POLITECNICO DI MILANO

Corso di Laurea in Ingegneria Informatica

Dipartimento di Elettronica e Informazione

PROGETTO E IMPLEMENTAZIONE DEL SISTEMA DI CONTROLLO PER UN PENDOLO INVERSO

Laboratorio di Automatica del Politecnico di Milano

Relatore: Prof. Sergio Bittanti

Correlatore: Prof. Fabio Previdi

Tesi di Laurea di:

Marco Triverio, matricola 676863

Anno Accademico 2007-2008

Stay hungry. Stay foolish.

Sommario

Il punto di partenza di questo Elaborato è la modellizzazione di un pendolo inverso al fine di studiarne la problematica di controllo.
Lo scopo del lavoro è la ricerca di un controllore lineare, che sappia mantenere l'equilibrio del sistema dati ragionevoli condizioni iniziali e disturbi esterni, e di un ricostruttore dello stato, che permetta di ottenere il valore delle variabili di stato di cui non si possiede una misura diretta.
Le simulazioni eseguite tramite `Matlab` e `Simulink` hanno mostrato un comportamento compatibile con le specifiche imposte e la realizzabilità del controllore è stata verificata sviluppando un prototipo presso il *Laboratorio di Automatica del Politecnico di Milano*: tale implementazione ha sfruttato attuatori e componentistica hardware a costo relativamente basso e presenta buona riutilizzabilità grazie alla libreria software sviluppata.

Ringraziamenti

Vorrei prima di tutto ringraziare chi mi ha aiutato in questo mio lavoro: naturalmente non solo il Prof. Bittanti e il Prof. Previdi, ma anche Piero, per i consigli a 360 gradi, Renato, per la spiegazione sugli encoder, Filippo, per aver chiesto a Renato, e Seth, per un piccolo suggerimento.

Ma esistono diverse persone che, pur non avendomi aiutato *direttamente* con la tesi, in un modo o nell'altro hanno reso fantastici questi tre anni: magari perché hanno finanziato dalle tasse universitarie ai concerti la sera, magari perché mi hanno fatto divertire dentro e fuori l'università o magari perché mi sopportano e supportano in ogni momento.
Non posso poi non citare le vere fonti di ispirazione del mio lavoro, Maccio e Rosario.

Un bacio infine alla reginetta dei pancake.

Indice

Elenco delle figure

Capitolo 1

Introduzione

1.1 Inquadramento generale e breve descrizione

Il seguente lavoro di tesi incomincia con la scelta di un problema di controllo, che viene studiato applicando le conoscenze acquisite nei corsi legati all'ambito dell'Automatica ed approfondite in corso d'opera, e che viene infine realizzato attingendo a capacità informatiche ed elettroniche. In particolare il punto di partenza di questo Elaborato è la modellizzazione di un pendolo inverso al fine di studiarne la problematica di controllo. Lo scopo del lavoro è la ricerca di un controllore lineare, che sappia mantenere l'equilibrio del sistema dati ragionevoli condizioni iniziali e disturbi esterni, e di un ricostruttore dello stato, che permetta di ottenere il valore delle variabili di stato di cui non si possiede una misura diretta.

Tale ricerca è stata convalidata da considerazioni teoriche, elaborazioni tramite Matlab e simulazioni tramite Simulink; i risultati ottenuti sono stati applicati prima in simulazione al sistema non lineare e, in seguito ad alcune tarature, anche al sistema fisico appositamente realizzato presso il Laboratorio di Automatica del Politecnico di Milano.

Per garantire la possibilità di riutilizzare lo stesso hardware per ricerche

future il software di controllo è stato basato su una libreria *open source* appositamente scritta.

1.2 Struttura della tesi

La tesi è strutturata nel modo seguente:

- Nella sezione due si illustra il processo di modellizzazione del sistema fisico.
- Nella sezione tre si descrive il processo di linearizzazione del modello e la sintesi del controllore (con valutazione delle prestazioni).
- Nella sezione quattro viene studiato il ricostruttore dello stato.
- Nella sezione cinque si mostra una realizzazione del sistema studiato e si analizza la bontà dello studio fatto.
- Nella sezione sei si riassumono i risultati raggiunti e si presentano alcune possibili direzioni di lavoro.
- Nell'appendice A si riporta il codice C e Matlab utilizzato

Capitolo 2

Modellizzazione del sistema

Il primo passo da compiere verso la costruzione del robot è dato dalla modellizzazione del sistema fisico: questa ci fornirà quattro equazioni di stato che saranno il fondamento necessario per la sintesi del controllore e le simulazioni tramite *Simulink*.

2.1 Stato dell'arte

Un pendolo **semplice** è costituito da un filo inestensibile a cui è appeso un punto materiale di massa che può oscillare attorno a un punto fisso detto *polo*: la componente della forza peso lungo il filo controbilancia la tensione del filo stesso, mentre la componente della forza peso perpendicolare al filo funge da forza di richiamo e produce il moto oscillatorio del pendolo.
Il pendolo **inverso** rappresenta un pendolo semplice rovesciato, rigido e privo di punto fisso: la parte più bassa può dunque muoversi per bilanciare le oscillazioni della parte più alta e garantire così l'equilibrio; il problema di controllo si riconduce dunque a volere stabilizzare la posizione di un'asta vincolata ad un carrello libero di traslare lungo una guida.
La modellizzazione del pendolo inverso è sicuramente un problema ben noto

e sono molte le fonti che riportano studi da cui derivano modelli con diversi livelli di approssimazione; ciò che è stato fatto è svolgere lo studio indipendentemente, validare quanto fatto attingendo da diverse fonti e scegliere quali dettagli trascurare riferendosi a quale sarebbe stata la realizzazione pratica.

La schematizzazione utilizzata è quella mostrata in figura 2.1. Il testo di

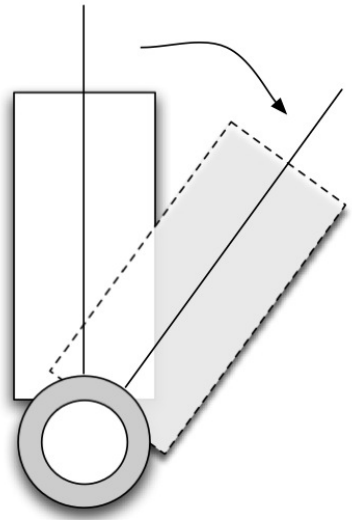

Figura 2.1: Schematizzazione del robot

riferimento per la modellizzazione è stato il sesto capitolo di "*Lectures on Dynamic Systems and Control*" [7].

2.2 Modellistica del pendolo inverso

Per determinare il modello fisico è necessario isolare le forze che agiscono sul pendolo inverso; per chiarezza è utile scomporre il robot in due parti, alta e bassa.

2.2.1 Carrello

La parte inferiore del pendolo inverso è rappresentata da ruote e piano di appoggio per i componenti. Le forze che agiscono *verticalmente* ven-

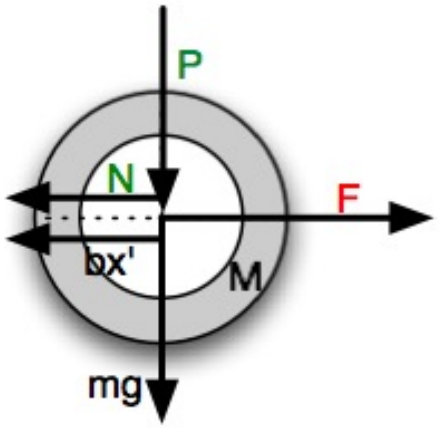

Figura 2.2: Bilancio forze parte inferiore

gono compensate dalla reazione del terreno; le forze che invece agiscono *orizzontalmente* sono date dall'equazione:

$$M\ddot{x} = F_{in} - N - b\dot{x} \tag{2.1}$$

Come rappresentato in figura 2.2, dove:

F_{in} è la forza motrice

N *e* **P** sono le forze di interazione con la parte superiore

M è la massa della parte inferiore

g è l'accelerazione gravitazionale

b$\dot{x}$ è l'attrito

2.2.2 Parte mobile

La parte superiore è rappresentata dal corpo del robot (che conterrà la scheda di controllo, il giroscopio e la componentistica elettronica).
Il bilancio delle forze che agiscono *orizzontalmente* è dato dall'equazione:

$$N = m\ddot{x} + ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2\sin\theta \tag{2.2}$$

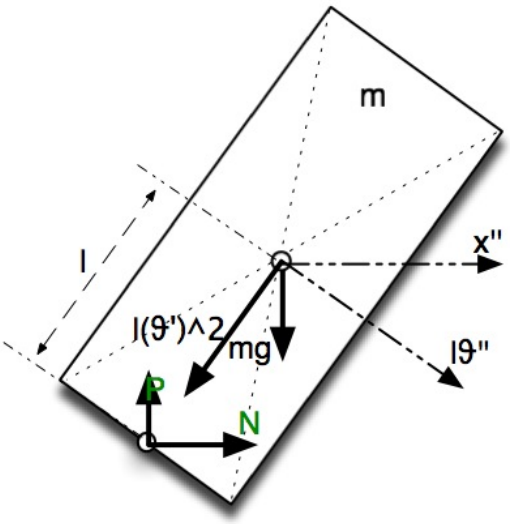

Figura 2.3: Bilancio forze parte superiore

Dove:

$m\ddot{x}$ forza data dall'accelerazione del robot

$l\ddot{\theta}\cos\theta$ forza data dall'accelerazione angolare

$l\dot{\theta}^2\sin\theta$ forza data dall'accelerazione centripeta

Analizzando invece l'equilibrio *verticale* della parte superiore:

$$\begin{aligned} P - mg &= m\frac{\delta^2 l\cos\theta}{\delta t^2} \\ P - mg &= -ml\dot{\theta}^2\cos\theta - ml\ddot{\theta}\sin\theta \\ P\sin\theta - mg\sin\theta &= -ml\dot{\theta}^2\cos\theta\sin\theta - ml\ddot{\theta}\sin^2\theta \end{aligned} \tag{2.3}$$

Considerando il centro di massa della parte superiore è possibile scrivere l'equazione che descrive l'equilibrio dei momenti:

$$I\ddot{\theta} = Pl\sin\theta - Nl\cos\theta \tag{2.4}$$

dove I è il momento d'inerzia rispetto al baricentro della parte superiore.

Nota di approfondimento: motivazione di una semplificazione Sviluppare i calcoli a questo punto vorrebbe dire riscrivere l'equazione 2.4 come:

$$\begin{aligned} I\ddot{\theta} &= l\sin\theta(mg - ml\dot{\theta}^2\cos\theta - ml\ddot{\theta}\sin\theta) + \\ &- l\cos\theta(m\ddot{x} - ml\dot{\theta}^2\sin\theta - ml\ddot{\theta}\cos\theta) \end{aligned} \tag{2.5}$$

e ottenere, semplificando:

$$(I + ml^2)\ddot{\theta} = mgl\sin\theta - ml\ddot{x}\cos\theta \tag{2.6}$$

A questo punto bisognerebbe sostituire quanto trovato all'interno dell'equazione 2.9.
Tale procedimento è sicuramente il più corretto ma, nel nostro caso, non porta necessariamente ad una modellizzazione migliore; infatti, considerando l'implementazione che verrà fatta si possono stimare alcuni parametri:

M la parte inferiore pesa all'incirca 0.285kg

m la parte superiore pesa all'incirca 0.09kg

l l'altezza della parte superiore è di 18 cm dunque $l = 9cm$

I il momento d'inerzia può anch'esso essere stimato sia sperimentalmente (a tal proposito si veda [12]) che numericamente (sfruttando le relazioni mostrate in [1] e [6])

In entrambi i casi il valore di I risulta *numericamente* trascurabile; si può dunque proseguire dalla equazione:

$$ml^2\ddot{\theta} = mgl\sin\theta - ml\ddot{x}\cos\theta \tag{2.7}$$

2.2.3 Equazioni del modello

L'equazione 2.7 che può anche essere scritta come:

$$\ddot{\theta} = \frac{1}{l}(g\sin\theta - \ddot{x}\cos\theta) \tag{2.8}$$

Unendo le equazioni 2.1 e 2.2 si ottiene:

$$(M+m)\ddot{x} + ml\ddot{\theta}\cos\theta = ml\dot{\theta}^2\sin\theta + F_{in} - b\dot{x} \tag{2.9}$$

A questo punto è possibile utilizzare quanto trovato nell'equazione 2.8 all'interno dell'equazione 2.9:

$$\begin{aligned} (M+m)\ddot{x} &+ m\cos\theta(g\sin\theta - l\ddot{x}\cos\theta) = ml\dot{\theta}^2\sin\theta + F_{in} - b\dot{x} \\ \ddot{x} &= \frac{ml\dot{\theta}^2\sin\theta - mg\sin\theta\cos\theta + F_{in} - b\dot{x}}{M + m\sin^2\theta} \end{aligned} \tag{2.10}$$

Sostituendo nella 2.8:

$$\begin{aligned} \ddot{\theta} &= \frac{1}{l}\left(g\sin\theta - \cos\theta\frac{ml\dot{\theta}^2\sin\theta - mg\sin\theta\cos\theta + F_{in} - b\dot{x}}{M + m\sin^2\theta}\right) = \\ &= \frac{Mg\sin\theta + mg\sin\theta(\sin^2\theta + \cos^2\theta) - ml\dot{\theta}^2\sin\theta\cos\theta - F_{in}\cos\theta + b\dot{x}\cos\theta}{l(M + m\sin^2\theta)} = \\ &= \frac{(M+m)g\sin\theta - ml\dot{\theta}^2\sin\theta\cos\theta - F_{in}\cos\theta + b\dot{x}\cos\theta}{l(M + m\sin^2\theta)} \end{aligned} \tag{2.11}$$

Possiamo a questo punto scrivere il sistema:

$$\begin{cases} \dot{x} &= \dot{x} \\ \ddot{x} &= \frac{ml\dot{\theta}^2\sin\theta - mg\sin\theta\cos\theta + F_{in} - b\dot{x}}{M + m\sin^2\theta} \\ \dot{\theta} &= \dot{\theta} \\ \ddot{\theta} &= \frac{(M+m)g\sin\theta - ml\dot{\theta}^2\sin\theta\cos\theta - F_{in}\cos\theta + b\dot{x}\cos\theta}{l(M + m\sin^2\theta)} \end{cases} \tag{2.12}$$

Il sistema è dunque del quarto ordine e presenta quattro variabili di stato: la posizione x, la velocità $\dot{x}$, l'angolo di piega θ e la velocità angolare $\dot{\theta}$.

2.3 Proprietà del sistema

2.3.1 Equilibrio

É possibile notare che esiste un valore $\overline{\boldsymbol{x}}$ del vettore di stato $\boldsymbol{x} = \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix}$ e un valore $\overline{u}$ dell'ingresso F_{in} per cui $\dot{\boldsymbol{x}} = \mathcal{F}(\overline{\boldsymbol{x}}, \overline{u}) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$.

Esiste dunque un punto di equilibrio in corrispondenza dei valori $\overline{\boldsymbol{x}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ e $F_{in} = \overline{u} = 0$.

2.3.2 Linearizzazione

Se le oscillazioni della parte superiore avvengono attorno al punto di equilibrio ricavato al paragrafo 2.3.1 e hanno entità limitata si può linearizzare il sistema 2.12 ottenendo:

$$\begin{cases} \dot{x} &= & \dot{x} \\ \ddot{x} &= & -\frac{b}{M}\dot{x} - \frac{mg}{M}\theta + \frac{F_{in}}{M} \\ \dot{\theta} &= & \dot{\theta} \\ \ddot{\theta} &= & \frac{b}{lM}\dot{x} + \frac{(M+m)g}{lM}\theta - \frac{F_{in}}{lM} \end{cases} \tag{2.13}$$

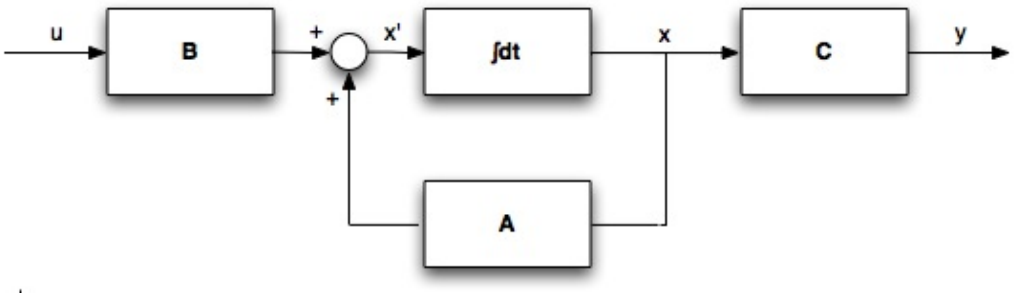

Figura 2.4: Rappresentazione dell'equazione in forma di stato

Scrivendo in forma matriciale:

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}F_{in} \tag{2.14}$$

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{b}{M} & -\frac{mg}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{b}{lM} & \frac{(M+m)g}{Ml} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ -\frac{1}{Ml} \end{bmatrix} F_{in} \tag{2.15}$$

Ci troviamo di fronte ad un sistema rappresentabile come in figura 2.4, in cui:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{b}{M} & -\frac{mg}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{b}{lM} & \frac{(M+m)g}{Ml} & 0 \end{bmatrix} \tag{2.16}$$

$$B = \begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ -\frac{1}{Ml} \end{bmatrix} \tag{2.17}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.18}$$

C rappresenta, come verrà spiegato nel capitolo 2.3.5, le variabili di stato di cui è disponibile la misura.

