamentale: nel caso di approssimazione di funzioni, l'uscita è, in ogni caso, una combinazione lineare degli ingressi. Non sono quindi possibili approssimazioni se non di tipo lineare.

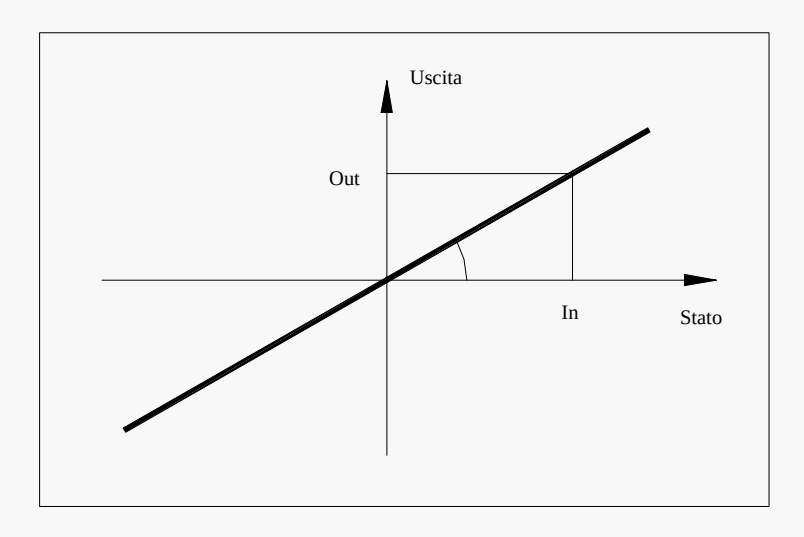


Figura 4.3. Funzione di trasferimento lineare.

In molte situazioni è indispensabile l'uso di funzioni di trasferimento differenziabili in ogni punto ma che siano non lineari, per evitare gli inconvenienti discussi precedentemente. Le più usate f.d.t. di questo tipo sono le funzioni a sigmoide e ad arcotangente. Spesso viene utilizzata la funzione a sigmoide in quanto è possibile variare il comportamento della rete in fase di training agendo sul coefficiente β della sigmoide.

La sigmoide, un grafico della quale è riportato in figura 4.4, è definita dalla seguente funzione:

$$f(x) = \frac{1}{1+e^{-\beta \cdot x}}$$

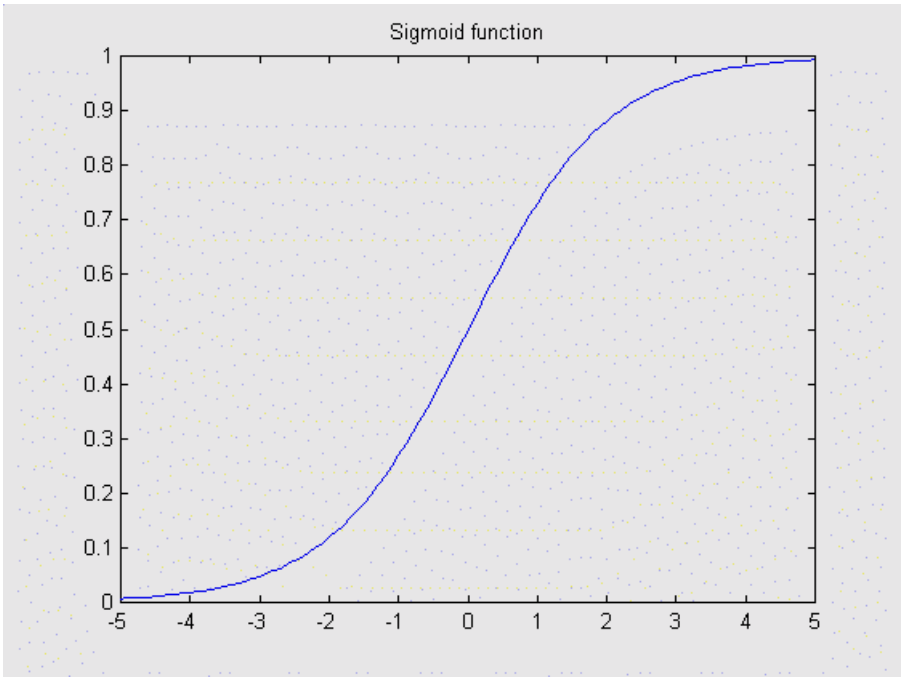


Figura 4.4. Funzione di trasferimento sigmoide.

4.5.2 Algoritmo di apprendimento

Per eseguire il training di una rete neurale di tipo backpropagation viene in genere utilizzato un algoritmo di tipo supervised, cioè nel quale si forniscono alla rete gli ingressi ed i valori desiderati delle uscite (training samples), e si modificano i pesi dei neuroni in modo che la rete impari a far corrispondere gli input ai relativi output. L'algoritmo è efficacemente descritto dalla seguente sequenza di operazioni:

ciclo epoche:
{
    per ogni esempio:
    {
        - si fornisce l'input alla rete;
        - si preleva l'output corrispondente;
        - si calcola la differenza con l'output desiderato;

- si modificano i pesi dei collegamenti tra i neuroni in base a tale errore, con una regola detta delta rule generalizzata, in modo che tale errore diminuisca;

}

- viene calcolato l'errore globale su tutti gli esempi;
- si ripete il ciclo epoche finche l'errore non raggiunge il valore desiderato

}

Un training set è composto dall'insieme dei training samples. Una rete neurale addestrata con l'algoritmo precedente tende quindi a minimizzare l'errore sul training set, cercando di diminuirlo ad ogni ciclo (detto *epoca*). In tal modo, la rete non impara ad associare ogni input ad un particolare output, ma impara a riconoscere la relazione tra ingressi ed uscite per quanto complessa possa essere. Diventa pertanto una scatola nera, la quale non determina esplicitamente la formula matematica che correla input e output, posto che questa esista, ma permette di ottenere risposte significative a partire da dati non compresi nel training set, sia interni al range di valori di addestramento (*interpolazione*), che all'esterno di esso (*estrapolazione*). Questa caratteristica delle reti e detta *potere di generalizzazione*.

4.5.3 La generalized delta rule

L'entità della variazione dei pesi viene calcolata col metodo del gradiente, valutato sulla *funzione errore*. L'applicazione del metodo del gradiente per il training delle reti neurali ha dato origine ad un algoritmo che si basa sulla cosiddetta *delta rule*. La delta rule viene ottenuta con una serie di passaggi matematici: la variazione dei pesi nei neuroni dello strato esterno è

$$\Delta w_{kj} = -\eta \cdot \frac{\partial f_{err}}{\partial w_{kj}}$$

con η = coefficiente di apprendimento;

f_{err} = funzione errore della rete;

w_{kj} = peso del k-esimo neurone di output rispetto a j-esimo neurone nascosto;

che è possibile riscrivere nel seguente modo:

$$\Delta w_{kj} = -\eta \cdot \frac{\partial f_{err}}{\partial I_j} \cdot \frac{\partial I_j}{\partial w_{kj}}$$

dove $I_j = \sum_{k=1}^{n} w_{kj} \cdot h_k$ è la somma pesata degli n ingressi del k-esimo neurone di uscita;

h_k = output del neurone k-esimo nello strato nascosto.

Definisco ora

$$\Delta_j = \frac{\partial f_{err}}{\partial I_j}$$

quindi posso riscrivere Δw_{kj} come

$$\Delta w_{kj} = -\eta \cdot \Delta_j \cdot \frac{\partial I_j}{\partial w_{kj}}$$

Ora vanno esplicitati i singoli componenti dell'equazione precedente. Immediatamente si ricava

$$\frac{\partial I_j}{\partial w_{kj}} = h_k$$

mentre il delta di variazione viene riscritto come segue:

$$\Delta_j = \frac{\partial f_{err}}{\partial y_j} \cdot \frac{\partial y_j}{\partial I_j}$$

Ora, ricordando che per una rete neurale si utilizza in genere un errore che è il quadrato della differenza tra l'output fornito e l'output desiderato, si ha

$$f_{err} = \frac{(D_j - y_j)^2}{2}$$

dove y_j = j-esimo output della rete;

D_j = j-esimo output desiderato dalla rete neurale;

quindi

$$\frac{\partial f_{err}}{\partial y_j} = \frac{\partial}{\partial y_j}\left(\frac{(D_j - d_j)^2}{2}\right) = -(D_j - d_j)$$

mentre

$$\frac{\partial y_j}{\partial I_j} = \frac{\partial}{\partial I_j}\left(\frac{1}{1+e^{-I_j}}\right) = y_j \cdot (1 - y_j)$$

Ora è possibile scrivere l'equazione della variazione dei pesi in funzione delle grandezze calcolate in precedenza, ottenendo:

$$\Delta w_{kj} = +\eta \cdot (D_j - y_j) \cdot y_j \cdot (1 - y_j) \cdot h_j$$

Gli ultimi due passaggi cambiano leggermente se si considera di parametrizzare l'esponenziale nella funzione sigmoide con un opportuno valore β. In questo caso si ottiene:

$$\frac{\partial y_j}{\partial I_j} = \frac{\partial}{\partial I_j}\left(\frac{1}{1+e^{-\beta \cdot I_j}}\right) = \beta \cdot y_j \cdot (1 - y_j)$$

e l'equazione che determina l'entità della variazione dei pesi diviene

$$\Delta w_{kj} = +\eta \cdot \beta \cdot (D_j - y_j) \cdot y_j \cdot (1 - y_j) \cdot h_j$$

4.5.4 Retropropagazione dell'errore

Il termine Backpropagation si riferisce al metodo utilizzato per la modifica dei pesi negli strati nascosti della rete, basato su una tecnica denominata, appunto, retropropagazione dell'errore. Anche questa tecnica è stata ricavata applicando alcune semplici regole matematiche alle equazioni che caratterizzano una rete neurale. In particolare si è partiti da

$$\Delta w1_{jk} = -\eta \cdot \frac{\partial f_{err}}{\partial w1_{jk}}$$

dove $w1_{jk}$ = peso della connessione tra il k-esimo neurone dello strato nascosto e il j-esimo neurone dello strato precedente (nel caso di rete a 2 strati decisionali, si tratta del layer di ingresso);

$\Delta w1_{jk}$ = variazione del peso in fase di training.

Procedendo come per la delta rule, scrivo

$$\Delta w1_{jk} = -\eta \cdot \frac{\partial f_{err}}{\partial I_k} \cdot \frac{\partial I_k}{\partial w1_{jk}}$$

nella quale, essendo

$$I_k = \sum_j w1_{jk} \cdot x_j$$

ottengo

$$\frac{\partial I_k}{\partial w1_{jk}} = x_j$$

Definisco

$$\Delta_k = -\frac{\partial f_{err}}{\partial I_k}$$

che posso riscrivere come segue:

$$\Delta_k = -\frac{\partial f_{err}}{\partial I_j} \cdot \frac{\partial I_j}{\partial h_k} \cdot \frac{\partial h_k}{\partial I_k}$$

dove h_k = attivazione del k-esimo neurone nascosto.

Δ_k può essere ancora scritto come

$$\Delta_k = -\sum_j \left(\frac{\partial f_{err}}{\partial I_j} \cdot \frac{\partial I_j}{\partial h_k} \cdot \frac{\partial h_k}{\partial I_k} \right)$$

I tre termini tra parentesi sono

$$\frac{\partial f_{err}}{\partial I_j} = \Delta_j$$

$$\frac{\partial I_j}{\partial h_k} = \frac{\partial}{\partial h_k} \left(\sum_k w2_{kj} \cdot h_k \right) = w2_{kj}$$

$$\frac{\partial h_k}{\partial I_k} = \frac{\partial}{\partial h_k}\left(\frac{1}{1+e^{-h_k}}\right) = h_k \cdot (1-h_k)$$

e posso scrivere la variazione del generico peso come

$$\Delta w1_{jk} = \eta \cdot \Delta_k \cdot x_j$$

dove x_j = j-esimo input dello strato sottostante.

Ottengo, dunque

$$\Delta_k = (\sum_j \Delta_j \cdot w2_{kj}) \cdot h_k \cdot (1-h_k)$$

con

$$\Delta_j = (D_j - y_j) \cdot y_j \cdot (1-y_j)$$

calcolato al paragrafo 4.5.3.

4.6 La rete EBP utilizzata in DEA

DEA usa una rete neurale di tipo backpropagation, come quella descritta nei paragrafi precedenti, per associare le coordinate sull'immagine ai valori di pilotaggio dei servomotori, come schematizzato in figura 4.5.

La rete neurale viene dunque utilizzata per approssimare una funzione del tipo

$$f : I \to M \text{ con } I, M \subset R^2 .$$

In realtà DEA utilizza due reti neurali identiche per approssimare due funzioni del tipo

$$f : I \to M \text{ con } I \subset R^2 \text{ e } M \subset R .$$

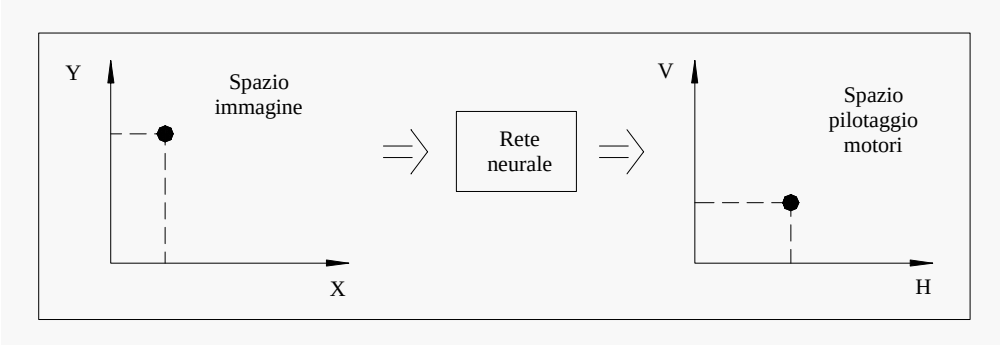


Figura 4.5. Dominio e codominio della funzione approssimata dalla rete

In sostanza si utilizzano due reti neurali alimentate dai medesimi ingressi che vengono addestrate ad approssimare le due componenti della funzione, le quali non sono altro che i valori di pilotaggio del motore orizzontale e di quello verticale. Durante la fase di addestramento, quindi, vengono forniti gli stessi input, e alle due reti vengono sottoposte, come uscite desiderate, i valori di pilotaggio associati ai due motori. La struttura della rete neurale che deriva da questa scelta è riportata in figura 4.6.

Le reti neurali hanno dunque due neuroni di ingresso e uno di uscita. Per lo strato nascosto si è scelto un numero di neuroni tra 12 e 16 per ciascuna rete, valori che sono stati determinati sperimentalmente osservando il comportamento della rete durante le fasi di addestramento e utilizzo.

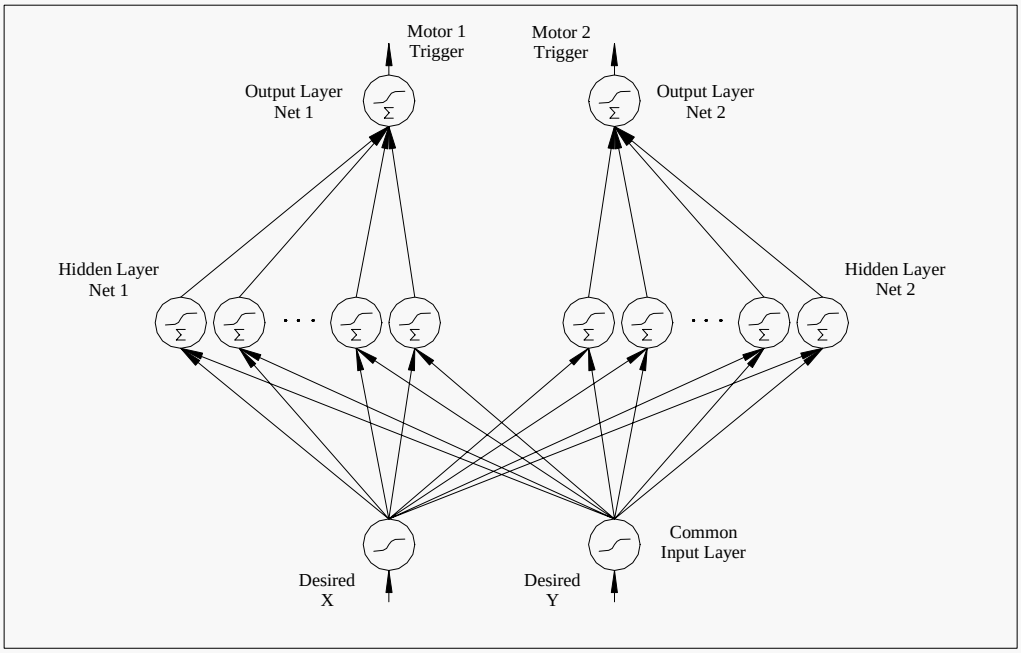


Figura 4.6. Rete neurale utilizzata in DEA.

L'addestramento viene effettuato utilizzando un training set composto da circa 100 esempi, anche se il numero esatto dipende, tra le altre cose, dalla dimensione dell'area di scansione, e quindi dalle impostazioni del modulo *FP* che individua i punti sul perimetro dell'area stessa.

Si è notato che l'addestramento è in genere molto veloce, quindi il numero massimo di epoche è stato posto pari a 50000.

Infine, il limite di precisione è pari a 0.001, il che garantisce, considerati gli altri parametri relativi al learning, una più che sufficiente approssimazione delle funzioni.

4.7 Test su reti backpropagation

In parallelo con lo sviluppo del lavoro di tesi, sono stati condotti una serie di test, allo scopo di estrapolare delle conclusioni di tipo qualitativo sul comportamento di reti backpropagation, utilizzate per approssimare funzioni reali di variabile reale al variare di alcuni parametri caratteristici.

Per effettuare i test è stato sviluppato un apposito ambiente di simulazione con interfaccia grafica, denominato NNSE, ovvero Neural Networks Simulation Environment. Questo ambiente, la cui schermata principale è riportata in figura 4.7, è stato concepito per garantire le seguenti feature:

- training della rete con visualizzazione dell'andamento dell'errore sia sul training set che sul validation set;
- visualizzazione della funzione approssimata e di quella approssimante durante il training;
- possibilità di impostare tutti i parametri da file per diverse simulazioni sulla stessa funzione obiettivo;
- possibilità di ripetere più cicli di learning durante la stessa sessione di lavoro;
- estrema semplicità nel cambiare la funzione obiettivo.

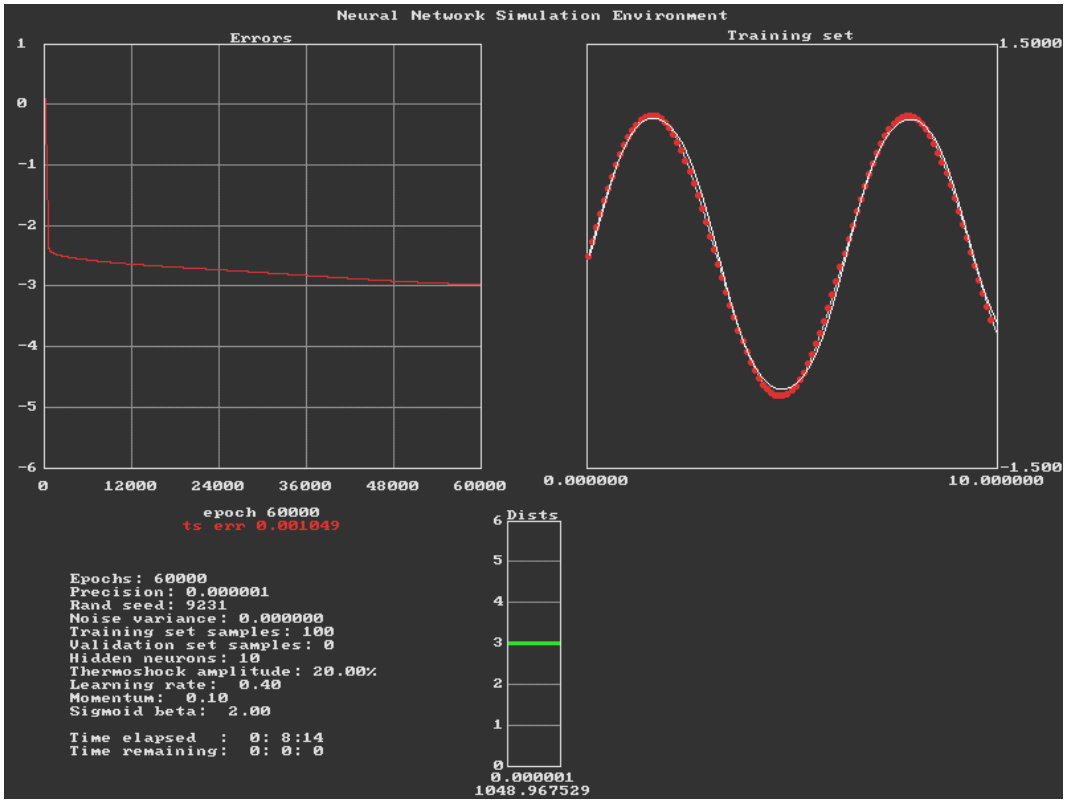


Figura 4.7. Schermata del programma di simulazione neurale.

In figura è riportato l'esempio dell'approssimazione della funzione *seno(x)* sull'intervallo [0, 10], utilizzando una rete con 10 neuroni nello strato nascosto e un coefficiente β pari a 2. Il training è stato eseguito su 60000 epoche, un training set composto da 100 esempi, il valore del momento pari a 0.1 e un learning rate pari a 0.4. Tutti i valori sono riportati a video, così come l'andamento dell'errore totale commesso sul training set ad ogni epoca (riquadro in alto a sinistra), la funzione approssimante e la funzione da approssimare con i relativi punti appartenenti al training set (riquadro in alto a destra). L'indicatore in basso al centro consente di monitorare la distanza dell'errore attuale dal limite di precisione impostato.

L'ambiente sviluppato consente di effettuare test sull'approssimazione di *funzioni reali di variabile reale*, impostando la funzione desiderata, il relativo dominio e il range di output corrispondente.

4.7.1 Parametri testati durante le prove

Per determinare il tipo di test da eseguire sulla capacità di approssimazione di funzioni da parte della rete, sono stati considerati principalmente i seguenti parametri:

- il coefficiente β che compare nell'esponenziale della sigmoide;
- il numero di neuroni nello strato nascosto;
- il tipo di funzione da approssimare;
- il numero di training samples presenti nel set di esempi;
- il livello del rumore sull'uscita desiderata.

Per poter testare tutte le condizioni possibili si sarebbero dovute considerare le combinazioni di tutti i valori attribuiti ai singoli parametri. Purtroppo una operazione di questo genere avrebbe richiesto delle risorse di calcolo alquanto dispendiosa, in termini di tempo di calcolo oppure di macchine utilizzate in parallelo. Per ovviare al problema si è scelto di effettuare un opportuno sottoinsieme di test, ritenuto significativo per poter estendere le conclusioni anche alle situazioni non direttamente testate.

4.8 Risultato dei test

La maggior parte dei test ha confermato le ipotesi qualitative che è possibile formulare conoscendo l'influenza dei parametri testati sull'apprendimento della rete. Per esempio, aumentando il numero di neuroni nello strato nascosto si ottiene un apprendimento più veloce e preciso, almeno fino ad un certo limite, il quale

dipende fortemente dal tipo di funzione da approssimare e dal numero di training samples forniti. Caso per caso è necessario individuare il numero corretto di neuroni da porre nello strato nascosto.

4.8.1 Test sul β delle sigmoidi

I risultati più interessanti sono stati ottenuti testando l'influenza del coefficiente β della sigmoide sull'apprendimento della rete. Variare il parametro in questione significa avere delle sigmoidi che cambiano forma, passando da un andamento quasi a scalino (per β elevati) ad un andamento quasi rettilineo (per β molto bassi). In figura 4.8 è possibile osservare diverse sigmoidi ottenute variando β.

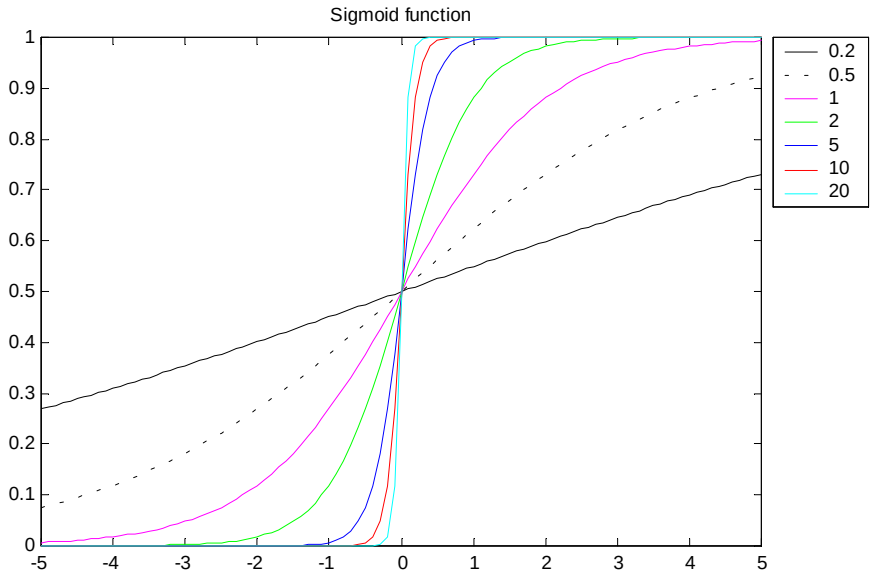


Figura 4.8. Sigmoidi al variare del parametro β.

Quello che si è notato è che aumentando β, a parità di tutti gli altri parametri, la velocità di convergenza verso il minimo dell'errore totale aumenta. Quindi è possibile velocizzare l'apprendimento utilizzando dei valori di β

maggiori di 1. Si è notato che dei buoni valori sono generalmente contenuti nell'intervallo [1.5, 4].

C'è però un problema nell'utilizzare questo espediente. Aumentando troppo β diventa sempre più importante la determinazione dei pesi di partenza dei neuroni, come descritto nel paragrafo seguente.

4.8.2 Determinazione dei pesi iniziali

La seconda importante osservazione derivata dai test sulle reti è data dalla necessità di impostare i pesi e le soglie iniziali in modo che siano sufficientemente piccoli. Dei pesi troppo grandi possono dare luogo all'annullamento dell'errore sul training set, in quanto la formula che calcola l'errore sul singolo training sample contiene un temine del tipo

$$y_j \cdot (1 - y_j)$$

che moltiplica tutto il resto, dove y_i è l'uscita della sigmoide calcolata sulla somma degli ingressi del neurone.

Si capisce che quanto più i pesi sono grandi, quanto più è facile che il punto nel quale viene valutata la sigmoide si trovi nel tratto asintotico. In questo caso il valore di y_i vale 0 oppure 1, e l'errore sul training sample si annulla. Questo problema si amplifica quanto più la forma della sigmoide tende a quella di uno scalino, in quanto il range disponibile per una buona valutazione della sigmoide si riduce.

Durante i test è capitato che fossero nulli gli errori su tutti i training sample (per valori di β troppo alti), e che quindi l'errore globale precipitasse di colpo a 0, ingannando la procedura di addestramento e facendole "credere" di aver raggiunto il minimo assoluto dell'errore.

Capitolo 5

Architettura software

5.1 Idea di base

La stesura del codice di DEA è stata effettuata con l'intento di perseguire due obiettivi principali:

1) rendere il progetto più modulare possibile, in modo da rendere ciascun singolo blocco indipendente dagli altri;
2) generare del codice riutilizzabile in altre applicazioni con il minimo sforzo di modifica del sorgente.

Si può notare che realizzare quanto proposto nel primo punto porta spesso a soddisfare anche i requisiti richiesti dal secondo.

Nel caso di DEA, tali aspetti sono intesi su due livelli di astrazione differenti. La modularizzazione richiesta nel primo punto viene pensata a livello di unità esecutive del progetto nel suo complesso, nel senso che il programma non è monolitico, ma è formato da più moduli che possono essere eseguiti indipendentemente dagli altri. Tali moduli si occupano di operazioni logicamente indipendenti dalle altre, come la generazione del training set della rete neurale oppure l'addestramento della rete stessa, oppure, ancora, la fase di tracking vera e propria.

Ciascun modulo produce dei dati che verranno utilizzati da uno o più moduli eseguiti successivamente e, similmente, ciascun modulo utilizza dati ricavati da moduli eseguiti in precedenza, quindi esiste un preciso ordine col quale devono essere serializzate le esecuzioni: ad esempio, la rete neurale non può essere addestrata se prima non viene generato il training set, oppure non si può far puntare DEA in un punto specifico del piano di movimento se prima la rete stessa non viene addestrata a dovere.

Questo approccio presenta il vantaggio che, una volta eseguito un modulo, si possono dare per acquisiti i dati da esso prodotti, e lavorare soltanto sui moduli successivi. Una nuova esecuzione di un modulo è richiesta, in genere, soltanto quando si effettua una modifica al sistema, come, per esempio, se si sposta qualche elemento (telecamera, base mobile, laser, etc.) o si sostituisce qualche componente.

Un secondo e non meno importante vantaggio di questo approccio, anche se poco significativo dal punto di vista dell'utente finale, è dovuto al fatto che isolare i diversi moduli permette di sviluppare ed effettuare il debug di una sola parte del progetto, in modo completamente indipendente dal resto del sistema complessivo, con guadagni in termini di chiarezza e semplicità di stesura.

I diversi moduli esecutivi prelevano i dati necessari al proprio funzionamento da files e memorizzano i valori generati in altri files, i *data file*. Il meccanismo di isolamento tra i moduli è realizzato col sistema che ciascun modulo "vede" i dati necessari alla propria esecuzione all'interno dei data file, senza curarsi del programma che li ha prodotti.

Il secondo punto è inteso essere soddisfatto a livello di librerie di funzioni. A tale scopo si sono implementate le librerie in maniera che possano essere semplici da riutilizzare ed estendere, minimizzando gli interventi sul codice sorgente. Le librerie sviluppate in quest'ottica sono principalmente:

- la libreria che contiene le funzioni di interfacciamento con la scheda National Intruments PCI-6025E;
- la libreria che implementa le reti neurali usate per pilotare i motori;

- la libreria che implementa il predittore di Kalman.

5.2 Sequenza esecutiva

Il funzionamento di DEA prevede due distinte fasi: una prima fase di taratura manuale, una seconda fase di calibrazione automatica e una terza fase operativa.

