Università degli Studi di Padova

Facoltà di Ingegneria

Dipartimento di Tecnica e Gestione dei Sistemi Industriali

Corso di laurea triennale in Ingegneria

Meccanica e Meccatronica

Curriculum Meccatronico

Tesi di Laurea

“ Regolatori e Sistemi di Controllo per una Smart Car ”

Laureando:

Stella Alessandro
1001918 – IMM

Relatore:

Ch.mo Prof.
Roberto Oboe

Anno Accademico: 2012-2013

Sommario

Con il seguente elaborato si vuole cercare di dare una breve, quanto approfondita analisi di tutti i sistemi di controllo e di regolazione adottati nella competizione "*Freescale Cup*". Saranno fornite dapprima delle brevi nozioni generali sulla competizione (regolamento, obiettivi, problematiche riscontrate durante tutta la fase di preparazione) e successivamente si passerà alla descrizione, nonché alla motivazione, delle scelte fatte nell'ambito del controllo del veicolo, soffermandosi soprattutto sulla parte *software*.
Questa iniziativa intrapresa dall'Università di Padova sotto la guida del professor Roberto Oboe ha permesso a noi studenti del secondo/terzo anno di Ingegneria Meccatronica di "scontrarci" con problematiche vere e reali nel mondo dell'ingegneria. Infatti, grazie a questa competizione, abbiamo potuto capire come applicare, almeno in parte, la teoria dei controlli e della programmazione che ci è stata insegnata durante il nostro corso di laurea triennale. Un'esperienza difficile ma al contempo unica che ci ha messo a dura prova sia dal punto di vista dello studio che da quello del lavoro di *team*; fondamentale è stato l'aiuto dei professori che hanno saputo indirizzarci con rara professionalità e indiscussa competenza verso le soluzioni più semplici ed efficaci.
Sicuramente questa esperienza va consigliata a chiunque voglia misurarsi con i problemi veri e propri di un futuro Ingegnere Meccatronico.

Indice

Capitolo 1 - Introduzione

Capitolo 2 – Definizioni Necessarie

Capitolo 3 – Parti del Veicolo che ne Condizionano il Controllo

Capitolo 4 – Schema a Blocchi

Capitolo 5 – Implementazione Software

Capitolo 1

Introduzione

1.1 Freescale Cup

Una competizione che coinvolge e mette a dura prova studenti d'Ingegneria di tutto il mondo. Piccole macchine comandate da un microcontrollore e niente più, autonome in tutte le funzioni, "vetture" in grado di seguire qualsiasi tipo di tracciato di reagire a diversi stimoli; tutto in modo totalmente automatico e senza l'intervento di un utente esterno. Questa è stata la proposta lanciata dall'Università di Padova a noi studenti di Ingegneria Meccatronica. Tutto avrebbe dovuto iniziare ai primi di settembre poi a febbraio ci sarebbe stata una competizione interna tra tre gruppi dell'università; per il vincitore la "gloria" e la possibilità di andare a Parigi per confrontarsi con le altre squadre europee, con la speranza di arrivare ai mondiali negli "*States*".
A settembre arrivano i *kit* dall'azienda promotrice (la: *Freescale Semiconductors*) composto da due motori in continua, un ponte H, un servomotore, una telecamera ed il telaio per la vettura. I tre team si mettono subito all'opera, nove teste diverse, nove competenze diverse, tre scelte competitive diverse: uno solo potrà arrivare alla vittoria.
Dopo sette mesi di duro lavoro si tiene la competizione nel laboratorio didattico di meccatronica del Dipartimento di Tecnica e Gestione dei Sistemi Industriali di Vicenza come giudici: il professor Roberto Oboe, il tutto fare Roberto Losco e i dottorandi (anche loro "gareggianti" con un veicolo innovativo).
Dopo una sfida ai millesimi di secondo ecco che vince il team dei "*Crash*".
È stata una competizione difficile fatta soprattutto di scelte diverse e di duro confronto per ciascun membro dei diversi team; ognuno ha imparato qualcosa di nuovo e di utile da questa gara, sicuramente un'esperienza che ci ha lasciato molto sia per quanto riguarda le nuove competenze che abbiamo potuto acquisire, sia riguardo la possibilità di lavorare in gruppo con persone di pari capacità.

A. **L' Azienda Promotrice**

Figura 1.1 logo dell'azienda *Freescale Semiconductors*

L'azienda *Freescale Semiconductors* è stata una delle prime e principali aziende che si è affacciata al mercato dei semiconduttori. Inizialmente nata come una divisione della Motorola ha contribuito attivamente, mediante la fabbricazione di migliaia di dispositivi a semiconduttore e di tracciamento a terra, allo sbarco sulla Luna dell'Apollo 11.
Nel 2003, Motorola ha annunciato che la loro divisione specializzata in semiconduttori si sarebbe divisa per creare la *Freescale Semiconductors*.

Nel 2011, dopo innumerevoli innovazioni e conquiste sul piano tecnologico, l'azienda riesce a lanciare la prima famiglia polifunzionale di processori a *base wireless*, che integra *DSP* e processori adatti alla comunicazione, atti a realizzare una vera e propria "*base station on chip*". Nello stesso anno la *Freescale* assieme alla *McLaren Electronic System* ha convertito alcuni veicoli *Nascar* da automobili a carburazione a vetture ad iniezione.
Una recente ricerca di mercato fatta da *ABI* ha rilevato che oltre il 60% delle frequenze radio appartiene a dispositivi realizzati con i semiconduttori prodotti dall'azienda.
Il pregio della *Freescale*, tuttavia, non è solo quello di privilegiare la ricerca e lo sviluppo di nuove tecnologie ma è anche quello di proporre alle università di tutto il mondo esperienze formative come quella della "*Freescale Cup*". Iniziativa quest'ultima che ha avuto inizio sin dal 2003 coinvolgendo da subito le principali nazioni mondiali tra le quali: Cina, India, Malesia, Europa, America Latina e Nord America.

B. **La Competizione**

La prima edizione della competizione si svolse nel 2003 presso la "*Hanyang University*" in Corea coinvolgendo oltre ottanta gruppi di studenti.
Anno dopo anno, questa gara ha acquisito sempre più importanza, arrivando a coinvolgere fino a 500 università e oltre 1500 studenti.
Ogni gruppo di studenti scelto dalle singole università deve preparare delle mini vetture (*Smart Car*) in grado di seguire, in modo completamente autonomo, un tracciato formato da una linea guida di colore nero e da uno sfondo bianco; la vettura più veloce vince.
I vari *team* universitari possono modificare il *kit* fornito dall'azienda aggiungendo sensori ed altro *hardware* atto a migliorare la velocità e la stabilità del veicolo.
Le difficoltà per questi studenti sono sicuramente molte: prima fra tutte, sicuramente, il fatto di doversi interfacciare all'*hardware* tramite linguaggio C e un *software* d'interfaccia poco amichevole quale *Code Warrior*; è inoltre sicuramente necessario da parte degli studenti avere buone conoscenze di strategie di controllo e di dinamica dei veicoli, nonché la padronanza delle tecniche di modulazione in Pwm.

C. **Il Regolamento**

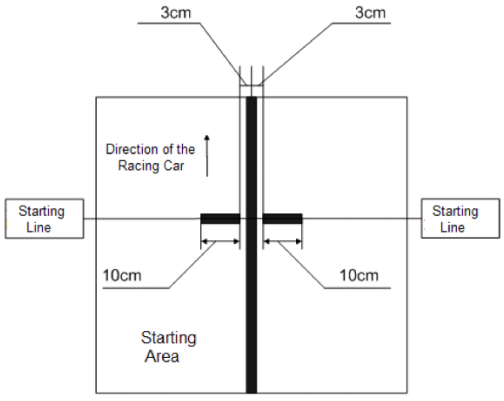


Figura 1.2 Dimensioni della linea nera

Vediamo ora i punti principali del regolamento della competizione:

Tracciato:

Come si può vedere dalla Fig. 1.2 il tracciato è composto da una linea di colore nero dello spessore di 6 cm posta su uno sfondo bianco per facilitare l'acquisizione del percorso da parte della " *Smart Car*".
La linea d'inizio e fine ha una conformazione particolare. Infatti è più larga di quella normale ed ha una sequenza ben precisa di alternanza di bianco e nero.

Modifiche *hardware* alla vettura:

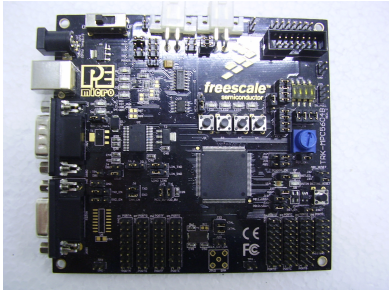


Figura 1.3 Il cuore del veicolo: il microcontrollore TRK-MCB5604B

Dev'essere utilizzato solo ed esclusivamente l'*hardware* fornito con il *kit*, eccezion fatta per i sensori che possono essere aggiunti fino ad un limite di sedici.
Si può inoltre utilizzare solo la batteria fornita nel *kit* come fonte di alimentazione ed è vietato usare un altro processore all'infuori del MCB5604B anch'esso fornito con l'equipaggiamento base.

Modifiche allo "chassis":

Figura 1.4 lo chassis del veicolo

La conformazione base dell'auto non può essere modificata;
Non si può modificare la distanza tra le ruote;
Nessuna parte della macchina deve eccedere i 250 [*mm*] di larghezza e i 400 [*mm*] di lunghezza;
Sono ammessi supporti e fori ausiliari.

Insuccesso della gara:

La macchina deve lasciare la " *Starting Area*" (vedi Fig.1.2) entro trenta secondi dal via;
La vettura deve fermarsi entro 1000 [*mm*] dopo l'attraversamento della linea di stop;
Durante la corsa non deve uscire dallo sfondo bianco della pista con più di tre ruote e deve concludere il suo turno in meno di 120 secondi.

Squalifica del team dalla competizione:

Attorno al tracciato non devono essere presenti circuiti di illuminazione o sensoristica ausiliaria.
Le auto una volta ispezionate non possono essere soggette a modifica *hardware* o *software*.

