Universitá degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

IDENTIFICAZIONE E CONTROLLO DI UN VEICOLO ELETTRICO

Laureando

Marco Bergamin

Relatore

Prof. Mauro Bisiacco

Padova, 21 febbraio 2013

The most exciting phrase to hear in science, the one that heralds new discoveries, is not 'Eureka' (I found it) but 'That's funny...'

Isaac Asimov

Indice

Capitolo 1

Introduzione

Lo scopo di questa tesi è quello di identificare e controllare un sistema a scatola grigia. Nello specifico un modellino motorizzato di automobile che sicuramente qualche appassionato di modellismo conosce già. Rappresenta per noi un sistema detto a "scatola grigia" perchè le equazioni che lo governano sono facilmente deducibili ma non è noto nessuno dei parametri delle equazioni.
Il fine del lavoro che svilupperemo è quello di controllare la frenata del modellino in modo che si fermi ad una fissata distanza dall'ostacolo posto di fronte mediante un controllore PID (proporzionale integrativo derivativo) ed un sensore di distanza ad ultrasuoni.

Nella prima parte della tesi daremo uno sguardo al sistema con cui avremo a che fare, e dopo aver fatto le dovute ipotesi di lavoro andremo a costruire un modello che possa descriverlo con la dovuta accuratezza, tenendo conto del fatto che complicare troppo il modello non ci è di nessun aiuto a causa dell'impossibilità di misurare la maggior parte dei parametri incogniti con i mezzi in nostro possesso.

Nella seconda parte andremo a costruire un circuito elettronico che ci permetta di effettuare il controllo servendoci di un *Microcontroller*, un piccolo *Driver* per motori DC (*Direct Current*) ed infine un sensore ad ultrasuoni.

Nella terza parte implementeremo a livello di microcontrollore un controllore di tipo PID.

Nella quarta ed ultima parte osserveremo come si comporta il sistema controllato e riusciremo a stimare i parametri incogniti della FDT a partire dalle misurazioni fatte. Andremo infine a effettuare alcune considerazioni sulla verosimilità delle ipotesi fatte inizialmente confrontando i risultati delle simulazioni con le misurazioni reali.

Capitolo 2

Sistema meccanico

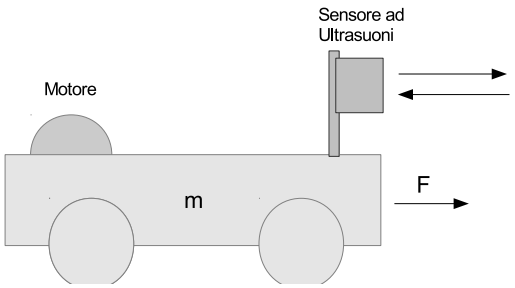

Figura 2.1: Sistema meccanico

2.1 Descrizione del sistema meccanico

Il sistema meccanico considerato è un modellino in scala di una autovettura dotato di un motore a corrente continua che fornisce una coppia trasferita alle ruote posteriori tramite una serie di riduttori meccanici. Il modellino non è in grado di sterzare quindi si muoverà lungo un determinato asse.

2.2 Ipotesi di Lavoro

Siccome non conosciamo le specifiche del motore, il suo rendimento, il rapporto di trasmissione dei riduttori ed altri parametri e non ci è possibile misurarli, non ci è di nessun aiuto tener conto di queste componenti durante la costruzione del modello. Per il momento le ignoreremo. Assumeremo quindi valide le seguenti ipotesi di lavoro:

- I momenti di inerzia delle ruote e dei riduttori interni sono trascurabili.
- Sul sistema agisce solo una generica forza di attrito proporzionale alla velocità.
- Il motore fornisce una coppia proporzionale alla tensione di alimentazione.

- L'ambiente in cui il sistema si muove è statico, in questo modo possiamo ricavare velocità e accelerazione del sistema osservando l'evoluzione temporale della distanza tra sistema ed oggetto rilevato dal sonar.

Le prime due ipotesi sono ragionevoli: intuitivamente possiamo immaginare che le costanti di tempo collegate ai momenti di inerzia dei riduttori e delle ruote siano molto più piccole rispetto alla costante di tempo collegata al moto dell'intero sistema, che diventa quella dominante.
La terza ipotesi invece non tiene conto della Forza Contro-Elettromotrice introdotta dagli avvolgimenti del motore che ruotano all'interno del campo magnetico. Questa forza va a ridurre la coppia effettivamente erogata in modo proporzionale alla velocità. Purtroppo siamo costretti a fare questa ipotesi perchè non disponiamo di un Driver che controlli il motore in corrente e perchè non sono note le specifiche del motore.
Abbiamo bisogno della quarta ipotesi perchè non disponiamo di un Encoder per contare i giri del motore (e di conseguenza valutare lo spostamento e velocità della macchina).
Inizialmente si era tentato di costruire un Encoder Ottico ad-hoc per il nostro sistema con un Emettitore ed un Ricevitore ad infrarossi che andava a contare i giri delle ruote. Tale problema è stato accantonato per una serie di limitazioni hardware del Microcontrollore che avrebbero costretto la creazione di un circuito dedicato all'encoder che avrebbe ingrandito troppo le dimensioni della scheda. La principale limitazione consisteva nel fatto che gli unici due piedini che potevano essere impiegati per catturare segnali Interrupt/Trigger provenienti dall'encoder erano già usati per la porta seriale. Con una Elettronica/Meccanica migliore avremmo quindi potuto rimuovere le ultime due ipotesi.

2.3 Costruzione di un Modello

Sotto queste ipotesi possiamo ricondurre il nostro sistema al classico problema del moto di un carrello soggetto a due forze distinte: una che imprime un'accelerazione ed un'altra (forza di attrito) che invece si oppone al moto. Chiameremo:

- F_{mot} la Forza che il motore imprime al sistema
- F_{att} la Forza di attrito che si oppone al moto del sistema
- F la Forza risultante applicata al sistema
- m la massa del sistema
- K_{att} il coefficiente di attrito
- K_{ft} il coefficiente che lega la forza che il motore imprime alla macchina alla tensione che gli viene fornita
- $v(t)$ la tensione applicata al motore
- $x(t)$ la posizione del sensore rispetto all'ostacolo posto davanti al sistema.

Le forze che agiscono sul sistema sono:

$$F(t) = F_{mot}(t) - F_{att}(t) \tag{2.1}$$

Dalla *Seconda Legge di Newton*:

$$F(t) = m\ddot{x}(t) \tag{2.2}$$

mentre la forza di attrito la esprimiamo come:

$$F_{att}(t) = K_{att}\dot{x}(t) \tag{2.3}$$

Esprimiamo il legame tra forza generata dal motore e tensione ad esso applicata come:

$$F_{mot}(t) = K_{ft}v(t) \tag{2.4}$$

e combinandole assieme otteniamo:

$$m\ddot{x}(t) = K_{ft}v(t) - K_{att}\dot{x}(t) \tag{2.5}$$

Passiamo ora dal Dominio del Tempo al Dominio di Laplace:

$$ms^2X(s) = K_{ft}V(s) - K_{att}(t)sX(s) \tag{2.6}$$

Dopo un paio di passaggi arriviamo all'espressione:

$$X(s) = \frac{\frac{K_{ft}}{K_{att}}}{\frac{m}{k_{att}}s + 1}\frac{1}{s}V(s) \tag{2.7}$$

che, defininendo i due parametri $K_m := \frac{K_{ft}}{K_{att}}$ e $\tau_m := \frac{m}{K_{att}}$ e chiamando $G(s)$ la FDT tra $v(t)$ e $x(t)$ conduce a

$$G(s) := \frac{K_m}{\tau_m s + 1}\frac{1}{s} \tag{2.8}$$

Se vogliamo dare un'interpretazione fisica: K_m è un coefficiente che lega velocità massima raggiunta dal sistema a regime con la tensione applicata, mentre τ_m fornisce l'informazione sul tempo necessario per raggiungere circa il 63% di questa velocità.