2.3.3 Stabilità

Il polinomio caratteristico del sistema linearizzato è dunque:

$$det(s\boldsymbol{I} - \boldsymbol{A}) = det \begin{bmatrix} s & -1 & 0 & 0 \\ 0 & s + \frac{b}{M} & \frac{mg}{M} & 0 \\ 0 & 0 & s & -1 \\ 0 & 0 & -\frac{(M+m)g}{lM} & s \end{bmatrix} \tag{2.19}$$

Trascurando l'attrito (rappresentato dal coefficiente b) si ottiene:

$$\begin{aligned} det(s\boldsymbol{I} - \boldsymbol{A}) &= det \begin{bmatrix} s & -1 & 0 & 0 \\ 0 & s & \frac{mg}{M} & 0 \\ 0 & 0 & s & -1 \\ 0 & -\frac{b}{lM} & -\frac{(M+m)g}{lM} & s \end{bmatrix} = \\ &= s^2 det \begin{bmatrix} s & -1 \\ -\frac{(M+m)g}{Ml} & s \end{bmatrix} = \\ &= s^2 \left(s^2 - \frac{(M+m)g}{Ml} \right) \end{aligned} \tag{2.20}$$

Da cui si nota che il sistema ha:

- due poli nell'origine
- due poli reali $s = \pm\sqrt{\frac{(M+m)g}{Ml}}$: il sistema è dunque instabile (come si poteva facilmente immaginare dalla fisica sottostante)

2.3.4 Raggiungibilità

Definizione informale

Un sistema è *raggiungibile* se in un tempo arbitrario non infinito, esso può essere condotto da un qualunque punto dello spazio degli stati ad un qualunque altro punto.

Analisi di raggiungibilità

Analizziamo la matrice di *raggiungibilità*:

$$R = \begin{bmatrix} B & BA & BA^2 & BA^3 \end{bmatrix} = \tag{2.21}$$

$$= \begin{bmatrix} 0 & \frac{1}{M} & 0 & \frac{mg}{M^2 l} \\ \frac{1}{M} & 0 & \frac{mg}{M^2 l} & 0 \\ 0 & -\frac{1}{Ml} & 0 & -\frac{(M+m)g}{M^2 l^2} \\ -\frac{1}{Ml} & 0 & -\frac{(M+m)g}{M^2 l^2} & 0 \end{bmatrix} \tag{2.22}$$

Il rango è massimo e dunque il sistema è completamente raggiungibile.

Un'importante considerazione Un sistema completamente raggiungibile, quando retroazionato, gode di una particolare proprietà: i poli ad anello chiuso possono essere scelti a piacere e questo faciliterà di molto la sintesi del controllore.

2.3.5 Osservabilità

Definizione informale

Con il termine *osservabilità* si indica la possibilità di poter risalire allo stato iniziale conoscendo l'evoluzione dell'ingresso e della corrispondente uscita. Se un sistema è completamente osservabile esiste, dato un ingresso, una corrispondenza biunivoca fra stati iniziali e uscite e, cioè, fra l'andamento dell'uscita del sistema e la traiettoria nello spazio degli stati.

Analisi

Per analizzare l'*osservabilità* è necessario introdurre la *trasformazione di uscita* $\mathbf{y} = \mathbf{y}(\mathrm{t})$; essa dipende dallo stato attuale e nel nostro caso descrive la lettura data dai sensori: infatti, delle quattro variabili di stato delineate al paragrafo 2.2.3, solo due (la posizione x e la velocità angolare $\dot{\theta}$) potranno essere misurate.

$$\boldsymbol{y} = \boldsymbol{C}\boldsymbol{x} = \tag{2.23}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} x \\ \dot{\theta} \end{bmatrix} \tag{2.24}$$

Si può dunque scrivere la matrice di osservabilità:

$$O = \begin{bmatrix} C \\ CA \\ CA^2 \\ CA^3 \end{bmatrix} = \tag{2.25}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{(M+m)g}{lM} & 0 \\ 0 & 0 & -\frac{mg}{M} & 0 \\ 0 & 0 & 0 & \frac{(M+m)g}{lM} \\ 0 & 0 & 0 & -\frac{mg}{M} \\ 0 & 0 & \left(\frac{(M+m)g}{Ml}\right)^2 & 0 \end{bmatrix} \tag{2.26}$$

Anche in questo caso il rango è massimo per cui il sistema è osservabile.

2.4 Esperimenti in Matlab

2.4.1 Una prima analisi

Considerando le seguenti stime dei parametri:

M la parte inferiore pesa all'incirca 0.285kg

m la parte superiore pesa all'incirca 0.09kg

l l'altezza della parte superiore è di 18 cm dunque $l = 9cm$

Nella prima parte dello script Matlab `analisi_modello_robot.m`, disponibile in Appendice A.1, è stato confermato quanto fin qui dimostrato teoricamente:

- Il sistema ha un autovalore positivo ed è dunque instabile; tale verifica è stata effettuata utilizzando la funzione `eig`
- Il sistema è osservabile (semplice verifica utilizzando il comando `rank`)
- Il sistema è raggiungibile ed è dunque possibile fissare a piacimento i poli del sistema in anello chiuso (semplice verifica utilizzando il comando `rank`)

2.4.2 Confronto tra sistema lineare e sistema non lineare

Per comparare il sistema non lineare, descritto dall'equazione 2.12, e il corrispondente sistema linearizzato, descritto dall'equazione 2.13 si confronteranno i comportamenti in *anello aperto* con ingressi di vario genere.
Prima di tutto è però necessario far sì che il sistema non lineare possa essere rappresentato come un blocchetto all'interno del Simulink, un noto toolbox di Matlab per creare simulazioni in maniera grafica: a tal proposito è stata scritta una *S-function di primo livello* (riportata in Appendice A.2).
Si può dunque impostare l'esperimento Simulink come mostrato in figura 2.5.
Gli esperimenti svolti hanno coperto le seguenti casistiche:

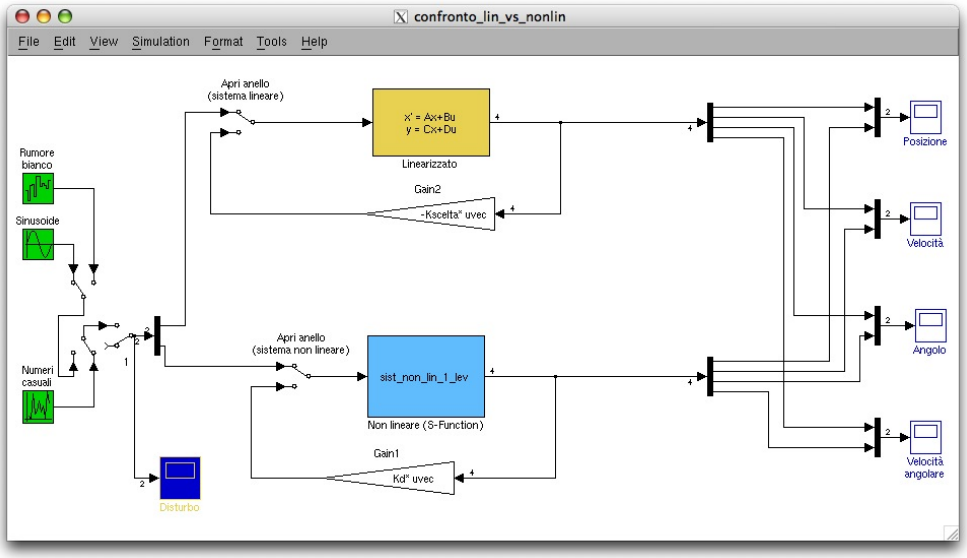

Figura 2.5: Esperimento Simulink: confronto tra sistema non lineare e il sistema linearizzato corrispondente

condizioni iniziali non nulle, nessun ingresso In questo caso si è posto l'angolo di piega a valori man mano più lontani dallo zero; come si nota dalle figure 2.6 e 2.7 l'approssimazione è buona per piccoli valori ma all'aumentare di questi la differenza di comportamento inizia a diventare evidente.

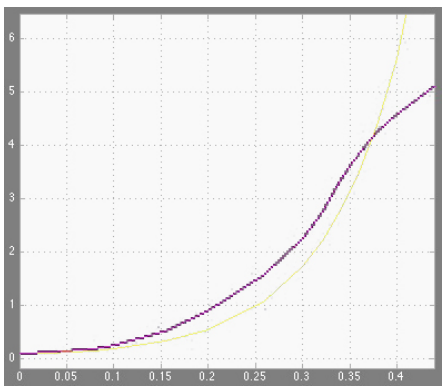

Figura 2.6: Condizione iniziale per l'angolo posta a 0.1 radianti

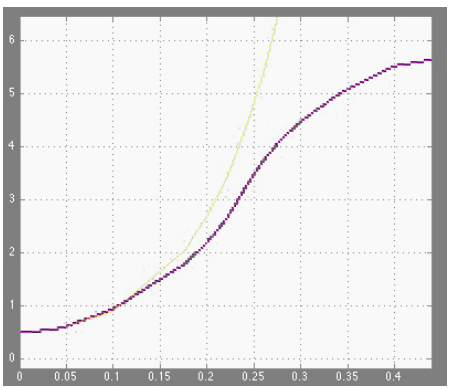

Figura 2.7: Condizione iniziale per l'angolo posta a 0.5 radianti

rumore bianco in ingresso, condizioni iniziali nulle il comportamento in questo caso è, prima che le uscite divergano, sostanzialmente identico come si evince dalle figure 2.8 e 2.9.

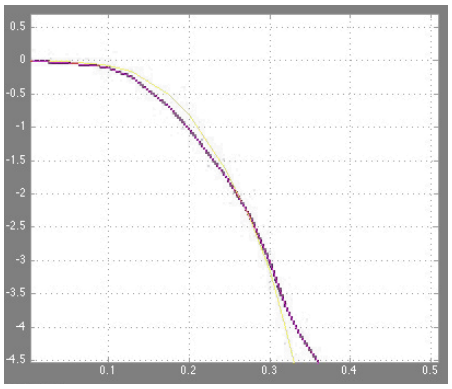

Figura 2.8: Esperimento: rumore bianco in ingresso

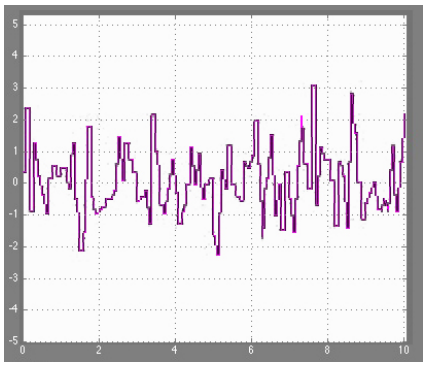

Figura 2.9: Segnale in ingresso

Si può dunque concludere che il sistema linearizzato approssimi bene il sistema non lineare soltanto nei primi istanti di funzionamento, prima della divergenza; inoltre quando si impongono delle condizioni iniziali non nulle, si riscontra un comportamento molto simile quando queste sono prossime al punto di equilibrio; man mano che ci si allontana le differenze diventano naturalmente molto marcate.

Capitolo 3

Sintesi del controllore

La modellizzazione del sistema permette di cominciare la creazione di un controllore che dovrà garantire l'equilibrio del robot rispettando i limiti fisici degli attuatori e ottimizzando il più possibile l'energia dell'azione di controllo. Il testo di riferimento per questo capitolo è "*Fondamenti di controlli automatici*" [5], i cui concetti sono stati approfonditi ricorrendo a [11].

3.1 Caratteristiche del controllore

3.1.1 Scelta di un controllore lineare

Una volta modellizzato il sistema fisico è possibile scegliere un controllore e chiudere l'anello, come mostrato in figura 3.1.
Il controllore che verrà utilizzato è di tipo proporzionale e può essere dunque rappresentato come un vettore K di quattro guadagni; l'azione di controllo è dunque proporzionale alle quattro variabili di stato (posizione, velocità, angolo e velocità angolare).

$$K = \begin{bmatrix} g_{posizione} & g_{velocita'} & g_{angolo} & g_{velocita'\ angolare} \end{bmatrix} \tag{3.1}$$

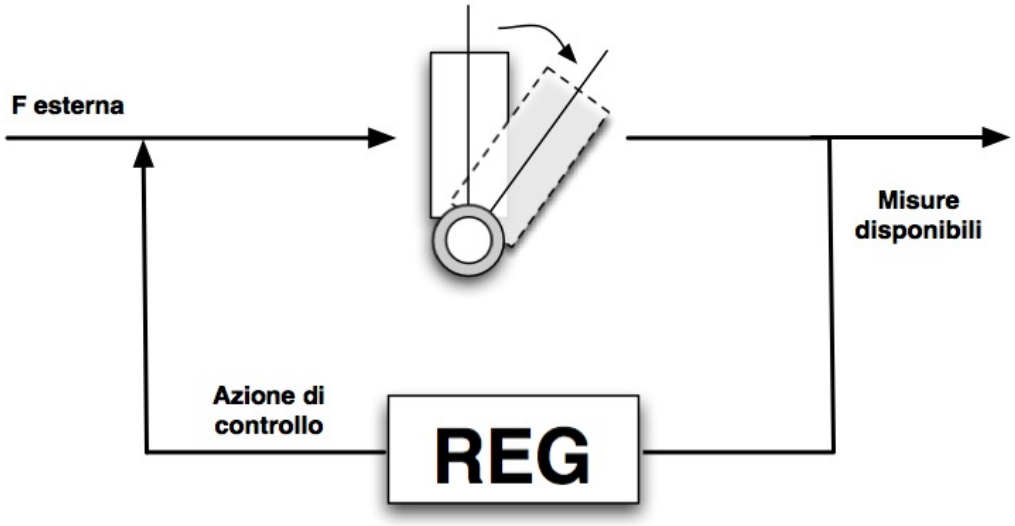

Figura 3.1: Sistema

Il sistema in anello chiuso, come mostrato in [11], può essere descritto da un'equazione di stato del tipo (le matrici A, B e C sono state definite nei capitoli 2.3.2 e 2.3.5):

$$\begin{aligned} A_{closed\ loop} &= A - BK \\ B_{closed\ loop} &= B \\ C_{closed\ loop} &= C \end{aligned} \tag{3.2}$$

Il sistema in anello chiuso può essere dunque rappresentato in due maniere del tutto equivalenti mostrate in figura 3.2.