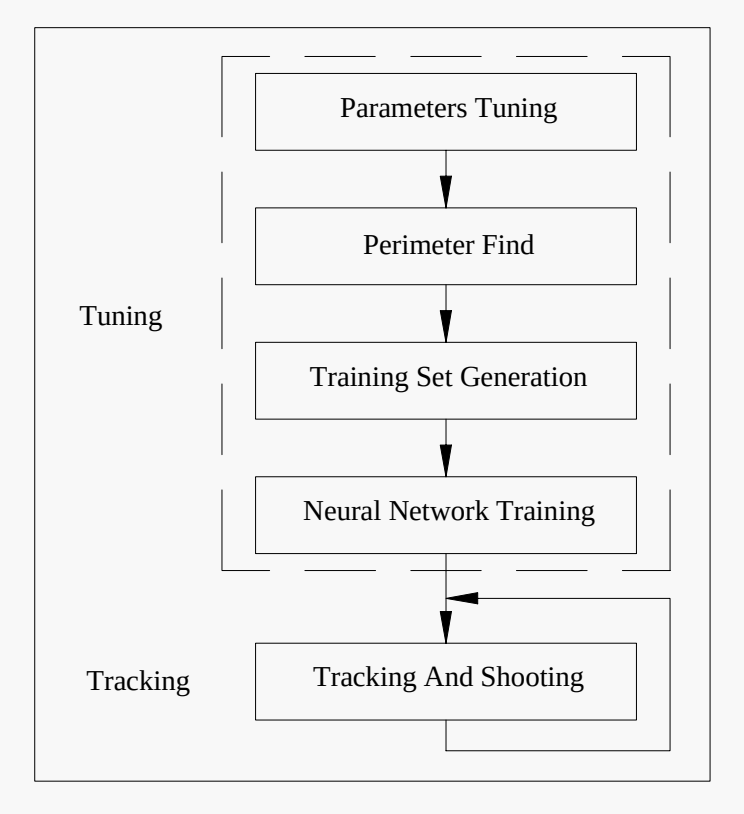


Figura 5.1. Sequenza di configurazione di DEA

La sequenza di fasi di configurazione e di effettivo funzionamento è schematizzata in figura 5.1, ed è descritta di seguito. Nella descrizione vengono utilizzati i nomi dei moduli esecutivi che verranno illustrati successivamente in questo stesso capitolo.

Si tenga ben presente che la taratura va effettuata soltanto la prima volta che il sistema si trova in una "nuova situazione", ovvero soltanto nel caso in cui le condizioni ambientali subiscono dei mutamenti, oppure se il dispositivo elettromeccanico su cui si basa il movimento di DEA viene modificato. Una volta effettuata la taratura è possibile eseguire più volte la fase operativa, mantenendo i settaggi invariati. In più, se variano determinati parametri, è possibile ricalibrare questi soli, mantenendo gli altri invariati. Ad esempio, se nell'ambiente in cui si muove il target cambia improvvisamente il livello di luminosità, come se in una stanza si accendesse o si spegnesse una luce, vanno generalmente reimpostate le soglie su cui si basa l'elaborazione dell'immagine, al fine di trovare le coordinate del baricentro del target. Ora, in una situazione del genere, è possibile riconfigurare le sole soglie, senza preoccuparsi di agire sul resto dei parametri.

La fase preliminare di taratura deve cominciare con l'impostazione dei limiti di pilotaggio dei motori, che sono indispensabili e fondamentali per poter pilotare i servomotori in tutta sicurezza, in quanto, come verrà chiarito più avanti in questo stesso capitolo, il driver che comanda i motori è progettato in modo da non fornire alcun comando che li possa danneggiare, anche nel caso riceva valori errati dai processi che necessitano di controllare il movimento di DEA.

Durante la fase di taratura manuale, inoltre, vengono eseguiti altri programmi che servono ad impostare correttamente il sistema in relazione alla dislocazione fisica dei vari componenti del sistema e all'ambiente nel quale si muoverà il target. In particolare, in questa fase, vengono tarate le soglie di luminosità necessarie a rintracciare sull'immagine sia il punto laser che il bersaglio. Inoltre si delimita il campo di movimento del target nell'immagine, in modo da ottimizzarne la ricerca del baricentro; ovviamente, se non si hanno a disposizione informazioni circa la traiettoria dell'oggetto, è comunque possibile definire l'area di scansione pari a tutta l'immagine: il sistema funzionerà benissimo anche in questo caso. E' inoltre possibile impostare la posizione di riposo dei motori: in genere è comodo fare in modo che, a riposo, il puntatore laser miri circa al centro dell'immagine, anche se un qualsiasi altro punto all'interno del

campo di visuale della telecamera andrebbe bene. In questa stessa fase è necessario tarare il filtro di Kalman, assegnando, sempre manualmente, un valore opportuno al ρ che ne caratterizza il comportamento. Inoltre va deciso il numero di integratori da utilizzare nel modello. In effetti DEA non prevede la possibilità di adeguare automaticamente online i parametri del filtro, cosicchè è necessario fornire i valori adeguati manualmente. Come si vedrà, è comunque possibile adeguare i parametri anche durante la fase operativa, ma il tutto avviene sempre tramite i comandi dell'operatore.

La seconda fase prevede la calibrazione automatica della rete neurale, per fare in modo che sia in grado di pilotare i motori senza l'ausilio del puntatore laser. In questa fase avviene la generazione del training set necessario all'addestramento, seguito dall'addestramento vero e proprio.

Ultimata la fase di tuning del sistema va avviato il modulo che utilizza i sottosistemi già completamente calibrati. Il programma effettua il tracking e la predizione della traiettoria del target, in modo da portare il sistema di fuoco nella posizione adatta per essere azionato al momento opportuno.

5.3 I data files

Per realizzare quanto descritto in precedenza è necessario realizzare un sistema di "comunicazione" tra i vari moduli dell'applicazione che, essendo dei programmi differenti che vengono eseguiti in successione oppure singolarmente a seconda delle necessità, non condividono il medesimo spazio di memoria. E' quindi indispensabile leggere i dati di ingresso e scrivere i risultati finali in un file su disco.

I data files sono dei file di testo che contengono i valori di inizializzazione utilizzati dai vari moduli esecutivi. Vengono inoltre scritti dai moduli per immagazzinare il risultato delle elaborazioni e renderle disponibili ai moduli che verranno eseguiti successivamente.

I data files sono organizzati internamente in modo che su ciascuna linea siano contenuti uno o più parametri, i quali possono essere sia numeri interi che floating point o, ancora, stringhe di testo. Tali file, in quanto semplici file di testo, possono venire modificati anche manualmente, se ve ne fosse la necessità, ma in tal caso è importante rispettare l'ordine con cui i parametri vengono elencati, in quanto la procedura di lettura carica i valori secondo, appunto, un ordine predeterminato. Questo fatto non è valido in generale, in quanto esistono dei data file per i quali non è necessario rispettare una sequenza rigida per l'elenco dei valori fatta eccezione, in genere, per le prime righe, che costituiscono una sorta di header del file. Ovviamente, quando un data file viene scritto da un modulo esecutivo che vi immagazzina i dati di output, questo rispetta l'ordine di scrittura dei vari parametri. L'elenco dei data file è il seguente:

- **2P.X** contiene i punti sui bordi dell'immagine necessari alla generazione automatica del training set per l'addestramento della rete neurale;
- **ISL.X** contiene i valori in pixel che quantificano il bordo per l'area di scansione dell'immagine;
- **KAL?.X** contiene i valori dei ρ e dei rispettivi guadagni che caratterizzano il comportamento del predittore di Kalman;
- **KF.X** contiene i parametri per l'inizializzazione automatica della batteria di filtri di Kalman nella libreria *kalman.h*;
- **LD.X** contiene i dati relativi al puntatore laser sull'immagine;
- **MD.X** contiene le costanti relative al pilotaggio dei servomotori, come i limiti degli intervalli di pilotaggio e la posizione di riposo, ed alcune costanti che determinano la curva di risposta;
- **NEUR.X** contiene tutti i parametri che impostano il programma di apprendimento della rete neurale;
- **OD.X** contiene i dati relativi al target, così come viene ripreso dalla telecamera;

- **SAD.X** contiene tutti i parametri di funzionamento del modulo che esegue il tracking dell'oggetto, ovvero SAD;
- **TSH.X** e **TSV.X** contengono il training generato automaticamente dall'apposito programma e che serve all'addestramento della rete neurale;
- **WH.X** e **WV.X** memorizzano i pesi e le soglie della rete neurale, in modo che, dopo l'addestramento, i moduli che ne necessitano l'utilizzo, possano leggere direttamente i valori della rete istruita.

Di seguito vengono descritti i diversi data file e verrà illustrata la struttura di ciascuno di essi.

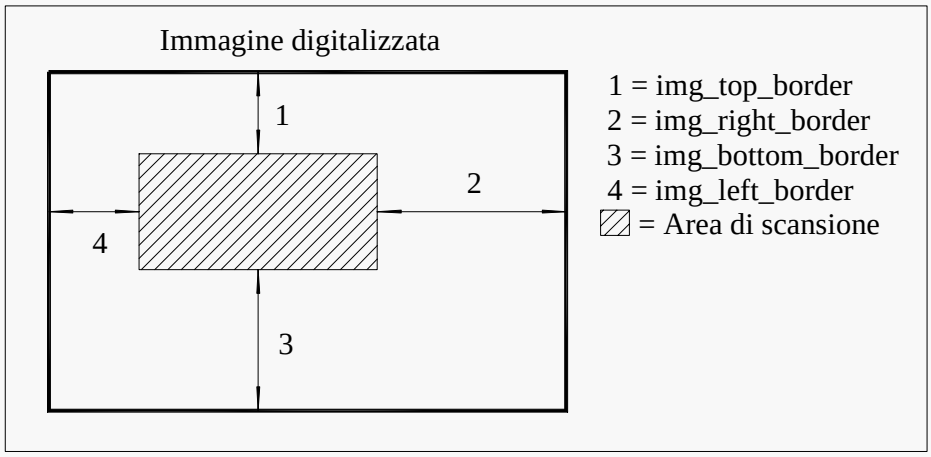
5.3.1 Punti perimetrali dell'area di scansione

Il data file **2P.X** è un primo esempio di file nel quale non è importante l'ordine dei dati immagazzinati. Il file contiene le coppie di valori di pilotaggio per i servomotori che puntano il laser ad opportune coordinate sull'immagine. Queste coordinate, come illustrato più in dettaglio al capitolo 5.4.2, servono a costruire una griglia per la generazione automatica del training set. La struttura di **2P.X** è la seguente:

```
nx
ny
... nx_top_points
... ny_right_points
... nx_bottom_points
... ny_right_points
```

dove

nx e *ny* sono, rispettivamente, il numero di punti individuati sui lati orizzontali del perimetro dell'immagine (superiore e inferiore) e il numero di punti individuati sui lati verticali (sinistro e destro):



5.3.2 Area di scansione

Lo Image Scan Limits data file, **ISL.X**, contiene le dimensioni in pixel dei bordi con cui viene limitata l'area di scansione dell'immagine quando viene effettuata la ricerca del puntatore laser oppure del target. La struttura di questo data file è semplicissima:

```
img_top_border
img_left_border
img_bottom_border
img_right_border
```

dove ciascun valore rappresenta il numero di pixel di rientro del rispettivo bordo nell'immagine, che serve appunto a definire lo spessore del bordo. Rispettivamente, si tratta del bordo superiore, sinistro, inferiore e destro dell'immagine, come riportato in figura 5.2.

Figura 5.2. Area di scansione.

5.3.3 Guadagni per il filtro di Kalman

I data file **KAL?.X**, dove il simbolo "?" può essere sostituito da un numero compreso tra 3 e 8, contengono ciascuno i valori dei guadagni precalcolati che corrispondono ai diversi ρ utilizzabili per il filtro di Kalman. L'indice che differenzia i file sta ad indicare il numero di integratori con cui è modellizzato il sistema e, dunque, anche il numero di guadagni per ciascun valore di ρ. La struttura di ogni singolo file è la stessa, ovvero:

```
n_int
n_rho
rho_1 gains_1
...
rho_n_rho gains_n_rho
```

dove

n_int rappresenta il numero di integratori con cui viene modellizzato il sistema per il quale si possono utilizzare i guadagni memorizzati nel file;

n_rho indica il numero di diversi valori di ρ immagazzinati nel file;

rho_1 ... *rho_n_rho* sono i valori veri e propri di ρ, ad ognuno dei quali corrispondono i rispettivi guadagni *gains_n*, che sono, per ogni ρ, in numero pari a *n_int*.

5.3.4 Inizializzazione del filtro di Kalman

Il Kalman Filter data file, ovvero **KF.X**, che viene utilizzato dalla procedura di inizializzazione della libreria *kalman.h* allo scopo di impostare in modo automatico i diversi parametri che ne determinano il funzionamento. Questo data file definisce, tra le altre cose, il ρ con cui effettuare le predizioni e il numero di integratori dei modelli. La sua struttura è la seguente:

```
n
dim_1
...
dim_n
filename_1
rho_1 seekmode_1
...
filename_n
rho_n seekmode_n
predstep
```

dove

n = numero di filtri in parallelo che si intende utilizzare;

dim_i = dimensione del modello usato nell'i-esimo filtro;

filename_i = nome del file da cui leggere i guadagni per il filtro i-esimo;

rho_i = valore iniziale del ρ per il filtro i-esimo;

seekmode_i = tecnica di ricerca del ρ nella look-up table dell'i-esimo filtro della batteria;

predstep = numero di passi in avanti per le previsioni bufferizzate.

5.3.5 Caratteristiche del puntatore laser

Il Laser Data file, cioè il file **LD.X**, contiene informazioni circa il puntatore laser così come viene visto dalla telecamera. Le voci presenti in questo data file sono le seguenti:

```
laser_threshold
laser_spot_width
laser_spot_height
```

dove

laser_threshold è la soglia utilizzata dalla procedura di scansione per isolare i punti che corrispondono al puntatore laser sull'immagine;

laser_spot_width e *laser_spot_height* sono, rispettivamente, la larghezza e l'altezza in pixel del puntatore laser. Anche queste informazioni vengono utilizzate dalla procedura di scansione dell'immagine per aumentarne la velocità.

5.3.6 Costanti di pilotaggio dei motori

Il Motor Data file, **MD.X**, contiene le costanti necessarie al driver per pilotare correttamente i servomotori. Queste costanti, tra le altre cose, servono a modellare la curva di funzionamento dei motori secondo le necessità. La struttura di **md.x** è la seguente:

```
hmin
hmax
vmin
vmax
hhome
vhome
hdelta
```

```
vdelta
```

dove

hmin e *hmax* sono il minimo e il massimo valore di pilotaggio possibile per il motore orizzontale;

vmin e *vmax* sono invece il minimo e il massimo valore nel range di pilotaggio per il motore verticale;

hhome e *vhome* sono i valori di riposo per i due motori;

hdelta e *vdelta* servono a definire la curva di comportamento caratteristica dei due motori, come descritto al paragrafo 3.3.2.

5.3.7 Parametri per il training dell'apparato neurale

Il data file **NEUR.X** contiene i parametri che determinano il funzionamento del programma atto ad istruire la rete neurale. Tali parametri sono i seguenti:

```
tset_in_fname
weight_out_fname
n_epochs
precision
random_seed
noise_level
tset_samples_number
vset_samples_number
hidden_number
```

il cui significafo è

tset_in_fname è il nome del file dal quale si desidera caricare il training set necessario all'addestramento della rete. La struttura del suddetto file deve avere le caratteristiche descritte, in questo stesso paragrafo, per i data file **TSH.X** e **TSV.X**;

weight_out_fname è il nome del file nel quale vengono memorizzati i pesi e le soglie dei singoli neuroni una volta che è stato completato l'addestramento;

n_epochs indica il numero massimo di epoche per cui si desidera iterare la fase di apprendimento;

precision specifica il valore di precisione che si desidera raggiungere per l'apprendimento;

random_seed è il seme da utilizzare per inizializzare il generatore di numeri pseudo-casuali del C. Mantenendo fissi gli altri parametri, cambiando questo valore è possibile ottenere delle curve di apprendimento leggermente diverse e, se necessario, ripetere esattamente una stessa sequenza di learning;

noise_level specifica il livello di rumore che si desidera aggiungere ai valori desiderati in uscita alla rete durante il training;

tset_samples_number e *vset_samples_number* sono il numero di esempi che compongono rispettivamente il training set e il validation set. I training samples che vengono letti dal file *tset_in_fname* vengono suddivisi in modo da rispettare questi valori, e si ha un errore nel caso in cui non ci siano esempi sufficienti.

hidden_number permette di decidere il numero di neuroni da porre nell'hidden layer della rete neurale.

5.3.8 Caratterisiche del target

L'Object Data file, ovvero il file **OD.X**, contiene dati relativi al target ripreso dalla telecamera, quali:

```
obj_threshold
obj_width
obj_height
```

dove

obj_threshold individua la soglia usata dalla procedura di scansione dell'immagine per isolare i punti che appartengono al target;

obj_width e *obj_height* sono le dimensioni dell'oggetto da colpire, in pixel, che vengono determinate automaticamente a partire dalle immagini fornite dalla telecamera.

5.3.9 Inizializzazione del modulo di tracking

E’ possibile impostare tutti i parametri principali che determinano il funzionamento del modulo di tracking (SAD) mediante SAD.X, il data file preposto a tale scopo. I parametri che possono essere modificati sono, nell’ordine:

```
plane_dist
fly_speed
valve_delay
hook_dist
min_ok_hooks
```

dove

plane_dist è la distanza media, in centimetri, del piano di moto dell’oggetto da punto di partenza del proiettile;

fly_speed è la velocità del proiettile stimata con i test descritti al capitolo 3.10, espressa in cm/ms;

valve_delay indica il ritardo di attivazione della valvola in millisecondi, anch’esso determinato grazie ai test discussi al capitolo 3.10;

hook_dist indica il raggio del cerchio entro il quale il test di aggancio della traiettoria da parte del filtro di Kalman considera la traiettoria stessa agganciata. Il valore è espresso in pixel;

min_ok_hooks è il numero di test consecutivi sulla posizione corrente e su quella predetta che devono dare esito positivo per abilitare la fase di sparo vera e propria.

5.3.10 Training set

I due Training Set data file, cioè **TSH.X** e **TSV.X**, che hanno la medesima struttura, contengono i training samples necessari all'addestramento delle reti neurali di DEA. La struttura la seguente:

```
n_samples
net_n_in net_n_out
min_in_range
max_in_range
min_out_range
max_out_range
in_1_1 in_1_2 out_1
...
in_n_1 in_n_2 out_n
```

dove

n_samples è il numero di training samples contenuti nel file;

net_n_in e *net_n_out* sono il numero di ingressi e di uscite della rete neurale per la quale utilizzare il training set contenuto nel file. Per le due reti neurali usate in DEA questi valori sono fissi, e valgono rispettivamente 2 e 1;

min_in_range e *max_in_range* sono i limiti del range di valori in ingresso alla rete, usati per scalare opportunamente i veri valori di ingresso;

min_out_range e *max_out_range* sono i limiti del range di valori in uscita dalla rete, usati anch'essi per scalare opportunamente i valori veri di uscita;

in_1_1 in_1_2 out_1 e tutti i valori seguenti fino alla terna i*n_n_1 in_n_2 out_n* sono la lista dei training samples veri e propri. Sono in gruppi di tre

per il semplice fatto che la rete neurale che muove i motori di DEA ha 2 ingressi e una sola uscita, quindi i primi due valori sono gli ingressi, mentre il terzo valore è pari all'uscita desiderata corrispondente.

5.3.11 Pesi e soglie della rete neurale

Gli Weight data file **WH.X** e **WV.X** hanno tra loro lo stesso tipo di struttura, e contengono i pesi e le soglie caratteristici di una rete neurale, in modo che sia richiesto un solo processo di addestramento e che, in seguito, sia possibile caricare una rete già addestrata risparmiando notevolmente tempo e risorse. La struttura del data file è la seguente:

```
ninp
nhid
nout
min_sig
max_sig
hidden_weight
output_weight
output_threshold
hidden_threshold
```

dove

ninp, *nhid* e *nout* specificano la struttura della rete neurale che è stata salvata nel file. Sono rispettivamente il numero di neuroni di ingresso, nello strato nascosto e di uscita;

min_sig e *max_sig* sono i valori che individuano l'intervallo di scaling per mappare gli ingressi e le uscite reali nel range adatto alla rete neurale;

hidden_weight e *output_weight* sono la lista dei pesi dello strato nascosto e di output della rete;

output_threshold e *hidden_threshold* sono, invece, la lista delle soglie che caratterizzano i neuroni dello strato nascosto e di output.

Non tutti i moduli utilizzano tutti i data file per il loro funzionamento, ma caricano, grazie a funzioni apposite, i soli dati necessari.

Nel prossimo paragrafo vengono descritti in dettaglio i moduli esecutivi, mentre il successivo è dedicato ad illustrare le relazioni tra ciascun modulo ed i data file che utilizza, sia in input che in output.

5.4 Moduli esecutivi

I moduli esecutivi sono i programmi che compongono DEA. Ciascuno di essi effettua una determinata operazione, partendo da opportuni dati d'ingresso e producendo generalmente uno o più data file di uscita, che verranno poi utilizzati da successivi moduli.

La seguente trattazione è pensata come una sorta di manuale utente per i singoli moduli, arricchito con dettagli implementativi che ne chiariscono il funzionamento.

I moduli necessari al funzionamento di DEA, e che vengono illustrati in questo capitolo, sono i seguenti:

- *camera*, che è studiato per permettere di impostare manualmente diversi parametri in fase di calibrazione del sistema;
- *fp* (Find Perimeter), il modulo che individua punti sul perimetro della finestra di scansione, che vengono usati per generare il training set per la rete neurale.
- *gts* (Generate Training Set), il modulo che produce i training sample da utilizzare per l'addestramento della rete neurale.
- *nnt* (Neural Net Training), il programma che si occupa di addestrare le due reti neurali, una per volta, usate in DEA.

- *sad* (Seek And Destroy), il modulo che utilizza tutti i dati prodotti in fase di calibrazione per fare il tracking del bersaglio, predirne la traiettoria e colpirlo.

Accanto ai moduli che vanno utilizzati per la calibrazione e per l'effettivo utilizzo di DEA, ne sono stati sviluppati altri due, dei quali uno viene usato per la misurazione della velocità della pallina e del ritardo della valvola, mentre l'altro è usato per produrre alcune simulazioni con il predittore di Kalman in ambiente real-time, e poter toccare con mano il comportamento del predittore al variare di diversi parametri. I moduli sono:

- *simkal* e *simkal1* sono invece le simulazioni real-time sul filtro di Kalman.
- *speed* serve appunto ad effettuare i test sulla velocità della pallina e ricavare informazioni anche sui ritardi di attivazione dell'elettrovalvola.

Nei paragrafi a seguire verranno descritti nel dettaglio i diversi moduli esecutivi.

5.4.1 CAMERA

Il modulo CAMERA è dedicato alla taratura manuale di molti parametri fondamentali per il corretto funzionamento del sistema. I valori vengono modificati monitorandone direttamente sul video, in tempo reale, l'influenza sul sistema. Le operazioni che possono essere effettuate dall'operatore sono le seguenti:

1) muovere i motori, facendo ruotare il dispositivo di puntamento;
2) accendere e spegnere il laser;
3) azionare l'elettrovalvola;
4) dimensionare l'area di scansione;
5) modificare il livello delle soglie che identificano sia il puntatore laser che il target;
6) portare il dispositivo di puntamento in posizione di riposo;

7) impostare la posizione di riposo;
8) portare il dispositivo di puntamento in posizione di carica del proiettile;
9) effettuare l'hardcopy della schermata grafica.

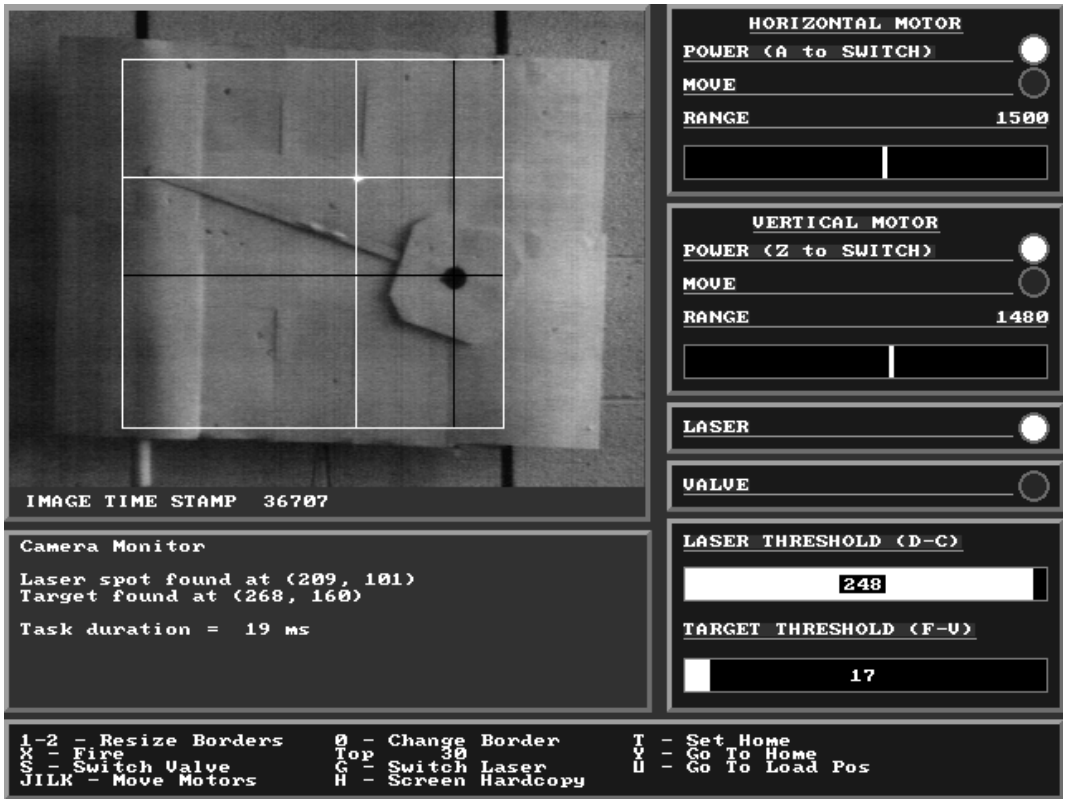


Figura 5.3. Schermata di CAMERA.

Mentre si compie la rotazione manuale del dispositivo di puntamento è possibile osservare il valore di pilotaggio dei motori. E' così possibile determinare opportunamente i limiti di tali valori, in modo da evitare danni ai servo durante il funzionamento automatico. Inoltre, mentre si spostano i puntatori, è possibile settare la posizione di riposo. Questa è una operazione da non sottovalutare, in quanto è indispensabile per il corretto funzionamento dei moduli esecutivi di calibrazione: questi ultimi, infatti, assumono che il puntatore laser sia contenuto nell'area di scansione quando i motori sono in posizione di riposo. Se così non è,

la procedura di calibrazione non può iniziare, come si avrà modo di evidenziare esaminandone il funzionamento.

Mentre il puntatore laser può essere semplicemente acceso o spento, l'elettrovalvola può invece essere azionata in due diversi modi. E' possibile aprire la valvola e mantenerla aperta fino al successivo comando di chiusura. In alternativa, è possibile effettuare uno "shoot", uno sparo, cioè aprire e chiudere automaticamente la valvola dopo un determinato lasso di tempo. L'accensione del puntatore laser e l'attivazione della valvola sono indispensabili per verificare l'allineamento dei due dispositivi: così si può tarare la direzione del puntatore laser (la cerbottana ha un allineamento fisso) per fare in modo che si posizioni sul punto di impatto del proiettile.

Sempre all'interno di CAMERA è possibile dimensionare opportunamente l'area di scansione dell'immagine, per isolare l'area in cui si muove il bersaglio. Nulla vieta, eventualmente, di considerare tutta l'immagine, impostando a zero il valore dei bordi.

L'operazione forse più importante che può essere espletata eseguendo il presente modulo è la calibrazione delle soglie di luminosità utilizzate dalle procedure di scansione dell'immagine per individuare i punti appartenenti al puntatore laser e all'oggetto da colpire. Come spiegato al paragrafo 2.6, il target deve essere di colore scuro su sfondo bianco, mentre il puntatore laser deve essere più chiaro dello sfondo. Per tarare efficacemente le soglie è necessario lavorare, a parità di illuminazione dell'ambiente, sia sul valore delle soglie che sul livello di illuminazione dell'immagine fornita dalla telecamera, regolando l'apposita ghiera posta sull'obiettivo. Sul video viene visualizzata l'immagine ripresa dalla telecamera e, quando vengono agganciati il bersaglio oppure il puntatore laser, vengono visualizzati dei marcatori che ne individuano il baricentro. La regolazione delle soglie può dirsi completata quando i marcatori evidenziano l'aggancio degli oggetti. Se si desidera, per esempio, regolare la soglia relativa al target, se l'illuminazione dell'ambiente è elevata è necessario scurire l'immagine agendo sull'apertura dell'obiettivo. Poi si alza la soglia (il target è scuro su sfondo

chiaro!) fino al punto in cui non compaiono i marcatori. Appena questi compaiono è utile alzare ancora di poco la soglia, per essere sicuri di trovare il target in qualsiasi posizione dell'immagine. La soglia non va alzata oltremodo, in quanto si rischia di considerare come punti del target altri punti scuri dell'area di scansione: questo fatto viene evidenziato, in genere, da una intermittenza dei marcatori sull'immagine. Questa è dovuta al fatto che il baricentro, calcolato su tutti i punti scuri, viene a trovarsi lontano dal vero baricentro dell'oggetto.

5.4.2 Perimeter finding

Per *perimeter finding* è intesa l'operazione di generare una serie associazioni tra punti sull'immagine e variabili di pilotaggio dei motori che consentono di posizionare il puntatore laser in tali punti. I punti sull'immagine si trovano lungo il perimetro dell'area di scansione, come evidenziato in figura 5.4.

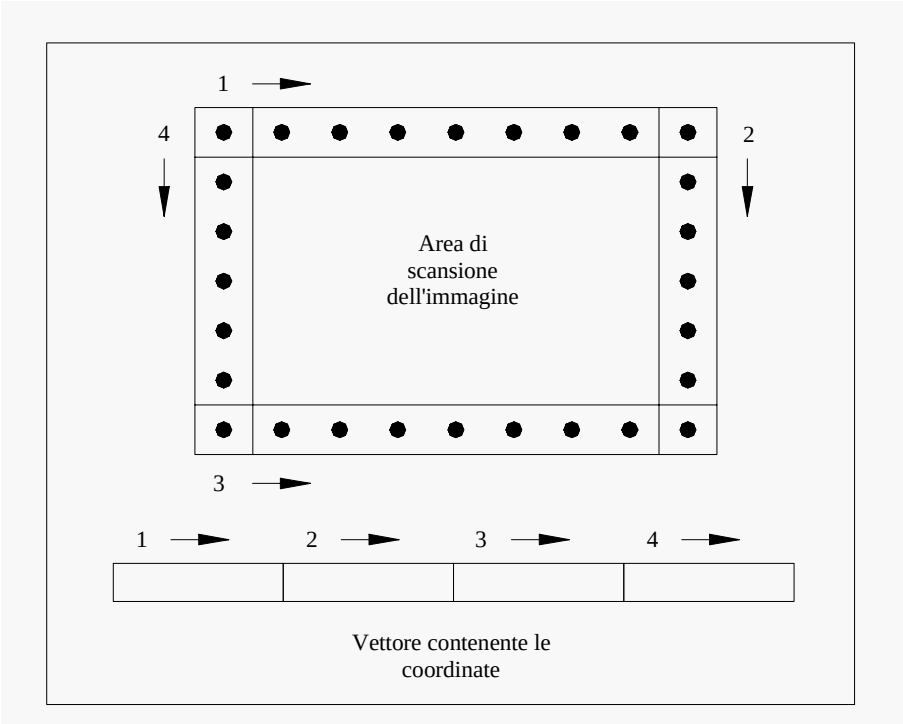


Figura 5.4. Punti sul perimetro dell'area di scansione.

