Università di Pisa

Corso di Laurea magistrale
in Ingegneria Robotica e dell'Automazione

Simulazione e Controllo del Pendolo Inverso di Furuta

Autori:
Francesco Petracci
Simone Silenzi

Matricola e-mail:
491648, petracci.francesco@gmail.com
460123, s.silenzi1@gmail.com

1 Introduzione

Questo progetto consiste nella simulazione dell'interazione real-time tra il pendolo inverso di Furuta, una scheda STM32F4 Discovery e un'interfaccia utente su pc.

L'applicazione finale è stata sviluppata in C sfruttando principalmente le librerie Allegro 4.4 e pthread. Alcune funzioni sono state generate in modo automatico sfruttando l'Embedded Coder di Matlab–Simulink, programma attraverso il quale è stato sviluppato e convalidato il modello fisico del pendolo e il relativo controllo.

Questo ha avuto come obbiettivo principale il poter portare e mantenere il pendolo in posizione verticale consentendo anche delle variazioni di orientazione. Un ulteriore sfida è stata rendere robusto il controllore sviluppato a vari disturbi quali rumori di misura, ritardi e piccole forze esterne.

2 Modello

3 Controllo

4 Interfaccia

L'interfaccia è divisa in due zone principali: quella di comunicazione con l'utente e le tre viste del pendolo.

Per quanto riguarda la zona di comunicazione questa è popolata dalle variabili di interesse e i modi con i quali modificarle. Si è quindi riportato a schermo:

- il riferimento su α
- il vettore di stato $[\alpha\ \theta\ volt]^T$
- i parametri di controllo
- disturbo, rumore e ritardo
- polo della funzione di trasferimento che genera il riferimento su α
- periodo del task di controllo
- per ciascun task il numero di deadline miss, il tempo di esecuzione in µs e il worst-case execution time in µs
- varie istruzioni su come resettare e come uscire dal programma

Per quanto riguarda le viste, ne sono riportate tre. La scelta di rappresentare il pendolo attraverso tre viste è funzionale a valutare l'efficacia dell'azione di controllo sui singoli angoli. Tramite la vista dall'alto viene messa in evidenza la posizione corrente del primo link, in verde, e la posizione desiderata, in grigio. Lo stesso viene fatto per il secondo link, in rosso, per mezzo della vista laterale.

Attraverso la terza vista, ovvero quella assonometrica, si rappresenta la struttura complessiva del pendolo che rende più intuitiva la comprensione dello stato attuale del