Infatti, se consideriamo la FDT che lega la velocità con la tensione:

$$sX(s) = \frac{K_m}{\tau_m s + 1}V(s) \tag{2.9}$$

e supponiamo di applicare in ingresso un gradino di tensione $V(s) = \frac{V}{s}$ al nostro sistema, con V tensione fissata, facendo la Trasformata Inversa di Laplace dell'espressione:

$$sX(s) = \frac{K_m}{\tau_m s + 1}\frac{V}{s} \tag{2.10}$$

otteniamo nel Dominio del Tempo l'equazione differenziale:

$$\dot{x}(t) = K_m V(1 - e^{-\frac{t}{\tau_m}}) \tag{2.11}$$

Con qualche semplice passaggio è possibile verificare che la dimensione di K_m è proprio $[\frac{m}{V \cdot s}]$ mentre la dimensione di τ_m è $[s]$.
Ci serviremo di queste relazioni per stimare K_m e τ_m a partire da misurazioni nel Dominio del Tempo e per costruire il modello Simulink.

Capitolo 3

Sistema di controllo

Il controllo che andremo a costruire è schematizzabile in questo modo:

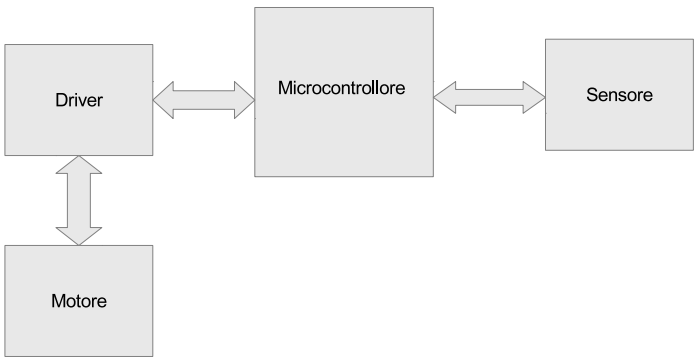

Figura 3.1: Sistema di controllo

Utilizzeremo un Microcontrollore per implementare il controllo. Il microcontrollore andrà a leggere la distanza misurata dal sensore. Dai dati acquisiti dal sensore ricaveremo sia come varia la distanza dall'ostacolo sia la velocità (grazie alla quarta ipotesi fatta introducendo il sistema meccanico). A partire da queste due grandezze il nostro Controllore PID andrà a modificare la tensione applicata al motore.

3.1 Microcontrollore

Nello specifico è stato scelto un Microcontrollore prodotto dall'ATMel, precisamente un ATMega328. Si tratta di un sistema a 8bit funzionante con Clock di 16MHz dotato di una serie di ingressi sia analogici (include un ADC a 10bit) che digitali. Include anche delle uscite Digitali, alcune delle quali possono essere usate per produrre un'onda con modulazione PWM ("pulse-width modulation") ad una frequenza di circa 490Hz, ed useremo proprio questi pin per pilotare il motore attraverso il driver: variando il Duty Cycle dell'onda variamo il valore efficace della tensione di alimentazione del motore.
Ecco lo schema della piedinatura del controllore:

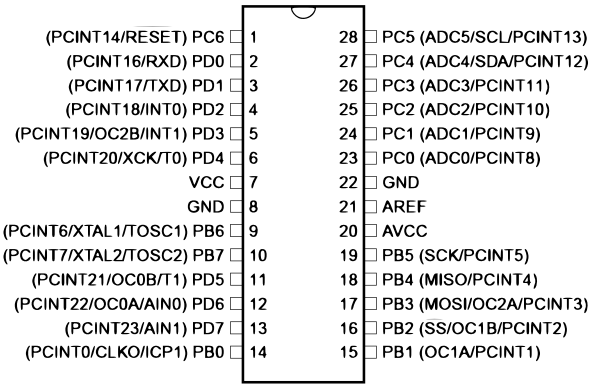

Figura 3.2: Schema AtMega328

I collegamenti che andremo a fare saranno i seguenti:

- pin 1-2-3: (Rispettivamente Reset, Rx e Tx) ci servono per la stabilire una connessione tra computer e microcontrollore. Questo ci serve per poter raccogliere le misure e per effettuare modifiche al firmware. Al pin di Reset andremo inoltre a collegare un pulsante che ci permetterà all'occorrenza di riavviare il software.
- pin 4: andremo a collegare un altro pulsante, che useremo come segnale di Start per far partire il Controllo.
- pin 7-8: a questi pin collegheremo l'almentazione del nostro Microcontrollore.
- pin 9-10: a questi pin collegheremo un oscillatore a cristallo a 16MHz.
- pin 11-12: queste sono due delle uscite Digitali (PWM) menzionate in precedenza. Collegheremo queste due uscite agli ingressi del Driver, il che ci permetterà di impostare la velocità (variando il Duty Cycle) ed il verso di rotazione del motore (selezionando una o l'altra uscita).
- pin 13-14: imposteremo il pin 13 come ingresso digitale per leggere la distanza misurata dal Sensore ad Ultrasuoni ed il pin 14 come uscita per generare un impulso che il sensore interpreterà come una richiesta di lettura della distanza.
- pin 19: lo imposteremo come uscita digitale per pilotare un Diodo Led. Esso potrà tornare utile come indicatore visivo per segnalare in che stato si trova il programma.

3.2 Sensore ad Ultrasuoni

Il sensore scelto è stato l'HC-SR04 (figura 3.3). La scelta è ricaduta su questo sensore per diversi motivi: è economico, immediato da usare e molto diffuso.
Secondo i dati forniti dal DataSheet dovrebbe avere una portata di circa $4m$ ed una risoluzione di circa $30mm$. Nella realtà si è notato che sebbene la risoluzione corrisponda circa a quella dichiarata, la portata è invece più limitata. Non siamo riusciti ad ottenere misure di distanze superiori ai $3.5m$. Il che purtroppo limita la capacità di misurare la distanza in funzione del tempo, complicandoci il lavoro per via del range di misura relativamente limitato.
Il sensore è dotato di 4 pin: due di essi sono Vdd e GND e servono per alimentarlo, gli altri due si chiamano TRIG ed ECHO. Il primo accetta in ingresso un impulso di una decina di μs a livello logico alto per inizializzare l'operazione di lettura della distanza. Il secondo restituisce un segnale a livello logico alto di durata proporzionale alla distanza: una durata di $58\mu s$ corrisponde ad $1cm$.
In realtà questo modo di restituire la distanza è abbastanza limitante nel caso in cui volessimo fare molte letture al secondo. Assumendo che legga sempre una distanza di $4m$, che corrisponde ad un impulso della durata di $23.2ms$, in un secondo potrebbe restituirci il valore della distanza di al più 43 letture. Questo significa che se ad esempio decidessimo di leggere la distanza ad intevalli di $50ms$, per quasi la metà del tempo il nostro microcontrollore sarebbe occupato a leggere il segnale restituito dal sonar. Sarebbe stato opportuno delegare questo compito ad un altro controllore più semplice, programmato esclusivamente per leggere la distanza dal sonar e fornirla al controllore principale sotto forma di pacchetto di pochi byte.

