CV Self Balancing

Facoltà di Ingegneria
Università di Napoli Federico II
Corso di Laurea Magistrale in Ingegneria dell'Automazione

Sistemi di Controllo Multivariabile

Candidato
Cesare Viscuso M58/7

Docente
Chiar.mo Prof. Ing. **Francesco Amato**

Anno Accademico 2011-2012

Per redigere questa relazione é stato usato il LaTeX nella distribuzione per Windows nota come MiKTeX (versione del software: 2.8) con l'ausilio del software editor WinEdt 7.0, per la parte grafica si é utilizzato GIMP (GNU Image Manipulation Program) 2.8

Contents

Nomenclatura

x - vettore di stato
u - ingresso del sistema
A - matrice dinamica del sistema
B - matrice degli ingressi sullo stato
C - matrice delle uscite
D - matrice di trasmissione diretta
θ - posizione angolare robot
ω - velocità angolare robot
$\dot{\theta}$ - velocità angolare
τ_m - Coppia Motrice (Nm)
τ_e - Coppia applicata (Nm)
Ra - Resistenza di armatura motore DC (Nm)
La - Induttanza armatura (H)
K_m - Costante di coppia ($\frac{Nm}{rad}$)
K_e - Costante di fem ($\frac{Vs}{rad}$)
Va - Tensione di armatura (V)
i - Corrente di armatura (A)
I_R - Inerzia rotore (Kg m^2)
M_w - Massa delle ruote connesse al robot (Kg)
M_p - Massa del robot escluso le ruote (Kg)
I_w - Momento di inerzia delle ruote (kg m^2)
I_p - Momento di inerzia del robot(kg m^2)
H_{fL} - Forze di attrito tra suolo e ruota sinistra (N)
H_{fR} - Forze di attrito tra suolo e ruota destra (N)
θ_w - Angolo di rotazione delle ruote (rad)

θ_p - Angolo di rotazione del pendolo (rad)
l - distanza tra centro della ruota e baricentro robot (m)
C_R - Coppia applicata dal motore destro alla ruota destra (Nm)
C_L - Coppia applicata dal motore sinistro alla ruota sinistra (Nm)
H_L, H_R, P_L, P_R Forze di reazione tra telaio e ruota (N)

Chapter 1

Modello Seegway

Il segway ha un comportamento simile a quello di pendolo capovolto che si poggia su un carrello. La dinamica del pendolo e la ruota sono analizzati separatamente. Poichè il comportamento del robot puó essere influenzato da disturbi come ad esempio coppia fornita dal motore, il modello matematico dovrà tenere conto di tali forze.

1.1 Modello lineare di un motore DC

Il robot è alimentato da due motori a corrente continua della Pololu. In questa sezione viene ricavato il modello nello spazio di stato del motore DC. Questo modello viene poi utilizzato nel modello dinamico robot per fornire una relazione tra la tensione di ingresso ai motori e la coppia di controllo necessaria per equilibrare il robot.
Le equazioni del motore Dc sono le seguenti:

$$\frac{di}{dt} = \frac{R}{L}i + \frac{K_e}{L}\omega + \frac{V_a}{L} \tag{1.1}$$

$$\frac{d\omega}{dt} = \frac{K_m}{I_R}i - \frac{K_e}{I_R}\omega - \frac{\tau_a}{I_R} \tag{1.2}$$

Entrambe le equazioni sono funzioni lineari di velocità e corrente e comprendono le derivate prime. Per il caso del bilanciamento del robot è sufficiente un modello semplificato del motore DC.

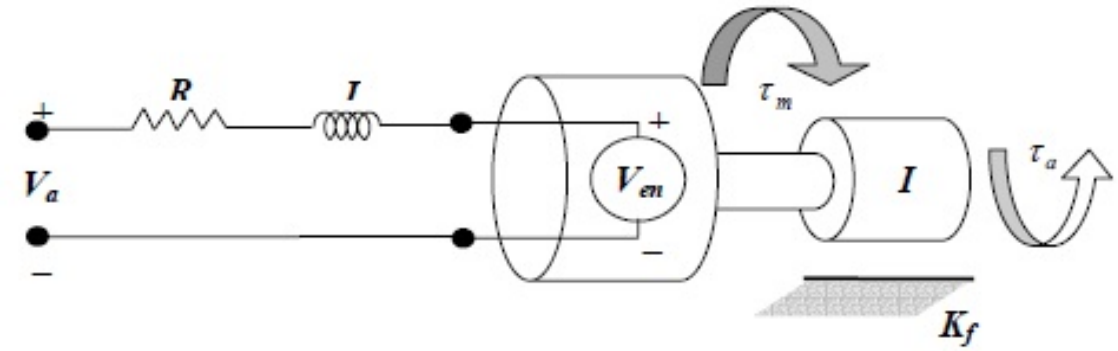

Figure 1.1: Motore DC

Per questo motivo, l'induttanza del motore L e l'attrito del motore $K_e\omega$ sono considerati trascurabili e quindi approssimabili a zero. Quindi la 1.1 e la 1.2 possiamo approssimarle come:

$$i = -\frac{K_e}{R}\omega + \frac{1}{R}V_a \tag{1.3}$$

$$\frac{d\omega}{dt} = \frac{K_m}{I_R}i - \frac{\tau_a}{I_R} \tag{1.4}$$

Sostituendo la 1.3 e la 1.4, ottengo un modelo approssimato del motore DC, che è solo una funzione della velocità motore, della tensione applicata e della coppia di carico.
Si è deciso di utilizzare un modello semplificato anche perchè, dal datasheet molto scarno fornito dal costruttore, si è riusciti a calcolare solo la resistenza di armatura R, e la costante di coppia K_m.

1.2 Modello carrellino

Iniziamo con lo studio delle equazioni delle ruote destra e sinistra[1], anche se, visto che le equazioni sono analoghe, si troverà l'equazione solo per la ruota destra, quella sinistra sarà automaticamente nota. Applicando la legge di Newton lungo l'asse x $\sum F_x = Ma$, possiamo dire che:

$$M_w\ddot{x} = H_{fR} - H_R \tag{1.5}$$

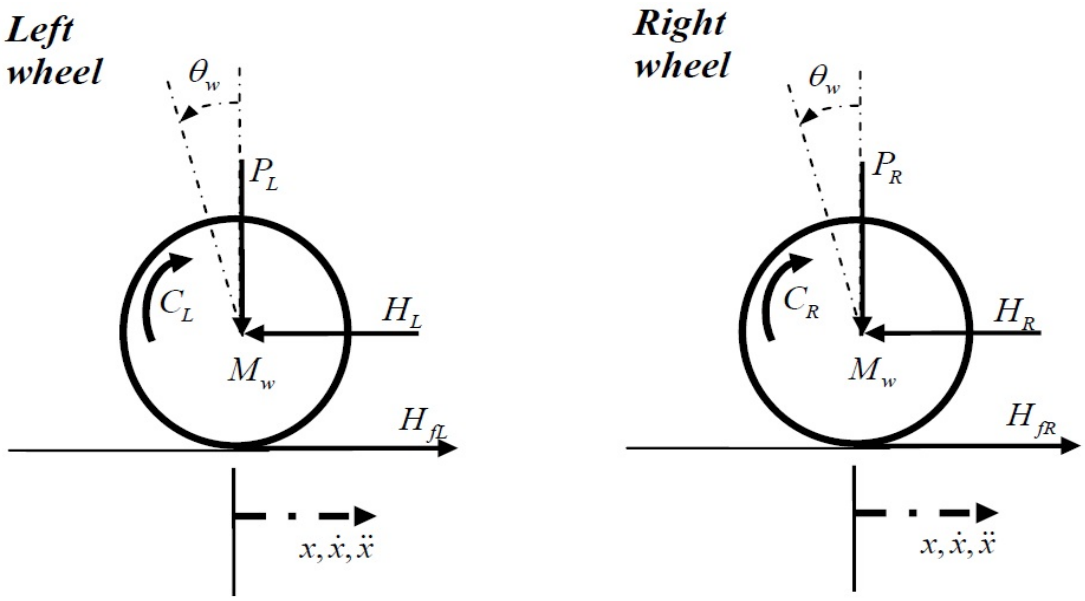

Figure 1.2: Gradi di liberta ruote

Mentre applicando la legge di Newton intorno al centro della ruota $\sum M_o = I\theta$ possiamo dire che

$$I_w\ddot{\theta}_w = C_R - H_{fR} * r \tag{1.6}$$

Dalle equazioni del motore DC, possiamo dire che la coppia prodotta dallo stesso è:

$$\tau_m = I_R\frac{d\omega}{dt} + \tau_a \tag{1.7}$$

Riaggiustando l'equazione, possiamo dire che la coppia che il motore fornisce alle ruote è:

$$C_R = K_m i_a = I_R\frac{d\omega}{dt} = -\frac{K_m K_e}{R} + \dot{\theta}_w + \frac{K_m}{R}V_a \tag{1.8}$$

Sostituendola nella 1.6 avremo che:

$$I_w\ddot{\theta}_w = -\frac{K_m K_e}{R}\dot{\theta}_w + \frac{K_m}{R}V_a - H_{fR} * r \tag{1.9}$$

da cui

$$H_{fR} = -\frac{I_w}{r}\ddot{\theta}_w - \frac{K_m K_e}{Rr}\dot{\theta}_w + \frac{K_m}{Rr}V_a \tag{1.10}$$

infine, sostituendola nella 1.5 ottengo l'equazione del moto per la ruota destra, di conseguenza per dualità sarà nota anche quella sinistra.

$$M_w\ddot{x} = -\frac{I_w}{r}\ddot{\theta}_w - \frac{K_m K_e}{Rr}\dot{\theta}_w + \frac{K_m}{Rr}V_a - H_R \quad \text{ruota destra} \tag{1.11}$$

$$M_w\ddot{x} = -\frac{I_w}{r}\ddot{\theta}_w - \frac{K_m K_e}{Rr}\dot{\theta}_w + \frac{K_m}{Rr}V_a - H_L \quad \text{ruota sinistra} \tag{1.12}$$

