Indice

Ringraziamenti

Desidero innanzitutto ringraziare il Professor Paolo Righettini per i preziosi insegnamenti che mi hanno permesso di scrivere questo documento e per le ore dedicate alla mia tesi. Inoltre, ringrazio l'Ingegnere Alberto Oldani per il suo supporto durante l'intero tirocinio ed il Professor Roberto Strada per la disponibilità nel derimere i dubbi durante il completamento di questo lavoro.

Ringrazio Nicola Pasta per il lavoro svolto assieme durante questi mesi nello svolgimento del tirocinio.

Desidero poi ringraziare con affetto tutte le persone che mi hanno sostenuto e mi sono stati vicini in questi anni: la mia famiglia, la mia ragazza Federica e tutti i miei amici, in particolare Dario, Simon, Priscilla e i membri dei Turks.

Prefazione

Il documento che avete tra le mani descrive l'esperienza di tirocinio che ho svolto durante l'anno accademico 2008/2009, legata al corso di Sistemi Meccatronici, nel laboratorio di meccatronica del centro di ricerca Point dell'Università di Bergamo, situato a Dalmine.

Il progetto consiste nello sviluppo di un algoritmo di controllo di un pendolo inverso e nella sua implementazione in un sistema real-time in ambiente Linux, utilizzando software open-source come RTAI e Comedi.

L'obiettivo di questo progetto è mantenere in equilibrio il pendolo in posizione verticale.

Ogni capitolo descrive una fase del lavoro che ci ha portato alla realizzazione del sistema.

Nel **capitolo primo** introduciamo il pendolo inverso, presentando le apparecchiature utilizzate ed i componenti del banco di lavoro.

Il **capitolo secondo** presenta il modello teorico del pendolo e le equazioni che ne determinano la dinamica e gli stati di equilibrio.

Nel **capitolo terzo** e **quarto** ci occupiamo della tematica del controllo, della stabilizzazione e del movimento del pendolo. Per mantenere stabile il sistema usiamo una tecnica detta di "riposizionamento dei poli".

Il **capitolo quinto** descrive il progetto RTAI, utilizzato per ottenere un sistema hard real-time.

Nel **capitolo sesto** affrontiamo l'argomento dell'acquisizione dei dati, analizzando il progetto Comedi.

Il **capitolo settimo** riguarda l'installazione e la configurazione del sistema utilizzato, dalla creazione del kernel real-time all'inizializzazione del sistema.

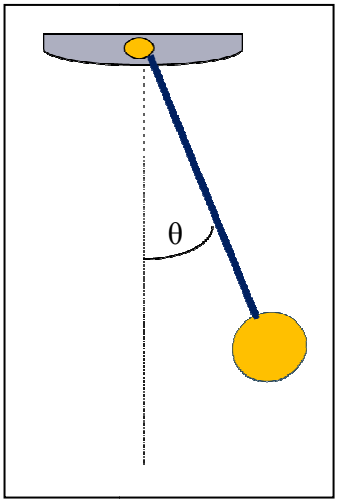
Il **capitolo ottavo** descrive l'implementazione del controllore attraverso l'utilizzo di task real-time.

Nel **capitolo nono** presentiamo l'interfaccia utente creata con le librerie Qt e Qwt per il linguaggio C++.

Nelle **appendici** forniamo informazioni aggiuntive su argomenti trattati durante lo sviluppo del progetto.

1 – Il pendolo inverso

Il pendolo è uno dei sistemi fisici più semplici ed è costituito da un'asta (teoricamente senza peso) le cui estremità sono una incernierata e l'altra fissata ad una massa. L'asta è libera di ruotare nell'estremità opposta alla massa, dando cosi al pendolo un grado di libertà.



1.1 – Pendolo semplice

Un sistema di questo tipo, sotto l'azione della gravità presenta due punti di equilibrio statico, uno di questi è stabile mentre l'altro è un punto di equilibrio instabile. Noi ci occuperemo di un pendolo che si trovi nell'intorno della seconda condizione, ovvero che abbia la massa al di sopra del centro di rotazione.

1.1 - Segway

Posizioniamo il fulcro del pendolo su di un carrello mobile controllato da un motore, in grado di spostarsi lungo un solo asse. Abbiamo implementato cosi quello che viene definito un "pendolo inverso", questo sistema, a differenza del pendolo semplice, presenta due gradi di libertà. Il problema dell'equilibrio del pendolo inverso rappresenta uno dei problemi classici del controllo automatico.

Pure essendo un problema abbastanza astratto, la sua risoluzione porta a svariate applicazioni pratiche ed industriali. Ne sono esempio il "Segway", in cui viene integrata

1.2 - Gru

la conoscenza di questo tipo di controllo per avere un mezzo di trasporto auto bilanciato, oppure il meccanismo di controllo delle grandi gru nelle zone portuali di carico-scarico dei container.

1.1 Banco del pendolo

La realizzazione del controllo del pendolo, come abbiamo detto, non è stata solo teorica, ma implementata su un sistema resoci disponibile dall'università. Il sistema consiste in un "banco pendolo" collegato ad un computer che implementa il controllo attraverso una scheda di acquisizione dati.

In figura possiamo vedere il pendolo da vicino e i vari componenti che costituiscono il banco, di particolare rilevanza sono il motore con l'azionamento, gli encoder (di cui uno integrato direttamente nel motore) e la base che ci permette di collegare il computer all'apparecchiatura elettromeccanica.

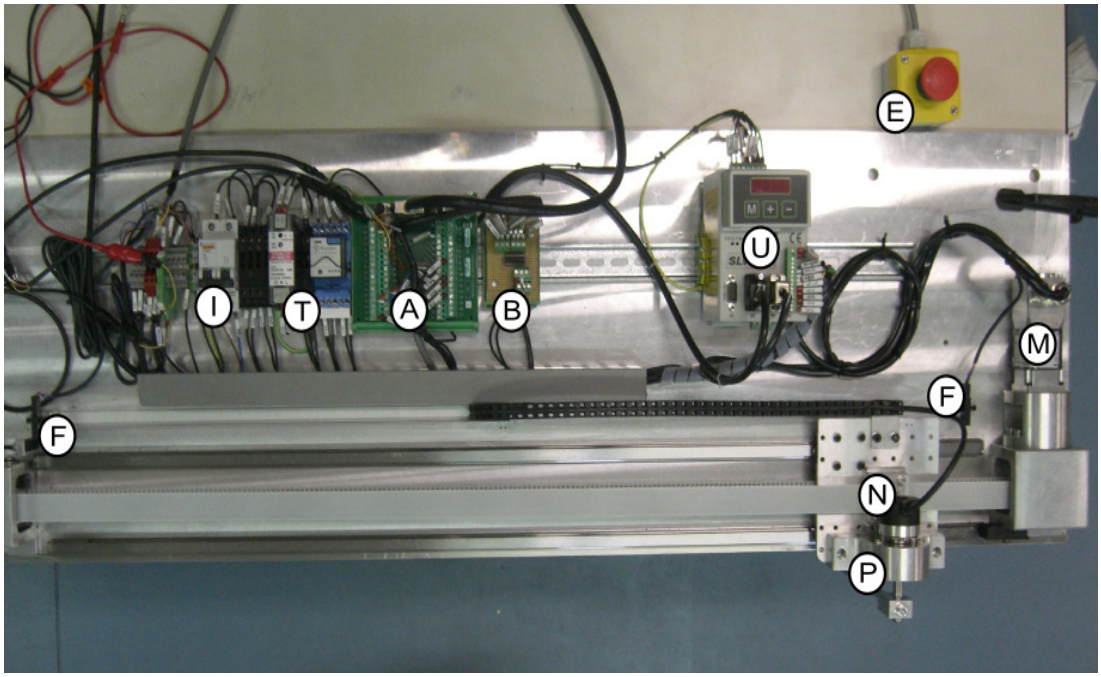


Figura 1.4 - banco del pendolo

P – Carrello del pendolo.
M – Motore Brushless.
I – Interruttore generale.
A – Base di Acquisizione dati.
F – Fine Corsa.
N – Encoder.
U – Azionamento.
T – Gruppo di trasformazione.
B – Line receiver.
E – Pulsante d'emergenza.

Il pendolo che ci viene fornito è costituito da vari blocchi che possiamo vedere segnati nella figura precedente. Notiamo subito il pendolo fisico, costituito da un'asta incernierata ad un carrello da una parte e portante una massa nota dall'altra.

Il carrello scorre, per mezzo di carrelli a ricircolo di sfere, su un binario a basso coefficiente d'attrito. Esso è collegato ad un motore di tipo "brushless" che viene comandato in coppia dal nostro controllore, contenente al suo interno un resolver. Montato sul carrello c'è un encoder di tipo incrementale che misura la posizione angolare del pendolo, mentre vicino agli estremi della guida abbiamo due sensori di fine-corsa.

Il motore e i fine-corsa sono collegati all'azionamento che controlla il movimento del motore. L'azionamento e gli encoder sono collegati a delle basi di collegamento con la scheda di acquisizione dati. Sulle basi convogliano tutti i segnali a bassa potenza del banco, sia in ingresso che in uscita.

L'alimentazione è fornita dalla rete elettrica che colleghiamo tramite un interruttore generale e dall'elettronica di protezione a un gruppo di trasformazione il quale ci da in uscita una corrente continua necessaria per la corretta alimentazione dei componenti.

Abbiamo infine un pulsante di emergenza da attivare in caso di pericolo per fermare l'erogazione di coppia al motore.

1.2 Motore Brushless

Il motore Brushless è un motore elettrico che possiamo descrivere come un'evoluzione del motore in corrente continua. L'idea alla base del brushless è, come dice il suo stesso nome, quella di eliminare le "spazzole". Queste sono necessarie al motore CC per commutare la corrente all'interno dei suoi avvolgimenti (che si trovano sul rotore) tramite quello che viene chiamato "collettore a lamelle".

Per eliminare le spazzole invertiamo i ruoli di statore e rotore rispetto a quelli assunti nel motore CC. La componentistica del motore si semplifica ed esso risulta costituito da:

- Tre o più coppie di avvolgimenti posti sullo statore, queste come vediamo nell'esempio sono "sfasate" fra di loro (di 120°).
- Un magnete permanente che costituisce il rotore.

Gli avvolgimenti non necessitano di un collegamento con collettore a lamelle perché in questo caso, quando il motore è in azione, trovandosi sullo statore non entrano in

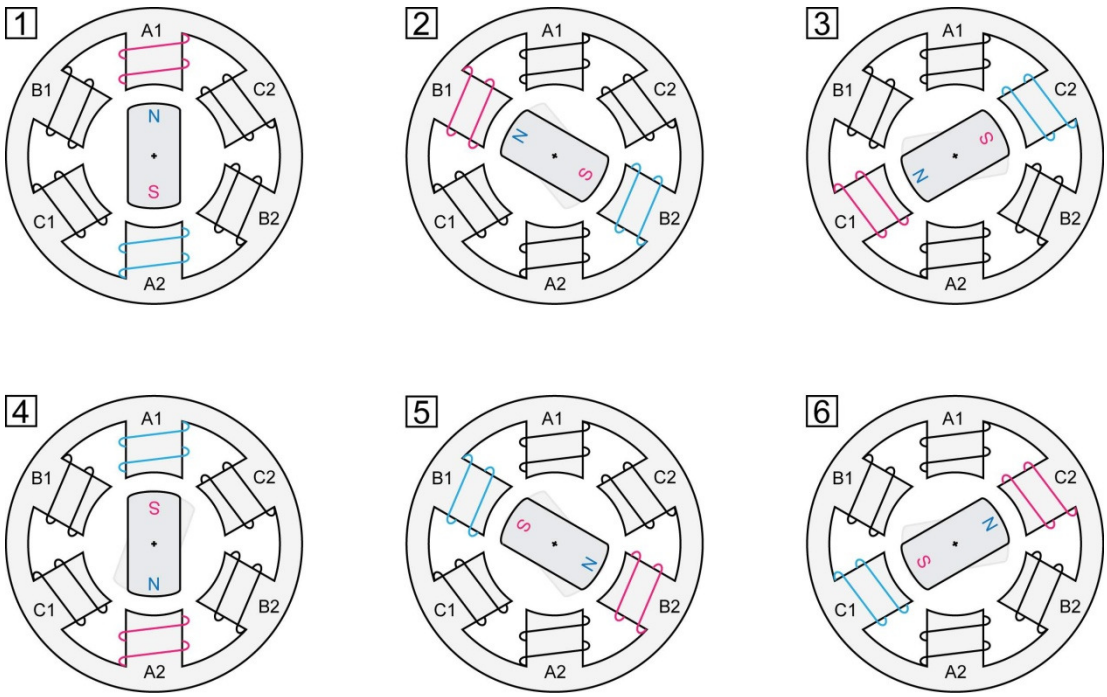


Figura 1.5 - Fasi di un motore brushless

rotazione. Grossi vantaggi di questo sistema sono la riduzione degli attriti e l'assenza delle scintille dovuto alla brusca inversione della corrente nei punti di contatto delle lamelle. Altro vantaggio è la migliore dissipazione del calore generato sullo statore.

Lo svantaggio principale di questi motori rispetto a quelli in corrente continua è che necessitano di un elettronica di controllo più sofisticata e quindi più costosa. Infatti, per creare un campo magnetico rotante, si deve commutare la corrente all'interno degli avvolgimenti in modo elettronico.

In figura 1.5 possiamo vedere uno schema delle varie "fasi" del movimento del rotore. Nella prima la bobina A1 e A2 sono alimentate in modo da generare un campo magnetico interno che guidi il rotore nella posizione indicata. Le fasi successive rappresentano la commutazione delle bobine in una sequenza che poi si ripete da capo

quando raggiungiamo l'ultima fase. L'alimentazione degli avvolgimenti è gestita come abbiamo detto dall'unità di comando.

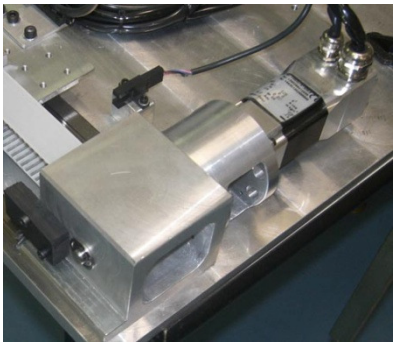


Figura 1.6 Dettaglio Motore

Possiamo vedere il motore da noi utilizziamo in figura 1.6. Il motore presenta due canali di comunicazione, uno per il comando in coppia (alimentazione) e l'altro per la lettura del resolver integrato. Questi segnali vengono letti da un'unità di comando SLVD1ND che elabora le informazioni e controlla in retroazione che il motore si comporti in modo corretto.

1.3 Encoder

Gli encoder sono dei trasduttori di posizione angolare, cioè sono dispositivi in grado di convertire un movimento di rotazione in un segnale elettrico. In figura 1.7 si può vedere un semplice schema di encoder che in generale è costituito dalle seguenti parti:

- Un disco ruotante sul proprio asse che presenta dei fori distribuiti in modo circolare sulla parte esterna.
- Uno o più LED accoppiati a un corrispondente numero di fotodiodi che permettono di leggere la presenza di un foro.

La sequenza di fori come abbiamo detto va a formare degli anelli concentrici al disco stesso. Ad ogni anello è associata una coppia LED-fotodiodo per poter leggere il segnale proveniente da esso. Il segnale in uscita dai fotodiodi è di tipo digitale e può rappresentare diversi schemi di codifica binari, il codice Gray è il più utilizzato in quanto diminuisce i problemi di lettura, poiché ogni valore differisce dal successivo di un solo bit.

Gli encoder si dividono in due categorie: encoder assoluti ed encoder incrementali. Quelli assoluti sono in grado di dirci quale è la posizione attuale leggendo un valore di n-bit dal disco. La risoluzione di questo tipo di encoder dipende dal numero di anelli (che rappresentano un bit d'informazione) disposti sul disco. Lo svantaggio maggiore è

che per aumentare la risoluzione si ha un maggiore ingombro e un elevato numero di LED-fotodiodi necessari a leggere le tracce.

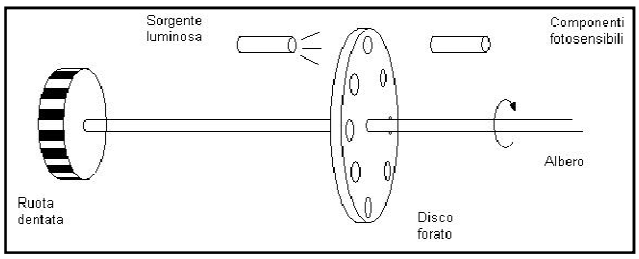


Figura 1.7 Schema di un encoder incrementale

L'altro tipo di encoder, quello incrementale, non legge un valore di posizione ma è in grado di rilevare un singolo "step" di rotazione del disco. Come si può intuire questo encoder necessita di un solo rilevatore LED-fotodiodo (due se vogliamo sapere anche in che direzione sta ruotando il disco) e quindi occupa meno spazio e richiede un circuito più semplice rispetto a quello assoluto. Per ricavare la posizione dobbiamo avere un angolo di riferimento zero che può essere fornito direttamente da una traccia con un solo foro oppure può essere stabilito dal controllore durante l'avvio del sistema tramite un procedimento di azzeramento.

Per capire se il disco sta ruotando in senso orario o antiorario si usano due anelli sfasati l'uno dall'altro di un quarto di periodo. Confrontando i due segnali possiamo capire la direzione di rotazione basandoci su quale dei due segnali risulta in ritardo.

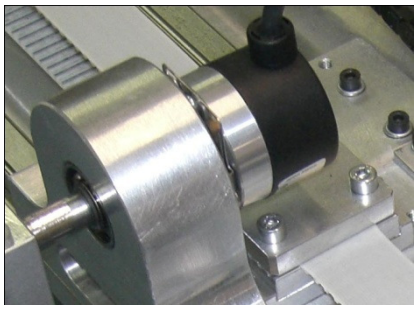


Figura 1.8 Encoder del pendolo

Nel nostro sistema faremo uso di un encoder incrementale montato sul pendolo e di un encoder virtuale fornito dal resolver integrato direttamente nel motore. Il blocco B della figura 1.4 mostra la circuiteria che si occupa di leggere i segnali degli encoder e trasmetterli come valore di posizione angolare (o posizione).

Interessante è la tecnica usata per cercare di eliminare i disturbi sulle linee di collegamento, il cui circuito è direttamente integrato nella scheda di acquisizione. Per fare ciò il segnale viene trasmesso su due linee:

- Sulla prima passa il segnale digitale cosi come esce dall'encoder (S).
- Sulla seconda trasmettiamo il segnale negato ($\overline{S}$).

Considerando che le linee sono molto vicine possiamo pensare che un eventuale disturbo (D) sarà uguale su entrambe, perciò avremo:

- Linea positiva: $S + D$.
- Linea negativa: $\overline{S} + D = -S + D$.

Durante la ricezione effettuiamo la differenza dei due segnali ottenendo un segnale che teoricamente non ha disturbi e risulta amplificato di un fattore due, infatti abbiamo che:

$$segnale = S + D - (-S + D) = 2S$$

1.4 Scheda di acquisizione

Abbiamo utilizzato per questo progetto una scheda di acquisizione dati della Sensoray, specificatamente la scheda s626.

La scheda s626 possiede i seguenti canali:

- 48 canali digitali di input ed output
- 16 canali analogici di input
- 4 canali analogici di output
- 6 canali encoder

I canali sono distribuiti su cinque jumper*:

- J1: 48 pin
- J2: 48 pin
- J3: 48 pin
- J4: 26 pin
- J5: 26 pin

Canali	Jumper	Pin
Input/Output Digitale	J2	1-48 dispari
	J3	1-48 dispari
Input Analogico	J1	4-36 pari (tranne il 20)
Output Analogico	J1	42-48 pari
Encoder	J4	1-26**
	J5	1-26**

* i pin quadrati corrispondono al numero 1

** Suddivisi in gruppi di 6 pin, corrispondenti ad A+, A-, B+, B-, I+, I-, oltre a vari pin di GROUND e +5V

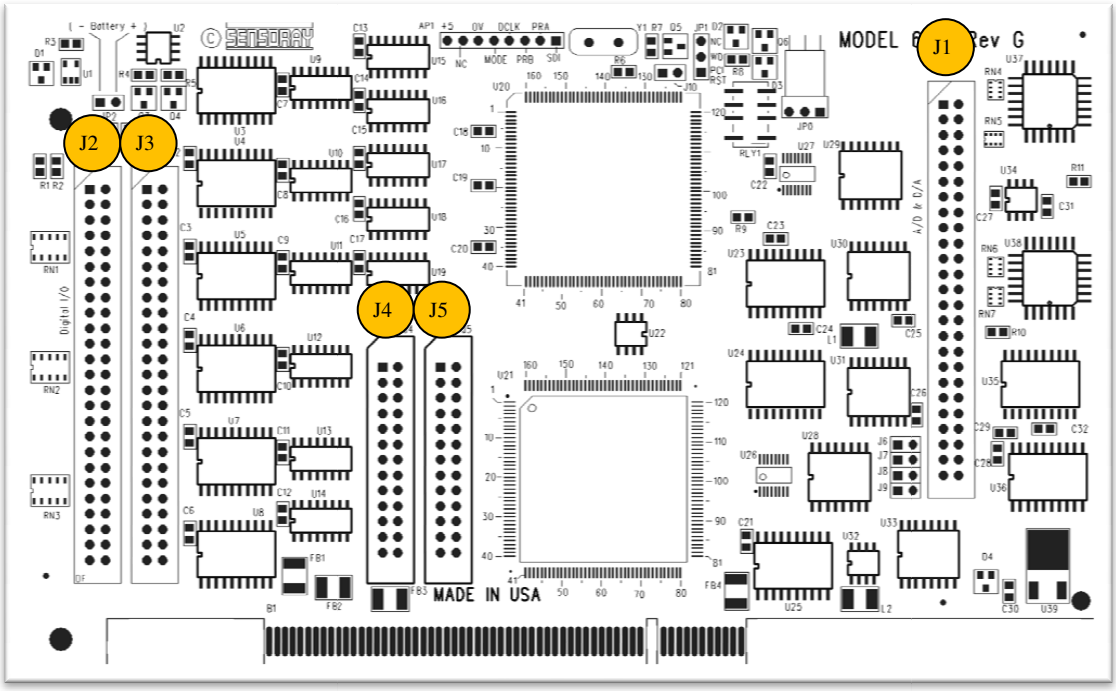


Figura 1.9 - Layout della scheda

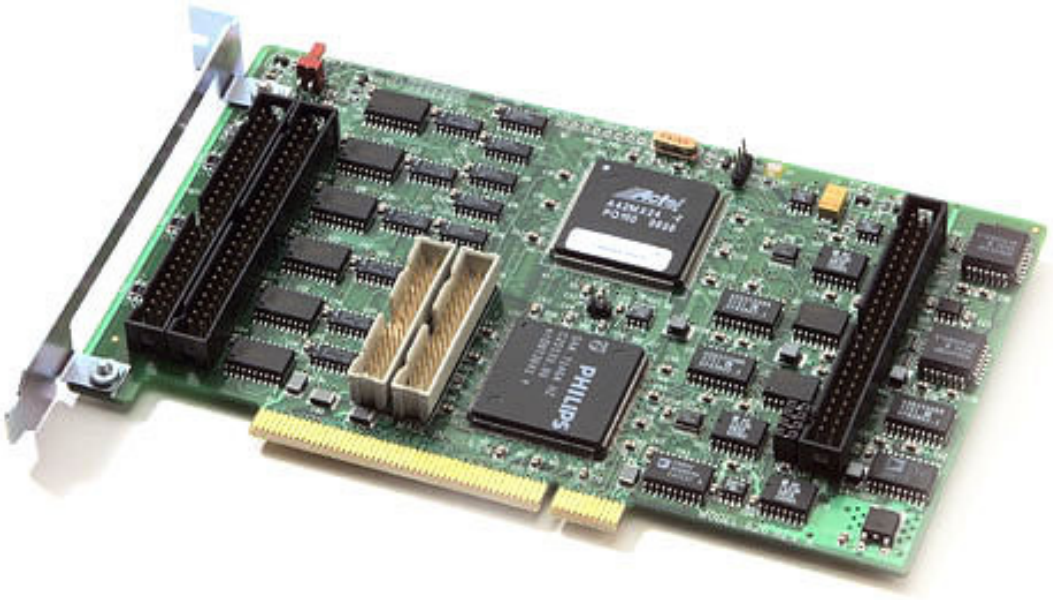


Figura 1.10 - Scheda DAQ Sensoray s626

2- Il modello

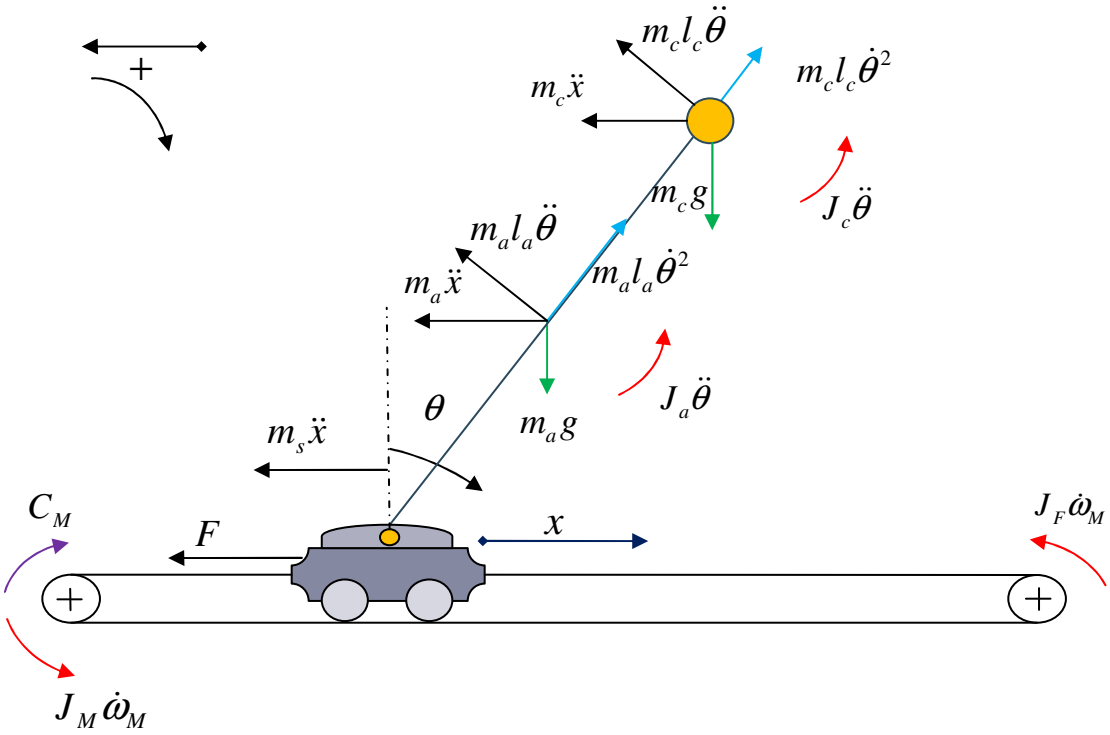


Figura 2.1 - Schema del pendolo inverso

In figura possiamo vedere lo schema del pendolo inverso da noi utilizzato con rappresentate le forze e le coppie agenti sulle componenti del sistema. Notiamo in particolare la presenza delle forze gravitazionali, delle forze di inerzia, della coppia motrice e delle coppie d'inerzia. Da questo modello semplificato possiamo ricavare le equazioni che ne descrivono completamente la dinamica in modo da poter successivamente studiare il sistema.