Figura 3.3: Sensore HC-SR04

3.3 Driver

Il driver è un L293D prodotto dalla Texas Instruments. Esso incorpora 4 piccoli driver half-H che configurati opportunamente possono essere usati per formare due driver full-H (chiamati anche ponte-H). Si tratta di una configurazione particolare che consente di alimentare il motore con una tensione di segno positivo o negativo in base alla configurazione degli ingressi. Questo ci consente di decidere il verso di rotazione del rotore del motore. Il driver è inoltre progettato per accettare in ingresso segnali PWM con frequenza fino a $5KHz$, consentendoci di modulare il valore efficace della tensione applicata al motore.
Di seguito forniamo un esempio applicativo (preso dal datasheet) di utilizzo del driver in configurazione full-H:

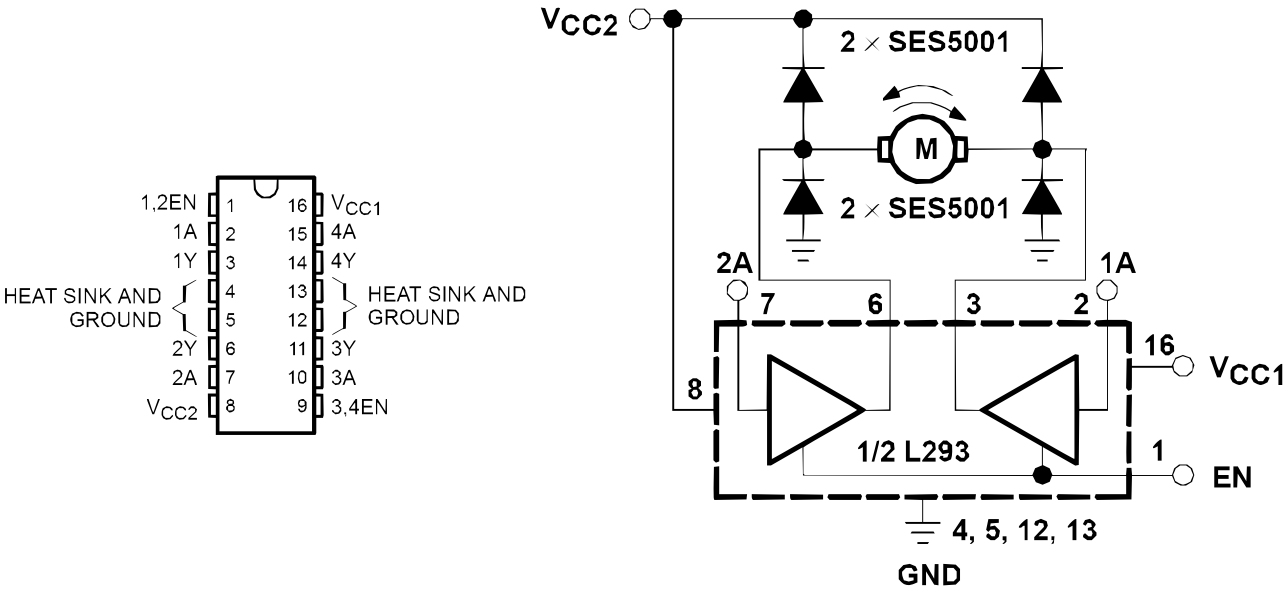

Figura 3.4: Driver L293D

Andremo a collegare le due uscite PWM del microcontrollore (pin 11 e 12) agli ingressi 2A ed 1A che si vedono in figura.
V_{cc2} è la tensione di alimentazione del motore, e può variare da $4.5V$ a $36V$, V_{cc1} è la tensione di alimentazione della parte "logica" del Driver che va a comunicare con il microcontrollore, e va impostata a $5V$. Il pin EN serve per abilitare l'output se impostato a livello logico alto ($5V$).
Ogni driver full-H può erogare al più $600mA$ continuativi. Per arrivare a $1200mA$ utilizzeremo i due driver in configurazione full-H collegati in parallelo.

3.4 Schema Circuitale

Lo schema del circuito finale è il seguente:

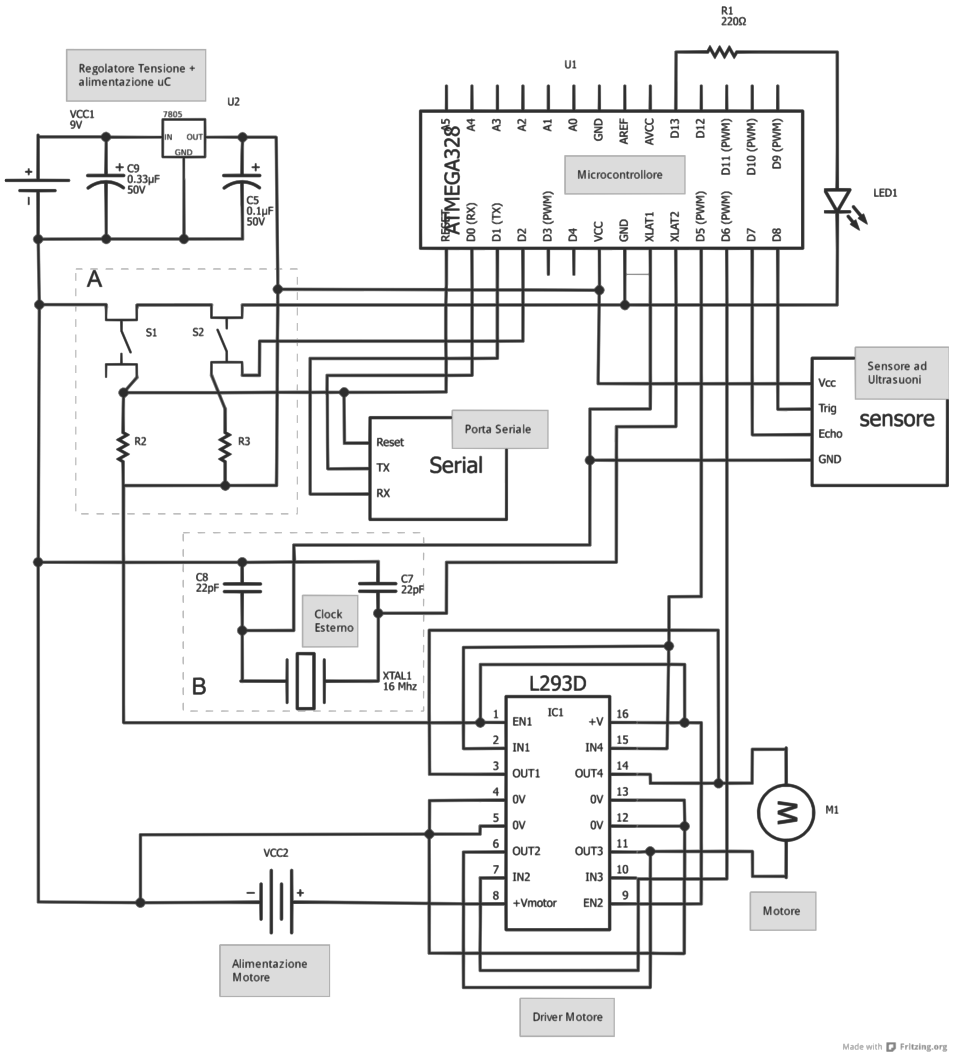

Figura 3.5: Schema del Circuito

Come è possibile vedere dalla figura, l'alimentazione del μC e della parte logica del Driver è affidata ad un regolatore di tensione $L7805$, che accetta in ingresso una tensione tra 7V e 35V e restituisce in uscita una tensione stabilizzata a 5V con una corrente massima di 1.5A, più che sufficiente per i nostri scopi.
Nel Blocco A vengono raffigurati i due pulsanti già menzionati nella sezione dedicata al Microcontrollore. Per garantire il loro corretto funzionamento abbiamo aggiunto una resistenza di PullUp a ciascuno di essi, in questo modo i piedini del controllore saranno sempre collegati a massa o a Vdd.
Nel Blocco B è presente l'oscillatore esterno e due capacità necessarie per il corretto funzionamento del microcontrollore a $16MHz$.
Riguardo l'L293D la configurazione è analoga a quella presentata precedentemente, con la differenza che abbiamo impostato due Driver full-H in parallelo per raddoppiare la potenza erogata.

Il sensore e la porta seriale sono stati schematizzati tramite due blocchi con relativi ingressi/uscite.
In figura 3.6 possiamo vedere una rappresentazione del circuito su breadboard.