1.2 Problema Considerato

La seguente tesi di laurea, come da titolo, si prefigge di analizzare in modo accurato la struttura e l'implementazione *software* dei sistemi di controllo adottati nella progettazione della *Smart Car* per la competizione *Freescale Cup*.
Si prenderanno dapprima in esame le principali definizioni utili a comprendere il duro lavoro svolto dai *team* e la complessità della competizione. A riguardo, infatti, verranno date le definizioni di Pid, *Anti Reset Wind Up* e Pwm calandoli nel caso specifico.
Verranno poi esaminate le principali sorgenti di *input* e di *output* presenti nel dispositivo, dandone una definizione teorica ed analizzando in modo approfondito i *data sheet*, ove disponibili.
Tutta questa parte è racchiusa nel capitolo 2, dove è inoltre presente una piccola descrizione del *software* utilizzato per l'implementazione dell'algoritmo di controllo ed un breve richiamo delle principali funzioni del linguaggio C, fondamentali per capire le parti di codice introdotte.
Fondamentale per analizzare le strategie di controllo da adottare su veicoli di questo tipo, è capire la componentistica in dotazione, mettendone in evidenza i limiti dei sensori e gli eventuali disturbi esterni che potrebbero compromettere la stabilità del sistema.
Verrà inoltre eseguita un'analisi del sistema mediante schema a blocchi per facilitare la comprensione e rendere l'elaborato scorrevole da un punto di vista logico.
Nell'ultimo capitolo verrà presentata una parte del codice per l'implementazione delle tecniche di controllo utilizzato dal *team* in particolare sarà analizzato l'algoritmo, evidenziandone pregi e difetti.
Essendo l'ambito del controllo una delle principali materie che un buon ingegnere meccatronico deve saper padroneggiare, ho ritenuto interessante studiare questo aspetto nell'ambito della possibilità offertaci dall'Università di Padova. Durante la competizione, infatti, ci sono mancate molte nozioni base necessarie per la piena comprensione e realizzazione del lavoro da svolgere, ma grazie alla presenza dei professori, molto disponibili e capaci, siamo riusciti a colmare queste lacune e ad entrare a pieno nella materia; capendo aspetti che non erano ben chiari nella parte di teoria. Ho quindi scelto l'argomento del controllo sia per una curiosità personale, per la voglia di approfondire tale argomento, sia per legare insieme gli aspetti teorici dei vari corsi di controllo e gli aspetti pratici di questa esperienza.

1.3 Struttura dell' Elaborato

Si tratterà ora la suddivisione dell'elaborato per capitoli riassumendo il contenuto di ogni sezione del testo in modo da permettere al lettore di farsi un'idea generale dell'argomento trattato. Ogni capitolo sarà provvisto di immagini ed eventuali riferimenti a testi specialistici per rendere la lettura più fluida e comprensibile.

Capitolo 2:

Il capitolo, come già detto nell'introduzione, dà e chiarisce le definizioni principali necessarie per capire l'importanza del problema del controllo nella competizione.
Si parte innanzitutto dal descrivere il *software* fornitoci dalla *Freescale Semiconductors*: "*Code Warrior*"; analizzandone i limiti, i pregi e la tipologia di linguaggio utilizzato.
Successivamente saranno definite le principali sintassi C adoperate, così da chiarificarne le modalità di utilizzo.
Vengono poi richiamati i concetti base e le principali tipologie di controllori che sono state utilizzate, motivandone la scelta. In particolare verrà descritto il Pid ed il motivo dell'aggiunta di un *Anti Reset Wind Up* nonché si analizzerà l'importanza della tecnica di controllo in Pwm.
Come ultima parte del capitolo verrà analizzata la componentistica *hardware* fornitaci con il *kit* e le modifiche che vi abbiamo apportato.
In questa sezione si prenderà quindi spunto dai *data sheet* disponibili, per avere chiaro il funzionamento del microcontrollore e dei convertitori analogico-digitale.

Capitolo 3:

Un sistema di controllo è influenzato da innumerevoli parametri: disturbi ed errori dei sensori sono argomenti che vanno tenuti in considerazione nella progettazione dei regolatori.
Proprio per questo motivo si è deciso di dedicare un intero capitolo all'analisi dei sensori, degli attuatori e dei processori presenti nel veicolo.
Questa sezione infatti cercherà di analizzare il funzionamento delle principali fonti di *input* e di *output* del controllore; per progettare il controllore infatti è strettamente necessario avere chiaro come i sensori processino i dati provenienti dall'esterno e come gli attuatori riescano ad elaborare gli ordini forniti dal processore.
È scopo di questo capitolo, inoltre, far capire le modifiche *hardware* necessarie che sono state fatte, concentrandosi soprattutto sulle cause che hanno portato a tali scelte.

Capitolo 4:

Questo ed il successivo capitolo rappresentano il cuore dell'elaborato. In questa parte del lavoro viene, infatti, ricondotto tutto lo schema dei controllori, assieme alle nozioni fornite nei capitoli precedenti a singole parti di uno schema a blocchi che permette di seguire passo passo la stesura dei principali controllori evidenziandone *input*, *output*, catene di retroazione e gestione dei limiti dei controllori.
Verranno da subito individuate e schematizzate le sorgenti di *input* di cui si sottolineeranno le caratteristiche riguardanti il sistema di controllo; verranno poi analizzati i metodi di attuazione delle direttive del controllore, concentrandosi sui dispositivi interessati da tale azione; in seguito verranno analizzate le fonti di errore soffermandosi sulle cause che le hanno generate e sulla risposta a tali errori da parte del sistema di controllo.
Nella parte restante del capitolo verranno inoltre prese in considerazione le catene di *feedback*, la struttura e i processi interessati dai vari controllori.
L'intento sarà quello di dare una chiara visione dei sistemi di controllo adottati capendone la struttura dei singoli componenti.

Capitolo 5:

Rappresenta il capitolo conclusivo e forse quello che tratta più da un punto di vista pratico il lavoro svolto.
In questo capitolo infatti, utilizzando tutte le informazione raggiunte dai capitoli precedenti, si presenteranno le problematiche dell'implementazione *software* dei vari controllori.
Si analizzerà quindi la parte di controllistica del codice stilato durante la competizione, analizzandone pregi e difetti, verificando che segua quanto supposto nella teoria.
L'obiettivo non sarà tanto quello di capire come sia stato utilizzato il linguaggio C, quanto dare una chiara visione di come sia difficile implementare nella pratica tutte le ipotesi fatte basandosi sulla teoria studiata e appresa durante il corso delle lezioni e di come si renda a volte necessario adottare delle approssimazioni per adattare al mondo reale il mondo teorico.

Capitolo 2

Definizioni Necessarie:

2.1 Software

A. **Code Warrior:**

Per programmare il microcontrollore della *Smart Car* siamo stati dotati di un *software* chiamato *Code Warrior* in grado di tradurre algoritmi scritti con il linguaggio C in istruzioni macchina; ci è stato dato inoltre un codice guida contenente l'implementazione dei principali *pin* e alcuni esempi didattici per capire il funzionamento del *software* nell'ambito della competizione.
Vediamo le principali funzioni del programma:
Una volta aperto il file di esempio ci si trova difronte ad una schermata di questo tipo:

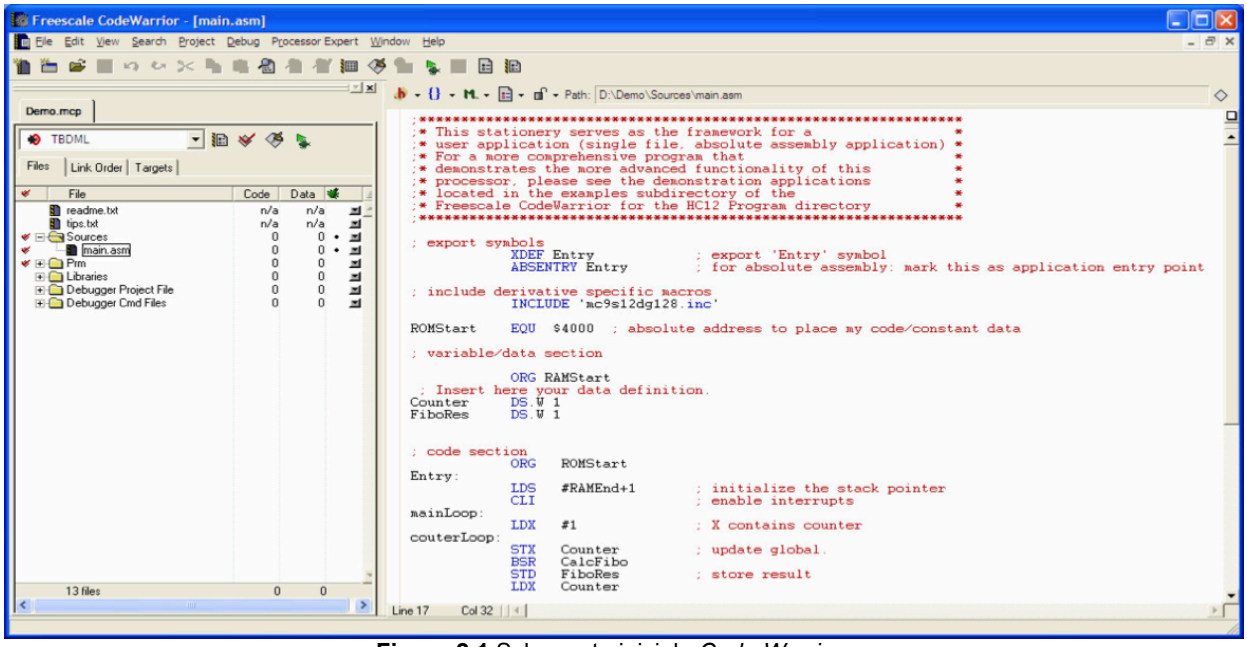


Figura 2.1 Schermata iniziale *Code Warrior*

Si nota a sinistra una colonna contenete la struttura del *software* che stiamo implementando, possiamo infatti utilizzare questa finestra per raggiungere tutti i file *header* e per identificare ed organizzare il nostro progetto in cartelle.
La colonna di destra contiene il *software* vero e proprio; si può utilizzare questa sezione per scrivere e sviluppare le varie funzioni del programma e per stendere il corpo del *main*.
Per compilare e fare eseguire il programma bisogna andare sul menu in alto e selezionare *Project->Compile*, oppure premere *ctrl+F7* poi bisogna *debuggare* l'algoritmo creato mediante la pressione del tasto *F5* della tastiera o alternativamente dal menu in alto selezionando il tasto *debug*.
Ora Code Warrior ci dà due possibilità:
La prima è quella di scaricare il programma direttamente sul microcontrollore ed avviarlo *"on board"*, altrimenti possiamo avviare il programma direttamente da *computer*.
Il secondo metodo ha il pregio di permetterci di leggere in *real time* il valore delle singole variabili del programma; tale procedimento è molto utile in fase di progettazione per capire se il *software* da noi creato si comporta nel modo desiderato.

Tuttavia con questa modalità risulta logisticamente difficile far eseguire l'intero algoritmo al veicolo in quanto occorrerebbe rincorrerlo con il computer in mano.
Per selezionare queste due modalità basterà modificare dal menu a tendina a sinistra passando da "*internal flash*" per la prima modalità a " *Ram*" per la seconda.