3.1.2 Definizioni delle condizioni

Essendo il sistema che rappresenta il pendolo inverso *raggiungibile* è possibile chiudere l'anello e ricercare un controllore lineare adatto imponendo i poli ad anello chiuso.
In linea teorica è possibile imporre qualsiasi valore, ma esistono delle condizioni che limitano tale libertà: in primo luogo ci si aspetta che il controllore

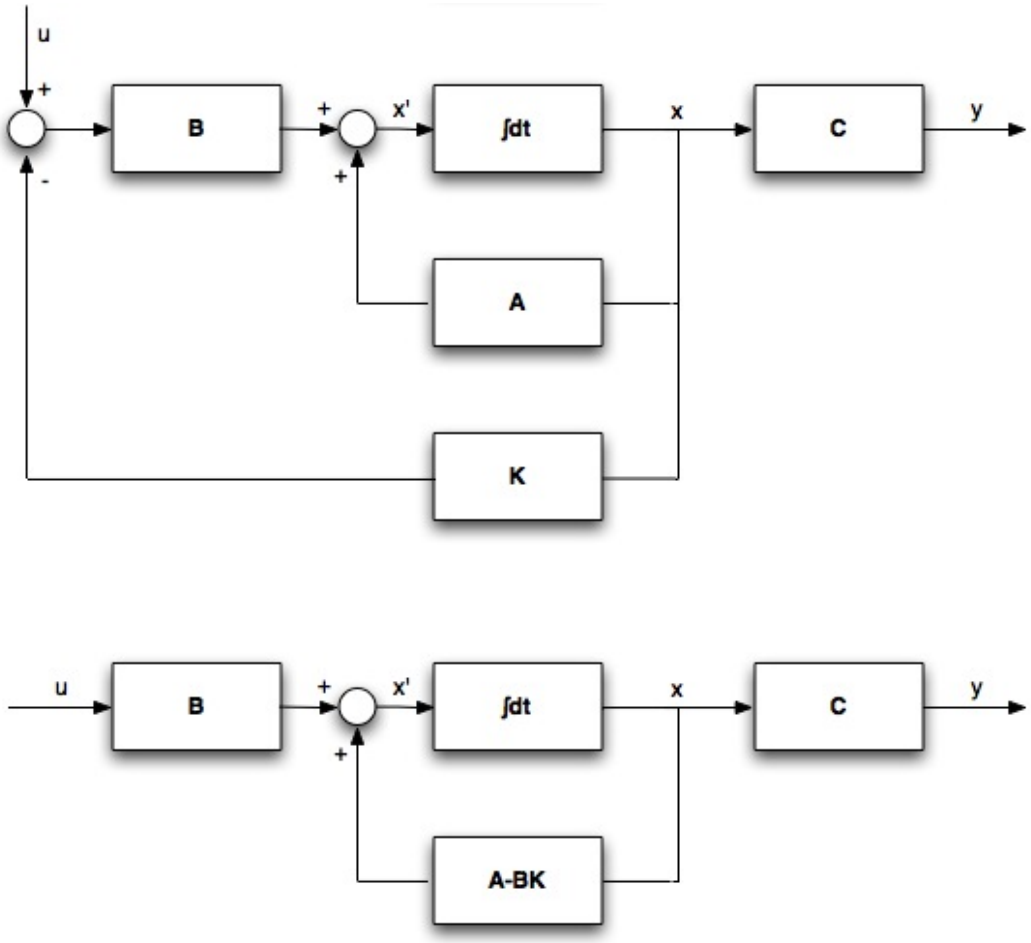

Figura 3.2: Rappresentazioni equivalenti del sistema in anello chiuso

stabilizzi il sistema in un tempo opportuno; in secondo luogo gli attuatori che andremo a utilizzare hanno dei limiti (di coppia e di potenza) che non vanno in alcun modo sormontati.

Vogliamo a questo punto definire alcune condizioni:

1. Si richiede un tempo di assestamento $T_{assestamento}$, ovvero il tempo necessario per riportare il sistema in equilibrio, compreso tra 1 e 2.5 s

    - viene specificato un range di valori che dovrebbe garantire l'equilibrio e che sia ragionevolmente compatibile con le caratteristiche degli attuatori che utilizzeremo (si veda anche il capitolo 5.2.2)

- il tempo di assestamento è dato dalla posizione del polo dominante p_{dom} della funzione di anello chiuso; sapendo che:

$$T_{assestamento} = \frac{5}{|p_{dom}|} \tag{3.3}$$

si ottiene che

$$p_{dom} = \frac{5}{T_{assestamento}} \tag{3.4}$$

$$\frac{5}{T_{assest\ maggiore}} \leq |p_{dom}| \leq \frac{5}{T_{assest\ minore}} \tag{3.5}$$

$$2 \leq |p_{dom}| \leq 5 \tag{3.6}$$

$$0.2 \leq |\tau_{polo\ dom}| \leq 0.5 \tag{3.7}$$

2. si vuole inoltre garantire che le oscillazioni del pendolo non superino i 20° di ampiezza; si vorrà dunque che:

$$|\theta(t)| \leq 20° \forall t \tag{3.8}$$

tale condizioni non può essere trasformata in un requisito sui poli ad anello chiuso e potrà dunque essere verificata solo sperimentalmente.

3. Si pone inoltre che lo *smorzamento* rientri nel seguente range:

$$0.7 \leq \xi \leq 1 \tag{3.9}$$

3.1.3 Area sulla mappa dei poli

Le condizioni 1 e 3 delineano una precisa area nel mappa dei poli: questa è mostrata in giallo in figura 3.3 e il polo dominante del sistema in anello chiuso dovrà essere disposto al suo interno.

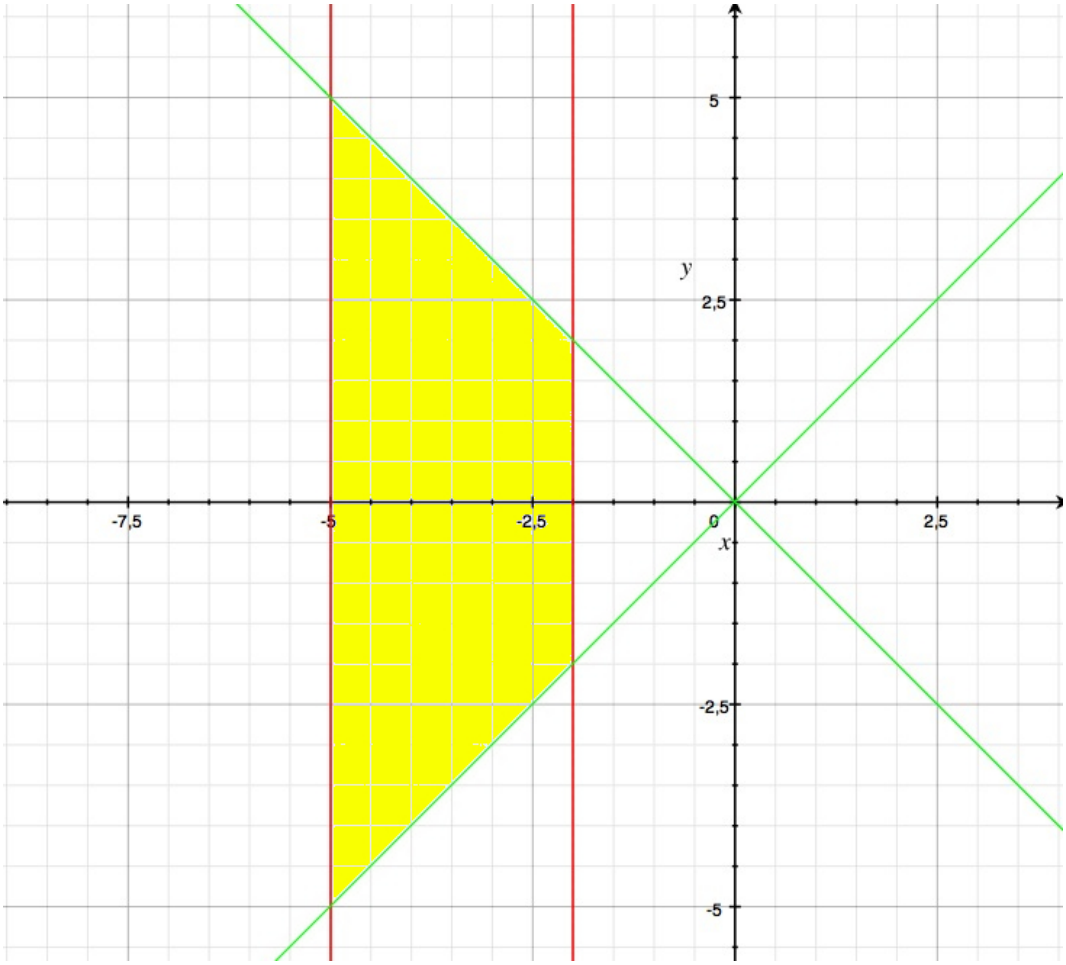

Figura 3.3: Area sulla mappa dei poli

3.1.4 Criteri di scelta

L'area delineata al capitoletto precedente non rappresenta una zona da cui scegliere indifferentemente il polo dominante; esistono infatti alcuni criteri per prediligerne uno piuttosto che un altro. Il primo è la scelta di un tempo di risposta e di uno smorzamento che garantiscano un'elongazione dell'angolo di piega del pendolo inferiore ai 20°; il secondo invece vigila sulla potenza impressa dall'azione di controllo, verificandone la compatibililità con gli attuatori. A tale proposito verranno valutati due fattori:

- l'integrale del quadrato dell'azione di controllo ($\int_0^\infty |u(t)|^2\, dt$): più

piccolo sarà e minore sarà la forza mediamente richiesta agli attuatori

- la *funzione di sensitività del controllo* $Q(s)$ (si veda anche [10]) che descrive lo sforzo di controllo: anch'essa dovrà essere la minore possibile

3.2 Esperimenti

3.2.1 Esperimenti in Matlab

Come delineato nel capitololetto 3.1.2 i poli in anello chiuso dovranno essere almeno quattro e la coppia di poli dominanti (eventualmente complessa e coniugata) dovrà rispettare precise condizioni.
Si possono naturalmente ipotizzare diversi *vettori dei poli* compatibili con tali richieste, caratterizzati da:

- tempo di assestamento più o meno veloce ($1sec \leq T_{assestamento} \leq 2.5sec$)
- poli complessi e coniugati oppure a parte immaginaria nulla
- poli su cui non si hanno condizioni più o meno "distanti" dai poli dominanti

Lo script Matlab `analisi_modello_robot.m` permette di creare la matrice degli n tentativi (di dimensione n x 4, essendo quattro i poli da fissare) e per ognuno:

- viene calcolata la matrice dei guadagni K tramite la funzione Matlab `place(A, B, poli_prescelti)`
- vengono calcolati $A_{closed\ loop}$, $B_{closed\ loop}$ e $C_{closed\ loop}$ sfruttando le relazioni 3.2
- viene svolto un primo esperimento con **condizioni iniziali non nulle** e **variabile esogena nulla**:

  - questo esperimento è sicuramente significativo in quanto rappresenta lo scenario di fronte al quale si troverà il pendolo inverso all'inizio del funzionamento
  - il vettore delle condizioni iniziali è $\begin{bmatrix} 0 \\ 0 \\ 0.45 \\ 0 \end{bmatrix}$
  - l'unica condizione iniziale non nulla è dunque relativa all'angolo; questa è espressa in *radianti* e corrisponde a circa 25° sessagesimali

- viene svolto un secondo esperimento con **condizioni iniziali nulle** e **impulso** in ingresso
  - anche questo esperimento risulta significativo in quanto rappresenta il disturbo più semplice che il pendolo può incontrare
  - in questo caso viene misurata anche l'azione di controllo e viene calcolato l'integrale $\int_0^\infty |u(t)|^2\, dt$; tale valore viene riportato nel titolo della finestra

- viene svolto un terzo esperimento con condizioni iniziali nulle e scalino in ingresso: questo esperimento risulta naturalmente meno significativo in quanto l'ingresso tipico del sistema è assimilabile più facilmente ad impulsi che non ad uno scalino

- viene calcolata la funzione di sensitività $Q(s)$ definita come:

$$Q(s) = \frac{R(s)}{1 + R(s)G(s)} \tag{3.10}$$

$$|Q(jw)|_{approssimata} = \frac{|C(jw)|}{|1 + F(jw)|} \approx \begin{cases} \frac{1}{|P(jw)|} & \omega \leq \omega_t \\ |C(jw)| & \omega > \omega_t \end{cases} \tag{3.11}$$

  e ne vengono tracciati i diagrammi di Bode.

Alcuni esempi numerici Di seguito vengono riportati alcuni dei valori numerici provati:

i. **[-60; -20; -4.1; -4]** l'energia di controllo è compatibile con gli attuatori. Il valore dell'oscillazione massima sfiora i 20°.
Questo esperimento è mostrato in figura 3.4

ii. **[-60; -20; -4+4i; -4-4i]** i poli complessi coniugati introducono delle oscillazioni ed aumentano l'energia richiesta dal controllo senza però migliorare il valore dell'oscillazione massima; dal momento che è possibile imporre tutti i poli scegliamo di non sollecitare la struttura inutilmente

iii. **[-80; -40; -4.1; -4]** in questo caso le oscillazioni si riducono a soli 10°; l'energia di controllo è però quasi doppia rispetto al caso [-60; -20; -4.1; -4]
Questo esperimento è mostrato in figura 3.5

La scelta dei poli ricade su **[-60; -20; -4.1; -4]**, che permettono un buon compromesso tra tempistica di risposta, angolo di oscillazione ed energia dell'azione di controllo.

3.2.2 Esperimenti in Simulink

Una volta selezionata la matrice dei guadagni K_{scelta} che meglio controlla il sistema in esame è necessario preparare Simulink per gli esperimenti successivi.
Simulink sicuramente brilla per la facilità con cui è possibile impostare le simulazioni ma purtroppo la gestione dei risultati non è potente quanto Matlab: per questo motivo le analisi approfondite richiederanno di tornare in Matlab.

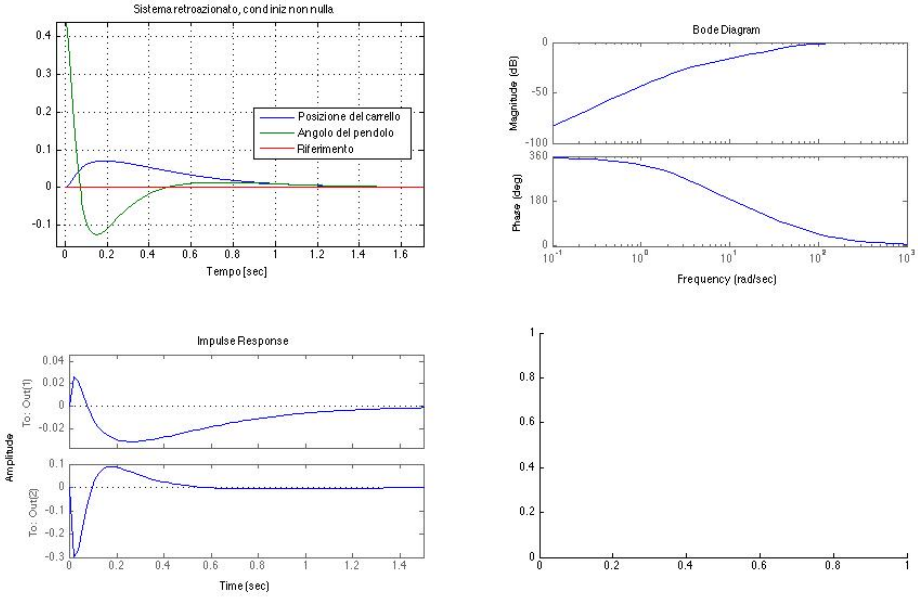

Figura 3.4: Esperimento con poli [-60; -20; -4.1; -4]

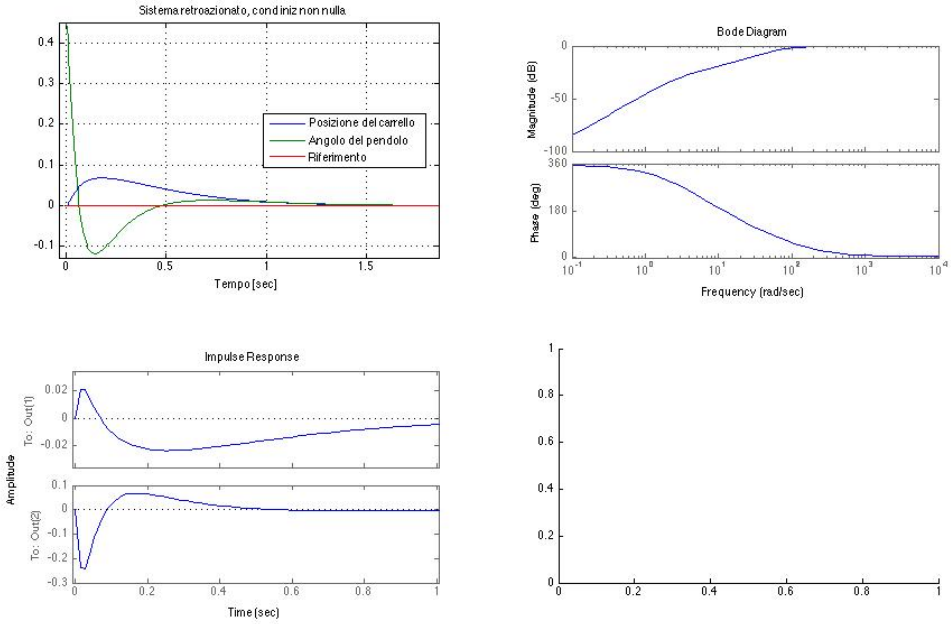

Figura 3.5: Esperimento con poli [-80; -20; -4.1; -4]

In Simulink sono stati dunque validati i risultati ottenuti nei precedenti paragrafi ripetendo alcuni degli esperimenti: gli esiti sono risultati compatibili e si è potuto dunque procedere con la realizzazione del ricostruttore dello stato.