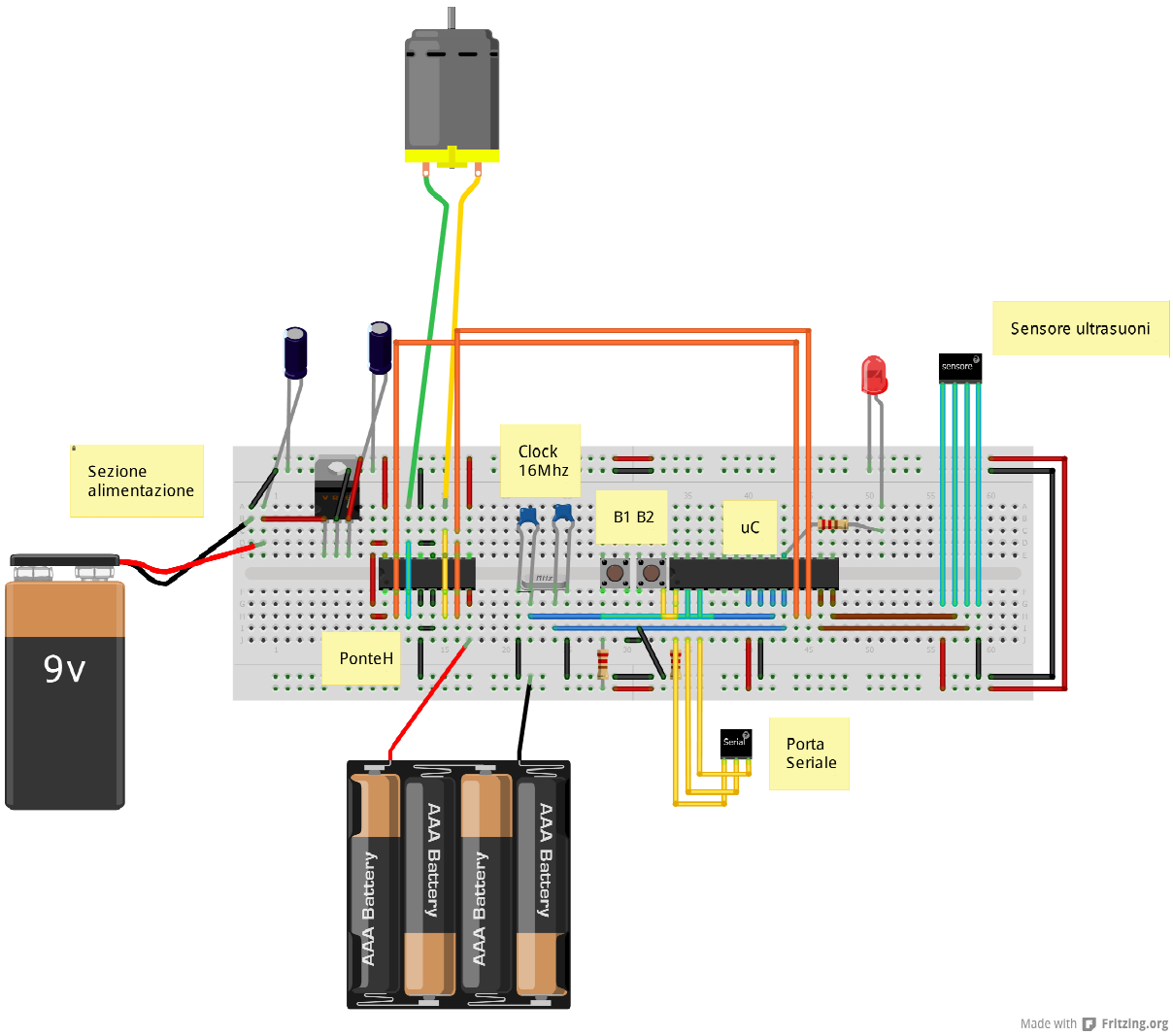

Figura 3.6: Rappresentazione su breadboard

La versione usata per i test è stata realizzata su una basetta mille-fori per prototipazione in modo da contenere il peso e sopratutto l'ingombro del circuito, come si vede nella figura seguente:

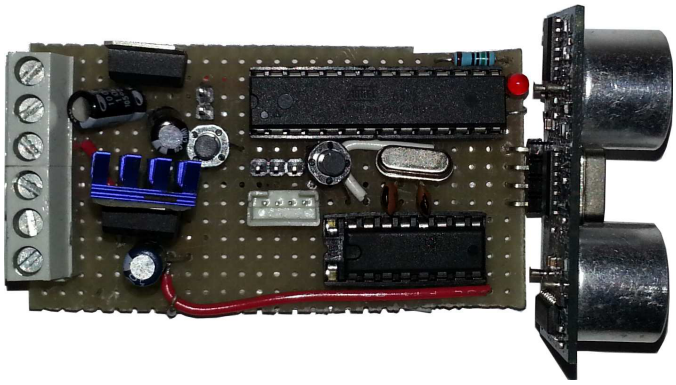

Figura 3.7: Circuito su basetta mille-fori

La scheda realizza il circuito mostrato in figura 3.5. Per connettere l'alimentazione (separata per motore e microcontrollore) e per collegare il motore a corrente continua sono stati montati dei morsetti (a sinistra in figura). Durante la realizzazione è stato aggiunto un secondo regolatore di tensione per il driver del motore, in questo modo volendo è possibile alimentare il motore con una tensione di 5 Volt utilizzando batterie che forniscono una tensione più alta (es: 12 Volt).
Eccola inserita nel modellino preso in esame:

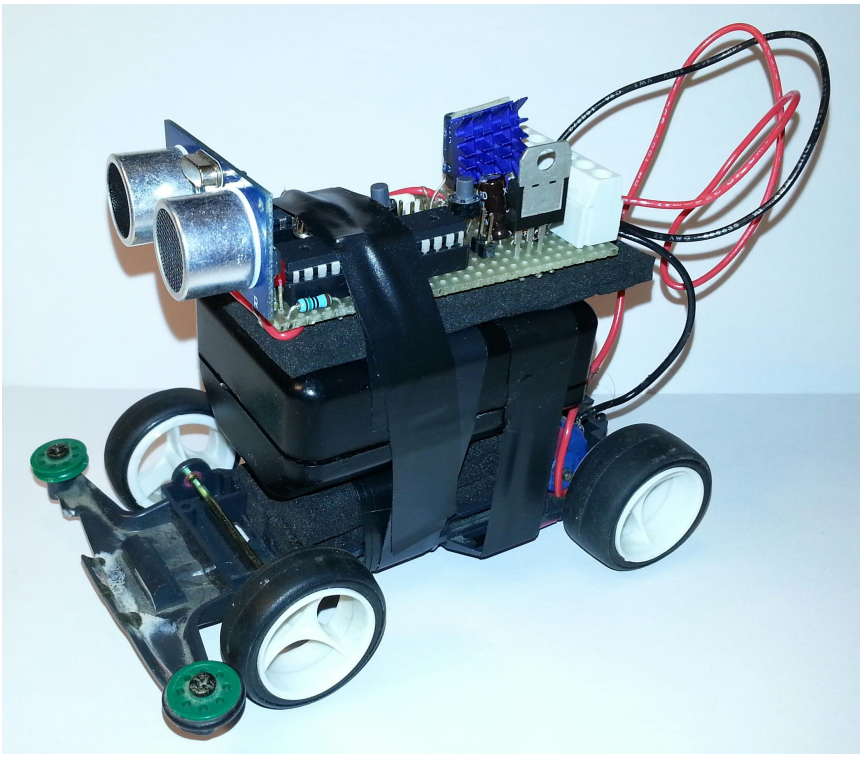

Figura 3.8: Sistema completo di controllore e veicolo

Vedremo nel prossimo capitolo come verrà implementato il controllore PID nel microcontrollore.

Capitolo 4

Software di controllo

4.1 Framework utilizzato

Uno dei motivi per cui si è scelto l'ATMega328 è che esso è supportato da Arduino.
Arduino è un Framework Open Source che permette di programmare molto rapidamente microcontrollori. Comprende una IDE (Integrated Development Environment) multipiattaforma scritta in Java, un linguaggio chiamato Wiring (derivato dal C) e sono disponibili molte librerie pronte all'uso.
Tramite gli strumenti messi a disposizione da questo Framework possiamo andare ad interagire con l'hardware con un buon grado di astrazione senza perdere troppo in flessibilità. Questo ci permette di ottenere un codice sorgente pulito ed immediato da interpretare rispetto a quanto ottenibile con il linguaggio C o con il linguaggio Assembly, mettendo meglio in evidenza gli aspetti che interessano questa tesi senza perdersi troppo nei dettagli implementativi.