B. **Linguaggio C:**

Nato nel 1972 da Dennis Ritchie, il linguaggio C è quello che è stato utilizzato per la programmazione tramite *Code Warrior*.
Vediamo ora in sintesi le principali sintassi C utili per capire l'algoritmo del programma.
Fondamentale per permettere al *software* di eseguire delle istruzioni per un determinato numero di cicli è l'istruzione "*for*" che viene implementata come segue:

" *int i; // prima la variabile di ciclo va inizializzata*
for(<inizializzazione variabile> (es i=0);<istruzione di continuazione del ciclo> (es. i <=10);<istruzione di avanzamento del ciclo>(es i++)) // con gli esempi citati il // programma farà undici cicli
{
//istruzione da eseguire per undici cicli;
}**"**

Notiamo che per commentare il codice è sufficiente inserire il carattere "//" oppure "/*" terminando però, in questo caso, il commento con il carattere "*/".
Alta importanza ha anche la dichiarazione delle funzioni, in C questa procedura risulta molto semplice infatti basta usare una sintassi di questo tipo:

" *tipo di output che la funzione ci restituisce* (<definizione degli input da dare alla funzione >(es. int i, int a, double b, char x)
{
//istruzioni che caratterizzano la funzione;
return <output che vogliamo che la funzione ci restituisca>; // se il tipo di funzione fosse stato // void non avrei dovuto metterlo
}**"**

Le tipologie di *output* usate principalmente nel nostro caso sono state: *void, int, double*.
Altro aspetto fondamentale della programmazione in C sono le istruzioni di "*go to*" e di "*break"* la prima è molto utile per saltare ad una specifica parte del programma, basterà infatti scrivere:

"
//parte precedente del programma

go to label ; // dove al posto di label si può mettere un nome rappresentativo della parte di // algoritmo dove vogliamo andare.

Label: // continuazione di programma **"**

il secondo è invece necessario nel caso si voglia far fermare completamente un ciclo per una qualsiasi motivazione, si scriverà quindi :
"*for()*
{
if(condizione)
break;
} **"**

Si riporta infine una tabella contenente le principali simbololgie logiche utili nei controlli "*if*" e nelle istruzioni logiche:

Nome	Simbolo	Descrizione (Sintassi)
Somma	+	A+b;
Sottrazione	-	a-b;
Moltiplicazione	*	A*b;
Divisione	/	a/B;
Negazione	!	!A (esegue il not bit a bit)
Inizializzazione	=	i=1 assegna ad "i" valore 1
Disuguaglianza	!=	A!=B (" se a è diverso da b")
Maggiore	>	a>b (" se a è maggiore di b")
Minore	<	A<b(" se a è minore di b")
Maggiore Uguale	>=	a>=b(" se a è maggiore o al più uguale a b")
Or	||	se la prima oppure la seconda condizione sono verificate
And	&&	Se entrambe le condizioni sono verificate
Uguaglianza	==	A==B ("se A è uguale a B ")
And (logico)	&	&a (esegue l'and bit a bit)
Or(logico)	|	|a (esegue l'or bit a bit)

Tabella 2.1 Principali sintassi logiche del linguaggio C.

2.2 Controllori

A. **PID:**

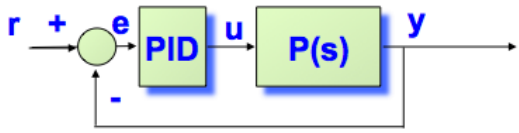


Figura 2.2 Schema a blocchi di un processo P(s) controllato tramite PID

Risulta sicuramente impossibile parlare di un sistema di controllo come quello adottato nella competizione della *Smart Car*, se prima non si definisce brevemente uno dei principali regolatori usati nel mondo: il PID acronimo che sta per Proporzionale Integrale Derivativo.
Questo controllore, uno dei primi ad essere stato implementato anche digitalmente, ha una struttura semplice a tre termini e costituisce la maggioranza dei regolatori industriali analogici.
Nella sua forma base il PID si presenta così:

$$C(s) = K_p + \frac{K_i}{s} + K_d s \qquad (2.1)$$

Il controllore C(s) è dato dalla somma di un coefficiente di proporzionalità con un polo moltiplicato da un coefficiente di integrazione e uno zero moltiplicato da un coefficiente di derivazione. Questo regolatore permette di evitare di modellare centinaia di sotto processi utilizzando un'unica struttura con dei parametri sintonizzabili caso per caso.
Nella sua forma analogica il PID permette di implementare funzioni di *auto-tuning* e di gestire un elevato numero di anelli di regolazione in *time-sharing*.

Normalmente i regolatori PID sono caratterizzati da tre parametri:

1- Tempo dell'azione integrale $T_i = \frac{Kp}{Ki}$;
2- Tempo di anticipo $T_d = \frac{Kd}{Kp}$;
3- Banda proporzionale B_p;

$$C(s) = K_p + \frac{K_i}{s} + K_d * s = K_p * \left(1 + \frac{1}{s*T_i} + s * T_d\right) \quad (2.2)$$

Vi sono diversi metodi per sintetizzare un controllore PID;
Il più immediato, forse, è quello basato sulla sintesi di Bode.

Vediamone ora i principali passi:

$$C_{PID}(s) = \frac{K_i}{s} C^*_{PID}(s) = \frac{K_i}{s}(1 + T_i s + T_i T_d s^2) \quad (2.3)$$

La parte $\frac{Ki}{s}$ viene fissata in base alle specifiche su tipo ed errore a regime, mentre la parte $C^*_{pid}(s)$ viene utilizzata per soddisfare la parte sulla pulsazione di attraversamento (ω_a) e sul margine di fase.
Analiticamente si procede in questo modo:

$$C^*_{PID}(j\omega_a) = (1 + jT_i\omega_a - T_i T_d {\omega_a}^2) = Me^{j\varphi} = M(\cos\varphi + j\sin\varphi) \quad (2.4)$$

$$\begin{cases} |C^*_{PID}(j\omega_a)| = M = \frac{\omega_a}{K_i|P(j\omega_a)|} \\ Arg[C^*_{PID}(j\omega_a)] = \varphi = m_\varphi - \frac{\pi}{2} - Arg[P(j\omega_a)] \end{cases} \quad (2.5)$$

$$\begin{cases} 1 - T_i T_d {\omega_a}^2 = M\cos\varphi \\ T_i\omega_a = M\sin\varphi \end{cases} \quad (2.6)$$

Bisogna ricordare che l'esistenza di questo controllore è verificata se e solo se vale:

$$M < \frac{1}{\cos(\varphi)} \quad (2.7)$$

L'aspetto dei PID, che forse ha avuto più rilevanza nella sua caratterizzazione all'interno della competizione *Freescale*, è la determinazione dei suoi parametri caratteristici.
Vediamone ora due metodi di calcolo teorico mediante la sintonizzazione di *Ziegler Nichols*:

1° Metodo:

- Si determina sperimentalmente un punto del diagramma di *Nyquist* del processo;
- Si inserisce la sola azione proporzionale del PID e si aumenta K_p sino ad arrivare al punto di passaggio tra stabilità ed instabilità per $K_p = K^*$;
- Per questo valore di K^* il diagramma di *Nyquist* di *K*P(jω)* passa per il punto critico *-1+j0* ad una certa pulsazione ω_c;

$$\begin{cases} |K_c P(j\omega_c)| = 1 \Longrightarrow |P(j\omega_c)| = \frac{1}{K_c} \\ Arg[P(j\omega_c)] = -\pi \end{cases} \tag{2.8}$$

- A regime si ottiene un'oscillazione persistente di cui si misura sperimentalmente il periodo T_0 dove :

$$T_0 = \frac{2\pi}{\omega c} \tag{2.9}$$

Arrivati a questo punto esistono delle tabelle che riportano i valori da assegnare a K_p, T_i e T_d a partire da K_c e T_0

tipo controllore	K_p	T_i	T_d
P	$0.5\ K_C$		
PI	$0.45\ K_C$	$T_0/1.2$	
PID	$0.6\ K_C$	$0.5\ T_0$	$T_0/8$

Tabella 2.2 Metodo *Ziegler Nichols*

2° Metodo:

- Si approssima la risposta al gradino del processo con quella di un sistema del primo ordine con ritardo;

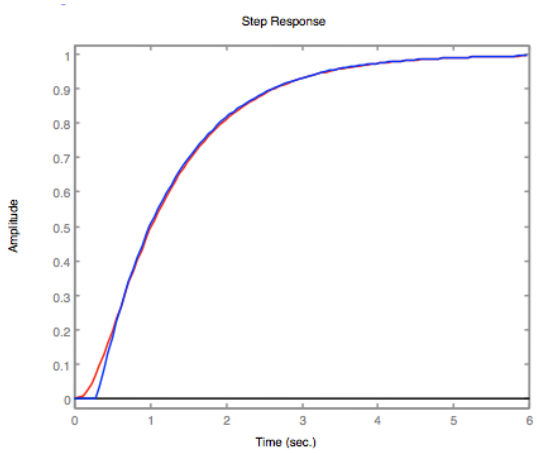


Figura 2.3 Esempio di risposta a gradino

$$P_a(s)^1 = K_G \frac{e^{-Ls}}{1+\tau s} \tag{2.10}$$

Anche qui esiste una tabella che attribuisce il valore dei parametri fondamentali del PID a seconda di K_G e L:

[1] Forma analitica della risposta a gradino del processo da controllare

tipo controllore	K_p	T_i	T_d
P	$\tau / K_G L$		
PI	$0.9\tau / K_G L$	3L	
PID	$1.2\,\tau / K_G L$	2L	0.5L

Tabella 2.3 Secondo metodo di sintesi

B. **Anti Reset Wind Up:**

Descriveremo ora una delle problematiche principali in cui ci si imbatte quando si implementa un PID.
Ricordando la definizione dei singoli termini del PID fatta precedentemente, si può notare come esso sia composto da due termini "istantanei" che variano di volta in volta in base all'ingresso e che non tengono di per sé conto della situazione precedente in cui si trovavano; questi sono il termine proporzionale e il termine derivativo.
Accanto a loro, tuttavia, c'è un termine fortemente influenzato dal suo storico, tale è il termine integrale che potrebbe arrivare a far saturare l'attuatore.
Per de-saturare il termine *I* ed evitare gli effetti indesiderati di sovraelongazione si ricorre al così detto *ARWU*.
Per i PID analogici si usa principalmente il seguente schema:

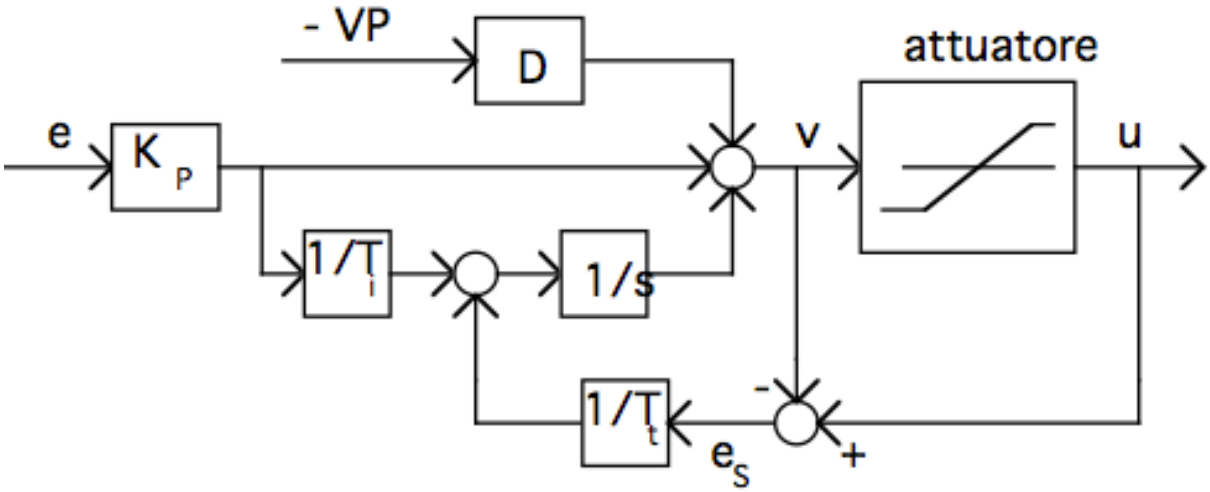


Figura 2.4 Schema di un Anti Reset Wind Up

Se l'attuatore non è in saturazione il PID funziona normalmente; se invece si raggiunge lo stadio di saturazione, allora viene fatto variare il termine *I* in modo che l'uscita *v* del controllore sia uguale all'uscita *u* dell'attuatore.
Tanto più basso sarà la costante di tempo di de-saturazione e tanto più velocemente *v* sarà riportata ad *u*.
Infatti minore sarà T_t e maggiore sarà il guadagno d'anello, diminuendo drasticamente gli effetti dei processi che tendono a far variare *v* come si può notare nella seguente figura:

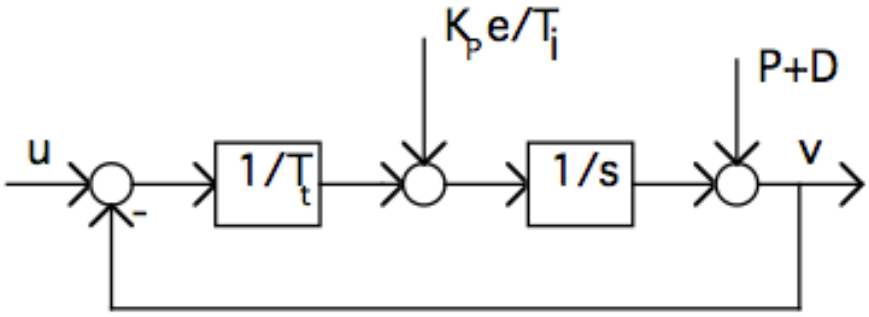


Figura 2.5 Schema a blocchi di un PID con Anti Reset Wind Up

Un esempio di *wind up* riferito alla *Smart Car* potrebbe rendere i concetti fin qui esposti di più facile comprensione:
Facendo riferimento allo schema in Fig. 2.4 si immagini che l'attuatore sia il nostro servo; ipotizzando che (come succede nella realtà) il veicolo non possa muovere di 360° le ruote, esso potrà muoversi al massimo in un range di valori che determinano la sterzata massima a destra (*Xdx*) e a sinistra (*Xsx*). Supponendo che ad un certo punto del tracciato il processo necessiti, per tornare a stabilità, di una sterzata maggiore di *Xdx* o di *Xsx* si incorrerebbe nella rottura del dispositivo oppure, nel caso in cui noi non avessimo limitato il campo di azione del PID, la parte integrale accumulerebbe valori di errore tali da portare il sistema in perenne instabilità visto che l' attuatore non potrebbe mai riuscire ad eliminare quel valore d'errore così elevato.

C. **PWM:**

La *Pulse Width Modulation* o modulazione a larghezza di impulsi è una tecnica di modulazione impulsiva in cui la portante è costituita da un treno d'impulsi e la modulante è di tipo analogico.

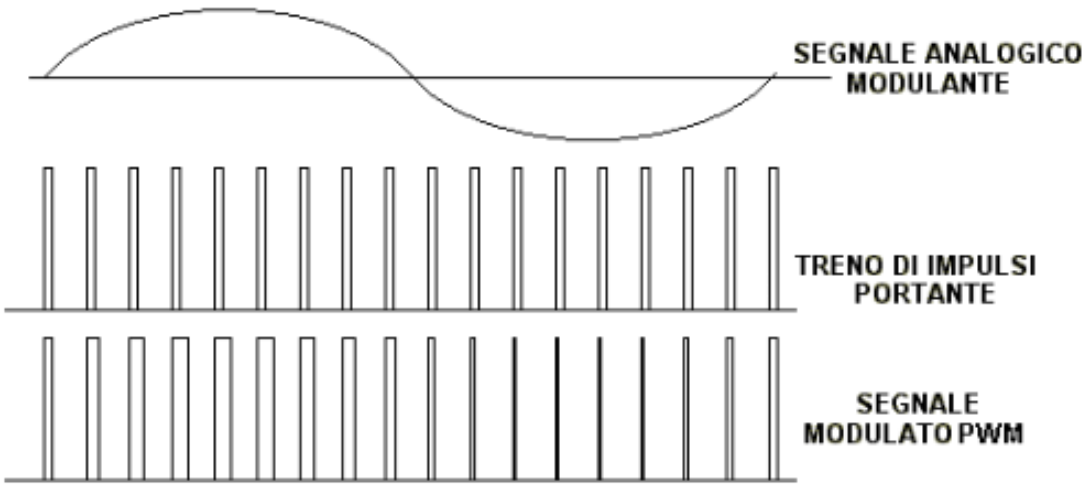


Figura 2.6 Esempio di segnale modulato in P.W.M.

Già dal nome possiamo capire che le ampiezze degli impulsi devono essere tutte uguali e che l'informazione data dal segnale modulante va a variare la larghezza, cioè la durata dell'ampiezza come mostrato in Fig. 2.6.
Questa tecnica viene molto utilizzata nel pilotaggio degli *inverter*.
La tensione e la frequenza, infatti, vengono solitamente modulate tramite *inverter*.

Per capire bene l'importanza di questo tipo di regolazione e per avere anche un riscontro pratico della sua definizione teorica, risulta utile ricorrere ad un esempio:
Sappiamo dal corso di *Fondamenti di Macchine ed Azionamenti Elettrici* che per variare l'ampiezza della tensione prodotta da un inverter occorre commutare opportunamente gli interruttori (ad esempio quelli di Fig. 2.7):

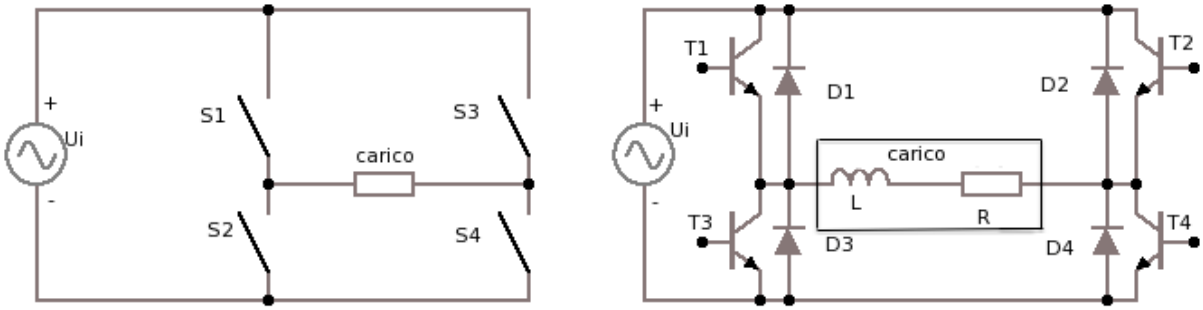


Figura 2.7 Esempio di inverter con relativi switch

Consideriamo ora un semplice caso in cui abbiamo due diversi istanti di commutazione $(a_1; a_2)$ con un segnale di comando ad onda quadra. Attraverso a_1 ed a_2 s'inseriscono all'interno dell'onda quadra alcuni buchi.

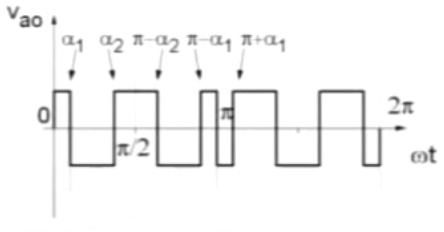


Figura 2.8 Segale di comando ad onda quadra modificato tramite $\boldsymbol{a_1}$ ed $\boldsymbol{a_2}$.

In questo caso i coefficienti di Fourier sono quelli mostrati dall'equazione (2.11)

$$V_{ao,n} = \frac{4}{\pi}\frac{V_{dc}}{2}\left[\int_0^{a_1} \sin(n\omega t)\, d\omega t - \int_{a_1}^{a_2} \sin(n\omega t)\, d\omega t + \int_{a_2}^{\pi/2} \sin(n\omega t)\, d\omega t\right] \qquad (2.11)$$

Il valore di picco sarà, quindi, quello calcolato in (2.12):

$$V_{ao,n} = \frac{2\,V_{dc}}{n\pi}\left[1 - 2\cos(na_1) + 2\cos(na_2)\right] \qquad (2.12)$$

dove n = 1,3,5.

Immaginiamo ora di voler eliminare la terza e la quinta armonica (caso di interesse pratico) si risolve il sistema di equazioni formato dalla (2.11) e dalla (2.12) ottenendo:

$$\begin{cases} 0 = 1 - 2\cos(3a_1) + 2\cos(3a_2) \\ 0 = 1 - 2\cos(5a_1) + 2\cos(5a_2) \end{cases} \qquad (2.13)$$

Ovvero a_1 = 23.62° e a_2 = 33.60°; ciò significa che per avere la terza e la quinta armonica completamente nulla, dobbiamo imporre questi due valori ad a_1 e a_2. Vediamo nella seguente tabella i valori che otteniamo per le prime nove armoniche nel caso dell'onda quadra e della PWM avendo imposto come valori di a_1 e a_2 quelli calcolati precedentemente:

Armonica	Onda quadra	PWM
1	1.0	0.8393
3	0.333	0
5	0.2	0
7	0.143	0.248
9	0.111	0.408
11	0.091	0.306

Tabella 2.4 Valori delle prime nove armoniche nel caso dell' onda quadra

Da tale tabella si può notare come tramite *Pwm* siamo riusciti a traslare verso l'alto tutte le frequenze.
Se applicassimo un filtro passabasso (come risulta qualsiasi carico *ohmico-induttivo*) avremmo un segnale molto più "pulito" da un punto di vista armonico.