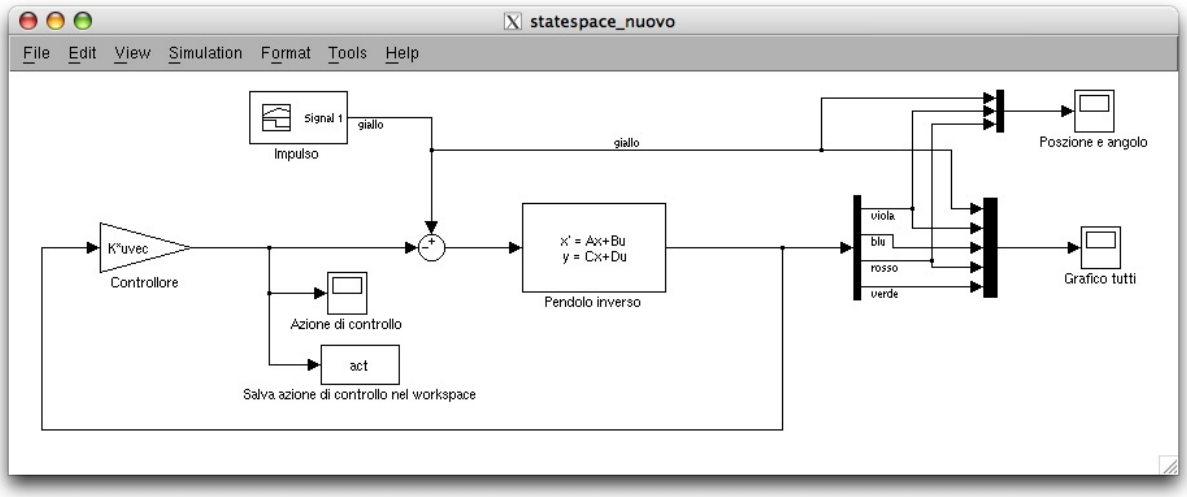

Figura 3.6: Esperimento in Simulink: impulso in ingresso con condizioni iniziali nulle

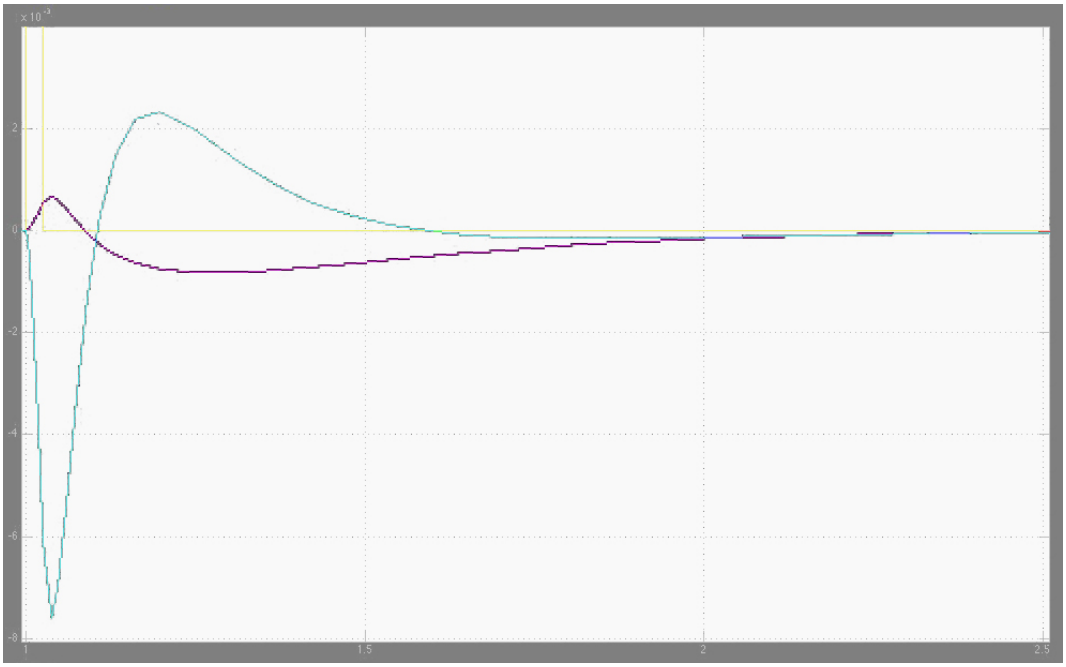

Figura 3.7: Grafico di posizione e angolo per l'esperimento di figura 3.6

Capitolo 4

Ricostruttore dello stato

In questo capitolo si continua il lavoro a partire dal controllore lineare e si sintetizza un *ricostruttore dello stato* che permetta di far convergere lo stato calcolato dal robot con lo stato reale del sistema.

4.1 Premessa teorica

4.1.1 La necessità di un ricostruttore dello stato

Il sistema che andremo a realizzare è caratterizzato da:

sistema reale rappresentato dal pendolo inverso; di questo si conosce solo ciò che i sensori misurando quindi non necessariamente tutte le variabili di stato nè, solitamente, le condizioni iniziali

modello ideale è all'interno del microcontrollore che implementerà il controllore lineare e il ricostruttore dello stato; rappresenta il modello semplificato in base a cui viene calcolata l'azione di controllo.
Esso deve non solo approssimare al meglio il sistema reale ma *deve* anche essere in grado di:

- ricostruire tutte le variabili di stato per cui non è disponibile la misura
- convergere velocemente ai valori corretti delle variabili di stato nel caso di condizioni iniziali non nulle

Come abbiamo visto nel capitolo precedente il controllore realizzato è di tipo *proporzionale lineare*. Ciò vuol dire che è nella forma:

$$K = \begin{bmatrix} g_{posizione} & g_{velocità} & g_{angolo} & g_{velocità\ angolare} \end{bmatrix} \tag{4.1}$$

Esso agisce dunque su ognuna delle quattro variabili di stato e applica sul sistema rappresentato dal pendolo inverso una forza che è combinazione lineare di queste.
Ma quali variabili di stato sono realmente misurabili? E se le condizioni iniziali non fossero esattamente nulle?

4.1.2 I sensori

Le misure a disposizione sono solamente due (si veda anche il capitolo 5.2.3):

posizione tramite gli encoder incrementali presenti all'interno dei motori (si veda anche il capitolo 5.2.2)

velocità angolare tramite un giroscopio (si veda anche il capitolo 5.2.3) viene misurato *non* l'angolo di piega ma la velocità angolare

Il controllore dunque ha la necessità di calcolare le due misure mancanti: velocità longitudinale e angolo di piega.
Per ottenerle si può seguire un approccio puramente matematico derivando la posizione e integrando la velocità angolare: derivazione e integrazione soffrono però di problemi ben noti ed è dunque preferibile procedere **ricostruendo asintoticamente lo stato** (si veda anche [11]).

4.1.3 Il ricostruttore dello stato

Il sistema reale può essere rappresentato dalle equazioni:

$$\begin{cases} \dot{\boldsymbol{x}} &= \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}F_{in} \\ \dot{\boldsymbol{y}} &= \boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}F_{in} \end{cases} \tag{4.2}$$

Nel caso in esame $\boldsymbol{D} = \boldsymbol{0}$.

Dal momento che questo sistema evolve senza la possibilità di misurare le condizioni iniziali si sfrutta il fatto che esiste un sistema nella stessa forma 4.2 che, con opportuni valori di $\boldsymbol{A}$, $\boldsymbol{B}$ e $\boldsymbol{C}$, converge allo stato del sistema reale qualunque siano la condizioni iniziali. In altre parole esiste un sistema che dati l'ingresso F_{in} e le uscite $\boldsymbol{y}$ disponibili del sistema reale, avrà uno stato $\hat{\boldsymbol{x}}$ che converge asintoticamente allo stato $\boldsymbol{x}$ del sistema reale.

Tale idea è schematizzata in figura 4.1

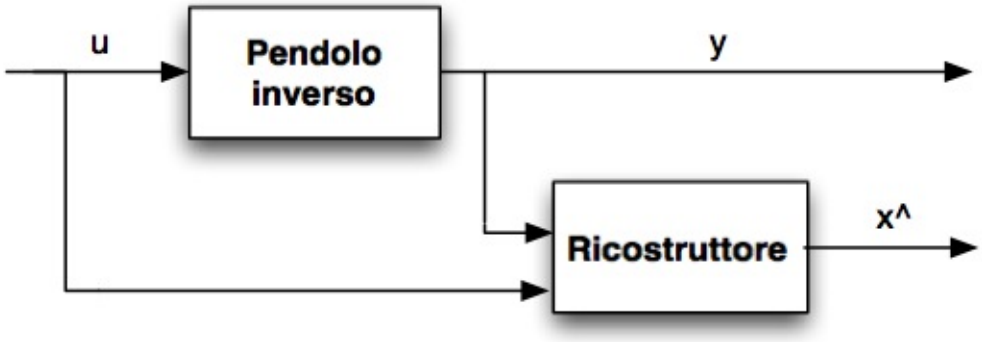

Figura 4.1: Schema generico di un ricostruttore

La realizzazione Simulink di tale concetto è data da una serie di passi:

- si rappresenta il sistema reale come lineare (come in equazione 4.2) oppure come non lineare (come verrà fatto nel 4.3.2)
- si crea una replica del sistema (lineare ed eventualmente discretizzato, come verrà fatto nel 4.3.2)

- la replica e il sistema originario avranno condizioni iniziali diverse essendo lo stato del sistema reale non noto

- si calcola l'*errore di ricostruzione dello stato* $e(t)$, dato dalla differenza tra lo stato predetto $\hat{\boldsymbol{x}}$ delle due variabili di cui si possiede una misura e lo stato reale $\boldsymbol{x}$ proveniente dai sensori

- questa quantità, opportunamente moltiplicata per una matrice dei guadagni L, viene negata e data in ingresso alla replica del sistema

- si può dunque osservare, essendo $F_{in} = -\boldsymbol{K}\hat{\boldsymbol{x}}$, come la replica sia governata dalle seguenti equazioni:

$$\begin{cases} \dot{\hat{\boldsymbol{x}}} &= \boldsymbol{A}\hat{\boldsymbol{x}} + \boldsymbol{B}F_{in} - \boldsymbol{L}(\boldsymbol{y} - \hat{\boldsymbol{y}}) \\ &= (\boldsymbol{A} - \boldsymbol{LC})\hat{\boldsymbol{x}} + \boldsymbol{B}F_{in} + \boldsymbol{Ly} \\ &= (\boldsymbol{A} - \boldsymbol{BK})\hat{\boldsymbol{x}} - \boldsymbol{L}(\boldsymbol{y} - \hat{\boldsymbol{y}}) \\ \dot{\hat{\boldsymbol{y}}} &= \boldsymbol{C}\hat{\boldsymbol{x}} \end{cases} \tag{4.3}$$

Dove:

$$\boldsymbol{E} = \boldsymbol{A} - \boldsymbol{LC} = \tag{4.4}$$

$$= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{mg}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{(M+m)g}{Ml} & 0 \end{bmatrix} - \begin{bmatrix} k_{11} & k_{21} \\ k_{12} & k_{22} \\ k_{13} & k_{23} \\ k_{14} & k_{24} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \tag{4.5}$$

$$= \begin{bmatrix} -k_{11} & 1 & 0 & -k_{21} \\ -k_{12} & 0 & -\frac{mg}{M} & -k_{22} \\ -k_{13} & 0 & 0 & 1 - k_{23} \\ -k_{14} & 0 & \frac{(M+m)g}{Ml} & -k_{24} \end{bmatrix} \tag{4.6}$$

Si ha dunque un sistema come quello rappresentato in figura 4.2.

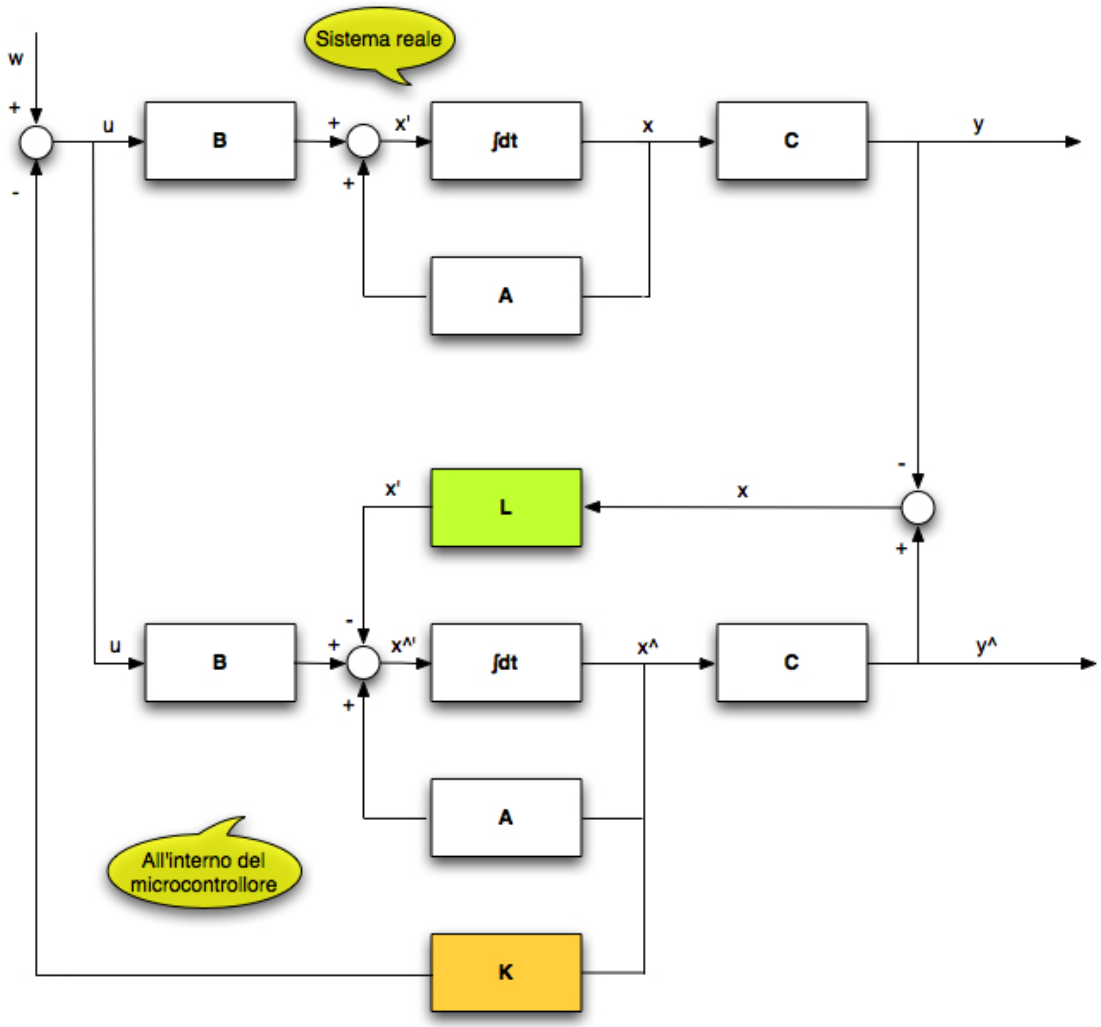

Figura 4.2: Schema del ricostruttore dello stato

4.1.4 Valutazione della matrice L

Dalle equazioni 4.2 e 4.3 è possibile ricavare la dinamica dell'errore $e(t)$; essa è descritta da:

$$\dot{e} = (\boldsymbol{A} - \boldsymbol{LC})e = \boldsymbol{E}e \tag{4.7}$$

che è l'equazione di stato di un sistema dinamico lineare nella forma 4.2 ma con ingresso nullo.

Essendo la 4.7 un problema formalmente identico al problema posto dalla 3.2 possiamo utilizzare le tecniche spiegate nel capitolo 3.2 per trovare la L che impone un certo "tempo di assestamento", ovvero il tempo che il ricostruttore impiegherà per allineare lo stato reale con quello in memoria. Nello script Matlab (il cui codice è disponibile in Appendice A.1) dunque sfrutteremo nuovamente il comando `place`.
I poli vengono imposti secondo questo criterio: dal momento che la convergenza dello stato predetto allo stato reale deve essere *molto* più veloce del tempo di risposta $T_{assestamento}$ imposto al capitolo 3.1.2, si prenderanno i poli ad anello chiuso scelti in fondo al capitolo 3.2 e li si moltiplicherà per 10.
In questo modo, essendo

$$T = \frac{5}{|polo|} \tag{4.8}$$

il tempo di annullamento dell'errore di stato sarà di un ordine di grandezza inferiore al tempo di assestamento del robot.