2.1 Dinamica del pendolo inverso

Le prime due equazioni le ricaviamo isolando il sottosistema del carrello-pendolo scrivendo il bilancio delle forze lungo l'asse x e la sommatoria dei momenti nel punto di rotazione del pendolo.

$$\sum F_x = 0 \qquad (2.1)$$

$$m_s\ddot{x} + m_a\ddot{x} + m_a l_a\ddot{\theta}\cos\theta - m_a l_a\dot{\theta}^2\sin\theta + m_c\ddot{x} + m_c l_c\ddot{\theta}\cos\theta - m_c l_c\dot{\theta}^2\sin\theta = 0$$

$$\sum M_o = 0 \qquad (2.2)$$

$$m_a l_a^2\ddot{\theta} + m_a\ddot{x}l_a\cos\theta + J_a\ddot{\theta} - m_a g l_a\sin\theta + m_c l_c^2\ddot{\theta} + m_c\ddot{x}l_c\cos\theta + J_c\ddot{\theta} - m_c g l_c\sin\theta = 0$$

Il sistema viene controllato dalla coppia del motore che non compare nelle equazioni precedenti, scriviamo quindi un bilancio di potenza tra motore e carrello in modo da ottenerla

$$C_M\omega_M - \left(J_M + J_F\right)\dot{\omega}_M\omega_M - F\dot{x} = 0 \qquad (2.3)$$

Compare nella (2.3) la velocità angolare del motore ω_M che è legata alla velocità del carrello $\dot{x}$ dal raggio della puleggia R

$$\omega_M = \frac{\dot{x}}{R}$$

Il bilancio di potenza ci fornisce quindi

$$F = \frac{C_M}{R} - \frac{\left(J_M + J_F\right)}{R^2}\ddot{x} \qquad (2.4)$$

Combiniamo la (1.1), la (2.2) e la (2.4) ricavate in precedenza e otteniamo

$$\ddot{\theta}\cos\theta\left(m_a l_a + m_c l_c\right) + \ddot{x}\left(m_s + m_a + m_c + \frac{J_M + J_F}{R^2}\right) - \dot{\theta}^2\sin\theta\left(m_a l_a + m_c l_c\right) = \frac{C_M}{R} \qquad (2.5)$$

$$\ddot{\theta}\left(m_a l_a^2 + J_a + m_c l_c^2 + J_c\right) + \ddot{x}\cos\theta\left(m_a l_a + m_c l_c\right) - g\sin\theta\left(m_a l_a + m_c l_c\right) = 0 \qquad (2.6)$$

Riscriviamo i coefficienti della (2.5) e della (2.6) in modo da semplificare la scrittura

$$K_1 = m_a l_a^2 + J_a + m_c l_c^2 + J_c$$
$$K_2 = m_a l_a + m_c l_c$$
$$K_3 = m_s + m_a + m_c + \frac{J_M + J_F}{R^2}$$

Raccogliamo la (2.6) e la (2.5) a sistema e abbiamo

$$\begin{cases} K_1\ddot{\theta} + K_2\ddot{x}\cos\theta = K_2 g \sin\theta \\ K_2\ddot{\theta}\cos\theta + K_3\ddot{x} = \dfrac{C_M}{R} + K_2\dot{\theta}^2 \sin\theta \end{cases} \qquad (2.7)$$

2.2 Stato di equilibrio

Dal sistema (2.7) ricavato passiamo al modello in forma matriciale

$$\begin{bmatrix} K_1 & K_2\cos\theta \\ K_2\cos\theta & K_3 \end{bmatrix} \cdot \begin{bmatrix} \ddot{\theta} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} K_2 g \sin\theta \\ \dfrac{C_M}{R} + K_2\dot{\theta}^2 \sin\theta \end{bmatrix} \qquad (2.8)$$

Questo sistema viene detto non lineare perché alcune delle variabili di stato presenta dei legami di tipo seno e coseno, inoltre c'è la presenza di una variabile di secondo grado.

Calcoliamo prima di tutto lo stato di equilibrio, cioè il valore costante delle variabili di stato a cui il sistema si assesterà applicando un ingresso costante, potremo così pensare di costruire un controllore che mantenga questo equilibrio stabile. Di conseguenza, le derivate delle variabili vengono poste a zero $\dot{\theta}, \dot{x}, \ddot{\theta}, \ddot{x} = 0$.

Per trovare lo stato di equilibrio del sistema non lineare, calcoliamo la variazione di energia potenziale del sistema rispetto a θ e x ed eguagliamola a zero.

$$U = m_a l_a g \cos\theta + m_c l_c g \cos\theta$$

$$\frac{dU}{d\theta} = -\sin\theta(m_a l_a g + m_c l_c g) = 0$$

$$\frac{dU}{dx} = 0$$

$$\theta = 0 \quad \text{oppure} \quad \theta = \pi \quad , \qquad \forall x \in \mathbb{R}$$

Avremo perciò due stati di equilibrio, uno per $\theta = 0$ ed uno per $\theta = \pi$, in entrambi i casi $\dot{\overline{\theta}} = \dot{\overline{x}} = 0 \quad \forall \overline{x}$

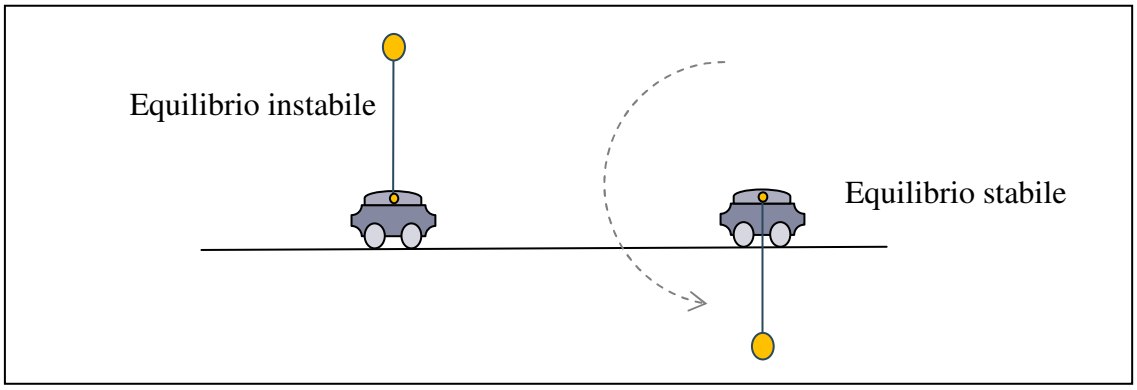


Figura 2.2 - Equilibri

Nel nostro caso, il punto di equilibrio scelto sarà la posizione da cui vogliamo che il pendolo non si sposti, cioè la posizione verticale verso l'alto per $\theta = 0$. Sappiamo già che si tratta di un punto di equilibrio instabile, in quanto possiamo subito verificare che applicando un minimo disturbo il sistema tenderà ad allontanarsi da tale stato. Per quanto riguarda la posizione della slitta, il punto di equilibrio che utilizzeremo corrisponderà al punto medio della corsa della slitta, così da evitare di raggiungere il fine corsa che comprometterebbe la stabilità.

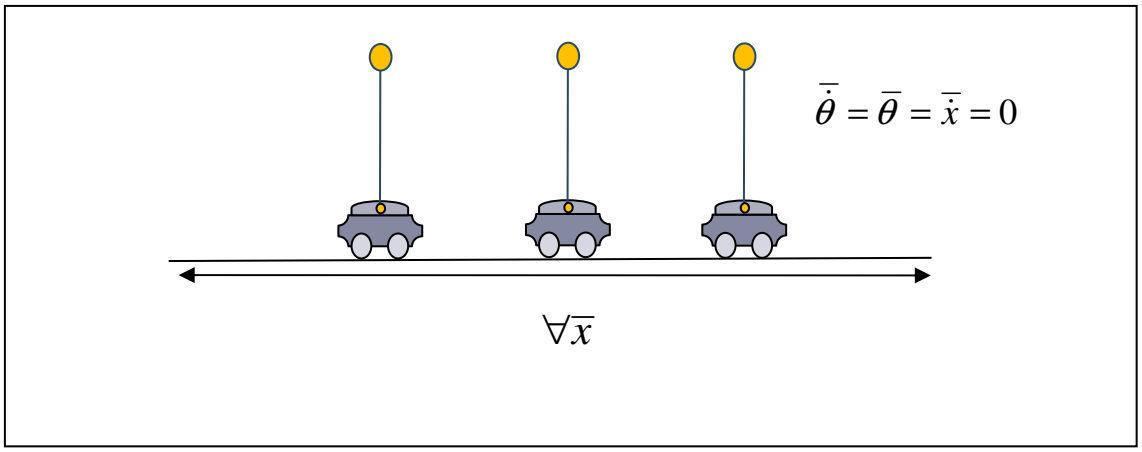


Figura 2.3 Stati di equilibrio possibili

2.3 Modello del Sistema

Procediamo dunque a linearizzare il sistema attorno al punto di equilibrio che vogliamo controllare. Per linearizzare applichiamo le seguenti trasformazioni che valgono per piccole oscillazioni del pendolo: $\cos\theta = 1$, $\sin\theta = \theta$, $\dot{\theta}^2 = 0$. Otteniamo quindi il seguente sistema:

$$\begin{bmatrix} K_1 & K_2 \\ K_2 & K_3 \end{bmatrix} \cdot \begin{bmatrix} \ddot{\theta} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} K_2 g\theta \\ \dfrac{C_M}{R} \end{bmatrix} \qquad (2.9)$$

Cerchiamo di rappresentare il sistema nella forma canonica dei controlli automatici. Consideriamo come ingresso la coppia motrice e come stati la posizione del carrello, la sua velocità, l'angolo del pendolo e la velocità angolare di rotazione, otterremo il vettore di stato z:

Partendo dallo stato $z = \begin{bmatrix} \theta & \dot{\theta} & x & \dot{x} \end{bmatrix}^T$ e dall'ingresso $u = C_m$

$$\begin{cases} \dot{z} = Az + Bu \\ y = Cz + Du \end{cases} \qquad (2.10)$$

Per il nostro progetto, ci interesserà la prima equazione, la quale esplicita il legame tra lo stato corrente del sistema e l'ingresso immesso rispetto alla variazione dello stato. Esplicitiamo per prima cosa le incognite che ci interessano, cioè $\ddot{\theta}$ e $\ddot{x}$, dalle equazioni di moto

$$\ddot{\theta} = \frac{K_2}{K_1} g\theta - \frac{K_2}{K_1} \ddot{x} \qquad \ddot{x} = -\frac{K_2}{K_3} \ddot{\theta} + \frac{C_M}{RK_3} \tag{2.11}$$

Attraverso semplici passaggi matematici e di sostituzione possiamo ricavare i seguenti legami

$$\ddot{\theta} = \frac{K_2 K_3 g}{K_1 K_3 - K_2^2} \theta - \frac{K_2}{K_1 K_3 - K_2^2} \frac{C_M}{R} \tag{2.12}$$

$$\ddot{x} = \frac{-K_2^2 g}{K_1 K_3 - K_2^2} \theta + \frac{K_1}{K_1 K_3 - K_2^2} \frac{C_M}{R} \tag{2.13}$$

Semplifichiamo ulteriormente i coefficienti in modo analogo a quanto fatto prima

$$C_1 = \frac{K_2 K_3 g}{K_1 K_3 - K_2^2} \qquad C_2 = -\frac{K_2}{K_1 K_3 - K_2^2} \frac{1}{R}$$

$$C_3 = -\frac{K_2^2 g}{K_1 K_3 - K_2^2} \qquad C_4 = \frac{K_1}{K_1 K_3 - K_2^2} \frac{1}{R}$$

Come passo finale esplicitiamo le matrici del sistema ottenuto, notando il semplice legame tra θ e $\dot{\theta}$ e tra x e $\dot{x}$.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ C_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ C_3 & 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ C_2 \\ 0 \\ C_4 \end{bmatrix} \tag{2.14}$$

2.4 Simulazione MATLab/Simulink

Durante tutta la progettazione facciamo uso del MATLab e del Simulink per poter simulare in un ambiente virtuale quello che abbiamo ricavato dalle varie sessioni di calcolo teorico. In questo modo possiamo garantire una certa sicurezza prima di applicare i nostri programmi al sistema fisico e verificare la teoria in modo più veloce ed efficace.

Il primo passo che compiamo per la simulazione è caricare tutti i parametri del modello fisico in un file script di MATLab, questo script dovrà essere richiamato all'inizio in modo da creare le costanti globali che useranno tutti gli altri script.

```
% init_pendulum.m
% Carica nel Workspace i parametri del pendolo
%
% questo script è necessario per poter usare le varie
simulazioni
% e gli altri script

% Ripulisco l'Ambiente
close all
clear all
clc

% Costanti Globali
global P g
g = 9.80665;       % Accelerazione di gravita' slm [m/s^2]

% Definizione Masse
P.ms = 1.304;                % Massa slitta [kg]
P.ma = 0.17;                 % Massa Asta [kg]
P.mc = 0.365;                % Massa Concentrata [kg]
```

```
% Definizione Lunghezze e Raggi
P.lc = 800e-3;                  % Lunghezza Asta [m]
P.la = P.lc / 2;
P.Dp = 29.40e-3;                % Diametro Pulegge [m]
P.r = P.Dp / 2;                 % Raggio Pulegge [m]

% Definizione Inerzie
P.Ja = 9010.92;                 % Momento Inerzia Ja [kgmm^2]
P.Ja = P.Ja * 10^-6;            % [kgm^2]
P.Jc = 116;                     % Momento Inerzia Jc [kgmm^2]
P.Jc = P.Jc * 10^-6;            % [kgm^2]
P.Jf = 9.048;     % Momento Inerzia Puleggia Folle [kgmm^2]
P.Jf = P.Jf * 10^-6;            % [kgm^2]
P.Jm = 12.134 + 15;       % Momento Inerzia Puleggia Motrice
                           % [kgmm^2]
P.Jm = P.Jm * 10^-6;            % [kgm^2]

% Costanti del Modello Fisico
P.K1 = P.ma * P.la^2 + P.Ja + P.mc * P.lc^2 + P.Jc;
P.K2 = P.ma * P.la + P.mc * P.lc;
P.K3 = P.ms + P.ma + P.mc + P.Jm / P.r^2 + P.Jf / P.r^2;
```

Facciamo uso di una struttura di MATLab per contenere tutti i parametri (denominata "P") mentre dichiariamo la gravità come costante a parte. I coefficienti "C" presenti nella (2.14) saranno creati nel capitolo 3 che tratta il controllo in retroazione del pendolo.

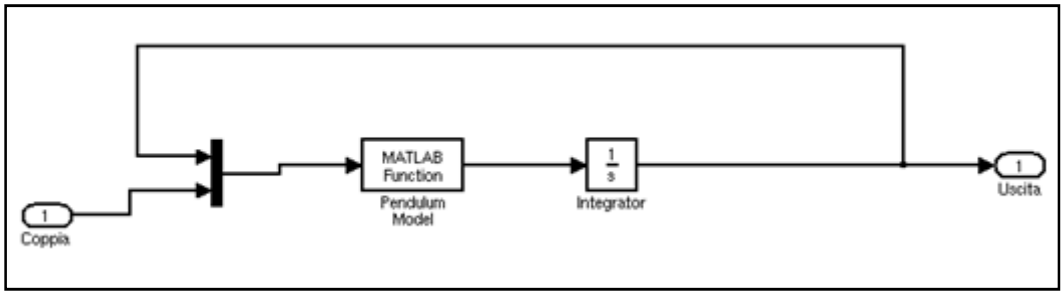


Figura 2.4 - Modello Simulink del pendolo inverso

Creiamo in simulink il blocco "Pendulum", il quale richiamerà lo script "pendulum_model.m" ed effettuerà un'integrazione tramite un blocco integratore. In quest'ultimo blocco sono dichiarate inoltre le condizioni iniziali del sistema.

```
function PM=pendulum_model(theta_p, x_p, theta, x ,Cm)
% Risolve il modello del pendolo
%
% Dati in input:
% theta_p = velocita' angolare
% x_p = velocita' del carrello
% theta = posizione angolare rispetto alla verticale (zero
rad verso
% l'alto)
% x = posizione del carrello (zero al centro)
% Cm = Coppia Motrice Applicata
%
% (x non viene utilizzata nei calcoli)

global P g

MM = [P.K1              P.K2*cos(theta)    0 0;
      P.K2*cos(theta)   P.K3               0 0;
```

```
      0                    0                    1 0;
      0                    0                    0 1];
BB  =  [P.K2*g*sin(theta)  (P.K2*theta_p^2  *  sin(theta)  +
Cm/P.r) theta_p x_p]';

PM = inv(MM) * BB;
```

Per la nostra simulazione diamo un disturbo all'angolo del pendolo di 0.1 radianti rispetto al punto di equilibrio. Otteniamo i seguenti grafici, notiamo che il pendolo, in mancanza di coppia di controllo, oscilla indefinitivamente , allontanandosi di gran lunga dallo zero.

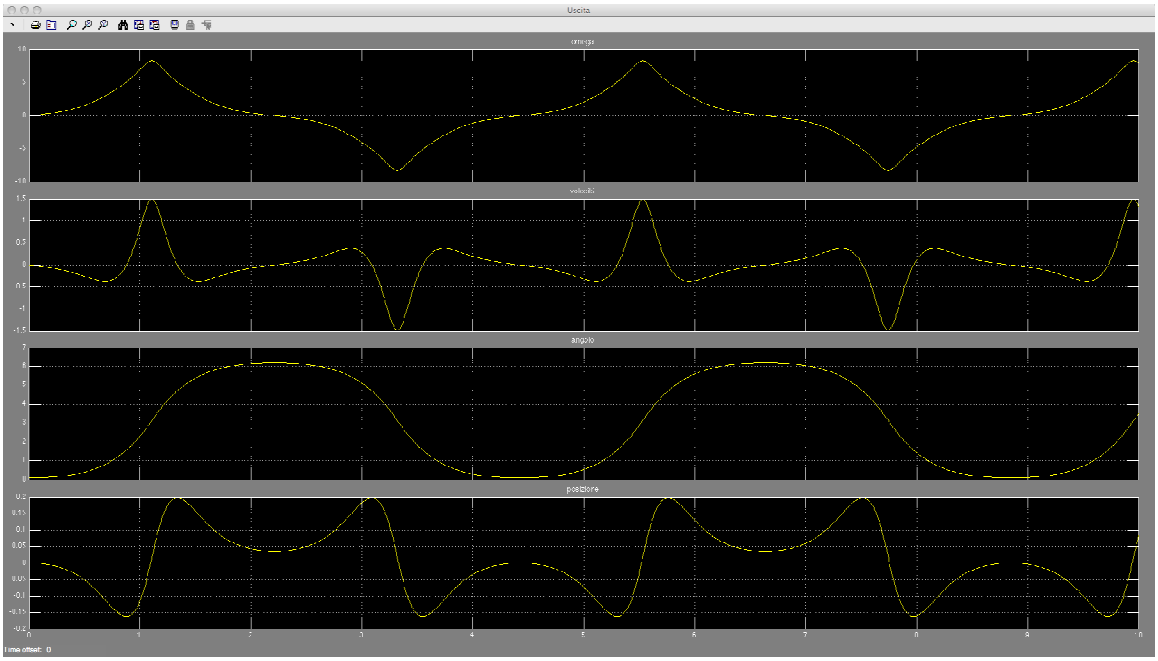


Figura 2.5 - Grafici della simulazione del pendolo

3- Controllo

Nel capitolo 1 abbiamo modellizzato il sistema slitta-pendolo inverso, scrivendo le relazioni che legano gli ingressi immessi e gli stati raggiunti dal sistema attraverso le matrici A e B:

$$\dot{X} = AX + Bu$$

$$\begin{cases} \dot{\theta} = a_{11}\theta + a_{12}\dot{\theta} + a_{13}x + a_{14}\dot{x} + b_1 C_m \\ \ddot{\theta} = a_{21}\theta + a_{22}\dot{\theta} + a_{23}x + a_{24}\dot{x} + b_2 C_m \\ \dot{x} = a_{31}\theta + a_{32}\dot{\theta} + a_{33}x + a_{34}\dot{x} + b_3 C_m \\ \ddot{x} = a_{41}\theta + a_{42}\dot{\theta} + a_{43}x + a_{44}\dot{x} + b_4 C_m \end{cases}$$

Ora che abbiamo tutte le informazioni necessarie per comprendere il sistema in ogni istante di tempo, possiamo progettare un controllore che ci permetterà di mantenere il pendolo inverso in posizione verticale.

3.1 Studio della stabilità dell'equilibrio

Il nostro sistema è linearizzato, perciò non ha senso parlare di stabilità del sistema, dovremo perciò analizzare la stabilità dell'equilibrio.

Possiamo già verificare che il nostro punto di equilibrio è instabile, in quanto possiamo controllare fisicamente che con una piccola spinta il pendolo si allontanerà dalla posizione di equilibrio. Verifichiamo comunque la stabilità dell'equilibrio attraverso il calcolo degli autovalori del sistema, i quali ci forniranno le informazioni necessarie a stabilire se l'equilibrio è stabile o instabile.