I parametri principali del modulo sono il numero di punti da generare nei lati orizzontali e in quelli verticali.

Il funzionamento del modulo è il seguente:

- vengono inizializzati i sottosistemi costituiti dal motore visivo e dal driver per i servo;
- vengono generate le coordinate dei punti in modo che questi siano equidistanti sui 4 lati dell'area di scansione;
- viene avviato il task che genera gli accoppiamenti punto - variabile di pilotaggio, che continua la sua esecuzione finchè ha generato tutte le associazioni;
- le associazioni vengono salvate nel data file *FP.X*, del quale si è già parlato al paragrafo 5.3.1. La sequenza viene memorizzata scrivendo sequenzialmente il vettore di figura 5.4 nel data file.

Il cuore del modulo è costituito dal task che porta il puntatore laser alle coordinate specificate. La dinamica del task è controllata in anello chiuso, con uno schema di regolazione di tipo PI (proporzionale-integrale). L'incremento del valore di pilotaggio dei motori è proporzionale alla differenza tra la posizione desiderata e quella attuale, opportunamente saturato per limitare l'escursione del movimento dei motori.

Il punto desiderato viene considerato raggiunto quando il puntatore laser entra in un opportuno raggio di distanza dal set-point (in pixel). A questo punto vengono memorizzate le variabili di pilotaggio e vengono associate alla posizione raggiunta, quindi si passa alla posizione successiva. Se, durante il movimento, per qualsiasi ragione, il puntatore laser viene perso (ad esempio se il moto dei motori lo portano all'esterno dell'area di scansione), il task ritorna alla posizione di riposo in modo automatico. Questo fa capire quanto sia importante che la posizione di riposo dei motori faccia puntare il laser all'interno dell'area di scansione.

E' importante calibrare oculatamente il raggio dell'area entro la quale si considera "centrato" l'obiettivo. A causa della scarsa risoluzione nel movimento dei servo, un raggio troppo stretto (1-2 pixel alla distanza di circa 1 metro e mezzo tra il dispositivo di puntamento e il piano immagine) provoca l'impossibilità da parte del sistema di centrare il punto. Un raggio troppo grande introduce un errore elevato nella generazione del training set. Quindi il raggio dovrebbe essere scelto il più piccolo possibile, ma che consentisse di centrare il punto in qualsiasi posizione dello spazio immagine.

Il controllo in anello chiuso ha lo svantaggio di essere lento nel portare il puntatore nella posizione desiderata, ma è l'unica soluzione per risolvere il problema. Il feedback del controllo avviene controllando sull'immagine lo spostamento del puntatore, che è ben diverso dal posizionare il puntatore generando direttamente i giusti valori di pilotaggio.

5.4.3 Training set generation

Il modulo che si occupa della generazione del training set è chiamato *GTS*. Esso rappresenta l'evoluzione di una serie di programmi studiati per generare il training set necessario all'addestramento della rete. Ne é scaturito un algoritmo veloce e in grado di originare un training set che copre correttamente tutta l'area di scansione, consentendo quindi una buona istruzione della rete.

L'algoritmo è basato sulla generazione di una griglia di punti nello spazio delle variabili di pilotaggio dei motori. La griglia viene generata in modo statico effettuando delle intersezioni sulle rette che congiungono i punti sul perimetro (vedi figura 5.5), i quali a loro volta vengono letti dal data file FP.X.

Una volta generata la griglia viene attivato il task che si occupa di posizionare i motori. Per ciascun punto della griglia il task posiziona il puntatore e ne registra la posizione. Viene così costruito il training set che associa le coordinate di un punto sull'immagine con la coppia di valori di pilotaggio dei motori.

L'algoritmo utilizzato rende il programma molto veloce, se confrontato con quello utilizzato per centrare i punti sul perimetro. Nel caso di FP, ad ogni istanza del task che posiziona i motori viene fatto un passo verso la locazione impostata, e sono necessarie diverse iterazioni per portare il puntatore sufficientemente vicino alla destinazione, il cui numero dipende dalle impostazioni dell'anelo di retroazione. Nel caso di GTS ogni tre iterazioni del task viene generato un training sample. Questo poiché il task è strutturato come un automa a stati finiti, il quale cambia stato ad ogni istanza. Gli stati sono 4, e ciascuno è preposto ad una differente azione:

1) vengono generati i nuovi valori di pilotaggio;
2) vengono comandati i motori;
3) si attende che i motori si fermino;

4) si scandisce l’immagine e si memorizzano le coordinate del puntatore laser.

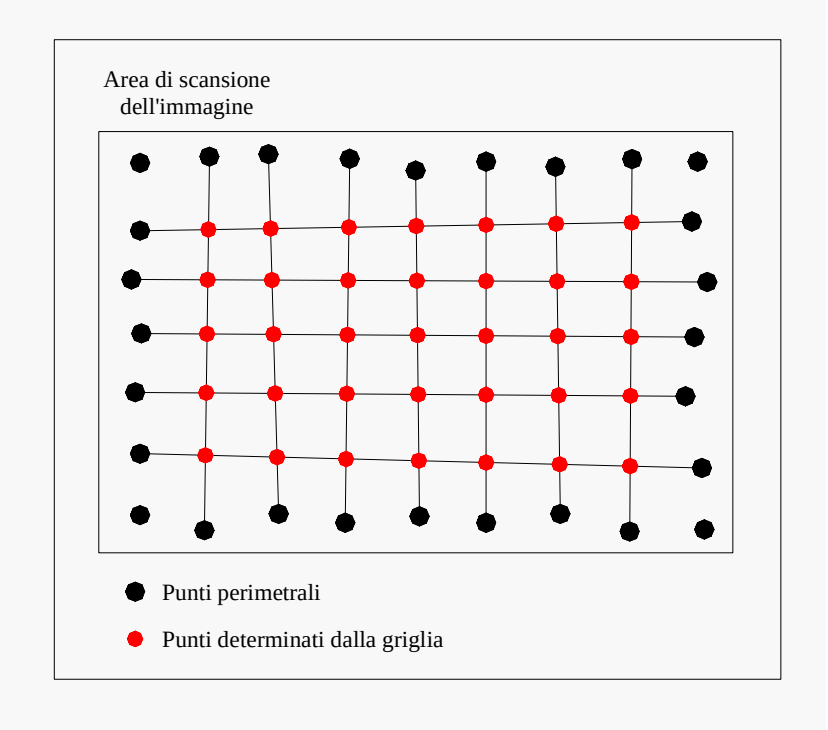


Figura 5.5. Griglia generata dal programma GTS.

Al termine dell’elaborazione vengono prodotti i due data file TSH.X e TSV.X, che contengono, rispettivamente, i training set per le due reti neurali che gestiscono separatamente i due motori.

5.4.4 Neural network training

La rete neurale viene addestrata dal modulo denominato *NNT*. Il procedimento di addestramento di una rete neurale backpropagation e la stessa rete utilizzata in DEA sono esaurientemente descritti al capitolo 4. In questa sede

si fornirà una breve descrizione del software che realizza il training per la rete utilizzata da DEA.

NNT legge i training set per ciascuna rete da TSH.X per la rete dedicata al motore orizzontale e da TSV.X per la rete dedicata al motore verticale, mentre i parametri di apprendimento vengono letti dal data file NEUR.X.

L'interfaccia utente e le funzionalità del programma sono molto semplici: permette di avviare l'apprendimento, di fermarlo in caso di necessità ed eventualmente di riprenderlo dall'istante in cui lo si è interrotto. Al termine dell'esecuzione vengono prodotti i due data file che memorizzano i pesi e le soglie delle reti addestrate, e che vengono letti dai moduli che necessitano di pilotare i motori (in particolare da SAD).

La particolarità di maggior rilievo di NNT è che il codice non è scritto per il sistema operativo HARTIK. Questo perché, innanzitutto, l'apprendimento è un procedimento offline, che non necessita di vincoli temporali stringenti. Inoltre in questo modo è stato possibile utilizzare una libreria appositamente sviluppata per creare videogiochi, chiamata *ALLEGRO*, che mette a disposizione una serie di funzionalità grafiche avanzate e molto sofisticate (come, ad esempio, il double buffering). Questo ha permesso di ottenere una presentazione grafica dei risultati molto fluida e gradevole.

5.4.5 Tracking and shooting

Il modulo esecutivo che sfrutta il lavoro di tuning e calibrazione svolto dai programmi descritti in precedenza è SAD (Seek And Destroy). SAD è, per questo motivo, il modulo decisamente più complesso del progetto. Il suo compito è quello di monitorare lo spostamento del target nell'area di scansione dell'immagine e, al contempo, consentire l'evoluzione dello stato del filtro di Kalman.

Tutto questo viene effettuato da un primo task agganciato al driver del frame-grabber, che effettua il tracking vero e proprio (il task i questione è denominato TRACKER). Una volta che il filtro ha agganciato la traiettoria viene stabilito un istante di sparo (tenuto conto del tempo di volo e del ritardo di attivazione della valvola) e la corrispondente posizione, e vengono comandati i servomotori, attraverso la rete neurale, perché portino la cerbottana nella posizione adatta a colpire il bersaglio.

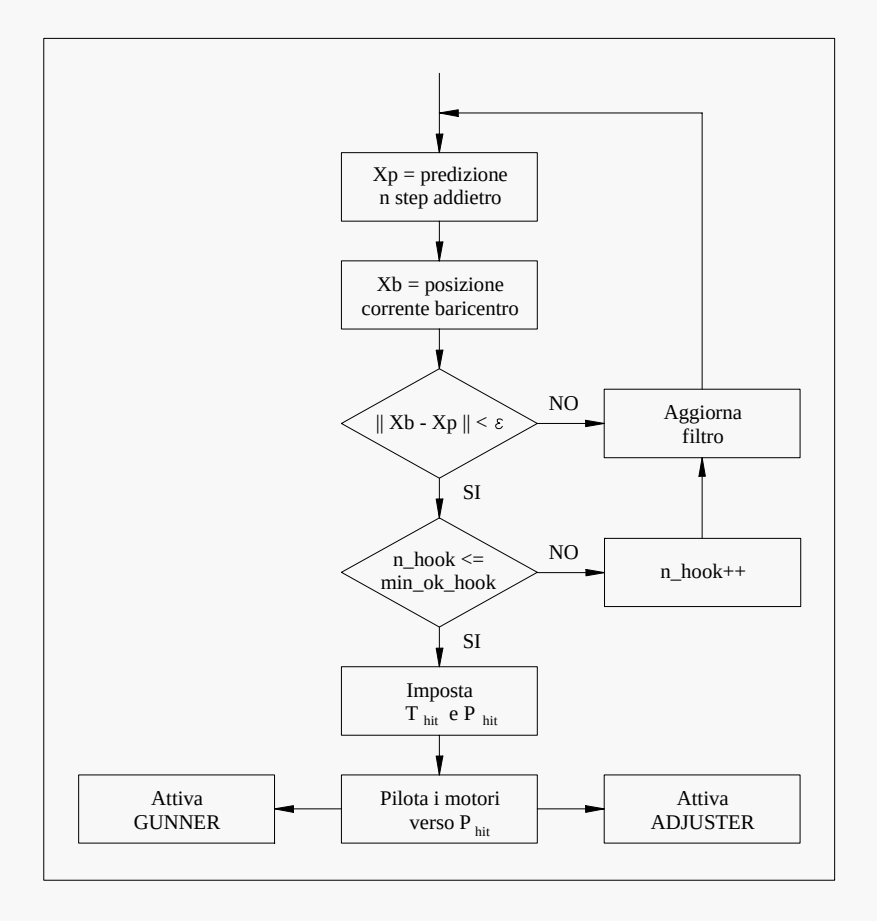


Figura 5.6. Diagramma di funzionamento di TRACKER.

Vengono quindi lanciati due task: il primo, GUNNER, è un task ad altissima frequenza. Esso controlla il timer di sistema, e quando questo eguaglia o supera l'istante previsto per lo sparo, attiva l'apertura della valvola.

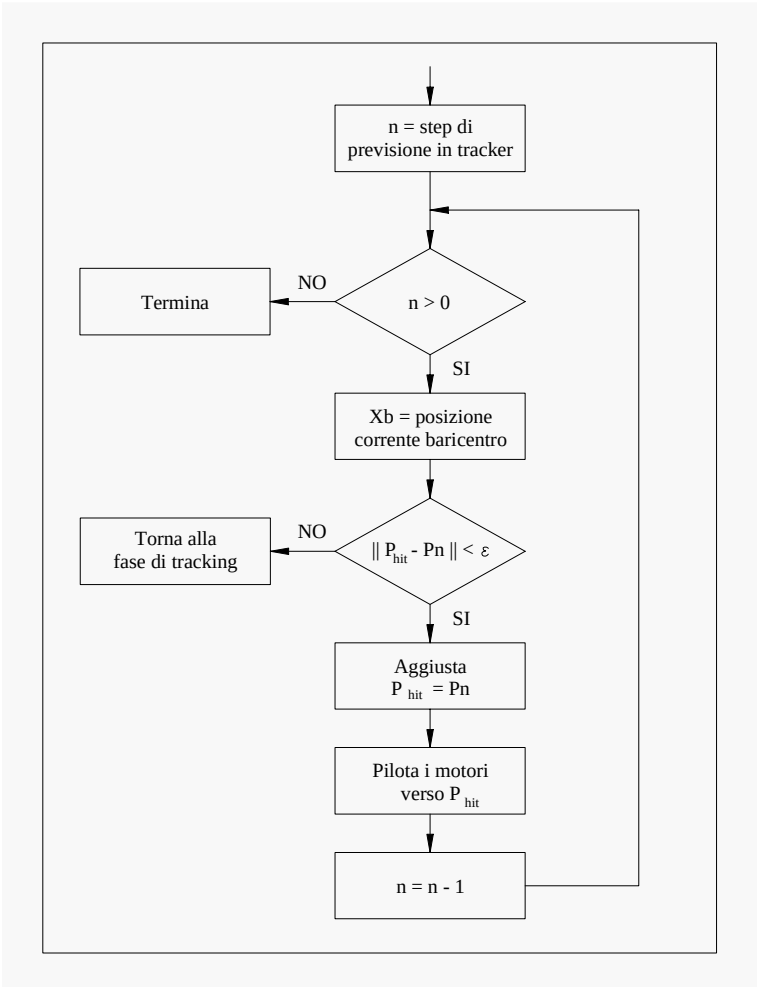


Figura 5.7. Diagramma di funzionamento di ADJUSTER.

Il secondo, ADJUSTER, sempre agganciato al driver del frame-grabber, tiene sotto controllo la traiettoria del target, in modo da verificare che questa non cambi troppo radicalmente; se così avviene riporta il sistema nella fase di tracking: riavvia TRACKER, termina GUNNER e si autotermina. Mentre

GUNNER attende l'istante di apertura della valvola, ADJUSTER continua a prevedere la traiettoria del target, in modo che, se la posizione prevista per l'istante fissato cambia leggermente, è in grado di correggerla e di pilotare i motori di conseguenza.

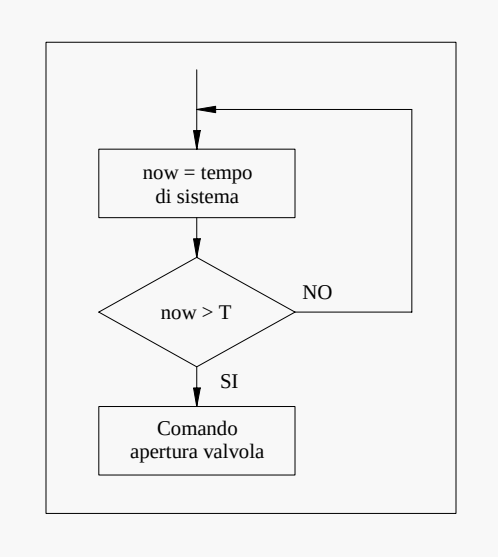


Figura 5.8. Diagramma di funzionamento di GUNNER.

Il problema della soluzione adottata è che, una volta addestrata la rete, non c'è più feed-back per poter correggere eventuali imprecisioni nel posizionamento. Ciò significa che il posizionamento stesso avviene in anello aperto, considerando esclusivamente i valori ritornati dalla rete neurale.

5.4.6 Real-time Kalman simulation

Il filtro di Kalman, dopo essere stato implementato e testato in ambiente MATLAB, è stato importato sotto HARTIK. Per verificarne il corretto funzionamento e per comprendere il comportamento al variare dei parametri in gioco è stato sviluppato un modulo un semplice programma di simulazione in tempo reale denominato SIMKAL. Una schermata tipica dell'esecuzione del

programma è riportata in figura 5.9, nella quale sono chiaramente visibili l'oggetto in moto (il puntino bianco pieno) e la traiettora previsa, costituita dai cerchietti sparsi sulla traiettoria reale dell'oggetto, rappresentata dal cerchio più ampio al centro dello spazio di movimento.

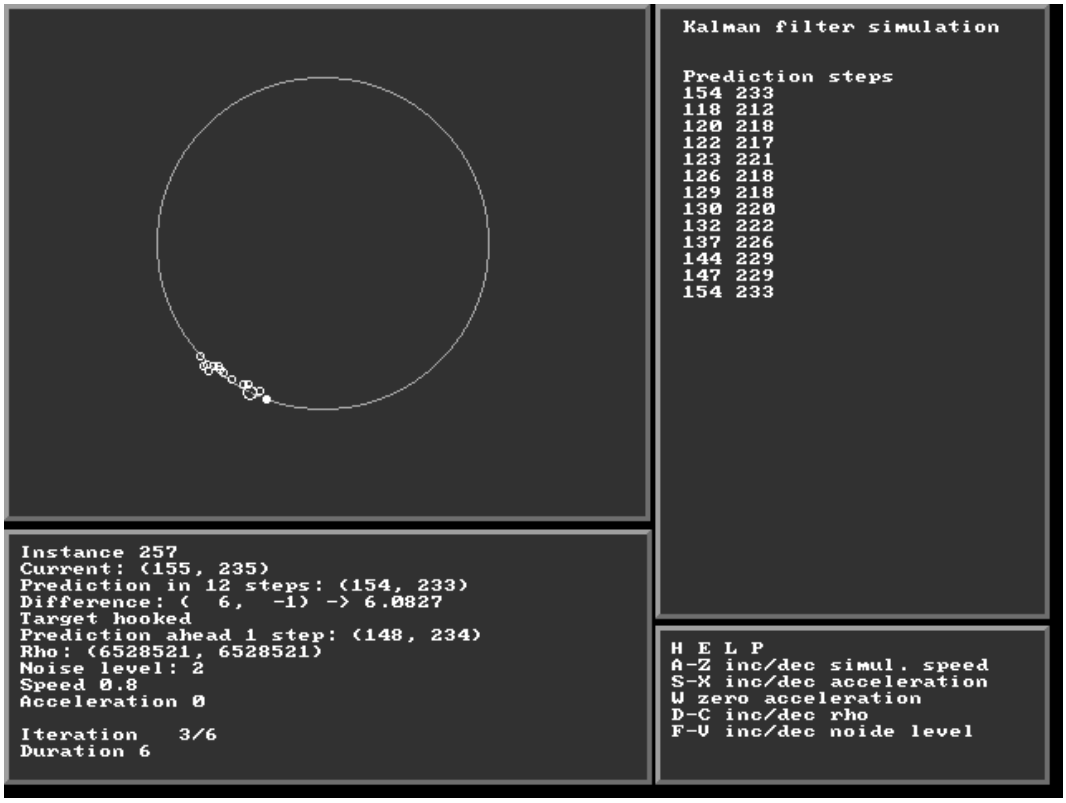


Figura 5.9. Il programma di simulazione SIMKAL.

La simulazione consiste nell'utilizzare il filtro di Kalman per effettuare il tracking di un oggetto che si muove di moto circolare uniforme. Il raggio della traiettoria è fisso e pari a 100 pixel, ed è chiaramente tracciata all'interno della finestra nella quale viene monitorato il movimento dell'oggetto.

Attraverso dei semplici comandi da tastiera è possibile cambiare le caratteristiche della simulazione mentre questa è in corso. I parametri sui quali si può agire sono:

1) accelerazione angolare dell'oggetto. Variando questo parametro è possibile modificare la frequenza del moto. A questo scopo è presente una funzione che annulla istantaneamente il valore dell'accelerazione: quando si è raggiunta la velocità desiderata è sufficiente premere il tasto relativo per fermare la variazione di velocità. Impostando una accelerazione con segno opposto a quello della velocità è possibile invertire il senso del moto;
2) rumore sui rilevamenti della posizione. E' possibile aggiungere del rumore generato in modo pseudo-casuale, aumentandone l'ampiezza per verificare il comportamento del filtro. Il rumore è caratterizzato dunque da una ampiezza massima Δ_{MAX} espressa in pixel, e viene calcolato separatamente sulle due componenti (x e y) del moto. In pratica viene estratto un numero pseudo-casuale compreso nell'intervallo $[-\Delta_{MAX}, +\Delta_{MAX}]$ e viene sommato al valore della coordinata;
3) i ρ che determinano il livello di filtraggio del predittore. I valori di ρ sono due, uno per ciascuna componente del moto. Sarebbe quindi possibile variarli indipendentemente per ottenere diversi livelli di filtraggio sulle due componenti. In questa simulazione, tuttavia, è prevista solo la possibilità di modificarli contemporaneamente della stessa quantità.
4) la velocità della simulazione. Sostanzialmente consiste nell'aumentare o diminuire il periodo del task che genera le coordinate dell'oggetto in movimento e che effettua l'aggiornamento dello stato del filtro. In questo modo è possibile aumentare la velocità di evoluzione della simulazione durante, ad esempio, il transitorio, mentre è possibile diminuirla per studiare il comportamento del filtro a regime. Si noti la differenza con la semplice variazione della velocità angolare dell'oggetto: in questo caso la velocità angolare non viene modificata, anche se l'effetto visivo può essere, a prima vista, il medesimo.

Alcuni parametri, invece, sono fissi per ciascuna simulazione, nel senso che possono essere modificati solo ricompilando il programma. Questi sono il numero di integratori del modello e la lunghezza del treno di previsioni.

L'output del programma, oltre che visualizzare il valore dei diversi parametri in tempo reale, prevede il tracciamento della sequenza delle previsioni, che è molto utile per valutare l'andamento sia del transitorio che il comportamento del filtro a transitorio esaurito, soprattutto in relazione al livello di rumore introdotto.

5.4.7 Velocità del proiettile e ritardo dell'elettrovalvola

La misura della velocità del proiettile e del ritardo di attivazione della valvola viene effettuato abbinando al circuito illustrato al capitolo 3.10 un programma appositamente scritto, il quale ha generato i risultati che sono discussi nella stessa sezione.

Il programma è molto spartano, ed effettua la seguente sequenza di operazioni:

1) avvia HARTIK;
2) inizializza il sottosistema costituito dalla scheda National Instruments PCI6025E, e in particolare imposta i registri per la misurazione della durata di un impulso;
3) esegue un conto alla rovescia di 5 secondi;
4) attiva la valvola, sparando il proiettile contro il sensore. Allo stesso tempo inizia il conteggio del tempo trascorso tra l'attivazione e l'impatto;
5) entra in un ciclo di attesa che si sblocca premendo il tasto ENTER. Il tasto va premuto dopo l'impatto;

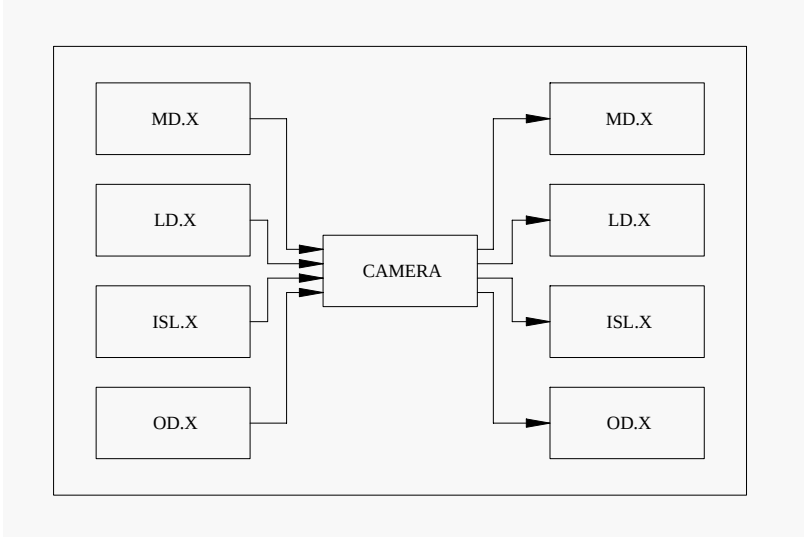
6) viene letto il valore del conteggio dal registro della scheda, che viene a sua volta presentato a video opportunamente convertito in microsecondi, in base alla durata del periodo di clock;

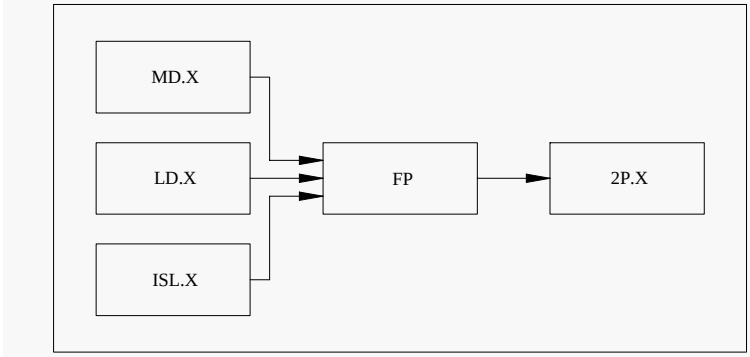
Il programma va rieseguito per ciascuna misurazione, ed è possibile, eventualmente, salvare incrementalmente in un file (*append*) il risultato del test. In alternativa i risultati vanno annotati manualmente per la successiva elaborazione.

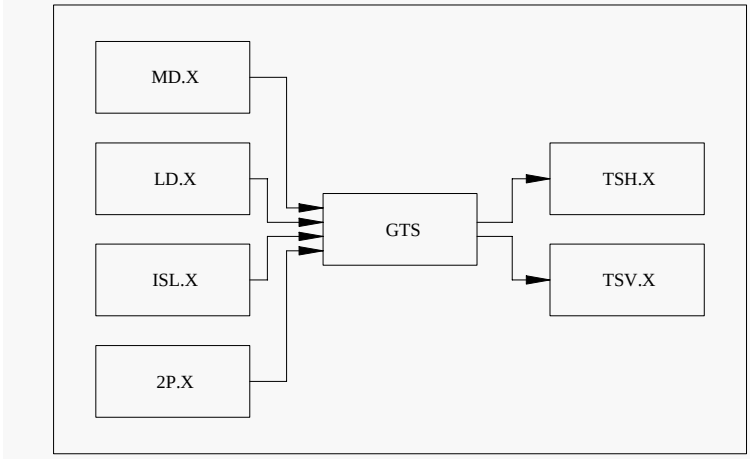
Quando il programma impartisce il comando di apertura alla valvola, attende poi circa 2 secondi prima di richiuderla, invece dei 40-60 millisecondi normalmente usati durante la fase di tracking e sparo. Questo accorgimento si è reso necessario per eseguire automaticamente lo svuotamento parziale del serbatoio del compressore, fino al punto da provocarne la ricarica. Così facendo, ad ogni test, il compressore si trova sempre al massimo livello di pressione, in modo che questo parametro non influisce sul tempo totale misurato.

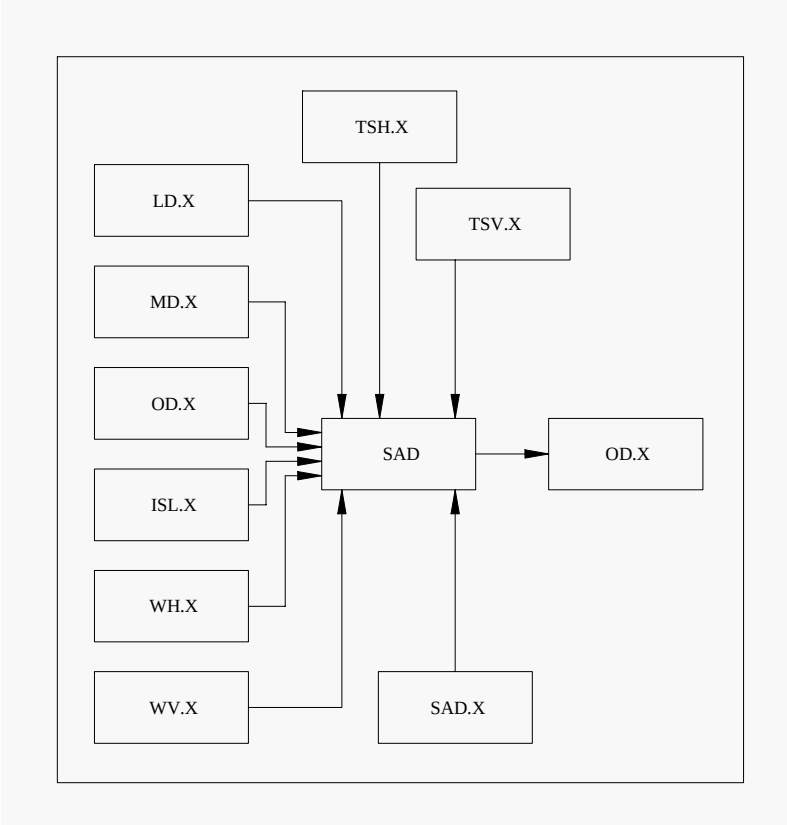
5.5 Moduli esecutivi e data file

Ciascun modulo esecutivo, per poter funzionare correttamente, legge i propri parametri di funzionamento dai data file opportuni e scrive il risultato dell'elaborazione in data file che verranno utilizzati dagli altri moduli. Di seguito sono presentati i diagrammi che illustrano l'I/O dei programmi da e verso i data file.


MD.X
LD.X
ISL.X
OD.X
CAMERA
MD.X
LD.X
ISL.X
OD.X


MD.X
LD.X
ISL.X
FP
2P.X


MD.X
LD.X
ISL.X
2P.X
GTS
TSH.X
TSV.X



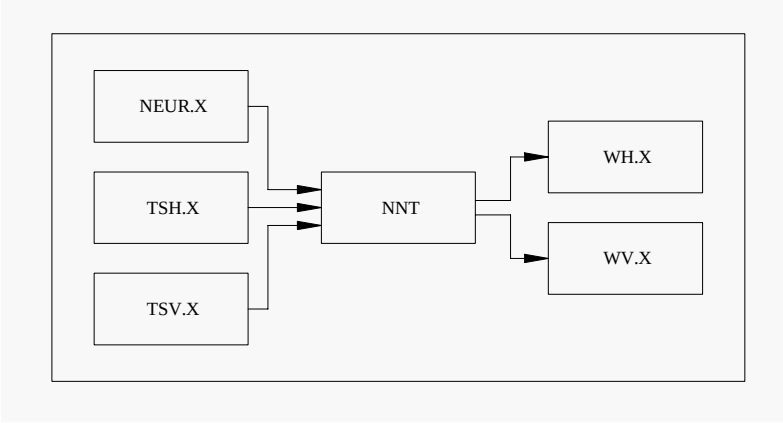


Figura 5.10. Diagrammi modulo - data file.