4.2 Alcuni esperimenti

Una volta calcolata la matrice L si può procedere con alcuni esperimenti che ne supportino la bontà.

4.2.1 Sistema reale lineare, controllore continuo

In questo primo esperimento si tratta il sistema reale come se fosse continuo e si utilizza il controllore ricavato nel precedente capitolo. Al sistema reale vengono assegnate le condizioni iniziali $\begin{bmatrix} 0 \\ 0 \\ 0.25 \\ 0 \end{bmatrix}$; le condizioni iniziali del sistema replica sono invece nulle.

Questo è l'esperimento più semplice che si possa fare e permette di iniziare a valutare le effettive prestazioni dello stimatore dello stato.
Lo schema Simulink è quello in figura 4.3.

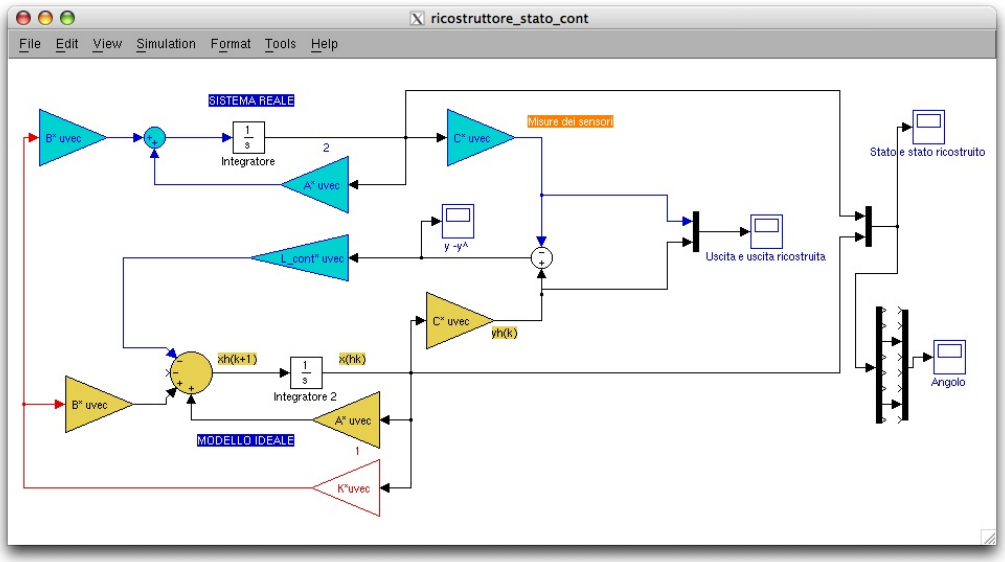

Figura 4.3: Esperimento Simulink: Sistema reale lineare, controllore continuo

Confrontando le figure 4.4 e 4.5 si può notare come l'errore converga effettivamente molto più velocemente rispetto al tempo di assestamento.

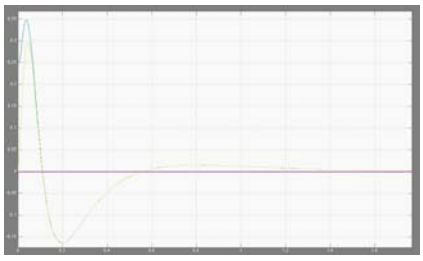

Figura 4.4: Grafico angolo predetto e reale

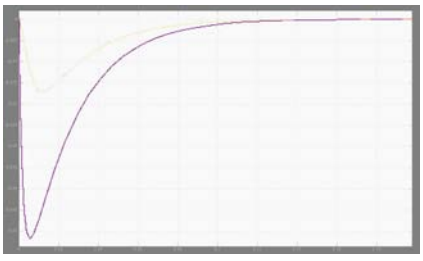

Figura 4.5: I due errori di stato

4.2.2 Sistema reale non lineare, controllore continuo

In questo esperimento entra in gioco il sistema non lineare, che meglio dovrebbe approssimare il sistema reale.
Nel capitoletto 2.4.2 abbiamo già analizzato le differenze di comportamento

in anello aperto tra il sistema non lineare e il corrispondente sistema linearizzato.
Riutilizzando la *S-function* sviluppata nel capitolo 2.4.2, possiamo sostituirla alla parte superiore dell'esperimento (corrispondente al "sistema reale"). Ripetendo le simulazioni si ottengono risultati sostanzialmente identici a quelli ottenuti nel capitoletto precedente.

4.3 Discretizzazione

4.3.1 Dal tempo continuo al tempo discreto

In questo capitolo si affronta la discretizzazione del controllore; questa si basa sulla relazione che permette di passare da poli a tempo continuo a poli a tempo discreto:

$$p_{continuo} \longrightarrow p_{discreto} = e^{p_{continuo} T_{campionamento}} \tag{4.9}$$

Si vede come sia necessario decidere un tempo di campionamento $T_{campionamento}$: il pendolo inverso sarà controllato da un microcontrollore **AVR Atmega168**, come spiegato nel capitolo 5.2.1. Dunque, considerando che tale microcontrollore è capace di campionare fino a 15kSPS (*kilo Sample Per Second*) [2], il valore di:

$$\begin{aligned} T_{campionamento} &= 1ms \\ f_{campionamento} &= \frac{1000}{sec} \\ &= 1kHz \end{aligned} \tag{4.10}$$

sembrerebbe ragionevole.
Se da un lato il limite inferiore per $T_{campionamento}$ è ben definito, il limite

superiore non lo è: è necessario dunque capire quale valore è meglio non superare al fine di non introdurre oscillazioni e instabilità. A tal fine sono stati predisposti alcuni esperimenti nel capitoletto 4.3.2.

Per modificare i precedenti esperimenti Simulink in modo da poter lavorare a tempo discreto è necessario ricalcolare tutte le matrici in gioco tenendo conto che il sistema è ora discretizzato ad una certa $T_{campionamento}$, ottenendo A_d, B_d, C_d e D_d (quest'ultima è nulla).
Si passa dunque da un sistema del tipo descritto dall'equazione 4.2 a un sistema:

$$\begin{cases} \boldsymbol{x}[k+1] &= \boldsymbol{A_d x}[k] + \boldsymbol{B_d} F_{in}[k] \\ \boldsymbol{y}[k] &= \boldsymbol{C_d x}[k] + \boldsymbol{D_d} F_{in}[k] \end{cases} \tag{4.11}$$

E' a questo punto necessario calcolare i poli del sistema discretizzato sfruttando la formula 4.9; si devono poi imporre i poli ad anello chiuso (esattamente come è stato fatto nel capitolo 3.2 per il sistema a tempo continuo) e utilizzare il comando Matlab `place` (che lavora anche con sistemi a tempo discreto) per ricavare la matrice dei guadagni K_d.
In seguito bisogna ricercare nuovamente la matrice L_{disc} del ricostruttore dello stato: le modalità sono identiche a quelle mostrate nel capitolo 4.1.4.
All'interno del microcontrollore si dovrà dunque implementare la versione discreta dell'equazione 4.3, ovvero:

$$\begin{aligned} \hat{\boldsymbol{x}}[k+1] &= \boldsymbol{A_d}\hat{\boldsymbol{x}}[k] + \boldsymbol{B_d} F_{in}[k] - \boldsymbol{L_d}(\boldsymbol{y}[k] - \hat{\boldsymbol{y}}[k]) \\ &= (\boldsymbol{A_d} - \boldsymbol{B_d K_d})\hat{\boldsymbol{x}}[k] - \boldsymbol{L_d}(\boldsymbol{y}[k] - \hat{\boldsymbol{y}}[k]) \end{aligned} \tag{4.12}$$

4.3.2 Altri esperimenti in Matlab

Sistema reale lineare, controllore discreto

Per adattare i precedenti esperimenti Simulink al tempo discreto bisogna:

- sostituire ad ogni componente il corrispondente discreto

- impostare il tempo di campionamento $T_{campionamento}$ dalla finestra delle opzioni che si ottiene facendo doppio click su un qualsiasi oggetto Simulink (di default il valore è "-1" che permette agli oggetti di "ereditare" il tempo di campionamento eventualmente impostato per gli altri oggetti)

- sostituire l'integratore con un ritardo unitario

- aggiungere un blocchetto "*Zero Order Hold*" per simulare il campionamento dei segnali provenienti dai sensori

L'esperimento Simulink corrispondente è mostrato in figura 4.6.
Dalla figura 4.7 si notano invece le conseguenze della discretizzazione: i

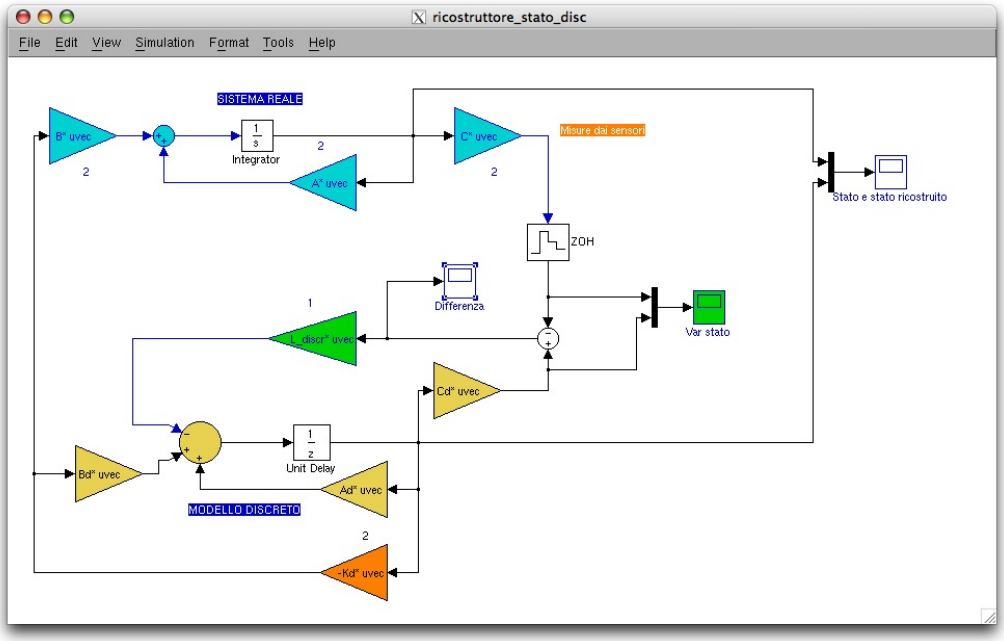

Figura 4.6: Esperimento Simulink: Sistema reale lineare, controllore discreto

tempi di risposta sono invariati ma tutti i grafici sono ora caratterizzati da una "seghettatura".

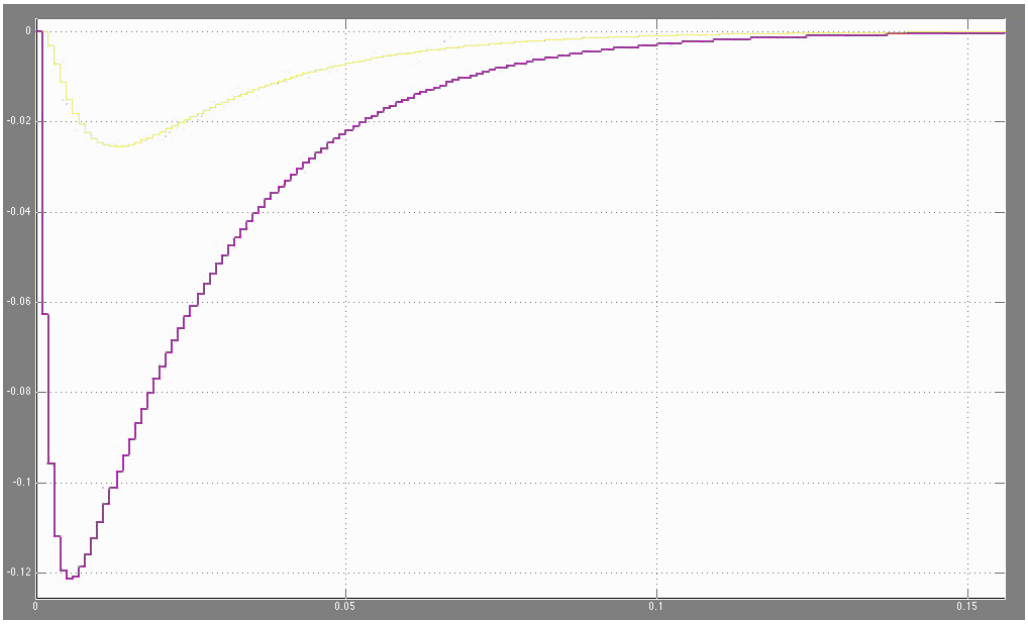

Figura 4.7: Angolo predetto e reale con controllore discreto

Sistema reale non lineare, controllore discreto

In questo esperimento entra nuovamente in gioco il sistema non lineare, che meglio dovrebbe approssimare il sistema reale.
Riutilizzando la *S-function* sviluppata nel capitolo 2.4.2 si ottengono risultati pressochè identici a quelli ottenuti nel capitoletto precedente. Esistono però due differenze da segnale: in primo luogo la condizione iniziale massima per cui il controllore riesce a garantire la stabilità del sistema reale è ridotta a 0.4 radianti; in secondo luogo si può notare come le prestazioni del controllore si degradino all'aumentare del tempo di campionamento.
Per valori vicini al valore da noi assunto ($T_{campionamento} = 1ms$) non si osservano particolari variazioni nella stabilità del controllo; più ci si avvicina però al valore "limite" di $44ms$ più il sistema diventa oscillatorio e, oltre tale valore, instabile.
Quando detto è bene riassunto dalla figura 4.8.

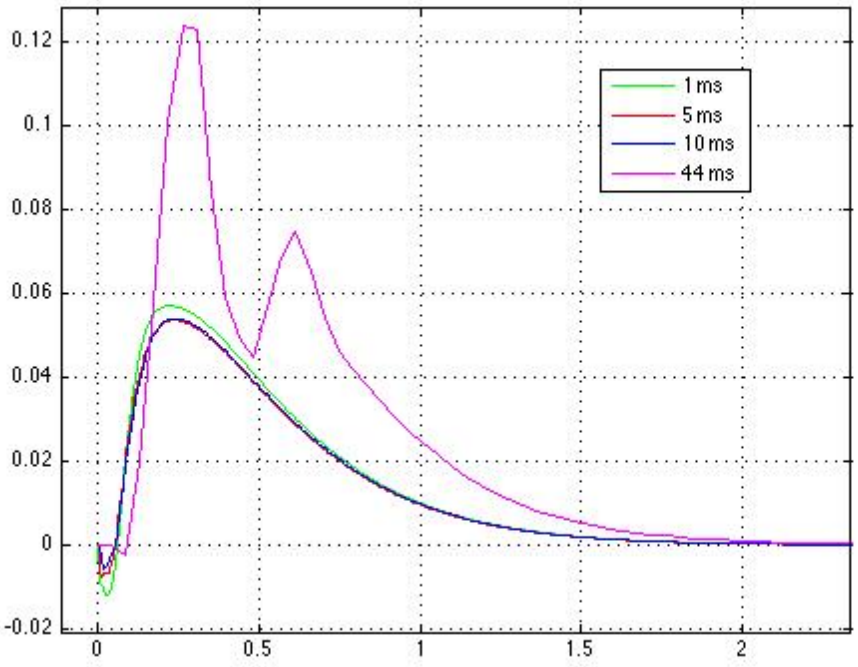

Figura 4.8: Prestazioni a confronto con tempi di campionamento tra 1 e 44 ms

Capitolo 5

Realizzazioni sperimentali

Lo studio teorico fin qui fatto viene ora validato con una realizzazione pratica: l'intento è quello di creare un robot che, essendo schematizzabile come un pendolo inverso, possa essere mantenuto in equilibrio dal controllore studiato.

5.1 Introduzione

Esistono diverse realizzazioni sperimentali di un pendolo inverso (si veda ad esempio [12]): tutte utilizzano una componentistica piuttosto costosa e propongono esperimenti che lasciano spazio a direzioni future di ricerca relative al solo ambito del controllo.

La realizzazione seguente si propone invece di creare un robot ispirato al Segway™ (mostrato in figura 5.1); questo non è dunque vincolato ad una rotaia come accade nella maggior parte degli esperimenti di laboratorio ma è autonomo e potrà

Figura 5.1: Segway attualmente in commercio

guadagnare, nei lavori futuri, la capacità di muoversi nello spazio.
Un'ulteriore differenza è la componentistica: i motori, i sensori e il microcontrollore sono tra i più economici disponibili al momento; ciononostante si ritiene che essi, entro certi limiti, possano essere adatti allo scopo.