Analizziamo perciò il polinomio caratteristico della matrice $(sI - A)$

$$\det(sI - A) = \det\begin{bmatrix} s & -1 & 0 & 0 \\ -C_1 & s & 0 & 0 \\ 0 & 0 & s & -1 \\ -C_2 & 0 & 0 & s \end{bmatrix} = 0$$

$$s \cdot \det\begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s \end{bmatrix} + \det\begin{bmatrix} -C_1 & 0 & 0 \\ 0 & s & -1 \\ -C_2 & 0 & s \end{bmatrix} = 0$$

Ricaviamo il polinomio caratteristico

$$s^4 \cdot -C_1 \cdot s^2 = 0$$

$$s^2 \cdot (s^2 - C_1) = 0$$

$$s^2 \cdot \left(s - \sqrt{C_1}\right) \cdot \left(s + \sqrt{C_1}\right) = 0$$

Risolvendo il polinomio caratteristico ricaviamo gli autovalori del sistema.

$$s_1 = 0; \quad s_2 = 0; \quad s_3 = \sqrt{C_1}; \quad s_4 = -\sqrt{C_1}$$

Dalla teoria sulla stabilità dei sistemi, sappiamo che un equilibrio è asintoticamente stabile se i suoi autovalori hanno tutti parte reale, in quanto in tale caso la matrice A degli stati converge asintoticamente verso zero e la soluzione del sistema converge verso il valore asintotico. Dato che abbiamo un autovalore positivo, siamo certi che il nostro equilibrio è instabile.

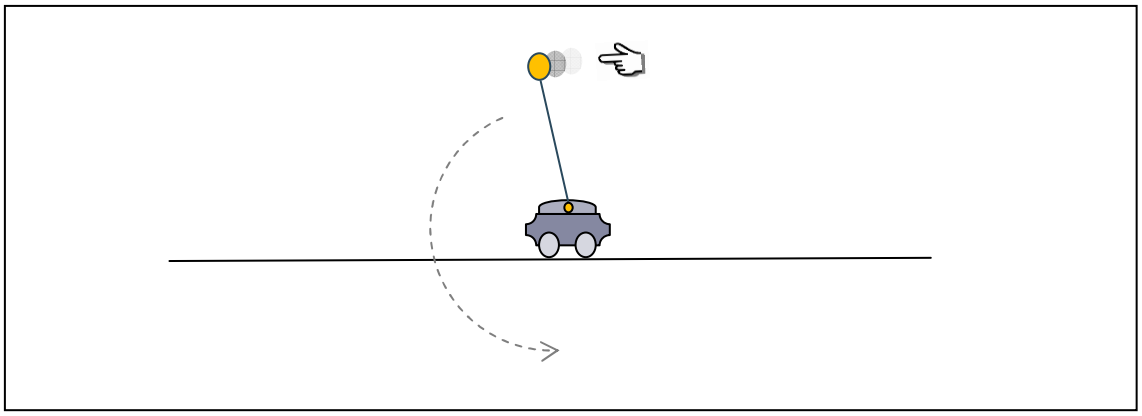


Figura 3.1 Instabilità del pendolo

3.2 Posizionamento dei poli

Per stabilizzare un sistema instabile possiamo ricorrere alla tecnica del posizionamento dei poli. Questa tecnica consiste nel controllare l'ingresso del sistema attraverso lo stato passato tramite una matrice di guadagni G, in modo da cambiare il valore dei poli del sistema, i quali come abbiamo visto determinano la stabilità. Il controllo si attua chiudendo ad anello il sistema in modo da retroazionare lo stato all'ingresso come è possibile vedere in figura 3.2.

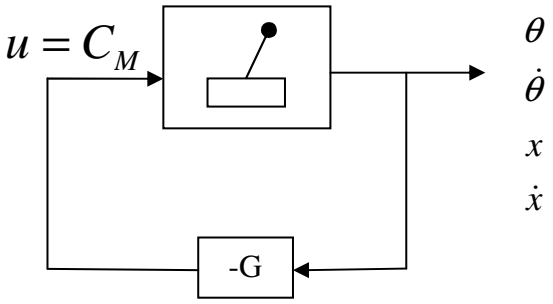


Figura 3.2 Controllore

$$C_m = -GX$$
$$\dot{X} = AX + B(-GX)$$
$$\dot{X} = (A - BG)X$$
$$\dot{X} = A_n X$$

Queste equazioni varrebbero se il sistema fosse lineare, ma noi ci concentreremo intorno al punto di equilibrio linearizzato per ovviare a questo problema.

Trasformiamo il sistema nella sua forma canonica di raggiungibilità, dove $d_i = -a_i - k_i$

$$A_n = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ d_0 & d_1 & d_2 & \dots & d_{n-1} \end{bmatrix}$$

Dalla teoria delle matrici, sappiamo che il polinomio caratteristico sarà nella forma seguente

$$\lambda^n + d_{n-1}\lambda^{n-1} + \ldots + d_1\lambda + d_0 = 0$$

Mentre il polinomio caratteristico desiderato sarà

$$\lambda^n + p_{n-1}\lambda^{n-1} + \ldots + p_1\lambda + p_0 = 0$$

Da cui otteniamo

$$p_i = a_i + k_i \quad per\ i = 0, \ldots, n-1$$

Possiamo perciò determinare i nuovi poli sommando un valore scelto ai poli di partenza.

Il nostro sistema avrà come ingresso una coppia motrice, perciò sarà quella la variabile che utilizzeremo per controllare. Il nostro controllore si occuperà di leggere lo stato del sistema e, attraverso una matrice G dei guadagni, fornirà una coppia motrice dipendente dallo stato corrente:

Per posizionare i poli, dobbiamo sceglierli in modo che siano tutti e quattro stabili. Inoltre, li posizioneremo a coppie di poli complessi coniugati, due più vicino all'asse immaginario (i quali ci forniranno i valori di velocità di risposta del sistema) e due più lontani (che saranno praticamente ininfluenti).

Scegliamo come frequenza il valore del polo negativo moltiplicato per 1.2 e come smorzamento il valore 0.7, avremo perciò il seguente grafico:

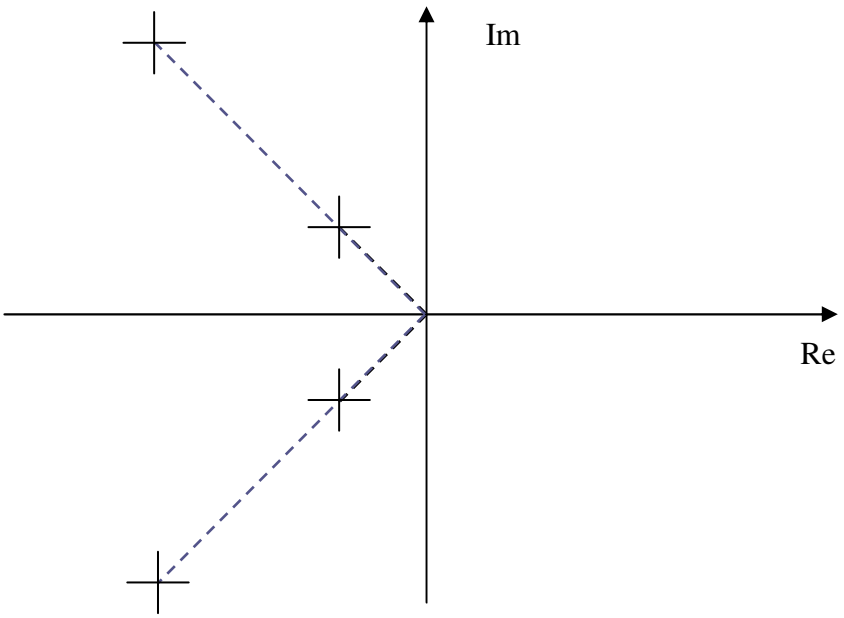


Figura 3.3 - Grafico dei poli

3.3 Calcolo della matrice dei guadagni

Con l'ausilio di MATLab calcoliamo la nostra matrice dei guadagni G, imponendo degli autovalori stabili, il che significa trovare il valore di G in modo che la matrice $\lambda I - A_n$ abbia autovalori nelle posizioni assegnate.

Prima di eseguire questo script è necessario caricare i parametri nell'ambiente di lavoro di MATLab tramite lo script init_pendulum.m visto nel capitolo 1.

```
% place_poles.m
% Riposiziona i poli del sistema pendolo
%
% crea nel Workspace la matrice "G" che contiente i
guadagni
% per stabilizzare il sistema del pendolo con il controllo
% in retroazione.
```

```
Den = 1 - ((P.K2^2) / (P.K1 * P.K3));

C1 = ((P.K2 / P.K1) * g) / Den;
C2 = -(P.K2 / (P.K1 * P.K3 * P.r)) / Den;
C3 = ((P.K2^2 / (P.K1 * P.K3)) * g) / Den;
C4 = (1/(P.r * P.K3)) / Den;

% Matrice del sistema
A = [0  1 0 0;
     C1 0 0 0;
     0  0 0 1;
     C3 0 0 0];

% Matrice degli ingressi
B = [0 C2 0 C4]';

% Poli
old_poles = eig(A);

% Nuovi Parametri
w0=1.2 * old_poles(3);
csi = cos(pi/4);

% Fattore di "Cattiveria"
factor = 2;

pole_1 = -csi * w0 + i * w0 * (1 - csi^2)^(1/2);
pole_2 = -csi * w0 - i * w0 * (1 - csi^2)^(1/2);

poles = [pole_1 pole_2 factor*pole_1 factor*pole_2];
```

```
G = place(A, B, poles);
```

3.4 Simulazione in MATLab/Simulink

Prima di provare l'algoritmo sviluppato nel mondo reale è richiesta una simulazione teorica in un ambiente virtuale. Effettuiamo tale simulazione con l'ausilio del Simulink. In figura 3.4 possiamo vedere il modello da noi utilizzato mentre in figura 3.5 i grafici della simulazione.

Colleghiamo il nostro blocco "Pendulum", il quale contiene il modello del sistema precedentemente creato, alla matrice dei guadagni in retroazione per ottenere il posizionamento dei poli.

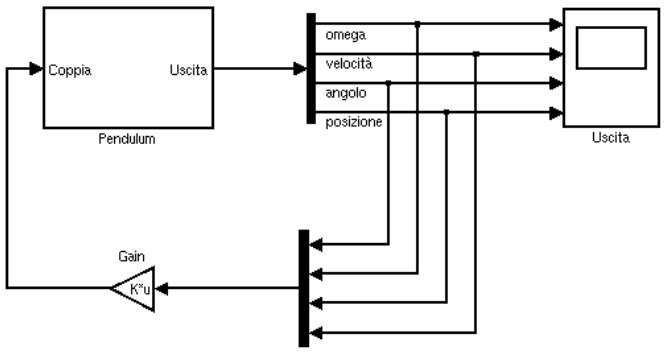


Figura 3.4 Modello Simulink

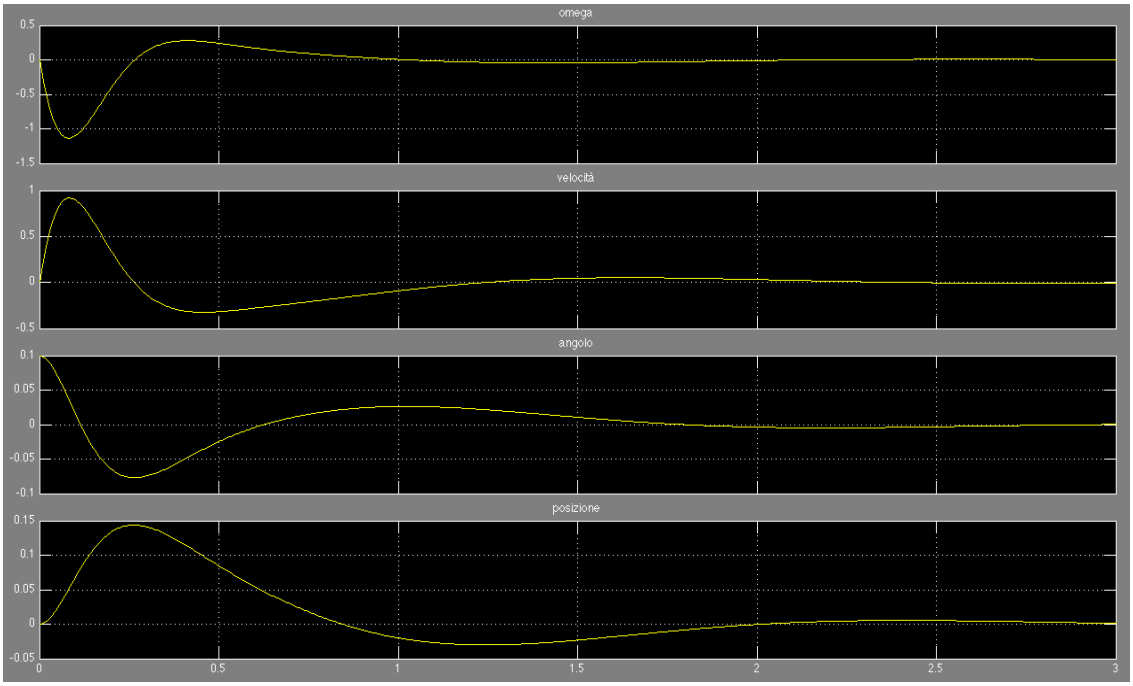


Figura 3.5 - Grafici degli stati (omega, velocità, angolo, posizione)

Dai grafici ottenuti notiamo come dopo un piccolo transitorio il pendolo torna alla posizione di equilibrio. E' da notare che il sistema fisico presenterà delle imperfezioni rispetto al nostro sistema virtuale che potrebbero compromettere i risultati.

4- Movimento

Ora che abbiamo un sistema stabile grazie al controllore, vogliamo aggiungere nuove funzioni, tra cui il movimento. Vogliamo cioè poter fornire al programma di controllo un tempo e una posizione finale e la slitta dovrà percorrere lo spazio tra il punto di partenza e quello finale entro il tempo stabilito, mantenendo il controllore attivo durante il movimento e perciò mantenendo il pendolo nella sua posizione di equilibrio. Raggiungeremo così un nuovo punto di equilibrio stabile.

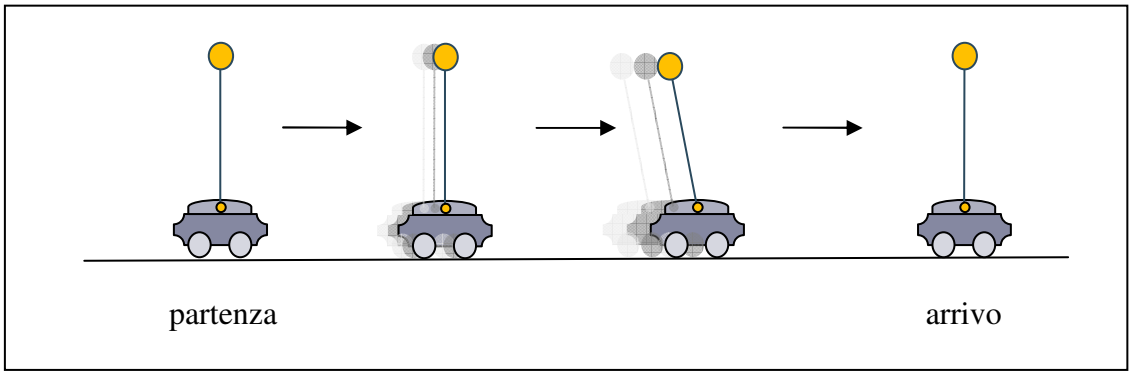


Figura 4.1 Movimento

Per effettuare tale spostamento, decidiamo di fornire alla slitta una legge di moto ad accelerazione costante, così da evitare improvvise impennate nella velocità e rischiare così di destabilizzare il sistema controllato.

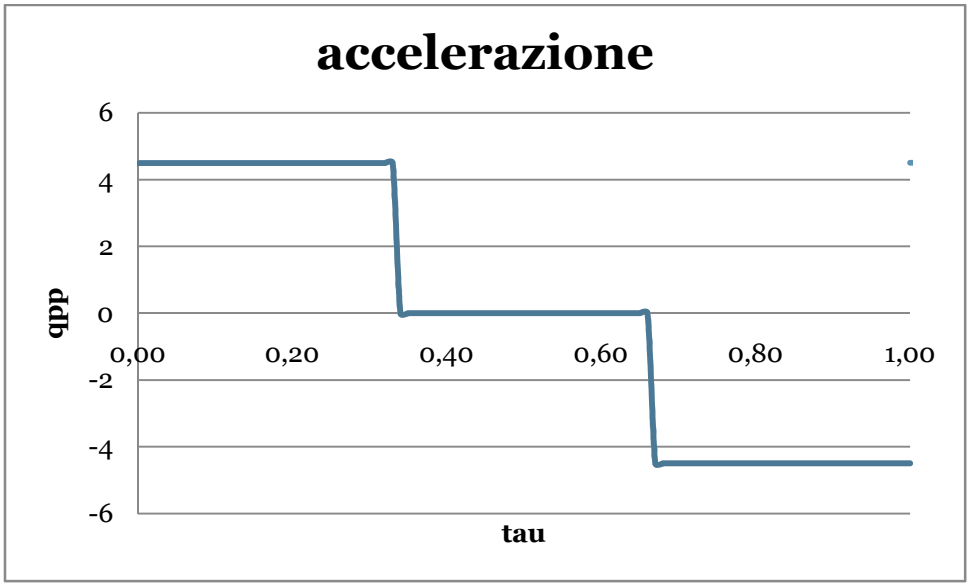
4.1 Profili di Moto

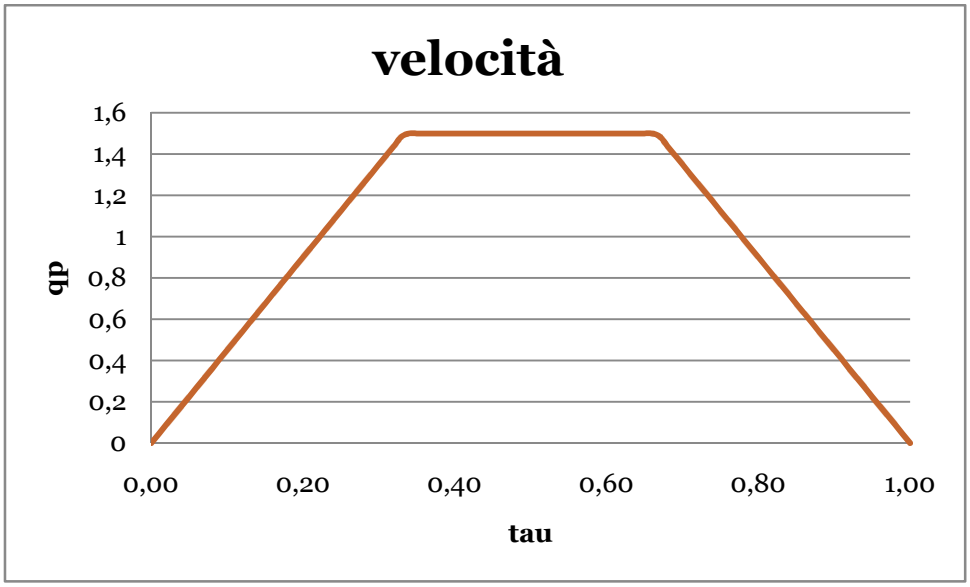
Vogliamo calcolare il nostro profilo di moto usando variabili adimensionali per il tempo (tempo attuale / periodo) e per la posizione (posizione attuale / spostamento totale), chiamate rispettivamente τ e q.

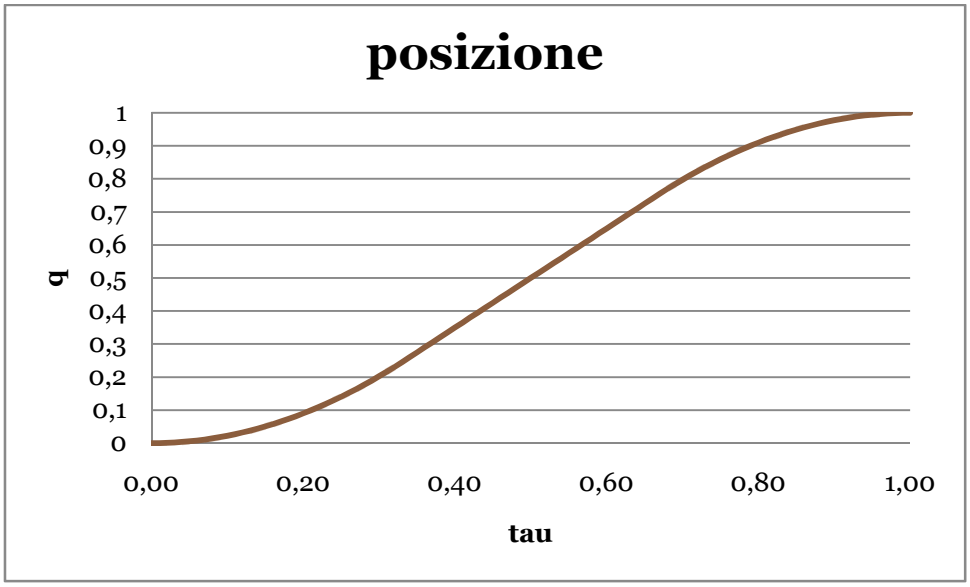
Inoltre vogliamo usare la legge d'accelerazione illustrata nel primo grafico che segue, applicando un'accelerazione fissa per un terzo del tempo, zero per un altro terzo e una accelerazione uguale alla precedente ma negativa per il terzo restante, così da avere un

integrale sul periodo nullo e perciò avere velocità nulla al tempo $\tau=1$. Gli altri grafici mostrano proprio la velocità e la posizione, ottenute derivando l'accelerazione.

Sapendo poi che le nostre variabili adimensionali q e τ variano nell'intervallo $[0,1]$, calcoliamo attraverso passaggi geometrici l'accelerazione che dobbiamo dare e, integrando, le leggi di moto.







4.2 Calcolo del profilo di moto

Per prima cosa notiamo che l'integrale della velocità tra 0 e 1 corrisponde alla posizione per $\tau = 1$, cioè $q = 1$:

$$\int_0^1 \dot{q}(\tau)\, dt = q(1) = 1$$

Sappiamo che questo integrale corrisponde all'area del trapezio sotteso alla funzione $\dot{q}(\tau)$ nell'intervallo $[0,1]$. Calcoliamo perciò l'altezza del trapezio, cioè la massima velocità $\dot{q}_{\max}$, con la formula geometrica corrispondente:

$$1 = \int_0^1 \dot{q}(\tau) = \frac{(B+b)\cdot h}{2} = \frac{\left(1+\frac{1}{3}\right)\cdot h}{2} \quad \Rightarrow \quad h = \frac{3}{2} = \dot{q}_{\max}$$

Ripetendo lo stesso ragionamento, l'integrale dell'accelerazione nell'intervallo $\left[0, \frac{1}{3}\right]$ è uguale alla velocità per $\tau = \frac{1}{3}$, cioè la $\dot{q}_{\max}$ calcolata.

$$\int_0^{1/3} \ddot{q}(\tau) = \dot{q}\left(\frac{1}{3}\right) = \dot{q}_{\max} = \frac{3}{2}$$

Questo integrale corrisponde all'area del rettangolo sotteso alla funzione $\ddot{q}(\tau)$ nell'intervallo $\left[0, \frac{1}{3}\right]$. Calcoliamo perciò l'altezza del rettangolo, corrispondente all'accelerazione massima $\ddot{q}_{\max}$:

$$\frac{3}{2} = \int_0^{1/3} \ddot{q}(\tau) = B \cdot h = \frac{1}{3} \cdot h \quad \Rightarrow \quad h = \frac{9}{2} = \dot{q}_{\max}$$

Avremo perciò la seguente legge d'accelerazione:

$$\ddot{q}(\tau) = \frac{9}{2} \quad \left(\tau < \frac{1}{3}\right); \qquad \ddot{q}(\tau) = 0 \quad \left(\frac{1}{3} < \tau < \frac{2}{3}\right); \qquad \ddot{q}(\tau) = -\frac{9}{2} \quad \left(\frac{2}{3} < \tau < 1\right);$$

Integrando la legge d'accelerazione calcoliamo la legge di velocità, sapendo che nell'intervallo $\left[\frac{1}{3}, \frac{2}{3}\right]$ la velocità è $\dot{q}_{\max}$:

$$\dot{q}(\tau) = \frac{9}{2}\tau \quad \left(\tau < \frac{1}{3}\right); \qquad \dot{q}(\tau) = \frac{3}{2} \quad \left(\frac{1}{3} < \tau < \frac{2}{3}\right);$$

$$\dot{q}(\tau) = \frac{3}{2} - \frac{9}{2}\left(\tau - \frac{2}{3}\right) = \frac{9}{2} - \frac{9}{2}\tau \quad \left(\frac{2}{3} < \tau < 1\right)$$

Integrando questa ultima legge calcoliamo la legge di moto:

$$q(\tau) = \begin{cases} \frac{9}{4}\tau^2 & \tau < \frac{1}{3} \\ \frac{1}{4} + \frac{3}{2}\left(\tau - \frac{1}{3}\right) & \frac{1}{3} < \tau < \frac{2}{3} \\ -\frac{9}{4}\tau^2 + \frac{9}{2}\tau - \frac{5}{4} & \frac{2}{3} < \tau < 1 \end{cases} \qquad (2.15)$$

Una volta trovata la nostra legge di moto adimensionalizzata, dobbiamo applicarla al nostro sistema. Per fare ciò, moltiplichiamo il risultato ottenuto per i coefficienti seguenti, con *t* istante di tempo corrente, *H* lunghezza totale del movimento e *T* tempo impiegato.

$$\tau = \frac{t}{T}$$
$$x = H \cdot q$$
$$\dot{x} = \frac{H}{T} \cdot \dot{q}$$
$$\ddot{x} = \frac{H}{T^2} \cdot \ddot{q}$$

4.3 Simulazione MATLab/Simulink

Anche in questo caso abbiamo costruito un modello in Simulink per poter simulare il comportamento del pendolo durante il movimento. Notiamo l'aggiunta, rispetto al caso precedente, della funzione di posizionamento che fornisce un offset alla posizione di controllo.