2.3 Hardware

A. **Micro Controller Unit TRK-MPC5604B:**

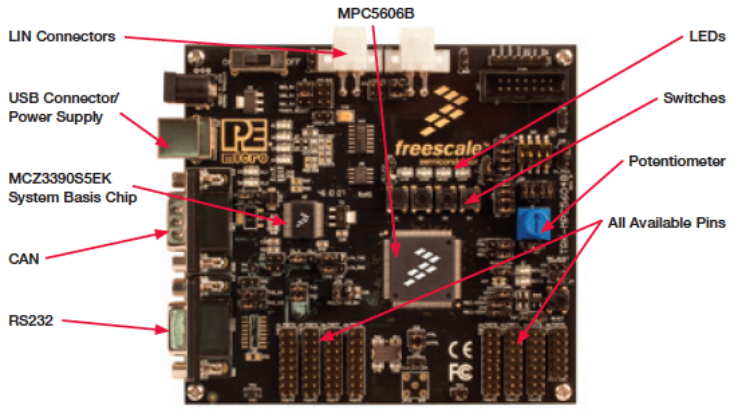


Figura 2.9 MicroController Unit TRK-MPC5604B

L'unico modo efficace per comprendere il funzionamento di questo micro controllore è di consultare i *data sheets* disponibili sul sito della *Freescale Semiconductors*. Leggendoli, si può notare come il *Qorivva MPC5604B* rappresenti una nuova generazione di microcontrollori a 32 bit basati su *Power Architecture* e come esso appartenga a una famiglia più ampia di prodotti ideati per applicazioni *automotive,* mirati ad affrontare e gestire il numero sempre in crescita di applicazioni elettroniche a bordo del veicolo. Questo microcontrollore lavora a *64 MHz* e offre alte prestazioni di *processing* ottimizzate per il consumo a bassa energia. Dai *data sheets*, possiamo ricavare le seguenti informazioni utili in fase di sviluppo del *software* e dell'*hardware* con cui il micro controllore andrà ad interfacciarsi:

CORE:

PowerPC e200z0 core running 48-64MHz

VLE ISA instruction set for superior code density

V ectored interrupt controller

Memory Protection Unit with 8 regions, 32byte granularity

MEMORY :

512Kbyte embedded program Flash, 64KByte data flash

64Kbyte embedded data Flash (for EE Emulation)

Up to 64MHz non-sequential access with 2WS

ECC-enabled array with error detect/correct

48Kbyte SRAM (single cycle access, ECC-enabled)

COMMUNICA TIONS :

3x enhanced FlexCAN__

64 Message Buffers each, full CAN 2.0 spec

4x LINFlex

3x DSPI, 8-16 bits wide & chip selects

1xI2C

ANALOG :

5V ADC 10-bit resolution

TIMED I/O :

16-bit eMIOS module

OTHER:

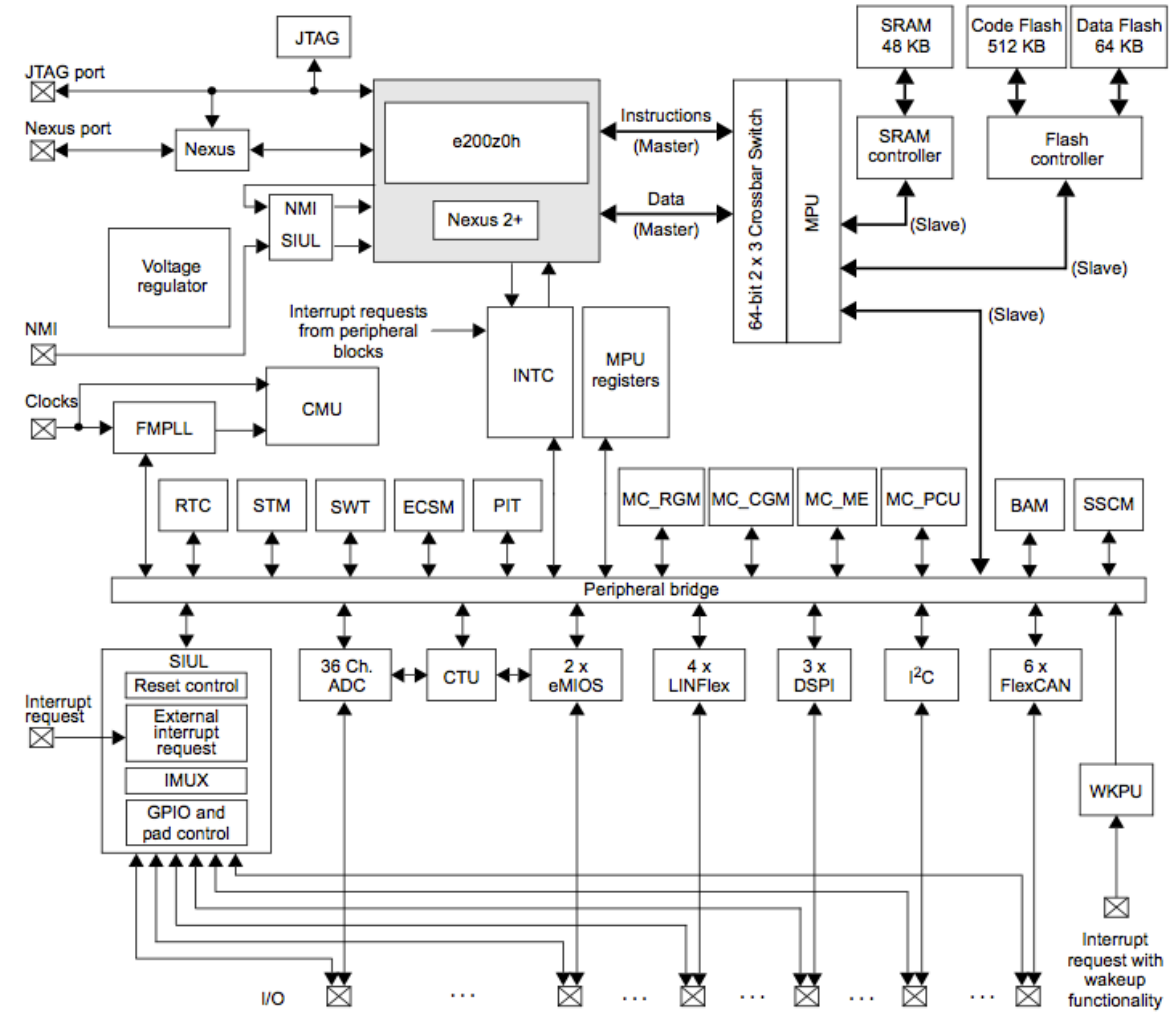
CTU (Cross Triggering Unit) to sync ADC with PWM Channels

I/O: 5V I/O, high flexibility with selecting GPIO functionality

Packages: 100LQFP, 144LQFP, 208MAPBGA (Development only)

Boot Assist Module for production and bench programming

Altra parte molto interessante ed utile da un punto di vista pratico è lo schema a blocchi del microcontrollore :



Legend:

ADC	Analog-to-Digital Converter
BAM	Boot Assist Module
FlexCAN	Controller Area Network
CMU	Clock Monitor Unit
CTU	Cross Triggering Unit
DSPI	Deserial Serial Peripheral Interface
eMIOS	Enhanced Modular Input Output System
FMPLL	Frequency-Modulated Phase-Locked Loop
I^2C	Inter-integrated Circuit Bus
IMUX	Internal Multiplexer
INTC	Interrupt Controller
JTAG	JTAG controller
LINFlex	Serial Communication Interface (LIN support)
ECSM	Error Correction Status Module
MC_CGM	Clock Generation Module
MC_ME	Mode Entry Module
MC_PCU	Power Control Unit
MC_RGM	Reset Generation Module
MPU	Memory Protection Unit
Nexus	Nexus Development Interface (NDI) Level
NMI	Non-Maskable Interrupt
PIT	Periodic Interrupt Timer
RTC	Real-Time Clock
SIUL	System Integration Unit Lite
SRAM	Static Random-Access Memory
SSCM	System Status Configuration Module
STM	System Timer Module
SWT	Software Watchdog Timer
WKPU	Wakeup Unit

Figura 2.10 Schema a blocchi del microcontrollore

B. **ADC**

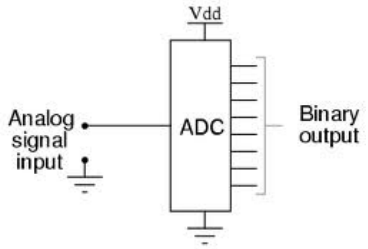


Figura 2.11 Schema di un A.D.C

L'ADC, acronimo che sta per *Analog to Digital Converter*, è un circuito di estrema importanza nella conversione di un segnale dal mondo reale al mondo digitale.
Per convertire un segnale analogico in una sequenza di bit, l'ADC utilizza principalmente 3 *steps*:

1- Campionamento (*sampling*)
2- Tenuta (*hold*)
3- Quantizzazione nel mondo delle ampiezze (*quantization*)

Ognuna di tali operazioni è implementata in un blocco circuitale specifico:

1. il campionatore preleva dal segnale d'ingresso una sequenza di campioni di tensione separati nel tempo da T_s secondi;
2. il circuito di tenuta memorizza temporaneamente il livello dei campioni in uscita dal campionatore;
3. il quantizzatore effettua la discretizzazione in ampiezza dei livelli di tensione dei campioni in ingresso.

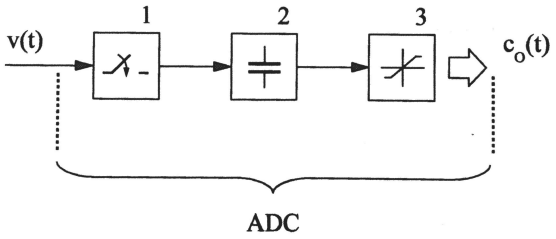


Figura 2.12 Schema di un Adc: 1) Campionatore, 2) Holder, 3) Quantizzatore

Vediamo ora nel dettaglio ciascuno dei circuiti che caratterizza un generico ADC:

1) **Campionamento**: è l'operazione che consiste nell'acquisire dal segnale d'ingresso una sequenza di campioni. La regola più comunemente usata per la scelta dei campioni è quella del campionamento uniforme. All'uscita del campionatore, il segnale campionato $v_s(t)$ è una sequenza di impulsi di tensione (campioni) distanziati nel tempo di un intervallo costante T_s, noto come tempo di campionamento, o periodo di campionamento. L'ampiezza dei campioni dipende dal segnale d'ingresso *v(t)*.
Il circuito di campionamento può essere pensato come un interruttore elettronico posto in serie alla linea d'ingresso, pilotato da un circuito oscillatore al quarzo (*clock*). La chiusura dell'interruttore avviene in corrispondenza del fronte di salita del segnale di *clock* e per un brevissimo intervallo di tempo. Dell'intero segnale d'ingresso *v(t)*, solo la porzione corrispondente agli istanti di campionamento passa per l'interruttore e prosegue lungo la catena di acquisizione. Tutto il resto è bloccato dallo stesso interruttore, non prosegue lungo la catena di acquisizione e non partecipa alle successive fasi di conversione in digitale, elaborazione e visualizzazione dei risultati.