5.6 Implementazione delle funzioni

Il linguaggio utilizzato per codificare DEA è il GNU C in ambiente operativo MS-DOS, opportunamente esteso per supportare il sistema real-time HARTIK (descritto in dettaglio in appendice A).

Il codice è organizzato in librerie che esportano un certo numero di funzioni, task, tipi di dati e costanti. I diversi moduli utilizzano parte delle funzioni includendo appositi file header secondo le specifiche proprie del linguaggio C. Un breve richiamo: a ciascun file di intestazione (con estensione .H) è associato un file con estensione .C che contiene il codice vero e proprio delle funzioni.

I questo paragrafo e nei successivi sottoparagrafi verranno descritti i diversi file di inclusione (header files) e le funzioni da essi fornite, con la descrizione del funzionamento che è implementato nel relativo file .C. Di seguito sono elencati, in ordine alfabetico, i diversi file di inclusione, con una breve descrizione del contenuto.

- **6025reg.h** contiene la definizione delle costanti simboliche utilizzate per accedere ai diversi registri ed ai singoli bit di ciascun registro della scheda National Instruments PCI-6025E;
- **elabor.h** contiene diverse funzioni di uso generale specifiche di DEA, come la ricerca del baricentro del target e del puntatore laser in una immagine;
- **general.h** contiene funzioni e tipi di dato di uso comune a diversi moduli e alle altre librerie.
- **ioni.h** (I/O National Instruments) esporta le funzioni di alto livello utilizzate per controllare tutte le periferiche di DEA attraverso l'output della scheda PCI-6025E;
- **kc.h** (Keyboard Control) contiene le funzioni di gestione della tastiera e di gestione dell'input dell'utente, da effettuare sempre mediante tastiera;

- **neural.h** contiene tutte le costanti e le funzioni relative alla gestione della rete neurale backpropagation, sia per quanto riguarda l'addestramento che per l'utilizzo della rete addestrata;
- **output.h** contiene tutte le funzioni e le costanti simboliche necessarie a gestire l'output grafico dei moduli esecutivi che lo utilizzano;
- **pci6025e.h** implementa le funzioni di basso livello per l'accesso alla scheda National Instruments PCI-6025E;
- **pxc.h** contiene le procedure necessarie ad utilizzare le funzionalità del frame-grabber Imagenation PXC200L;
- **kalman.h** esporta le funzioni di gestione del predittore di Kalman, in una forma sufficientemente generale da poter essere utilizzata in altri progetti quasi senza modifiche: se gli scopi per cui viene utilizzata sono i medesimi che in DEA, cioè la previsione del moto di un oggetto del quale non si conosce la dinamica, la libreria può essere riutilizzata senza modifica alcuna, anche se i parametri del filtro e del moto cambiassero.

L'elenco completo delle funzioni, delle costanti e dei tipi di dato implementati nei diversi moduli è riportato in appendice B.

Nei paragrafi seguenti verranno spiegati i dettagli di funzionamento propri dei singoli moduli, con un occhio di riguardo all'implementazione delle funzionalità i cui aspetti teorici sono descritti nel capitolo 2.

5.6.1 elabor

elabor.h esporta funzioni utili all'elaborazione delle immagini catturate dal frame-grabber, che servono a trovare le coordinate del baricentro sia del puntatore laser e che del target.Le funzioni che espletano il servizio sono, rispettivamente, la *find_spot()* e la *find_laser_spot()*.

La scansione dell'immagine avviene soltanto nella finestra delimitata dalle costanti *img_XXX_border* (XXX = top, left, bottom oppure right), che indicano la distanza in pixel della finestra dal bordo dell'immagine. Il significato delle costanti è schematizzato al paragrafo 5.3.2. Le suddette costanti vengo lette e salvate in file dalle funzioni *load_isl()* e *save_isl()*.

Una seconda tecnica per migliorare le prestazioni della procedura di scansione è quella di utilizzare le informazioni sulla dimensione dell'oggetto cercato (sia il punto laser che il bersaglio). I dati vengono letti e salvati su file con le routine *load_ld()* e *save_ld()* per il laser, *load_od()* e *save_od()* per il target. Le funzioni di I/O da e verso i file, è bene ricordarlo, vanno usate strettamente al di fuori della coppia di istruzioni *sys_init()* e *sys_end()*, pena il bloccaggio della macchina.

5.6.2 general

Contiene alcune funzioni e tipi di dato di uso comune a diversi moduli e alle altre librerie. In particolare sono state poste in questa libreria alcune utili funzioni di calcolo matematico che vengono usate in abbondanza nel pacchetto. Essendo una semplice collezione di routine a sé stanti, se ne rimanda l'illustrazione all'appendice B.

5.6.3 ioni

Il modulo ioni.h incorpora le funzioni di alto livello da utilizzare per accedere ai componenti hardware di DEA, cioè il controllo dei motori, l'apertura e la chiusura della valvola elettro-pneumatica e l'accensione/spegnimento del laser. Il modulo mantiene informazione sulle operazioni di inizializzazione, in modo da generare un messaggio di errore se vengono invocate delle funzioni prima che sia stato eseguito il setup del sottosistema corrispondente. Questo è particolarmente

vantaggioso durante la fase di sviluppo: le funzioni effettuano dei test su flag opportuni, ed eseguono il codice solo se tali flag garantiscono che il setup è stato effettuato. Per consentire l'utilizzo delle funzionalità messe a disposizione dal modulo è quindi necessario inizializzare i vari sottosistemi interni al modulo, richiamando le primitive apposite.

Il modulo, quindi, gestisce tutti gli aspetti relativi al pilotaggio diretto dell'hardware, costituendo un sottosistema che rende indipendente la logica di controllo dalle strategie adottate per realizzare il controllo stesso. In particolare viene reso trasparente agli strati superiori il meccanismo di movimento dei motori, che implementa le tecniche discusse a tal proposito al paragrafo 3.3.2, e questo mediante un driver apposito che si prende carico di pilotare i motori secondo le richieste degli altri processi. Lo schema di funzionamento e interazione con gli altri processi del driver è schematizzato in figura 5.11, nella cui interpretazione è molto esplicativa la direzione delle frecce, la quale indica la lettura/scrittura delle variabili protette da SRP da parte dei task.

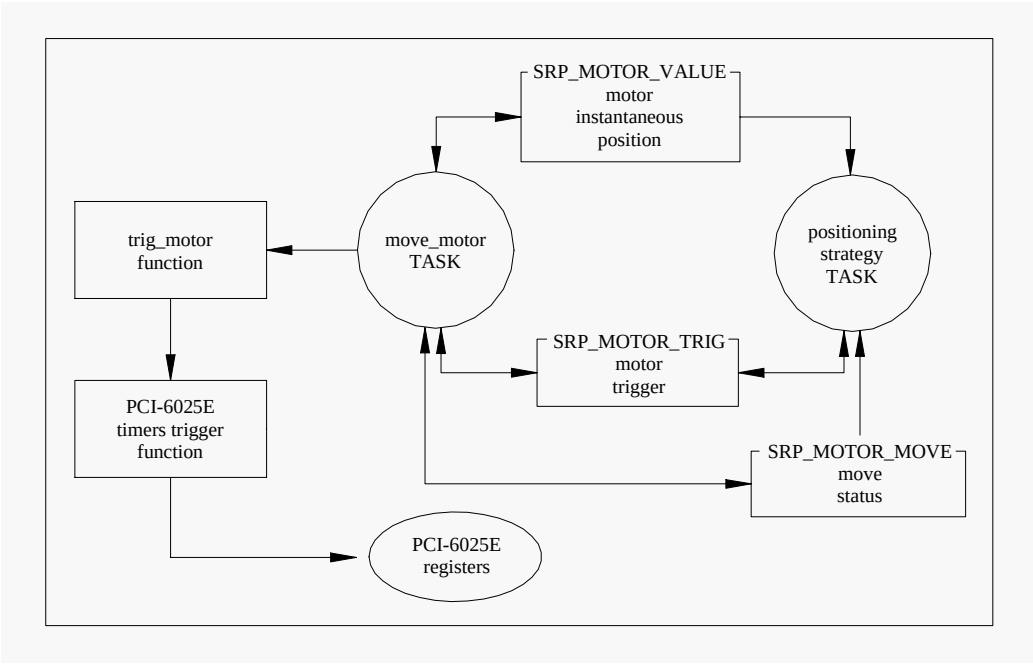


Figura 5.11. Driver di controllo dei servomotori.

E’ in ioni.h che vanno ad essere caricate ed usate le costanti che delimitano i valori di pilotaggio dei motori, in modo che il driver, quando dovesse ricevere il comando di pilotare i motori con un valore fuori dal range corretto, evita di portare a termine l’operazione, la quale causerebbe dei danni, più o meno gravi, all’hardware, in particolare al meccanismo di movimento.

I task implementati in ioni.h usano il protocollo SRP gestito da HARTIK per la condivisione delle risorse tra i processi, secondo quanto illustrato al paragrafo 2.3.3. Analizzando il codice si noterà che ogni processo che utilizza tali risorse, sia in ioni.h che negli altri moduli, lo dichiara in fase di creazione. I codici mnemonici di queste risorse sono:

- **SRP_MOTOR_VALUE** deve essere dichiarata da tutti i task che accedono alle variabili di controllo dei motori in lettura (in scrittura non è possibile);
- **SRP_MOTOR_MOVE** deve essere dichiarata da tutti i processi che accedono alle variabili che indicano lo stato dei motori: in movimento oppure fermi;
- **SRP_MOTOR_TRIG** deve essere dichiarata da tutti i processi che necessitano di pilotare uno o entrambi i motori.

La dichiarazione delle risorse utilizzate avviene come indicato al paragrafo A.6.1 dell’appendice A dedicata ad HARTIK.

L’ultima caratteristica inerente i motori è che, in talune situazioni operative, e nella fattispecie durante la calibrazione del sistema, quando un movimento dei motori causa uno spostamento del puntatore laser sull’immagine, è richiesto che, per esempio, aumentando il valore di pilotaggio del motore orizzontale, si abbia un aumento della coordinata del punto laser sull’immagine. Altro esempio, si desidera che diminuendo il valore di pilotaggio del motore verticale, diminuisca il valore della coordinata verticale del punto. In sostanza, uno spostamento nello spazio delle variabili di controllo dei motori deve corrispondere ad uno

spostamento “nella stessa direzione” nello spazio delle coordinate del punto laser sull’immagine. Questo non è vero in generale, perché dipende innanzitutto da come sono stati montati i motori, ma anche dalla posizione della telecamera relativamente alla base di moto di DEA: è chiaro che ribaltando la telecamera i rapporti vengono invertiti.

Per tenere conto dell’aspetto sopracitato, sono stati introdotti due parametri, *motor_versus_x* e *motor_versus_y*. Ognuna di queste variabili può assumere i due valori discreti +1 e –1: varranno +1 se le direzioni di spostamento nei due spazi di variabili è concorde, secondo quanto spiegato in precedenza, mentre varranno -1 in caso contrario. Una volta impostati i valori, è sufficiente moltiplicarne il valore per un eventuale incremento delle variabili di controllo dei motori per ottenere un movimento delle coordinate concorde con la direzione dell’incremento.

5.6.4 kc

Include le funzioni di gestione dell’input dell’utente, che può avvenire unicamente attraverso la tastiera. Le funzioni esportate sono *wait_for_key()*, *unblock_key_waiting()*, *keyb_action1()* e *keyb_action1()*.

wait_for_key(), la funzione più importante della libreria, può essere utilizzata solo dopo l’avvio di HARTIK, in quanto utilizza le primitive di controllo della tastiera fornite dal sistema. Una chiamata a questa funzione prima dell'avvio di HARTIK o dopo la sua terminazione causa un crash della macchina.

E’ molto usata nel corso del progetto, a causa della tecnica di controllo del flusso di elaborazione usata in tutti i moduli esecutivi.

Ciascun modulo, dopo aver lanciato i processi che effettuano le operazioni proprie del modulo, entrano in un loop controllato dalla tastiera, in modo che l’input dell’utente possa essere catturato e gestito opportunamente. Il loop è effettuato in maniera standard proprio dalla funzione *wait_for_key()* che, in base ai parametri, esegue determinate operazioni alla pressione dei tasti previsti. Due

dei parametri rappresentano i codici ASCII dei tasti che devono essere premuti per terminare il loop: i tasti possibili sono 2 per poter gestire la terminazione in modo il case-unsensitive, cioè consentire la terminazione del loop sia quando viene premuta la maiuscola che la minuscola, nel caso di tasti alfabetici. Se si desidera condizionare il ciclo con un solo tasto (per esempio ESC) basta porre entrambi i parametri uguali allo stesso valore.

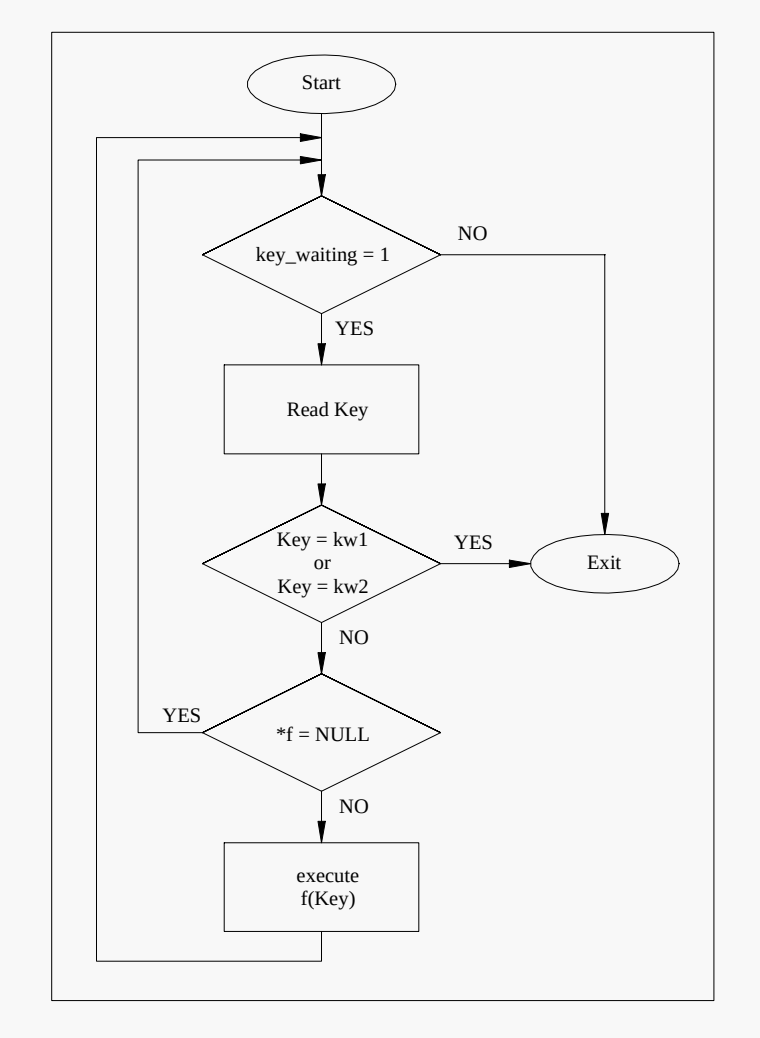


Figura 5.12. Diagramma di flusso di *wait_for_key()*.

Con l'uso della sola primitiva *wait_for_key()* è possibile terminare il ciclo solo alla pressione di un tasto, quindi è richiesta la presenza dell'utente che comandi la terminazione di un programma.

Per fare in modo di uscire dal ciclo a richiesta di un processo, per esempio quando questo termina al propria elaborazione, e quindi garantire un possibile funzionamento batch dell'intero sistema, è stata introdotta la primitiva *unblock_key_waiting(),* che cambia il valore di un flag globale al modulo kc.h, che a sua volta viene controllato ad ogni iterazione del loop in *wait_for_key()*. Cambiando questo flag il loop ha termine senza che venga premuto alcun tasto. Quando un task richiama la funzione *unblock_key_waiting()* il loop ha termine, e il programma principale, bloccato a livello della *wait_for_key()* può proseguire.

La gestione del particolare tasto (o combinazione di tasti) premuto è demandata ad una funzione che accetta come parametro il codice del tasto premuto. Per poter controllare direttamente diversi parametri di funzionamento del software, sia in fase operativa che, soprattutto, in fase di calibrazione, il primo parametro di *wait_for_key()* è quindi un puntatore a funzione che accetta come parametro il codice ASCII del tasto premuto.

Ogni modulo, definendo una propria funzione di questo tipo, la quale in genere effettua uno switch tra i codici dei tasti a cui è sensibile, è in grado di personalizzare la gestione della tastiera a seconda delle necessità.

La funzione di gestione personalizzata va strutturata nel modo seguente:

```
int function_name(int ascii_code)
{
  switch (ascii_code) {
    case key_code : {
      /* Gestione del tasto premuto */
      break;
    }
  }
}
```

Il modulo esporta, inoltre, le funzioni *keyb_action1()* e *keyb_action2()*, che implementano la gestione della tastiera comune alla maggior parte dei moduli del progetto.

5.6.5 neural

Contiene tutte le costanti e le funzioni relative alla gestione della rete neurale, sia per quanto riguarda l'addestramento che per l'utilizzo della rete addestrata.

La rete neurale di DEA, è comunque da evidenziare, viene addestrata con un tool esterno, che è stato sviluppato con una copia molto simile della presente libreria, ma che non funziona in ambiente HARTIK. Questo comporta, in pratica, utilizzare funzioni diverse (con diverso nome) per accedere ai file, cosa che rende impossibile lo sviluppo di una unica libreria per il normale ANSI C e per HARTIK. La scelta di non utilizzare HARTIK per l'addestramento è motivata dal fatto che per il GNU C esiste una potente libreria grafica, appositamente studiata per programmare i videogames, che garantisce maggiore velocità, potenza e affidabilità del motore grafico di HARTIK. Inoltre l'addestramento non richiede vincoli temporali espliciti, ed è quindi possibile effettuarlo in ambiente non real-time.

In seguito si descriverà unicamente la versione per HARTIK della libreria, in quanto la versione per GNU C è perfettamente duale.

La parte più importante della libreria sono i tipi di dato esportati: le strutture *NET* e *TSET*, che rispettivamente incapsulano una rete neurale back-propagation e il training set da utilizzare per l'addestramento. Definendo variabili di questo tipo è possibile, in un unico programma, utilizzare più reti neurali e più training set diversi, indipendentementi tra loro. Le funzioni operano su strutture di questo

tipo, che contengono i dati di partenza e anche i risultati dell'elaborazione. Il tutto rimane racchiuso all'interno della struttura.

Per coloro i quali non sono digiuni di programmazione object oriented, dovrebbe apparire immediatamente evidente che un tale approccio implementativo per questa libreria, utilizzando un qualsiasi linguaggio orientato agli oggetti (come il C++), sarebbe stato decisamente più elegante. Purtroppo lo sviluppo in HARTIK, a causa del controllo di basso livello della macchina richiesto per realizzare un sistema real-time robusto, non prevede estensioni ad oggetti.

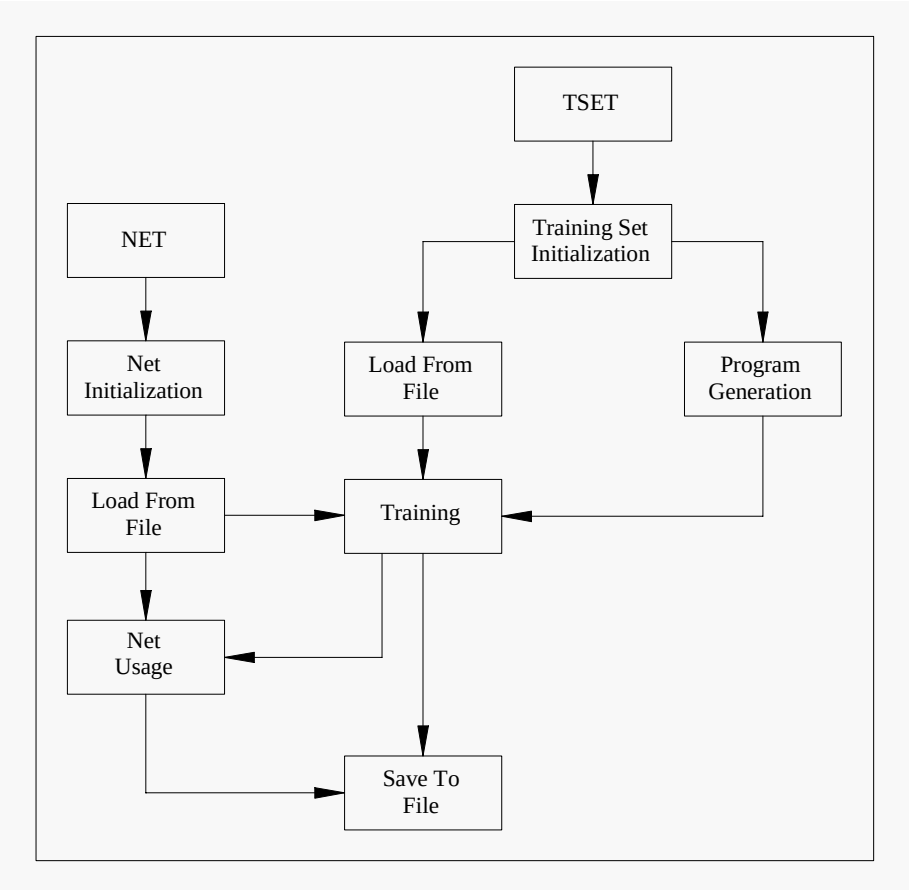


Figura 5.13. Uso della libreria neural.h.

Per illustrare l'utilizzo della libreria bisogna distinguere due diverse fasi: l'apprendimento e l'utilizzo di una rete addestrata: diverse funzioni, infatti, servono soltanto alla fase di training.

In entrambi i casi è prevista una prima fase di inizializzazione delle variabili utilizzate. E' necessario innanzitutto dichiarare la struttura che contiene la rete, e inizializzarla con la funzione *init_net()*. In alternativa è possibile caricare automaticamente una rete da file (istruita o meno) grazie alla funzione *load_net()*.

A questo punto è necessario eseguire il training la rete, se non la si è caricata da file già istruita. Per fare ciò è necessario utilizzare le strutture dati di tipo *TSET*, che vanno dichiarate e riempite con i training sample. In genere questi ultimi vengono letti da file attraverso la funzione *load_tset()*. Dopo aver impostato i parametri che definiscono la sequenza di training, per mezzo della funzione *set_learning_parms()*, è possibile eseguire uno o più cicli di apprendimento richiamando il numero di volte necessario la funzione *learn()*.

Una rete neurale addetrata può essere semplicemente utilizzata mediante le funzioni *set_input()*, *compute_out()* e *get_output()*.

Infine è possibile salvare in un file una rete, rendendola disponibile per un successivo utilizzo, per mezzo della procedura *save_weight()*.

5.6.6 output

Il modulo *output.h* contiene una serie di routines predisposte per realizzare una interfaccia utente grafica comune ai diversi moduli del progetto. Tutta la gestione dell'output dei vari moduli di DEA si poggia sulle funzioni esportate da questa libreria.

Il motore grafico viene avviato richiamando la funzione *init_output_engine()*, che, non accettando parametri, inizializza la modalità grafica con la risoluzionedi 640x480 punti e 256 livelli di grigio. Per verificare lo stato

del motore grafico è fornita la funzione *graph_mode()*, che restituisce 1 nel caso di modalità grafica e 0 se ci si trova in modo testo.

La funzione più importante della libreria è *make_default_screen()* che serve a generare una schermata standard per i diversi moduli di DEA, come quella riportata in figura 5.14. La disposizione, la forma e le dimensioni dei vari oggetti presenti nella schermata sono definiti da costanti nella libreria, in modo che, cambiando semplicemente queste costanti, si è in grado di personalizzare l'aspetto dell'applicazione.

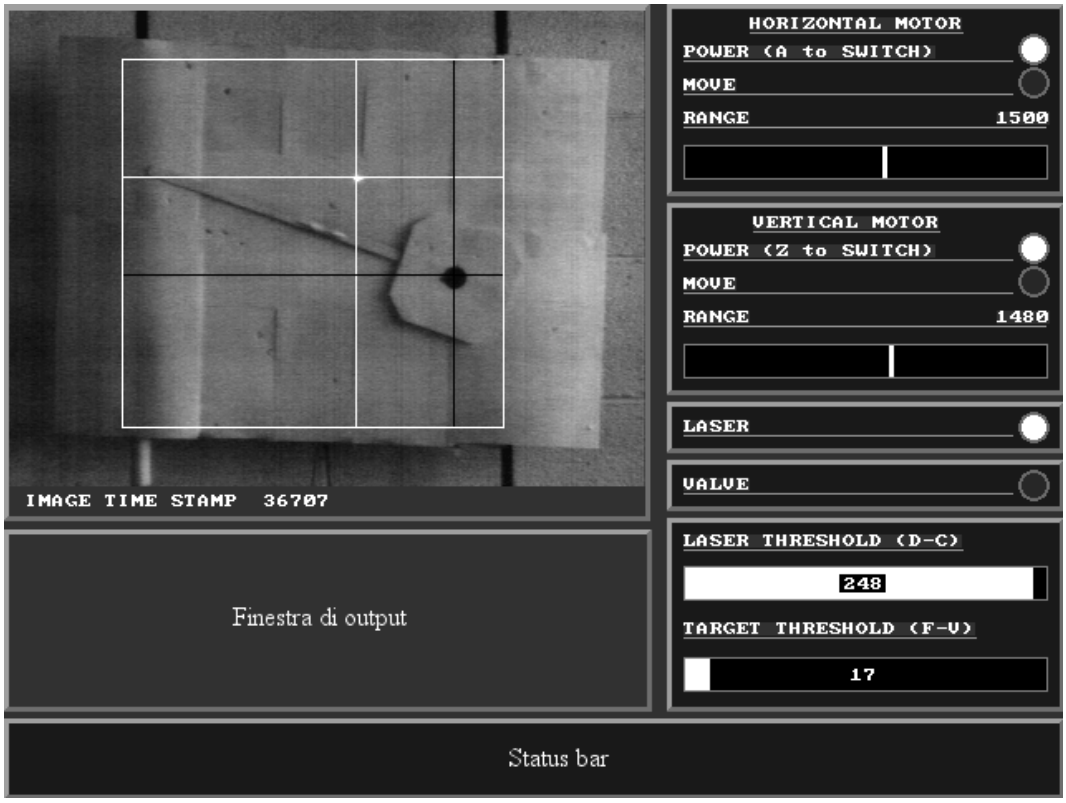


Figura 5.14. Schermata di default per i moduli esecutivi di DEA.

Le altre funzioni sono utilizzate per aggiornare i dati sul video durante l'esecuzione dei vari moduli. In particolare *image_to_screen()* stampa a video il frame catturato dal grabber ed il time stamp relativo. Le altre funzioni, come *update_driver(), update_laser_thresh_bar()* e *update_target_thresh_bar(),*

servono anch'esse ad aggiornare il contenuto dello schermo durante l'esecuzione dei programmi: vengono aggiornati lo stato del driver dei servo e il livello delle soglie per il puntatore laser e per il bersaglio.

Sono fornite le procedure per gestire la barra di stato, alla quale è associato un task periodico con criticità SOFT e periodo opportuno, che realizza l'eventuale lampeggio del testo visualizzato nella barra di stato. Un task che desidera scrivere nella status bar deve indicare, al momento della creazione, di utilizzare la risorsa *SRP_STATUSBAR*. L'effettiva scrittura del testo avviene con la procedura *set_statusbar()*, la quale permette anche di indicare se il testo deve lampeggiare o rimanere fisso sul video. Il modulo esecutivo che utilizza la barra di stato deve eseguire la procedura *go_statusbar_MOD()*, il quale serve a creare e ad avviare il task che gestisce la barra.

Infine, la libreria fornisce gli strumenti per effettuare l'hardcopy di una schermata e salvarla in un file in standard PGM. Il PGM è un formato grafico particolarmente semplice, che adatto ad immagini di qualsiasi dimensione ma soltanto monocromatiche a 256 livelli di grigio.

Per abilitare l'hardcopy al termine dell'elaborazione, il programma principale deve richiamare la procedura *enable_hard_copy()*. Quando l'elaborazione del programma principale termina, chiamando la funzione *shut_down_sys()*, nel modulo elabor.h, viene effettuato l'hardcopy dello schermo e salvato in una variabile interna alla libreria. Questo avviene dopo la chiusura del sistema (ovvero dopo la *sys_end()*), in modo da non avere interferenze da parte dei task che scrivono sullo schermo. Solo dopo aver completato l'operazione viene chiusa anche la grafica. Rimane compito del programma principale richiamare la funzione *save_pgm_screen()* per la scrittura vera e propria nel file.

5.6.7 pci6025e

Il modulo *pci6025e.h* implementa le procedure di basso livello per l'accesso alle funzionalità della scheda National Intruments PCI-6025E.

Nel file **6025reg.h** sono dichiarate le costanti simboliche che servono a migliorare la chiarezza del codice, nonché ad effettuare delle chiamate alle funzioni corrette senza conoscere i codici numerici dei singoli parametri.

Un esempio dell'utilizzo delle funzioni di controllo dei timer è rappresentato dal seguente frammento di codice:

```
init_pci_mite();
init_GPCT_DIO();
G0_reset();
timer_clock_configure(0);
PFI_pin_program(PFI_PIN_4, PFI_SET_INPUT, PFI_7_NONE);
G0_continuos_pulse_train_generation(G_IN_TIMEBASE2,
       G_COUNT_RISING_EDGE, G_LEVEL_GATING,
       G_GATE_PFI4, G_DISABLE_INVERSION, G_TOGGLE_ON_TC,
       G_ACTIVE_LOW, G_ENABLE_SWITCHING, 5, 10, 4);
enable_G0_out();
G0_Arm_All();
G0_pulse_train_change(4, 10);
```

Nell'esempio, dopo l'inizializzazione del bus PCI e quella dei contatori, viene impostato il clock di default per entrambi i timer, con frequenza di 100 KHz, e viene programmato il pin 4 dei PFI per essere utilizzato come segnale di abilitazione del conteggio. Successivamente viene programmato il timer per generare una onda quadra con le seguenti caratteristiche:

- 5 periodi di clock di attesa prima di generare il primo impulso;
- un low-time pari a 10 periodi di clock,
- un high-time pari a 5 periodi di clock, quindi un duty-cycle pari a 1/3.

5.6.8 pxc

Il modulo *pxc.h* esporta le procedure necessarie all'interfacciamento con il frame-grabber Imagenation PXC200.

Per l'utilizzo di tale frame-grabber è stato predisposto un apposito driver. Il suo utilizzo è molto semplice, e prevede i seguenti passi:

- richiamare la funzione **PXC_Initiate()** per effettuare il setup della scheda. Questa funzione esegue le seguenti operazioni:
  - inizializzazione del bus PCI al quale è connessa la scheda della Imagenation;
  - abilitazione la scrittura in memoria tramite DMA;
  - controllo della presenza del video input;
  - creazione del CAB che contiene l'immagine catturata dal grabber;
  - scrittura nei registri della scheda per indicare al frame-grabber le caratteristiche desiderate per l'immagine, come il fattore di scala, le temporizzazioni e le dimensioni, sia orizzontali che verticali, inoltre inserisce l'indirizzo dell'handler nel canale DMA;
  - infine, c'è l'inizializzazione della coda nella quale sono memorizzati i puntatori ai task che necessitano della lettura dal CAB.