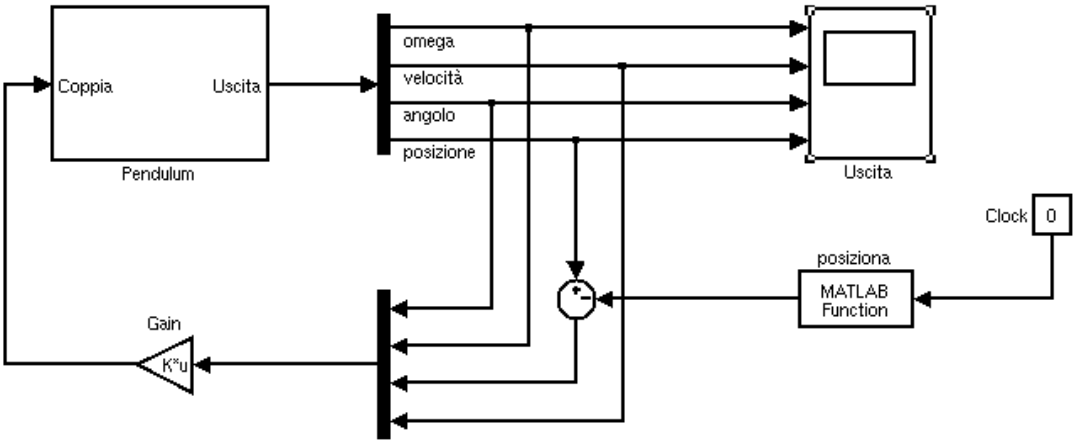


Figura 4.2 Modello Simulink

Il modello fa uso della seguente funzione di MATLab creata appositamente per il movimento

```
function [x,x_p,x_pp] = pendulum_position(H,T,t)
% Ritorna un vettore con i valori per lo spostamento del
pendolo

% Normalizzo il tempo
tau = t/T;

% Verifico in quale "terzo" mi trovo
if (tau < 1/3)
    q = 9/4 * tau^2;
    q_p = 9/2 * tau;
    q_pp = 9/2;
else
    if (tau < 2/3)
        q = 3/2 * tau - 1/4;
        q_p = 3/2;
        q_pp = 0;
    else
        if (tau <= 1)
            q = -9/4 * tau^2 + 9/2 * tau - 5/4;
            q_p = -9/2 * tau + 9/2;
            q_pp = -9/2;
        else
            q = 1;
            q_p = 0;
            q_pp = 0;
        end
    end
end

x = H * q;
x_p = H / T * q_p;
```

```
x_pp = H / (T^2) * q_pp;
```

Di seguito possiamo vedere i risultati della simulazione di un movimento di 10cm.

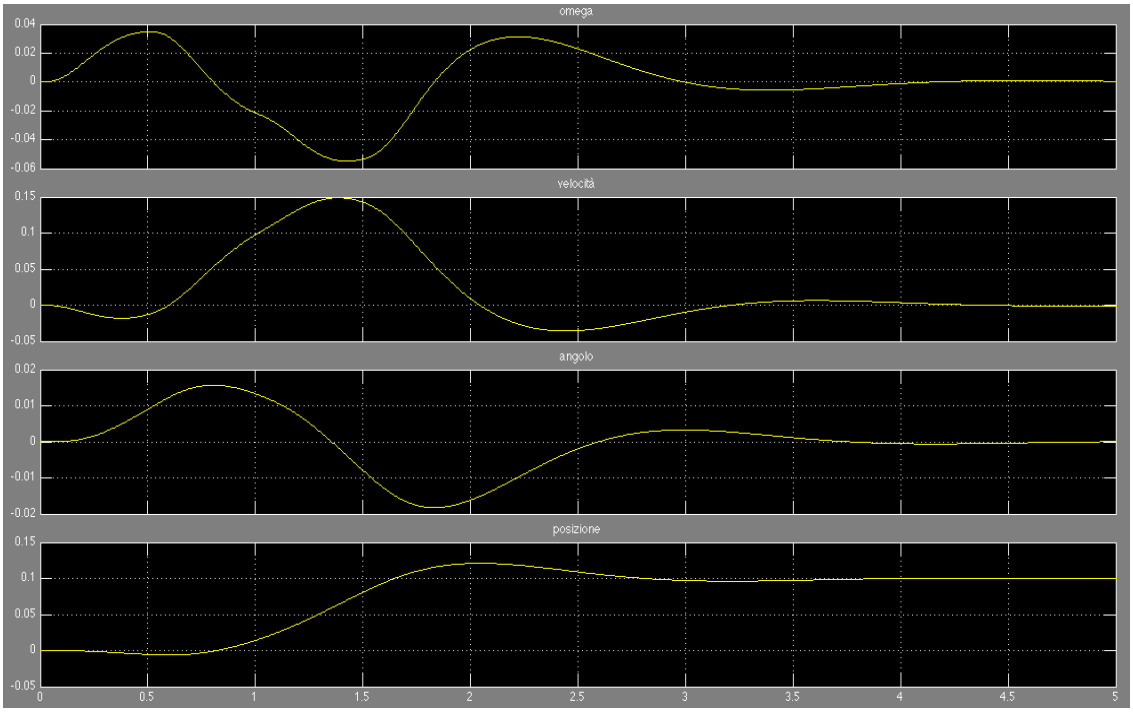


Figura 4.3 Grafici del movimento

Notiamo che l'equilibrio del pendolo è mantenuto durante tutto il movimento e che al termine della simulazione ci assestiamo sul nuovo punto di equilibrio.

A seguito di diverse simulazioni con il sistema fisico, abbiamo notato che il nostro controllore riesce a mantenere stabile il sistema anche per uno spostamento di 10 centimetri in 1 secondo.

5– RTAI

RTAI (RealTime Application Interface) è un progetto “open source” nato al Politecnico di Milano che permette la creazione e l’esecuzione di applicazioni con ristretti vincoli temporali in ambiente Linux, creando di fatto un sistema operativo real-time.

Il sistema operativo Linux non permette l’esecuzione di task con vincoli temporali, in quanto il normale funzionamento del sistema operativo prevede routines di interrupt, scelte di scheduling e pre-emption che seguono regole diverse dai sistemi real-time e perciò non possiamo essere sicuri che un processo venga eseguito entro la sua deadline, cioè il termine temporale massimo entro il quale l’esecuzione di un processo deve terminare con successo. Questo limite è mandatorio per le deadline hard, mentre è preferibile per le deadline soft.

5.1 Alternative per un sistema Linux real-time

Esistono due diversi approcci per l’implementazione di un sistema real-time in ambiente Linux:

5.1.1 Preemption improvement

Questa tecnica tende a migliorare la pre-emption del sistema operativo Linux.

In pratica ciò viene realizzato modificando l’algoritmo di pre-emption del kernel, agendo direttamente sul codice sorgente di Linux. Il metodo più comunemente usato consiste nel minimizzare il tempo di esecuzione del codice non soggetto a pre-emption.

Questo approccio ha però delle limitazioni:

- Le garanzie sulla latenza sono dettate da casi generali e non possono essere in alcun modo assolute, in quanto effettuare il controllo del codice sorgente del kernel, per via del suo continuo mutare, sarebbe proibitivo.

- La manutenzione del kernel è resa molto più difficile, in quanto chiunque voglia aggiungere del codice al kernel Linux deve tenere conto della pre-emption necessaria al funzionamento real-time del sistema.

- Le modifiche al sorgente del kernel sono ampie, ciò aumenta la possibilità di bug ed inconsistenze nel funzionamento del kernel stesso.

Gli sforzi combinati di "TimeSys" e "Linux kernel preemption project" vanno verso questa direzione.

5.1.2 Interrupt abstraction

Questa tecnica si basa sull'aggiunta di uno strato di astrazione software al di sotto del kernel Linux che gestisce le funzioni chiave del processore.

Il codice sorgente di Linux è praticamente lasciato inalterato, così che il funzionamento del kernel e dei moduli collegati, a seguito di modifiche, è sempre garantito.

Le limitazioni di questo approccio sono le seguenti:

- I processi real-time devono essere creati come moduli kernel caricabili, il che porta a difficoltà aggiuntive per il programmatore, in quanto la programmazione di moduli è diversa dalla programmazione di processi Linux e le API utilizzate differiscono dallo standard POSIX.

- I processi possono essere creati solamente in kernel-space, il che può portare a problemi in quanto il codice può portare a disturbi nell'esecuzione di altri processi, aumentando il rischio di malfunzionamenti. (Ultimamente lo sviluppo si sta muovendo verso la possibilità di processi in user-space, attraverso il modulo LXRT)

Questo approccio è sviluppato dai progetti RTLinux e RTAI.

5.1.3 Perché RTAI?

Il nostro obiettivo consiste nell'avere un sistema hard real-time, il quale ci garantisca il completo rispetto dei vincoli temporali. Questo comportamento non è garantito dall'approccio "preemption improvement", il quale è accettabile solo per sistemi soft-real time.

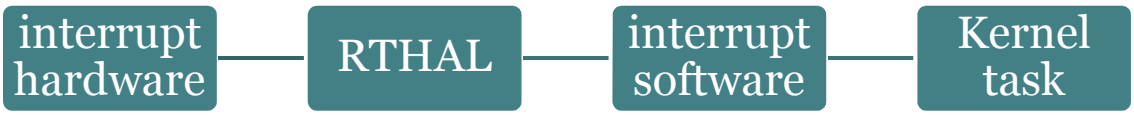
Per questo motivo, anche se prevede uno sforzo maggiore nella programmazione, optiamo per l'approccio "interrupt abstraction" ed in particolare per il progetto RTAI, in quanto il suo sviluppo è ancora attivo al giorno d'oggi.

5.2 Kernel Linux-rtai

RTAI consiste principalmente in una patch del kernel linux, la quale introduce uno strato di astrazione tra l'hardware e il sistema operativo. Nel sistema operativo così creato, lo scheduling del processore e gli interrupt sono gestiti da RTAI, non più da Linux, ciò ci permette di ignorare gli interrupt e regolare i vincoli temporali di esecuzione dei diversi processi.

L'intero kernel Linux sarà relegato ad un processo di RTAI con minima priorità, così da garantire che anche in presenza di interrupt i processi real-time giungeranno a buon fine entro i vincoli imposti. In pratica, Linux sarà considerato come "idle task", così che la sua esecuzione sarà permessa solamente quando non ci sono processi real-time in attesa.

Questo meccanismo di emulazione software dell'hardware di controllo degli interrupt garantisce che il kernel Linux non potrà mai bloccare gli interrupt o rendersi invulnerabile alla pre-emption.



I processi real-time vengono creati attraverso moduli caricabili del kernel, in modo da sfruttare il kernel-space.

RTAI fornisce una API non standard, creata appositamente per il progetto. Le funzioni fornite sono contenute in moduli kernel, i quali devono essere caricati nel kernel all'accensione del sistema per poter essere utilizzati.

Di seguito presentiamo i moduli da noi utilizzati:

- *rtai_sched.ko*: Fornisce le funzioni primarie per la creazione, l'esecuzione ed il controllo dei processi real-time. Fornisce inoltre le funzioni dei timer, dei semafori, dei messaggi , delle mailbox e delle RPC (Remote Procedure Call).
- *rtai.ko*: Fornisce le funzioni per la gestione degli interrupt (contiene il layer HAL).
- *rtai_fifos*: Fornisce le funzioni per la gestione delle FIFO.

5.2.1 Hard real-time e Soft real-time

Per il nostro progetto, abbiamo bisogno di un sistema hard real-time. La differenza tra soft e hard real-time sta nel rispetto delle deadline.

In un sistema soft real-time, il mancato completamento di un task entro la sua deadline non è consigliabile, ma è comunque tollerabile. Un esempio di sistema soft real-time può essere la riproduzione di un DVD, dove una mancata esecuzione può peggiorare la qualità del filmato, ma non pregiudica l'intero sistema.

In un sistema hard real-time, il mancato completamento di un task entro la sua deadline procura un danno irreparabile al sistema ed è perciò da evitare completamente. Un esempio di sistema hard real-time è il sistema di controllo della temperatura del nocciolo di una centrale nucleare, dove un piccolo errore porterebbe a danni enormi.

Per il nostro progetto abbiamo bisogno di un sistema hard real-time, in quanto la mancata esecuzione del task periodico del controllore pregiudicherebbe pesantemente l'equilibrio del sistema.

6 - Comedi

Il progetto Comedi (Control and Measurement Device Interface) sviluppa driver, strumenti e librerie open source per l'acquisizione di dati.

Il progetto si divide in tre parti:

Comedi è una collezione di driver open source per le schede di acquisizione dati più comuni. I driver sono implementati con un modulo kernel per Linux principale che provvede alle funzioni comuni e con specifici moduli di basso livello per ogni scheda.

Comedilib è una libreria user-space che consente di avere un interfaccia più accessibile ai device Comedi. Nella distribuzione di Comedilib è inclusa la documentazione, utility di configurazione e calibrazione e programmi dimostrativi.

Kcomedilib è un modulo kernel per Linux (distribuito con Comedi) che fornisce la stessa interfaccia (con alcune limitazioni) di Comedilib per il kernel-space, così da permetterne l'utilizzo in processi real-time.

Per il nostro progetto, utilizzeremo le API di Kcomedilib e i driver forniti da Comedi per comunicare con il sistema fisico attraverso la scheda di acquisizione dati che abbiamo scelto, la Sensoray s626.

6.1 Configurazione

Comedi permette di effettuare una configurazione della scheda e di associarla ad un device di linux. In seguito verrà eseguito l'accesso a tale scheda attraverso questo device node.

Per effettuare la configurazione, Comedi fornisce un comando chiamato `comedi_config`.

```
comedi_config /dev/comedi0 s626
```

Il primo parametro corrisponde al device node a cui viene associata la scheda, il secondo parametro è il codice della scheda, così che Comedi possa riconoscerla.

La scheda s626 non necessita di altri parametri per la configurazione, in quanto Comedi si occuperà di configurarla correttamente in modo automatico.

6.2 Device driver

Utilizziamo Comedi per ottenere i dati della scheda e la corrispondenza tra i pin sull'hardware e i canali del driver.

I device driver in Comedi sono divisi in sotto-devices, ognuno dei quali contiene più canali.

Comedi fornisce un comando, `comedi_test`, che esegue dei test sulla scheda di acquisizione, a patto che sia stata configurata precedentemente .

```
comedi_test -t info
```

Questo comando, con l'opzione `-t info`, visualizza su schermo i dati della scheda associata al device. (Di default, questo device è `/dev/comedi0`)

```
overall info:
  version code: 0x00074c
  driver name: s626
  board name: s626
  number of subdevices: 6
subdevice 0:
  type: 1 (analog input)
  flags: 0x00419000
  number of channels: 16
  max data value: 16383
  ranges:
    all chans: [-5,5] [-10,10]
```

```
  command:
    start: now|ext|int
    scan_begin: follow|timer|ext
    convert: now|timer|ext
    scan_end: count
    stop: none|count
  command fast 1chan:
    start: now 0
    scan_begin: follow 0
    convert: timer 200000
    scan_end: count 1
    stop: count 2
subdevice 1:
  type: 2 (analog output)
  flags: 0x00030000
  number of channels: 4
  max data value: 16383
  ranges:
    all chans: [-10,10]
  command:
    not supported
subdevice 2:
  type: 5 (digital I/O)
  flags: 0x00030000
  number of channels: 48
  max data value: 1
  ranges:
    all chans: [0,5]
  command:
    not supported
subdevice 3:
  type: 5 (digital I/O)
```

```
  flags: 0x00030000
  number of channels: 16
  max data value: 1
  ranges:
    all chans: [0,5]
  command:
    not supported
subdevice 4:
  type: 5 (digital I/O)
  flags: 0x00030000
  number of channels: 16
  max data value: 1
  ranges:
    all chans: [0,5]
  command:
    not supported
subdevice 5:
  type: 6 (counter)
  flags: 0x10030000
  number of channels: 6
  max data value: 16777215
  ranges:
    all chans: [0,1]
  command:
    not supported
```

Dai dati visualizzati ricaviamo il seguente rapporto tra pin sulla scheda e canali del driver:

Sotto-device	**Funzione**
Sub0	Input Analogico
Sub1	Output Analogico
Sub2	I/O Digitale (47-17 J2)

Sub3	I/O Digitale (15-1 J2 e 47-33 J3)
Sub4	I/O Digitale (31-1 J3)
Sub5	Encoder

6.3 Collegamento

Il nostro progetto prevede diversi ingressi e uscite, utilizzeremo le seguenti configurazioni:

Sotto-device	**Canale**	**Funzione**
Sub1	0	Output Analogico per la coppia immessa nel sistema
Sub3	14	Input Digitale per il pulsante di emergenza
Sub3	15	Output Digitale per l'abilitazione del driver
Sub5	0	Encoder sul motore
Sub5	1	Encoder sul pendolo

6.4 Comedi - RTAI

All'avvio del sistema real-time, sarà necessario caricare i moduli del kernel di Comedi. I moduli necessari sono i seguenti:

- `comedi.ko` – il modulo di default contenente le funzioni base di comedi.
- `comedi_fc.ko` – un modulo utilizzato dai driver.
- `kcomedilib.ko` – il modulo contenente le librerie in kernel-space.
- `s626.ko` – il modulo specifico per la scheda di acquisizione Sensoray 626

Ovviamente, l'ultimo modulo cambierà in base alla scheda di acquisizione utilizzata.

RTAI può essere configurato per garantire il supporto a Comedi. In tal caso, tra i moduli di RTAI che sarà necessario caricare all'avvio del sistema avremo anche `rtai_comedi.ko`. Questo modulo garantisce le funzioni dei moduli di Comedi per l'utilizzo con il kernel RTAI.

7 - Kernel Setup

Per ottenere un sistema operativo hard real-time, abbiamo bisogno di effettuare diverse operazioni di setup, queste operazioni sono solitamente molto delicate in quanto vanno a modificare la base stessa del sistema e di conseguenza necessitano di una cura e di una rifinitura particolare.

Per il nostro setup, abbiamo utilizzato un PC con processore Pentium 4 e la distribuzione Ubuntu Linux versione 8.10.

Nel seguente capitolo utilizzeremo le versioni dei programmi in nostro possesso. Per utilizzare gli stessi comandi con altre versioni, sarà necessario sostituire le nostre versioni con quelle più adatte.

7.1 Vanilla Kernel

Per prima cosa, scarichiamo RTAI dal sito ufficiale (www.rtai.org) . Noi abbiamo usato RTAI 3.7.

Scompattiamo ora il file con il comando `tar <file> -C <destinazione>`. Una volta scompattato il file in una cartella a nostra scelta (noi useremo(e consigliamo `/usr/src`) apriamo il seguente percorso: (cambiare x86 in caso si usi una diversa architettura)

```
cd /usr/src/rtai-3.7/base/arch/x86/patches
```

Qui troveremo una lista di patch RTAI per Kernel linux, cerchiamo la versione più recente del kernel, la quale andremo poi a scaricare. Nel nostro caso, scegliamo la patch `hal-linux-2.6.28.7-x86-2.2.06.patch`, il che significa che useremo il kernel 2.6.28.7 .

Scarichiamo ora il kernel dal sito `http://www.kernel.org/pub/linux/kernel/v2.6/` e scompattiamo

anche questo nella cartella `/usr/src`. Useremo una versione del kernel detta "vanilla", ciò sta a significare che il kernel è nella sua versione base, senza modifiche, in quanto tutte le diverse distribuzioni di Linux applicano loro modifiche al kernel. Questo accorgimento ci permetterà di avere un kernel più stabile e compatto, e ci assicurerà che la patch di RTAI non entrerà in conflitto con modifiche precedenti.

Siamo ora pronti per la creazione del kernel.

7.2 Configurazione del Kernel

Per prima cosa, accertiamoci di avere i seguenti programmi installati: `automake`, `autoconf`, `libtool`, `bison`, `doxygen`, `patch`, `libqt3-mt-dev`, `g++`. In caso contrario, scarichiamoli dai repository con il comando: `apt-get install <nome_programma>`.

Apriamo la cartella dei sorgenti del kernel /usr/src/linux-`2.6.28.7` e applichiamo la patch:

```
patch -N -p1 < /usr/src/rtai3.7/base/arch/x86/patches/hal-linux-2.6.28.7-x86-2.2.06.patch
```

La patch sarà così applicata ai sorgenti del kernel Linux per poter ospitare RTAI.

Prima di configurare il kernel, abbiamo bisogno di un file di configurazione. Possiamo scegliere fra varie alternative, tra cui usare lo stesso file di configurazione del sistema che stiamo utilizzando per la creazione del kernel oppure un file per la configurazione di default. Scegliamo invece di utilizzare un file di configurazione base, il quale sarà praticamente vuoto a parte le funzioni strettamente necessarie. In questo modo, possiamo aggiungere successivamente le funzioni che ci serviranno e non avremo un kernel troppo pesante:

```
cp arch/x86/Kconfig .config
```

Utilizziamo il comando seguente per caricare la configurazione base, dovremo rispondere a tutte le domande che ci vengono poste con N (no), Y (sì) o M (modulo),

per ora selezioniamo i default (mostrati con la lettera maiuscola, oppure premendo il tasto invio senza immettere un alcun carattere).

```
make oldconfig
```

Ora eseguiamo il comando seguente per far partire la schermata di configurazione, in cui dovremo selezionare diverse opzioni:

```
make xconfig
```

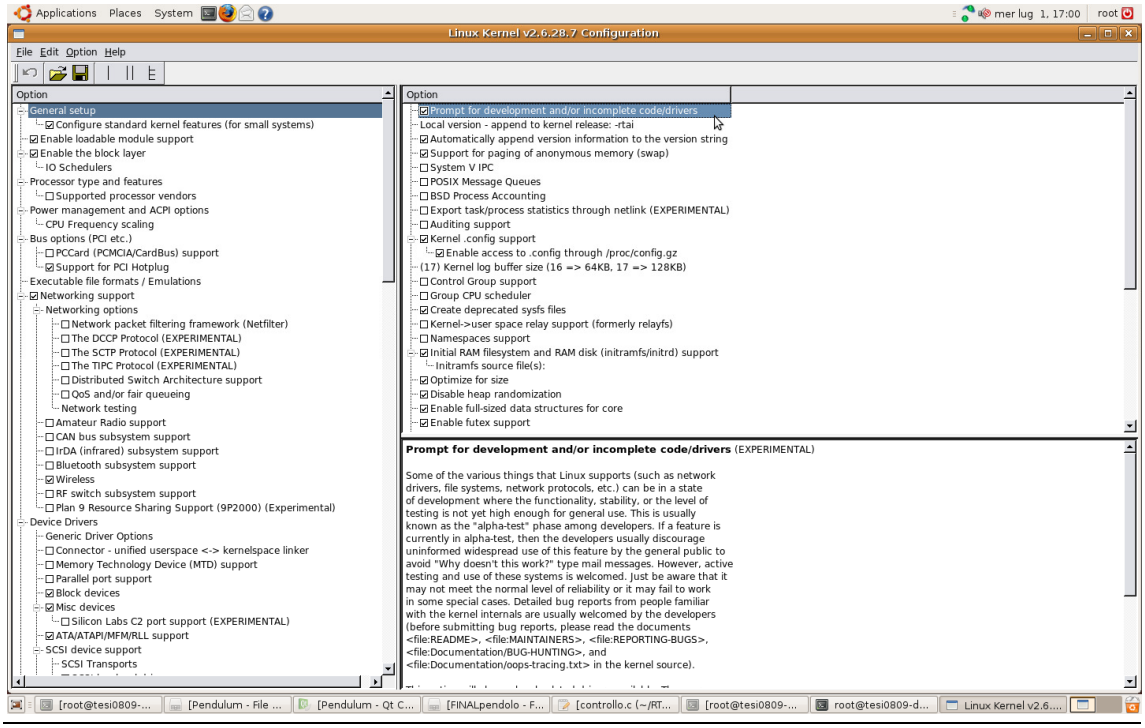


Figura 7.1 - Configurazione Kernel

Per ottenere un kernel il più piccolo possibile, abilitiamo solo lo stretto indispensabile, mettendo il segno di spunta alle seguenti opzioni:

- *General setup*
  - *Local version – append to kernel release*: inseriamo come valore "-rtai" Automatically append version information to the version string: per abilitare l'aggiunta della scritta "–rtai"

    - Initial RAM filesystem and RAM disk (initramfs/initrd) support: per abilitare l'immagine RAM di boot.
- *Enable loadable module support*
    - *Module unloading*
    - *Forced module unloading*
    - Assicuriamoci che l'opzione M*odule versioning support* non sia abilitata, in quanto fare conflitto con RTAI.
- *Processor type and features*
    - *PC compatible*
    - *Processor family:* selezionare il processore corrispondente al nostro sistema
- *Device drivers*
    - *Block device*
        - *Loopback device support*: per abilitare il mount di un file regolare come block device, utilizzato dall'initrd.