A questo punto poichè è nota la relazione tra movimento angolare e movimento lineare $\theta r = x$, possiamo banalmente affermare che:

$$\ddot{\theta}_w r = \ddot{x} \Rightarrow \ddot{\theta}_w = \frac{\ddot{x}}{r} \tag{1.13}$$

$$\dot{\theta}_w r = \dot{x} \Rightarrow \dot{\theta}_w = \frac{\dot{x}}{r} \tag{1.14}$$

quindi riscrivendo la 1.12 avremo:

$$M_w\ddot{x} = -\frac{I_w}{r^2}\ddot{x} - \frac{K_m K_e}{Rr^2}\dot{x} + \frac{K_m}{Rr}V_a - H_R \quad \text{ruota destra} \tag{1.15}$$

$$M_w\ddot{x} = -\frac{I_w}{r^2}\ddot{x} - \frac{K_m K_e}{Rr^2}\dot{x} + \frac{K_m}{Rr}V_a - H_L \quad \text{ruota sinistra} \tag{1.16}$$

Sommando la 1.15 e la 1.16 avremo:

$$2\left(M_w + \frac{I_w}{r^2}\right)\ddot{x} = -2\frac{K_m K_e}{Rr^2}\dot{x} + 2\frac{K_m}{Rr}V_a - (H_L + H_R) \tag{1.17}$$

1.3 Modello pendolo capovolto

Ancora una volta, utilizzando la legge di Newton del moto, la somma delle forze in direzione orizzontale $\sum F_x = M_p\ddot{x}$ sarà :

$$(H_L + H_R) - M_p l\ddot{\theta}_p \cos\theta_p + M_p l\dot{\theta}_p^2 \sin\theta_p = M_p\ddot{x} \tag{1.18}$$

ovvero:

$$(H_L + H_R) = M_p\ddot{x} + M_p l\ddot{\theta}_p \cos\theta_p - M_p l\dot{\theta}_p^2 \sin\theta_p \tag{1.19}$$

La somma delle forze nella direzione perpendicolare al pendolo

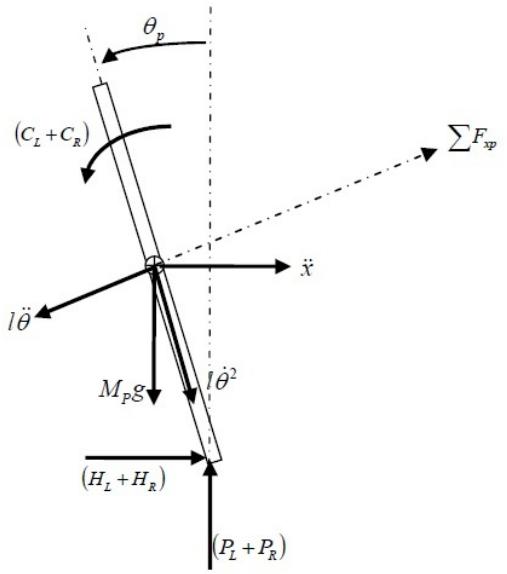

Figure 1.3: Gradi di liberta pendolo

$\sum F_{xp} = M_p \ddot{x} \cos\theta_p$ sarà :

$$-(H_L + H_R)\cos\theta_p + (P_L + P_R)\sin\theta_p - M_p g \sin\theta_p - M_p l\ddot{\theta} = M_p \ddot{x}\cos\theta_p \tag{1.20}$$

La somma dei momenti intorno al centro di massa del pendolo
$\sum M_o = I\alpha$ sarà :

$$-(H_L + H_R)\, l\cos\theta_p - (P_L + P_R)\, l\sin\theta_p - (C_L + C_R) = I_p\ddot{\theta}_p \tag{1.21}$$

La coppia totale applicata dai due motori al pendolo, come descritto per la ruota destra nella 1.8, dopo aver tenuto conto delle trasformazioni lineari, sarà data da:

$$C_R + C_L = -2\frac{K_m K_e}{Rr}\dot{x} + 2\frac{K_m}{R}V_a \tag{1.22}$$

sostituendola nella 1.21 avremo:

$$-(H_L + H_R)\, l\cos\theta_p - (P_L + P_R)\, l\sin\theta_p = I_p\ddot{\theta}_p - 2\frac{K_m K_e}{Rr}\dot{x} + 2\frac{K_m}{R}V_a \tag{1.23}$$

A questo punto, se moltiplico la 1.20 per $-l$

$$\lfloor -(H_L + H_R)\, l\cos\theta_p - (P_L + P_R)\, l\sin\theta_p \rfloor + M_p lg\sin\theta_p + M_p l^2\ddot{\theta} = -M_p l\ddot{x}\cos\theta_p \tag{1.24}$$

A questo punto, inserendo la 1.23 tra le $\lfloor \quad \rfloor$ della 1.24 avremo:

$$I_p\ddot{\theta}_p - 2\frac{K_m K_e}{Rr}\dot{x} + 2\frac{K_m}{R}V_a + M_p lg\sin\theta_p + M_p l^2\ddot{\theta} = -M_p l\cos\theta_p\ddot{x} \tag{1.25}$$

Per eliminare $(H_L + H_R)$ dalla dinamica dei motori, sostituiamo la 1.19 nella 1.17, avremo:

$$2\left(M_w + \frac{I_w}{r^2}\right)\ddot{x} = -2\frac{K_m K_e}{Rr^2}\dot{x} + 2\frac{K_m}{Rr}V_a - M_p\ddot{x} - M_p l\ddot{\theta}_p\cos\theta_p + M_p l\dot{\theta}_p^2 sin\theta_p \tag{1.26}$$

Riordinando le equazioni 1.25 e 1.26, avremo le due equazioni non lineari del moto del sistema:

$$\begin{cases} \left(I_p + M_p l^2\right)\ddot{\theta}_p - 2\frac{K_m K_e}{Rr}\dot{x} + 2\frac{K_m}{R}V_a + M_p lgsin\theta_p = -M_p l\cos\theta_p\ddot{x} \\ 2\frac{K_m}{Rr}V_a = \left(2M_w + \frac{2I_w}{r^2} + M_p\right)\ddot{x} + 2\frac{K_m K_e}{Rr^2}\dot{x} + M_p l\ddot{\theta}_p\cos\theta_p - M_p l\dot{\theta}_p^2 sin\theta_p \end{cases} \tag{1.27}$$

1.3.1 Linearizzazione equazioni non lineari

Le equazioni finali ottenute (1.27) sono non lineari, ma possono essere linearizzate se assumiamo che $\theta_p = \pi + \phi$, dove ϕ rappresenta un piccolo scostamento rispetto la direzione verticale verso l'alto.
Questa considerazione ci permette di affermare che:

$$\cos\theta_p = -1 \tag{1.28}$$

$$\sin\theta_p = -\phi \tag{1.29}$$

$$\left(\frac{d\theta_p}{dt}\right)^2 = 0 \tag{1.30}$$

Il modello linarizzato sarà :

$$\left(I_p + M_p l^2\right)\ddot{\phi} - \frac{2K_m K_e}{Rr}\dot{x} + \frac{2K_m}{R}V_a - M_p g l \phi = M_p l \ddot{x} \tag{1.31}$$

$$\frac{2K_m}{Rr}V_a = \left(2M_w + 2\frac{I_w}{r^2} + M_p\right)\ddot{x} + 2\frac{K_m K_e}{Rr^2}\dot{x} - M_p l \ddot{\phi} \tag{1.32}$$

1.4 Modello nello spazio di stato

Per ottenere la rappresentazione nello spazio di stato, le equazioni 1.32 e 1.31 sono riorganizzate come segue:

$$\ddot{\phi} = \frac{M_p l}{\left(I_p + M_p l^2\right)}\ddot{x} + \frac{2K_m K_e}{Rr\left(I_p + M_p l^2\right)}\dot{x} - \frac{2K_m}{R\left(I_p + M_p l^2\right)}V_a + \frac{M_p g l}{\left(I_p + M_p l^2\right)}\phi \tag{1.33}$$

$$\ddot{x} = \frac{2K_m}{Rr\left(2M_w + \frac{2I_w}{r^2} + M_p\right)}V_a - \frac{2K_m K_e}{Rr^2\left(2M_w + \frac{2I_w}{r^2} + M_p\right)}\dot{x} + \frac{M_p l}{\left(2M_w + \frac{2I_w}{r^2} + M_p\right)}\ddot{\phi} \tag{1.34}$$

Sostituendo la 1.33 nella 1.31 e la 1.34 nella 1.32 e dopo una serie di manipolazioni algebriche si ottiene l'equazione del sistema complessivo nello spazio di stato:

$$\begin{bmatrix}\dot{x}\\ \ddot{x}\\ \dot{\phi}\\ \ddot{\phi}\end{bmatrix} = \begin{bmatrix}0 & 1 & 0 & 0\\ 0 & \frac{2K_m K_e(M_p l r - I_p - M_p l^2)}{Rr^2\alpha} & \frac{M_p^2 g l^2}{\alpha} & 0\\ 0 & 0 & 0 & 1\\ 0 & \frac{2K_m K_e(r\beta - M_p l)}{Rr^2\alpha} & \frac{M_p g l \beta}{\alpha} & 0\end{bmatrix}\begin{bmatrix}x\\ \dot{x}\\ \phi\\ \dot{\phi}\end{bmatrix} + \begin{bmatrix}0\\ \frac{2K_m(I_p + M_p l^2 - M_p l r)}{Rr\alpha}\\ 0\\ \frac{2K_m(M_p l - r\beta)}{Rr\alpha}\end{bmatrix}V_a \tag{1.35}$$

Dove:
$\beta = \left(2M_w + \frac{2I_w}{r^2} + M_p\right)$
$\alpha = \left[I_p\beta + 2M_p l^2\left(M_w + \frac{I_w}{r^2}\right)\right]$
Per una maggiore chiarezza, detto $x \in \Re^{4x1}$ il vettore dello spazio di stato, si

avrà che $x_1 = x, \quad x_2 = \dot{x}, \quad x_3 = \phi, \quad x_4 = \dot{\phi}$, avremo:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{2K_m K_e (M_p lr - I_p - M_p l^2)}{Rr^2\alpha} & \frac{M_p^2 g l^2}{\alpha} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{2K_m K_e (r\beta - M_p l)}{Rr^2\alpha} & \frac{M_p g l \beta}{\alpha} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{2K_m (I_p + M_p l^2 - M_p lr)}{Rr\alpha} \\ 0 \\ \frac{2K_m (M_p l - r\beta)}{Rr\alpha} \end{bmatrix} V_a \tag{1.36}$$

Nel modello appena descritto stiamo ipotizzando che il veicolo rimanga sempre a contatto con il terreno e che non c'è slittamente alle ruote. Si vedrà nei capitoli seguenti, che la scelta delle ruote ad elevato coefficiente di attrito (essendo costituite da una mescola molto molto morbida) rende l'approssimazione un problema non insormontabile.