  La **PXC_Initiate()** accetta come parametro il numero di processi che si prevede debbano utilizzare concorrentemente il CAB. Questo valore va incrementato di 1 per il corretto funzionamento del driver. La funzione restituisce il periodo di attivazione minimo dei task che vengono inseriti nella coda.
- Per attivare l'handler occorre richiamare la funzione **PXC_Start()**.
- Per poter accedere al CAB contenente le immagini è necessario creare un task (si veda l'appendice A per le modalità di creazione) aperiodico, con criticità HARD e con tempo minimo di interarrivo pari a un multiplo intero del periodo di attivazione restituito da **PXC_Initiate()**.

- Dopo aver attivato l'handler e creato il task (non necessariamente in quest'ordine), è possibile inserire il processo nella coda gestita dal driver, mediante la procedura **PXC_Push_Listener()**, che accetta come parametro il task PID.
- All'interno del processo va utilizzata la funzione **PXC_GetCab()** che ritorna un puntatore al CAB contenente l'immagine, il quale viene poi manipolato con le normali funzioni di accesso ai CAB fornite da HARTIK;
- Se si desidera eliminare un task dalla coda e quindi fare in modo che non venga attivato automaticamente dal driver, è possibile farlo grazie alla funzione **PXC_Pop_Listener()**;
- Infine, quando non si ha più necessità di utilizzare il driver, è possibile terminare il sottosistema richiamando la **PXC_Stop()**.

La versione attuale del driver supporta soltanto un numero limitato di modi operativi, in particolare:

- periodo di campionamento fisso pari a 20 millisecondi;
- immagini a 256 livelli di grigio;
- dimensione dell'immagine pari a 384x288 pixel.

5.6.9 kalman

Il presente modulo di inclusione implementa le funzionalità del predittore di Kalman, la cui teoria è descritta al paragrafo 2.5. Le caratteristiche principali della libreria sono le seguenti:

- possibilità di utilizzare diversi filtri in parallelo, alimentati dagli stessi valori di ingresso, che evolvono indipendentemente ciascuno dagli altri;
- possibilità di impostare l'ordine del modello di ciascun filtro della batteria in modo indipendente dagli altri;

- due procedure di inizializzazione diverse, a seconda della sorgente dei parametri di impostazione: da programma utente oppure automatizzata da file esterno;
- due strategie di predizione in avanti diverse ed indipendenti: una istantanea e l'altra bufferizzata;
- memorizzazione degli ultimi valori di ingresso forniti alla batteria di filtri;
- possibilità di utilizzare un solo modello per il sistema da osservare;
- caricamento dei guadagni da file esterno in una lookup table;
- sviluppo di tool esterni per la simulazione del funzionamento dei sistemi osservati e per la generazione dei guadagni.

I valori che limitano le funzionalità della libreria sono definiti da alcune costanti all'interno del file *kalman.h*, e sono i seguenti:

Massimo ordine del modello di previsione	7
Numero massimo di filtri indipendenti	2
Dimensione del buffer di previsione	100
Numero di ρ caricabili da file	130
Dimensione del buffer dei valori di input	10

Questi valori possono essere modificati semplicemente ricompilando la libreria, variando i valori delle *#define* all'interno del codice di *kalman.h*.

La libreria può dunque trattare sistemi con più ingressi, e con uscite che possono essere indipendenti tra loro: di fatto gli stessi ingressi vengono forniti a una batteria di filtri in parallelo, i cui stati evolvono indipendentemente gli uni dagli altri.

L'altra caratteristica fondamentale della libreria è che, i valori dei guadagni del filtro vengono letti da file e caricati in una look-up table, dalla quale vengono prelevati secondo opportuni criteri. Esiste un file di guadagni per ogni possibile dimensione del modello che, per le ragioni esposte nella trattazione teorica, è bene non sia troppo elevata. Questi file possono essere comodamente generati mediante

un programma di elaborazione matriciale. In DEA si è sviluppata una piccola applicazione che gira in ambiente MATLAB, i cui dettagli sono illustrati in appendice C.

Ai fini del funzionamento del pacchetto, a prescindere dagli strumenti utilizzati per creare i file dei guadagni, è importante che questi ultimi abbiano la struttura di un file di testo organizzato secondo quanto descritto al paragrafo 5.3.3. La procedura di caricamento automatico legge le impostazioni dal data file relativo e setta opportunamente il sottosistema.

La scelta di creare una look-up table dalla quale prelevare i valori dei guadagni corrispondenti ai diversi valori di ρ ha motivi puramente di efficienza e, in un certo senso, di semplicità: è infatti molto comodo e agevole manipolare matrici e creare file con la struttura desiderata all'interno di tool appositamente progettati, che mettono a disposizione funzionalità potenti a tale scopo; non è proprio così agevole effettuare le stesse elaborazioni all'interno di codice C, che, tra l'altro, non dispone di librerie sviluppate per il calcolo matriciale. Inoltre, la sensibilità del filtro al variare di ρ non è elevatissima. E' così possibile campionare, in un determinato range e con una opportuna granularità, diversi valori di ρ, calcolando i guadagni relativi, ed essere sicuri che utilizzare tali valori di ρ campionati produce lo stesso risultato che calcolando i guadagni relativi un ρ arbitrario.

L'inizializzazione del sistema può avvenire richiamando le funzioni appropriate con i rispettivi parametri, oppure eseguendo una unica funzione, che legge i parametri da file e richiama le medesime funzioni passandovi tali valori (vedi appendice A, paragrafo 5).

La tipica sequenza base di comandi per inizializzare il predittore è la seguente:

```
init_kalman(int nvi, int *orders);
```

```
void load_gains(int d, char *fname);
... avvio di HARTIK ...
rho = set_rho(int d, float rho, int seekmode);
set_prediction_step(int n);
```

Come si può notare, la prima operazione da fare è l'inizializzalizzazione del sottosistema costituito dal predittore, o meglio, dalla batteria di predittori. *init_kalman()* serve appunto a definire quanti filtri indipendenti e in parallelo si desidera utilizzare, col paramentro *nvi*, e l'ordine del modello di ciascun filtro (*orders* è un vettore contenente le dimensioni dei singoli filtri). Ora si può procedere con la lettura dei guadagni che, come qualsiasi accesso ai file, deve essere effettuata prima dell'avvio del sistema real-time. Si deve richiamare la funzione *load_gains()* per ciascun filtro della batteria, in modo da caricare i guadagni della dimensione corretta. Infine, le funzioni *set_rho()* e *set_prediction_step()*, che possono invece essere chiamate sia prima che dopo l'avvio di HARTIK, servono rispettivamente ad impostare il ρ iniziale e i relativi guadagni ed a specificare il numero di step in avanti ai quali effettuare le previsioni che verranno memorizzate nel buffer. In effetti, al limite, la libreria potrebbe essere utilizzata appieno anche senza avviare HARTIK, in quanto è costituita da semplici funzioni che non richiedono elaborazione real-time.

La seconda modalità di inizializzazione, l'inizializzazione da file, avviene chiamando una sola funzione, nel seguente modo:

```
void init_kalman_subsystem(char *fname);
```

La funzione si preoccupa di impostare il sistema, effettuando i medesimi step visti in precedenza, utilizzando i parametri specificati nel file *fname*.

Per poter funzionare correttamente, il file di inizializzazione, che è un normalissimo file di testo, deve avere la struttura illustrata al paragrafo 5.3.4.

La funzione *init_kalman_subsystem()*, dovendo accedere ai file, deve essere chiamata prima dell'eventuale avvio di HARTIK.

L'evoluzione dello stato interno della batteria di filtri avviene richiamando la funzione *add_survey()*, che provvede a inserire il nuovo rilevamento nel buffer degli input e calcola il nuovo stato di ciascun filtro della batteria secondo quanto illustrato al paragrafo 2.5.5, nel quale si descrivono le basi teoriche del filtro di Kalman.

La caratteristica più importante della libreria è, senza dubbio, l'implementazione di due diverse strategie di predizione indipendenti fra loro, di cui una bufferizzata. Queste due strategie sono esemplificate in figura 5.15.

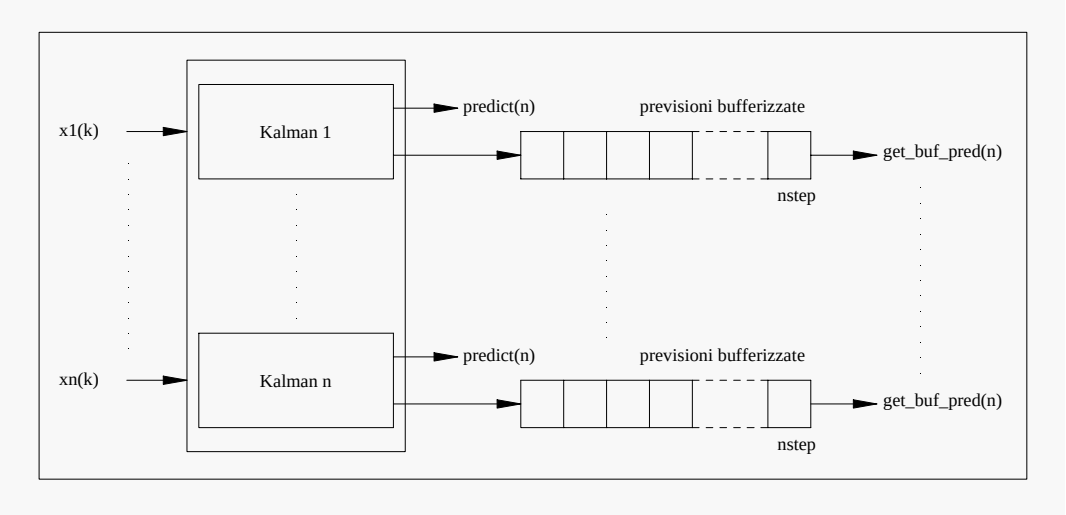


Figura 5.15. Strategie di previsione della libreria.

La prima e più semplice strategia di previsione può essere effettuata richiamando la funzione *predict()*, nel seguente modo:

```
predict(TSurvey *pr, TSurvey s, int n);
```

Alla funzione devono essere passati *s*, cioè il nuovo valore di input e *n*, cioè il numero di step in avanti al quale deve corrispondere la previsione. Nel

parametro *pr*, passato per riferimento, viene restituita la previsione attuata. Il predittore, a partire dal suo stato interno attuale, che è evoluto con le chiamate successive alla funzione *add_survey()*, calcola i valori previsti *n* step avanti dall'istante attuale e ne restituisce il valore in *pr*. Il vantaggio principale di questa strategia è la possibilità di predire in avanti di un numero arbitrario di passi.

Questa previsione non influenza in alcun modo le previsioni memorizzate nel buffer, e questo può essere utilizzato proficuamente per controllare la veridicità delle previsioni bufferizzate, come si illustrerà in seguito.

La seconda strategia di predizione fa uso delle funzioni *set_predictione_step()* e *get_buf_pred()*. La prima serve ad impostare, in fase di inizializzazione del filtro, il numero di passi in avanti a cui si desidera effettuare le previsioni ogni qualvolta lo stato del filtro viene aggiornato (con *add_survey()*). Le predizioni vengono poi immagazzinate in un buffer, dal quale possono essere lette richiamando la funzione *get_pred_buf()*. Con questa strategia è possibile conoscere la previsione per un istante successivo effettuata ai passi precedenti. Ad ogni step, man mano cioè che lo stato del filtro evolve e che si presentano nuove previsioni, viene scartata quella più vecchia e il buffer viene aggiornato. Per maggior chiarezza si faccia riferimento ai seguenti schemi, che illustrano come il buffer viene aggiornato e cosa significano i valori ivi memorizzati.

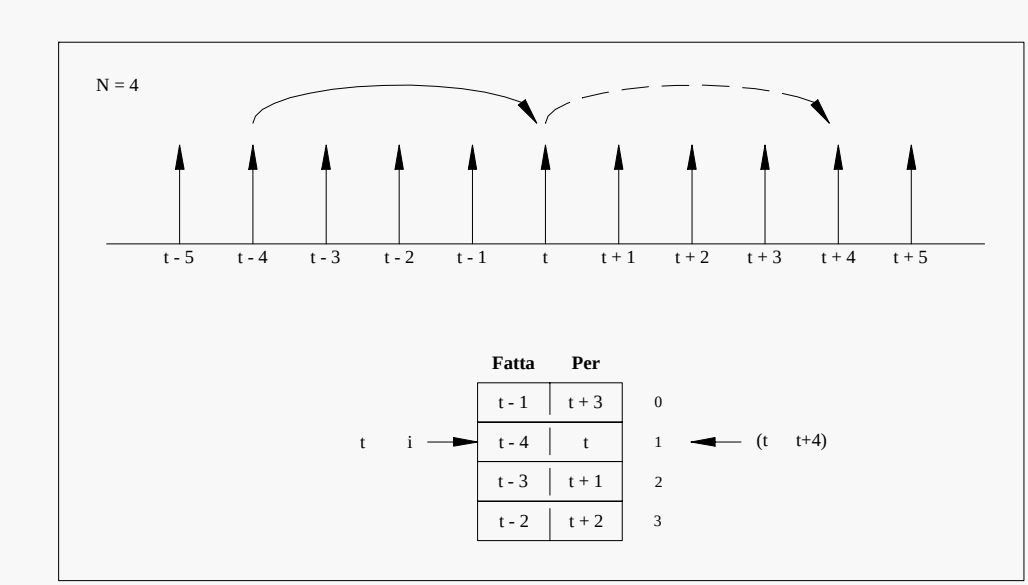


Figura 5.16. Buffer circolare del predittore di Kalman.

La lettura dal buffer circolare avviene in due modi diversi a seconda che il buffer stesso sia già stato aggiornato con una nuova previsione oppure no. Se la previsione non è ancora stata inserita e si desidera estrarre la previsione per l'istante t+k, la locazione dal quale prelevare il valore si calcola come

$$(i + k) \% N$$

dove "%" indica l'operatore modulo. Se la previsione è già stata inserita, la locazione dalla quale prelevare il valore è

$$(i + k - 1) \% N.$$

Il difetto di questo approccio è che il numero di step in avanti al quale si può effettuare la previsione è limitato dalle dimensioni del buffer. Questo, in realtà, non è una limitazione veramente stringente, in quanto, innanzitutto, le dimensioni

del buffer possono essere impostate semplicemente ricompilando la libreria e, soprattutto, raramente, se non mai, capita di dover effettuare delle previsioni troppo in avanti nel tempo.

Le due strategie di previsione illustrate possono essere efficacemente combinate per verificare la bontà delle previsioni. Un esempio può essere molto più esplicativo di una semplice descrizione generale:

1) il valore di step in avanti per le previsioni bufferizzate viene posto pari a n;
2) a ciascun passo, chiamando la funzione *get_pred_buf(&pr, 1)* si ottiene la previsione per l'istante corrente effetuata n passi addietro;
3) si confrontano tale previsione con la posizione corrente: se sono "sufficientemente" concordi il filtro ha agganciato lo stato dell'oggetto e può effettuare delle previsioni corrette a lungo termine, altrimenti si trova ancora in fase di assestamento, oppure è necessario cambiare il ρ.

Come si evince dalla precedente trattazione, la libreria è studiata in modo da essere il più generale possibile, l'unica vera limitazione è quella di utilizzare una unica tipologia di modelli per il sistema da prevedere, che si traduce in una forma particolare per la matrice A, e per il vettore C, come descritto al paragrafo 2.5.3.

Modificare la libreria in modo da adattarla ad esigenze più specifiche è comunque un'operazione estremamente semplice. Le modifiche da apportare per poter trattare sistemi con modelli diversi da quello attuale sono le seguenti:

1) la funzione *init_kalman()* va cambiata in modo che costruisca la matrice A adatta al modello che si intende utilizzare;
2) nella funzione *add_survey()* va utilizzata l'opportuno vettore C, che in questa funzione non compare per motivi di ottimizzazione, in quanto, nel caso di DEA è importante soltanto il primo valore presente nel vettore di stato;

3) la funzione *predict()*, ssendo una versione accorpata delle due precedenti, deve ricevere entrambe le modifiche illustrate nei punti 1 e 2.

Ulteriori e più affinate versioni della libreria dovrebbero prevedere il passaggio dinamico delle matrici A, B, e C che modellizzano il sistema del quale si vuole stimare lo stato in fase di inizializzazione. Inoltre grande peso avrebbe il poter generare i guadagni del predittore senza leggerli da tabelle precompilate, e questo richiederebbe di integrare un libreria di elaborazione matriciale, in quanto sono ricieste diverse operazioni tra matrici, tra cui prodotti, trasposizioni e, su tutte, l'inversione di matrici quadrate, il tutto con dimensione variabile. Purtroppo, verosimilmente, in ambiente HARTIK non si potrebbero utilizzare librerie (ve ne sono molte) prodotte per il normale ANSI C, in quanto la maggior parte di queste, tra le altre funzionalità, prevedono l'allocazione dinamica delle matrici in base alle dimensioni necessarie, e in HARTIK questa è una operazione non consentita. La libreria di calcolo matriciale andrebbe sviluppata ad hoc per HARTIK.

5.7 Software MATLAB™

Lo sviluppo sotto HARTIK del modulo relativo al predittore di Kalman è passato attraverso una prima fase di simulazione e sviluppo in ambiente MATLAB™.

MATLAB è un pacchetto software prodotto dalla Mathworks™, disponibile sia in ambiente Windows™ che per Macintosh™, studiato per effettuare il calcolo matriciale in modo semplice ed immediato. MATLAB dispone di un potente ambiente integrato nel quale è possibile realizzare velocemente programmi che necessitano di calcoli più o meno complessi su matrici. E' inoltre possibile produre grafici di funzioni 2D e 3D, ed utilizzare complessi strumenti per la simulazioni di sistemi dinamici, reti neurali, filtraggio digitale e quant'altro.

Il filtro di Kalman è stato prima codificato in MATLAB, opportunamente simulato ed infine importato sotto HARTIK. Questo ultimo passaggio ha richiesto la scrittura di codice per realizzare il calcolo matriciale, che invece in MATLAB è completamente trasparente. La simulazione è stata di fondamentale importanza anche per testare il codice C, in quanto è stato possibile fornire in ingresso il medesimo segnale sia al programma in MATLAB che alla simulazione real-time in HARTIK, controllando l'output prodotto per confrontarne il comportamento. Questo ha permesso di sviluppare una libreria per HARTIK robusta e affidabile.

Il codice MATLAB del filtro di Kalman è riportato in appendice C, in quanto può essere di grande utilità per capirne il funzionamento, e certamente lo è molto di più che analizzare centinaia di righe di codice C.

Il MATLAB è stato utilizzato anche per un'altra importante funzione, indispensabile per il corretto funzionamento della libreria, ovvero la generazione dei data file che contengono i ρ ed i relativi guadagni per il filtro. Anche il codice di questo programma è riportato in appendice C.

Questa scelta si è resa necessaria al fine di evitare di dover scrivere una vera e propria libreria di calcolo matriciale per HARTIK, in quanto, oltre che ad operazioni banali quali prodotti e trasposizioni di matrici e vettori, il calcolo dei guadagni per il filtro di Kalman prevede il calcolo dell'inversa di una matrice quadrata, che richiede la scrittura di codice per il calcolo numerico affatto banale.

Dunque si è preferito scrivere un semplice programma in MATLAB che genera, per qualsiasi dimensione del modello del sistema, un file contenente svariati valori del ρ e dei guadagni corrispondenti. Con banalissime modifiche al codice è possibile aumentare il numero ρ diversi da inserire nel file (attualmente sono 120), oppure variare il range entro il quale campionare i valori di ρ stessi, che, al momento, va da 10^{-10} a 10^{+12}.

Capitolo 6

Risultati

6.1 Velocità del proiettile e ritardo della valvola

Innanzitutto vengono descritto i risultati riguardanti i test sulla velocità del proiettile e sul ritardo dell'elettrovalvola. I valori ricavati dagli esperimenti sono riportati in tabella 1. Da questi valori si può estrapolare una stima per il ritardo di attivazione della valvola, che è sconosciuto, e quindi stimare anche la velocità della pallina sulla base del tempo di volo.

Sono stati effettuati test a quattro diverse distanze: 30 centimetri, 1 metro, 1.78 e 1.99 metri. La minima distanza, pari a 30 cm, è determinata unicamente dalla lunghezza della canna entro la quale viene posto il proiettile.

distanza [m]	0,3	1	1,78	1,99
colpo				
1	19,03	27,71	38,03	40,56
2	18,95	27,67	38,78	41,12
3	19,16	28,09	37,38	41,5
4	18,87	27,63	37,83	41,17
5	18,96	28,12	36,75	41,74
6	18,81	27,36	38,12	40,78
7	19,01	27,57	37,5	40,83
8	18,91	27,88	37,27	39,99
9	18,98	27,85	37,08	40,44
10	19,27	27,37	37,24	40,64
11	19,21	27,57	36,89	40,28
12	18,96	27,51	37,39	40,38
13	19,15	27,46	37,06	41,45
14	18,1	27,57	37,25	40,64
15	20,24	27,94	37,55	41,36
16	19,05	27,99	38,67	42,18
17	18,94	27,31	39,37	41,94
18	18,8	27,95	38,21	41,93
19	19,06	27,51	39,05	42,27
20	18,97	27,24	37,77	41,95
21	19,1	27,67	39,05	42,2
22	18,96	28,01	38,25	41,89
23	19	27,8	38,24	42,37
24	18,91	27,64	38,33	41,5
25	19,1	27,83	38,69	41,88
media	19,02	27,69	37,91	41,3196

Tabella 1. Risultati dei test.

La pressione del compressore, per ogni tiro, è pari a quella che si ottiene caricandolo completamente, tenuto conto delle immancabili imprecisioni dei meccanismi. Il valore di pressione è quindi pari a circa 8 atmosfere.

Per ciascuna distanza sono stati considerati 25 campioni di tempo, in modo da poterne calcolare la media e considerarla come il tempo effettivo di durata dell'impulso misurato. Una volta ottenute le coppie tempo-distanza si è effettuata la regressione lineare dei valori, con il risultato illustrato in figura 6.1.

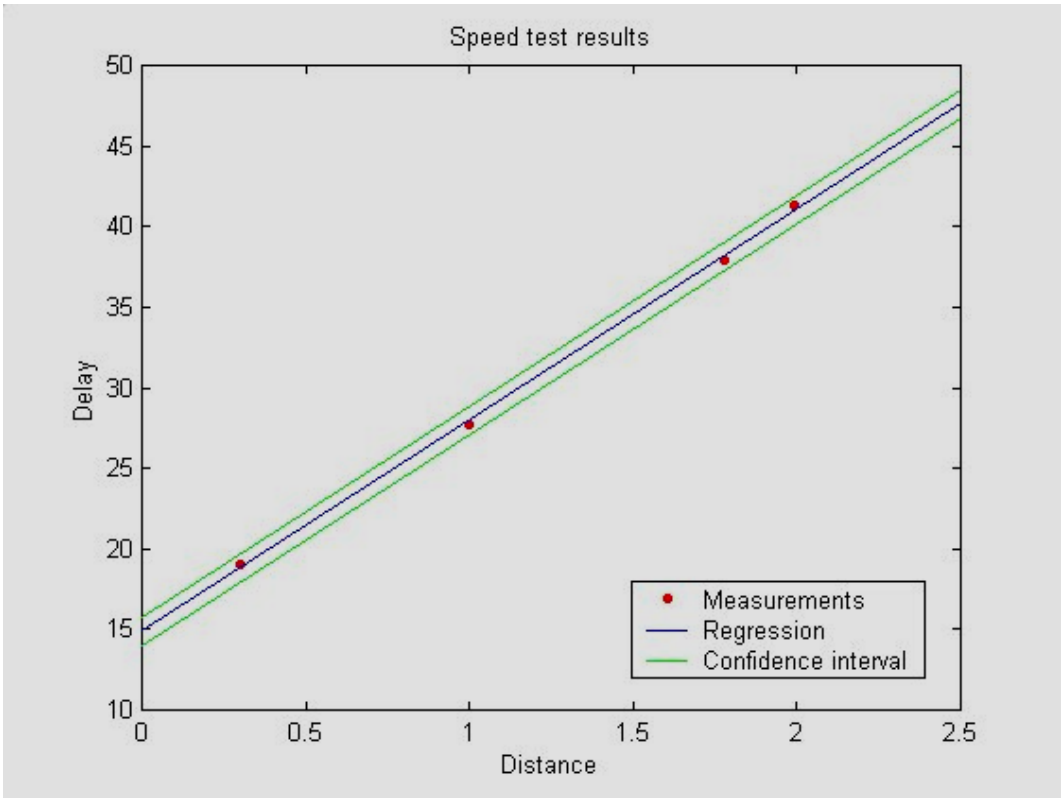


Figura 6.1. Risultato della regressione lineare.

Il calcolo della regressione lineare comporta l'ipotesi che il moto del proiettile sia lineare uniforme, con una fase di accelerazione sufficientemente breve che, tra l'altro, può essere inclusa nel ritardo di attivazione dell'elettrovalvola. La seconda assunzione, implicita al calcolo della regressione, è di considerare trascurabile anche l'attrito dell'aria sulla pallina.

Come si evince dall'analisi del grafico risultante, le ipotesi poste sono sufficientemente valide. Si nota infatti che la retta approssima sufficientemente bene i punti forniti.

Dalla regressione lineare si ricava direttamente il ritardo di attivazione dell'elettrovalvola, che è pari all'intersezione della retta con l'asse delle ordinate. La velocità della pallina sarà invece pari all'inverso della pendenza della retta, in

quanto il grafico riporta il tempo in ordinata e la distanza in ascissa. I valori trovati sono i seguenti:

velocità della pallina = 0.0764 m/ms
ritardo di attivazione della valvola = 14.8844 ms

L'intervallo di confidenza riportato in figura 6.1, invece, è

[14.0081, 15.7607] ms

6.2 Calibrazione

Per dare un'idea dei tempi di calcolo richiesti da una tipica sessione di taratura, vengono riportati i dati registrati durante uno dei test effettuati.

Il training set è costituito da una matrice (si veda il capitolo 5.4.3) 10×10 di training samples. In queste condizioni il modulo che genera i $10\times4=40$ punti sul perimetro, ovvero FP, impiega circa 6 minuti per terminare l'operazione (con un raggio di precisione nell'individuare il punto pari a 3 pixel). Il modulo GTS, che invece genera il training set vero e proprio, impiega circa 3 minuti e 20 secondi per generare i 100 punti previsti.

Per quanto riguarda la rete neurale, dotata di 12 neuroni nello strato nascosto e addestrata col training set precedentemente generato (quindi composto da 100 elementi), è stata caratterizzata dalle seguenti prestazioni:

Ciclo di apprendimento		Precisione ottenuta	
Epoche	Durata	Motore orizzontale	Motore verticale
240	2 sec	0.000729	0.001339
7400	2 min	0.000507	0.001124
40000	10 min	0.000051	0.00031

Dalle cifre riportate in tabella si può capire come la rete apprenda molto rapidamente nella primissima fase dell'addestramento, tanto che, nel giro di poche epoche ed in breve tempo, viene raggiunto l'ordine di grandezza del risultato finale. Si è comunque notato che, lasciando procedere l'addestramento fino a raggiungere un numero relativamente elevato di epoche si è in grado di migliorare l'errore globale di almeno un ordine di grandezza. Questo è un comportamento che si è registrato in tutte le sessioni di calibrazione della rete.

6.3 Test di sparo

Per testare l'efficacia di DEA nello sparare ad un bersaglio sono stati effettuati una serie di test, sia con target fermo che in movimento a diverse velocità. Nel caso di bersaglio mobile si è testato solo il caso di moto circolare uniforme, in modo da far permanere il bersaglio all'interno dell'area di visione per tutta la durata della fase di tracking.

I test sono stati eseguiti ponendo il dispositivo di puntamento a diverse distanze dal target, per saggiarne la precisione all'aumentare della distanza stessa. Le prove sono state fatte con un target di dimensioni fisse: un cerchio di circa 3 centimetri di diametro.

I test sono identificati da lettere progressive (A, B, ...). I bersagli corrispondenti sono stati conservati per toccare con mano il risultato delle prove, e sono stati etichettati con i relativi identificatori progressivi.

Per le prove a bersaglio fermo, utili per verificare la bontà del puntamento garantito dalla rete neurale e della calibrazione laser-cerbottana, viene anche riportata una mappa delle posizioni del target nel piano immagine, in modo da poter valutare se proprio la posizione è influenza la precisione.

6.3.1 Test A

I primi test effettuati sono stati condotti impostando i parametri come descritto in tabella 2. I risultati di questa sessione di test sono invece riportati in tabella 3. I valori riportati in grassetto indicano i colpi che hanno centrato il bersaglio (si ricordi che quest'ultimo ha raggio pari a 1.5 cm).

Caratteristiche fisiche	Distanza del target	150 cm
	Target in movimento	SI'
	Velocità	circa 3 sec/giro
	Raggio della traiettoria	20 cm
	Ritardo valvola	15 ms
	Velocità proiettile	0,764 ms
Rete neurale	Neuroni nascosti	
Filtro di Kalman	Rho	8.675.042
	Previsione in avanti	12 step
	Raggio di aggancio	3 pixel
	Numero di agganci	10

Tabella 2. Parametri del test A.

		Distanza dal centro del bersaglio [cm]		
Numero di sparo	1	2.4	2.2	2.4
	2	2	2.6	3.2
	3	2.6	**1.2**	3.3
	4	1.8	3.2	2.1
	5	2.3	4.7	**0.8**
	6	**1.5**	**0.7**	2.5
	7	**1.4**	2.3	
	8	-	**0.4**	

Tabella 3. Risultato del test A.

6.3.2 Test B

Il secondo test della serie è stato effettuato con i medesimi parametri in tabella 2, cambiando soltanto il valore dei ρ utilizzati nel filtro di Kalman. Questi ultimi sono stati posti pari a

$$\rho = 4.913.127$$

I risultati di questo test sono elencati in tabella 4.

		Distanza dal centro del bersaglio [cm]
Numero di sparo	1	2.3
	2	1.8
	3	1.8
	4	**0.2**
	5	2.4
	6	3.2
	7	**0.7**
	8	**1.3**

Tabella 4. Risultato del test B.

6.3.3. Test C

In questo esperimento il bersaglio è fermo. Tutti gli altri parametri sono come in tabella 2. In particolare la rete neurale non necessita nemmeno di essere riaddestrata (così come per il test B), in quanto non c'è alcuna variazione nella configurazione fisica del sistema o del piano di moto. I risultati dell'esperimento corrente sono riportati in tabella 5, mentre in figura 6.2 viene schematizzata la mappa delle posizioni del target in corrispondenza dei diversi colpi sparati.

		Distanza dal centro del bersaglio [cm]
Numero di sparo	1	**1**
	2	**1.4**
	3	2.3
	4	3.2
	5	1.8
	6	2.7
	7	3.5

Tabella 5. Risultato del test C.