In base al chipset utilizzato, controlliamo le opzioni nei blocchi seguenti (in caso di dubbio, abilitiamo tutte le opzioni generiche di supporto):

- *ATA/ATAPI/MFM/RLL support*
- *SCSI device support*
- *SATA support*

- *File systems*
    - *Second extended fs support*
    - *Kernel automonter*
- *Kernel hacking*
    - Assicuriamoci che l'opzione *Compile the kernel with frame pointers* non sia abilitata, in quanto fare conflitto con RTAI.

Possiamo aggiungere altre funzioni al nostro kernel, come funzioni integrate oppure come moduli, in base alle nostre necessità. La differenza tra i due approcci sta nel fatto che i moduli possono essere caricati e rimossi dal kernel anche durante il suo utilizzo, ciò fornisce l'indubbio vantaggio di poter utilizzare un kernel più leggero e caricare i moduli solo quando ciò è necessario. Inoltre, in questo modo il modulo viene caricato in una zona di memoria diversa dal kernel stesso, garantendo che in caso di

malfunzionamenti non venga pregiudicato l'intero sistema (caratteristica molto importante per i sistemi real-time!)

7.3 Compilazione del Kernel

Effettuiamo ora la compilazione, i sorgenti verranno compilati per creare un immagine del kernel. Eseguiamo i seguenti comandi, che effettueranno successivamente la pulizia dei sorgenti, la compilazione del kernel, la compilazione dei moduli e l'installazione dei moduli.

```
make clean
make bzImage
make modules
make modules_install
```

Creiamo ora un'immagine initrd (initial ram disk), il quale è un piccolo filesystem montato in memoria da Linux nella fase iniziale della procedura di boot, il quale ci permetterà di caricare successivamente il kernel completo. L'initrd verrà fornito al kernel dal bootloader.

```
mkinitramfs -o /boot/initrd.img-2.6.28.7-rtai
```

Ora copiamo i file immagine e la tabella dei simboli nella cartella `/boot`.

```
cp arch/x86/boot/bzImage /boot/vmlinuz-2.6.28.7-rtai
cp System.map /boot/System.map-2.6.28.7-rtai
ln -s /boot/System.map-2.6.28.7-rtai /boot/System.map
```

7.3.1 Grub setup

Apriamo la cartella `/boot/grub` la quale conterrà il nostro bootloader GRUB, il programma che si avvierà all'accensione del PC e che ci permetterà di avviare il sistema operativo da noi scelto.

Apriamo il file `menu.lst` e scendiamo fino in fondo, vedremo delle linee di testo simili alle seguenti, le quali corrispondono ai diversi sistemi operativi caricati dal bootloader. Aggiungiamo la seguente opzione:

```
title        RTAI Ubuntu 8.10, kernel 2.6.28.7-rtai
uuid         <valore_UUID>    ← Questo valore è da copiare
dalle altre opzioni
kernel       /boot/vmlinuz-2.6.28.7-rtai
root=UUID=$<valore_UUID> ro quiet splash
initrd       /boot/initrd.img-2.6.28.7-rtai
quiet
```

Riavviamo il PC e dovremmo vedere la nuova opzione apparire nella schermata di GRUB. Selezioniamola e controlliamo che il boot del kernel vada a buon fine. In caso contrario, il problema sarà molto probabilmente nella configurazione del kernel. Sarà opportuno perciò riconfigurare il kernel con diverse opzioni, stando ben attenti alle opzioni per i file system e per i device dei dischi fissi.

7.4 Comedilib

Effettuiamo l'installazione di Comedilib, le librerie utente del progetto Comedi. Per prima cosa, scarichiamo i file sorgente di Comedilib dal sito www.comedi.org ed estraiamoli in una cartella a nostra scelta. Per il nostro progetto utilizzeremo `/usr/local/src`, cartella in cui metteremo tutti i sorgenti dei programmi utilizzati. La versione di Comedilib da noi utilizzata è la 0.8.1.

Prima di eseguire l'installazione, assicuriamoci di avere i seguenti programmi installati: `swig`, `flex`. In caso contrario, utilizziamo il comando apt-get install per procurarceli.

Successivamente, seguendo le istruzioni fornite con la distribuzione di Comedilib, utilizzeremo i comandi autogen e configure presenti nella cartella contenente i sorgenti. Infine, usiamo il comando make per compilare i sorgenti ed installare le librerie.

```
cd $PROG_SRC_DIR/comedilib
./autogen.sh
./configure --with-udev-hotplug=/lib --sysconfdir=/etc
make
make install
```

Usiamo ora il commando `make dev` per creare I device nodes nella cartella `/dev` da associare alle schede di acquisizione. Questi device nodes sono virtualizzazioni dei device hardware e ci permettono di accedere ad essi attraverso codice da Linux.

Comedilib è ora installato correttamente nel nostro sistema.

7.5 RTAI - passo primo

Configuriamo e installiamo ora RTAI, così da creare i moduli necessari ed avere a disposizione le diverse utilty.

7.5.1 Configurazione

Apriamo la cartella contenente i sorgenti di RTAI e creiamo una nuova cartella chiamata `rtai-build`, ciò è consigliato per evitare di mischiare i file sorgenti con i file creati in fase di compilazione.

```
cd /usr/src
mkdir rtai-build
cd rtai-build
```

Effettuiamo ora la configurazione di RTAI con il comando `make xconfig`, il quale ci mostrerà una schermata molto simile a quella utilizzata per configurare il kernel di Linux.

Per ora non selezioniamo nulla, controlliamo solo che la cartella dei sorgenti linux corrisponda a quella desiderata, lasciamo infine come cartella di destinazione `/usr/realtime`.

Non selezioniamo assolutamente il supporto per Comedi a questo stadio della configurazione, in quanto avremo errori di compilazione se lo facciamo. Dovremo prima installare Comedi e poi abilitarne il supporto. (Molto ironicamente, per installare Comedi avremo bisogno che RTAI sia installato.)

7.5.2 Installazione

Compiliamo ed installiamo RTAI con i seguenti comandi:

```
make
make install
```

Aggiungiamo successivamente la linea `export PATH=$PATH:/usr/realtime/bin` al file `.bashrc` presente nella home. Questa aggiunta permetterà l'utilizzo dei file eseguibili installati nella cartella `/usr/realtime`.

7.5.3 Kernel test

Ora che RTAI è installato nel nostro sistema, effettuiamo dei test per controllare che il nuovo kernel real-time non abbia problemi. Riavviamo il sistema ed avviamo il kernel realtime, poi carichiamo i moduli necessari al funzionamento di RTAI per poter effettuare il test:

```
insmod /usr/realtime/modules/rtai_hal.ko
insmod /usr/realtime/modules/rtai_up.ko
insmod /usr/realtime/modules/rtai_fifos.ko
```

Apriamo ora il percorso `/usr/realtime/testsuite/kernel/latency` ed avviamo il test con il comando `./run`. Nel nostro caso abbiamo il seguente output, anche aprendo diverse applicazioni nel frattempo:

```
## RTAI latency calibration tool ##
# period = 100000 (ns)
# avrgtime = 1 (s)
# do not use the FPU
# start the timer
# timer_mode is oneshot

RTAI Testsuite - KERNEL latency (all data in nanoseconds)
RTH|    lat min|    ovl min|    lat avg|    lat max|    ovl max|   overruns
RTD|      -1171|      -1171|       -282|      19507|      19507|          0
RTD|      -1171|      -1171|       -519|       4309|      19507|          0
RTD|      -1179|      -1179|       -338|       2464|      19507|          0
RTD|      -1166|      -1179|       -213|       3116|      19507|          0
RTD|      -1156|      -1179|       -396|        668|      19507|          0
RTD|      -1231|      -1231|       -344|       4069|      19507|          0
RTD|      -1074|      -1231|       -157|       2103|      19507|          0
RTD|      -1171|      -1231|       -355|       2368|      19507|          0
RTD|      -1179|      -1231|       -382|        661|      19507|          0
RTD|      -1177|      -1231|       -343|       4153|      19507|          0
RTD|      -1181|      -1231|       -247|        876|      19507|          0
RTD|      -1241|      -1241|       -327|       1424|      19507|          0
RTD|      -1183|      -1241|       -227|       2103|      19507|          0
```

Analizzando i dati ottenuti, notiamo subito che l'ultima colonna riporta sempre il valore zero, ciò significa che non si verificano overrun nel nostro sistema, perciò il comportamento hard real-time del kernel è garantito.

7.6 Comedi

Effettuiamo l'installazione di Comedi, scaricando i sorgenti dal sito www.comedi.org e scompattandoli nella cartella prescelta (useremo di nuovo `/usr/local/src`). Per il nostro progetto, abbiamo utilizzato la versione 0.7.76.

Così come per l'installazione di Comedilib, eseguiamo nuovamente i seguenti passaggi, dopo essere entrati nella cartella dei sorgenti di Comedi:

```
./autogen.sh
./configure --with-linuxdir=$LINUX_DIR --with-rtaidir=/usr/realtime
make
make install
```

Per permettere al comando `modprobe` di funzionare quando dovremo caricare I moduli di Comedi e Comedilib, eseguiamo il comando `depmod -a`.

L'installazione di Comedilib fornirà degli headers che verranno utilizzati per richiamare le librerie corrispondenti, specificatamente il file comedilib.h. Questo file contiene però le chiamate alle librerie user-space e, se provassimo a utilizzare le funzioni dichiarate al suo interno con un programma in user-space, avremo un errore. Comedi fornisce invece le librerie kcomedilib da utilizzare in kernel-space, dovremo perciò sostituire gli headers precedenti con questi.

```
cp include/linux/comedi.h include/linux/comedilib.h /usr/include/
cp include/linux/comedi.h include/linux/comedilib.h /usr/local/include/
```

Creiamo infine alcuni link per permettere ai nostri programmi di trovare le librerie:

```
ln -s /usr/include/comedi.h /usr/include/linux/comedi.h
```

```
ln -s /usr/include/comedilib.h
/usr/include/linux/comedilib.h
ln -s /usr/local/lib/libcomedi.a /usr/lib/libcomedi.a
```

Apriamo il file `/etc/ld.so.conf` ed aggiungiamo la linea include `/usr/local/lib`. Eseguiamo successivamente il comando `ldconfig`. Questo accorgimento permetterà alle nuove librerie di essere riconosciute.

Comedi è ora installato e configurato correttamente nel nostro sistema.

7.7 RTAI - passo secondo

Configuriamo nuovamente RTAI come abbiamo fatto precedentemente, questa volta abilitiamo però il supporto di Comedi (Comedi support over LXRT). Assicuriamoci che la cartella di Comedi indicata sia `/usr` e non `/usr/local`.

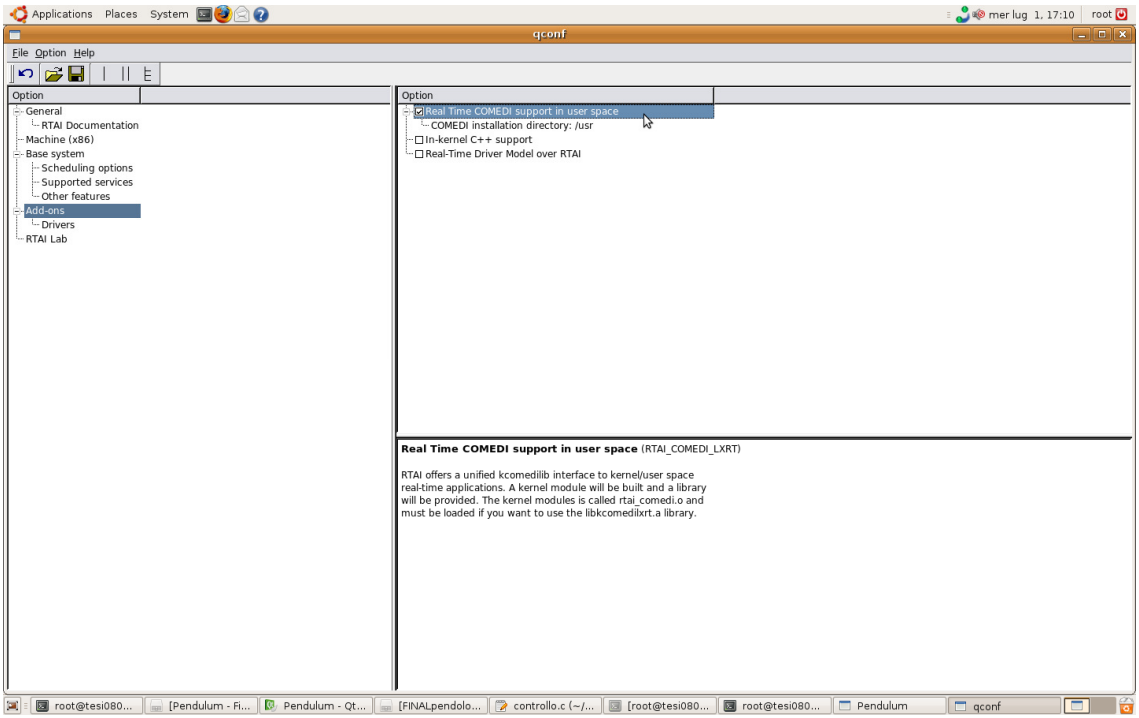


Figura 7.2 - Configurazione RTAI

Salviamo la nuova configurazione e compiliamo nuovamente RTAI. Se tutto è andato a buon fine, il sistema è ora pronto per compilare ed eseguire i nostri programmi in real-time.

7.8 Megabatch

Per rendere i nostri programmi più portabili e più facilmente utilizzabili, abbiamo creato un file batch contenente comandi per la shell di Linux. Questo file si occuperà di eseguire in automatico la preparazione del nostro sistema. L'utente avrà il solo compito di configurare il kernel in base alle specifiche della propria macchina.

Assieme al file `megabatch`, l'utente troverà una cartella denominata `packages` contenente i sorgenti necessari. In caso l'utente volesse utilizzare versioni differenti dei programmi, dovrà modificare i file nella cartella `packages` e le versioni corrispondenti nel file `megabatch`.

Il file `megabatch` è strutturato nel seguente modo:

- all'inizio delle costanti, contenenti le directory e le versioni dei programmi da compilare e installare.
- successivamente i comandi per installare tutti i prerequisiti software, scaricandoli dai repository.
- di seguito i comandi per estrarre i file sorgenti nelle cartelle designate.
- infine i comandi per eseguire l'installazione del sistema.

Durante l'esecuzione di `megabatch` si aprirà la finestra di configurazione del kernel. Una volta scelte le opzioni corrispondenti, l'utente dovrà chiudere la finestra salvando il file .config.

```
#/usr/bin/bash
### Notes ###
# - We'll put software in /usr/local/src, linux and RTAI
sources in /usr/src
#
```

```
### Constants ###
SRC_DIR="/usr/src"
PROG_SRC_DIR="/usr/local/src"
RTAI_VERSION="3.7"
RTAI_DIR="$SRC_DIR/rtai-$RTAI_VERSION"
LINUX_VERSION="2.6.28.7"
LINUX_DIR="$SRC_DIR/linux-$LINUX_VERSION"
INSTALL_DIR=$(pwd)
UUID=$(cat cmdline | awk '{print substr($0, 11,
length($0)-27)}')

### Software Requirements ###
# General
apt-get -y install automake autoconf libtool bison doxygen
# Compiling
apt-get -y install patch libqt3-mt-dev g++
# Comedilib
apt-get -y install swig flex

### Extracting Sources ###
cd $INSTALL_DIR/packages
tar xzf linux-$LINUX_VERSION -C $SRC_DIR
tar -x --bzip2 -f rtai-$RTAI_VERSION -C $SRC_DIR
tar xzf comedilib.tar.gz -C $PROG_SRC_DIR
tar xzf comedi.tar.gz -C $PROG_SRC_DIR

### RTAI kernel compilation ###

## Configuration
cd $SRCDIR
mv linux-$LINUX_VERSION linux-$LINUX_VERSION-rtai
```

```
ln -s linux-$LINUX_VERSION-rtai linux
cd $SRC_DIR/linux
patch -N -p1 < $RTAI_DIR/base/arch/x86/patches/hal-linux-
$LINUX_VERSION-x86-2.2.06.patch
#cp arch/x86/Kconfig .config
# make oldconfig
make xconfig

## Build
make clean
make bzImage
make modules
make modules_install

## Setup
mkinitramfs -o /boot/initrd.img-$LINUX_VERSION-rtai
cp arch/x86/boot/bzImage /boot/vmlinuz-$LINUX_VERSION-rtai
cp System.map /boot/System.map-$LINUX_VERSION-rtai
ln -s /boot/System.map-$LINUX_VERSION-rtai /boot/System.map

## Grub Setup
cd /boot/grub
echo '' >> menu.lst
echo '# This entry added by RTAI kernel installer' >>
menu.lst
echo "title RTAI Ubuntu 8.10, kernel
$LINUX_VERSION-rtai" >> menu.lst
echo "uuid $UUID" >> menu.lst
echo "kernel /boot/vmlinuz-$LINUX_VERSION-rtai
root=UUID=$UUID ro quiet splash" >> menu.lst
echo "initrd /boot/initrd.img-$LINUX_VERSION-rtai"
>> menu.lst
```

```
echo 'quiet' >> menu.lst

## Xorg Fix for Mouse and Keyboard
cd $TXT_DIR
cat xconf_fix.txt >> /etc/X11/xorg.conf

### Comedilib ###
cd $PROG_SRC_DIR/comedilib
./autogen.sh
./configure --with-udev-hotplug=/lib --sysconfdir=/etc
make
make install
make dev
echo 'export PATH=$PATH:/usr/realtime/bin' >> ~/.bashrc

### RTAI configuration: first step ###
# Comedi Support Won't Work

## Configuration
cd $SRC_DIR
mkdir rtai-build
cd rtai-build
make -f $RTAI_DIR/makefile xconfig

## Build
make
make install

### Comedi ###
cd $PROG_SRC_DIR/comedi
./autogen.sh
```

```
./configure --with-linuxdir=$LINUX_DIR --with-
rtaidir=/usr/realtime
make
make install
depmod -a
cp include/linux/comedi.h include/linux/comedilib.h
/usr/include/
cp include/linux/comedi.h include/linux/comedilib.h
/usr/local/include/
ln -s /usr/include/comedi.h /usr/include/linux/comedi.h
ln -s /usr/include/comedilib.h
/usr/include/linux/comedilib.h
ln -s /usr/local/lib/libcomedi.a /usr/lib/libcomedi.a

## Configure Libraries
echo 'include /usr/local/lib' >> /etc/ld.so.conf
ldconfig

### RTAI configuration: second step ###
# Add Comedi Support and RtaiLab and MATH
# Make sure Comedi Dir is /usr and NOT /usr/local

## Configuration
cd $SRC_DIR
cd rtai-build
make -f $RTAI_DIR/makefile xconfig

## Build
make
make install
```

7.9 Inizializzazione del sistema

Di seguito abbiamo lo script `setup.sh`, il quale si occuperà di ripristinare i devices nodes che potrebbero scomparire al riavvio del sistema, caricare i moduli di RTAI, Comedi e Comedilib e di configurare il device driver. Questi comandi devono essere eseguiti ad ogni avvio del sistema real-time.

```
#!/bin/bash

# rtai-inode: RTAI inode creation for UDEV systems, creates
/dev/rtf(n)
rm -f /dev/comedi* /dev/rtf* /dev/rtai_shm
for n in `seq 0 9`;
do
        rm -f /dev/rtf$n;
        mknod -m 666 /dev/rtf$n c 150 $n;
done ;
# create shared memory inode
mknod -m 666 /dev/rtai_shm c 10 254
# create Comedi inodes
for i in `seq 0 15`;
do
        rm -f /dev/comedi$i;
        mknod -m 666 /dev/comedi$i c 98 $i ;
done;

# loading modules
insmod /usr/realtime/modules/rtai_hal.ko
insmod  /usr/realtime/modules/rtai_up.ko                # or
rtai_lxrt.ko
insmod /usr/realtime/modules/rtai_fifos.ko
insmod /usr/realtime/modules/rtai_sem.ko
insmod /usr/realtime/modules/rtai_mbx.ko
```

```
insmod /usr/realtime/modules/rtai_msg.ko
insmod                  /usr/realtime/modules/rtai_netrpc.ko
ThisNode="127.0.0.1"
insmod /usr/realtime/modules/rtai_shm.ko
insmod /usr/realtime/modules/rtai_tasklets.ko
modprobe comedi
modprobe kcomedilib
modprobe comedi_fc
modprobe s626                                    # acq. card
hardware-specific
insmod /usr/realtime/modules/rtai_comedi.ko
insmod /usr/realtime/modules/rtai_math.ko    # librerie per
le funzioni matematiche

# configuring device
comedi_config -v /dev/comedi0 s626
```

8 - Implementazione

Il controllore verrà implementato attraverso un task real-time, il quale all'inizio di ogni periodo si occuperà di leggere i dati del sistema, elaborarli e fornire il comando in coppia al motore in base alla funzione desiderata.

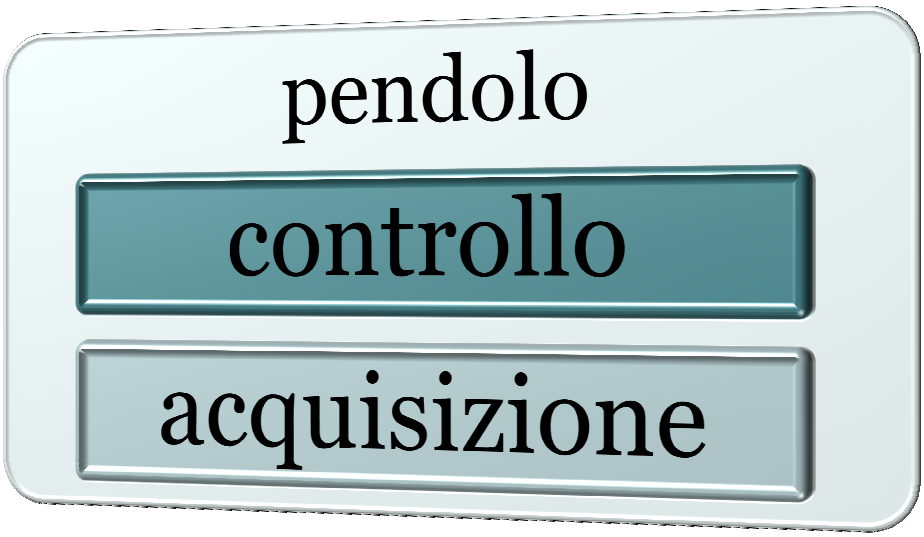
Per ottenere le funzionalità hard real-time, RTAI richiede che il programma venga caricato come modulo kernel, così da poter usufruire di uno spazio di memoria controllabile.

Abbiamo creato perciò tre moduli kernel caricabili attraverso programmazione in C. La divisione in diversi moduli è stata effettuata per ottenere una virtualizzazione del software, che è perciò composto da tre strati. Ciò ci permetterà di rendere l'applicazione più portabile, in quanto sarà necessario modificare solamente un modulo per trasferire l'applicazione su di un sistema diverso.

I moduli utilizzati sono:

- pendolo – modulo principale, contiene le funzioni di gestione dei task real-time
- controllo – modulo che implementa il controllore
- acquisizione – modulo che si occupa dell'acquisizione dei dati

Di seguito analizziamo più approfonditamente i moduli.