5.2 Hardware utilizzato

5.2.1 Arduino

Il "cuore" della realizzazione sperimentale è dato dal *microcontrollore*, il componente al cui interno verranno implementati il controllore lineare e il ricostruttore dello stato.
Il microcontrollore scelto è un **Atmel AVR ATmega168**: questo è stato preferito ai *PIC* prodotti da *Microchip* non tanto per le caratteristiche tecniche ma perchè è parte di una piattaforma di sviluppo chiamata **Arduino** [4].
Con una spesa di poco superiore ad un normale *AVR* si ha infatti a disposizione una piattaforma (mostrata in figura 5.2) che:

- è formata da una scheda su cui sono presenti vari componenti fondamentali per la programmazione e utili al momento dell'utilizzo: l'ATmega, uno stabilizzatore di tensione, una porta USB, un convertitore seriale-USB, etc...
- è studiata per rendere il più semplice possibile il lavoro di programmazione; sul microcontrollore è infatti precaricato un *bootloader* che permette di scrivere in memoria i programmi direttamente via USB
- è *open-hardware* per cui sono disponibili tutte le specifiche della scheda (schema e file EAGLE) [3]
- offre *macro C* e *librerie* già pronte per semplificare la gestione dell'hardware più disparato: ad esempio è possibile interagire con scher-

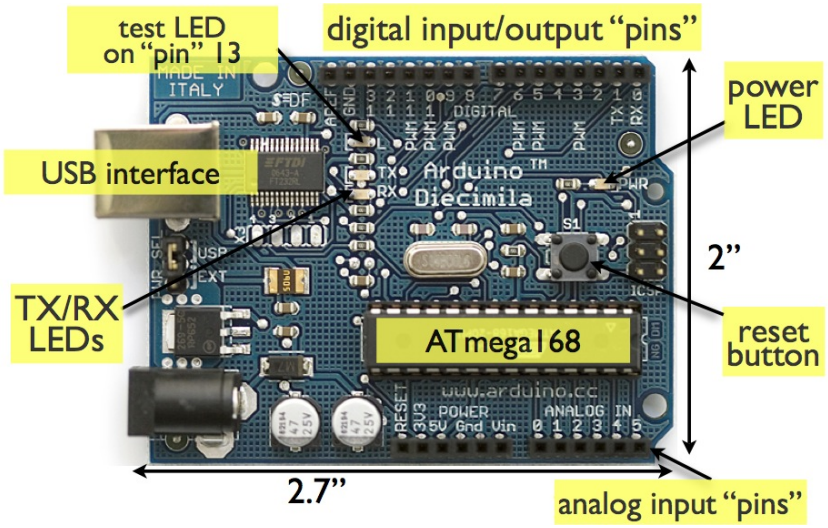

Figura 5.2: Arduino "Diecimila"

mi LCD e ricevitori GPS chiamando delle funzioni che mascherano la complessità sottostante (protocollo di comunicazione, tempistica, etc...) [4]

- offre un *Integrated Development Environment* (*IDE*) open-source, multi-piattaforma e semplice da usare
  - si collega Arduino alla porta USB
  - si scrive il programma e lo si compila con un click
  - lo si carica sul microcontrollore con un solo click
- è stata progettata e viene portata avanti da una vasta *community* internazionale; la produzione della scheda avviene inoltre interamente in Italia in industrie in cui sono garantite paghe dignitose e un ambiente di lavoro sicuro

Le principali caratteristiche tecniche di Arduino sono:

Microcontrollore	*ATmega168*
Tensione di funzionamento	*5V*
Tensione di alimentazione consigliata	*7-12 V*
Tensione di alimentazione minima e massima	*6-20 V*
Ingressi/uscite digitali	*14 (di cui 6 PWM)*
Ingressi analogici	*6*
Memoria Flash	*16 KB (2 KB usati dal bootloader)*
SRAM	*1 KB*
EEPROM	*512 bytes*
Frequenza del clock	*16 MHz*

5.2.2 Attuatori e ruote

Caratteristiche

Gli attuatori del moto (figura 5.5) e le ruote utilizzate provengono dal kit *LEGO™ Mindstorm NXT* disponibile presso il *Laboratorio di Automatica*. I motori hanno le seguenti caratteristiche (si veda anche [9]):

- doppia alimentazione fino a 9V che permette bidirezionalità
- encoder *incrementale* interno con precisione di 1°; richiede alimentazione a 5V
- velocità di rotazione senza carico con alimentazione a 9V: 170 rpm
- corrente senza carico: 60 mA
- le relazioni *round per minute*-torsione e corrente-torsione sono mostrate nelle figure 5.3 e 5.4
  - torsione con medio carico: $0.167Nm$ (con corrente 0.55 A e 117 rpm)
  - torsione massima: $\sim 0.5Nm = 0.5J$ (con corrente di 2 A)

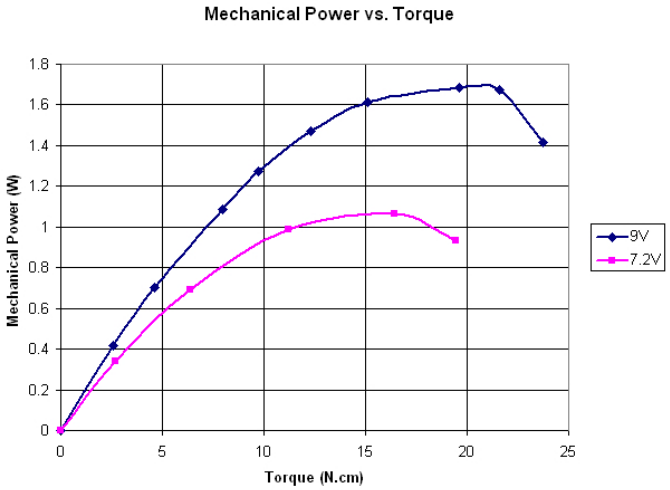

Figura 5.3: Grafico rpm-torsione

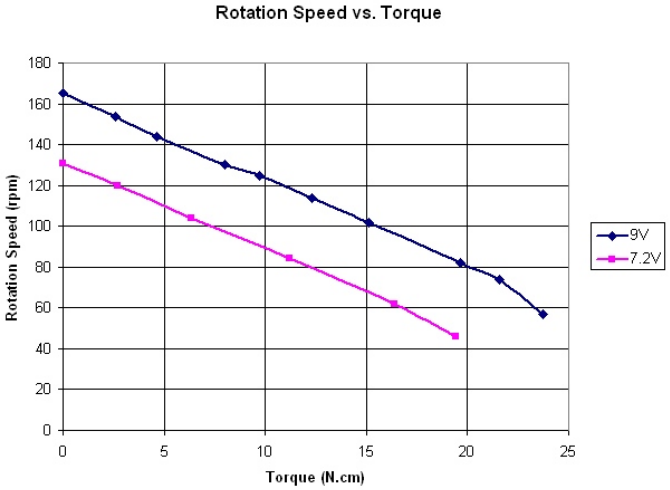

Figura 5.4: Grafico corrente-torsione

Connettori

Per comandare i motori è stato necessario fare il *reverse engineering* dei connettori e del significato dei singoli segnali al loro interno.

I motori LEGO sono infatti alimentati con un cavo a sei polarità con connettore RJ-12 *non standard* essendo il gancio di sicurezza spostato di qualche millimetro.

I segnali all'interno del cavo hanno il seguente significato:

- Primo segnale di alimentazione (da 0 a 9V)
- Secondo segnale di alimentazione (da 0 a 9V)
- Terra
- Alimentazione dell'encoder (5V)
- Primo segnale dell'encoder (onda quadra tra 0 e 5V)
- Secondo segnale dell'encoder (onda quadra tra 0 e 5V)

Ponte H

Siccome Arduino non può fornire la corrente necessaria per comandare i motori è stato necessario ricorrere ad un integrato, l'**L293D**, in grado di fornire la corrente necessaria e agire anche come Ponte H.
Tramite i quattro ingressi dell'integrato è dunque possibile comandare due motori bidirezionalmente.

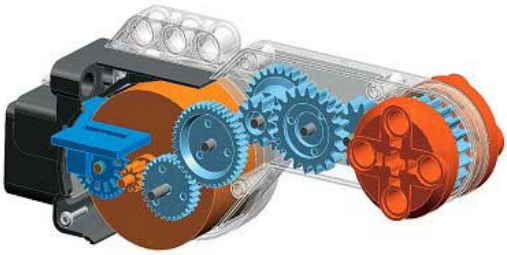

Figura 5.5: LEGO™ NXT Interactive Servo Motor

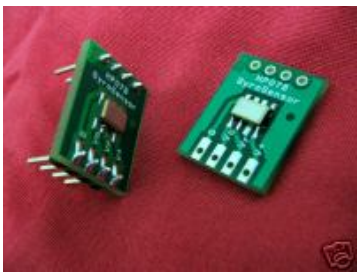

Figura 5.6: Giroscopio

5.2.3 Sensori

Encoder

I motori LEGO sono dotati di encoder incrementali interni con precisione pari a 1 grado sessagesimale.
Il segnale dell'encoder è diviso, come mostrato in figura 5.7 su due cavi:

- il primo indica con un fronte di salita o di discesa l'aumento della posizione (sia in un verso che nell'altro)
- il secondo permette di capire la direzione di rotazione

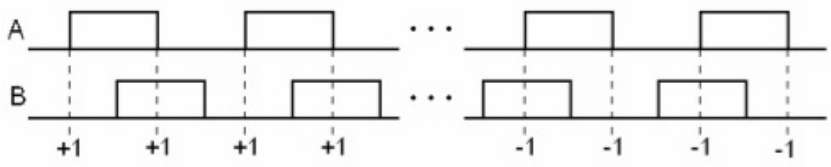

Figura 5.7: Encoder incrementale dei motori LEGO

Giroscopio

Per determinare la velocità angolare è stato utilizzato il giroscopio **Epson XV-8100CB** [8].
Questo sensore è caratterizzato da:

- essere alimentato a 3V
- rilevare al massimo $\pm 100 \frac{gradi}{s}$
- avere un fattore di scala di $2.5 \frac{mV}{\frac{gradi}{sec}}$
- avere due uscite:

  V_r è un valore costante e pari a circa 1.345V; è il voltaggio di riferimento rispetto a cui misurare V_o

  V_o ha una tensione pari a $V_r \pm V_{vel\ angolare}$

  misurando dunque la tensione tra V_o e V_r si ottiene $V_{vel\ angolare}$, un valore che è proporzionale alle velocità angolare

L'output del sensore può essere dunque sia positivo che negativo (a seconda del senso di rotazione) e avrà un range piuttosto limitato rispetto all'*Analog to Digital Converter* (ovvero l'analog in) di Arduino: questo infatti accetta

in ingresso tensioni tra 0 e V_{cc} [2] (nel nostro caso 5V) e le campiona con precisione di 10 bit (ha una *risoluzione* di 4.88mV); inoltre l'*accuratezza* dell'ADC è di ±2 LSB.
É dunque necessario amplificare il segnale e "traslarlo" per ottimizzare al massimo l'utilizzo dell'ADC; a tal proposito si sfrutta un *amplificatore differenziale con correzione di errore*, il cui schema è raffigurato nella 5.8.
Abbiamo:

$$V_{OUT} = V_{IN2}\frac{R_F + R_G}{R_G}\frac{R_2}{R_1 + R_2} + V_{REF}\frac{R_F + R_G}{R_G}\frac{R_1}{R_1 + R_2} - V_{IN}\frac{R_F}{R_G} \quad (5.1)$$

che nel caso in cui $R_2 = R_F$ e $R_1 = R_G$ diventa:

$$V_{OUT} = (V_{IN2} - V_{IN1})\frac{R_2}{R_1} + V_{REF} \quad (5.2)$$

Naturalmente si avrà:

V_{IN1} a cui verrà agganciato V_r del giroscopio

V_{IN2} a cui verrà agganciato V_o del giroscopio

V_{REF} a cui verrà agganciata una tensione costante all'incirca a metà della tensione accettata dall'ADC

In figura 5.9 è rappresentata la realizzazione pratica del circuito di condizionamento per il giroscopio: si può notare come questa presenti un potenziometro per regolare la V_{REF} dell'equazione 5.2.
La realizzazione pratica sfrutta l'**LM324**, un amplificatore operazionale particolarmente facile da reperire, economico e dall'ottimo *slew rate*.

5.2.4 Costo totale dell'hardware

Il costo totale dell'hardware è dato da:

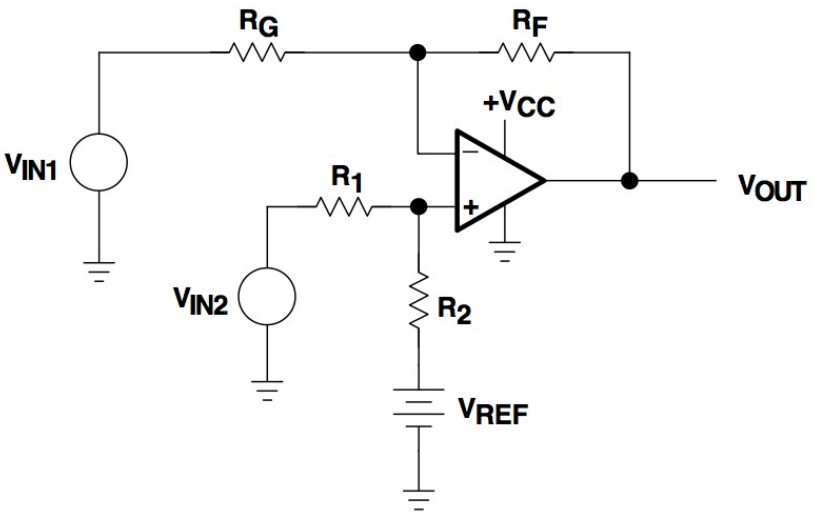

Figura 5.8: Amplificatore differenziale con correzione d'errore

Arduino NG (revision C)	16 €	+
Stabilizzatori di tensione	3 €	+
Resistenze, scheda millefori, pila,...	10 €	+
Integrati vari	4 €	+
Due attuatori lego e ruote	30 €	+
Giroscopio	30 €	=
Costo totale	**93 €**	

5.3 La libreria software `arduino2lego.h`

L' interazione con i componenti del robot (motori, encoder, giroscopio...) è stata astratta e facilitata dalla creazione di una libreria C++, denominata `arduino2lego.h`.

Con un solo comando è stato reso possibile impartire una velocità ai motori oppure leggere il valore di velocità angolare in radianti o in gradi oppure ancora conoscere i metri percorsi dal robot.

Ecco alcuni esempi di utilizzo:

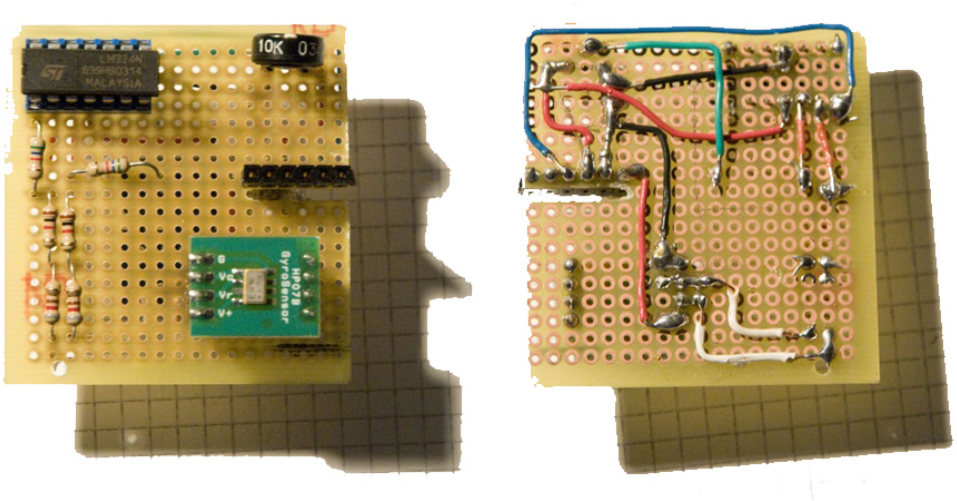

Figura 5.9: Circuito di condizionamento (fronte e retro)

`motor.setSpeed(N)` imposta la velocità di rotazione; N è un numero compreso tra -255 e 255.