1.5 Proprietà strutturali del sistema

Valutiamo le proprietà strutturali del sistema in questione:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -0.8126 & 0.43 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 5.539 & 18.26 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0.362 \\ 0 \\ -2.467 \end{bmatrix} V_a$$

Figure 1.4: Modello con valori numerici

1.5.1 Stabilità

Per valutare la stabilità, possiamo considerare il segno della parte reale degli autovalori della matrice dinamica A, da Matlab, utilizzando il comando $eig(A)$, avremo:
Possiamo notare come il sistema sia giustamente instabile, infatti c'è la presenza di un autovalore a parte reale maggiore di zero.

```
>> eig(A)

ans =

            0
   4.3266e+00
  -9.5005e-01
  -4.1892e+00
```

Figure 1.5: Autovalori di A

1.5.2 Controllabilità

Per valutare la controllabilità del sistema, possiamo utilizzare il seguente codice:

```
%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;
```

Figure 1.6: Codice controllabilità

ottenendo in risposta che il sistema è controllabile.

1.5.3 Osservabilità

Per valutare l'osservabilità del sistema, possiamo utilizzare il seguente codice:

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

Figure 1.7: Codice osservabilità

ottenendo in risposta che il sistema è osservabile.

Chapter 2

Strategie di controllo

Un robot self balancing è un ottimo banco di prova per la teoria del controllo, perché presenta dinamiche di un sistema non-lineare e instabile.
L'obiettivo è quello di mostrare come il sistema può essere controllato mediante controllori lineari nello spazio di stato. I controllori sono progettati utilizzando il modello sviluppato nel capitolo precedente. Possiamo notare da un semplice grafico, come il sistema sia naturalmente instabile, ecco una simulazione a ciclo aperto con un ingresso impulsivo di tensione 12 V:

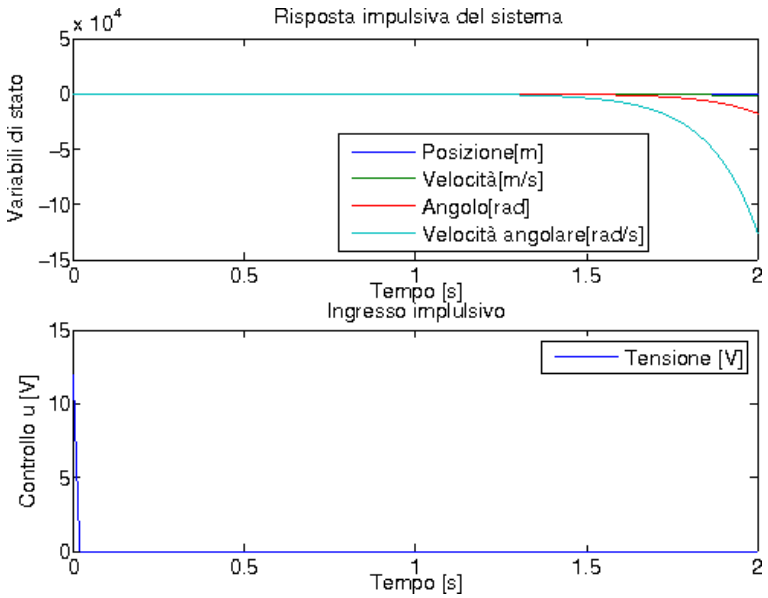

Figure 2.1: Risposta impulsiva a ciclo aperto

Le tecniche utilizzate sono quelle studiate al corso, ovvero Assegnamento dei poli ed LQR.

Assegnamento dei poli Il problema del controllo consiste nel progettare un controllore C che, sulla base delle informazioni fornite da un segnale di riferimento r(t) e uno di uscita y(t), sia in grado di generare un segnale di controllo u(t), che faccia evolvere il sistema S in modo da minimizzare lo scostamento tra y e r. Lo schema classico di un sistema di controllo in retroazione è il seguente:

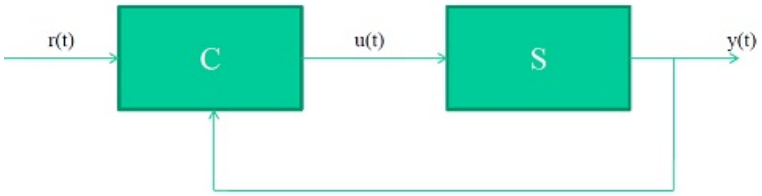

Figure 2.2: Schema di sistema di controllo in retroazione di uscita

Da quanto detto, è ovvio che l'informazione basata sull'uscita y(t) è incompleta, in quanto basata solo su una combinazione lineare delle variabili di stato. Un'informazione completa (full information), dovrebbe essere invece basata sulla conoscenza di tutte le variabili di stato. Per questo motivo, una generalizzazione della tecnica della controreazione può essere ottenuta utilizzando uno schema del genere:

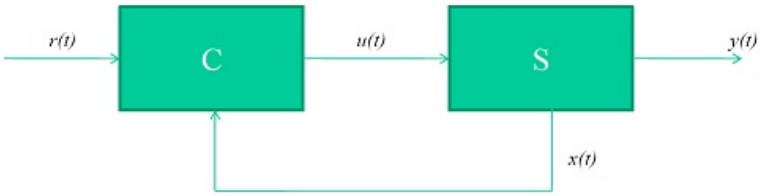

Figure 2.3: Schema di sistema di controllo in retroazione di stato

Questo schema ha trovato una larga utilizzazione nella risoluzione di problemi di controllo (stabilizzazione, allocazione dei poli, controllo ottimo, etc.). Ovviamente lo schema presentato è implementabile se tutto lo stato è misurabile. Negli approcci summenzionati, oggetto di analisi durante il corso

di studi, la struttura del controllore è quasi sempre molto semplice riducendosi ad una combinazione lineare delle variabili di stato. Una prima tecnica di controllo che verrà analizzata è quella che si preoccupa semplicemente di progettare un controllore che stabilizzi il sistema a ciclo chiuso.

LQR Con LQR si intende Regolatore Lineare Quadratico ed è una tecnica di controllo ottimo. Le tecniche di controllo ottimo sono basate sul progetto della legge di controllo in modo da rendere minimo (o massimo) un opportuno funzionale detto indice di costo (o di prestazione).
L'indice di costo è di solito definito in termini di norme di segnali, e tiene tipicamente conto della necessità di soddisfare obiettivi contrastanti tra loro, ciascuno dei quali viene pesato in maniera opportuna nel funzionale da ottimizzare.
Il progetto del controllo secondo queste tecniche consiste quindi in un problema di ottimizzazione in cui i vincoli sono costituiti dalla dinamica dell'impianto da controllare. L'incognita è rappresentata dalla sequenza di ingressi da applicare in un dato intervallo di tempo, eventualmente infinito.

2.1 Applicazione Tecniche di controllo

Applichiamo in questa sezione le tecniche studiate:

1. Pole Placement

2. LQR

2.1.1 Pole Placement

Nel lavoro, è stato calcolato il vettore **k**, mediante una LMI, imponendo l'appartenenza degli autovalori a sinistra dell'ascissa -4, ovvero imponendo che:

$$AP + BL + PA^T + L^T B^T + 2\alpha P < 0 \tag{2.1}$$

con $\alpha = 4$

2.1.2 LQR

Le matrici Q ed R sono dei parametri di progetto da scegliere in maniera oculata. In particolare si segue la seguente strada:

1. Per ottenere piccole azioni di controllo, si sceglie un valore elevato di R
2. Per quanto concerne Q, si fissa una struttura diagonale e si impostano gli elementi sulla diagonale sufficientemente grandi se si vuol mantenere la corrispondente variabile a piccoli valori (e viceversa)

Sulla base di queste regole si sono scelti i seguenti valori di R e Q:
$R = 1000$

$$Q = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 100 \end{pmatrix} \tag{2.2}$$

Il controllore è banalmente calcolato con Matlab tramite il comando **K=lqr(sys,Q,R)**