8.1 Modulo “pendolo”

Il modulo “pendolo” è il modulo principale contenente le funzioni per la creazione, esecuzione e distruzione del task real-time. Questo modulo si occupa anche di richiamare le funzioni dei moduli sottostanti, così da permetterne la comunicazione.

Come modulo kernel caricabile, questo modulo contiene due funzioni principali, le funzioni di inizializzazione e di chiusura del modulo, le quali verranno richiamate rispettivamente al caricamento e alla rimozione del modulo.

La funzione init_module si occupa di richiamare le funzioni di inizializzazione dei moduli di controllo e di acquisizione, creare il task real-time ed inizializzare il timer periodico.

La costante TICK_PERIOD è imposta a 10 millisecondi, il valore del periodo del nostro task real-time. TASK_PRIORITY è imposta a 1, il vaore più alto di priorità.

```
int init_module(void)
{
	// Funzione chiamata quando il modulo è inserito
	int ret =OK;
	RTIME tick_period;

	// Inizializzazione della scheda di acquisizione
	ret = init_acquisizione();
	if (ret < OK) {
		rt_printk("ERRORE: inizializzione della scheda
di acquisizione fallita.\n");
		return ERRORE;
	} else {
		rt_printk("Scheda di acquisizione
inizializzata.\n");
	}
```

```
	// Inizializzazione del controllore con vettore dei guadagni GAIN
	ret = init_control(-1.1805,-5.9737, -1.1007, -2.5818);
	if (ret < OK) {
		rt_printk("ERRORE: inizializzione del controllo fallita.\n");
		return ERRORE;
	} else {
		rt_printk("Controllo inizializzato.\n");
	}

	// Inizializzazione del task realtime
	rt_set_periodic_mode();
	rt_task_init(&task, task_handler, 1, STACK_SIZE, TASK_PRIORITY, 1, 0);
	tick_period=start_rt_timer(nano2count(TICK_PERIOD));
	rt_task_make_periodic(&task, rt_get_time() + tick_period, tick_period);

	return ret;
}
```

La funzione `cleanup_module` si occupa di fermare il timer, cancellare il task real-time e terminare il controllo.

```
//=========================================================
====================
// Funzione di chiusura del modulo

void cleanup_module(void)
{
```

```
	// Funzione chiamata quando il modulo è rimosso

	end_control(&scrittura);

	scrittura_dati(scrittura);

	stop_rt_timer(); // Ferma il timer
	rt_task_delete(&task);
	return;
}
```

All'inizio di ogni periodo, cioè ogni 10 millisecondi, verrà richiamata la funzione `task_handler`. Questa funzione si occuperà di acquisire i dati di stato del sistema e trasferirli al controllore ed infine di ricevere i dati elaborati dal controllore e trasferirli in scrittura. Al termine dell'esecuzione, la funzione `rt_task_wait_period` indicherà al task di attendere l'inizio del prossimo periodo.

Utilizzando questa funzione saremo in grado di acquisire ed inviare i dati ad un momento preciso e prefissato, la cui precisione è garantita dall'utilizzo di un sistema hard real-time.

```
static int task_handler(int t)
{
	// Funzione chiamata dal task ad ogni esecuzione
	int ret;

	while (1) {

		// Lettura
		ret = lettura_dati(&lettura);
```

```
            if (ret < OK) {
                  rt_printk("ERRORE:   Lettura   fallita.
Terminazione del task.\n");
                  cleanup_module();
                  return ERRORE;
            }

            // Controllo
            ret = controllo(lettura, &scrittura);
            if (ret == ERRORE) {
                  rt_printk("ERRORE:  Controllo  fallito.
Terminazione del task.\n");
                  cleanup_module();
                  return ERRORE;
            } else if (ret == EMERGENZA) {
                  rt_printk("ERRORE:  Emergenza  attivata.
Ueoueoueoueoueoueo! Terminazione del task.\n");
                  cleanup_module();
                  return EMERGENZA;
            }

            // Scrittura
            ret = scrittura_dati(scrittura);
            if (ret < OK) {
                  rt_printk("ERRORE:  Scrittura  fallita.
Terminazione del task.\n");
                  cleanup_module();
                  return ERRORE;
            }

            rt_task_wait_period(); // Attendi la fine del
periodo
```

```
        }
}
```

8.2 Modulo “controllo”

Il modulo di controllo si occupa di elaborare i dati ricevuti e restituire il valore di coppia da immettere nel sistema, inoltre contiene diverse funzioni di utilità. Questo modulo contiene anche le funzioni di comunicazione tra sistema real-time e sistema Linux, così da permettere un controllo utente dell’applicazione.

8.2.1 Funzione principale

La funzione principale del modulo di controllo viene richiamata dal task ad ogni periodo. Questa funzione si occupa di eseguire l’elaborazione dei dati ricevuti e di richiamare la funzione desiderata (tra azzeramento, controllo e movimento). La funzione fornisce in uscita i valori di coppia e di abilitazione.

L’esecuzione della funzione desiderata è controllata da delle variabili di stato. Ad ogni esecuzione, verrà richiamata la funzione necessaria, la quale ritornerà un valore di stato che determinerà il passo successivo.

E’ da notare il fatto che l’esecuzione del task real-time corrisponde di per sé ad un ciclo continuo, per questo motivo non verranno inseriti cicli for o while all’interno del controllo.

```
int controllo (Lettura lettura, Scrittura* scrittura)
{
      int ret;
      double coppia = 0.0;
      static        int        statoAzzeramento        =
ESEGUI_AZZERAMENTO_SLITTA;
      static int statoMovimento = ESEGUI_MOVIMENTO;
```

```
	double pos, vel, ang, ome;

	// Inizializza
	ret = controlloEmergenza(lettura.emerg);
	if (ret == EMERGENZA) return EMERGENZA;
	ret = externStato(lettura);
	if (ret < 0) return ERRORE;
	getStato(&pos, &vel, &ang, &ome, lettura.m_enc,
lettura.p_enc);

	// Controlla
	switch(gFunzione)
	{
		case FERMO:
			break;
		case AZZERAMENTO:
			if (statoAzzeramento ==
ESEGUI_AZZERAMENTO_SLITTA){
				statoAzzeramento =
autoAzzeraSlitta(&coppia);
			} else if (statoAzzeramento ==
ESEGUI_AZZERAMENTO_PENDOLO) {
				statoAzzeramento =
autoAzzeraPendolo();
			} else {
				statoAzzeramento =
ESEGUI_AZZERAMENTO_SLITTA;
				gFunzione = FERMO;
			}
			break;
		case CONTROLLO:
```

```
                    getFeedbackCoppia(pos, vel, ang, ome,
&coppia);
                    break;
             case MOVIMENTO:
                    if          (statoMovimento          ==
ESEGUI_MOVIMENTO){
                          statoMovimento   =   move(gNewPos,
gTime,  rt_get_time());
                          getFeedbackCoppia(pos, vel, ang,
ome, &coppia);
                    } else {
                          statoMovimento                   =
ESEGUI_MOVIMENTO;
                          gFunzione = CONTROLLO;
                    }
                    break;
      }

      // Concludi
      scrittura->enable = gEnable;
      scrittura->coppia = filtraCoppia(coppia);

      return ret;

}
EXPORT_SYMBOL(controllo);
```

8.2.2 Inizializzazione e finalizzazione

La funzione di inizializzazione `init_control` del controllo recupera i valori della matrice di gain del controllore, richiama l'inizializzazione dei filtri digitali e crea le FIFO di comunicazione.

La funzione di finalizzazione `end_control` chiude le FIFO di comunicazione e ferma il controllo.

La funzione `initFilters` si occupa di inizializzare i filtri. Utilizziamo dei filtri passa-basso di secondo grado con frequenza di taglio a 20Hz per l'encoder del pendolo, 80Hz per l'encoder della slitta e 8Hz per la coppia erogata. I coefficienti dei filtri sono stati calcolati con l'ausilio di Matlab, i calcoli vengono presentati nelle appendici.

```
int init_control(double ang, double ome, double pos, double
vel)
{
	int ret = OK;

	// Filtri
	initFilters();

	GAIN.ang = ang;
	GAIN.ome = ome;
	GAIN.pos = pos;
	GAIN.vel = vel;

	ret = rtf_create(FIFO_IN, 8000);
	if (ret >= OK) ret = rtf_create_handler(FIFO_IN,
handleCommands);

	if (ret >= OK) ret = rtf_create(FIFO_OUT,
sizeof(Lettura)*200);
```

```
	return ret;
}
EXPORT_SYMBOL(init_control);
```

```
int end_control(Scrittura* scrittura)
{
	int ret = OK;

	ret = rtf_destroy(FIFO_IN);
	if (ret >= OK) ret = rtf_destroy(FIFO_OUT);
	scrittura->coppia = 0;
	gEnable = 0;
	scrittura->enable = 0;

	return ret;
}
EXPORT_SYMBOL(end_control);
```

```
/*
 * Inizializzazione dei filtri
 */
void initFilters ()
{
	// filtro a 80 Hz: y[0] = -0.1855*y[2] - 0.8615*y[1]
+ 0.5117*u[2] + 1.023*u[1] + 0.5117*u[0]
	filter80Hz.coefY[0] = 0.0;
	filter80Hz.coefY[1] = -0.8615;
	filter80Hz.coefY[2] = -0.1855;
```

```
	filter80Hz.coefU[0] = 0.5117;
	filter80Hz.coefU[1] = 1.023;
	filter80Hz.coefU[2] = 0.5117;

	// filtro a 20 Hz: y[0] = -0.0521*y[2] + 0.4565*y[1]
+ 0.1489*u[2] + 0.2978*u[1] + 0.1489*u[0]
	filter20Hz.coefY[0] = 0.0;
	filter20Hz.coefY[1] = 0.4565;
	filter20Hz.coefY[2] = -0.0521;
	filter20Hz.coefU[0] = 0.1489;
	filter20Hz.coefU[1] = 0.2978;
	filter20Hz.coefU[2] = 0.1489;

	// filtro a 8 Hz: y[0] = -0.358*y[2] + 1.197*y[1] +
0.04034*u[2] + 0.08068*u[1] + 0.04034*u[0];
	filter8Hz.coefY[0] = 0.0;
	filter8Hz.coefY[1] = 1.197;
	filter8Hz.coefY[2] = -0.358;
	filter8Hz.coefU[0] = 0.04034;
	filter8Hz.coefU[1] = 0.08068;
	filter8Hz.coefU[2] = 0.04034;
}
```

8.2.3 Elaborazione dati

I dati acquisiti vengono elaborati dalla funzione `getStato`, la quale richiama le funzioni di aggiornamento del pendolo e della slitta. Queste funzioni aggiorneranno le variabili di stato del sistema trasformando i dati con funzioni di filtro e conversione. Le variabili di stato verranno utilizzate successivamente per l'esecuzione del controllo, sono perciò delle variabili globali, disponibili a tutte le funzioni del modulo. Queste variabili rimangono perciò in memoria ad ogni esecuzione e non vengono rimosse al termine di ogni periodo. Solamente a queste funzioni è permesso l'aggiornamento delle variabili di stato.

```
/*
 * Aggiorna le variabili globali associate alla slitta
 */
int aggiornaStatoSlitta(lsampl_t m_enc_in)
{
	int res = OK;

	// Dati Filtrati della Slitta
	static double u[] = {0.0, 0.0, 0.0};
	static double y[] = {0.0, 0.0, 0.0};

	u[0] = m_enc_in;
	u[0] = r2mm(u[0]); // angolare -> lineare
	if (res == OK)
	{
		filter(filter20Hz, u, y);
		gStatoPos = y[0];
		gStatoVel = (y[0] - y[1]) / DELTA_T;
	}

	return res;
}

/*
 * Aggiorna le variabili globali associate al pendolo
 */
int aggiornaStatoPendolo(lsampl_t p_enc_in)
{
	int res = OK;

	// Dati Filtrati del Pendolo
	static double u[] = {0.0, 0.0, 0.0};
```

```
	static double y[] = {0.0, 0.0, 0.0};

	u[0] = p_enc_in;
	u[0] = -u[0];
	if (res == OK)
	{
		filter(filter80Hz, u, y);
		gStatoAng = y[0];
		gStatoOme = (y[0] - y[1]) / DELTA_T;
	}

	return res;
}

/*
 * Aggiorna le variabili globali di stato
 * e ne ritorna i valori corretti in offset e modulo
 */
int getStato(double *pos, double *vel, double *ang, double
*ome, lsampl_t m_enc, lsampl_t p_enc)
{
	double angolo;
	int res = aggiornaStatoSlitta(m_enc);
	if (res == OK) res = aggiornaStatoPendolo(p_enc);

	*pos = (gStatoPos - gOffsetPos) / 1000; // mm --> m
	*vel = gStatoVel / 1000; // mm/s --> m/s

	angolo = gStatoAng - gOffsetAng;
	angolo = angolo - floor(angolo / (2 * PI)) * (2 *
PI);
	if (angolo > PI) angolo -= 2 * PI;
```

```
	*ang = angolo;

	*ome = gStatoOme;

	return res;
}
```

8.2.4 Controllo, Azzeramento ed Acquisizione

Queste funzioni eseguono il calcolo della coppia da erogare per ottenere diversi risultati. Le funzioni vengono eseguite una volta per periodo dal controllore, così da creare un ciclo con vincoli temporali precisi.

La funzione di controllo principale `getFeedbackCoppia` è abilitata quando vogliamo che il pendolo inverso sia controllato in feedback. Questa funzione calcola la coppia necessaria da immettere al sistema moltiplicando il valore delle variabili di stato per la matrice di gain ed aggiorna la coppia.

```
/*
 * Ritorna la coppia per il feedback
 * gli Algoritmi di controllo vengono implementati qui
 */
int getFeedbackCoppia (double ang, double ome, double pos,
double vel, double* coppia)
{

	pos -= gPosOscil / 1000.0;
	if (fabs(ang) < CONTROL_LIMIT)
	{
		*coppia = -1.0 * (GAIN.ang * ang + GAIN.ome *
ome + GAIN.pos * pos + GAIN.vel * vel);
	}
	return 0;
}
```

Le funzioni di azzeramento si occupano di determinare lo zero degli encoder incrementali.

Inizialmente, viene richiamata la funzione `autoAzzeraSlitta`, la quale fornirà una coppia di azzeramento per spostare la slitta prima al fine-corsa sinistro, poi al destro, salvandone le posizioni assolute. Facendo una media tra i due valori si trova perciò il punto centrale della corsa, il quale verrà preso come zero della slitta. La slitta verrà portata in tale posizione.

Al termine di questi passi la funzione `autoAzzeraSlitta` ritornerà il valore `ESEGUI_AZZERAMENTO_SLITTA`, la variabile di stato associata all'azzeramento verrà aggiornata ed il task passerà alla funzione di azzeramento del pendolo.

La funzione `autoAzzeraPendolo` lascia oscillare per un piccolo lasso di tempo il pendolo, tenendo traccia dell'angolo minimo e dell'angolo massimo raggiunti. Una media tra i due valori sommata a mezzo giro fornirà nuovamente lo zero, questa volta associato alla posizione verticale instabile del pendolo.

Due variabili di stato, una per il pendolo ed una per la slitta, terranno in memoria il valore dell'offset calcolato per lo zero dei due encoder. Questi valori verranno sommati al valore letto per ottenere la posizione corretta relativa allo zero prescelto.

```
/*
 * Aggiorna l'offset di posizionamento della Slitta
 * in modo automatico
 * ritorna ESEGUI_AZZERAMENTO_PENDOLO quando finisce
 */
int autoAzzeraSlitta(double* coppia)
{
	static int stato = 0, cont = 0;
	static double pos_sx, pos_dx;

	switch (stato)
	{
```

```
            case 0: // Init
                  rt_printk("Azzeramento della slitta in
corso...\n");
                  cont = 0;
                  stato = 1;
                  break;
            case 1: // Sinistra
                  *coppia = -COPPIA_AZZERAMENTO;
                  cont++;
                  if (cont >= 50 && fabs(gStatoVel) <
MIN_VEL)
                  {
                        // Fine Corsa Sx
                        pos_sx = gStatoPos;
                        stato = 2;
                        cont = 0;
                  }
                  break;
            case 2: // Destra
                  *coppia = COPPIA_AZZERAMENTO;
                  cont++;
                  if (cont >= 50 && fabs(gStatoVel) <
MIN_VEL)
                  {
                        // Fine Corsa Dx
                        pos_dx = gStatoPos;
                        stato = 3;
                        cont = 0;
                  }
                  break;
            case 3:
                  gOffsetPos = (pos_dx + pos_sx) / 2.0;
```

```
                    *coppia = -COPPIA_AZZERAMENTO;

                    if   (fabs(gStatoPos   -   gOffsetPos)   <
MIN_POS)
                    {
                          coppia = 0;
                          stato = 4;
                          cont = 0;
                    }
                    break;
             case 4:
                    cont++;
                    *coppia = 0;
                    if (cont >= 100)
                    {
                        stato = 0;
                        rt_printk("Azzeramento della slitta
completato!\n");
                        return ESEGUI_AZZERAMENTO_PENDOLO;
                    }
                    break;
      }
      return ESEGUI_AZZERAMENTO_SLITTA;
}

/*
 * Aggiorna l'offset dell'angolo del Pendolo
 * in modo automatico
 * ritorna FINE_AZZERAMENTO quando finisce
 */
int autoAzzeraPendolo(void)
{
```

```
	static double ang_max, ang_min;
	static int cont = 0;
	rt_printk("Azzeramento del pendolo in corso...\n");
	if (cont > (2 * ((2 * PI) / W0)) / DELTA_T)
	{
		gOffsetAng = (ang_max + ang_min) / 2 + PI;
		cont = 0;
		return 2;
	}
	else if (cont == 0)
	{
		ang_max = gStatoAng;
		ang_min = gStatoAng;
	}
	else if (gStatoAng > ang_max) ang_max = gStatoAng;
	else if (gStatoAng < ang_min) ang_min = gStatoAng;

	cont++;
	rt_printk("Azzeramento del pendolo completato!\n");
	return FINE_AZZERAMENTO;
}
```

Possiamo utilizzare la funzione azzeraManuale per azzerare gli encoder manualmente, nel caso di problemi con l'azzeramento automatico.

```
/*
 * Azzeramento Manuale
 */
void azzeraManuale(void)
{
	gOffsetPos = gStatoPos;
	gOffsetAng = gStatoAng + PI;
```

```
}
```

La funzione di movimento `move` si occupa di aggiornare una variabile di stato globale, la quale sarà un offset da sommare alla posizione della slitta. In questo modo, il controllo aggiornerà ad ogni periodo la nuova posizione di oscillazione e sposterà la slitta sulla corsa fino alla posizione desiderata.

La posizione desiderata ed il tempo di percorrimento verranno inseriti dall'utente.

```
/*
 * Sposta la posizione di oscillazione del pendolo
 * ritorna FINE_MOVIMENTO quando finisce
 */
int move (double new_pos, double time, RTIME i)
{
	static RTIME init_tick = 0;
	static int stato = 0;
	static double h, init_pos;
	double tau, q;

	switch (stato)
	{
		case 0: // Init
			init_tick = i;
			init_pos = gStatoPos;
			h = new_pos - init_pos;
			stato = 1;
			break;
		case 1: // Calcolo q
			tau = 1.0 * ((i - init_tick) * DELTA_T)
/ time;
			if (tau < 1.0/3.0)
```

```
                    {
                        q = 9.0 / 4.0 * tau * tau;
                    }
                    else if (tau < 2.0/3.0)
                    {
                        q = 3.0 / 2.0 * tau - 1.0 / 4.0;
                    }
                    else if (tau <= 1.0)
                    {
                        q = -9.0 / 4.0 * tau * tau + 9.0
/ 2.0 * tau - 5.0 / 4.0;
                    }
                    else
                    {
                        stato = 0;
                        return FINE_MOVIMENTO;
                    }

                    gPosOscil = h * q + init_pos; //
Modifico Offset
                    break;
        }

        return ESEGUI_MOVIMENTO;
}
```

8.2.5 Controllo dell'emergenza

La funzione di controllo dell'emergenza si occuperà semplicemente di fermare il controllo e l'erogazione di coppia e di chiudere l'applicazione.

```
int controlloEmergenza(lsampl_t emerg)
{
```

```
        if (emerg == 0)
        {
                // FERMA TUTTO
                gEnable = 0;
                return EMERGENZA;
        } else {
                return OK;
        }
}
```

8.2.6 Funzioni di comunicazione

Le funzioni di comunicazione si occupano della comunicazione tra task real-time e processi Linux, in pratica permette all'utente di ricevere o inserire dei dati nel sistema attraverso delle FIFO condivise. La FIFO del sistema real-time viene associata ad un block device di Linux corrispondente (rtf), così da permetterne la lettura e la scrittura da entrambi i sistemi.

La funzione di lettura dei dati `externStato` si occupa di inserire i dati ricevuti dallo stadio di acquisizione in una FIFO di output, così che essi possano essere elaborati successivamente dall'utente per ottenere dati statistici o grafici.

La funzione `handleCommands` viene richiamata ogniqualvolta la FIFO di input viene scritta. In questo modo, l'utente potrà utilizzare un programma in user-space per immettere dei comandi nel sistema. Questa funzione è utilizzata per determinare il comportamento del controllore.

```
/*
 * Scrive nella FIFO i dati letti
 */
int externStato(Lettura* lettura){
```

```
	int ret = OK;
	if ( gEnable == 1)
	{
		ret   =   rtf_put(FIFO_OUT,   &(lettura->m_enc),
sizeof(lsampl_t));
		ret   =   rtf_put(FIFO_OUT,   &(lettura->p_enc),
sizeof(lsampl_t));
		ret   =   rtf_put(FIFO_OUT,   &(lettura->emerg),
sizeof(lsampl_t));
		ret    =    rtf_put(FIFO_OUT,    &(gFunzione),
sizeof(gFunzione));
	}


	return ret;
}


/*
 * Handler della FIFO di inserimento dei comandi
 */
void handleCommands(void)
{
	Comando comando;
	rtf_get(FIFO_IN, &comando, sizeof(Comando));
	switch(comando)
	{
		case AZZERA:
			if (gEnable == 1 && gFunzione == FERMO)
gFunzione = AZZERAMENTO;
			break;
		case CONTROLLA:
```

```
                    if (gEnable == 1 && gFunzione == FERMO)
gFunzione = CONTROLLO;
                    break;
             case MUOVI:
                    if (gEnable == 1 && gFunzione ==
CONTROLLO) gFunzione = MOVIMENTO;
                    rtf_get(FIFO_IN,&gNewPos, sizeof(int));
                    rtf_get(FIFO_IN, &gTime, sizeof(int));
                    break;
             case ABILITA:
                    gEnable = 1;
                    break;
             case DISABILITA:
                    gEnable = 0;
                    gFunzione = FERMO;
                    break;
             case FERMA:
                    gFunzione = FERMO;
                    break;
             case AZZERA_MANUALE:
                    azzeraManuale();
                    break;
      }
}
```

8.2.7 Funzioni ausiliarie

Abbiamo infine delle funzioni secondarie di utilità: funzioni di conversione per ottenere dati nell'unità di misura desiderata e funzioni di filtro per ottenere dei dati filtrati.

```
//=====================================================
====================
```

```
// Funzioni di Conversione

/*
 * Conversione da radianti a millimetri
 */
double r2mm (double radianti)
{
	return (radianti * (RAGGIO));
}

/*
 * Conversione da radianti a gradi
 */
double r2g (double radianti)
{
	return (radianti / (2.0 * PI) * 360.0);
}



//=========================================================
=====================
// Funzioni di Filtro

/*
 * Funzione Filtro
 */
void filter(Filter f, double u[3], double y[3])
{
	int i;

	// Shift Uscite
```

```
	y[2] = y[1];
	y[1] = y[0];

	// Calcola uscita filtrata
	y[0] = 0.0;
	for (i = 1; i < 3; i++) y[0] += f.coefY[i]*y[i];
	for (i = 0; i < 3; i++) y[0] += f.coefU[i]*u[i];

	// Shift Ingressi
	u[2] = u[1];
	u[1] = u[0];
}

/*
 * Filtra la coppia del motore
 */
double filtraCoppia(double coppia)
{
	// Dati Filtrati
	static double u[] = {0.0, 0.0, 0.0};
	static double y[] = {0.0, 0.0, 0.0};

	u[0] = coppia;
	filter(filter8Hz, u, y);
	return y[0];
}
```

8.3 Modulo "acquisizione"

Il modulo che si occupa dell'acquisizione contiene le funzioni che comunicano direttamente con la scheda di acquisizione dati. Per questo motivo, nel caso si volesse utilizzare una scheda differente, bisognerebbe modificare solamente questo modulo.

I canali di acquisizione corrispondenti ai pin che utilizzeremo sono salvati in alcune variabili all'inizio del file, rendendone immediata la modifica.