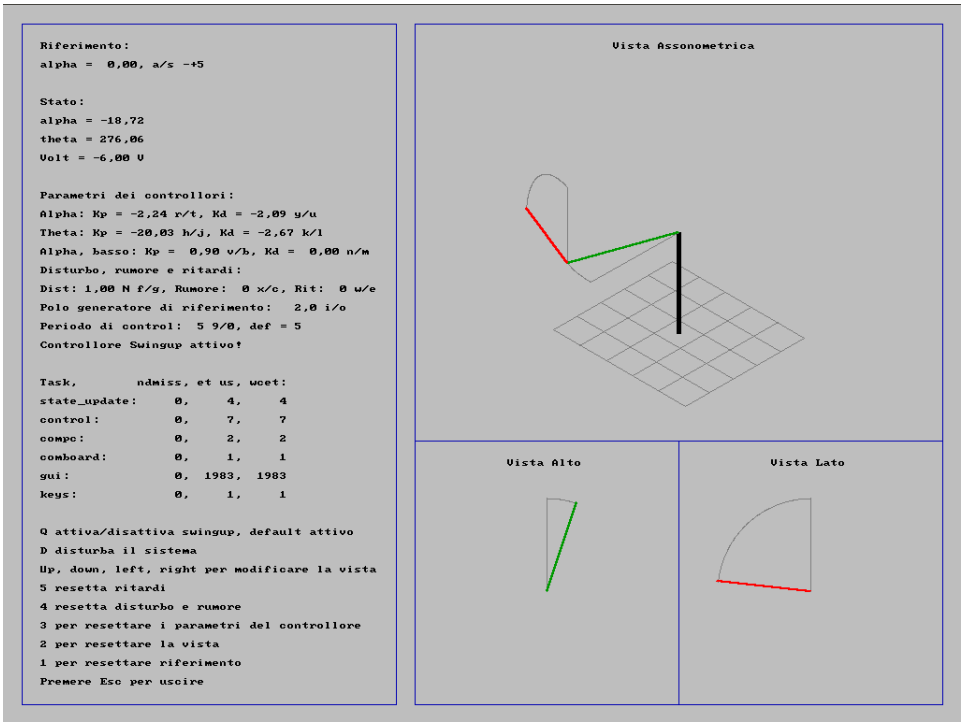

Figura 1: Interfaccia durante l'esecuzione

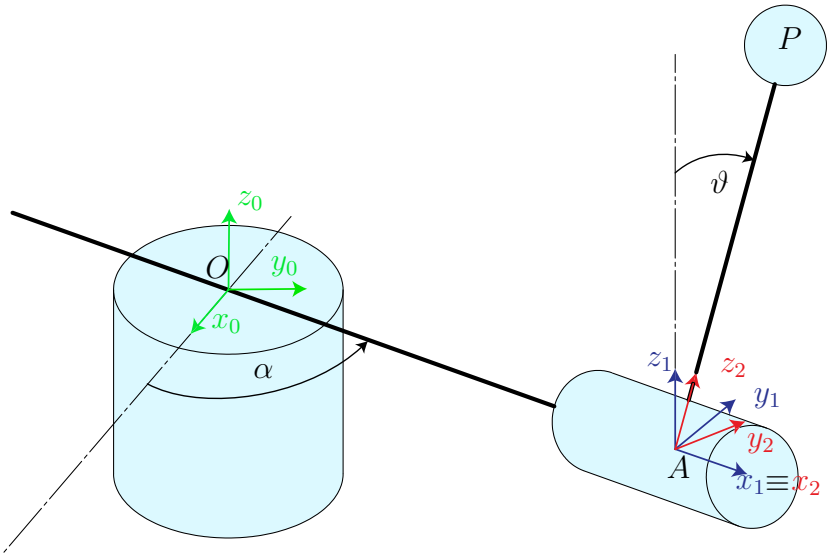

Figura 2: Schema del Pendolo

sistema. Al fine di facilitare l'osservazione del pendolo durante il suo moto, è stato deciso di rendere tale vista interattiva, consentendo all'utente di spostare il punto di vista rispetto al quale osserva il sistema.

Facendo riferimento alla figura 2, i punti di interesse per disegnare il pendolo sono O, A e P, rispettivamente i punti intorno ai quali ruotano il primo link, il secondo link e l'estremità di quest'ultimo. I sistemi di riferimento intermedi, utilizzati per la parametrizzazione del problema, sono riportati in figura 2. Per agevolare lo sviluppo delle relazioni che legano un qualsiasi punto dello spazio fisico ad un punto nel piano immagine dell'interfaccia è risultato utile strutturare il problema in coordinate omogenee in quanto permettono di rappresentare le rototraslazioni come trasformazioni lineari. Queste vengono rappresentate da un'unica matrice ${}^{i}T_{ij}$ composta da $R_i(\varphi)$, generica rotazione intorno all'asse i di un angolo φ, e da ${}^{i}d_{ij}$ vettore congiungente le origini dei sistemi di riferimento i e j scritto in coordinate i:

$$ {}^{i}T_{ij}(\varphi) = \left[\begin{array}{ccc|c} & R_z(\alpha) & & {}^{i}d_{ij} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{1} $$

Per il generico vettore posizione scritto in coordinate omogenee vale la seguente relazione di cambio sistema di riferimento: ${}^{i}p = T_{ij}\ {}^{j}p$. Nel caso in esame sono state utilizzate le seguenti matrici:

$$ {}^{0}T_{01}(\alpha) = \left[\begin{array}{ccc|c} & R_z(\alpha) & & \begin{matrix} l_1 \\ 0 \\ 0 \end{matrix} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \quad {}^{1}T_{12}(\vartheta) = \left[\begin{array}{ccc|c} & R_x(-\vartheta) & & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{2} $$