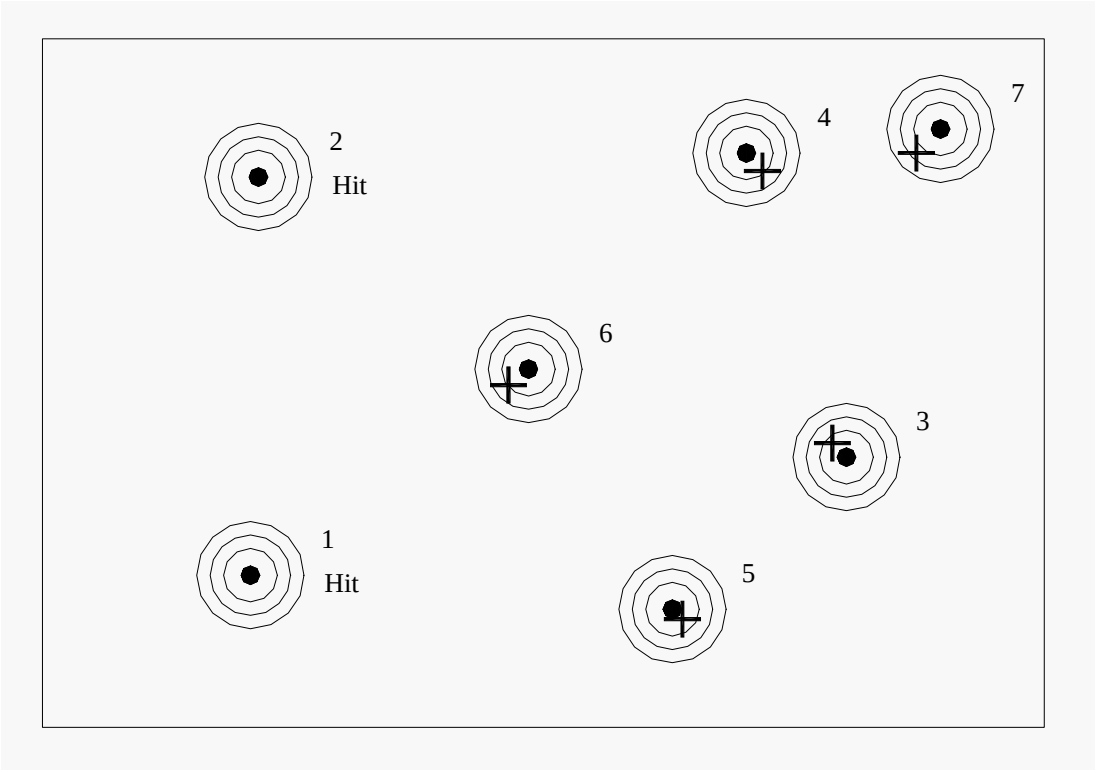


Figura 6.2. Mappa dei colpi per il test C.

Si noti anche che il valore del ρ utilizzato nel filtro di Kalman (comunque pari a quello riportato in tabella), non è fondamentale per il buon funzionamento

dell'esperimento: qualsiasi valore sufficientemente grande da filtrare il rumore dovuto, eventualmente, a problemi di luminosità, sarebbe accettabile.

6.3.4 Test D1

In questa prova il bersaglio è in movimento e la distanza e sempre pari a 150 cm, quindi non si ha necessità di ritarare la rete. In questo caso, però, viene aumentata la velocità fino a fargli impiegare circa 2 secondi per ciascun giro, con un incremento del 50% rispetto ai casi precedenti. I dati rilevanti, che variano rispetto al test A, sono elencati in tabella 6.

Caratteristiche fisiche	Distanza del target	150 cm
	Target in movimento	SI'
	Velocità	circa 2 sec/giro
Filtro di Kalman	Rho	892.518
	Previsione in avanti	12 step
	Raggio di aggancio	30 pixel

Tabella 6. Dati del test D1.

		Distanza dal centro del bersaglio [cm]
Numero di sparo	1	5.2
	2	3.8
	3	6.4
	4	**1.4**
	5	3.6
	6	5.5

Tabella 7. Risultato del test D1.

6.3.5 Test D2

I parametri di questa sessione di prove è invariata rispetto alla precedente, ma per tentare di migliorare la precisione si è lavorato sul raggio di aggancio e sul valore dei guadagni con cui si controlla il filtro di Kalman. In questo test i due valori sono:

Hook radius = 20

Rho = 91.825

I risultati del test sono riportati in tabella 8.

		Distanza dal centro del bersaglio [cm]
Numero di sparo	1	7
	2	4.6
	3	4.6
	4	5.2
	5	5.6
	6	6.2

Tabella 8. Risultato del test D2.

6.3.6 Test D3

Un ultimo test con gli stessi parametri dei punti precedenti, variando ulteriormente il raggio di aggancio e il ρ :

Hook radius = 20

Rho = 91.825

I risultati del test sono riportati in tabella 9.

		Distanza dal centro del bersaglio [cm]
Numero di sparo	1	3.6
	2	3.7
	3	4.5
	4	**1.1**
	5	6.3
	6	4
	7	5

Tabella 9. Risultato del test D3.

6.3.7 Test E1

Nei test della serie E viene aumentata la distanza del bersaglio, mentre la velocità del target viene diminuita. I dati completi sono riportati in tabella 10.

Caratteristiche fisiche	Distanza del target	240 cm
	Target in movimento	SI'
	Velocità	circa 3 sec/giro
Filtro di Kalman	Rho	892.518
	Previsione in avanti	12 step
	Raggio di aggancio	6 pixel
	Numero di agganci	5

Tabella 10. Parametri del test E1.

I risultati dei test alla distanza di 2.4 metri (come si osserverà anche dai risultati riportati nei prossimi paragrafi) ha confermato le ipotesi che il sistema perde di precisione all'aumentare della distanza. La causa principale di questo comportamento è da attribuire alle imprecisioni nel posizionamento dei servo, che non consentono quindi di ottenere un puntamento ottimale.

		Distanza dal centro del bersaglio [cm]
Numero di sparo	1	4.6
	2	2.5
	3	2.3
	4	2.7
	5	2.7
	6	4

Tabella 11. Risultato del test E1.

6.3.8 Test E2

Anche questo test è stato effettuato con gli stessi parametri del precedente, in particolare per quanto riguarda la distanza e la velocità del target. Anche i guadagni per il filtro di Kalman non sono variati. Si è pensato di ridurre soltanto il valore del raggio di aggancio, ponendo

Hook radius = 3

		Distanza dal centro del bersaglio [cm]
Numero di sparo	1	4.5
	2	**1.1**
	3	**1.3**
	4	4.6
	5	2.1
	6	6.6
	7	**1.1**
	8	3.8
	9	4.4

Tabella 12. Risultato del test E2.

I risultati riportati in tabella 12 confermano che, riducendo semplicemente il valore del raggio di aggancio del filtro, la precisione di sparo aumenta discretamente.

6.3.9 Test F1

Viene infine presentato il test di sparo a bersaglio fermo dalla distanza di 2.4 metri, che si dimostra utile per individuare in quali regioni dell'area di scansione il puntamento è più preciso e viceversa. Anche in questo caso viene fornita una mappa dei punti in cui è avvenuto lo sparo, e questa è riportata in figura 6.3.

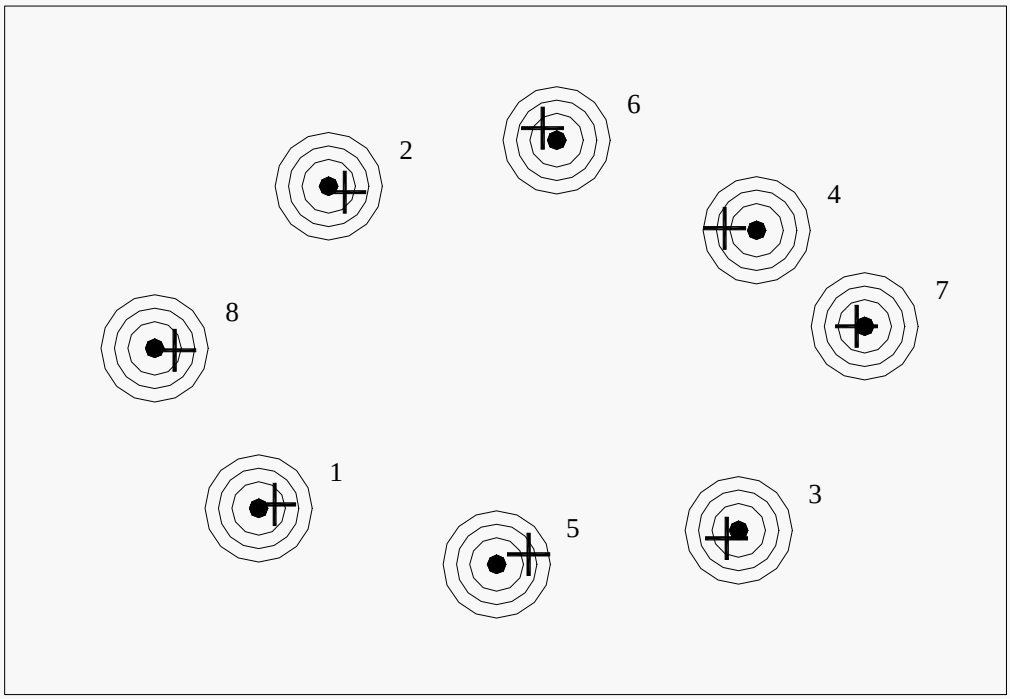


Figura 6.3. Mappa dei colpi per il test F1.

I risultati del test sono riportati in tabella 13.

		Distanza dal centro del bersaglio [cm]
Numero di sparo	1	2.1
	2	3.1
	3	2.6
	4	4.1
	5	3.6
	6	3.2
	7	1.7
	8	2.3

Tabella 13. Risultato del test E2.

6.4 Conclusioni

I test di sparo hanno dato buoni risultati sin dalle prime prove non documentate, cioè quelle che hanno permesso di calibrare con più cura i diversi parametri.

Per quanto riguarda le valutazioni sul sistema nel suo complesso, la prima conclusione a cui si è giunti, ancor prima di effettuare i test descritti ai paragrafi precedenti, è che i motori utilizzati, ovvero i servo da modellismo, sono un po' troppo imprecisi per poter garantire delle prestazioni soddisfacenti. I limiti dei motori si evidenziano quanto più la distanza dal target aumenta, poiché, intuitivamente, l'angolo minimo di pilotaggio, rimanendo invariato, causa un aumento della lunghezza del segmento spazzato dal movimento. Questo svantaggio si contrappone al netto vantaggio in termini di costo dei servomotori usati e alla facilità di utilizzo caratteristica di questi tipo di azionamenti.

Una soluzione sicuramente più efficiente (e più costosa) consiste nell'utilizzare dei motori passo-passo, possibilmente ridotti mediante delle puleggie, e montando sull'albero finale un encoder o un potenziometro, per tenere traccia della posizione angolare. Questo aumenterebbe notevolmente la precisione del puntamento.

Per quanto riguarda la rete neurale, il pregio della soluzione adottata consiste sicuramente nella velocità con la quale si ottiene l'associazione tra coordinate sull'immagine e valori di pilotaggio dei motori, una volta che la rete è addestrata. Inoltre, come illustrato al paragrafo relativo alle scelte di progetto, il paradigma neurale può essere esteso in modo relativamente semplice anche con l'aumento della complessità del sistema. L'unico difetto di questa scelta è che, una volta addestrata la rete, non esiste più feed-back tra ciò che si desidera e ciò che effettivamente viene fatto dalla rete: se la rete non approssima in modo sufficientemente accurato la relativa funzione, questo si ripercuote con delle imprecisioni nel puntamento, anche a bersaglio fermo. Una possibile soluzione consiste nell'effettuare un qualche tipo di addestramento online su una rete già precedentemente addestrata, magari ricorrendo a tecniche come il *reinforcement learning*.

La scelta di utilizzare il filtro di Kalman per la stima e la previsione della traiettoria è stata decisamente positiva: il filtro ha funzionato ottimamente in tutte le diverse condizioni nelle quali è stato testato. Si tenga presente che il modello utilizzato per il moto del bersaglio è assolutamente generale, in quanto si basa unicamente sui legami spazio-velocità-accelerazione e sulle derivate di ordine superiore di quest'ultima.

Complessivamente, quindi, è stato realizzato un sistema di tiro che funziona in modo abbastanza buono e, inoltre, si sono importate nel sistema real-time HARTIK una serie di funzionalità di sicuro interesse, anche per lo sviluppo di progetti futuri.

Appendice A

HARTIK

A.1 Cos'è HARTIK

HARTIK è un completo e potente sistema operativo real-time basato sul linguaggio di programmazione C. E' stato sviluppato presso la Scuola Superiore S. Anna di Pisa, con lo scopo di controllare, attraverso sistemi multisensoriali, complessi meccanismi robotici. Una accurata descrizione di HARTIK è possibile ottenerla consultando [3]. Il codice e il manuale del sistema possono essere scaricati dall'indirizzo

`http://hartik.sssup.it/`

Controllare un robot attraverso i suoi sensori significa gestire diversi processi periodici, come l'acquisizione dati e gli anelli di controllo in retroazione, che richiedono scadenze regolari di attivazione e altri tipi di vincoli, sia temporali sia non, in modo da garantire il corretto funzionamento del sistema. Inoltre si richiede l'esecuzione di attività aperiodiche, come la gestione di eccezioni di funzionamento o situazioni di errore. Infine, spesso, sistemi robotici complessi

necessitano di interagire con periferiche disponibili in ambiente distribuito su macchine eterogenee.

HARTIK è stato sviluppato per soddisfare le esigenze precedentemente esposte. Le caratteristiche peculiari del sistema sono le seguenti:

- gestione di processi periodici (con Earliest Deadline First) e aperiodici (con Total Bandwidth Server), con criticità hard, soft o non real-time;
- espressione flessibile di diversi tipi di vincoli temporali;
- creazione dinamica di qualsiasi tipo di task;
- efficiente protocollo di accesso a risorse condivise, garantito dallo Stack Resource Policy;
- garanzia on-line per la prevedibilità di task creati run-time;
- comunicazione non bloccante tra task, grazie ai Cyclic Asynchronous Buffer (CAB);
- strategie di rigetto e recupero dei processi per la gestione controllata dei sovraccarichi indipendente dalla strategia di scheduling mediante l'algoritmo Robust Earliest Deadline (RED);
- possibilità di girare su macchine di diversa architettura (su macchine Motorola MC 68030 con bus VME, Intel 80486 e Pentium con bus EISA o PCI).

Nei paragrafi a seguire verranno elencate le principali feature di HARTIK utilizzate per il funzionamento di DEA. In particolare verranno trattate le primitive di creazione e gestione dei task, l'accesso al real-time clock, l'utilizzo dei CAB e del protocollo SRP, la gestione di tastiera e output grafico.

Tutti gli argomenti che riguardano i sistemi real-time trattati nel progetto sono trattati in modo chiaro ed esauriente dal punto di vista teorico in [2].

A.2 Il real-time clock

Il real-time clock di HARTIK è la scala temporale sulla quale vengono basati tutte le valutazioni di natura temporale del sistema, come il controllo sul tempo massimo di esecuzione e quello sullo sfondamento delle deadline. L'unità minima della scala è chiamata *tick*, e può essere impostata pari a 1 millisecondo o a 1 microsecondo al momento dell'avvio di HARTIK. I parametri relativi ai processi vengono specificati come multipli del tick.

Durante l'esecuzione dei programmi può essere necessario accedere al valore del real-time clock; ciò è possibile facendo uso della primitiva **sys_clock()**, che restituisce il numero di tick trascorsi dall'avvio del sistema.

A.3 Avvio del sistema

L'avvio di HARTIK avviene richiamando una apposita primitiva di inizializzazione, specificando i parametri necessari a configurare l'ambiente secondo le necessità. Anche le diverse periferiche del sistema (mouse, tastiera, video in modalità grafica, etc.) devono essere inizializzate per poter essere usate correttamente. Di seguito vengono elencate le primitive da richiamare per attivare HARTIK e i vari sottosistemi.

- **sys_init()** avvia il sistema operativo real-time HARTIK;
- **sys_end()** esegue lo shut-down di HARTIK; dopo questa istruzione il programma può continuare, in quanto l'istruzione non ne causa la terminazione;
- **keyb_init()** inizializza il task che si occupa di gestire la tastiera; utilizzare le primitive di lettura da tastiera senza aver prima richiamati questa primitiva causa il blocco della macchina;
- **grx_init()** inizializza il motore grafico di HARTIK;
- **grx_close()** ritorna al modo testo.

A.4 Creazione dei task

La creazione dei task in HARTIK avviene mediante la primitiva **task_create()**. Questa accetta una serie di parametri che identificano il tipo di processo e le sue caratteristiche. La sintassi completa della funzione è

PID task_create(name, task_ptr, criticity, periodicity, period, task_model);

il significato dei singoli parametri è il seguente

- **name** è una stringa di caratteri contenete il nome del task, ed è usato un fase di debugging o nella visualizzazione di messaggi d'errore;
- **task_ptr** è un puntatore alla funzione che contiene il codice del processo;
- **criticity** può assumere i valori HARD, SOFT o NRT (non real-time);
- **periodicity** può essere impostata a PERIODIC oppure APERIODIC;
- **period** è il periodo per un task periodico, oppure il tempo minimo di interarrivo per un task aperiodico;
- **task_model** è una struttura contenente, tra le altre cose, il computation time del task e le eventuali risorse utilizzate;

La funzione restituisce un PID (Process IDentifier) che individua univocamente il task creato. In base alla combinazione dei parametri criticity e periodicity, si possono individuare 5 classi di processi, in base alla loro importanza rispetto ai vincoli temporali:

- **HARD + PERIODIC** individua un task di tipo HARD, avente deadline critica coincidente col periodo. E' un processo garantito al momento della creazione e il superamento della deadline per questa categoria di task genera un errore di *time overflow*.
- **HARD + APERIODIC** sono task di tipo SPORADIC; ha una deadline coincidente col minimo tempo di interarrivo ed è garantito dal sistema al momento della creazione, in modo che il superamento di una deadline causi un errore generale.

- **SOFT + APERIODIC** sono task di classe APERIODIC, cioè task ad attivazione non regolare e deadline non critica. Non vengono garantiti dal sistema e sono schedulati in base alla deadline.
- **SOFT + PERIODIC** sono task di classe SOFT, cioè ad attivazione regolare ma con deadline non critica; essi non sono garantiti dal sistema e vengono schedulati in base alle rispettive deadline.
- **NRT,** sia **PERODIC** che **APERIODIC**, sono processo Non Real-Time senza vincoli temporali. Tali processi vengono schedulati in base a priorità fisse assegnate al momento della creazione.

Una volta che un task è stato creato esso va attivato mediante la primitiva task_activate(task_PID), in modo che il processo possa eseguire il suo codice. Una volta attivato sono disponibili alcune primitive per la gestione dell'attività del task. Brevemente:

- **task_endcycle()** termina una istanza del task, sia periodico che aperiodico, restituendo il controllo al sistema;
- **task_sleep()** permette di sospendere a tempo indeterminato l'esecuzione di un altro processo, che riprende soltanto quando viene richiamata la primitiva **task_activate()**;
- **task_abort()** permette ad un processo di auto-sospendersi;

A.5 Accesso ai file

Le primitive di accesso ai file sono simili, ma con nomi, sintassi e funzionamento leggermente diverse, a quelle del C standard. L'aspetto più importante di questo insieme di primitive è che possono essere utilizzate solo prima di avere avviato HARTIK oppure dopo aver terminato l'esecuzione del kernel. In sostanza, nessun task può accedere ai file durante l'esecuzione (cosa che toglierebbe prevedibilità all'intero sistema), ma si può utilizzare l'I/O da e

verso i file per inizializzare il sistema e/o salvare i risultati di una elaborazione. L'elenco completo delle funzioni di acceso è:

- **DOS_FILE DOS_fopen(file_name, access_mode)**, in base al valore di **access_mode** apre il file **file_name** in lettura ("r"), scrittura ("w") o in append ("a"), ritornando un puntatore a una struttura di tipo **DOS_FILE**;
- **DOS_fclose(DOS_FILE)** chiude l'accesso al file specificato;
- **DWORD DOS_fread(void *buf, DWORD size, DWORD num, DOS_FILE *f)** legge un blocco massimo di **num** byte dal file **f**, depositandoli nel buffer **buf**, che ha dimensione pari a **size**. La funzione ritorna il numero di byte effettivamente letti;
- **DWORD DOS_getline(void *abuf, DWORD count, int ch, DOS_FILE *f)** legge una linea di caratteri dal file **f**, depositandoli nel buffer puntato da **abuf**. La funzione legge un massimo di **count** caratteri, oppure fino al raggiungimento del carattere **ch**. Ritorna il numero di caratteri effettivamente letti.

A.6 Risorse condivise

HARTIK mette a disposizione diverse tecniche per gestire la condivisione di risorse tra processi. Una sezione di codice che utilizza una rsorsa condivisa è detta *sezione critica*.

Innanzitutto dispone del meccanismo semaforico classico, che però non è efficiente e può dar luogo a problemi alquanto gravi come l'inversione di priorità. Non è dunque adatto ad essere utilizzato con processi di criticità HARD, in quanto potrebbe causare lo sfondamento di una deadline critica.

Il protocollo SRP (Stack Resource Policy) supera tutti i problemi che derivano dall'uso dei semafori classici, garantendo migliori prestazioni globali del sistema pur mantenendo la stessa semplicità di utilizzo dei normali semafori.

Un secondo ed efficiente meccanismo per lo scambio di informazioni tra processi è dato dall'uso dei CAB (Cyclic Asynchronous Buffer).

Nei prossimi paragrafi vengono descritte le due tecniche dal punto di vista dell'utilizzatore, tralasciando i dettagli teorici.

A.6.1 Stack Resource Policy

Gli SRP devono essere usati dichiarando una costante che identifica la risorsa, e racchiudendo entro una coppia di primitive apposite le istruzioni che fanno uso della risorsa; è però necessario dichiararlo preventivamente, al momento della creazione del task.

Le funzioni che sono state predisposte per l'utilizzo del protocollo sono le seguenti:

- **task_use_res(model, resource, ms)** permette di indicare che il task in via di creazione, descritto da **model** utilizza la risorsa di indice **resource**, e che la sezione critica relativa alla risorsa ha una durata massima pari a **ms** microsecondi;
- **void srp_wait(resource)** apre la sezione critica relativa alla risorsa **resource**;
- **void srp_signal(resource)** chiude la sezione critica relativa alla risorsa **resource**.

A questo punto vorrei sottolineare un aspetto che non viene affatto evidenziato nel manuale utente di HARTIK, e che può causare notevoli noie e la perdita di molto tempo per il debugging.

L'utilizzo di una risorsa avviene in genere dichiarando una costante, che funge da indice all'interno di un vettore nel kernel di HARTIK, vettore che serve a tenere traccia dell'uso della risorsa da parte dei task. Ora, il vettore è dichiarato staticamente, ed ha una dimensione di 20 elementi (dipende comunque dal valore

impostato al momento della compilazione del kernel), quindi, dichiarando un identificatore di risorsa maggiore di 19 si ottiene una sicura scrittura oltre i limiti del vettore, in spazi di memoria assegnati presumibilmente ad altre variabili. Questo è un errore alquanto subdolo, poichè non è affatto facile da rilevare, essendo dipendente da ciò che si trova nella locazione di memoria che viene sovrascritta errorneamente. Si può andare dalla stanpa di un carattere errato sullo schermo al blocco della macchina.

A.6.2 Cyclic Asynchronous Buffer

La comunicazione mediante CAB, che è di tipo 1 a molti, permette ai processi di essere asincroni tra di loro: il task che deve scrivere nel CAB lo fa senza preoccuparsi di eventuali letture concorrenti e cancellando il valore precedentemente memorizzato, mentre un task che deve leggere dal CAB avrà sempre a disposizione l'ultimo valore che vi è stato scritto.

L'utilizzo dei CAB avviene mediante le seguenti primitive:

- **CAB cab_create(char *name, int dim_mes, BYTE num_mes)** alloca spazio in memoria per una variabile di tipo **CAB**. **name** è una stringa che identifica il CAB ed utile in fase di debugging e **dim_mes** specifica la dimensione in byte del messaggio che deve essere contenuto nel CAB. **num_mes** indica quanti task possono leggere contemporaneamente dallo stesso CAB, in quanto il gestore del protocollo alloca memoria per creare l'esatto numero di copie del messaggio e garantire, quindi, che tutti i processi possano leggere in parallelo la propria copia;
- **char *cab_reserve(CAB id)** serve a ottenere il puntatore alla locazione di memoria ove deve essere scritto il messaggio da parte del task che aggiorna il CAB;

- **int cab_putmes(CAB id, char *pbuf)** scrive il messaggio puntato da **pbuf** nel CAB **id**.
- **char *cab_getmes(CAB id)** è la primitiva che deve essere utilizzata dal task che vuole leggere il contenuto del CAB identificato da **id**. Viene restituito un puntatore all'area di memoria che contiene il messaggio;
- **int cab_unget(CAB id, char *pun_mes)** serve a comunicare al gestore del protocollo che è terminata la fase di lettura del CAB **id,** e che la relativa area di memoria può essere rilasciata;
- **void cab_delete(CAB id)** elimina il CAB **id**.

A.7 Grafica

Dopo aver inizializzato il motore grafico come illustrato al paragrafo A.1, si possono utilizzare le comuni primitive di tracciamento per l'output grafico. Si tenga presente che il video grafico è mappato in modo che la coordinata (0, 0) coincida con l'angolo in alto a sinistra del monitor. Ecco una panoramica di tali primitive:

- **grx_getpixel(x, y)** ritorna il colore del punto alle coordinate (**x**, **y**);
- **grx_putpixel(x, y, col)** scrive un punto di colore **col** alle coordinate (**x**, **y**);
- **grx_line(x1, y1, x2, y2, col)** disegna una linea di colore **col** dal punto (**x1**, **y1**) al punto (**x2**, **y2**);
- **grx_rect(x1, y1, x2, y2, col)** disegna un rettangolo di colore **col** che ha come spigolo superiore sinistro il punto (**x1**, **y1**) e come angolo inferiore destro il punto (**x2**, **y2**);
- **grx_box(x1, y1, x2, y2, col)** disegna un rettangolo che ha come spigolo superiore sinistro il punto (**x1**, **y1**) e come angolo inferiore destro il punto (**x2**, **y2**) riempiendolo con il colore **col**;

- **grx_circle(x, y, rad, col)** disegna un cerchio di colore **col** e raggio **rad** (misurati in pixel) centrato nel punto (**x**, **y**);
- **grx_disc(x, y, rad, col)** disegna un disco, cioè un cerchio pieno, di colore **col** e raggio **rad** (misurati in pixel) centrato nel punto (**x**, **y**);
- **grx_getimage(x1, y1, x2, y2, BYTE *img)** cattura una immagine dallo schermo e la pone nel buffer puntato da **img**. L'immagine è contenuta nel rettangolo che ha come spigolo superiore sinistro il punto (**x1**, **y1**) e come angolo inferiore destro il punto (**x2**, **y2**);
- **grx_putimage(x1, y1, x2, y2, BYTE *img)** stampa sullo schermo l'immagine contenuta nell'area di memoria puntata da **img** nel rettangolo che ha come spigolo superiore sinistro il punto (**x1**, **y1**) e come angolo inferiore destro il punto (**x2**, **y2**);
- **grx_text(str, x, y, fg, bg)** visualizza la stringa di testo **str** alle coordinate **x** e **y** con i colori **fg** per il testo e **bg** per lo sfondo.

Appendice B

Librerie di funzioni

Di seguito vengono elencate le funzioni sviluppate per realizzare DEA, con una breve descrizione del funzionamento di ciascuna di esse. Vengono raggruppate in base ai moduli di appartenenza.

B.1 elabor

- **TPixel**

  tipo di dato che corrisponde ad un pixel di una immagine a 256 livelli di grigio, cioè il tipo di immagine fornita dal frame-grabber di DEA.

- **TPixCoord**

  TRealCoord

  definiscono due tipi di dato molto usati nel progetto, che incapsulano l'oggetto "punto" su una immagine. Le variabili membro sono **x** e **y**, le coordinate del punto. In TPixCoord sono di tipo intero, mentre il TRealCoord sono float.

- **void load_ld(void)**

void save_ld(void)

rispettivamente carica e salva i valori caratteristici del puntatore laser catturato dall'immagine della telecamera.

- **void load_od(void)**

  void save_od(void)

  rispettivamente carica e salva i valori caratteristici dell'oggetto da colpire, così come viene catturato dall'immagine della telecamera.

- **void load_isl(void)**

  void save_isl(void)

  carica e salva, rispettivamente, i limiti della scansione pixel per pixel dell'immagine fornita dalla telecamera.

- **void init_hartik(void)**

  avvia HARTIK, inizializzando il sistema e la tastiera. Il setup del motore grafico viene effettuato da una funzione apposita.

- **void shut_down_sys(char *msg)**

  procedura standard di arresto del sistema. Questa funzione controlla appositi flag e termina HARTIK solo se questo è stato preventivamente avviato, così come per il motore grafico. Inoltre toglie coppia ai motori, spegne il laser e chiude l'elettrovalvola. Terminate queste operazioni visualizza il messaggio contenuto nel parametro **msg**. Dopo la chiamata di questa funzione sono possibili altre elaborazioni, poiché il programma non viene terminato.

- **void close_all(char *msg)**

  Dopo aver arrestato il sistema chiamando **shut_down_sys()**, termina definitivamente il programma. Viene usata, in genere, in caso di errore.

- **void set_laser_spot_width(int w)**

  void set_laser_spot_height(int h)

  void set_laser_threshold(int th)

funzioni che modificano la dimensione e il livello di luminosità del puntatore laser nell'immagine.

- **inline DWORD get_laser_spot_width(void)**

  inline DWORD get_laser_spot_height(void)

  inline int get_laser_threshold(void)

  restituiscono la dimensione in pixel e il livello di luminosità del punto laser sull'immagine.

- **inline int get_img_top_border(void)**

  inline int get_img_bottom_border(void)

  inline int get_img_left_border(void)

  inline int get_img_right_border(void)

  ritornano i valori dei quattro limiti di scansione dell'immagine, rispettivamente superiore, inferiore, sinistro e destro.

- **inline int get_pix_threshold(void)**

  ritorna il livello di luminosità dei punti che appartengono all'oggetto da colpire.

- **void set_pix_threshold(int th)**

  imposta il livello di luminosità dei punti che appartengono all'oggetto da colpire; funzione utilizzata in fase di calibrazione del sistema.

- **char find_spot(TPixCoord *data, TPixel *in_frame)**

- **char find_laser_spot(TPixCoord *data, TPixel *in_frame)**

  la funzione scandisce l'immagine puntata da in_frame alla ricerca del punto laser. I limiti di scansione sono quelli impostati con la chiamata della funzione load_isl(). Il passo di scansione viene calibrato in base alla dimensione del puntatore. I valori di ritorno è un flag che vale 1 se il puntatore è stato trovato (0 altrimenti), nel qual caso la struttura data conterrà le coordinate del baricentro del punto.

- **void get_last_pos(TPixCoord *ps)**

  void get_c_last_pos(int *x, int *y)

ritornano i medesimi valori, cioè le coordinate nell'immagine registrate nell'ultimo rilevamento andato a buon fine. Le funzioni sono due per comodità: una ritorna le singole componenti in due variabili diverse, mentre l'altra in una struttura di tipo TPixCoord.

B.2 general

- **float frand(float xmi, float xma)**

  ritorna un numero pseudo-casuale compreso tra xmi e xma.

  - **long int _abs(long int n)**

    ritorna il valore assoluto di n. Da utilizzare con numeri interi.