Questo modulo utilizza le librerie kcomedilib per l'acquisizione dei dati.

```
// Canali di acquisizione

comedi_t* device;

int range = 0;
int aref = AREF_GROUND;


// Encoder Motore COUNTER
int m_enc_subdev = 5;
int m_enc_chan = 0;
double m_maxdata;

// Encoder Pendolo COUNTER
int p_enc_subdev = 5;
int p_enc_chan = 1;
double p_maxdata;


// Pulsante Emergenza DIGITAL INPUT
int emerg_subdev = 3;
int emerg_chan = 14;

// Attivazione "Enable" DIGITAL OUTPUT
```

```
int enable_subdev = 3;
int enable_chan = 15;

// Coppia ANALOG OUTPUT
int coppia_subdev = 1;
int coppia_chan = 0;
```

La funzione di inizializzazione si occupa di aprire il device node creato precedentemente al setup del sistema e di inizializzare i canali di comunicazione necessari, configurando i canali digitali per l'input o per l'output ed utilizzando un'istruzione di configurazione per inizializzare gli encoder. Salviamo inoltre il valore massimo che l'encoder ci può fornire così da poter implementare una conversione in complemento a 2.

```
int init_acquisizione()
{
	int ret;

	// Inizializzazione del device per la scheda
	device=comedi_open("/dev/comedi0");

	// Inizializzazione dei canali di comunicazione
	ret = comedi_dio_config(device, emerg_subdev, emerg_chan, COMEDI_OUTPUT);
	comedi_dio_write(device, emerg_subdev, emerg_chan, 0);
	if (ret >= OK) ret = comedi_dio_config(device, emerg_subdev, emerg_chan, COMEDI_INPUT);

	if (ret >= OK) ret = comedi_dio_config(device, enable_subdev, enable_chan, COMEDI_OUTPUT);
```

```
	// Configurazione degli encoder
	instr.insn=INSN_CONFIG;
	instr.n=1;
	instr.data=&initvalue;
	instr.subdev=p_enc_subdev;
	instr.chanspec=CR_PACK(p_enc_chan,range,aref);
	comedi_do_insn(device, &instr);

	instr.insn=INSN_CONFIG;
	instr.n=1;
	instr.data=&initvalue;
	instr.subdev=m_enc_subdev;
	instr.chanspec=CR_PACK(m_enc_chan,range,aref);
	comedi_do_insn(device, &instr);

	// Valore massimo degli encoder
	p_maxdata = comedi_get_maxdata(device, p_enc_subdev,
p_enc_chan);

	m_maxdata = comedi_get_maxdata(device, m_enc_subdev,
m_enc_chan);

	return ret;

	return ret;
}
EXPORT_SYMBOL(init_acquisizione);
```

La funzione di lettura dei dati utilizza la funzione `comedi_dio_read` delle librerie kcomedlib per ottenere il dato dell'ingresso ditigale di emergenza. Utilizziamo delle

istruzioni di read per ottenere i dati degli encoder, inoltre facciamo una conversione da step dell'encoder in radianti.

```
int lettura_dati(Lettura* lettura)
{
	int ret;
	lsampl_t val;
	double rad;

	// Encoder Motore
	instr.insn = INSN_READ;
	instr.n = 1;
	instr.data = &val;
	instr.subdev = m_enc_subdev;
	instr.chanspec=CR_PACK(m_enc_chan,range,aref);
	ret = comedi_do_insn(device, &instr);

	if (val <= m_maxdata / 2) rad = val;
	if (val > m_maxdata / 2) rad = (val - m_maxdata);
	rad = 1.0*rad/256*2*PI;
	lettura->m_enc = rad;

	// Encoder Pendolo
	instr.insn = INSN_READ;
	instr.n = 1;
	instr.data = &val;
	instr.subdev = p_enc_subdev;
	instr.chanspec=CR_PACK(p_enc_chan,range,aref);
	if (ret >= OK) ret = comedi_do_insn(device, &instr);

	if (val <= p_maxdata / 2) rad =val;
	if (val > p_maxdata / 2) rad = (val - p_maxdata);
```

```
	rad = 1.0*rad/1024*2*PI;
	lettura->p_enc = rad;

	// Emergenza
	if (ret >= OK) ret = comedi_dio_read(device,
emerg_subdev, emerg_chan, &(lettura->emerg));

	return ret; }
EXPORT_SYMBOL(lettura_dati);
```

La funzione di scrittura dei dati utilizza le funzioni `write` di kcomedilib per immettere il valore di coppia e l'abilitazione al sistema. Il valore della coppia è trasformato prima in voltaggio, poi nel suo valore in bit corrispondente alla scheda utilizzata.

```
int writeCoppia(double coppia)
{
	int ret;
	double volt;
	lsampl_t maxdata;

	// Conversione Coppia -> Volt
	volt = (0.8 * coppia / 0.46) * (9.76 / 2.25);

	// Saturazione
	if (volt > 9.76) volt = 9.76;
	else if (volt < -9.76) volt = -9.76;

	// Conversione Volt -> bit
	maxdata = comedi_get_maxdata(device, coppia_subdev,
coppia_chan);

	volt = ((volt+9.76) * maxdata) / (9.76*2);
```

```
	ret = comedi_data_write(device, coppia_subdev,
coppia_chan, range, aref, volt);

	return ret;

}

/*
 * Scrittura dei dati dalla scheda
 */
int scrittura_dati(Scrittura scrittura)
{
	int ret;

	// Driver Enable
	ret = comedi_dio_write(device, enable_subdev,
enable_chan, scrittura.enable);

	// Coppia
	ret = writeCoppia(scrittura.coppia);

	return ret;
}
EXPORT_SYMBOL(scrittura_dati);
```

8.4 Strutture dati

All'interno dei nostri programmi, utilizziamo delle strutture dati comuni tra i diversi moduli. Per evitare incongruenze a causa di modifiche successive, queste strutture condivise sono raccolte in un file a parte chiamato `strutture.h`.

Le strutture che utilizziamo sono:

- Lettura: rappresentante gli ingressi
- Scrittura: rappresentante le uscite
- Comando: tipo enumerativo che elenca i comandi che possiamo inserire
- Stato: tipo enumerativo che elenca gli stati in cui il controllore può trovarsi

```
#ifndef _RTAI_COMEDI_H_
typedef unsigned int lsampl_t;
#endif

typedef struct Lettura
{
	lsampl_t m_enc;
	lsampl_t p_enc;
	lsampl_t emerg;
}Lettura;

typedef struct Scrittura
{
	lsampl_t enable;
	double coppia;
}Scrittura;

typedef enum {AZZERA, CONTROLLA, MUOVI, DISABILITA,
ABILITA, FERMA, AZZERA_MANUALE, ESCI = 9} Comando;

typedef enum {AZZERAMENTO, CONTROLLO, MOVIMENTO, FERMO}
Stato;

const int EMERGENZA = -2;
```

```
const int ERRORE = -1;
const int OK = 0;
```

8.5 Makefile

Per compilare i moduli creati, utilizziamo il comando `make`, il quale permette di compilare più sorgenti in modo efficiente attraverso dei comandi contenuti all'interno di un Makefile.

La struttura del Makefile prevede della flag e dei comandi.

Le flag vengono inserite all'inizio del file e hanno la forma:

`<flag> += <valore>`

In base ai comandi utilizzati, le flag assumeranno un determinato significato.

La flag `obj-m` indica che il Makefile è un kbuild Makefile, utilizzato per la creazione del kernel linux. La flag indica al compilatore gli oggetti da creare, il valore `-m` indica che gli oggetti saranno dei moduli kernel, a differenza del valore `-y` che indica che gli oggetti saranno incorporati nel kernel.

La flag `EXTRA_CFLAGS` ci permette di aggiungere dei parametri al compilatore stesso, in questo caso la dicitura `-I` indica i path aggiuntivi da cui verranno caricati gli headers, mentre –mhard-float indica un'opzione per l'utilizzo dei valori float (necessario per il nostro modulo).

I comandi sono inseriti con la seguente forma:

`<sezione> : <comandi>`

Le sezioni possono essere richiamate singolarmente se viene passato il nome della sezione come parametro al comando `make`.

La sezione `all` esegue a sua volta il comando `make modules` nel Makefile della creazione del kernel per il sistema corrente, passando come cartella dei moduli la cartella corrente.

La sezione `clean` cancella i file creati, così da permettere una nuova compilazione.

```
obj-m += pendolo.o acquisizione.o controllo.o
EXTRA_CFLAGS += -I/usr/realtime/include -I/usr/include/ -
I/usr/local/include -mhard-float
```

```
all:
	make -C /lib/modules/$(shell uname -r)/build
M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build
M=$(PWD) clean
```

Una volta compilati i moduli, utilizzando il comando `insmod` possiamo caricare i tre moduli nel kernel così da avviare il task real-time.

9- Interfaccia utente

Mentre il task real-time verrà eseguito ciclicamente nello spazio kernel, l'utente può aver bisogno di ricevere informazioni sul sullo stato del controllore e sui dati rilevati dalla scheda, inoltre avrà bisogno di inviare comandi diretti al controllore per compiere le operazioni desiderate. Per risolvere questa problematica, abbiamo creato un'interfaccia utente grafica, la quale girerà nello spazio utente, utilizzando le librerie Qt e Qwt per C++.

9.1 Interfaccia grafica

Presentiamo di seguito l'interfaccia grafica utilizzata:

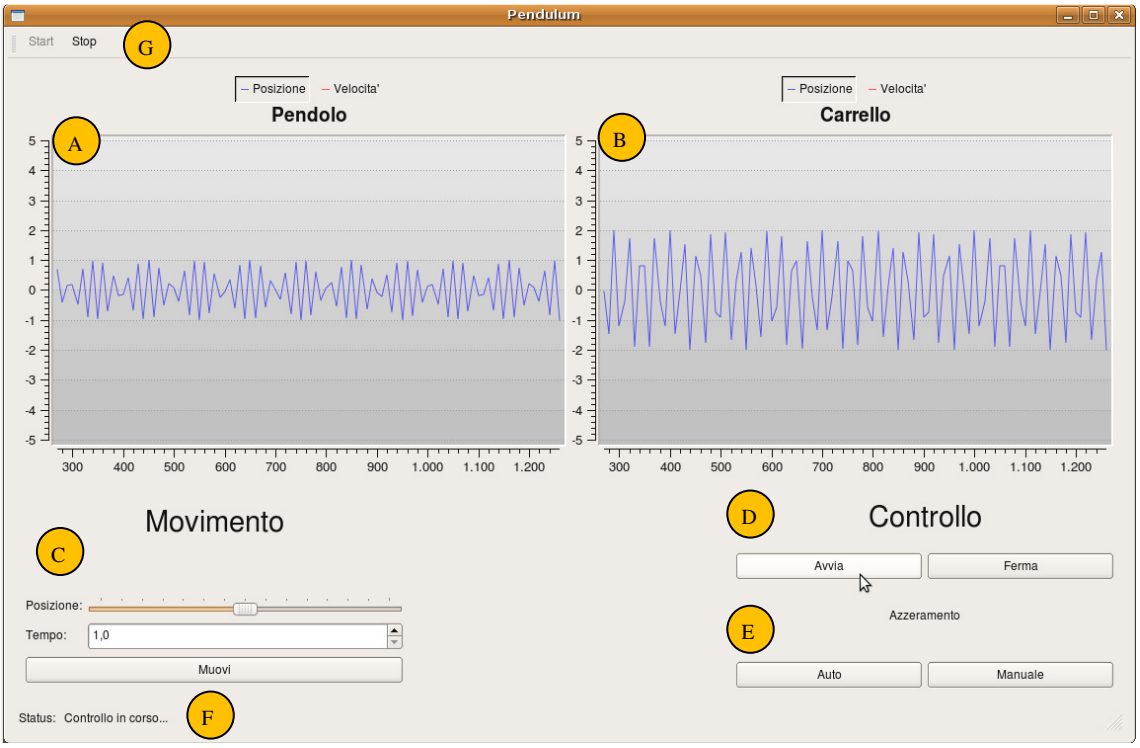


Figura 9.1 - interfaccia Pendulum

L'interfaccia presenta i seguenti componenti:

A. Grafico dell'angolo e della velocità angolare del pendolo, visualizzati in gradi e gradi al secondo.

B. Grafico della posizione e della velocità della slitta, visualizzati in millimetri e millimetri al secondo.
C. Pannello di azionamento del movimento.
D. Pannello di azionamento del controllo.
E. Pannello di azionamento dell'azzeramento.
F. Barra dello stato.
G. Barra dell'avvio.

Questi oggetti ci permetteranno di utilizzare le diverse funzioni messe a disposizione dal nostro controllore real-time.

9.1.1 Avvio

Per avviare l'applicazione, premere **Start** sulla barra dell'avvio, oppure digitare la combinazione CTRL+T. I grafici cominceranno così a mostrare i valori letti dagli encoder e la barra di stato visualizzerà il messaggio di pronto.

9.1.2 Azzeramento

Per avviare l'azzeramento automatico, utilizziamo il pulsante **Auto** sul pannello dell'azzeramento. La slitta comincerà a muoversi a sinistra e a destra per poi fermarsi nel punto centrale della corsa. Questa procedura è necessaria per poter utilizzare gli encoder incrementali.

Durante l'azzeramento, la barra di stato ci comunicherà che la funzione è in fase di esecuzione. Quando la barra di stato visualizzerà nuovamente che il sistema è in attesa, esso sarà pronto per essere controllato.

Utilizzando il pulsante **Manuale** possiamo determinare arbitrariamente il punto di azzeramento. La posizione corrente della slitta e l'angolo corrente del pendolo sommato a pigreco saranno presi come zero dei due encoder.

Bisogna inoltre assicurarsi di tenere il pendolo in posizione di riposo (fermo verso il basso) prima di avviare l'azzeramento manuale.

9.1.3 Controllo

Utilizziamo il pulsante **Controlla** nel pannello del controllo per avviare il controllore, il motore comincerà a fornire una coppia e la slitta si muoverà per mantenere l'equilibrio

del pendolo. Il controllo verrà attivato solo entro un certo angolo rispetto allo zero del pendolo, in questo modo si evita che il controllore immetta una coppia elevata per mantenere un equilibrio ormai pregiudicato, rischiando di compromettere la sicrezza delle persone e di surriscaldare il motore. Prima di avviare il controllo, è necessario eseguire l'azzeramento.

Premendo il tasto **Ferma**, il controllo verrà spento.

9.1.4 Movimento

Con il pannello del movimento possiamo spostare il punto di equilibrio della slitta in un qualsiasi punto della corsa (limitato però per evitare di spostare il punto di equilibrio troppo vicino ai fine corsa). Utilizzando lo slider di posizione, selezioniamo il punto di arrivo assoluto della slitta tra -350 e 350 millimetri (la lunghezza totale della corsa è 800), mentre utilizzando il box del tempo selezioniamo il tempo di percorrimento tra 1 e 10 secondi.

Premendo ora il pulsante **Muovi**, la slitta comincerà a spostarsi verso il punto indicato e il controllore rimarrà attivato durante tutta la durata del movimento.

Valori troppo bassi di tempo legati a valori elevati di spostamento possono portare all'instabilità del sistema.

9.1.5 Spegnimento

Il sistema può essere fermato utilizzando il pulsante **Stop** sulla barra dell'avvio. In questo modo, qualsiasi erogazione di coppia verrà fermata ed il sistema smetterà di leggere i dati degli encoder.

Alternativamente, il pulsante rosso di emergenza situato vicino al banco del pendolo ci permetterà di bloccare l'intero sistema nel caso di rischi alla sicurezza di persone e oggetti.

9.2 Codice

L'interfaccia è stata creata con codice nel linguaggio C++, utilizzando la suite di sviluppo QtCreator come editor e sfruttando le librerie grafiche Qt e Qwt. Di seguito analizziamo il codice generato.

9.2.1 Craezione interfaccia grafica

Il codice presenta una funzione principale `main`, il cui scopo è semplicemente quello di avviare l'applicazione grafica e mostrare la finestra principale.

```
#include <QtGui/QApplication>
#include "mainwindow.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

La classe `MainWindow` è contenuta nel file `mainwindow.cpp`. Il costruttore di `MainWindow` si occupa di creare le azioni e preparare l'interfaccia grafica (GUI).

```
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    createActions();
    setUpGui();
}
```

La funzione `createActions` inizializza le azioni che utilizzeremo, legandole a delle shortcut e assegnando ad ognuna uno slot. All'innesco (trigger) di un'azione, verrà mandato un segnale allo slot corrispondente, attivandolo.

```
/**
  * Inizializza le Azioni
  */
void MainWindow::createActions()
{
    // Start
    startAction = new QAction("S&tart", this);
    startAction->setShortcut(Qt::CTRL | Qt::Key_T);
    connect(startAction,     SIGNAL(triggered()),     this,
SLOT(start()));

    // Stop
    stopAction = new QAction("Sto&p", this);
    stopAction->setShortcut(Qt::CTRL | Qt::Key_P);
    stopAction->setEnabled(false);
    connect(stopAction,     SIGNAL(triggered()),     this,
SLOT(stop()));
}
```

La funzione `setUpGui` crea i pannelli dell'interfaccia e li dispone nella finestra principale. Oltre ad una barra degli strumenti superiore e ad una barra di stato, vengono creati i pannelli dei grafici, del controllo, del movimento e dell'azzeramento.

```
/**
  * Inizializza la MainWindow con tutti i vari componenti
  */
void MainWindow::setUpGui()
{
```

```
    // Creazione Pannelli
    createToolBar();
    createStatusBar();
    createPlotPanel();
    createControlPanel();
    createMovePanel();
    createZeroPanel();

    // Creazione Main Widget
    QWidget *mainWidget = new QWidget;
    this->setCentralWidget(mainWidget);

    // Disposizione Componenti
    QGridLayout *guiBox = new QGridLayout;
    QSpacerItem           *spacer           =           new
QSpacerItem(300,100,QSizePolicy::Minimum,QSizePolicy::Minim
um);
    guiBox->addWidget(plotPanel, 0, 0, 1, 3);
    guiBox->addWidget(movePanel, 1, 0, 2, 1);
    guiBox->addItem(spacer, 1, 1, 2, 1);
    guiBox->addWidget(controlPanel, 1, 2);
    guiBox->addWidget(zeroPanel, 2, 2);
    guiBox->addWidget(statusBar, 3, 0, 1, 3);

    guiBox->setSpacing(10);
    guiBox->columnStretch(2);

    mainWidget->setLayout(guiBox);
}
```

Le funzioni di creazione dei pannelli inserisce i pulsanti, le etichette e gli altri widget necessari all'interno dei pannelli designati. Inoltre, i segnali di attivazione dei pulsanti

vengono collegati agli slot corrispondenti e le azioni di start e stop vengono abilitate nella barra degli strumenti.

```
/**
  * Crea la barra degli strumenti
  */
void MainWindow::createToolBar()
{
    toolBar = addToolBar("Toolbar");
    toolBar->addAction(startAction);
    toolBar->addAction(stopAction);
}

/**
  * Crea la barra dello stato
  */
void MainWindow::createStatusBar()
{
   statusBar = new QStatusBar;

   QLabel *statusTitle = new QLabel("Status: ");

   statusBar->addWidget(statusTitle);

   statusLabel = new QLabel;
   statusLabel->setAlignment(Qt::AlignHCenter);
   statusLabel->setMinimumSize(statusLabel->sizeHint());
   statusLabel->setIndent(3);

   statusBar->addWidget(statusLabel);
}
```

```
/**
  * Crea il pannello con i grafici
  */
void MainWindow::createPlotPanel()
{
    plotCarrello = new EncoderPlot;
    plotCarrello->setTitle("Carrello");

    plotPendolo = new EncoderPlot;
    plotPendolo->setTitle("Pendolo");

    QHBoxLayout *plotBox = new QHBoxLayout;
    plotBox->addWidget(plotPendolo);
    plotBox->addWidget(plotCarrello);

    plotPanel = new QWidget;

    plotPanel->setLayout(plotBox);
}

/**
  * Crea il pannello con i comandi di controllo
  */
void MainWindow::createControlPanel()
{
    QGridLayout *layout = new QGridLayout;

    QLabel *controlLabel = new QLabel("<font size =
8>Controllo</font>");
    controlLabel->setAlignment(Qt::AlignHCenter);
    layout->addWidget(controlLabel, 0, 0, 1, 2);
```

```
    controlGo = new QPushButton("Avvia");
    connect(controlGo,       SIGNAL(clicked()),       this,
SLOT(control()));
    layout->addWidget(controlGo, 1, 0);

    controlStop = new QPushButton("Ferma");
    connect(controlStop,      SIGNAL(clicked()),      this,
SLOT(stopControl()));
    layout->addWidget(controlStop, 1, 1);

    controlPanel = new QWidget;
    controlPanel->setMaximumSize(400,100);
    controlPanel->setMinimumSize(400,100);
    controlPanel->setLayout(layout);
}

/**
  * Crea il pannello con i comandi di movimento
  */
void MainWindow::createMovePanel()
{
    QGridLayout *layout = new QGridLayout;

    QLabel   *moveLabel   =   new   QLabel("<font   size   =
8>Movimento</font>");
    moveLabel->setAlignment(Qt::AlignHCenter);
    layout->addWidget(moveLabel, 0, 0, 1, 2);

    QLabel *positionLabel = new QLabel("Posizione:");
    layout->addWidget(positionLabel, 1, 0);
    movePosition = new QSlider(Qt::Horizontal);
    movePosition->setRange(-350, 350);
```

```
    movePosition->setTickInterval(50);
    movePosition->setTickPosition(QSlider::TicksAbove);
    layout->addWidget(movePosition, 1, 1);

    QLabel *timeLabel = new QLabel("Tempo:");
    layout->addWidget(timeLabel, 2, 0);
    moveTime = new QDoubleSpinBox;
    moveTime->setRange(1.0, 10.0);
    moveTime->setDecimals(1);
    moveTime->setSingleStep(0.5);
    layout->addWidget(moveTime, 2, 1);

    moveDo = new QPushButton("Muovi");

    connect(movePosition, SIGNAL(valueChanged(int)), this,
SLOT(setMovePos(int)));
    connect(moveTime, SIGNAL(valueChanged(double)), this,
SLOT(setMoveTime(int)));
    connect(moveDo, SIGNAL(clicked()), this, SLOT(move()));
    layout->addWidget(moveDo, 3, 0, 1, 2);

    movePanel = new QWidget;
    movePanel->setMaximumSize(400,200);
    movePanel->setMinimumSize(400,200);
    movePanel->setLayout(layout);
}

/**
  * Crea il pannello con i comandi di azzeramento
  */
void MainWindow::createZeroPanel()
{
```

```
    QGridLayout *layout = new QGridLayout;

    QLabel *zeroLabel = new QLabel ("<font size =
8>Azzeramento</font>");
    zeroLabel->setAlignment(Qt::AlignHCenter);
    layout->addWidget(zeroLabel, 0, 0, 1, 2);

    autoZeroDo = new QPushButton("Auto");
    connect(autoZeroDo,      SIGNAL(clicked()),      this,
SLOT(autoZero()));
    layout->addWidget(autoZeroDo, 1, 0);

    manualZeroDo = new QPushButton("Manuale");
    layout->addWidget(manualZeroDo, 1, 1);
    connect(manualZeroDo,     SIGNAL(clicked()),     this,
SLOT(manualZero()));

    zeroPanel = new QWidget;
    zeroPanel->setMaximumSize(400,100);
    zeroPanel->setMinimumSize(400,100);
    zeroPanel->setLayout(layout);

}
```

9.2.2 Comunicazione con il task real-time

La comunicazione con il task real-time avviene attraverso delle FIFO condivise tra RTAI e Linux.

La funzione `openComm` si occupa di aprire i block device delle FIFO, fifoIn in lettura e fifoOut in scrittura. FifoIn è il canale di input dei valori inviati da RTAI verso Linux, mentre fifoOut è il canale di output dei comandi da Linux a RTAI.

La funzione `readValues` effettua una lettura del canale di input, ricevendo i dati della posizione dell'encoder del pendolo e del motore, dello stato di emergenza e dello stato del controllore.