Figura 4.1: Logo Arduino

4.2 Implementazione del controllo

Concettualmente il software di controllo è molto semplice. Lo possiamo schematizzare tramite un diagramma a blocchi in questo modo:

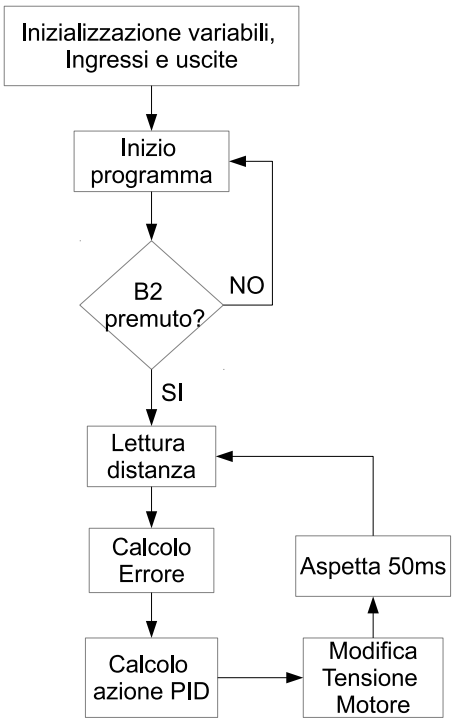

Figura 4.2: Sistema del software di controllo

Ogni blocco corrisponde ad una determinata parte del programma.
Il cuore del programma è rappresentato da un ciclo che viene eseguito ogni $50ms$, riportato di seguito:

```
void loop()
{
  time1 = micros();
  if(ready==true){
    dist = readDistance();
    error =  pref - dist;
    PIDComp();
    setVoltage();
    debuginfo();
  }else{
    if(digitalRead(b2) == HIGH){
      ready = true;
      digitalWrite(led,HIGH);
    }
```

```
  }
  time2 = micros();
  delayMicroseconds(50000-(time2-time1)); //aspetta 50ms
}
```

La funzione *micros()* restituisce il tempo trascorso in μs dall'inizio dell'esecuzione del programma. La utilizziamo per tener conto della durata (variabile e non trascurabile) delle operazioni interne al ciclo, questa durata viene poi sottratta ai 50ms in modo da far eseguire ogni ciclo ad intervalli regolari.
La variabile "ready" viene impostata a "true" quando il bottone B2 viene premuto, dopo di che rimane in quello stato e comincia l'azione del controllore PID.
Viene letta la distanza, calcolato l'errore, poi viene chiamata la funzione *PIDComp* che calcola l'azione del controllore PID in base all'errore ed infine viene chiamata la funzione *setVoltage()* che imposta la tensione da applicare al motore.
La funzione *debuginfo()* viene usata solo per stampare lo stato delle variabili che ci servono per fare le misurazioni. Avremmo potuto fare a meno di suddividere il programma in sottofunzioni ma questo ne avrebbe compromessa la leggibilità.

Vediamo la funzione *readDistance()*:

```
float readDistance(){
  ppdur = pdur;
  pdur = duration;
  digitalWrite(pingPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(pingPin, LOW);
//timeout equivalente a 4 metri
  duration = pulseIn(echoPin, HIGH, 23200);
  if(duration == 0){
    duration = pdur;
  }
  return (duration/5800);
}
```

Teniamo in memoria le due letture precedenti per poter stimare la velocità, poi come già accennato introducendo il sensore mandiamo un impulso della durata di $10\mu s$ al pin di *TRIG* del Sensore ed andiamo a leggere la sua risposta dal pin *ECHO*. Dividiamo la durata dell'impulso per 5800 al fine di convertirla in metri.

Vediamo adesso le funzioni *PIDComp()* e *setVoltage()*:

```
//calcola l'azione del controllore PID tenendo conto della saturazione del Driver
void PIDComp(){
tvt = vt;
  vt = (kp*error) + kd*derror() + ki*interror();
  if(vt < 0){vt = -vt;}
  if(vt > (255*tm)/5 ){vt =(255*tm)/5 ; }
  //se sta a meno di 5mm dal rif. il controllo PID si ferma
```

```
  if(vt < 30){vt = 0;}
}
//imposta la tensione da dare al motore
void setVoltage(){
  if(tvt <= 0 ){
    analogWrite(motoreindietro,0);
    analogWrite(motoreavanti, vt);
  }else{
    analogWrite(motoreavanti,0);
    analogWrite(motoreindietro,vt);
  }
}
```

La funzione *PIDComp()* calcola la tensione da applicare al motore in base al controllore PID. L'azione PID la andiamo a calcolare in questa riga di programma:

```
  vt = (kp*error) + kd*derror() + ki*interror();
```

Le funzioni *derror()* e *interror()* calcolano rispettivamente la derivata (nel senso ingegneristico) e l'integrale dell'errore.

```
  if(vt > (255*tm)/5 ){vt =(255*tm)/5 ; }
```

Questa riga di programma necessita di una breve spiegazione: la funzione *analogWrite(pin, value)* ci permette di generare un'onda quadra in uscita da un pin fissato con un duty cycle impostabile da un minimo di 0 (duty cycle 0%) a 255 (duty cycle 100%). La variabile tm ci permette di impostare il valore massimo del valore efficace della tensione. Con tm=3 ad esempio avremo un duty cycle massimo pari al 60%. Considerando che il driver è alimentato a 5Volt, otteniamo un tensione efficace di 3Volt. Possiamo quindi variare il duty cycle con una precisione di 8bit.
Con la funzione *setVoltage()* andiamo ad applicare il valore di tensione calcolato, tenendo conto del segno della tensione.

Il codice sorgende completo è riportato nell'Appendice A.

Capitolo 5

Identificazione del sistema, simulazioni e misurazioni

5.1 Identificazione

In questo capitolo andremo ad identificare i parametri incogniti del sistema, e vedremo anche attraverso delle misurazioni quanto il modello sviluppato nel primo capitolo si avvicini alla realtà.
Vediamo per prima cosa il modello Simulink. Mettiamo in ingresso al nostro sistema un gradino di tensione di 5Volt ed andiamo a stimare i due parametri incogniti osservando come esso reagisce. Riportiamo di seguito per comodità le formule (2.10) e (2.11) ricavate nel secondo capitolo che legano tensione e velocità rispettivamente nel dominio di Laplace e nel dominio nel Tempo:

$$sX(s) = \frac{K_m}{\tau_m s + 1}\frac{V}{s} \tag{5.1}$$

$$\dot{x}(t) = K_m V(1 - e^{-\frac{t}{\tau_m}}) \tag{5.2}$$

Il modello Simulink che rappresenta la risposta del nostro sistema è riportato di seguito. Nella funzione di trasferimento sono già stati inseriti i seguenti valori:

$$\tau_m = 2, \qquad K_m = 0.56$$

Questi valori sono stati stimati cercando una buona corrispondenza tra risposta al gradino del sistema reale e quello simulato, in seguito verificheremo la bontà del modello così costruito in svariate situazioni.