  - **double _fabs(double n)**

    ritorna il valore assoluto di n. Da utilizzare con numeri reali.

  - **int inbox(int xc, int yc, int x1, int y1, int x2, int y2)**

    restituisce 1 se il punto (xc, yc) è contenuto nel rettangolo che ha (x1, y1) come angolo alto a sinistra, e (x2, y2) come angolo basso a destra. Altrimenti ritorna 0.

  - **int same(TPixCoord p1, TPixCoord p2)**

    restituisce 1 se se i due punti coincidono, ovvero se le coordinate corrispondenti sono uguali tra loro.

  - **void next_token(char *buf, unsigned int long *i, char *token)**

    è una funzione di utilità che scandisce il buffer di testo puntato da *buf* partendo alla locazione di offset pari a *i*. Da questo punto parte a ricercare il primo token che incontra, e lo ritorna nella stringa *token*. Viene considerato token un insieme di caratteri ASCII isolato tra dei separatori, che sono lo spazio, le tabulazioni e gli '\n'.

- **float rescale(float min1, float max1, float min2, float max2, float n)**

  ritorna il valore di n riscalato dall'intervallo [min1, max1] per portarlo nell'intervallo [min2, max2].

B.3 ioni

Le funzioni sono elencate di seguito. Per brevità sono state elencate solo le funzioni di accesso al timer general-purpose G0: le funzioni che comandano G1 sono completamente identiche, se non che il nome prevede un 1 invece dello 0.

- **void load_md(void)**
  void save_md(void)
  esegue l'I/O su file per leggere o scrivere i parametri relativi al pilotaggio dei motori. Il caricamento di questi parametri è fondamentale per l'utilizzo delle funzioni di controllo dei motori.
- **inline DWORD get_hmin(void)**
  inline DWORD get_hmax(void)
  inline DWORD get_vmin(void)
  inline DWORD get_vmax(void)
  ritornano i valori che delimitano il range di pilotaggio dei motori, limitato dalle caratteristiche costruttive dei motori stessi.
- **inline DWORD get_hhome(void)**
  inline DWORD get_vhome(void)
  ritornano il valore delle variabili di controllo dei motori impostate come valori di riposo per i servo.

- **int ni_board_ready(void)**
  restituisce 1 se è stato eseguito il setup della scheda PCI-6025E, e quindi questa è pronta ad eseguire le primitive di controllo. Ritorna 0 altrimenti.
- **int is_hmotor_moving(void),**
  int is_vmotor_moving(void),
  int is_motors_moving(void)
  ritornano 1 se, rispettivamente, il motore orizzontale, quello verticale oppure entrambi sono in moto, oppure 0 se gli stessi sono fermi.
- **DWORD get_hmotor_trig(void)**

DWORD get_vmotor_trig(void)

ritornano i valori attualmente forniti al driver per pilotare i motori.

- **DWORD get_hmot(void)**

  DWORD get_vmot(void)

  ritornano i valori dei segnali di pilotaggio effettivamente inviati dal driver ai motori, che possono essere diversi da quelli desiderati, in quanto i motori possono essere in movimento.

- **void trig_hmotor(DWORD value)**

  void trig_vmotor(DWORD value)

  imposta un nuovo valore per il controllo dei motori. value indica la durata in millisecondi dell'impulso PWM che guida i servo.

- **void motor_switch_on(int m)**

  accende il motore specificato da m (che può valere HMOTOR oppure VMOTOR). Si presti attenzione al fatto che questa funzione non agisce sull'alimentazione del motore, ma abilita soltanto il treno di impulsi.

- **void motor_switch_off(int m)**

  spegne il motore specificato da m (che può valere HMOTOR oppure VMOTOR). Come la precedente, non toglie alimentazione al motore, ma interrompe il treno di impulsi PWM che pilotano il motore, in modo che questo, di fatto, non applichi più alcuna coppia al carico.

- **void motor_goto_home(void)**

  porta i motori in posizione di riposo.

- **int motor_power_status(int m)**

  ritorna lo stato attuale del segnale di gate per il motore m, che indica se il treno di impulsi viene correntemente inviato al motore.

- **void laser_light(void)**

  accende il puntatore laser, fornendo tensione.

- **void laser_turnoff(void)**

  spegne il puntatore laser, togliendo alimentazione.

- **void valve_open(void)**

  apre la valvola dell'aria.
- **void valve_close(void)**

  chiude la valvola dell'aria.
- **void cannon_fire(void)**

  genera un impulso nell'apertura della valvola pneumatica: la apre e la chiude dopo un certo delay di ritardo, effettuando di fatto uno "sparo".
- **int laser_status(void)**

  int valve_status(void)

  ritorna lo stato di attivazione del laser e della valvola. Le funzioni valgono 1 se il laser è acceso o se la valvola è aperta, 0 altrimenti.
- **void go_goto_home_MOD(TIME periodo)**

  esegue il setup del sottosistema che si occupa di riportare il motore nella posizione di riposo (home).
- **void go_move_MOD(TIME periodo)**

  esegue l'avviamento del sottosistema che gestisce il movimento dei due motori.
- **void init_cannon(void)**

  crea il processo che effettua l'apertura e chiusura temporizzata della valvola pneumatica, simulando lo sparo.

B.4 kc

- **void wait_for_key(const char check_mode, BYTE kw1, BYTE kw2)**

  esegue il loop di attesa della pressione di un tasto, dal quale si esce solo con la pressione di uno dei due tasti il cui codice ASCII è **kw1** o **kw2**. Altri tasti premuti vengono elaborati in base al valore del parametro **check_mode**.
- **void unblock_key_waiting(void)**

permette di sbloccare il ciclo di attesa via software, senza premere alcun tasto. E' utile per terminare i moduli esecutivi in modo automatico senza richiedere l'intervento dell'utente.

B.5 neural

neural.h e *neural.c* contengono il codice relativo alla gestione dei una rete neurale error back-propagation. Dati i particolari accorgimenti utilizzati per sviluppare la libreria, questa permette di utilizzare più di una rete neurale all'interno del medesimo programma, ciascuna completamente indipendente dalle altre. La struttura e le tecniche implementative della libreria sono spiegate in dettaglio al paragrafo 5.6.5.

- **MAX_N_L1**
  MAX_N_L2
  MAX_N_L3
  dimensioni massime, in numero di neuroni, dei 3 layer standard della rete neurale. Questi valori sono utilizzati per dimensionare gli array statici che contengono i pesi e le soglie della rete.
- **MAX_N_TS**
  è la dimensione massima del generico training set: una struttura di tipo training set non può contenere più di *MAX_N_TS* training sample.
- **DEFAULT_NET**
- **DEFAULT_TSET**
  costanti usate per inizializzare staticamente le strutture di tipo *TSET* e *NET*. Vanno assegnate in fase di dichiarazione delle strutture stesse.
- **TTrainingSample**
  struttura che incapsula le caratteristiche del singolo training sample, il componente minimo di un training set, validation set, o test set. Il

training sample è composto dai valori di ingresso e i corrispondenti valori di uscita da utilizzare per addestrare la rete.

- **TSET**

  struttura che contiene i dati relativi ad un training set. Tali dati sono i seguenti:

  - **int n_ts**

    numero di training sample contenuti nell'insieme;

  - **int tset_initialized**

    flag che indica se il training set è stato inizializzato con una procedura standard della libreria;

  - **TTrainingSample s[MAX_N_TS]**

    vettore contenente i training training samples;

  - **float min_in_range**

    float max_in_range

    float min_out_range

    float max_out_range

    valori di scala degli input e output values: vengono usati per mappare nel range [0, 1] gli ingressi e le uscite alla rete neurale in fase di training; sono inoltre indispensabili per l'utilizzo della rete, in quanto mappano i valori di I/O non normalizzati in ingresso e riscalano i valori di uscita nel range corretto, congruente con il mapping usato in fase di addestramento;

- **NET**

  struttura che incapsula i dati di una generica rete neurale back-propagation a 3 strati di neuroni. Tutti i vettori sono sovradimensionati: in effetti spesso non vengono completamente riempiti di valori significativi, tranne nel caso in cui si utilizzino il massimo numero di neuroni per ciascun layer. I dati interni alla struttura sono dunque i seguenti:

- **net_initialized**

  flag che indica se la rete è stata inizializzata da una delle funzioni di libreria. I pesi possono essere generati a caso per un successivi apprendimento oppure possono essere caricati da file per utilizzare una rete precedentemente addestrata;
- **N_L1**

  N_L2

  N_L3

  numero di neuroni effettivamente allocati per ciascun layer (L1, L2 e L3 sono, rispettivamente, il layer di input, l'hidden layer e il layer di output);
- **MINW**

  MAXW

  valore di soglia per i valori di uscita dai singoli neuroni. Sostanzialmente si tratta di riscalare la sigmoide in un intervallo diverso da [0, 1] (per esempio [0.1, 0.9]), cosa che può risultare particolarmente utile in diversi campi applicativi;
- **epoch**

  numero di epoche d'addestramento attualmente trascorse. E' una variabile che, in fase di training, è utile a riprendere l'addestramento dall'epoca raggiunta;
- **eps**

  precisione che si intende raggiungere con l'apprendimento. Una volta raggiunto questo valore limite, la funzione che implementa il training restituisce il controllo al programma principale;
- **epoch_per_iter**

  numero di epoche che si desidera iterare per l'apprendimento della rete. La funzione di training ritorna al programma chiamante ogni *epoch_per_iter*. Il programma principale può eventualmente avviare un altro ciclo di apprendimento;

- **l1[MAX_N_L1]**
  l2[MAX_N_L2]
  l3[MAX_N_L3]
  vettori che contengono, sia in fase di apprendimento che durante l'utilizzo della rete, le uscite correnti dei neuroni di input, nascosti e di output rispettivamente;
- **outd[MAX_N_L3]**
  vettore utilizzato soltanto in fase di training; contiene le uscite desiderate in corrispondenza degli ingressi attualmente forniti alla rete;
- **w2[MAX_N_L2][MAX_N_L1]**
  w3[MAX_N_L3][MAX_N_L2]
  vettori contenenti i pesi dell'hidden layer e del layer di output;
- **th2[MAX_N_L2]**
  th3[MAX_N_L3]
  soglie dei neuroni nascosti e di output;
- **dw2[MAX_N_L2][MAX_N_L1]**
  dw3[MAX_N_L3][MAX_N_L2]
  dth2[MAX_N_L2]
  dth3[MAX_N_L3]
  vettori contenenti gli incrementi dei pesi e delle soglie dei neuroni dello strato nascosto e di quello di output calcolati in fase di aggiustamento di tali grandezze, operazione indispensabile per il training della rete;
- **err2[MAX_N_L2]**
  err3[MAX_N_L3]
  errori nel layer nascosto e di output calcolati in fase di addestramento, sulla base dei quali vengono modificati i pesi dei neuroni;
- **eta**

learning rate utilizzato attualmente per l'apprendimento. Il valore di *eta* potrebbe cambiare dinamicamente durante l'apprendimento per ottimizzarne l'andamento;

- **momentum**

  valore del momento utilizzato in fase di addestramento. Anche questo valore, come quello del learning rate, può essere aggiustato dinamicamente in fase di training per velocizzare e/o migliorare l'apprendimento;

- **float sigmoid(float x, float beta)**

  calcola la funzione sigmoide di *x* con coefficiente *beta*, ovvero:

$$f(x) = \frac{1}{1 + e^{-\beta \cdot x}}$$

- **int init_tset(TSET *t, int n, float min_in, float max_in, float min_out, float max_out)**

  inizializza una struttura di tipo *TSET*, ritornando 0 se tutto è stato eseguito correttamente oppure –1 in caso di errore. Il significato dei parametri è il seguente:

  - **n**

    numero di elementi contenuti nell'insieme;

  - **min_in**, **max_in**, **min_out**, **max_out**

    valori utilizzati per scalare opportunamente gli ingressi e le uscite della rete;

- **int split_tset(TSET *inset, TSET *ts, TSET *vs)**

  suddivide l'insieme puntato da *inset* nei sottoinsiemi *ts* e *vs*, che devono precedentemente essere stati inizializzati per impostare il numero di elementi che si desidera inserirvi. Questa funzione è utile per creare degli insiemi di training, di validazione e di test a partire da un unico insieme di partenza, che può essere, ad esempio, letto da file. La funzione ritorna 0 in caso di successo, mentre ritorna –1 nel caso *inset* non contenga elementi sufficienti per riempire *ts* e *vs*.

- **void reset_tset(TSET *t)**

  azzera il contenuto di un training set;
- **void mix_ts(TSET *t)**

  mescola casualmente i training samples contenuti nella struttura *t*. Per training set ordinati è utile effettuare questa operazione, poiché un ordine troppo rigoroso potrebbe influire negativamente sul processo di apprendimento;
- **float tset_error(NET *n, TSET *t)**

  calcola l'errore quadratico medio complessivo della rete *n* sul training set *t*. Ritorna –1 se la rete o il training set non sono correttamente inizializzati;
- **int add_abs_noise(TSET *t, float var, int n)**

  aggiunge del rumore pseudo-casuale con ampiezza massima pari a *var* ai primi *n* valori di uscita del training set puntato da *t*. Ritorna 0 in caso di successo, oppure –1 in caso di errore.
- **int load_tset(TSET *t, char *fname)**

  inizializza il training set *t* leggendo i valori nel file *fname*. Restituisce 0 nel caso in cui l'operazione sia stata eseguita correttamente, altrimenti ritorna –1. Il file contenente il training set deve avere la struttura descritta paragrafo 5.3.10.
- **int init_net(NET *n, int n1, int n2, int n3,**

  float eta0, float m0, float minw, float maxw)

  inizializza la rete neurale *n* con i seguenti parametri:
  - **n** è il puntatore alla struttura che contiene la rete neurale da inizializzare;
  - **n1**, **n2** e **n3** specificano il numero di neuroni nei diversi strati della rete, rispettivamento quello di input, nascosto e quello di output;
  - **eta0** e **m0** sono i valori iniziali del learning rate e del momento, usati nella fase di training della rete;

- **minw** e **maxw** indicano i limiti del range in cui vanno rimappati dei valori di input e di output.

- **int load_net(NET *n, char *fname)**
  carica una rete neurale nella struttura puntata da **n**. I parametri per la lettura sono contenuti nel file **fname**, il quale deve avere la struttura descritta al paragrafo 5.3.11.
- **void reset_net(NET *n)**
  inizializza la rete neurale **n**, scrivendo i valori di default nelle variabili della struttura.
- **void set_input(NET *n, float *fin, float min, float max)**
  imposta i valori di ingresso alla rete neurale **n**. Gli input vengono presi dalla zona di memoria puntata da **fin**, che deve essere un vettore di dimensione appropriata, tale che contenga un numero sufficiente di valori per l'inizializzazione. **min** e **max** indicano l'intervallo delle variabili di input prima dello scaling.
- **void set_ts_input(NET *n, TSET *t, int which_ts)**
  pone in ingresso alla rete neurale puntata da **n** i valori presi dal training set puntato da **t**. Il training sample desiderato viene identificato dall'indice **which_ts**.
- **void set_desired_out(NET *n, float *fout, float min, float max)**
  fornisce le uscite desiderate alla rete neurale **n** nella fase di addestramento. I dati vengono prelevati dalla locazione di memoria puntata da **fout**, i quali vengono scalati dal range [**min**, **max**] al range di funzionamento della rete.
- **void get_output(NET *n, float *nums, float min, float max)**
  preleva le uscite della rete **n** ponendole nella zona di memoria puntata da **nums**. I valori sono scalati dall'intervallo di lavoro della rete a [**min**, **max**].
- **void compute_out(NET *n)**

calcola l'uscita della rete neurale **n** a partire degli attuali valori di ingressi caricati nella struttura.

- **void compute_err(NET *n)**
  calcola l'errore tra l'uscita corrente e l'uscita desiderata, in corrispondenza di una stessa combinazione di ingressi.
- **void adjust_weights(NET *n)**
  effettua la correzione dei pesi dello strato di uscita e di quello nascosto della rete **n**, in base alla delta rule e alla tecnica di retropropagazione dell'errore.
- **void set_learning_parms(NET *n, double eps, long int epi)**
  imposta i parametri di apprendimento per la rete **n**. Tali parametri sono: **eps**, cioè la precisione che si desidera raggingere per la funzione errore e **epi**, ovvero il numero di iterazioni massime per ciascun ciclo di apprendimento.
- **long learn(NET *n, TSET *t, TSET *v)**
  esegue il ciclo di apprendimento per la rete **n**, utilizzando il training set contenuto in **t** e il validation set **v**. La funzione ritorna -1 se il ciclo di apprendimento è stato completato senza raggiungere la precisione desiderata, altrimenti ritorna l'epoca alla quale è stata raggiunto il limite di precisione. E' possibile non utilizzare il validation set passando NULL come valore per **v**.
- **int save_weights(NET *n, char *fname)**
  salva nel file **fname** tutti i dati inerenti la rete neurale **n**.

B.6 output

- **void init_output_engine(void)**
  avvia il motore grafico.
- **int graph_mode(void)**

restituisce la modalità grafica corrente: 0 indica il modo testo, mentre 1 indica la modalità grafica.

- **void image_to_screen(TImageData *in_frame, int x_ofs, int y_ofs)**

  visualizza a video l'immagine puntata da **in_frame**, con l'angolo superiore sinistro posto alle coordinate **x_ofs** e **y_ofs**. Visualizza, inoltre, il time-stamp proprio dell'immagine. La funzione effettua i controlli sulla modalità video corrente, generando una condizione di errore se non si è in modo grafico. Si ha errore anche se l'immagine viola le dimensioni della finestra grafica.

- **void update_driver(int hmv, int vmv, float hfrac, float vfrac,**
  int hval, int vval)

  aggiorna i dati relativi al driver che controlla i motori. Visualizza lo stato di moto, la percentuale di pilotaggio e una percentuale dello spazio percorso dai motori per portarsi nella posizione attualmente richiesta processo che controlla i motori.

- **void update_thresh_bar(int th)**

  aggiorna la barra che visualizza il livello di soglia attualmente utilizzato per scandire l'immagine. Il valore di **th** deve essere compreso tra 0 e 255.

- **void make_default_screen(void)**

  realizza la schermata principale comune ai diversi moduli esecutivi, disegnando le diverse finestre, le barre e i led.

- **void hpower_led_set(int onoff)**

  void vpower_led_set(int onoff)

  void laser_led_set(int onoff)

  void valve_led_set(int onoff)

  impostano dinamicamente il valore dei led visualizzati sullo schermo, in base allo stato dei diversi dispositivi hardware, rispettivamente: il

motore orizzontale e verticale (che possono essere o meno in moto), il laser (acceso/spento), la valvola pneumatica (aperta/chiusa).

- **void set_statusbar(char *st, int blk)**

  imposta il valore del messaggio che deve visualizzato nella barra di stato. **blk** indica se si desidera una scritta lampeggiante (blk = 1) oppure no (blk = 0).

- **TASK statusbar_TASK(void)**

  è il task periodico che si occupa della visualizzazione del messaggio lampeggiante sullo schermo.

- **void go_statusbar_MOD(void)**

  crea ed avvia il processo che gestisce la visualizzazione dei messaggi lampeggianti sulla status bar.

B.7 pci6025e

E' la libreria di interfacciamento alla scheda National Instruments PCI-6025E. Contiene tutte le procedure di inizializzazione del bus PCI, dei vari chip proprietari a bordo della scheda e le funzioni necessarie a gestire alcune funzionalità dei timer e dell'I/O digitale che sono state sfruttate in DEA. La scrittura delle altre funzioni, sia di controllo dei timer che degli ADC e DAC, che non è stata completata in questa prima stesura della libreria, è comunque notevolmente agevolata dalla visione delle procedure già implementate, in quanto si tratta di scrivere in appositi registri con modalità standard già sperimentate. Sostanzialmente basta sostituire i codici dei registri e dei singoli bits nei registri stessi per sviluppare tutte le altre funzionalità della scheda non supportate dalla versione corrente del driver.

- **int wbit(WORD *w, WORD ofs, WORD bits, WORD n)**

pone gli n bit della parola w, a partire dall'offset ofs (0 indica il bit meno significativo) pari al contenuto degli n bit meno significativi di bits. Questa funzione è usata estensivamente all'interno del modulo.

- **int find_NI_devices(void)**
  scandisce il bus PCI alla ricerca tutte le periferiche della National Instruments. Quando ne viene trovata una, i dati relativi vengono salvati in strutture interne al modulo. Il modulo fornisce le funzioni di gestione della sola scheda PCI-6025E.
- **int setup_mite(void)**
  Esegue le istruzioni di setup dei registri del chip che si occupa di rimappare in memoria lo spazio di indirizzamento del chip DAQ-STC montato sulla scheda.
- **void DIO_pin_config(BYTE b)**
  configura le linee di I/O digitale per l'input o per l'output in base ai valori degli 8 bit del parametro b. Un bit a 0 indica linea in ingresso, mentre 1 significa linea d'uscita.
- **BYTE DIO_get_setting(void)**
  restituisce le impostazioni di ingresso/uscita correnti per le linee di I/O digitale.
- **void DIO_parallel_output(BYTE b)**
  scrive gli 8 bit di b sulle uscite digitali. Se una linea è configurata per l'ingresso una scrittura non ne cambia il valore in input.
- **int DIO_parallel_put_bit(int value, int n)**
  scrive il bit meno significativo di value sull'n-esima uscita digitale. n deve essere un valore tra 0 e 7 compresi.
- **BYTE DIO_parallel_input(void)**
  legge gli 8 bit presenti sulle linee di I/O digitale. Se la linea è di ingresso legge il valore in input; se una linea è in output il valore ritornato corrisponderà all'ultimo valore scritto.

- **void timer_clock_configure(WORD divider)**
  imposta il clock che verrà utilizzato da entrambi i contatori.
- **void G0_reset(void)**
  inizializza il timer general-purpose G0, ponendo i giusti valori in tutti i registri relativi.
- **void enable_G0_out(void)**
  abilita l'output del contatore G0.
- **void G0_Arm_All(void)**
  fa in modo che il contatore G0 inizi il conteggio secondo i parametri impostati con le dovute funzioni.
- **void G0_Disarm_All(void)**
  termina la sequenza di conteggio di G0.
- **int G0_Armed(void)**
  restituisce lo stato attuale del contatore G0, e vale 1 se sa contando, oppure 0 altrimenti.
- **void G0_continuos_pulse_train_generation(**
  WORD source, WORD source_polarity,
  WORD gating_mode, WORD gate, WORD gate_polarity,
  WORD output_mode, WORD output_polarity,
  WORD bank_switching,
  DWORD delay, DWORD period, DWORD duration)
  imposta i parametri relativi alla forma d'onda e le opzioni di controllo e output relative al treno di impulsi che verrà generato dal contatore G0. Imposta i registri in modo da poter successivamente variare il duty-cycle senza interrompere il treno di impulsi stesso. Il significato dei parametri è il seguente:
  - **source** specifica la sorgente del clock su cui si basa il conteggio;
  - **source_polarity** indica su quale fronte del clock effettuare lo step di conteggio;

- **gating_mode** imposta una delle possibili modalità di controllo del conteggio;
- **gate** specifica la sorgente del segnale di gate (in genere quale pin della scheda);
- **gate_polarity** indica se il conteggio va effettuato mentre il segnale di gate è a livello basso oppure alto;
- **output_mode** imposta la modalità di output;
- **output_polarity** indica la polarità del segnale di uscita;
- **bank_switching** abilita o meno la possibilità di scambiare il banco di registri, operazione indispensabile per cambiare periodo e durata dell'impulso senza resettare il contatore;
- **delay** è il ritardo iniziale introdotto prima della generazione del primo impulso della sequenza; indica il numero di periodi di clock da attendere;
- **period** indica il numero di periodi di clock che generano il low-time del segnale di uscita;
- **duration** indica il numero di periodi di clock che generano l'high-time del segnale di uscita, cioè della durata dell'impulso.

- **int G0_pulse_train_change(DWORD period, DWORD duration)**
  cambia le impostazioni del treno di impulsi mentre questo viene correntemente generato, utilizzando le nuove quantità di **period** e **duration**;
- **void PFI_pin_program(int number, WORD value, WORD sel7)**
  imposta il pin number dei PFI come ingresso o uscita a seconda del valore di value;
- **WORD get_G_status_register(void)**
  restituisce il valore dello status register comune ai due contatori. I singoli bit di questo registro forniscono preziose informazioni circa il funzionamento dei contatori.

- **WORD get_joint_status_1(void)** e **WORD get_joint_status_2(void)** restituiscono i valori dei due registri joint status 1 e 2, che contengono diverse informazioni circa il funzionamento dei timer.
- **void G0_pulse_width_measurement(**
  DWORD initvalue, WORD source, WORD source_polarity,
  WORD gate, WORD gate_polarity,
  WORD output_mode, WORD output_polarity)

  utilizza il contatore G0 per misurare la durata di un impulso, cioè il tempo che intercorre tra il rise e il fall (o viceversa, in base al valore di *gate_polarity*) di un segnale in ingresso alla scheda. La misurazione viene fatta in termini di periodi di clock intercorsi tra i due eventi del gate selezionato. Il valore finale del conteggio, che parte dal valore di *initvalue*, può essere letto dall'HW Save Register, mentre il valore parziale del conteggio può essere ricavato leggendo dal Save Register. Con tali valori è possibile calcolare, in base al periodo di clock impostato, la durata effettiva dell'impulso o del conteggio parziale. I parametri hanno il seguente significato:
  - **initvalue** è il valore iniziale caricato nel registro al quale partirà il conteggio del timer;
  - **source** specifica la sorgente del clock su cui si basa il conteggio;
  - **source_polarity** indica su quale fronte del clock effettuare lo step di conteggio;
  - **gate** specifica la sorgente del segnale di gate del quale si vuole misurare la durata dell'impulso;
  - **gate_polarity** indica se iniziare il conteggio quando si ha un rise del gate oppure in caso di fall dello stesso segnale;
  - **output_mode** imposta la modalità di output;
  - **output_polarity** indica la polarità del segnale di uscita.
- **void G0_single_period_measurement(**

DWORD initvalue, WORD source, WORD source_polarity,
WORD gate, WORD gate_polarity,
WORD output_mode, WORD output_polarity)

utilizza il contatore G0 per misurare il periodo del segnale selezionato come gate. Il periodo si considera inizi con il rise (o con un fall) del segnale e termini con un'altra corrispondente transizione. Il valore finale del conteggio, che parte dal valore di *initvalue*, può essere letto dall'HW Save Register, mentre il valore parziale del conteggio può essere ricavato leggendo dal Save Register. Con tali valori è possibile calcolare, in base al periodo di clock impostato, la durata effettiva dell'impulso o del conteggio parziale. I parametri hanno il seguente significato:

- **initvalue** è il valore iniziale caricato nel registro al quale partirà il conteggio del timer;
- **source** specifica la sorgente del clock su cui si basa il conteggio;
- **source_polarity** indica su quale fronte del clock effettuare lo step di conteggio;
- **gate** specifica la sorgente del segnale di gate (in genere quale pin della scheda);
- **gate_polarity** indica se iniziare il conteggio quando si ha un rise del gate oppure in caso di fall dello stesso segnale;
- **output_mode** imposta la modalità di output;
- **output_polarity** indica la polarità del segnale di uscita.

- **DWORD G0_read_save_register(void)**
  permette di leggere il valore contenuto nel Save Register del timer G0.
- **DWORD G0_read_HW_save_register(void)**
  permette di leggere il valore contenuto nell'HW Save Register del timer G0.

Per il significato specifico delle diverse funzioni, ma soprattutto per capire il significato dei relativi parametri, è consigliabile consultare il manuale della scheda e, soprattutto, del chip DAQ-STC. Il tutto può essere scaricato gratuitamente dal sito della National Instruments all'indirizzo

http://www.ni.com/support

B.8 pxc

- **PXC_MAXX**

  PXC_MAXY

  larghezza e altezza in pixel dell'immagine catturata dal frame-grabber. La versione attuale del driver può gestire soltanto immagini con formato 384x288.

- **PXC_MAX_LISTENERS**

  indica il numero massimo di task che possono accedere contemporaneamente al CAB contenente l'immagine catturata dal grabber.

- **TImageData**

  struttura dati che contiene effettivamente l'immagine e il time-stamp, che indica l'istante in cui l'immagine si è resa disponibile al frame-grabber, valore riferito al clock di sistema.

- **TIME PXC_Initiate(int number)**

  inizializza il driver del grabber, il gestore del DMA ed il bus PCI. Deve essere indicato il numero di task *più uno* che si prevede debbano accedere contemporaneamente al CAB contenente l'immagine. La funzione restituisce il periodo caratteristico del driver espresso in millisecondi: i task che vengono agganciati al driver dalla funzione *PXC_Push_Listener()* devono avere tempo minimo di interarrivo (in

quanto sono task aperiodici) pari ad un multiplo intero del periodo caratteristico di funzionamento del grabber.

- **void PXC_Start(void)**

  avvia il driver del frame-grabber Conexant PXC, effettuando le operazioni di impostazione del bus PCI e abilitazione del DMA.

- **void PXC_Close(void)**

  termina il driver, fermando il DMA e togliendo dalla coda tutti i processi precedentemente inseriti.

- **int PXC_Push_Listener(PID l)**

  aggiunge il processo indicato dal suo PID, **l**, alla coda dei task che utilizzano il frame-grabber. Attenzione: il task deve essere dichiarato aperiodico, hard o soft secondo le necessità, con minimo tempo di interarrivo pari a un multiplo intero del periodo caratteristico del driver. Non è necessario attivare il task prima dell'inserimento nella coda del driver, anche se una attivazione non influisce sul corretto funzionamento del driver.

- **int PXC_Pop_Listener(void)**

  elimina dalla coda dei processi che usano il driver l'ultimo task che è stato inserito.

- **CAB PXC_GetCab(void)**

  restituisce un puntatore al CAB che contiene i dati relativi all'ultima immagine catturata dal frame-grabber. Il processo che usa il CAB deve possedere un CAB ed una variabile di tipi TImageData nella quale memorizzare l'immagine da elaborare.

B.9 kalman

Le seguenti costanti servono a dimensionare le massime capacità elaborative messe a disposizione dalla libreria che implementa il predittore di Kalman.

Sostanzialmente controllano la dimensione massima dei vettori che contengono variabili, matrici e quant'altro è necessario alla batteria di predittori. Questo è necessario in quanto, in ambiente HARTIK, non è possibile gestire variabili allocate dinamicamente, quindi i vettori necessari vanno sovradimensionati per contenere tutti i dati necessari. Le costanti che seguono, ovviamente, controllano l'occupazione di memoria da attribuire al modulo *kalman.c*, che può essere cambiata secondo le necessità semplicemente ricompilando il codice relativo.

- **MAX_ORDER**

  Limita l'ordine massimo del modello usato per il sistema di cui osservare lo stato. In pratica costituisce la massima dimensione della matrice *A*.

- **MAX_GAINS**

  Dimensione massima della look-up table, ovvero massimo numero di valori di rho e relativi guadagni che possono essere caricati da file.

- **MAX_SURVEYS**

  Numero massimo di rilevamenti forniti al predittore che vengono conservati nel buffer interno al modulo *kalman.c*.

- **MAX_NVI**

  Numero massimo di variabili indipendenti che possono essere trattate contemporaneamente dalla batteria di osservatori dello stato.