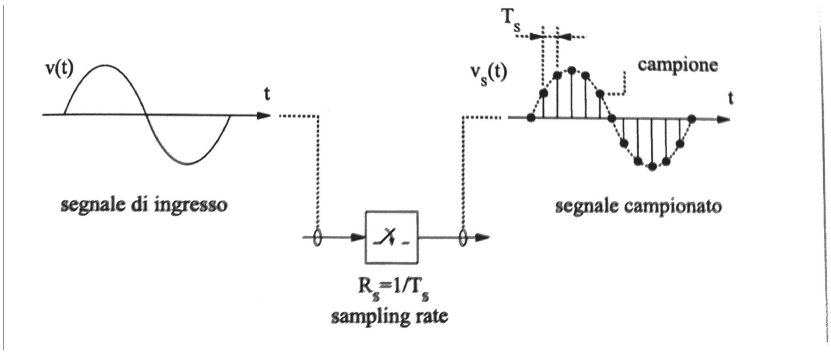


Figura 2.13 *Sampling* di un segnale sinusoidale

2) **Tenuta**: è l'operazione che consiste nel mantenere i livelli di tensione $v_s(t)$ dei campioni acquisiti per un intervallo di tempo pari al tempo di campionamento, cioè fino al campionamento successivo. Ciò che esce dal blocco di *hold* è un segnale costante a tratti.
La funzione del circuito di *track* ed *hold* avviene in 2 fasi: fase di carica e fase di tenuta.

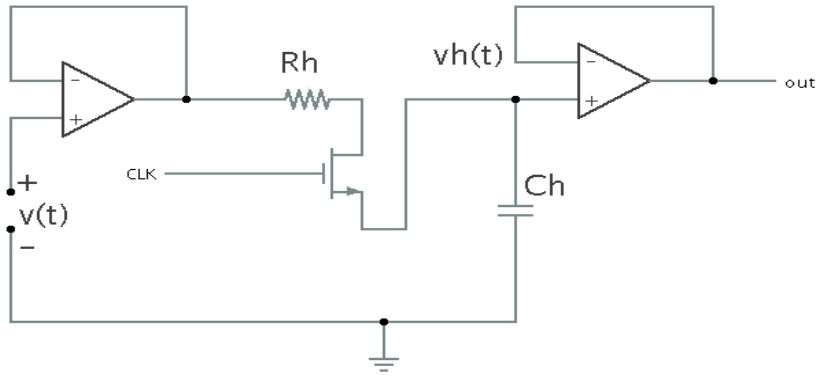


Figura 2.14 Circuito *track and hold*

3) **Quantizzazione**: è quell'operazione attraverso la quale il segnale campionato $v_h(t)$ presente all'uscita del circuito di *track and hold* è trasformato in una sequenza di livelli di tensione quantizzati $v_o(t)$ o di codici $c_o(t)$, uno per ogni campione. L'operazione di quantizzazione del segnale d'ingresso avviene campione per campione. In ogni intervallo di durata *Ts*, il dispositivo approssima il livello di tensione assunto da $v_h(t)$ ad uno dei possibili livelli di uscita v_o, in genere a quello più vicino.
I livelli di v_o sono ottenuti suddividendo il campo di valori di ingresso del dispositivo di ampiezza R, in N intervalli di ampiezza Q detti passo di quantizzazione.
La caratteristica più importante di un circuito di quantizzazione è la sua transcaratteristica, la quale ne determina le prestazioni e le eventuali inefficienze.

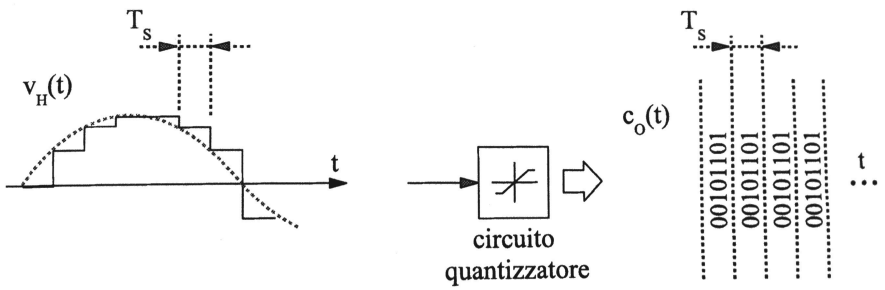


Figura 2.15 Fase di quantizzazione

Capitolo 3

Parti del Veicolo che ne Condizionano il Controllo

3.1 Line Scan Camera

A. **Struttura e Funzionamento**

Il sensore principale, quello da cui dipende più del 90% dell'analisi del tracciato da parte del veicolo *Freescale*, è la telecamera fornita con il *kit*. La *Smart Car* è un veicolo autonomo e la camera rappresenta il suo più importante punto di interfaccia con il mondo esterno; infatti è ciò che permette al veicolo di analizzare e predire l'andamento del tracciato.

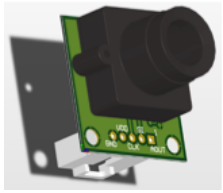


Figura 3.1 Sensore TSL1401CL

Il sensore in questione è il *TSL1401CL* prodotto dalla *Taos*; esso è costituito principalmente da una linea di 128x1 fotodiodi con una superficie di 3524.3 [μm^2], distanziati tra loro di 8 [μm] e da un fuoco regolabile con una lente di 7.9 [mm]. L'utente, inoltre, può facilmente variare la frequenza di campionamento e questo comporta una enorme flessibilità da parte del sensore.

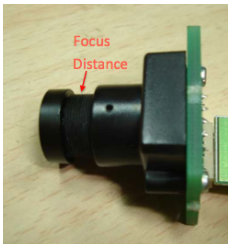


Figura 3.2 Regolazione del fuoco della lente

Una "pecca" di questo dispositivo è il fatto che l'uscita è solo di tipo analogico e quindi risulta fondamentale per l'utente adottare un metodo semplice e rapido per processare il segnale.
Ora che si è visto com'è composto il sensore si può passare ad illustrare il principio di funzionamento facendo riferimento a Fig.3.3.

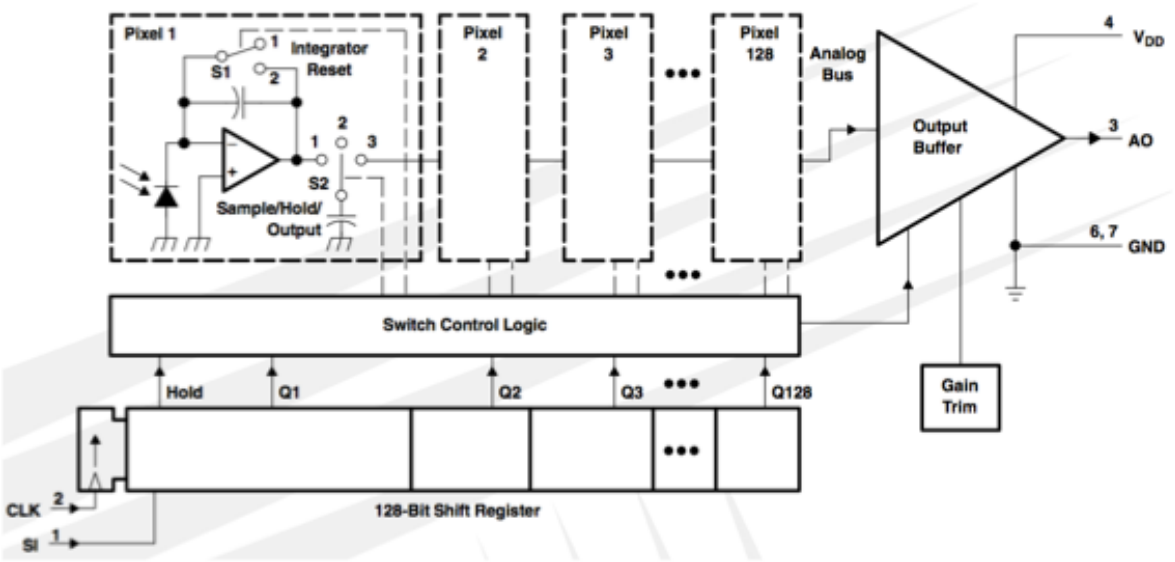


Figura 3.3 Schema funzionale del sensore

Da tale schema a blocchi si può notare che il principio di funzionamento della telecamera è abbastanza semplice:
Come noto la luce trasporta energia, questa energia quando colpisce il fotodiodo genera una corrente che carica un condensatore di *sampling* attraverso il circuito d'integrazione.
La carica di tale condensatore si mantiene grazie all'intervento dello *switch 2* e sarà proporzionale all'intensità luminosa che ha colpito quel preciso *pixel* e al tempo d'integrazione.
Prima di proseguire con la descrizione del principio di funzionamento della "*Camera*" risulta fondamentale dare alcune definizioni :

Vdd: Tensione di alimentazione
A0: *Analog Output*
Gnd: *Ground*
SI: *Serial Input*: Segnale che definisce l'inizio di ogni acquisizione.
Clk: Segnale di *clock* utilizzato per scandire gli istanti di commutazione degli *switch*.