Con queste sono state quindi ricavate le coordinate in sistema di riferimento 0 dei punti di interesse:

$$ \begin{aligned} {}^{0}\mathrm{OA} &= \begin{bmatrix} l_1 \cos\alpha \\ l_1 \sin\alpha \\ 0 \end{bmatrix} \\ {}^{0}\mathrm{OP} &= \begin{bmatrix} l_1 \cos\alpha - l_2 \sin\alpha \sin\vartheta \\ l_1 \sin\alpha + l_2 \cos\alpha \sin\vartheta \\ l_2 \cos\vartheta \end{bmatrix} \end{aligned} \tag{3} $$

Per poter rappresentare su schermo il pendolo è necessario infine proiettare le coordinate spaziali dei punti ottenute in uno spazio bidimensionale. A tale scopo viene utilizzata la seguente relazione, in cui si indicano con s gli assi assonometrici, con ρ l'angolo di longitudine e con λ quello di latitudine:

$$ R_{s0} = R_z(\rho) R_y(-\lambda) \begin{bmatrix} 0 & 0 & -1 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \tag{4} $$

In questa è stato incluso il cambio orientazione necessario per passare al sistema di coordinate di Allegro. Il generico punto ${}^{0}[x\ y\ z]^t$ risulta avere componenti:

$$ x_s = x \sin\rho + y \cos\rho, \quad y_s = x \cos\rho \sin\lambda + y \sin\lambda \sin\rho - z \cos\lambda \tag{5} $$

	gui	*keys*	*compc*	*comboard*	*control*	*state_update*
`state_pc`	r		w			
`ref_pc`	r	w	r			
`par_control_pc`	r	w	r			
`view`	r	w				
`state_buffer`			r	w		
`ref_buffer`			w	r		
`par_control_buffer`			w	r		
`state_board`				r	r/w	r/w
`ref_board`				w	r	
`par_control_board`				w	r	
`dn`		w			r/w	r
`par_dn`	r	w			r	

Tabella 1: Risorse condivise

5 Overview del codice

Il software è stato sviluppato in C usando principalmente Allegro 4.4, pthread e l'Embedded Coder di Simulink–Matlab. La cartella principale ha 3 principali sotto–cartelle: *Matlab* contiene i file sviluppati per elaborare il modello fisico e i disegni dell'interfaccia, *include* gli header del codice utilizzati e infine *src* i file sorgenti. Una cartella aggiuntiva, *build*, viene creata al momento della compilazione e contiene i file object e il file binario.

Una parte dei file sorgenti e dei file header sono stati generati in modo automatico tramite l'utilizzo appunto dell'Embedded Coder. Questi sono stati contrassegnati ad inizio file da un'intestazione e sono stati inoltre descritti brevemente nella prima parte del file `main.c`.

5.1 Data structures

Le strutture dati condivise sono tutte protette da mutex, e sono necessarie per lo scambio di informazioni tra i task. Sono inoltre state separate tra variabili lato pc, buffer e lato scheda. È possibile vedere il loro utilizzo da parte dei task nella tabella 1 in cui si è specificato se un task legge tale struttura con *r* read, o se la modifica con *w* write.

Gli elementi dentro la struttura `state_pc` sono:

- `alpha`, l'angolo in gradi del primo link
- `theta`, l'angolo in gradi del secondo link
- `voltage`, il voltaggio corrente dell'alimentazione del motore

Gli elementi dentro la struttura `state_board` sono:

- `CNT_alpha`, la lettura dell'encoder riferito all'angolo del primo link
- `CNT_theta`, la lettura dell'encoder riferito all'angolo del secondo link
- `CCR`, Capture and Compare Register utilizzato per settare il voltaggio del motore

Nelle structure `ref` ci sono:

- `alpha`, riferimento per l'angolo del primo link
- `theta`, riferimento per l'angolo del secondo link
- `swingup`, specifica in che configurazione vogliamo il pendolo

Le strutture `par_control` contengono i vari parametri del controllore, abbiamo quindi:

- `up_kp_alpha`, costante proporzionale del controllore *Up* su α
- `up_kp_theta`, costante proporzionale del controllore *Up* su ϑ
- `up_kd_alpha`, costante derivativa del controllore *Up* su α
- `up_kd_theta`, costante derivativa del controllore *Up* su ϑ
- `down_kp_alpha`, costante proporzionale del controllore *Down* su α
- `down_kp_alpha`, costante derivativa del controllore *Down* su α
- `dpole_ref`, polo per la generazione di riferimento
- `ref_gen_num`, vettore contenente i coefficienti del numeratore della funzione di trasferimento del generatore di riferimento
- `ref_gen_den`, vettore contenente i coefficienti del denominatore della funzione di trasferimento del generatore di riferimento

Nella struttura `view` si ha:

- `lon`, angolo per la vista che descrive la longitudine, chiamato ρ nella sezione 4
- `lon`, angolo per la vista che descrive la latitudine, chiamato λ nella sezione 4

Per quanto riguarda i disturbi e i rumori abbiamo due strutture. La prima `dn` contiene:

- `kick`, segnala l'attivazione del disturbo
- `dist`, forza applicata all'estremità del pendolo
- `noise`, vettore di due termini contenente l'errore di lettura sui due angoli
- `delay`, numero di campioni di ritardo nel controllo

La seconda `par_dn` invece contiene:

- `dist_amp`, ampiezza della forza applicata all'estremità del pendolo
- `noise_amp`, ampiezza massima del rumore espressa in numero di passi dell'encoder

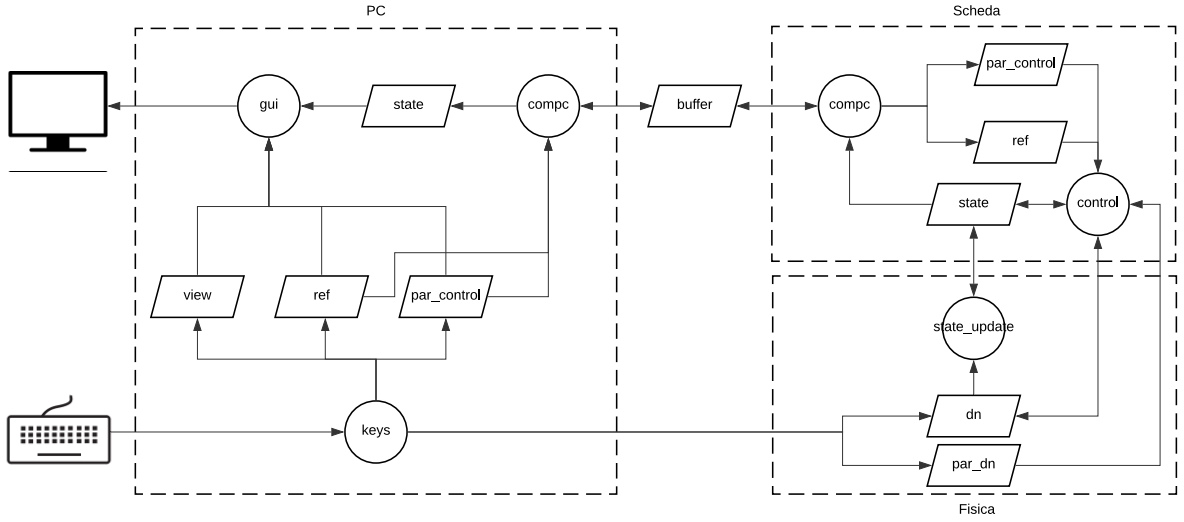

Figura 3: Diagramma delle risorse e dei task

5.2 Task

L'idea base nella suddivisione dei task è stata il voler separare l'interfaccia pc dalla simulazione della scheda e dalla simulazione della fisica del pendolo. A tal proposito sono stati creati i seguenti task:

- *state_update*, aggiorna lo stato in base al modello descritto nella sezione 2
- *control*, legge lo stato e il riferimento e modifica l'azione di controllo in base alle leggi sviluppate nella sezione 3
- *comboard*, comunica tra scheda e il buffer condiviso
- *compc*, comunica tra pc e il buffer condiviso
- *keys*, legge che tasti vengono premuti e compie azioni di conseguenza
- *gui*, disegna e aggiorna l'interfaccia utente esposta nella sezione 4

Task	Periodo	Priorità
state_update	1 ms	99
control	5 ms	90
comboard	5 ms	30
compc	5 ms	30
keys	20 ms	20
gui	40 ms	10

Tabella 2: Prioprietà task

Per differenziare ulteriormente le simulazioni delle varie componenti è stata assegnato un diverso core della CPU ad ogni gruppo di task. Si ha quindi *state_update* su un processore a lui dedicato, *control* e *comboard* su un secondo e infine *keys*, *compc* e *gui* su un terzo.

5.2.1 *state_update*

Questo task si occupa di simulare la fisica del pendolo. Essendo impossibile eseguire l'aggiornamento dello stato in modo continuo, la velocità di esecuzione è molto importante per approssimare in modo appropriato il modello discreto. Per questo motivo è il task di maggior priorità e con minore tempo di esecuzione.

La funzione principale di questo task è `physics` ottenuta tramite l'Embedded Coder di Simulink–Matlab.

5.2.2 *control*

I suoi scopi sono quelli di simulare le azioni del microcontrollore tramite la funzione `controller` e quello di aggiornare la structure `dn` per mezzo della funzione `disturbance and noise`.

Le funzioni `controller` e `disturbance and noise` sono state ottenute dall'uso dell'Embedded Coder di Simulink–Matlab. La prima, con le opportune modifiche, è la funzione da caricare sul microcontrollore stesso. La priorità di *control* è secondaria solo alla simulazione fisica.

5.2.3 *compc* e *comboard*

Questi sono i due task di comunicazione, *compc* lato pc e *comboard* lato scheda. Il primo si occupa di scrivere il riferimento e i vari parametri di controllo sul buffer e di leggere lo stato del pendolo in memoria sul buffer. Il secondo in modo opposto, legge il riferimento e i parametri di controllo e scrive lo stato.

5.2.4 *keys*

Si occupa delle varie interazioni tra tastiera e le strutture dati a cui ha accesso il pc. I vari tasti che modificano riferimento, parametri e vista sono visibili nell'interfaccia accanto alla variabile che vanno a modificare e in basso per i tasti di reset.

5.2.5 *gui*

Scrive su schermo tutte le statistiche relative ai task, i parametri, i riferimenti e lo stato. Inoltre si occupa di disegnare le varie viste descritte nella sezione 4.

A differenza dei precedenti questo task non è coinvolto nella simulazione del pendolo, la sua esecuzione è quindi secondaria e il suo periodo può essere rilassato fino a valori che siano sufficienti a visualizzare le informazioni su schermo senza troppa latenza. Il periodo è stato quindi scelto per avere 25 frame al secondo.

6 Test e Conclusioni

Avendo effettuato una ripartizione dei task su core distinti non è possibile effettuare un'analisi classica del U_{lub}. Dai dati raccolti, però, si evince come questa separazione e il basso fattore di utilizzazione dei singoli task rendano fattibile la schedulazione: ciò si riflette su un esiguo numero di deadline miss.

Il controllore finale è risultato adeguato a rumori di misura, a ritardi modesti e riesce a recuperare le posizioni desiderate dopo l'azione di piccole forze esterne. Andando a modificare i vari parametri è stato osservato che la variazione più critica è senz'altro il cambiamento del periodo del task di controllo. Questo infatti ha al suo interno delle discretizzazioni del modello fisico e andando a cambiare tempo di campionamento le funzioni di trasferimento usate non sono più valide.