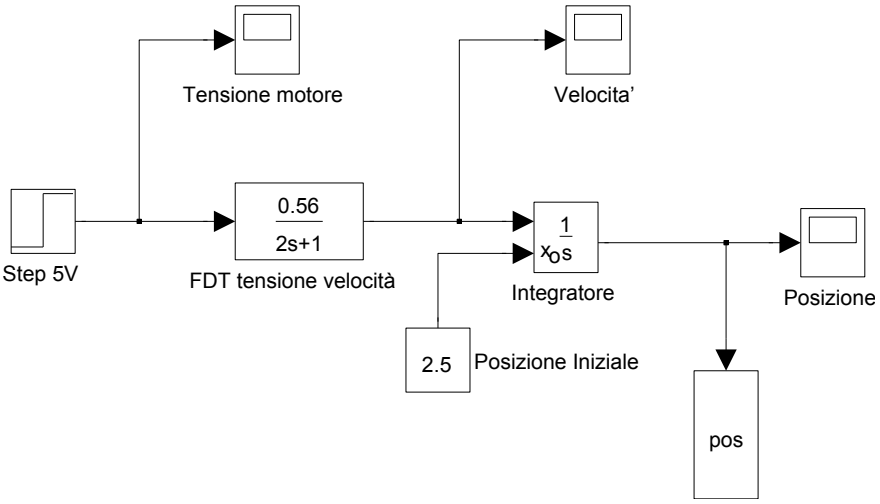

Figura 5.1: Modello della risposta al gradino

Vediamo ora il confronto tra l'andamento della velocità misurato e quello ricavato mediante Simulink:

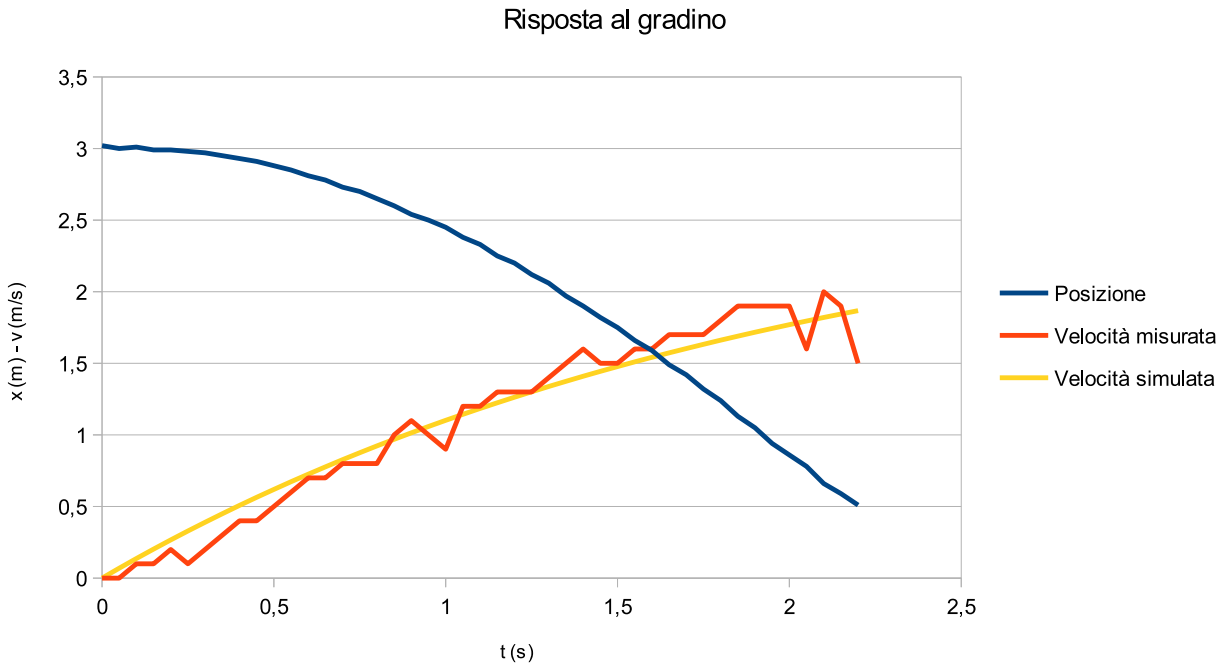

Figura 5.2: Misurazione della risposta al gradino

La velocità misurata presenta un discreto rumore, probabilmente dovuto sia alla natura economica dei componenti utilizzati sia al fatto che le misure sono state raccolte connettendo la macchina (molto piccola e leggera) ad un computer mediante un cavo che seppur leggero sicuramente ha disturbato il moto in modo non trascurabile.

5.2 Modello Simulink del sistema controllato

Il modello Simulink del sistema controllato mediante l'azione PID è il seguente:

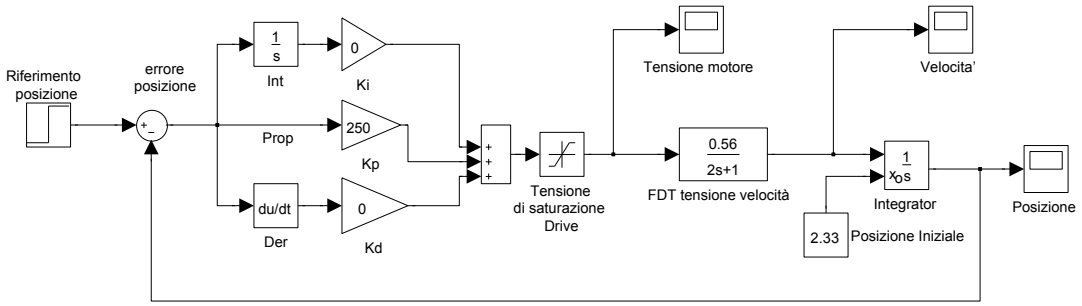

Figura 5.3: Modello del Simulink sistema controllato

In ingresso al controllore PID è presente il segnale di errore, ovvero la differenza tra il riferimento di posizione cercato e la posizione istantanea della macchina (entrambe tenendo l'ostacolo rilevato dal sensore come zero di riferimento).
Il riferimento di posizione cercato è definito nella variabile p_{ref} ed è un parametro molto importante perchè indica la distanza dall'ostacolo che la macchina cercherà di raggiungere per azione del controllore PID.
La Funzione di Trasferimento del controllore PID è la seguente:

$$C_{PID}(s) = K_p + K_d s + \frac{K_i}{s} \tag{5.3}$$

In uscita al controllore PID troviamo un blocco che serve a modellare la non-linearità del *Driver di Tensione* dovuta alla tensione di saturazione (in questo caso impostata a 5Volt), infine troviamo il blocco che rappresenta il nostro sistema da controllare.

Nella prossima sezione andremo a confrontare i risultati di alcune misurazioni sul campo con le corrispondenti controparti simulate.

5.3 Analisi dell'Azione Proporzionale

Vediamo ora come si comporta il nostro sistema soggetto ad una azione di controllo puramente proporzionale.
Questa misura è stata fatta con le seguenti impostazioni:

$$p_{ref} = 1m, \qquad K_p = 250, \qquad K_d = 0, \qquad K_i = 0$$

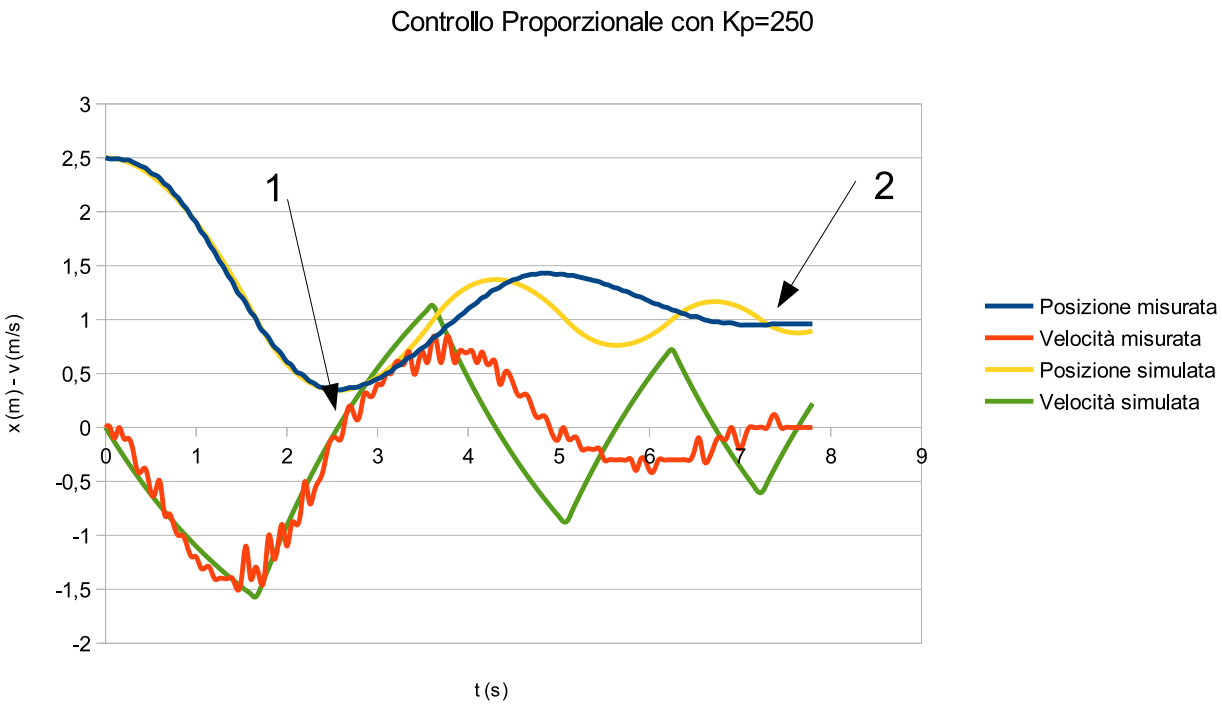

Figura 5.4: Controllo di tipo Proporzionale

Dal grafico si nota un'ottima corrispondenza tra il comportamento simulato e quello reale fino al punto in cui la velocità si annulla la prima volta (freccia 1), da quel punto in poi è presente una certa discrepanza. Si nota inoltre che il sistema reale si assesta ben prima di quello simulato (freccia 2). Dalle misurazioni (qui non riportate) si osserva che la nostra macchina si assesta intorno a $7.35s$ ad una distanza di $0.96m$, commettendo quindi un errore di $0.04m$ rispetto al riferimento di $1m$.
Questo comportamento apparentemente anomalo è in larga parte dovuto al fatto che il coefficiente di attrito non è indipendente dalla velocità come nel modello Simulink ma aumenta mano a mano che la velocità diminuisce. La situazione è ulteriormente peggiorata nel momento in cui la velocità si annulla, in tal caso entra in gioco una componente di attrito di tipo statico non modellata che si oppone alla ripartenza della macchina.
Si potrebbe tener conto di questi aspetti in ulteriori analisi più approfondite.

5.4 Analisi dell'Azione Proporzionale-Derivativa

Vediamo ora come si comporta il nostro sistema soggetto ad un'azione di controllo di tipo proporzionale-derivativo. Questa misura è stata fatta con le seguenti impostazioni:

$$p_{ref} = 0.8m, \qquad K_p = 700, \qquad K_d = 100, \qquad K_i = 0$$

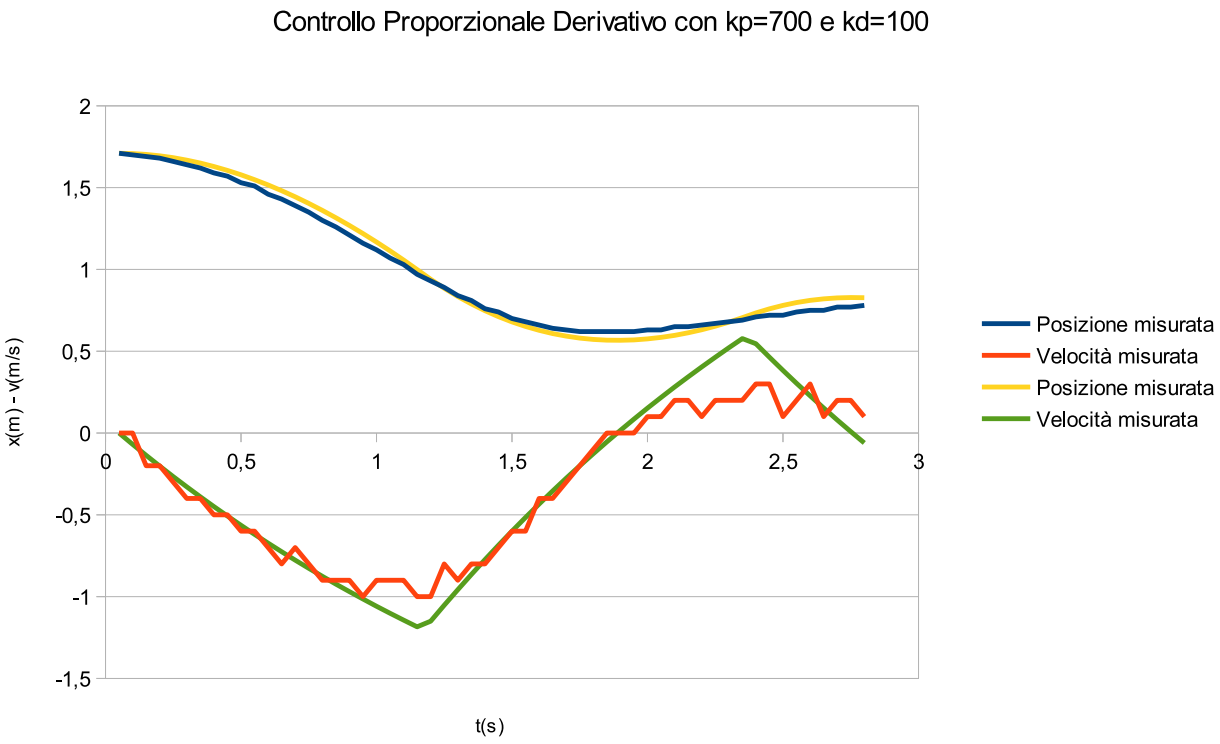

Figura 5.5: Controllo di tipo PD, prima misura

Si può notare una corrispondenza molto buona tra l'andamento simulato e quello reale. C'è ancora della sovraelongazione dovuta ad un'azione derivativa troppo limitata che costringe la macchina a fermarsi ed a tornare indietro.

Proviamo ad effettuare un'altra misura con le seguenti impostazioni:

$$p_{ref} = 0.8m, \qquad K_p = 700, \qquad K_d = 300, \qquad K_i = 0$$

Avendo lasciato invariata l'azione proporzionale e triplicato quella derivativa, vediamo come si comporta il sistema:

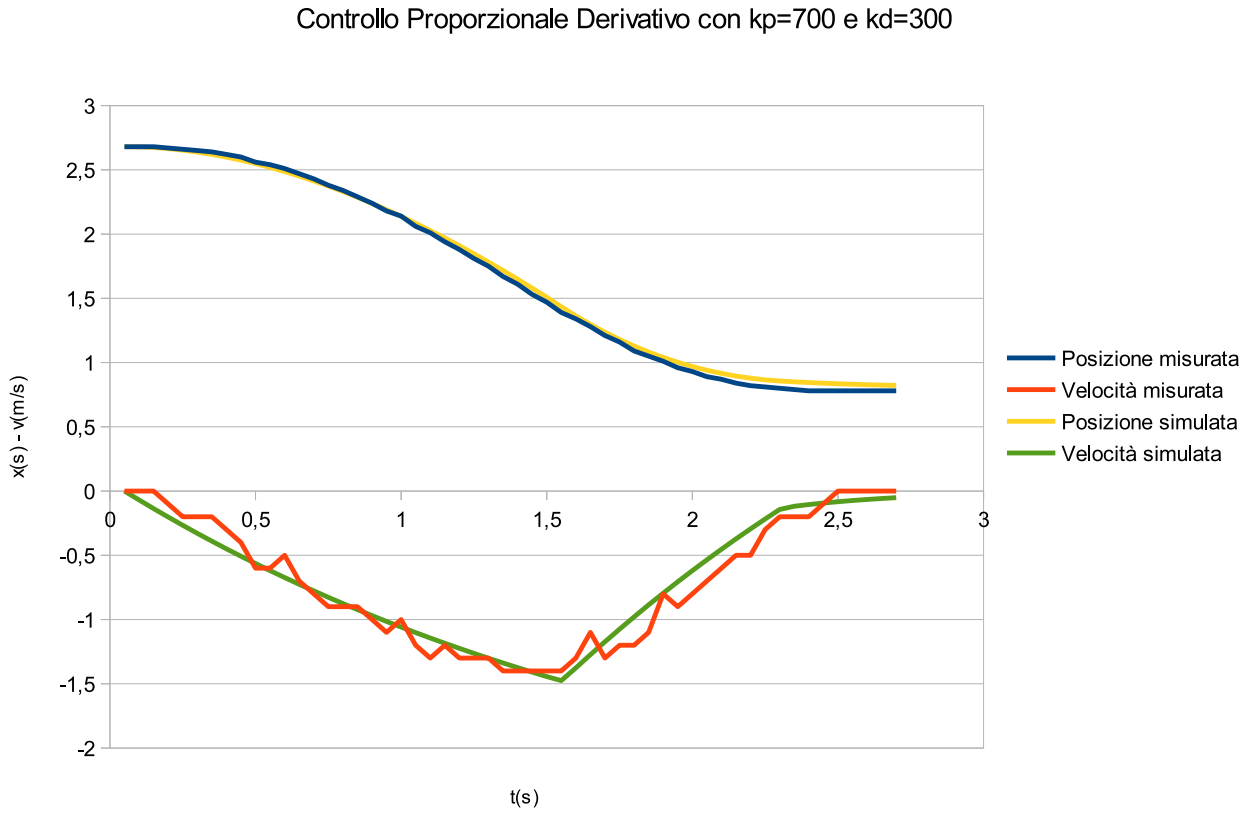

Figura 5.6: Controllo di tipo PD, seconda misura

In tal caso la corrispondenza tra l'andamento simulato e quello reale è ancora maggiore, segno che le approssimazioni fatte non creano problemi. Dalle misurazioni si nota che l'assestamento avviene dopo circa $2.75s$ e ad una distanza di $0.79m$, commettendo un errore di $1cm$ rispetto al riferimento.

Le misure fatte fino a questo punto di sono rivelate soddisfacenti, purtroppo abilitando anche l'azione integrativa il sistema non risponde molto bene, probabilmente per il fatto che il movimento è relativamente lento e viene integrato un errore molto grande che rende la componente integrativa dominante rispetto alle altre due.

Sarebbe stato interessante effettuare un'ultima misura abilitando l'azione integrativa solo in prossimità del riferimento ed aggiungendo una forza di disturbo al sistema. Per esempio facendo muovere la macchina in salita su di un piano inclinato con pendenza fissata sarebbe stato possibile sfruttare la forza di gravità per aggiungere una componente di questo tipo. In questo modo si sarebbe potuto osservare come l'azione integrativa riesca a compensare questo tipo di disturbo nei casi in cui l'azione di tipo PD risulti troppo debole (vicinanza al riferimento, velocità prossima a zero). Purtroppo è risultato abbastanza laborioso riuscire ad effettuare una misura di questo tipo.

Capitolo 6

Conclusioni

Implementare un controllore di tipo PID si è rivelato un lavoro complicato e non privo di imprevisti (come capita spesso quando si affronta un problema per la prima volta), e si è dovuto accettare qualche compromesso per contenere la mole di lavoro: è stato rimosso l'*Encoder* che sarebbe servito a misurare con precisione la rotazione delle ruote e le misurazioni sono state effettuate tramite un cavo tra automobile e computer e non senza fili come previsto inizialmente (inizialmente era stato utilizzato un modulo *Bluetooth LMX9838* che funzionava ma si è danneggiato nel passaggio da breadboard a basetta mille-fori e non è stato possibile reperirne un altro in tempo), questo avrebbe permesso l'acquisizione di misure più precise.

Tuttavia il prodotto finale è riuscito a confermare la potenza e la versatilità di questo tipo di controllo attraverso prove sul campo. Siamo riusciti ad approssimare il nostro sistema con un modello del primo ordine e ad identificarlo ottenendo una buona corrispondenza tra il modello Simulink ed il comportamento reale.

In presenza di maggior tempo a disposizione sarebbe stato interessante analizzare il sistema più da vicino in modo da migliorare l'accuratezza del modello Simulink anche nei casi meno fortunati, come quello visto nella sezione 5.3.

Appendice A

Codice sorgente Arduino

```
//assegno ad ogni pin di I/O un nome
int led = 12;
const int pingPin = 7;
const int echoPin = 8;
const int encoder = 3;
const int motoreavanti = 5;
const int motoreindietro = 6;
const int b2 = 2;

//imposto i parametri
float pref = 1;
float kp=600;
float kd=250;
float ki=0;
boolean dbinfo=true;

//inizializzo le varibili necessarie al programma
float duration=0, pdur=0, ppdur=0;
float error=0, ierror=0;
float dist=0;
float vt=0, tvt=0;
float tm=5;
int time=0;
boolean ready = false;
unsigned long time1, time2;

//imposto i pin del microcontrollore
void setup()
{
  Serial.begin(9600);
  pinMode(led, OUTPUT);
  pinMode(b2,INPUT);
  pinMode(pingPin, OUTPUT);
```

```
  pinMode(echoPin, INPUT);
  pinMode(encoder, INPUT);
}

//ciclo principale
void loop()
{
  time1 = micros();
  if(ready==true){
    dist = readDistance();
    error =  pref - dist;
    PIDComp();
    setVoltage();
    debuginfo();
  }else{
    if(digitalRead(b2) == HIGH){
      ready = true;
      digitalWrite(led,HIGH);
    }
  }
  time2 = micros();
  delayMicroseconds(50000-(time2-time1)); //aspetta 50ms
}

//calcola derivata dell'errore
float derror(){
  return ((ppdur-duration)/5800)/(0,1); // in metri sec
}

//calcola l'integrale dell'errore
float interror(){
  ierror=ierror + (error/5800);
  return ierror;
}


//calcola l'azione del controllo PID
//tenendo conto della saturazione del Driver
void PIDComp(){
  vt = (kp*error) + kd*derror() + ki*interror();
  tvt = vt;
  if(vt < 0){vt = -vt;}
  if(vt > (255*tm)/5 ){vt =(255*tm)/5 ; }
  if(vt < 30){vt = 0;}
}
```

```
//imposta la tensione da dare al motore
void setVoltage(){
  if(tvt <= 0 ){
    analogWrite(motoreindietro,0);
    analogWrite(motoreavanti, vt);
  }else{
    analogWrite(motoreavanti,0);
    analogWrite(motoreindietro,vt);
  }
}

//lettura della distanza
float readDistance(){
  ppdur = pdur;
  pdur = duration;
  digitalWrite(pingPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(pingPin, LOW);
//timeout equivalente a 4 metri
  duration = pulseIn(echoPin, HIGH, 23200);
  if(duration == 0){
    duration = pdur;
  }
  return (duration/5800);
}

//stampa lo stato delle variabili di interesse
void debuginfo(){
  if(dbinfo==true){
    Serial.print("time ");
    Serial.print(time);
    Serial.print(" dist ");
    Serial.print(dist);
    Serial.print(" error ");
    Serial.print(error);
    Serial.print("vt ");
    Serial.println(vt);
    time=time+1;
  }
}
```

Ringraziamenti

Per concludere questa tesi vorrei ringraziare tutte le persone che ho incontrato in questi tre anni a Padova e gli amici che mi hanno accompagnato in questo percorso dandomi il loro sostegno.

Vorrei dire grazie anche a tutti i professori che mi hanno trasmesso la passione per le materie trattate nei loro corsi, in particolare il Prof.Mauro Bisiacco che tramite i corsi da lui tenuti mi ha facilitato nella scelta della laurea specialistica e motivato nel proseguire gli studi in questo settore.

Infine un grazie di cuore alla mia famiglia che continua a sostenermi nel corso degli anni.

Bibliografia

[1] M. Bisiacco e M. E. Valcher, Controlli automatici, Edizioni Libreria Progetto Padova, 2008.

[2] P. Mazzoldi, M. Nigro e C. Voci, Fisica, Volume 1, EdiSES s.r.l., 1991.

[3] MATLAB Documentation R2009a