- **MAX_PRED_BUF**

  Valore massimo di passi in avanti al quale può essere fatta una predizione memorizzabile nel buffer circolare, che ospita le previsioni effettuate, ad ogni nuova osservata, *nstep* passi in avanti (con *nstep* impostato dalla funzione *set_prediction_step()*).

- **void init_kalman(int nvi, int *orders)**

  inizializza la batteria dei predittori di Kalman. Vanno indicati il numero di variabili indipendenti delle quali si vuole effettuare la stima e l'ordine del modello da utilizzare nel predittore. orders sarà, in genere, un vettore

di int che contiene l'ordine di ciascun modello: si deve fare attenzione ad utilizzare un vettore di dimensione opportuna!

- **int get_order(int d)**

  restituisce l'ordine del modello utilizzato per la d-esima variabile da osservare.

- **float set_rho(int d, float rho, int seekmode)**

  imposta il rho al valore desiderato per la d-esima variabile, opportunamente scelto nella look-up table. Quest'ultima deve essere preventivamente essere stata caricata mediante una chiamata a *load_gains()*. Il parametro *seekmode* indica la modalità di scelta del valore reale di rho all'interno della tabella. I valori possibili sono:

  - 0 : usa il valore di rho più vicino a quello specificato dal parametro *rho*;
  - 1 : approssima rho per difetto nella look-up table;
  - 2 : approssima rho per eccesso nella look-up table.

- **int get_rho_id(int d)**

  ritorna l'indice che, nella look-up table, individua il valore di rho correntemente utilizzato.

- **float get_rho(int d)**

  restituisce il valore vero e proprio del rho usato nel predittore di Kalman per la d-esima variabile.

- **void set_rho_by_id(int d, int id)**

  imposta il valore di rho, per la variabile d-esima, pari a quello individuato dall'indice *id* nella look-up table.

- **void load_gains(int d, char *fname)**

  carica la look-up table contenente i valori di rho per la variabile d-esima. I valori vengono letti dal file identificato dalla stringa *fname*. Il file, per essere correttamente letto dalla funzione, deve avere la struttura riportata al paragrafo 5.3.3.

- **void get_gains(int d, float *g, int n)**

  ritorna, nell'array puntato da *d*, i guadagni relativi alla d-esima variabile individuati, nella relativa look-up table, dall'indice *n*.

- **int get_ngains(int d)**

  ritorna il numero di guadagni caricati nella look-up table relativa alla d-esima variabile indipendente.

- **void init_kalman_subsystem(char *fname)**

  è una funzione che accorpa tutte le funzioni di inizializzazione del sottosistema costituito dalla batteria di filtri di Kalman. I parametri per la corretta inizializzazione vengono letti dal file *fname*, che deve avere la struttura descritta nel paragrafo 5.3.3.

- **void predict(TSurvey *pr, TSurvey s, int n)**

  effettua una predizione n passi in avanti, a partire dallo stato corrente della batteria di filtri. Per *n* pari a 0 viene restituito lo stato corrente delle variabili di cui di vuole effettuare il monitoraggio.

- **void set_prediction_step(int n)**

  fissa il numero di passi in avanti al quale si desidera effettuare le previsioni che verranno memorizzate nel buffer interno alla libreria. Questo valore è comune a tutte le variabili indipendenti.

- **int get_nstep(void)**

  ritorna il numero di passi in avanti ai quali si effettuano le previsioni che vengono memorizzate nel buffer.

- **int get_buf_pred(TSurvey *pr, int n, int when)**

  restituisce la previsione a *n* passi immagazzinata nel buffer al corrispondente step precedente. *when* indica se l'aggiornamento del buffer, all'istante corrente, è stato o meno già effettuato, in quanto, nei due casi, cambia la posizione relativa nel buffer in cui è memorizzata l'informazione richiesta. Per la modalità di memorizzazione nel buffer, e quindi per la spiegazione del suo contenuto al generico istante sono

riportate in dettaglio al paragrafo 5.6.9. La funzione ritorna 0 in caso di successo, mentre ritorna –1 nel caso in cui sa richiesta una previsione per un numero di passi superiore a quello contenuto nel buffer.

- **void add_survey(TSurvey s)**

  aggiunge una nuova rilevazione al buffer interno alla libreria. Effettua inoltre una previsione di *nstep* passi (*nstep* viene impostato mediante la funzione *set_prediction_step()*) e la deposita nel buffer delle previsioni.

- **int get_survey(TSurvey *s, int n)**

  ritorna in *s* il rilevamento effettuato *n* passi addietro e immagazzinato nel buffer dei rilevamenti interno al modulo *kalman.c*. La funzione restituisce 0 se tutto ha funzionato correttamente, mentre ritorna –1 se è stato passato un indice non corretto.

- **int get_n_survey(void)**

  ritorna il numero di rilevamenti contenuti attualmente nel buffer. In condizioni di regime, tale valore corrisponderà alle dimensioni del buffer, mentre nel transitorio di riempimento, indicherà quanti rilevamenti sono stati forniti al predittore fino all’istante attuale.

Appendice C

Codice MATLAB

C.1 Soluzione dell'equazione di Riccati

Per poter generare i guadagni necessari al funzionamento del predittore di Kalman è indispensabile risolvere l'equazione di Riccati, come descritto nel capitolo 2.5.5. MATLAB dispone di un solutore per questo tipo di equazione, ma che non funziona correttamente quando gli autovalori della matrice A sono troppo vicini alla regione di instabilità. Una tecnica generale per la soluzione dell'equazione è quella di utilizzare il metodo iterativo descritto nello stesso capitolo. Il codice MATLAB che svolge questa operazione è il seguente:

```
function [P, out_eps, nout] = riccati(P0, A, B, C, Q, R, eps, nin)
% riccati.m try to solve the Riccati's equation
% with an iterative method
% [P, out_eps, nout] = riccati(P0, A, B, C, Q, R, eps, nin)
% solve the Riccati's equation in the form:
%                                                          -1
% P(k+1) = A P(k) A' + B Q B' - A P(k) C' * (C P(k) C' + R)  *
%          C P(k) A'
% eps = desired precision
% nin = max iteration number
% out_eps = precision reached
% nout = number of iteration performed
```

```
%

ni = nargin;
no = nargout;

if ni ~= 8
  error('Wrong number of parameters')
end

if no ~= 3
  error('Wrong number of return values')
end

error(abcdchk(A,B));
if ~length(A) | ~length(B)
  error('A and B matrices cannot be empty.')
end

Pk = P0;

for i = 1:nin,
  Pk1 = A * Pk * A' + B * Q * B' - A * Pk * C' *
        inv(C * Pk * C' + R) * C * Pk * A';
  Pk = Pk1;
  if (norm(Pk1 - Pk, 2) / norm(Pk, 2)) < eps
    break
  end
end

nout = i;
P = Pk;
out_eps = eps;
```

Il programma ha dunque la forma di una funzione, *riccati()*, che risolve iterativamente l'equazione omonima.

Questa funzione riceve in input le matrici che descrivono il sistema dinamico, cioè *A*, *B* e *C*, il valore di partenza per l'iterazione *P0*, le matrici che descrivono il rumore sui dati, ovvero *Q* ed *R*, la precisione desiderata per l'iterazione (*eps*) in termini di residuo della soluzione ed infine il numero massimo di iterate, specificato da *nin*. In uscita si ha, invece, il risultato dell'approssimazione (*P*), la precisione raggiunta dall'iterazione ed il numero di cicli effettivamente compiuti, rispettivamente *out_eps* e *nout*.

C.2 Simulazione del predittore di Kalman

La simulazione che viene effettuata dal seguente programma è stata sviluppata per testare il funzionamento del predittore di Kalman.

```
function [zk, xk, zk_pred, xk_pred, An] = tk(nA, rho, nstep)
% tk.m
% Simulazione del filtro di Kalman
%
% [zk, xk, zk_pred, xk_pred, An] = tk(nA, rho, nstep)
%

error(nargchk(3, 3, nargin))

close all

% Varianza del rumore
max_noise = 0;

% Passo di campionamento
dt = 0.04;

% Generazione dei valori di test
k = [0:dt:2*pi];
nk = max(size(k));
noise = 2 * (rand(size(k)) - 0.5) * max_noise;
x0 = 0;
v0 = 170;
a0 = 10;
zk_real = round(100 * sin(1 * k));
zk = round(zk_real) + round(noise);

% Matrici caratteristiche del sistema in esame
A = diag(ones(1, nA), 0) + diag(ones(1, nA - 1),1);
B = zeros(1, nA);
B(nA) = 1;
B = B';
C = zeros(1, nA);
C(1) = 1;

% Imposta il parametro che controlla il predittore
Q = 1;
R = rho;

% Calcola la soluzione dell'equazione di Riccati con metodo
% iterativo
P0 = eye(size(A));
[P, eps, nout] = riccati(P0, A, B, C, Q, R, -inf, 100);

norm(-P + A * P * A' - A * P * C' * inv(C * P * C' + R) * C * P *
```

```
     A' + B * Q * B')

% Calcola il guadagno del filtro di Kalman
K = A * P * C' * inv(C * P * C' + R)

abs(eig(A - K * C))

xk = zeros(nA, nk);
xk1 = zeros(nA, nk);

xk_pred = zeros(nA, nk);

pow = zeros(1, nstep + 1);
pow(1) = 1;
An = polyvalm(pow, A)

for i = 1:(nk - nstep)
  xk(:,i + 1) = A * xk(:,i) + K * (zk(i) - C * xk(:,i));
  xk_pred(:,i + nstep) = An * xk(:,i + 1);
  zk_pred(i + nstep) = C * xk_pred(:,i + nstep);
end

% Stampa il grafico
axis xy
grid on
plot(k, zk_real, 'g.', k, zk, 'b.', k, xk_pred(1,:), 'r.');
legend('Trajectory', 'Noisy measurement', 'Prediction', 0);
title('Functions');
xlabel('k - Discrete time');
```

Il programma utilizza il predittore per stimare lo stato di un sistema dinamico che evolve nel tempo secondo una legge nota, che è codificata all'interno del programma stesso: è quindi molto semplice variare la funzione che determina l'evoluzione del sistema, in modo da testare il filtro nelle più diverse condizioni operative. Nel listato precedente, e negli esempi che seguono, è utilizzata una funzione sinusoidale. E' anche possibile aggiungere del rumore al segnale, in modo da verificare il comportamento del filtro aumentando progressivamente i disturbi sugli ingressi.

La simulazione è codificata in MATLAB come una funzione che accetta i seguenti parametri:

- *nA*, il numero di integratori del modello;
- *rho*, ovvero il parametro che controlla la sensibilità al rumore del filtro;

- *nstep*, cioè il numero di passi in avanti a cui si desidera che il predittore di Kalman effettui le previsioni.

In uscita, invece, sono restituiti i seguenti valori:

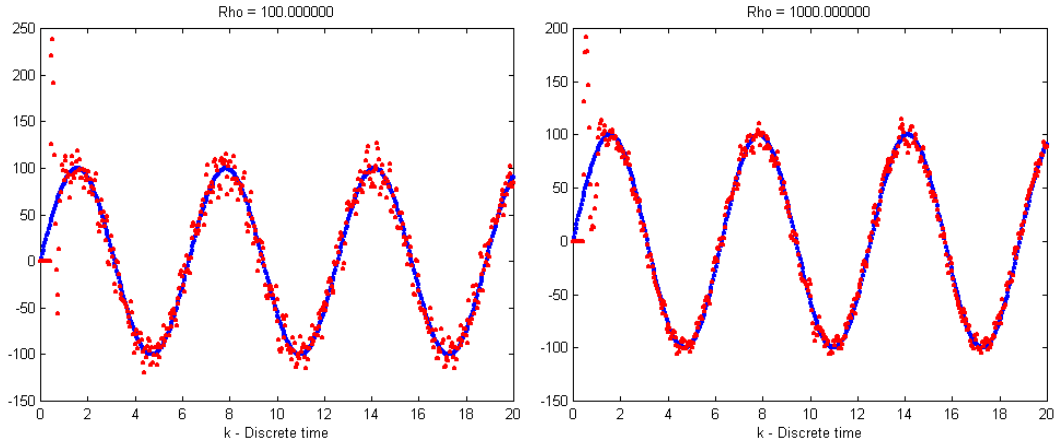
- *zk*, il vettore contenente la sequenza di valori che viene passata in ingresso al filtro, sulla base dei quali viene fatta la previsione;
- *xk*, il vettore di stato del filtro;
- *zk_pred*, l'uscita del sistema prevista dal filtro, sulla base del proprio stato interno, *nstep* passi in avanti;
- *xk_pred*, il vettore che contiene lo stato del sistema previsto dal filtro *nstep* passi in avanti;
- *An*, la matrice che modellizza il sistema sotto osservazione.

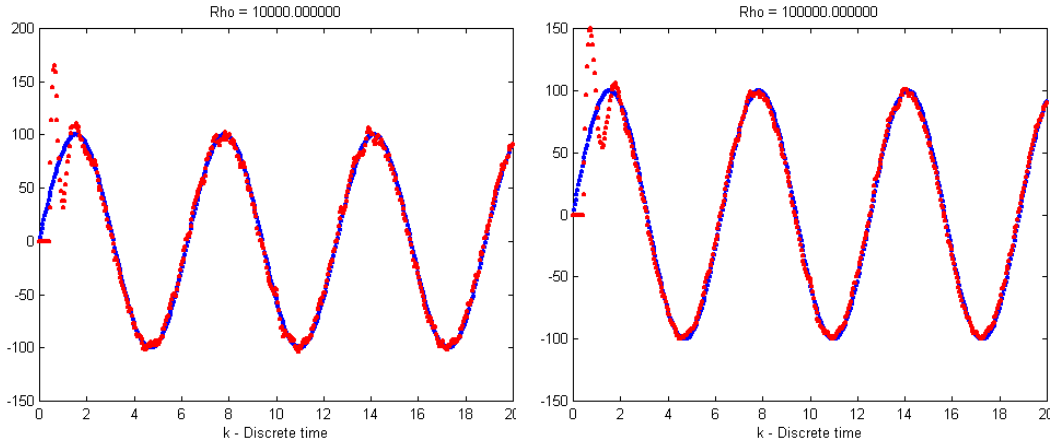
Il programma esegue in sequenza i seguenti step:

- inizia controllando che il numero di parametri sia corretto;
- costruisce il vettore contenente i valori di ingresso al filtro, i quali sono poi arrotondati all'intero più vicino per simulare l'errore di quantizzazione che si ha, in DEA, trattando con i pixel, che sono espressi da numeri interi. Il rumore può essere aggiunto agendo sulla variabile *max_noise*.
- vengono prodotte le matrici che modellizzano il sistema sotto osservazione, le quali hanno una dimensione compatibile col numero di integratori specificato dal parametro *nA*;
- viene risolta l'equazione di Riccati per mezzo della funzione descritta al precedente paragrafo, necessaria per poter calcolare i guadagni del predittore in base al valore di *rho* desiderato;
- si calcola la potenza *nstep*-esima della matrice *A*, che serve per poter effettuare la predizione in avanti del numero di passi corrispondente;
- si effettua la simulazione vera e propria: in un ciclo vengono passati al predittore i diversi valori di ingresso, in modo da far evolvere il suo stato interno. Contemporaneamente si effettuano le previsioni *nstep* passi in avanti, e le si memorizzano in un vettore per poterle poi visualizzare e restituire in uscita alla simulazione;

- viene visualizzato un grafico che comprende la funzione vera e propria incluso l'errore di quantizzazione, la funzione alla quale viene aggiunto il rumore determinato da *max_noise*, le previsioni a *nstep* passi effettuate dal filtro.

Di seguito sono riportati alcuni grafici che mostrano il comportamento del filtro in alcuni casi significativi.





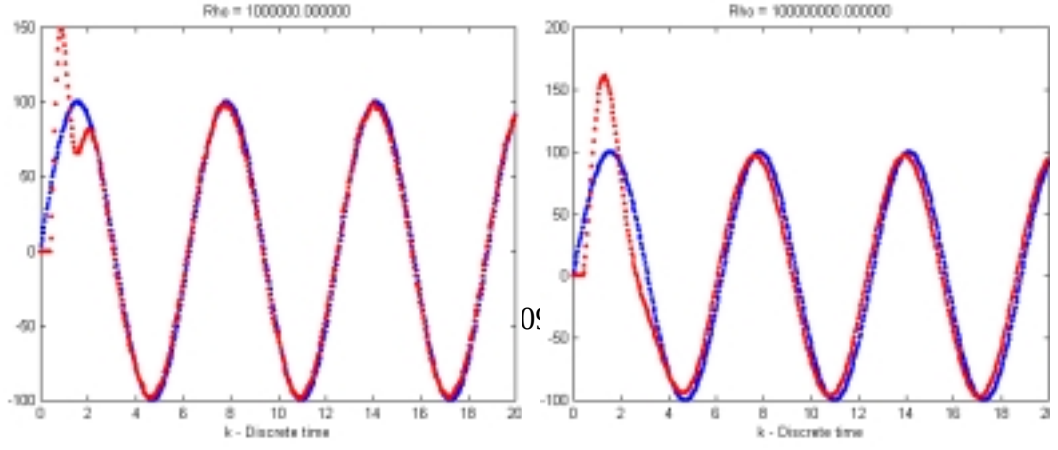


Figura C.1. Comportamento del filtro in assenza di rumore.

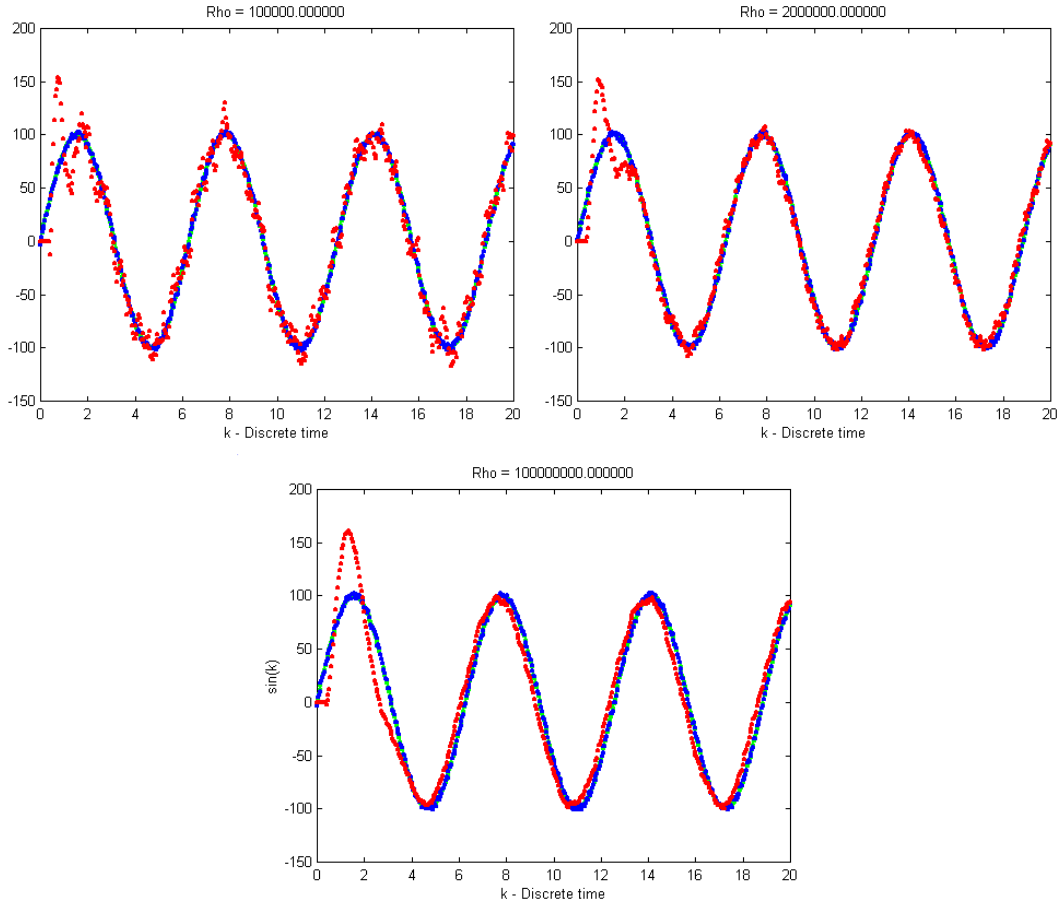


Figura C.2. Comportamento del filtro con errore del 3%.

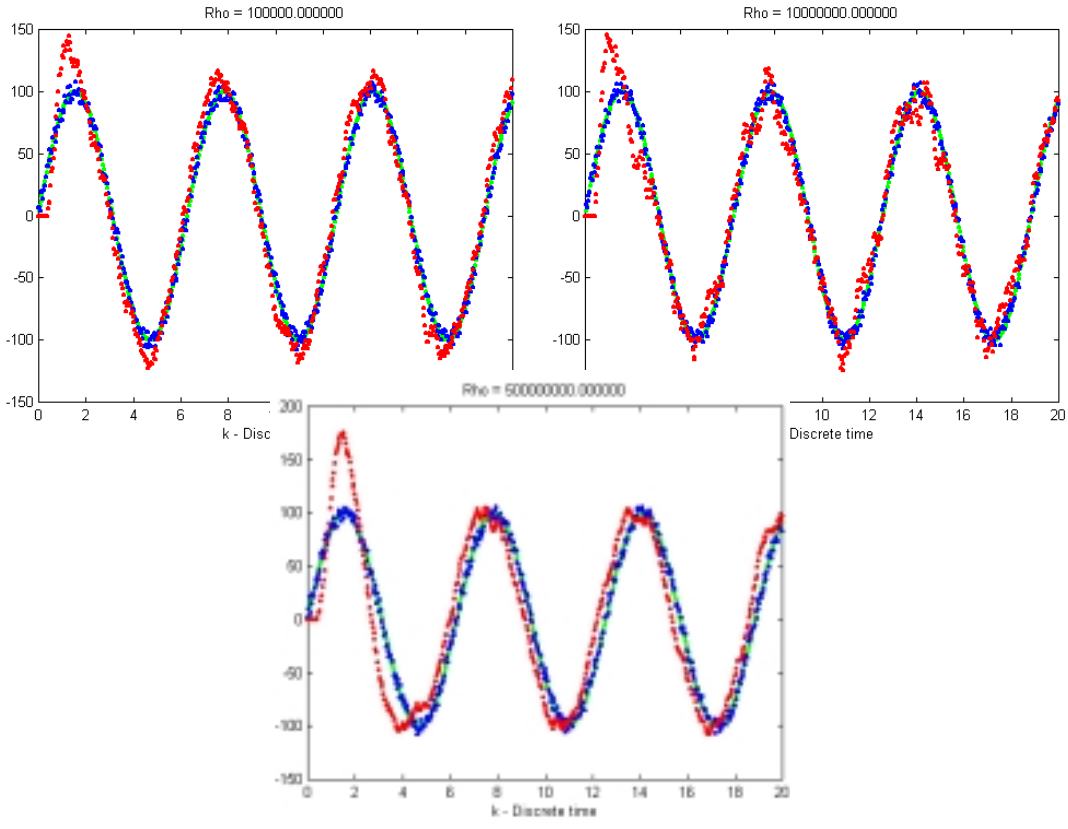


Figura C.3. Comportamento del filtro con rumore pari all'8%.

In tutte le figure si ha:

- punti verdi = traiettoria reale;
- punti blu = traiettoria a cui è stato aggiunto il rumore;
- punti rossi = traiettoria predetta dal filtro;
- la funzione di cui stimare l'andamento è una sinusoide con frequenza unitaria e ampiezza pari a 100, cioè

$$f(x) = 100 \cdot sin(x)$$

- la predizione viene fatta 10 passi in avanti.

Osservando le figure si intuisce immediatamente il comportamento del filtro: aumentando ρ è possibile aumentare la capacità di filtraggio, ma, come si può notare, il tempo di assestamento che porta la traiettoria predetta ad assomigliare a quella reale è più lungo: questo fatto è immediatamente riscontrabile osservando il primissimo tratto della traiettoria predetta, che è il transitorio di assestamento del filtro. Un importante aspetto può non emergere ad una prima analisi delle figure: nelle prime due la dimensione del modello è pari a 4, ciò significa che la matrice *A* appartiene a $M^{4\times4}$, mentre nell'ultima il modello ha dimensione 3, cioè la matrice *A* appartiene a $M^{3\times3}$. Agire sulla dimensione del sistema è una ulteriore possibilità di tarare il sistema. Nei casi in figura, con un errore relativamente grande è utile abbassare tale dimensione, in modo da rendere il filtro meno sensibile al rumore. Alzare la dimensione del sistema significa considerare più derivate dell'accelerazione, e questo consente al filtro, in genere, di poter predire andamenti più "bizzarri" della funzione, ma lo stesso filtro può essere ingannato dalla presenza di rumore: il filtro segue una variazione rapida della variabile osservata dovuta al rumore, perdendo la traiettoria.

C.3 Generazione dei guadagni

Il programma che si occupa di generare i data file contenenti i guadagni del predittore è listato di seguito. Esso è strutturato come una funzione che accetta come ingressi il numero di integratori con cui modellizzare il sistema (*nA*) e il nome del file nel quale si desidera memorizzare i dati (*f*), il quale ha la struttura corretta per poter essere letta dalle funzioni della libreria che caricano in memoria i ρ ed i corrispondenti guadagni, come descritto al paragrafo 5.3.3.

Il funzionamento del programma è molto simile a quello della simulazione descritto al paragrafo precedente, con la differenza che questo non produce alcun output su schermo, se non la durata complessiva dell'operazione, e scrive il proprio output su file.

Innanzitutto vengono generate automaticamente le matrici che modellizzano il sistema, con dimensione definita dal valore di *nA*. Poi viene costruito il vettore contenente i ρ dei quali si desidera calcolare i guadagni. Successivamente il programma effettua una iterazione e, per ciascun valore di ρ, calcola i corrispondenti guadagni. Infine, il tutto viene salvato nel file col nome specificato.

```
function genK(nA, f)
% genK.m
%
% Generazione di file contenenti i guadagni del predittore di
% Kalman
% per diversi valori di rho
% genK(nA, fname)
%

error(nargchk(2, 2, nargin))

fname = char(f);

% Matrici caratteristiche del sistema in esame
A = diag(ones(1, nA), 0) + diag(ones(1, nA - 1),1);
B = zeros(1, nA);
B(nA) = 1;
B = B';
C = zeros(1, nA);
C(1) = 1;
```

```
% Usato per eliminare il doppione alla frontiera dei due insiemi
low_part = logspace(-10, 2, 39);
rho = [low_part(1, 1:38) logspace(2, 12, 82)];
nrho = max(size(rho));

outfile = fopen(fname, 'w');

fprintf(outfile, '%d\n', nA);
fprintf(outfile, '%d\n', nrho);

tic
for i = 1:nrho,

  % Imposta il parametro che controlla il predittore
  Q = 1;
  R = rho(i);

  % Calcola la soluzione dell'equazione di Riccati con metodo
  % iterativo
  P0 = eye(size(A));
  [P, eps, nout] = riccati(P0, A, B, C, Q, R, -inf, 100);

  norm(-P + A * P * A' - A * P * C' * inv(C * P * C' + R) * C *
       P * A' + B * Q * B');

  % Calcola il guadagno del filtro di Kalman
  K = A * P * C' * inv(C * P * C' + R);

  fprintf(outfile, '%f ', rho(i));
  fprintf(outfile, '%f ', K);
  fprintf(outfile, '\n');
end
fprintf(outfile, '\n');
toc

fclose(outfile);
```

Bibliografia

[1] G. Buttazzo, B. Allotta, F. Fanizza. *Mousebuster: a Robot System for Catching Fast Moving Objects by Vision*. Proceedings of IEEE International Conference on Robotics and Automation, Atlanta (May 1993).

[2] Giorgio C. Buttazzo. *Sistemi in tempo reale*. Pitagora Editrice Bologna (1995).

[3] Giorgio C. Buttazzo, Luigi Palopoli, Luca Abeni, Giuseppe Lipari, Gerardo Lamastra, Antonio Casile, Massimiliano Giorgi, Paolo Gai. *HARTIK user manual*. RETIS Lab., Scuola Superiore S. Anna, via Carducci, 40 - 56100 Pisa. Revisione 3.3.1. (1998).

[4] Gregory D. Hager, Peter N. Belhumeur. *Efficient Region Tracking of Image Regions With Changes in Geometry and Illumination.*

[5] Gregory D. Hager, Peter N. Belhumeur. *Real-Time Region Tracking With Parametric Models of Geometry and Illumination.* In Proc. IEEE Conf. Comp. Vision and Patt. Recog., pages 403-410. IEEE Computer Society Press (1996).

[6] Gregory D. Hager, Kentaro Toyama. *X Vision: A Portable Substrate for Real-Time Vision Application.* (1996).

[7] Ian Horswill, Chris Barnhart. *Unifying segmentation, tracking, and visual search.*

[8] Seth Hutchinson, Greg Hager, Peter Corke. *A Tutorial on Visual Servo Control*. (May 14, 1996).

[9] Brian W. Kernighan, Dennis M. Ritchie. *Linguaggio C ANSI C, Seconda Edizione*. Gruppo Editoriale Jackson (1989).

[10] Steve Lawrence, C. Lee Giles, Ah Chung Tsoi. *What Size neural Network Gives Optimal Generalization? Convergence Properties of Backpropagation.* Technical Report UMIACS-TR-96-22 and CS-TR-3617, Institute fo Advance Computer Studies, University of Maryland, College Park, MD 20742. (June 1996).

[11] Jonathan A. Marshall, Viswanath Srikanth. *Curved trajectory prediction using a self-organizing neural network*. International Journal of Neural Systems, vol. 10, No. 1 - World Scientific Publishing Company (February, 2000).

[12] Mohinder S. Grewal, Angus P. Andrews. *Kalman filtering, Theory and Practice*. Prentice Hall Information and System Science Series, Prentice Hall Inc. (1993).

[13] National Instruments. *DAQ - PCI E Series Register-Level Programmer Manual*. National Instruments Corporation (November 1998).

[14] National Instruments. *DAQ-STC Technical Reference Manual - System Timing Controller for Data Aquisition*. National Instruments Corporation (January 1999).

[15] Hanna Pasula and Stuart Russel, Michael Ostland and Ya'acov Ritov. *Tracking many ojects with many sensors.*

[16] Tomaso Poggio and Federico Girosi. *A Theory of Networks for Approximation and Learning*. Carnegie Mellon University School of Computer Science (1995).

[17] David Prewer, Les Kicthen. *A Simple Fast Edge-Based Visual Tracker*. Technical Report 97/20, Department of Computer Science, The University of Melbourne, Australia (September 1997).

[18] Peter Arnold Ruymgaart, Tsu-teh, Soong. *Mathematics of Kalman-Bucy filtering*. Berlin, Springer (1985).

[19] Warren S. Sarle. *Neural Network FAQ*. Dowloaded from ftp://ftp.sas.com/pub/neural/FAQ.html (2000).

[20] Riccardo Scattolini. *Complementi di controlli automatici e controllo digitale*. Edizioni CUSL (1996).

[21] Michael Schmitt. *Lower Bounds on the Complexity of Approximating Continuous Functions by Sigmoid Neural Networks.* In S.A. Solla, T.K. Leen, K-R Muller. *Advances in Neural Information Processing Systems 12*, pp. 328-334. MIT Press, Cambridge, Mass. (2000).