2.2 Filtro di Kalman

Lo scopo del Filtro di Kalman [2] è quello di fornire la stima ottimale in caso di rumore gaussiano a media nulla, in modo da stimare lo stato di un processo e minimizzare la media quadratica degli errori. Il sistema lineare a tempo discreto in esame è costituito dallo stato $x(k)$ e dall'uscita $y(k)$ che soddisfano le seguenti equazioni:

$$x_{k+1} = Ax_k + Bu_k + w_k \tag{2.3}$$

$$y_k = Cx_k + Du_k + v_k \tag{2.4}$$

dove $x \in \Re^n$, $y \in \Re^q$ e $u \in \Re^p$ mentre w_k e v_k sono vettori di rumori gaussiani a media nulla che entrano nella dinamica e nelle misure. I rumori sono indipendenti fra loro e non dipendono neanche dallo stato iniziale x_0, anche

esso assunto come una variabile gaussiana.
Dato lo stato iniziale x_0 affetto da rumore, è possibile definire il valore atteso m_0, con varianza P_0.
Per i disturbi nello stato e nella misura, la loro media è nulla mentre presentano rispettivamente matrice di covarianza Q_k semidefinita positiva, e R_k definita positiva (in quanto l'uscita è sempre affetta da disturbi).
Il Filtro di Kalman è costituito da una fase predittiva in cui viene calcolato lo stato a priori $\hat{x}^-$, e una fase di correzione in cui viene calcolato lo stato a posteriori $\hat{x}$ a seguito della misurazione $y(k)$.
Ora si possono definire l'errore di stima a priori e l'errore di stima a posteriori:

$$e_k^- = x_k - \hat{x}^- \tag{2.5}$$

$$e_k = x_k - \hat{x} \tag{2.6}$$

Si definiscono quindi la covarianza dell'errore apriori (valore atteso dell'errore apriori) e la covarianza dell'errore a posteriori (valore atteso dell'errore a posteriori) come:

$$P_k^- = E[e_k^{-T} e_k^-] \tag{2.7}$$

$$P_k = E[e_k^T e_k] \tag{2.8}$$

Lo scopo del Filtro di Kalman è quello di rendere minimo il valore atteso a posteriori $P(k)$ con un'opportuna scelta del guadagno K_k.
La stima a posteriori $\hat{x}$ dello stato si ricava dalla somma della stima a priori $\hat{x}^-$ dello stato, con una differenza fra l'uscita misurata $y(k)$ e la sua previsione data da $C\hat{x}^-$. La stima a posteriori quindi sarà:

$$\hat{x} = \hat{x}^- + K_k(y_k - C\hat{x}^-) \tag{2.9}$$

La matrice $K_k \in \Re^{nxm}$ è chiamata guadagno di Kalman, e si determina in modo da rendere minima la covarianza dell'errore a posteriori dato dall'equazione

2.8; il suo valore è pari a

$$K_k = P_k^- C_k^T (C P_k^- C^T + R_k)^{-1} \tag{2.10}$$

Quindi l'algoritmo di Kalman può essere scomposto in due parti una fase di predizione e una fase di correzione. I passaggi della fase di predizione sono:

$$\hat{x}_k^- = A\hat{x}_{k-1} \tag{2.11}$$

$$P_k^- = AP_{k-1}A^T + Q_{k-1} \tag{2.12}$$

Nella fase di correzione i passaggi sono:

$$K_k = P_k^- C^T (CP_k - C^T + R_k)^{-1} \tag{2.13}$$

$$\hat{x}_k = \hat{x}_k^- C^T (CP_k - C^T + R_k)^{-1} \tag{2.14}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(y_k - C\hat{x}_k^-) \tag{2.15}$$

$$P_k = (I - K_k C)P_k^- \tag{2.16}$$

Quindi, una volta noto l'istante iniziale $\hat{x}_0$, è possibile eseguire i passaggi algoritmici descritti dal filtro di kalman.

Chapter 3

Simulazioni

Lo schema su cui si sono basate le simulazioni è il seguente:

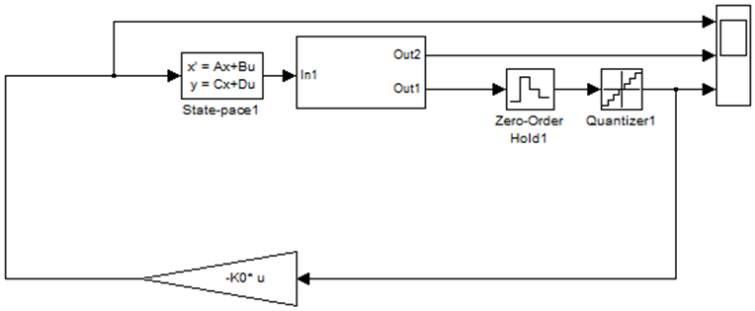

Figure 3.1: Schema di sistema di controllo in retroazione di stato

Il sottosistema a valle del blocco $state-space$ è un sistema per fornire un disturbo impulsivo all'angolo θ, per capire come risponde il sistema di controllo, eccone i particolari:

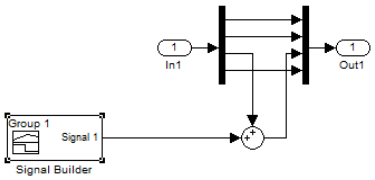

Figure 3.2: Disturbo sulla θ

Ecco un esempio di simulazione con controllore LQR, in prima istanza senza alcun tipo di disturbo:

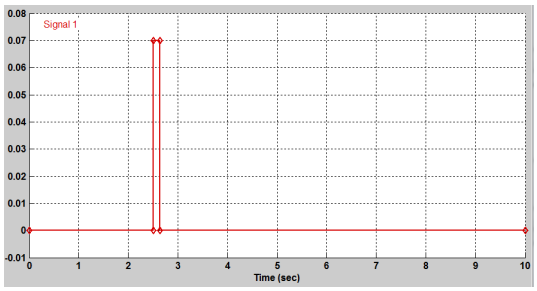

Figure 3.3: Segnale di disturbo

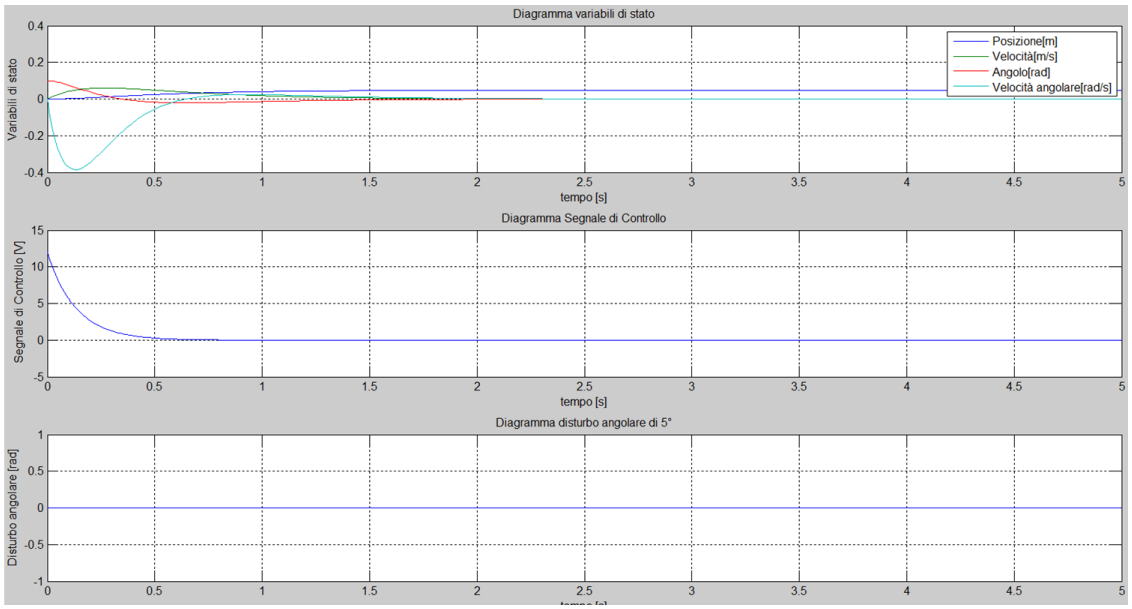

Figure 3.4: Simulazione LQR senza disturbo

Notiamo come in circa un secondo l'angolo θ va correttamente a zero. Proviamo adesso una simulazione con il disturbo di Fig.3.3, in cui abbiamo un disturbo angolare di 5°, all'istante 2.5s, per capire effettivamente come reagisce il controllore al disturbo. Notiamo una ottima risposta al disturbo infatti il

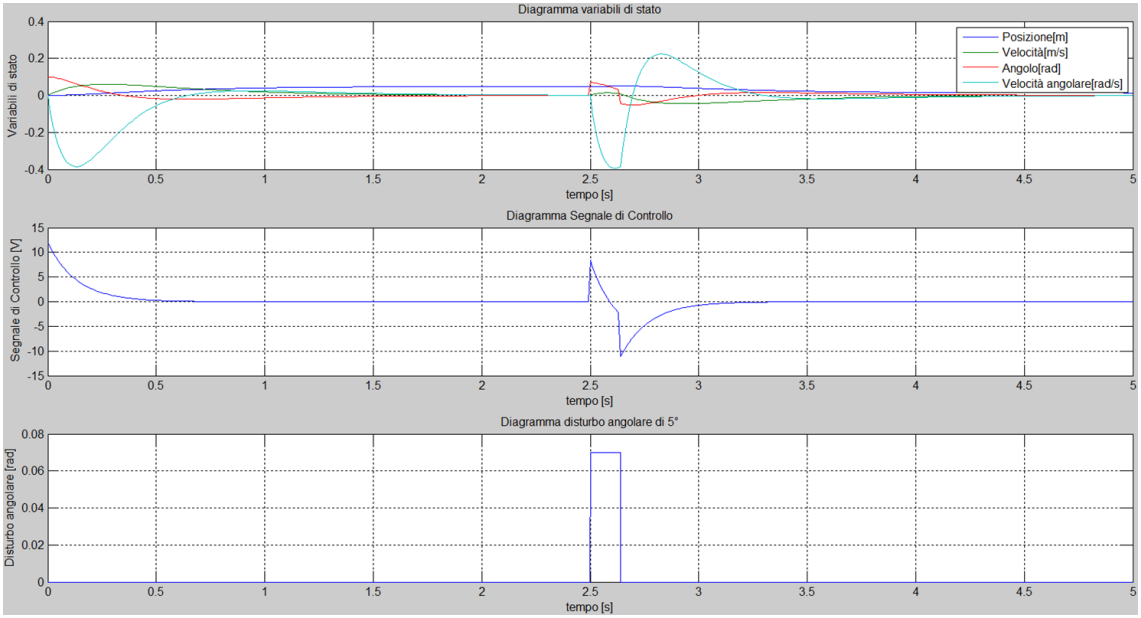

Figure 3.5: Simulazione LQR senza disturbo

controllore riposiziona in posizione verticale l'asta in appena 1 secondo.
Notiamo anche che le condizioni iniziali di simulazione sono date dal vettore $cond\_iniz = [0, 0, 0.1, 0]^T$, ed anche in questo caso in un tempo vicino al secondo l'azione del controllo si esaurisce ed il sistema viene equilibrato.
Non vi sono problemi sulla variabile di controllo in quanto essa non va mai in saturazione e si mantiene nei limiti fisici del problema in esame, infatti non supera mai il valore dei 12 Volt.