Si può inoltre notare come la tensione analogica in uscita sia strettamente legata ad una logica di *reset* e ad un registro di *shift* a 128 *bit*; queste hanno inizio solamente quando *SI* ha valore logico "1".
Affinché tutti i condensatori di campionamento vengano simultaneamente scollegati dai rispettivi circuiti, per permettere l'inizio di un periodo di *reset*, è necessario che all'interno del circuito venga generato un segnale di *hold* e che venga trasmesso agli interruttori dei singoli *pixel*. All'arrivo di *SI* la carica accumulata dai condensatori verrà amplificata e poi trasmessa generando *A0*.
È necessario infine che il segnale *SI* venga "resettato" a 0 prima del nuovo fronte di salita del segnale di *clock*.
In queste condizioni si avrà un'uscita nominale di 0 [*V*] con una *Vdd* = 5 [*V*] (assenza di luce), un'uscita pari a 2 [*V*] corrispondente al bianco e 4.8 [*V*] nel caso di saturazione.
Dopo aver esaminato il modello circuitale del sensore, vediamo ora delle interessanti osservazioni pratiche.
Dapprima diamo i parametri del sensore ricavabili dai *data sheets*:

- *Focusable imaging lens*
- *5-pin physical interface on PCB on .100" grid*
- *Simple three-pin MCU interface with analog pixel output*

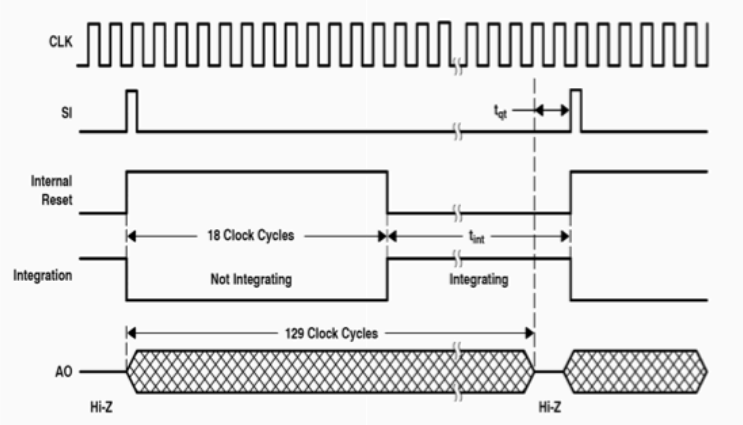
- *Lens: 7.9 [mm] focal length, fixed aperture, manual focus, 12[mm] x 0.5[mm] thread*
- *Exposure Time: 267μs to 68ms*
- *Resolution: 128 pixels*
- *Built-In amplifier stage to improve white/black differentiation*

In accordo con i *data sheets* e con quanto spiegato dal modello circuitale si possono notare le seguenti limitazioni:
L'ammontare totale della carica accumulata da ciascun *pixel* è direttamente proporzionale all'intensità luminosa e al tempo di integrazione, che è facilmente calcolabile con la seguente espressione:

$$\mathrm{T} = \left(\frac{1}{\mathrm{fmax}}\right) * (\mathrm{n} - 18)\mathrm{pixels} + 20\mu\mathrm{s} \qquad (3.1)$$

dove *n* è il numero di *pixel*.

Si può, dunque, verificare che il tempo minimo di integrazione risulta essere 33.75 [μs] mentre il massimo periodo di integrazione è legato alla saturazione dei condensatori, ossia a 100 [ms].
La frequenza di lavoro, dunque, può variare dai 5 [kHz] agli 8 [kHz].
L'azienda produttrice, inoltre, dà una ben specifica definizione di tempo di acquisizione:
" The integration time is the following: It occurs between the 19th CLK cycle and the next SI pulse. The CLK frequency itself has little to do with the integration time. One each rising edge, the clock outputs one of the previously sampled intensity values. This means that integration time should be set by varying the time between SI pulses, not changing the clock frequency. Make the CLK frequency high, and have as much time as needed between the two SI pulses to obtain the desired intensity value."



The minimum integration time is calculated by the following formula.

$$T_{int(min)} = \left(\frac{1}{maximum\ clock\ frequency}\right) \times (n - 18)\ pixels + 20\mu s$$

Figura 3.4 Visualizzazione schematica dei segnali principali del sensore e calcolo del minimo tempo d'integrazione

Passiamo ora all'aspetto operativo di calibrazione della lente e dell'applicazione del sensore.
Per mettere a fuoco la camera è necessario eseguire i seguenti passi:
- Connettere *SI* e *A0* all'oscilloscopio;
- Settare *SI* in modo che si possa vedere chiaramente e "*triggerare*" *A0* con *SI* utilizzando la funzione di *trigger* presente nell'oscilloscopio;
- Puntare la *Camera* verso un pezzo di carta bianco con una linea nera.
- Ruotare il fuoco finché l'immagine non appare più chiara.

Apparirà un'immagine come la seguente:

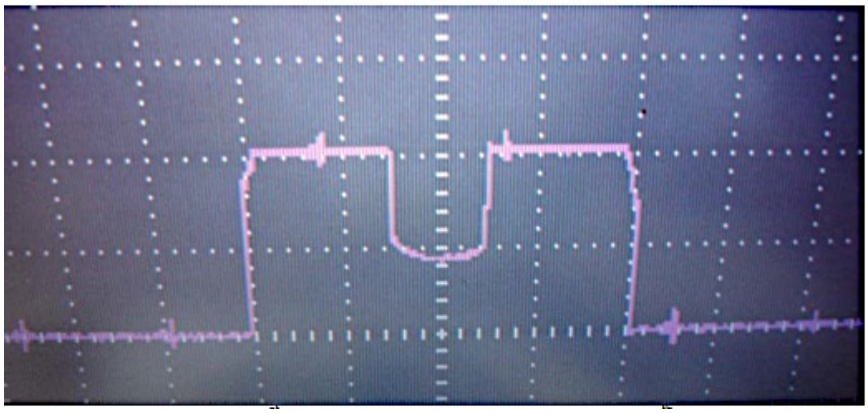


Figura 3.5 Immagine della telecamera vista mediante DSO.

Prima di concludere, è utile affrontare la problematica dell'interpretazione e gestione dei segnali provenienti dalla *TSL1401CL*.
Il principio di funzionamento del sensore risulta chiarito oltre che dalle considerazioni precedenti anche dalla seguente figura reperibile dal sito della *Freescale*:

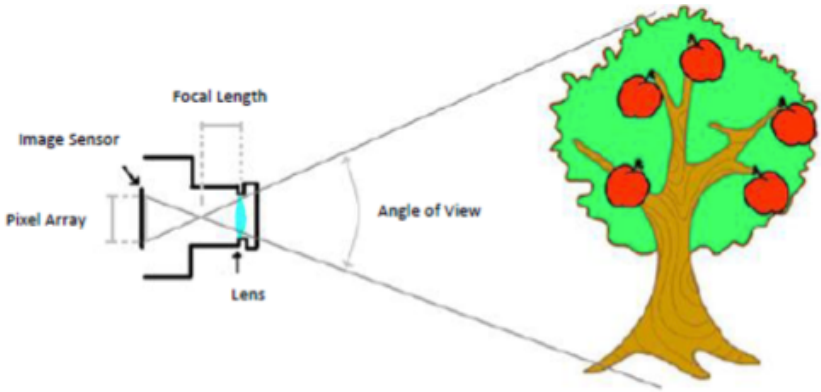


Figura 3.6 Esempio di acquisizione di un'immagine tramite TSL1401CL

Come detto questo sensore è lineare quindi un'immagine reale viene linearizzata una volta passata attraverso il fuoco della camera riproducendo un segnale analogico come quello mostrato in Fig.3.7.

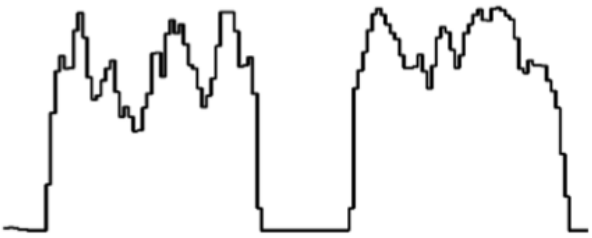


Figura 3.7 Segnale analogico che arriva alla camera

I *pixel* con una componente più alta di energia luminosa saranno quelli alti mentre quelli bassi saranno quelli meno irradiati.
A livello *software* l'utente dovrà quindi considerare oltre a questo aspetto anche la gestione del segnale di *clock*, del *SI* e di *A0*.
Infatti, questi tre segnali, come già osservato, consentono di variare il tempo di integrazione e dunque la "bontà" del segnale d'ingresso:
Come si nota in Fig.3.8 *SI* e *A0* devono avere lo stesso periodo;
SI e *Clk* devono essere tra loro sfasati di mezzo periodo mettendo così in fase *Clk* ed *A0*.
Al termine del 129° ciclo sarà inoltre cura dell'utente abbassare il segnale di *Clock* per migliorare la carica dei condensatori di campionamento.

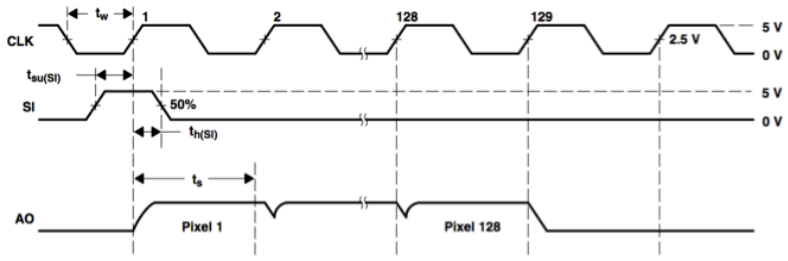


Figura 3.8 Andamento nel tempo dei segnali di *clk,A0,SI*

3.2 Ponte ad H

A. **Struttura**

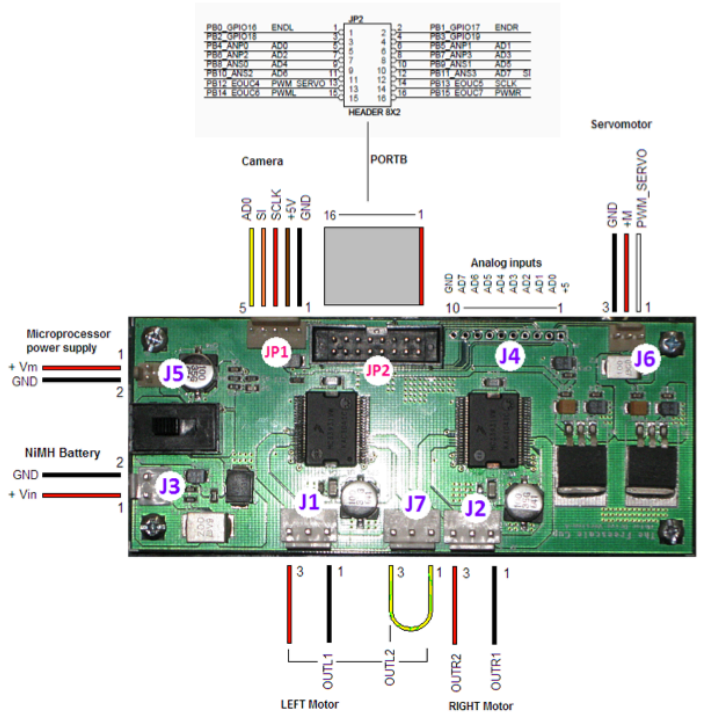


Figura 3.9 Immagine del ponte ad H con relativi segnali di I/O

Il ponte ad H o "*H-bridge*" fa parte della famiglia dei dispositivi per l'elettronica di potenza.
Analizziamone ora le caratteristiche mediante lo schema in Fig.3.10 riferito al caso di un motore in CC come carico:

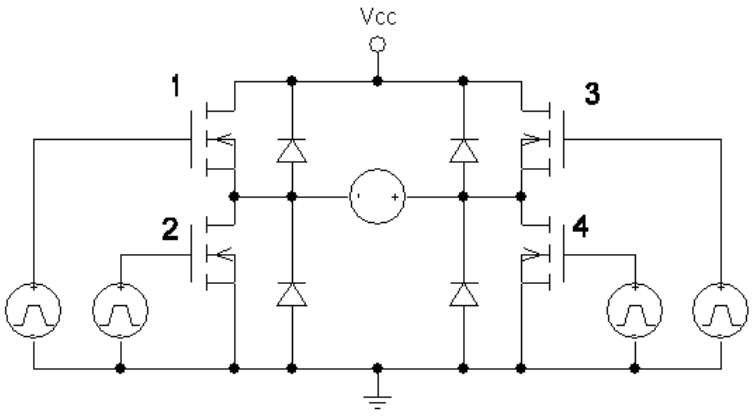


Figura 3.10 H-bridge con carico ohmico-induttivo

Come si può notare questo dispositivo permette di pilotare il motore in entrambe le direzioni: *forward* e *reverse* tramite tecnica di modulazione in *Pwm*.
I *mosfet* 2 e 4 sono detti *sink* in quanto assorbono la corrente proveniente dal motore.
Si può facilmente riscontrare come, a seconda dei transistori che la nostra modulazione attiverà, il motore sarà percorso da una corrente che ne determinerà il verso di rotazione.