`motor.setSpeedUntil(N, pos)` in questo caso la velocità indicata viene mantenuta fino a che il robot non avrà percorso il numero di centimetri specificato da *pos*

`motors.setSpeed(N)` è possibile lavorare anche su più motori contemporaneamente e utilizzare tutti i comandi disponibili sul motore singolo

`gyroscope.getAngSpeed()` ritorna la velocità angolare in *gradi* al secondo

`gyroscope.getRadSpeed()` ritorna la velocità angolare in *radianti* al secondo

`touchSensor.getValue()` comunica con il sensore di contatto LEGO e restituisce 0 o 1 a seconda che al momento della lettura stia avvenendo un contatto

Questa libreria naturalmente rappresenta una buona base per l'interazione tra componentistica LEGO (e non) e Arduino; la sua natura *open source* ne

permette l'evoluzione anche al di fuori della mia direzione.
Il progetto è stato pubblicato su **Sourceforge** e l'interfaccia è consultabile in appendice A.3.

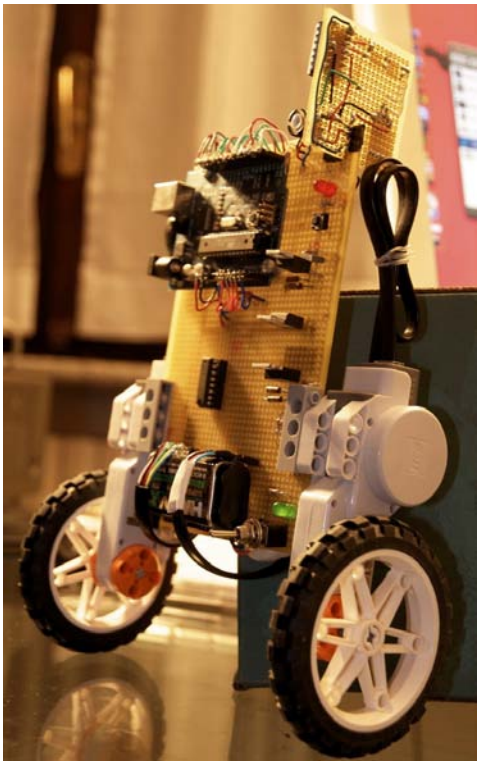

Figura 5.10: Fronte

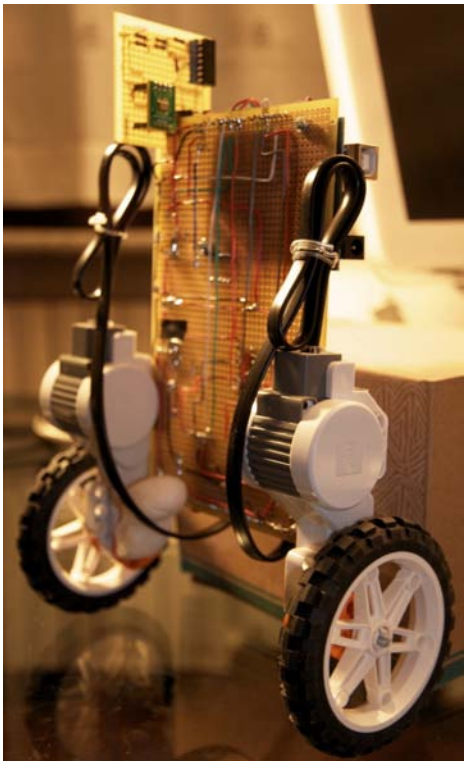

Figura 5.11: Retro

5.4 Il comportamento sperimentale

5.4.1 Implementazione di ricostruttore e controllore

Descrizione

Lo scopo dell'esperimento è verificare la realizzabilità del ricostruttore e del controllore lineare (mostrati nel capitolo 4.3.2) che, all'interno di Simulink, funzionano egregiamente.
A tal fine si è implementato un programma che:

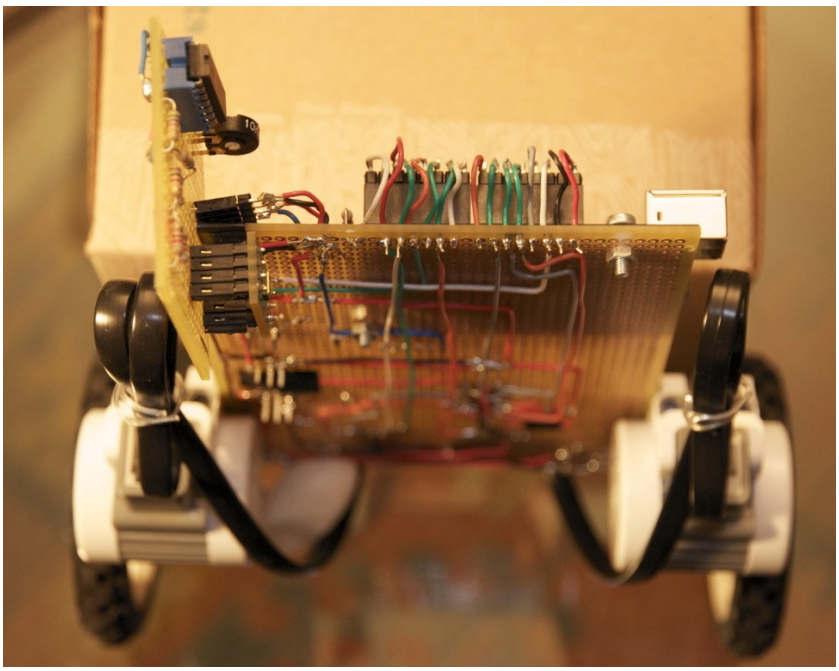

Figura 5.12: Dall'alto

- crea gli oggetti C++ necessari al funzionamento del robot (sfruttando la libreria `arduino2lego.h`)
- calibra il giroscopio
- attende la pressione di un pulsante posizionato sul corpo del robot; una pressione corta fa partire il programma normalmente mentre una pressione prolungata attiva la modalità di *debug*
- la modalità normale attiva un ciclo che:
  - viene eseguito una volta ogni n millisecondi dove n è il tempo di campionamento prescelto
  - implementa l'equazione 4.12, per comodità riporta di seguito

$$\hat{\boldsymbol{x}}[k+1] \quad = \quad (\boldsymbol{A_d} - \boldsymbol{B_d}\boldsymbol{K_d})\hat{\boldsymbol{x}}[k] - \boldsymbol{L_d}(\boldsymbol{y}[k] - \hat{\boldsymbol{y}}[k])$$

  - il vettore $\hat{\boldsymbol{x}}$ ha condizioni iniziali nulle

  - il vettore $\boldsymbol{y}$[k] rappresenta la misura data dai due sensori disponibili al momento k
  - applica ai motori una forza pari a $-\boldsymbol{K_d}\hat{\boldsymbol{x}}[\boldsymbol{k}]$

- la modaltà *debug* permette, entro limiti abbastanza stringenti (a tal proposito si veda il capitolo 5.4.3), di registrare e analizzare off-line gli stati predetti, i valori dei sensori e la forza impressa ai motori

Comportamento riscontrato

Tempo di campionamento In corso d'opera il valore del tempo di campionamento è stato alzato leggermente rispetto a quello ipotizzato nel capitolo 4.3.1 al fine di evitare alcuni casi in cui il controllore sforava il tempo previsto.

$$\begin{aligned} T_{campionamento} &= 4ms \\ f_{campionamento} &= \frac{250}{sec} \\ &= 250Hz \end{aligned}$$

Robustezza al rumore Il valore riportato dal giroscopio si è rivelato critico per la stabilità del ricostruttore: per questo motivo è stato creato un esperimento Simulink per dimostrare la resistenza al rumore; in questo è stata aggiunta alla rilevazione sensoriale un segnale costante con entità variabile (fino a $1\frac{rad}{s}$).
I risultati sono stati più che soddisfacenti: il sistema, in simulazione, non ha mai raggiunto l'instabilità.

Stabilità del ricostruttore In simulazione il ricostruttore si è mostrato non solo stabile ma anche robusto al rumore.
L'implementazione pratica invece non riesce purtroppo nell'intento; gli stati

divergono in pochi millisecondi e l'azione di controllo non è dunque efficace. Questo accade per diversi motivi:

- il sensore è risultato essere piuttosto rumoroso
- l'ADC introduce un errore dovuto all'arrotondamento
- l'ADC ha una *accuratezza* di ± 2 LSB che aggiunge incertezza alla misura

Tutti questi fattori contribuiscono a rendere instabile il ricostruttore e obbligano a seguire una strada differente: implementare il solo controllore lineare e calcolare le variabili di stato non fornite dai sensori attraverso derivazione e integrazione.

5.4.2 Implementazione del solo controllore

Descrizione

In seguito ai problemi riscontrati con il ricostruttore si è deciso di seguire una strada diversa: rinunciare al ricostruttore dello stato e procedere con un'implementazione più semplice.
Per risalire dalle due misure alle quattro variabili di stato è dunque necessario implementare nel software un derivatore e un integratore per ricavare dalla posizione e dalla velocità angolare rispettivamente i valori di velocità e angolo di piega.
Per verificare la bontà di questa idea anche nel caso in cui le condizioni iniziali fossero differenti è stato creato un esperimento Simulink, mostrato in figura 5.13: il robot viene stabilizzato per condizioni iniziali piuttosto lontane dalla posizione perfettamente verticale e anche in presenza di disturbi esterni.

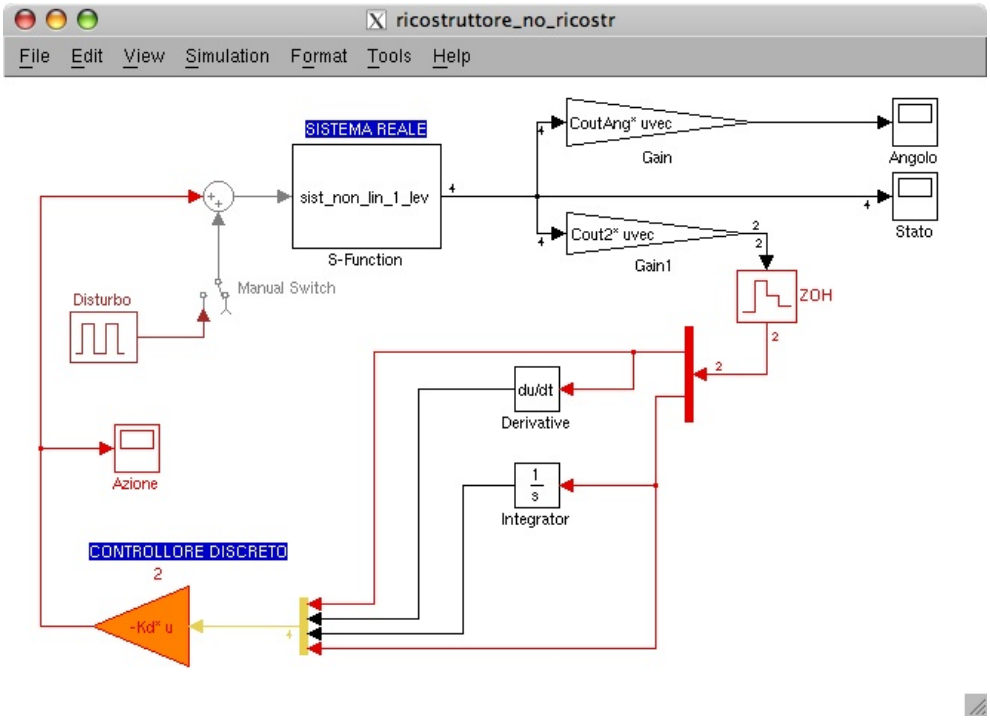

Figura 5.13: Funzionamento del controllore senza ricostruttore

Comportamento riscontrato

Il comportamento sperimentale è risultato soddisfacente: il robot ha dimostrato di poter rimanere in equilibrio.

Gli stessi motivi che causano l'instabilità del ricostruttore sono però alla base delle leggere oscillazioni che si riscontrano in particolare dopo alcuni istanti di funzionamento.

In questo caso ci si rende particolarmente conto della criticità dell'*angolo*; questo viene ricavato per integrazione e quindi fattori come:

- il rumore sul sensore
- l'errore di arrotondamento dell'ADC
- l'accuratezza dell'ADC
- gli errori dovuti agli arrotondamenti fatti dal microcontrollore

non possono che minare tale delicatissima operazione.
Se al valore reale si sovrappone un errore a media non perfettamente nulla (cosa che di fatto accade se si pensa agli arrotondamenti) l'integrazione tenderà (anche se magari in maniera piuttosto lenta) a divergere.
Alcune soluzioni a questo problema sono state date nel capitolo 5.4.4.

5.4.3 Alcuni problemi con Arduino

Arduino si è dimostrata una piattaforma estremamente flessibile per la creazione di un robot; l'ambiente di programmazione, seppure ancora in versione *alpha*, non ha dato particolari problemi.
Alcuni intoppi, particolarmente onerosi in termini di tempo, sono però sintomo del fatto che questa piattaforma non può essere usata produttivamente in ambito aziendale.

External power bug

Una delle complicazioni si è verificata nel momento in cui Arduino è stato alimentato non più tramite USB ma con un alimentatore esterno: dopo diverse ore di ricerche ho scoperto l'*external power bug*, un difetto di progettazione che impedisce di alimentare la scheda se non tramite la porta USB. La grande *community* che segue Arduino ha fortunatamente pubblicato un *fix* per le schede esistenti (*Arduino NG rev.C* e precedenti) che è poi diventato parte integrante di *Arduino NG rev.D* e *Arduino Diecimila*.

Debugging e logging

L'attività di *debugging* è risultata particolarmente complicata e ha messo in evidenza uno dei più grandi difetti della scheda.
In linea teorica il *debug* su Arduino viene affidato ad alcuni LED e alla

comunicazione tramite la porta USB (*logging*) di alcuni dati significativi: purtroppo quest'ultimo metodo è risultato pressochè inutilizzabile in quanto la trasmissione di un dato impegna il microcontrollore per tempi di gran lunga superiori al tempo di campionamento; questo obbliga a memorizzare in RAM i valori di interesse e comunicarli in blocco al computer dopo un determinato numero di secondi. Sfortunatamente 1 KByte di RAM permettono la memorizzazione di poche decine di valori, complicando enormemente un'attività semplice come la verifica dei conti.
Infine non esiste un modo semplice per capire se il programma è in esecuzione e se si è verificato qualche problema (overflow, segmentation fault, ...).
Tutte queste difficoltà potrebbero essere facilmente risolte sfruttando il pin presente sull'ATmega168 dedicato al *debug* e integrando la sua funzione all'interno dell'IDE.

5.4.4 La criticità del giroscopio

In entrambi i programmi sviluppati il giroscopio si è dimostrato l'elemento critico per la precisione del controllo.
Dalla velocità angolare è necessario, o tramite il ricostruttore o tramite integrazione, ricavare l'angolo: è facile capire che anche un piccolo errore nella stima dell'angolo può portare il robot a sbilanciarsi e a decidere l'azione di controllo basandosi su dei valori non del tutto corretti.
I seguenti problemi:

- il rumore sul giroscopio
- gli errori dati dall'ADC (arrotondamento e accuratezza)
- gli errori introdotti dai calcoli
- la conseguente imprecisione dell'integrazione

non possono essere risolti se non adottando un sensore differente.
Disporre ad esempio di un sensore che restituisca direttamente l'angolo

renderebbe la misura decisamente più precisa e eliminerebbe totalmente le oscillazioni.

Capitolo 6

Direzioni future di ricerca e conclusioni

L'esperimento ci ha permesso di capire se, con una spesa decisamente limitata, è stato possibile realizzare un esperimento solitamente svolto con strumenti di laboratorio.
Inoltre dai risultati ottenuti è possibile ipotizzare diverse direzioni di ricerca future.

6.1 Conclusioni

6.1.1 Modellizzazione

La modellizzazione si è dimostrata sufficientemente precisa: l'aggiunta di alcune variabili al modello non ha mai portato miglioramenti nè in simulazione nè al robot.
Inoltre il confronto con il sistema non lineare è risultato soddisfacente per valori di piega sufficientemente piccoli.
L'unico elemento che ha mostrato un comportamento non coerente con le simulazioni è stato il ricostruttore dello stato, che come descritto nel capi-

toletto 5.4.1, nella realizzazione pratica non è risultato stabile: la colpa è probabilmente da darsi al giroscopio, come spiegato nel capitolo 6.1.2.

6.1.2 Hardware utilizzato

Giroscopio

L'hardware utilizzato, seppur particolarmente economico, si è dimostrato all'altezza dello scopo: l'unica nota negativa è data dal giroscopio; infatti la necessità di dover integrare la misura restituita ha introdotto un errore troppo elevato.
L'unica soluzione possibile è la sostituzione del giroscopio con un sensore che restituisca direttamente l'angolo.