In queste situzioni è stato utilizzato il vettore $Kd1 = [-0.1, -67.7, -121.24, -17.12]$. Con il controllore Pole Placement i risultati sono molto simili, quindi si è scelta una simulazione più significativa per mostrare come utilizzando una LMI in grado di mappare i poli a sinistra di -4, il sistema risponde più velocemente. Come si può notare la risposta risulta complessivamente più veloce.

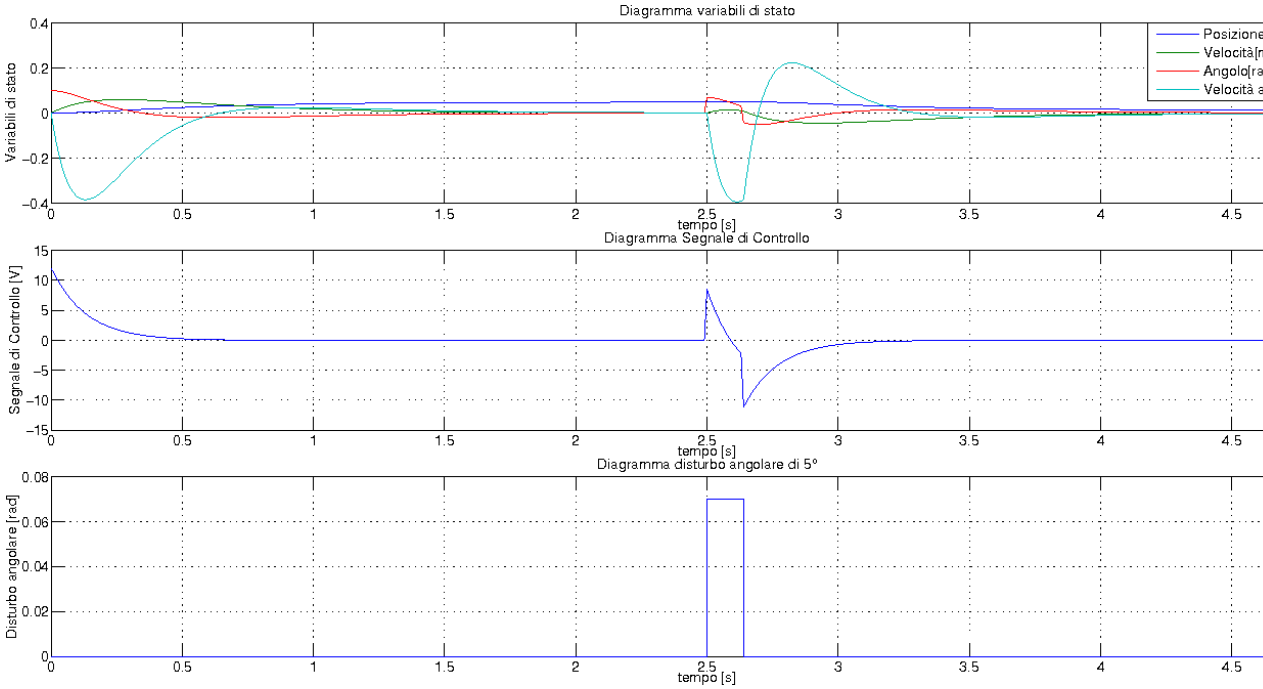

Figure 3.6: Simulazione Pole Placement con autovalori a parte reale minore di -4

Chapter 4

Realizzazzione fisica del prototipo

Al fine di avere un maggior feedback su quanto studiato si è creato un modello reale su cui provare quanto testato in fase di simulazione.
I componenti utilizzati sono i seguenti:

1. 1 Arduino Uno Rev3
2. 2 Ruote Banebots
3. 2 Hub Banebots
4. 2 Motori DC Pololu 19:1 con encoder a 64 CPR
5. 1 IMU: Inertial Unit Measurement
6. 1 Dual driver Pololu VNH2SP30
7. 1 Batteria Lipo da 7V
8. 1 Batteria Lipo da 12 V
9. 1 Breadboard e componentistiche varie

4.1 Arduino Uno Rev3

L'Arduino Uno é una scheda basata sul microcontrollore Atmel ATmega328. Ha 14 Ingressi/uscite digitali (6 delle quali con PWM), 6 ingressi analogici,

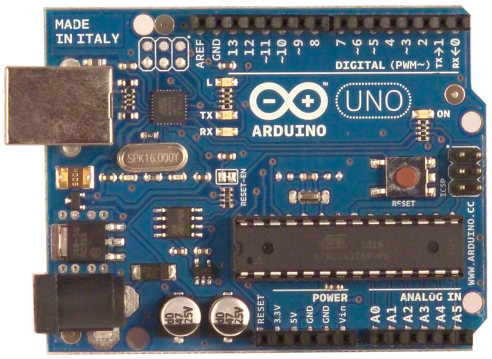

Figure 4.1: Arduino uno Rev3

un cristallo a 16 MHz, una porta USB, una presa di alimentazione, una connessione ICSP, e un bottone di reset.
Dispone di tutto il necessario per gestire il microcontrollore a bordo; per utilizzarla basta connettere la scheda al computer con un cavo USB o alimentarla con un alimentatore (max 12V) o delle batterie esterne La UNO differisce dalle schede precedenti perché non usa l'FTDI USB-to-serial chip. Le caratteristiche introdotte nella versione R3 sono:

1. ATmega16U2 invece del 8U2 come convertitore USB-seriale.

2. piedinatura 1.0: sono stati aggiunti i pin SDA e SCL per la comunicazione TWI vicino al pin AREF e vicino al pin RESET altri sono stati collocati altri due nuovi pin, il pin IOREF che consentirà agli shield di adattarsi alla tensione fornita dalla scheda, ed un pin non collegato, riservato per scopi futuri.

3. Circuito di RESET piú forte.

La Arduino Uno è la più recente scheda Arduino con l'USB a bordo, per fare un confronto con le schede precedenti fate riferimento alla pagina index of Arduino boards.

Microcontrollore	ATmega328
Tensione di funzionamento	5V
Tensione di Alimentazione (raccomandata)	7-12V
Massima Tensione supportata (non raccomandata)	20V
I/O digitali	14 (6 dei quali con uscita PWM)
ingressi analogici	6
Corrente in uscita per I/O Pin	40 mA
Corrente in uscita per 3.3V Pin	50 mA
Memoria Flash	32 KB (ATmega328) di cui 0.5 KB usata bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Velocità di clock	16 MHz

Figure 4.2: Specifiche Arduino uno Rev3

4.2 Ruote Banebots

Le ruote BaneBots sono realizzate in polipropilene e sono rivestite in gomma termoplastica, questa struttura garantisce un peso molto basso e un'ottima resistenza, uniti ad una elevata durata ed una trazione eccezionale. Nel caso in esame le ruote hanno un raggio di 98mm.

Figure 4.3: Ruote Banebots

4.3 Hub Banebots

Sono realizzati in alluminio con una finitura nera.
Vengono forniti con gli accessori riportati nella foto.

Per fissare il grano occorre una chiave esagonale misura inglese da 1/16".
Nel caso in esame l'hub ha un asse da 6mm.

Figure 4.4: Hub Banebots

4.4 Motori DC Pololu 19:1 con encoder a 64 CPR

Questo motoriduttore è un motore DC che opera a 12V, è potente e di alta qualità, dotato di una riduzione 19:1 in metallo.
Questo motoriduttore è integrato con un encoder in quadratura che fornisce una risoluzione di 64 impulsi per ogni rivoluzione dell'asse del motore. Naturalmente, tenendo in considerazione la riduzione del motore stesso, i 64

Figure 4.5: Encoder

CPR corrispondono a 1216 impulsi per ogni rivoluzione dell'asse esterno del motoriduttore. L'asse di questo motoriduttore è lungo 15.5mm, ha un diametro di 6mm ed è caratterizzato dalla tipica forma a D. Questo motoridut-