```
/**
  * Legge i valori dalla FIFO
  */
bool    MainWindow::readValues(double    &pendolo,    double
&motore, unsigned int &emergenza)
{
    int ret;

    ret = read(fifoIn, &motore, sizeof(motore));
    ret = read(fifoIn, &pendolo, sizeof(pendolo));
    ret = read(fifoIn, &emergenza, sizeof(emergenza));
    ret = read(fifoIn, &stato, sizeof(stato));

    return ret != -1;
}

/**
  * Apre i canali di comunicazione
  */
bool MainWindow::openComm()
{
    fifoIn = open("/dev/rtf0", O_RDONLY);
    if (fifoIn < 0)
    {
        return false;
    }

    fifoOut = open("/dev/rtf1", O_WRONLY);
    if (fifoOut < 0)
```

```
    {
        return false;
    }

    return true;
}
```

9.2.3 Handler del timer

Gli eventi temporizzati vengono gestiti da un handler, il quale viene richiamato ogniqualvolta il timer segna il passaggio ad un nuovo periodo. La funzione `timerEvent` si occupa di leggere i dati dalla fifo di input, aggiornare i grafici, eseguire il controllo dell'emergenza e aggiornare la barra di stato.

```
void MainWindow::timerEvent(QTimerEvent *)
{
    double pendolo, motore;
    unsigned int emergenza;
    static int cont = 0;

    // - lettura dei dati in input
    if (readValues(pendolo, motore, emergenza))
    {

        if (cont == 20)
        {
            // - aggiornare i plot e lo stato
            plotPendolo->addValue(pendolo);
            plotCarrello->addValue(motore);
            cont = 0;
        }
        cont++;
```

```
    }

    // - controllo dell'emergenza
    if (emergenza == 1)
    {
       stop();
    }

    // - aggiornamento dello stato
    updateStatus();
}
```

9.2.4 Slot

Le librerie Qt inroducono la funzionalità di comunicazione denominata “Slot and Signals”. Le funzioni, le azioni e i pulsanti dell’interfaccia grafica possono inviare dei segnali. Altre funzioni chiamate slot restano in ascolto, in attesa di segnali. Quando un segnale viene inviato, lo slot corrispondente verrà attivato e il codice contenuto sarà eseguito. E’ possibile collegare un segnale a più slot e più segnali allo stesso slot.

Lo slot `start` abiliterà il controllore, inviando il comando di abilitazione ed attivando il timer per l’acquisizione.

```
/**
  * Slot: fa partire il sistema
  */
void MainWindow::start()
{
    if ( ! openComm())
    {
        // Errore
        return;
```

```
    }

    // comando enable
    Comando comando = ABILITA;
    write(fifoOut, &comando, sizeof(comando));

    timerTicks = 0;
    timerId = startTimer(TIMER_INTERVAL);

    startAction->setEnabled(false);
    stopAction->setEnabled(true);
}
```

Lo slot `stop` disabiliterà il controllore, inviando il comando di diasbilitazione e fermando il timer dell'acquisizione.

```
/**
  * Slot: ferma il sistema
  */
void MainWindow::stop()
{
    killTimer(timerId);
    closeComm();

    // comando disable
    Comando comando = DISABILITA;
    write(fifoOut, &comando, sizeof(comando));


    startAction->setEnabled(true);
    stopAction->setEnabled(false);
```

```
}
```

Lo slot `control` avvia la funzione di controllo, inviando il comando CONTROLLA al task. Lo slot `stopControl` ferma il controllo.

```
/**
  * Slot: avvia il controllo
  */
void MainWindow::control()
{
    Comando comando = CONTROLLA;
    write(fifoOut, &comando, sizeof(comando));
}

/**
  * Slot: ferma il controllo
  */
void MainWindow::stopControl()
{
    Comando comando = FERMA;
    write(fifoOut, &comando, sizeof(comando));
}
```

Lo slot `move` avvia il movimento inviando il comando MUOVI ed i parametri di posizione e tempo. I parametri vengono aggiornati attraverso gli slot `setMovePos` controllato da uno slider e `setMoveTime` controllato da un box di input.

```
/**
  * Slot: avvia il movimento
  */
```

```
void MainWindow::move()
{
    Comando comando = MUOVI;

    write(fifoOut, &comando, sizeof(comando));
    write(fifoOut, &comando, sizeof(comando));
    write(fifoOut, &comando, sizeof(comando));
}

/**
  * Slot: settala posizione finale del movimento
  */
void MainWindow::setMovePos(int pos)
{
    newMovePos = pos;
}

/**
  * Slot: setta il tempo di percorrimento del movimento
  */
void MainWindow::setMoveTime(double time)
{
    newMoveTime = time;
}
```

Lo slot `autoZero` avvia l'azzeramento automatico, inviando il comando AZZERA. Lo slot `manualZero` avvia l'azzeramento manuale, inviando il comando AZZERA_MANUALE.

```
/**
  * Slot: avvia l'azzeramento automatico
```

```
  */
void MainWindow::autoZero()
{
    Comando comando = AZZERA;
    write(fifoOut, &comando, sizeof(comando));
}

/**
  * Slot: avvia l'azzeramento manuale
  */
void MainWindow::manualZero()
{
    Comando comando = AZZERA_MANUALE;
    write(fifoOut, &comando, sizeof(comando));
}
```

9.2.5 Aggiornamento dello stato

La funzione `updateStatus` visualizza lo stato del sistema sotto forma di un messaggio nella barra dello stato.

```
/**
  * Aggiorna la barra dello stato
  */

void MainWindow::updateStatus()
{
    static int statoCont = 0;
    switch (stato)
    {
        case FERMO:
            statusLabel->setText("Sistema in attesa.");
            break;
```

```
        case AZZERAMENTO:
            if     (statoCont     ==     0)     statusLabel-
>setText("Azzeramento in corso.");
            else   if   (statoCont   ==   10)   statusLabel-
>setText("Azzeramento in corso..");
            else   if   (statoCont   ==   20)   statusLabel-
>setText("Azzeramento in corso...");
            statoCont ++;
            if (statoCont == 30) statoCont = 0;
            break;

        case CONTROLLO:
            if     (statoCont     ==     0)     statusLabel-
>setText("Controllo in corso.");
            else   if   (statoCont   ==   10)   statusLabel-
>setText("Controllo in corso..");
            else   if   (statoCont   ==   20)   statusLabel-
>setText("Controllo in corso...");
            statoCont ++;
            if (statoCont == 30) statoCont = 0;
            break;

        case MOVIMENTO:
            if     (statoCont     ==     0)     statusLabel-
>setText("Movimento in corso.");
            else   if   (statoCont   ==   10)   statusLabel-
>setText("Movimento in corso..");
            else   if   (statoCont   ==   20)   statusLabel-
>setText("Movimento in corso...");
            statoCont ++;
            if (statoCont == 30) statoCont = 0;
```

```
            break;

    }
}
```

9.2.6 Grafici dell'encoder

I file `encoder.cpp` e `encoder.h` contengono il codice di `EncoderPlot`, classe usata per creare i grafici della posizione e della velocità degli encoder. Vengono create due istanze della classe, una per l'encoder del pendolo e una per l'encoder del motore.

Il costruttore di `EncoderPlot` inizializza i valori di posizione, velocità e tempo dei grafici e richiama le funzioni di creazione del grafico e delle curve.

```
EncoderPlot::EncoderPlot()
{
    // Init Global Variable
    timerInterval = 10;
    dataCount = 0;

    setUpPlot();
    setUpCurves();

    // Time
    for ( int i = 0; i < MaxDataSize; i++ )
    {
        time[i] = i * timerInterval;
    }
}
```

La funzione `setUpPlot` crea e configura l'area del grafico, la funzione `setUpCurves` crea le curve di posizione e velocità.

```
/**
  * Configura il plot
  */
void EncoderPlot::setUpPlot()
{
    // Size
    setMinimumSize(400, 400);

    // Dummy Title
    setTitle("Title");

    // Legend
    plotLegend = new QwtLegend;
    insertLegend(plotLegend, QwtPlot::TopLegend);

    // Background
    Background *bg = new Background();
    bg->attach(this);

    // Axis
    //enableAxis(QwtPlot::yRight);
    setAxisScale(QwtPlot::xBottom, 0, MaxDataSize, 1000);
    setAxisScale(QwtPlot::yLeft, -5, +5, 1);

    // Grid
    plotGrid = new QwtPlotGrid();
    plotGrid->setMajPen(QPen(Qt::gray, 0, Qt::DotLine));
    plotGrid->enableX(false);
    plotGrid->attach(this);
}

/**
```

```
  * Configura le curve
  */
void EncoderPlot::setUpCurves()
{
    // Position Curve
    posCurve = new EncoderCurve("Posizione");
    posCurve->setColor(Qt::blue);
    posCurve->attach(this);

    // Speed Curve
    speCurve = new EncoderCurve("Velocita'");
    speCurve->setColor(Qt::red);
    speCurve->setZ(posCurve->z() - 1);
    speCurve->attach(this);
}
```

La funzione `addValue` trasla i valori nel tempo ed aggiunge il nuovo valore alla curve, inserito come parametro. Infine, la funzione aggiorna l'asse dei tempi e ridisegna il grafico.

```
void EncoderPlot::addValue(double value)
{
    if (dataCount == MaxDataSize)
    {
        // Shift Valori
        for (int i = 0; i < MaxDataSize; i++)
        {
            position[i] = position[i + 1];
            speed[i] = speed[i + 1];
            time[i] += timerInterval;
        }
```

```
    }

    if (dataCount < MaxDataSize)
    {
        dataCount++;
    }

    // Nuovi Valori
    position[dataCount - 1] = value;
    double prevPos = 0.0;
    if (dataCount > 1)
    {
        prevPos = position[dataCount - 2];
    }
    speed[dataCount - 1] = 1000.0 * (value - prevPos) /
timerInterval;

    setAxisScale(QwtPlot::xBottom,                time[0],
time[MaxDataSize-1], 1000);

    // Plotta
    posCurve->setRawData(time, position, dataCount);
    speCurve->setRawData(time, speed, dataCount);
    replot();
}
```

La funzione `setTimerInterval` è utilizzata per inizializzare l'intervallo temporale di aggiornamento del grafico.

```
/**
  * Setta l'intervallo temporale
```

```
  */
void EncoderPlot::setTimerInterval(int timerInterval)
{
    this->timerInterval = timerInterval;
}
```

Le classi `Background` ed `EncoderCurve` sono delle classi di utilità per lo stile grafico dello sfondo e delle curve.

```
#include "encoderplot.h"

class Background: public QwtPlotItem
{
public:
    Background()
    {
        setZ(0.0);
    }

    virtual int rtti() const
    {
        return QwtPlotItem::Rtti_PlotUserItem;
    }

    virtual void draw(QPainter *painter,
        const QwtScaleMap &, const QwtScaleMap &yMap,
        const QRect &rect) const
    {
        QColor c(Qt::white);
        QRect r = rect;
```

```
        for ( int i = 10; i > -10; i -= 1 )
        {
            r.setBottom(yMap.transform(i - 1));
            r.setTop(yMap.transform(i));
            painter->fillRect(r, c);

            c = c.dark(102);
        }
    }
};

class EncoderCurve: public QwtPlotCurve
{
public:
    EncoderCurve(const QString &title):
        QwtPlotCurve(title)
    {
        setRenderHint(QwtPlotItem::RenderAntialiased);
    }

    void setColor(const QColor &color)
    {
        QColor c = color;
        c.setAlpha(150);

        setPen(c);
    }
};
```

Appendici

A – RTAI-Lab

Il processo per la creazione di programmi real-time utilizzando Linux-RTAI è difficile e tedioso, in quanto il programmatore è costretto a creare dei moduli kernel che prevedono l'utilizzo di librerie e funzioni particolari e comunque diverse da quelle user-space. Inoltre, la creazione dell'interfaccia grafica è un'altra attività dispendiosa in quanto a tempo.

Per ovviare a queste problematiche, è stato creato RTAI-Lab, una catena di strumenti open-source per aiutare il programmatore nella creazione e gestione di software real-time. L'utilizzo di RTAI-Lab ci permetterà di creare facilmente applicazioni real-time e verificarne lo stato attraverso un'interfaccia grafica pre-impostata.

RTAI-lab si compone di:

- Linux-RTAI, il sistema operativo real-time per eseguire i task creati, così come fatto da noi durante la durata del progetto.
- Comedi, il progetto per l'utilizzo di driver di acquisizione dati per potersi interfacciare con il sistema fisico.
- Scilab, un pacchetto open-source di software scientifico per la computazione numerica, in diretta concorrenza con Matlab.
- Scicos, un programma grafico open-source di modellazione di sistemi e simulazione, molto simile a Simulink.
- Xrtailab, un oscilloscopio interattivo creato con le librerie Mesa ed Efltk.

La procedura per la creazione di software real-time prevede dei passi molto più semplici rispetto alla programmazione diretta in C di moduli kernel come fatto da noi.

Per creare un programma, è solamente necessario avviare Scilab e Scicos con la patch di RTAI (fornita con la distribuzione di RTAI-lab), creare graficamente il programma desiderato utilizzando i blocchi di Scicos ed utilizzare il comando "generate real-time code" per ottenere il codice già pronto. Infine, avviando il programma successivamente

compilato, possiamo connetterlo ad Xrtailab per visualizzare i dati ricevuti in modo grafico.

Sfortunatamente, per una questione di sviluppi differenti dei progetti Scilab, Scicos ed RTAI, il supporto ad RTAI-lab è venuto a mancare per le ultime versioni di tali programmi che sono ora incompatibili tra di loro. Da notare è la creazione di Scicoslab, un programma dei creatori di Scicos che tenta di mantanere una versione stabile del binomio Scilab-Scicos, ma che non è supportato da RTAI fino alla versione 3.7.

Per questo motivo, abbiamo invece utilizzato codice creato da noi sia per quanto riguarda la creazione dei moduli kernel che l'interfaccia utente grafica.

Ultimamente è nato anche un altro progetto, denominato Hart, che cerca di coniugare il sistema RTAI con Scicoslab e Comedi, integrando il tutto. Questo progetto è però ancora in fase di partenza.

B - Filtri

Con i primi tentativi, abbiamo visto che il sistema risponde in modo poco fluido, a causa delle vibrazioni dell'asta ed a problemi di derivazione numerica, in quanto la lettura dei dati dell'encoder fornisce un valore digitale e perciò discontinuo.

La funzione di un filtro è quella di attenuare alcune frequenze e lasciarne passare altre. In particolare, i filtri passa-basso lasciano passare frequenze minori di una data frequenza detta "frequenza di taglio" ω_c, mentre i filtri passa-alto lasciano passare frequenze maggiori di ω_c.

Nel nostro caso utilizzeremo un filtro passa-basso. Il nostro filtro sarà passivo, in quanto non introdurrà energia nel sistema. Inoltre, utilizzeremo un filtro di secondo grado così da avere un'attenuazione maggiore sopra la frequenza di taglio.

Applicheremo i nostri filtri agli ingressi ed alle uscite analogiche, vale a dire alla posizione dei due encoder e alla coppia in uscita.

I filtri sono solitamente applicati attraverso dei circuiti elettronici, ma il loro comportamento può essere anche simulato ad un programma o un circuito, costruendo così un "filtro digitale". Noi seguiremo questa strada.

Un filtro passa-basso di secondo grado avrà una funzione di trasferimento con 2 poli reali non distinti, la costante di tempo τ corrisponde alla frequenza di taglio $\omega_c = \frac{1}{2\pi\tau}$.

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

Il grafico di bode che otterremo sarà simile a quello mostrato in figura e figura.

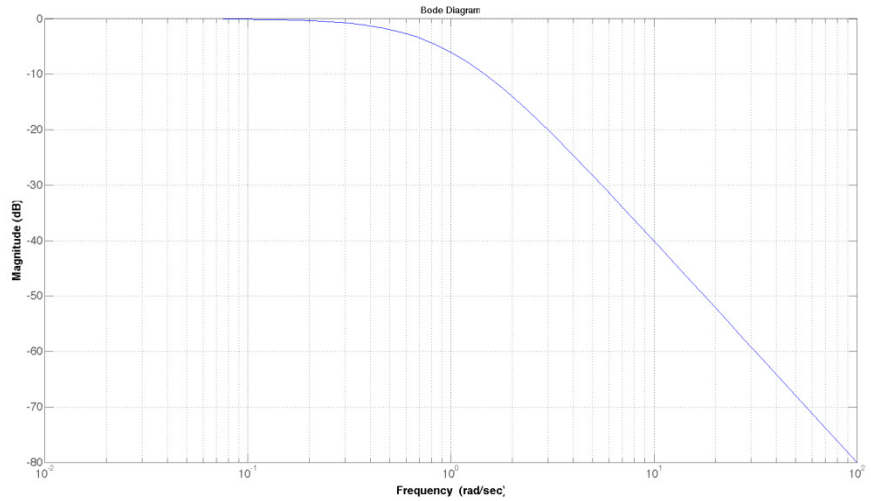


Figura B.1 – Diagramma del modulo

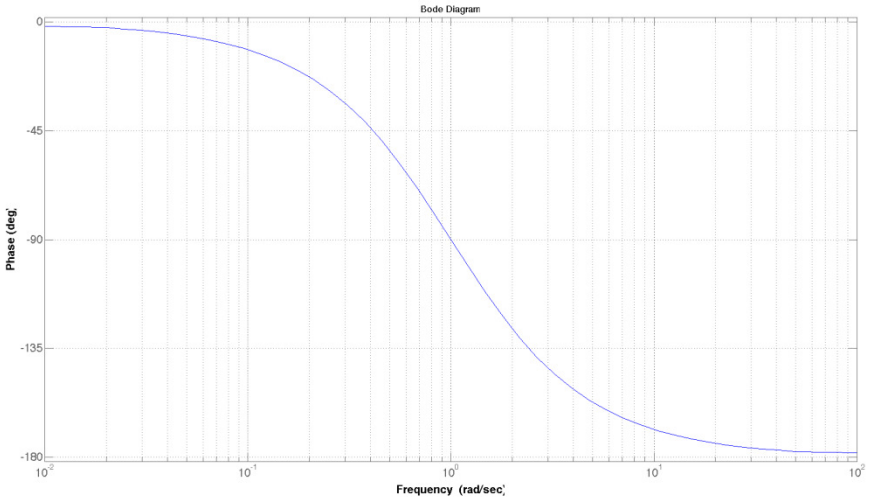


Figura B.2 – Diagramma della fase

Notiamo che superata frequenza di taglio il filtro comincia ad attenuare il modulo delle frequenze. Scelta la frequenza di taglio ω_c calcoliamo la costante di tempo τ come $\frac{1}{2\pi\omega_c}$

I segnali fisici, come il voltaggio o la corrente, variano con continuità nel tempo, assumono perciò un valore per ogni istante di tempo reale t, questi segnali sono chiamati analogici. I controllori elettronici, invece, lavorano su segnali discretizzati, per

cui il segnale assume un valore solo in certi istanti di tempo, separati da un periodo T. Questi segnali sono chiamati digitali.

Per poter filtrare i segnali di posizione letti dall'encoder o la coppia data al sistema con l'ausilio del motore, i quali vengono letti o scritti una volta per periodo (nel nostro caso ogni 10 millisecondi), avremo bisogno di trasformare i nostri filtri analogici in filtri digitali attraverso una conversione. Questo procedimento viene chiamato campionamento, mentre l'inverso è chiamato tenuta.

Trasformiamo perciò la nostra funzione di trasferimento continua nella variabile s in una funzione di trasferimento discreta nella variabile z. Per fare ciò, abbiamo diversi metodi lineari di primo ordine: Eulero in avanti, Eulero all'indietro e metodo di Tustin.

Utilizzeremo il metodo di Tustin, il quale permette una migliore approssimazione del segnale alla funzione continua. Effettuiamo perciò la conversione dalla funzione continua alla funzione discreta che utilizzeremo (T_c tempo di campionamento):

$$s \to \frac{2}{T_c}\frac{z-1}{z+1}$$

$$\frac{1}{\left(1+\tau\frac{2}{T_c}\frac{z-1}{z+1}\right)^2}$$

Usiamo MATLab per effettuare il calcolo velocemente

```
function F=discrete_lowpass_2(cutF, Tc)
% Ritorna un filtro passa basso di secondo ordine
%
% Dati in input:
% cutF = frequenza di taglio [Hz]
% Tc = tempo di campionamento [s]
```

```
tau = 1 / (2 * pi * cutF);

s = tf('s');
sys = 1 / ((1 + tau * s) ^ 2);
F=c2d(sys, Tc, 'Tustin');
```

Otteniamo una funzione di trasferimento discretizzata nella forma seguente:

$$\frac{Y(z)}{U(z)} = \frac{\alpha_1 s + \alpha_0}{\beta_2 s^2 + \beta_1 s + \beta_0}$$

Trasformiamo questa funzione nell'equazione ricorsiva e calcoliamo l'antitrasformata di Laplace che utilizzeremo nelle librerie per applicare il filtro:

$$Y(z)\left(\beta_2 s^2 + \beta_1 s + \beta_0\right) = U(z)\left(\alpha_1 s + \alpha_0\right)$$
$$\beta_2 y(k+2) + \beta_1 y(k+1) + \beta_0 y(k) = \alpha_1 u(k+1) + \alpha_0 u(k)$$

$$y(k) = \frac{\alpha_1}{\beta_2} u(k-1) + \frac{\alpha_0}{\beta_2} u(k-2) - \frac{\beta_1}{\beta_2} y(k-1) - \frac{\beta_0}{\beta_2} y(k-2)$$

Conclusioni

Svolgendo questo progetto abbiamo acquisito la conoscenza di diversi ambienti di sviluppo, affrontato problemi pratici riguardanti la meccanica e l'elettronica ed affrontato i problemi legati all'automatica.

Nel **capitolo primo** abbiamo perciò analizzato i diversi componenti, acquisendo una conoscenza più approfondita dell'elettronica utilizzata e dei motivi che ci hanno spinto ad usarla.

Nel **capitolo secondo** abbiamo affrontato la dinamica del sistema, calcolandone le equazioni dello stato e gli equilibri, verificando infine attraverso simulazioni il comportamento del nostro pendolo inverso.

Il **capitolo terzo** ci ha permesso di affrontare le problematiche legate all'automazione, studiando la stabilità dell'equilibrio, selezionando un controllore adeguato e creandolo, per poi simularne il comportamento.

Nel **capitolo quarto** abbiamo affrontato la tematica del movimento, analizzando matematicamente le equazioni del moto e simulando la funzione finale.

Con il **capitolo quinto** abbiamo presentato il progetto RTAI, studiandone la struttura e le funzionalità ed applicandole al nostro sistema.

L'utilizzo di Comedi nel **capitolo sesto** ci ha permesso di affrontare il problema dell'acquisizione dei dati con un software dedicato, sfruttandone la compatibilità con RTAI.

Nel **capitolo settimo** abbiamo presentato i passaggi effettuati per la creazione del sistema, affrontando le problematiche risultanti dall'utilizzo del sistema operativo Linux e sfruttando le sue potenzialità.

Con il **capitolo ottavo** abbiamo implementato il controllore con dei moduli kernel che gestiscono il task real-time utilizzato, sfruttando la conoscenza del linguaggio C.

Infine, nel **capitolo nono** abbiamo creato un'interfaccia utente per comunicare con il controllore real-time, sfruttando le librerie grafiche Qt e Qwt così da generare un'interfaccia grafica intuitiva.

Concludendo, abbiamo visto che utilizzando un sistema Linux real-time con RTAI possiamo creare programmi hard real-time efficienti con un costo molto basso rispetto ad una soluzione con software proprietario (come LabView della National Instruments), inoltre abbiamo un controllo più approfondito e possiamo modificare praticamente ogni aspetto del progetto. Un altro vantaggio dato da questo approccio è la possibilità di utilizzare un solo computer come host (contenente i programmi utente) e target (contenente i task real-time).

Il problema sta nella difficoltà della programmazione, nella necessità di conoscere molto bene il sistema utilizzato e nella più ardua manutenibilità del codice. Queste problematiche possono però essere superate con un'adeguata preparazione del programmatore.

Bibliografia

[1] P. Bolzern, *Fondamenti di controlli automatici,* Mc Graw-Hill, 2004

[2] Manuale di Istruzioni s626

[3] J. Blanchette, *C++ GUI programming with Qt 4,* 1st Edition

[4] Linux Kernel archive *www.linux.org*

[5] Sito ufficiale del progetto RTAI *www.rtai.org*

[6] Tutorial RTAILab *www.rtai.org/RTAILAB/RTAI-Lab-tutorial.pdf*

[7] Linux Kernel archive *www.linux.org*

[8] Loadable Module Kernel HOWTO *tldp.org/HOWTO/module-HOWTO/*

[9] Sito ufficiale progetto Comedi *www.linux.org*

[10] Sito ufficiale Scilab *www.scilab.org*

[11] Sito ufficiale Scicos *www.scicos.org*

[12] Sito ufficiale Scicoslab *www.scicoslab.org*

[13] Sito ufficiale Hart toolbox *hart.sourceforge.net*

[14] Sito ufficiale librerie Qt *www.qtsoftware.com*

[15] Sito ufficiale librerie Qwt *qwt.sourceforge.net*