Arduino

La velocità computazionale di Arduino è risultata sufficiente a gestire il tempo di campionamento $T_{campionamento} = 4kHz$ che, seppur più alto rispetto a quanto inizialmente ipotizzato, è risultato sufficiente per un corretto funzionamento.
L'unico aspetto di Arduino insoddisfacente è l'ambiente di sviluppo che, mancando di un *debugger*, non permette un *testing* facile e approfondito.

Componentistica

I **motori LEGO** hanno svolto egregiamente il loro lavoro anche se l'assenza di documentazione sulle caratteristiche interne ha reso la modellizzazione meno precisa rispetto a quanto avrebbe potuto essere; inoltre il *reverse engineering* dei connettori ha richiesto parecchio tempo.
Gli **encoder LEGO**, seppur poco precisi, si sono mostrati adatti per l'applicazione in esame; il segnale standard ha facilitato la comprensione del loro funzionamento e la creazione della componente software atto a gestirlo.

Alcune note di demerito vanno invece ad alcuni **stabilizzatori di tensione** utilizzati, che sono stati soggetti a forte riscaldamento: se da un lato ciò non limita in maniera evidente l'esperimento, dall'altro hanno richiesto alcune modifiche alla scheda di controllo e potrebbero portare il robot ad una minore longevità.

6.2 Direzioni future di ricerca

Controllore

Il controllore lineare è, per sua natura, estremamente semplice. I suoi limiti possono essere notati fin dai primi esperimenti: ad esempio, imponendo delle condizioni iniziali pressoché impossibili (superiori ad 1 radiante) la simulazione arriva alla stabilità.
Il controllore lineare infatti non tiene conto dei limiti fisici degli attuatori e non può considerarsi comunque affidabile (come mostrato nel capitolo 4.3.2) per angoli di piega superiori ai 20°.
Lo studio di un controllore *LQR* oppure *PID* potrebbe portare, al prezzo di una maggiore complessità, a risultati più attinenti al sistema reale anche se non necessariamente migliori dal punto di vista pratico.

Inseguimento e traiettorie

Aggiungendo un segnale di ingresso $w(t)$ è possibile sviluppare il problema dell'*inseguimento* e iniziare a far muovere il robot su una traiettoria rettilinea; sfruttando il *differential drive* è infine possibile far muovere il robot su un piano.

La libreria `arduino2lego.h`

La libreria `arduino2lego.h` garantisce grande facilità di controllo. Anche per questa, però, è possibile immaginare qualche sviluppo futuro, dal sup-

porto di un numero maggiore di periferiche (anche I2C dal momento che Arduino fornisce una libreria per astrarre tale protocollo) ad una vera e propria "blocchettizzazione" delle capacità del robot.

Arduino

Arduino non è risultato completamente occupato durante lo svolgimento dell'esperimento; in particolare sono risultati liberi:

- 4 porte digitali I/O (di cui una PWM), ulteriormente espandibili con uno *shift register* come ad esempio il **74HC595**
- 5 ingressi analogici
- una buona percentuale di cicli CPU
- il 50% della memoria dedicata a contenere il programma

É dunque ancora possibile aggiungere funzionalità al robot.

Miniaturizzazione

In corso d'opera è stata messa in commercio una nuova versione di "Arduino" detta **Arduino Nano**.
Questa presenta le stesse identiche caratteristiche tecniche dell'*Arduino NG*:

- ATmega168 (in un package ridotto)
- porta mini-USB con chip FTDI
- lo stesso numero di porte digitali I/O e due ingressi analogici in più
- connettori ICSP
- stabilizzatore di tensione

Le dimensioni sono di soli 1,85 x 4,32 cm; il prezzo è però doppio rispetto alla versione utilizzata nella realizzazione vista al capitolo 5: "Arduino Nano" costa infatti 49$.

Bibliografia

[1] Fouad Akbar, Mark Haas, Thamer Hummadi, Phillip Lipson, Joe Peel, and James R. Weeks. Inverted pendulum design project. `http://www.obrador.com/EE471Design/Some%20Important%20Numbers.html`.

[2] Atmel. *Atmel 8-bit Microcontroller with 8K Bytes In-System Programmable Flash: ATmega48/V, ATmega88/V, ATmega168/V*, 2007.

[3] M. Banzi, D. Cuartielles, T. Igoe, G. Martino, and D. Mellis. Schemi elettronici e file EAGLE di "Arduino". `http://www.arduino.cc/en/Main/ArduinoBoardDiecimila`.

[4] Massimo Banzi. Sito del progetto "Arduino". `http://www.arduino.cc`.

[5] Paolo Bolzern, Riccardo Scattolini, and Nicola Schiavoni. *Fondamenti di controlli automatici (seconda edizione)*. McGraw-Hill, 2004.

[6] Catharine H. Colwell. A chart of common moments of inertia. `http://dev.physicslab.org/Document.aspx?doctype=3&filename=RotaryMotion_MomentInertiaChart.xml`.

[7] Mohammed Dahleh, Munther A. Dahleh, and George Verghese. *Lectures on Dynamic Systems and Control*, chapter 6. Department of Elec-

trical Engineering and Computer Science Massachuasetts Institute of Technology, 2004.

[8] Epson. *Ultra miniature size Gyro Sensor (for portable GPS applications) XV-8100CB*, 2006.

[9] Philippe E. Hurbain. Lego 9V Technic Motors compared characteristics. `http://www.philohome.com`.

[10] Leonardo Lanari. Analisi delle prestazioni. `http://www.dis.uniroma1.it/~lanari/ContProcessi/MatDidCP/Prestazioni06.pdf`, 2006.

[11] Fabio Previdi. Lucidi del corso di "Controlli Automatici". `http://dinamico2.unibg.it/previdi/index.html`.

[12] Andrew K. Stimac. *Standup and Stabilization of the Inverted Pendulum*, page 13 and 57. Department of Mechanical Engineering of Massachusetts Institute of Technology, 1999.

Appendice A

Codice

A.1 Parti principali di `analisi_modello_robot.m`

Lo script `analisi_modello_robot.m` è servito a supportare lo studio teorico, a confermare alcuni calcoli e ad automatizzare tutte quelle parti che sarebbero state troppo ripetitive da svolgere a mano.

A.1.1 Osservabilità e raggiungibilità

```
%definizione modello
A=[0 1 0 0; 0 0 -m*g/M 0; 0 0 0 1; 0 0 (M+m)*g/(M*l) 0];
B=[0; 1/M ;0 ; -1/(M*l)];
C=[1 0 0 0; 0 0 0 1];
cond_iniz = [0; 0; 0.45; 0];

sys=ss(A,B,C,D);
disp('Autovalori (il sistema e instabile!)');
lambda=eig(A)

%Verifica osservabilita'
obs = obsv(A,C);
no_obs = length(A) - rank(obs);
if(no_obs == 0 )
    disp ('Il sistema e osservabile');
else disp('Il sistema non e osservabile');
end;
```

```
%Verifica controllabilita'
con = ctrb(A,B);
no_con = length(A) - rank(con);
if(no_con == 0)
    disp ('Il sistema e raggiungibile');
else disp('Il sistema non e raggiungibile');
end;
```

A.1.2 Calcolo della matrice K

```
% matrice dei tentativi
poli = [-60 -20 -4 -4.1
           -60 -20 -4+4*i -4-4*i;
           -60 -40 -4 -4.1
           -80 -20 -4 -4.1
           -80 -40 -4 -4.1
        ];
% Autovalori in anello chiuso possono essere fissati
% a piacere perche' il sistema e' completamente raggiungibile

% calcolo vettore dei guadagni
count = 1; %tiene conto del tentativo attuale
dim = size(poli); %vettore: [#righe #colonne]
K=[]; %inizializzazione
for count=1:dim(1)
    Ktemp = place(A,B,poli(count, :));
    K = [K; Ktemp];
end
```

A.1.3 Esperimenti

```
% matrice dei tentativi
poli = [-60 -20 -4 -4.1
        %-60 -20 -5 -5.1;
        %-60 -20 -2 -2.1;
        %-20 -10 -4 -4.1;
        -60 -20 -4+4*i -4-4*i;
        -60 -40 -4 -4.1
        -80 -20 -4 -4.1
        -80 -40 -4 -4.1
        ];
```

```
% Autovalori in anello chiuso possono essere fissati
% a piacere perche' il sistema e' completamente raggiungibile

% calcolo vettore dei guadagni
count = 1; %tiene conto del tentativo attuale
dim = size(poli); %vettore: [#righe #colonne]
K=[]; %inizializzazione
for count=1:dim(1)
    Ktemp = place(A,B,poli(count, :));
    K = [K; Ktemp];
end

%% ESPERIMENTI SUL MODELLO %%
T=0:0.005:10;
r=zeros(size(T));
a = [];

for count=1:dim(1)   %ciclo su tutti i tentativi
    Kcorrente = K(count, :);

    % Ac,Bc,Cc,Dc -> sistema anello chiuso
    Ac=A-B*K(count, :);
    Bc=B;
    Cc = [1 0 0 0 ; 0 0 1 0] - D*K(count,:); %invece di avere la velocita
                                             %ang in uscita metto angolo
    Dc=D;
    sysc=ss(Ac,Bc,Cc,Dc);

    % Calcolo energia e creazione titolo
    action = []
    act = [];
    sim('statespace'); %inizia simulazione simulink
    action = act.^2; %eleva al quadrato vettore azione
    energia = trapz(action); %calcola integrale
    en_str = num2str(energia);
    titolo = mat2str( poli(count, :) );
    titolo = strcat(titolo, ' - energia: ', en_str);
    figure('Name', titolo ,'NumberTitle','off')

    % ESPERIMENTO 001 - Condizioni iniziali non nulle
    subplot(2,2,1);
    X0= cond_iniz';    %[0 0 0.45 0];   0.45 rad == 25 gradi
    [YI,T,XI] = initial(sysc,cond_iniz,T);
    plot(T,YI,T,r), grid
    title('Sistema retroazionato, cond iniz non nulla')
```

```
    legend('Posizione del carrello','Angolo del pendolo','Riferimento',0)
    xlabel('Tempo [sec]')

    % ESPERIMENTO 002 - Risposta all' impulso
    subplot(2,2,3);
    impulse(Ac,Bc,Cc,Dc)

    % ESPERIMENTO 003 -  Risposta allo scalino
    % esperimento poco sensato
    %subplot(2,2,4);
    %step(Ac,Bc,Cc,Dc);

    % FUNZIONE SENSITIVITA' DEL CONTROLLO Q(S)
    [n_loop, d_loop] = ss2tf( Ac,Bc,Cc,Dc );
    % arrotondamenti
    for jj=1:2
        for ii=1:5
            if abs(n_loop(jj,ii))<1e-4
                n_loop(jj,ii)=0;
            end
        end
    end
    for ii=1:5
        if abs(d_loop(ii))<1e-4
            d_loop(ii)=0;
        end
    end
    fdt_loop_pos = tf(n_loop(1,:), d_loop)
    fdt_loop_ang = tf(n_loop(2,:), d_loop)
    q_pos = series( fdt_loop_pos, inv(fdt_pos) );
    q_ang = series( fdt_loop_ang, inv(fdt_ang) );
    subplot(2,2,2);
    bode(q_pos);

    f=f+1;
end
```

A.1.4 Ricostruttore dello stato

```
%% RICOSTRUTTORE dello STATO
% tempo continuo
A1 = A';
B1 = B';
```

```
C1 = C';
D1 = D';
poli_l_cont = 10*poli(1,:); %un ordine più veloce
[L_cont, prec, msg] = place(A1,C1,poli_l_cont);
L_cont = L_cont';

% tempo discreto
Ts = 1e-3; %tempo di campionamento 1 kHz = 1000 1/s
[Ad Bd Cd Dd]=c2dm(A,B,C,D,Ts,'tustin');
sd = c2d(sys,Ts,'tustin');
poli_cont = poli(1, :);
poli_discr = exp( Ts*poli_cont );
poli_l_discr = exp( Ts*poli_l_cont );
Kd = place(Ad,Bd,poli_discr);

A1d = transpose(Ad);
B1d = transpose(Bd);
C1d = transpose(Cd);
D1d = transpose(Dd);
[L_discr, prec, msg] = place(A1d,C1d,poli_l_discr);
L_discr = L_discr';
```

A.2 La S-function del sistema non lineare

Questa S-function è stata scritta per sviluppare un blocchetto personalizzato di Simulink il quale rappresentasse il sistema non lineare rappresentato dal pendolo inverso.

```
function [sys,x0,str,ts] = sist_non_lin_1_lev(t,x,u,flag, Cout, angInit)

switch flag
    case 0  % initialize
        str=[];
        ts = [0 0];
        s = simsizes;
            s.NumContStates = 4;
            s.NumDiscStates = 0;
            dimOut = size(Cout);
            s.NumOutputs = dimOut(1);
            s.NumInputs = 1;
            s.DirFeedthrough = 0;
```

```
            s.NumSampleTimes = 1;
        sys = simsizes(s);
        %angInit = 0.2;
        x0 = [0; 0; angInit; 0];
    case 1  % derivatives
        sys = mdlDerivatives(t,x,u);
    case 3  % output
        sys = Cout*x;
    case {2 4 9}
        % 2:discrete  4:calcTimeHit  9:termination
        sys =[];
    otherwise
        error(['Error - unhandled flag =',num2str(flag)])   ;
end


function sys = mdlDerivatives(t,x,u)

    %variabili (peso tot = 375 g)
    m =  0.170; %0.09; %0.085;
    M =  0.238; %0.285; %0.305;
    l = 0.1;   %scheda grande = 16 cm, scheda gyro = 4 cm, motori = 2 cm
    g = 9.81;

    x1 = x(1);      %posizione
    x2 = x(2);      %velocita
    x3 = x(3);      %angolo
    x4 = x(4);      %vel angolare

    somma = (M + m*(sin(x3))^2);
    F_in = u;

    dx1 = x2;
    dx2 = ( +m*l*((x4)^2)*sin(x3) - m*g*sin(x3)*cos(x3) + F_in ) / somma;
    dx3 = x4;
    dx4 = ( -m*l*((x4)^2)*sin(x3)*cos(x3) + (M+m)*g*sin(x3) - F_in*cos(x3) ) / (l*somma);

    sys = [dx1;dx2;dx3;dx4];
% End of mdlDerivatives.
```

A.3 arduino2lego.h

Questa libreria è stata pubblicata su **Sourceforge** e permette di interagire con vari componenti LEGO (attuatori e sensori) mascherando la complessità sottostante.

```
/*
 *  arduino2lego.h   --   ARDUINO TO LEGO LIBRARY
 *  Library to easily control LEGO NXT motors and NXT sensors
 *
 *  version 1.1
 *
 *  Created by Marco Triverio   --   arduino [dot] poet [at] gmail [dot] com
 *  June - July 2008
 *  Released under GPLv3 licence
 *
 */

#ifndef arduino2lego_h  //prevents problems if someone #include library twice
#define arduino2lego_h

//CONSTANTS
#include "WConstants.h"
#include "a2lsymbols.h"


class NXTMotor
{
  public:
    //NXTMotor(short pinP, short pinM, short interrupt);
    NXTMotor(short pinP = 10, short pinM = 11, short sensePin = SENSEPIN);
    void setSpeed(short speed);
    void setSpeedUntil(short speed, int posiz);
    void stopAndWait(short time);
    void breakMotorFor(short time);
    void sendPos();
    void debugInternals();
    void resetPos();
    int getPos();
    float getSpeed();
    float getMetres();
    void increasePos();
    void decreasePos();
    int encoderEnabled();
```

```
 private:
    short _pinP;
    short _pinM;
    int _pos; //encoder value
    float _speed;
    long _last;
    short _sensePin;
};



class TwoNXTMotors
{
  public:
    TwoNXTMotors(NXTMotor *left, NXTMotor *right);
    void setSpeed(short speed);
    void setSpeedUntil(short speed, int posiz);
    void stopAndWait(short time);
    int getPos();
    float getSpeed();
    void increasePos();
    void decreasePos();
    void sendPos();
    float getMetres();
 private:
    NXTMotor *_right;
    NXTMotor *_left;
    NXTMotor *_withenc;
};


class EpsonGyro
{
  public:
    EpsonGyro(int pinIn = GYROPIN);
    float getAngle();
    float getAngSpeed();
    float updateAngSpeed();
    float calibrate(int debug = 0);
    float getRad();
    float getRadSpeed();
    float getCalibration();
    int getRawRead();
  private:
    int _pinIn;
```

```
    float _theta;
    float _thetadegsec;
    float _thetadegsecold;
    float _zeroPoint;
};


class LED
{
  public:
    LED(short pin=13);
    void on();
    void off();
  private:
    int _pin;
};

#endif
```