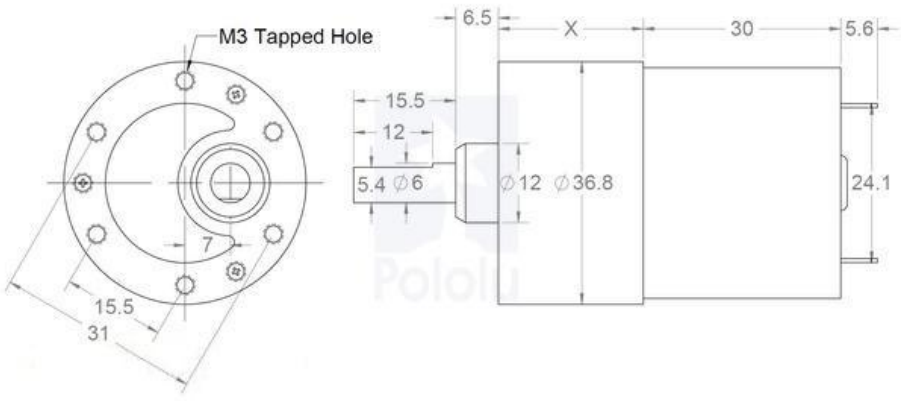

Figure 4.6: Dimensioni del motore

tore è stato progettato per funzionare con una tensione di alimentazione pari a 12V, ma al contempo esso può cominciare a ruotare anche con tensioni inferiori. L'encoder utilizzato è un encoder ad effetto Hall a due canali. Il sensore ad effetto Hall viene utilizzato per rilevare la rotazione di un disco magnetico posizionato sul retro dell'asse del motore. L'encoder in quadratura

Specifiche

- Dimensioni: 37D x 64L mm
- Peso: 213 g (7.5 oz)
- Diametro dell'asse: 6 mm
- Alimentazione: 12 V
- Rapporto di riduzione: 19:1
- Velocità a vuoto: 500rpm
- Corrente a vuoto: 300 mA
- Corrente in stallo: 5000 mA
- Coppia in stallo: 5 Kg-cm

Figure 4.7: Specifiche motore DC

fornisce una risoluzione di 64 impulsi per rivoluzione e quindi, per effetto della riduzione del motore stesso, una risoluzione di 19x64=1216 impulsi per ogni rivoluzione dell'asse esterno del motoriduttore. Il motore si interfaccia con l'esterno tramite l'utilizzo di sei fili colorati della lunghezza di 28 cm circa, ognuno dei quali corrispondente a segnali diversi come specificato nel seguente

elenco:

1. Nero: alimentazione del motore
2. Rosso: alimentazione del motore
3. Blu: Vcc del sensore Hall (3.5 - 20 V)
4. Verde: GND del sensore Hall
5. Giallo: uscita del sensore A
6. Bianco: uscita del sensore B

4.5 IMU: Inertial Measurement Unit

Figure 4.8: IMU ITG3200/ADXL345

Questa Imu, prodotta dalla Sparkfun, presenta integrati l'accelerometro ADXL345 e un giroscopio ITG-3200. Con questa scheda si hanno 6 gradi di libertà.
Questi sensori comunicano con il protocollo I2C. E' stato necessario l'utilizzo del filtro di kalman, per evitare nella lettura dell'angolo l'errore di drift prodotto dal giroscopio.

4.6 Dual driver Pololu VNH2SP30

Il Dual driver Pololu VNH2SP30 permette di pilotare due motori DC in PWM, con una corrente massima di 14A, 30A di picco. La frequenza di

switching è di 20 Khz.
Inoltre, ha anche due uscite di *currente sense*, che forniscono un valore proporzionale alla corrente che sta erogando il driver.

Dimensioni

Dimensioni:	1,86 "x 1,86"

Caratteristiche generali

Motore driver:	VNH2SP30
Motore canali:	2
Tensione minima di funzionamento:	5,5 V
Tensione massima:	16 V
Uscita in corrente continua per canale:	14 A
Corrente di picco in uscita per canale:	30 A
Corrente di uscita continua in parallelo:	25 A
Senso corrente:	0,13 V / A
Massima frequenza PWM:	20 kHz
Protezione contro l'inversione di tensione?	Y 1

Figure 4.9: Dimensione e specifiche del Dual Driver Pololu VNH2SP30

4.7 Batterie Lipo

Le batterie LiPo hanno sostituito le NiMh in ambito modellistico, poichè sono migliori di quest'ultime in termini di peso/potenza essendo più leggere e più performanti.
La caratteristica fondamentale per cui si è passati dall'utilizzo delle NiMh alle Lipo è che non soffrono di autoscarica come le NiMh.
D'altro canto, le LiPo sono più "sensibili" e quindi necessitano di essere utilizzate in maniera corretta, ad esempio è obbligatorio l'utilizzo di un bilanciatore quando vengono caricate.
Il bilanciatore sostanzialmente controlla le singole celle e garantisce una car-

ica più sicura e migliore.
In sostanza, esso si occupa di tenere le celle tutte alla stessa tensione durante la fase del caricamento.

4.7.1 Caratteristica batteria da 7 Volt

Caratteristiche Tecniche:	
Tensione Nominale	7,2V
Numero Elementi	2
Capacita' mAh	2000
Valore C (in scarica)	12
Corrente di scarica	max 24A
Dimensioni	110 x 36 x 19 mm
Peso	125gr
Connettore di potenza	Multiplex M6
Connettore di bilanciamento	MPX/TP/FP

Figure 4.10: Specifiche batteria Lipo 7 Volt

4.7.2 Caratteristica batteria da 12 Volt

Caratteristiche LIPO 12 Volt:

1. Tensione nominale 11,1 V
2. capacità di 1800 mAh-20C.
3. Continuos Discharge: 20C (36 A)
4. Max Discharge: 40C (72 A)
5. Balance charge connector: US Standard Connector.
6. Dimensioni: 104x35x23 mm
7. peso: 152 g. N.B.

4.8 Breadboard e componentistiche varie

Si è fatto uso di breadboard, fili necessari per il collegamento, resistenze, oltre la necessità di un saldatore a stagno utile al fine di avere collegamenti pià robusti.

Figure 4.11: Breadboard e componenti varie utilizzate

Chapter 5

Appendice

Di seguito verrano proposti i codici utilizzati in Matlab e sul microcontrollore Arduino

5.1 Codice Matlab

```
clc
clear all
g=9.81; %Gravity(m/s^2)
r=0.098; %Radius of wheel(m)
Mw=0.2; %Mass of wheel(kg) //PESANO 102 g l'una
Mp=1.3; %Mass of body(kg)
Iw=0.000144; %Inertia of the wheel(kg*m^2) *http://www.claredot.net/it/sez
%%%%%%%%%%inerzia
b=0.08 %lunghezza lato b
a=0.18 %lunghezza lato a
d=0.05 % lunghezza asse di rotazione da quello baricentroco
Mparall=1.4 %Massa parallelepipedo
JparallBar=Mparall*(a^2+b^2)/12 %momento rispetto baricentr
Jruote=JparallBar+Mparall*d^2 %per Hugens
%%%%Inerzia fine
```

```
Ip=Jruote;%Ip=0.0041; %Inertia of the body(kg*m^2)
l=0.03%l=0.07; %Length to the body's centre of mass(m)
%Motor's variables
%con duty cicle 130 al 56% di 12 V ho
volt=12*.56;
wvuot=39; %rad/s
Km=volt/wvuot; %Motor torque constant (Nm/A)
torqueStop=0.69; %Nm
Ke = Km %Back EMF constant (Vs/rad)
R = volt^2/(torqueStop*wvuot); %Nominal Terminal Resistance (Ohm)
% Va = voltage applied to motors for controlling the pendulum
%%%%%%%%%%
% System Matrices
%%%%%%%%%%
%pre-calculated to simplyfy the matrix
%Denominator for the A and B matrices
beta = (2*Mw+(2*Iw/r^2)+Mp);
alpha = (Ip*beta + 2*Mp*l^2*(Mw + Iw/r^2));
A = [0 1 0 0;
0 (2*Km*Ke*(Mp*l*r-Ip-Mp*l^2))/(R*r^2*alpha) (Mp^2*g*l^2)/alpha 0;
0 0 0 1;
0 (2*Km*Ke*(r*beta - Mp*l))/(R*r^2*alpha) (Mp*g*l*beta)/alpha 0]
B = [ 0;
(2*Km*(Ip + Mp*l^2 - Mp*l*r))/(R*r*alpha);
0;
(2*Km*(Mp*l-r*beta)/(R*r*alpha))]
C= [1 0 0 0;
    0,1,0,0;
0 0 1 0;0,0,0,1]
D=[0;0;0;0]

cond_iniz=[0,0,0.1,0]
%%
```

```
B=12/231*B;
sys=ss(A,B,C,D);
%% Funzioni di trasferimento aggiustate
[numh,denh] = ss2tf(A,B,C,D);

% arrotondamenti
for jj=1:4
 for ii=1:5
 if abs(numh(jj,ii))<1e-4
    numh(jj,ii)=0;
 end

 end
end
for ii=1:4
 if abs(denh(ii))<1e-4
    denh(ii)=0;
 end

end

g11=tf(numh(1,:),denh(1,:))
g12=tf(numh(2,:),denh(1,:))
g13=tf(numh(3,:),denh(1,:))
g14=tf(numh(4,:),denh(1,:))
%%


disp('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;
```

```
%%Inizio LMI
B=B;
treno=ss(A,B,C,D)

%Inoltre è possibile avere conferma dell asintotica stabilità
%del sistema in quanto esiste una matrice definita
%positiva P tale che A'P + PA < 0')
P00=sdpvar(4)
vincoli0=[P00>0,A'*P00+P00*A<0]
solvesdp(vincoli0)
disp('Utilizzando il toolbox per LMI, si dimostra che il sistema e'
disp('se esiste una matrice P definita positiva, si vedano gli auto
P000=double(P00)
eig(P000)
```

```
disp('Osservabilità')
disp('Il sistema è osservabile se il rango della matrice di osservabilità
rank(obsv(treno))
disp('controllabilità')
disp('Il sistema è controllabile se il rango della matrice di controllabi
rank(ctrb(treno))
%determiniamo ora il guadagno del controllore K attraverso la risoluzione
%di una LMI
P = sdpvar(4);
L = sdpvar(1,4);
vincoli=[P>0,A*P+B*L+P*A'+L'*B'<0];
solvesdp(vincoli);
P0 = double(P);
L0 = double(L);
disp('matrice dei guadagni trovata con LMI')
K0 = L0*inv(P0)
eig(A+B*K0)
disp('sistema asintoticamente stabile')


%controlore con autovalori a parte reale minore di -3
P = sdpvar(4);
L = sdpvar(1,4);
vincoli=[P>0,A*P+B*L+P*A'+L'*B'<0];
solvesdp(vincoli);
P0 = double(P);
L0 = double(L);
disp('matrice dei guadagni trovata con LMI')
K0 = L0*inv(P0)
eig(A+B*K0)
disp('sono come richiesto a parte reale minore di -3')
%Fine LMI
```

```
%%LQR
Q0=diag([1 1 100 100]);
Q1=diag([1 1 100 1]);%diag([1 1 100 100]);
Q2=diag([1 1 1 1]);
Q3=diag([100 10 1 1]);
R=100;
kd0=lqr(sys,Q0,R)
kd1=lqr(sys,Q1,R)
kd2=lqr(sys,Q2,R)
kd3=lqr(sys,Q3,R)



% tempo discreto
Ts=1e-2; %10ms

[F,G,H,J]=c2dm (A,B*12/231,C,D,Ts,'zoh')
sys1=ss(F,G,H,J,Ts)
kdTD=lqr(sys1,Q1,R)


%%
%Simulazione a Ciclo aperto
T=0:0.02:2
%Impulse response input
U=zeros(size(T));
U(1)= 12;
U(10)=0;

[Y,X]=lsim(A,B,C,D,U,T);
%plot the states
subplot(2,1,1)
plot(T,[X(:,1) X(:,2) X(:,3) X(:,4)]), xlabel('Tempo [s]'),
```

```
ylabel('Variabili di stato')
legend('Posizione[m]','Velocità[m/s]','Angolo[rad]','Velocità angolare[rad
title(' Risposta impulsiva del sistema')
%plot the control input
subplot(2,1,2)
plot(T,U), xlabel('Tempo [s]'), ylabel('Controllo u [V]')
legend('Tensione [V]')
title('Ingresso implulsivo')

%%plot dati simulazione
subplot(3,1,1)
plot(dati.time,dati.signals.values)
title('Diagramma variabili di stato')
xlabel('tempo [s]')
ylabel('Variabili di stato')
legend('Posizione[m]','Velocità[m/s]','Angolo[rad]','Velocità angolare[rad
grid
subplot(3,1,2)
plot(vcontrollo.time,vcontrollo.signals.values)
title('Diagramma Segnale di Controllo')
xlabel('tempo [s]')
ylabel('Segnale di Controllo [V]')
grid
subplot(3,1,3)
plot(disturbo.time,disturbo.signals.values)
title('Diagramma disturbo angolare di 5°')
xlabel('tempo [s]')
ylabel('Disturbo angolare [rad]')
grid
```

5.2 Codice Arduino

Il software è articolato in vari blocchi concettuali:

5.2.1 Main

```
#include <Wire.h>
#include "prova.h"
/* Used for timing */
unsigned long timer;
unsigned long inizio;
double forza=0;
//Used for timing
int STD_LOOP_TIME  = 10000;
int lastLoopTime = STD_LOOP_TIME;
int lastLoopUsefulTime = STD_LOOP_TIME;
unsigned long loopStartTime = 0;

//encoder
volatile double countRight = 0;
double lin_speed=0;

////////////////////////////
////////////////////////////
////////IMU VARIABLES
////////////////////////
////////////////////////
// Accelerometer ADXL345
#define ACC (0x53)    //ADXL345 ACC address
#define A_TO_READ (6)        //num of bytes we are going to read ea
// Gyroscope ITG3200
#define GYRO 0x68 // gyro address, binary = 11101001 when AD0 is co
#define G_SMPLRT_DIV 0x15
#define G_DLPF_FS 0x16
#define G_INT_CFG 0x17
#define G_PWR_MGM 0x3E
#define G_TO_READ 8 // 2 bytes for each axis x, y, z
```

```
// offsets are chip specific.
int g_offx = 120;
int g_offy = 20;
int g_offz = 93;
char str[512];
boolean firstSample = true;
float RwAcc[3];  //projection of normalized gravitation force vector on x
float Gyro_ds[3];  //Gyro readings
float RwGyro[3];        //Rw obtained from last estimated value and gyro
float Awz[2];           //angles between projection of R on XZ/YZ plane a
float RwEst[3];
float wGyro = 10.0;

/* Kalman filter variables and constants */
const double Q_angle = 0.001; // Process noise covariance for the acceler
const double Q_gyro = 0.003; // Process noise covariance for the gyro - S
const double R_angle = 0.03; // Measurement noise covariance - Sv

double angle; // The angle output from the Kalman filter
double bias = 0; // The gyro bias calculated by the Kalman filter
double P_00 = 0, P_01 = 0, P_10 = 0, P_11 = 0;
double dt, y, S;
double K_0, K_1;

////////////////////////////
////////////////////////////
////////CONTROL VARIABLES
////////////////////////////
////////////////////////
float state[4] = {
  0, 0, 0, 0};   //state vector
float force = 0;
//control parameters
```

```
const float matr_Kn[4] = {
  -0.4, -150.6958, -588.8032, -118.8468};//{ -0, 0, -67917, -14536}

//////////////////////
//////////////////////
//////////////////////
void setup(){
  Serial.begin(115200);
  /////////IMU
  initAcc();
  initGyro();
  Wire.begin();
  /* Setup encoders */
  pinMode(leftEncoder1,INPUT);
  pinMode(leftEncoder2,INPUT);
  //attachInterrupt(1, rencoder, FALLING);
  attachInterrupt(1,readEncoder,CHANGE);
  /* Setup motor pins to output */
  sbi(leftPwmPortDirection,leftPWM);
  sbi(leftPortDirection,leftA);
  sbi(leftPortDirection,leftB);
  sbi(rightPwmPortDirection,rightPWM);
  sbi(rightPortDirection,rightA);
  sbi(rightPortDirection,rightB);

  /* Set PWM frequency to 20kHz - see the datasheet http://www.atme
  // Set up PWM, Phase and Frequency Correct on pin 9 (OC1A) & pin
  TCCR1A = 0; // clear all
  TCCR1B = 0; // clear all
  TCCR1B |= _BV(WGM13) | _BV(CS10); // Set PWM Phase and Frequency
  ICR1H = (PWMVALUE >> 8); // ICR1 is the TOP value - this is set s
  ICR1L = (PWMVALUE & 0xFF);
```

```
  /* Enable PWM on pin 9 (OC1A) & pin 10 (OC1B) */
  // Clear OC1A/OC1B on compare match when up-counting
  // Set OC1A/OC1B on compare match when downcountin
  TCCR1A |= _BV(COM1A1) | _BV(COM1B1);
  setPWM(leftPWM,0); // Turn off pwm on both pins
  setPWM(rightPWM,0);

  loopStartTime = micros();
  timer = loopStartTime;
}
void loop(){
  lastLoopUsefulTime = micros()-loopStartTime;
  if(lastLoopUsefulTime<STD_LOOP_TIME)
    delayMicroseconds(STD_LOOP_TIME-lastLoopUsefulTime);
  loopStartTime = micros();
  inizio=micros();
  getImu();
  //Serial.println(micros()-loopStartTime); stampo tempo lettura imu

    //control part
  state[0] = getPos();
  state[1] = getSpeed();
  state[2] = getAngle();
  state[3] = getGyro();
  force = 0;
  force=matr_Kn[0]*state[0]+matr_Kn[1]*state[1]+matr_Kn[2]*state[2]+matr_
  force=-(float)force; //%**********controllo qui
  if(abs(getAngle()-0) > .60){ //(getAngle()-0) > .50)
    force=0;
    Drive(0);
  }
  else {
    Drive((float)force*0.3);
```

```
  }
  // Serial.println(micros()-inizio); 6ms per leggere e attuare il
  printAll();
}
```

5.2.2 IMU

```
//////////////////////////
//////////////////////////
////IMU CODE
//////////////////////////
//////////////////////////
void initAcc() {
  //Turning on the ADXL345
  writeTo(ACC, 0x2D, 0);
  writeTo(ACC, 0x2D, 16);
  writeTo(ACC, 0x2D, 8);
  //by default the device is in +-2g range reading
}

void getAccelerometerData(int * result) {
  int regAddress = 0x32;    //first axis-acceleration-data register
  byte buff[A_TO_READ];

  readFrom(ACC, regAddress, A_TO_READ, buff); //read the acceleratio

  //each axis reading comes in 10 bit resolution, ie 2 bytes.
Least Significat Byte first!!
  //thus we are converting both bytes in to one int
  result[0] = (((int)buff[1]) << 8) | buff[0];
  result[1] = (((int)buff[3])<< 8) | buff[2];
  result[2] = (((int)buff[5]) << 8) | buff[4];
}
```

```
void rawAccToG(int * raw, float * RwAcc) {
  RwAcc[0] = ((float) raw[0]) / 256.0;
  RwAcc[1] = ((float) raw[1]) / 256.0;
  RwAcc[2] = ((float) raw[2]) / 256.0;
}

//initializes the gyroscope
void initGyro()
{
  /**************************************
  * ITG 3200
  * power management set to:
  * clock select = internal oscillator
  *     no reset, no sleep mode
  *   no standby mode
  * sample rate to = 125Hz
  * parameter to +/- 2000 degrees/sec
  * low pass filter = 5Hz
  * no interrupt
  **************************************/
  writeTo(GYRO, G_PWR_MGM, 0x00);
  writeTo(GYRO, G_SMPLRT_DIV, 0x07); // EB, 50, 80, 7F, DE, 23, 20, FF
  writeTo(GYRO, G_DLPF_FS, 0x1E); // +/- 2000 dgrs/sec, 1KHz, 1E, 19
  writeTo(GYRO, G_INT_CFG, 0x00);
}


void getGyroscopeData(int * result)
{
  /**************************************
  Gyro ITG-3200 I2C
  registers:
  temp MSB = 1B, temp LSB = 1C
```

```
  x axis MSB = 1D, x axis LSB = 1E
  y axis MSB = 1F, y axis LSB = 20
  z axis MSB = 21, z axis LSB = 22
  ************************************/

  int regAddress = 0x1B;
  int temp, x, y, z;
  byte buff[G_TO_READ];

  readFrom(GYRO, regAddress, G_TO_READ, buff); //read the gyro data

  result[0] = ((buff[2] << 8) | buff[3]) + g_offx;
  result[1] = ((buff[4] << 8) | buff[5]) + g_offy;
  result[2] = ((buff[6] << 8) | buff[7]) + g_offz;
  result[3] = (buff[0] << 8) | buff[1]; // temperature
}


// convert raw readings to degrees/sec
void rawGyroToDegsec(int * raw, float * gyro_ds) {
  gyro_ds[0] = ((float) raw[0]) / 14.375;
  gyro_ds[1] = ((float) raw[1]) / 14.375;
  gyro_ds[2] = ((float) raw[2]) / 14.375;
}


void normalize3DVec(float * vector) {
  float R;
  R = sqrt(vector[0]*vector[0] + vector[1]*vector[1] + vector[2]*ve
  vector[0] /= R;
  vector[1] /= R;
  vector[2] /= R;
}
```

```
float squared(float x){
  return x*x;
}


void getInclination() {
  int w = 0;
  float tmpf = 0.0;
  int signRzGyro;

  if (firstSample) { // the NaN check is used to wait for good data from
    for(w=0;w<=2;w++) {
      RwEst[w] = RwAcc[w];     //initialize with accelerometer readings
    }
  }
  else{
    //evaluate RwGyro vector
    if(abs(RwEst[2]) < 0.1) {
      //Rz is too small and because it is used as reference for computing
      //in this case skip the gyro data and just use previous estimate
      for(w=0;w<=2;w++) {
        RwGyro[w] = RwEst[w];
      }
    }
    else {
      //get angles between projection of R on ZX/ZY plane and Z axis, bas
      for(w=0;w<=1;w++){
        tmpf = Gyro_ds[w];                    //get current gyro rate
        tmpf *= lastLoopUsefulTime / 1000000.0f;
//get angle change in deg
```

```
        Awz[w] = atan2(RwEst[w],RwEst[2]) * 180 / PI;
//get angle and convert to degrees
        Awz[w] += tmpf;              //get updated angle according to
      }

      //estimate sign of RzGyro by looking in what qudrant the angle
      //RzGyro is pozitive if  Axz in range -90 ..90 => cos(Awz) >=
      signRzGyro = ( cos(Awz[0] * PI / 180) >=0 ) ? 1 : -1;

      //reverse calculation of RwGyro from Awz angles, for formulas
http://starlino.com/imu_guide.html
      for(w=0;w<=1;w++){
        RwGyro[0] = sin(Awz[0] * PI / 180);
        RwGyro[0] /= sqrt( 1 + squared(cos(Awz[0] * PI / 180)) * sq
        RwGyro[1] = sin(Awz[1] * PI / 180);
        RwGyro[1] /= sqrt( 1 + squared(cos(Awz[1] * PI / 180)) * sq
      }
      RwGyro[2] = signRzGyro * sqrt(1 - squared(RwGyro[0]) - squared
    }

    //combine Accelerometer and gyro readings
    for(w=0;w<=2;w++) RwEst[w] = (RwAcc[w] + wGyro * RwGyro[w]) / (

    normalize3DVec(RwEst);
  }

  firstSample = false;
}

 void getImu(){
  if(!Serial.available()) {
    int acc[3];
    int gyro[4];
```

```
    getAccelerometerData(acc);
    rawAccToG(acc, RwAcc);
    normalize3DVec(RwAcc);

    getGyroscopeData(gyro);
    rawGyroToDegsec(gyro, Gyro_ds);

    getInclination();
  }
 }

void serialFloatPrint(float f) {
  byte * b = (byte *) &f;
  Serial.print("f:");
  for(int i=0; i<4; i++) {

    byte b1 = (b[i] >> 4) & 0x0f;
    byte b2 = (b[i] & 0x0f);

    char c1 = (b1 < 10) ? ('0' + b1) : 'A' + b1 - 10;
    char c2 = (b2 < 10) ? ('0' + b2) : 'A' + b2 - 10;

    Serial.print(c1);
    Serial.print(c2);
  }
}


//-------------------- Functions
//Writes val to address register on ACC
void writeTo(int DEVICE, byte address, byte val) {
   Wire.beginTransmission(DEVICE); //start transmission to ACC
   Wire.write(address);        // send register address
```

```
    Wire.write(val);        // send value to write
    Wire.endTransmission(); //end transmission
}


//reads num bytes starting from address register on ACC in to buff
void readFrom(int DEVICE, byte address, int num, byte buff[]) {
  Wire.beginTransmission(DEVICE); //start transmission to ACC
  Wire.write(address);        //sends address to read from
  Wire.endTransmission(); //end transmission

  Wire.beginTransmission(DEVICE); //start transmission to ACC
  Wire.requestFrom(DEVICE, num);    // request 6 bytes from ACC

  int i = 0;
  while(Wire.available())    //ACC may send less than requested (ab
  {
    buff[i] = Wire.read(); // receive a byte
    i++;
  }
  Wire.endTransmission(); //end transmission
}
//////////////////////////
//////////////////////////
//////GET DATA CODE
//////////////////////////
//////////////////////////
float getGyro(){
return  (Gyro_ds[0]*0.0175)-0.16;
}
float getAngle(){
    dt = double(lastLoopUsefulTime)/1000000; // Convert from micros
  // Discrete Kalman filter time update equations - Time Update ("P
```

```
  // Update xhat - Project the state ahead
  angle += dt * (((Gyro_ds[0]*0.0175)-0.15) - bias);
  // Update estimation error covariance - Project the error covariance ah
  P_00 += -dt * (P_10 + P_01) + Q_angle * dt;
  P_01 += -dt * P_11;
  P_10 += -dt * P_11;
  P_11 += +Q_gyro * dt;
  // Discrete Kalman filter measurement update equations - Measurement Up
  // Calculate Kalman gain - Compute the Kalman gain
  S = P_00 + R_angle;
  K_0 = P_00 / S;
  K_1 = P_10 / S;
  // Calculate angle and resting rate - Update estimate with measurement
  y = RwAcc[0]*1.57 - angle;
  angle += K_0 * y;
  bias += K_1 * y;
  // Calculate estimation error covariance - Update the error covariance
  P_00 -= K_0 * P_00;
  P_01 -= K_0 * P_01;
  P_10 -= K_1 * P_00;
  P_11 -= K_1 * P_01;
  return angle;
}
```

5.2.3 Motore

```
void printAll(){
  Serial.print("Posizione= ");
 Serial.print(state[0]);
Serial.print(",  velocitÃ = ");
 Serial.print(state[1]);
 Serial.print(",  angolo= ");
 Serial.print(state[2]);
 Serial.print(",  velocitÃ  angolare= ");
```

```
  Serial.print(state[3]);
  Serial.print(",  Force= ");
  Serial.println(force);
}


void setPWM(uint8_t pin, int dutyCycle) { // dutyCycle is a value b
  if(pin == leftPWM) {
    OCR1AH = (dutyCycle >> 8);
    OCR1AL = (dutyCycle & 0xFF);
  } else if (pin == rightPWM) {
    OCR1BH = (dutyCycle >> 8);
    OCR1BL = (dutyCycle & 0xFF);
  }
}

void Drive(float torque){
  int offset;
  if(torque > 400)
    torque = 400;
  if(torque<-400)
  torque=-400;
if (torque <=-1) {
      cbi(leftPort,leftA);
      sbi(leftPort,leftB);
      sbi(rightPort,rightA);
      cbi(rightPort,rightB);
    offset= map(abs(torque),0,400,20,400);

    }
    else  if (torque >=1) { //controllare ad azionare con segno opp
      sbi(leftPort,leftA);
      cbi(leftPort,leftB);
```

```
        cbi(rightPort,rightA);
        sbi(rightPort,rightB);
        offset= map(abs(torque),0,400,20,400);
      }
       else    {
        sbi(leftPort,leftA);
        cbi(leftPort,leftB);
        sbi(rightPort,rightA);
        cbi(rightPort,rightB);
        offset= 0;
      }
      setPWM(leftPWM, offset); // Left motor pwm
      setPWM(rightPWM,offset); // Right motor pwm
    }
float getPos(){
  return countRight*0.001;//posnow = ((count*2*pi/464)*radius;//? //SPOSTA
}
double getSpeed()  {
// calculate speed, volts and Amps
static double countRightAnt = 0;// last count
 // lin_speed =((countRight - countRightAnt)*(60*(100/STD_LOOP_TIME)))/(2
    //lin_speed =(countRight - countRightAnt)*2.6352;    //((countRight - c
  lin_speed =(countRight - countRightAnt)*(double)0.0967; // per 4 Ms=2.6
 countRightAnt = countRight;
return lin_speed;
}
void rencoder()  {
// pulse and direction, direct port reading to save cycles
  if (PIND & 0b00000001)    countRight--;
// if(digitalRead(encodPinB1)==HIGH)   count++    (on DI #2)
  else                      countRight++;
}
void readEncoder(){
```

```
        if((bool)(PIND & _BV(3)) == (bool)(PIND & _BV(6))) // pin 2 =
            countRight++;
    else
        countRight--;
}
```

List of Figures

Bibliography

[1] Rich Chi Ooi, Balancing a Two-Wheeled Autonomous Robot, 2003;

[2] Paolo Roberto Di Gregorio, Realizzazione di un vicon con led IR mediante Filtro di Kalman, 2012;