Ad esempio:

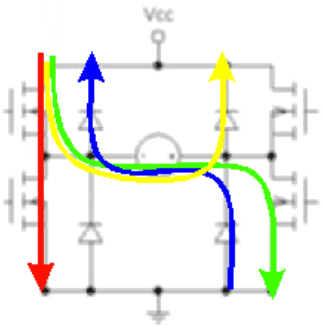


Figura 3.11 Andamento delle correnti nel ponte ad H

Attivando il transistore in alto a sinistra e quello in basso a destra otterremo il passaggio della corrente nel verso indicato dalla linea verde di Fig. 3.11, che farà ruotare il motore in un determinato verso. Utilizzando la coppia simmetrica di transistori otterremo la rotazione nel verso opposto.
Attivando i due transistori in alto o i due in basso otterremo un cortocircuito come evidenziato dalla linea rossa di Fig. 3.11. Risulta superfluo dire che questo caso, una volta implementata la modulazione, sarà evitato.
La linea blu indica la situazione in cui tutti e quattro i transistori sono spenti; una volta esaurita la scarica dell'induttore se il motore era prima in moto ora andrà verso l'arresto. Se è attivo solamente il *transistor* 3 o 1 la corrente seguirà la linea gialla e si avrà una forte azione frenante in quanto ai capi del motore vi sarà un cortocircuito.
Come detto in precedenza, è possibile pilotare il ponte tramite una modulazione in *Pwm*.

Principalmente ne esistono due modalità :

1) Si utilizza un segnale *Pwm* e una costante che indica il verso di rotazione del motore. Facendo variare il *duty cicle* dallo 0% (condizione di fermo) al 100% (condizione di piena potenza) ed inviando il segnale alla coppia di transistori indicati dalla linea verde in Fig. 3.11 oppure ai loro simmetrici; mantenendo spenti gli altri due si otterrà l'avanzamento del motore.

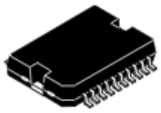
2) Si utilizza un solo segnale in *Pwm* che indichi quale coppia di transistori dev'essere accesa (ad esempio segnale alto coppia 1,4 segnale basso coppia 3,2). L'effetto sarà di forzare la corrente nel motore prima in un verso e poi nell'altro; a causa dell'induttanza (si ricorda che il motore è assimilabile ad un carico ohmico-induttivo) si avrà una stabilizzazione attorno al 50% del *duty cicle*. Ciò implica che per far andare il motore in un verso basterà superare il 50% del *duty cicle* mentre per farlo andare nel verso opposto basterà scendere sotto il 50%.

B. **Data Sheet:**

Vediamo ora i *data sheets* delle caratteristiche principali dell'H-bridge incorporato nel *kit* di base:

33886

H-BRIDGE



VW SUFFIX (PB-FREE)
98ASH70702A
20-PIN HSOP

ORDERING INFORMATION		
Device	Temperature Range (T_A)	Package
MC33886PVW/R2	-40 to 125 °C	20 HSOP

Figura 3.12 Data sheet H-bridge

PIN CONNECTIONS

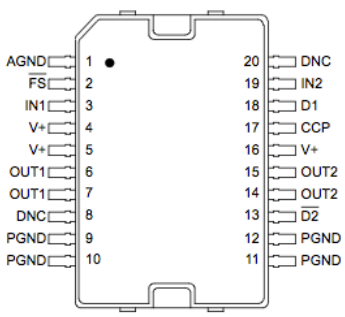


Figura 3.13 Descrizione del collegamento dei pin

Features:

- *5.0[V] to 40[V] Continuous Operation*
- *120[mΩ] $R_{DS(ON)}$ H-Bridge MOSFETs*
- *TTL/CMOS Compatible Inputs*
- *PWM Frequencies up to 10 kHz*
- *Active Current Limiting via Internal Constant OFF-Time PWM (with Temperature-Dependent Threshold Reduction)*
- *Output Short-circuit Protection*
- *Under-voltage Shutdown*
- *Fault Status Reporting*

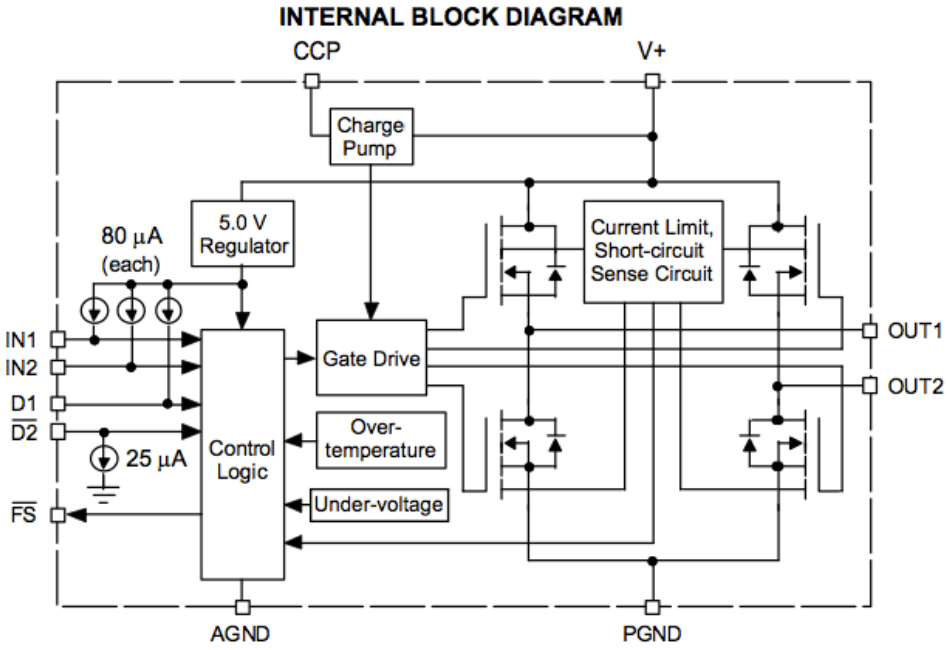


Figura 3.14 Schema a blocchi dell'interno del ponte ad H

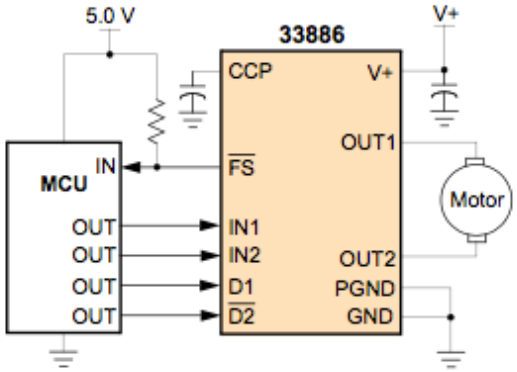


Figura 3.15 Collegamento di un motore al ponte

Pin Number	Pin Name	Formal Name	Definition
1	AGND	Analog Ground	Low-current analog signal ground.
2	$\overline{FS}$	Fault Status for H-Bridge	Open drain active Low Fault Status output requiring a pull-up resistor to 5.0 V.
3	IN1	Logic Input Control 1	True logic input control of OUT1 (i.e., IN1 logic High = OUT1 logic High).
4, 5, 16	V+	Positive Power Supply	Positive supply connections.
6, 7	OUT1	H-Bridge Output 1	Output 1 of H-Bridge.
8, 20	DNC	Do Not Connect	Either do not connect (leave floating) or connect these pins to ground in the application. They are test mode pins used in manufacturing only.
9–12	PGND	Power Ground	Device high-current power ground.
13	$\overline{D2}$	Disable 2	Active Low input used to simultaneously tri-state disable both H-Bridge outputs. When D2 is logic Low, both outputs are tri-stated.
14, 15	OUT2	H-Bridge Output 2	Output 2 of H-Bridge.
17	CCP	Charge Pump Capacitor	External reservoir capacitor connection for internal charge pump capacitor.
18	D1	Disable 1	Active High input used to simultaneously tri-state disable both H-Bridge outputs. When D1 is logic High, both outputs are tri-stated.
19	IN2	Logic Input Control 2	True logic input control of OUT2 (i.e., IN2 logic High = OUT2 logic High).

Tabella 3.1 Riassunto delle funzioni dei pin

Rating	Symbol	Value	Unit
Supply Voltage	V+	40	V
Input Voltage (1)	V_{IN}	-0.1 to 7.0	V
$\overline{FS}$ Status Output (2)	$V_{\overline{FS}}$	7.0	V
Continuous Current (3)	I_{OUT}	5.0	A
ESD Voltage for VW Package Human Body Model (4) Machine Model (5)	 V_{ESD1} V_{ESD2}	 ±2000 ±200	V
Storage Temperature	T_{STG}	-65 to 150	°C
Ambient Operating Temperature (6)	T_A	-40 to 125	°C
Operating Junction Temperature	T_J	-40 to 150	°C
Peak Package Reflow Temperature During Reflow (7), (8)	T_{PPRT}	Note 7.	°C
Approximate Junction-to-Board Thermal Resistance (and Package Dissipation = 6.0 W) (9)	$R_{\theta JB}$	~5.0	°C/W

Tabella 3.2 Riassunto dei principali parametri del ponte ad H

3.3 Motore CC

A. **Struttura**

Impossibile iniziare la trattazione del controllo di un veicolo senza prima aver dato una definizione ed una caratterizzazione dei motori in *DC* (*direct current*).[1]

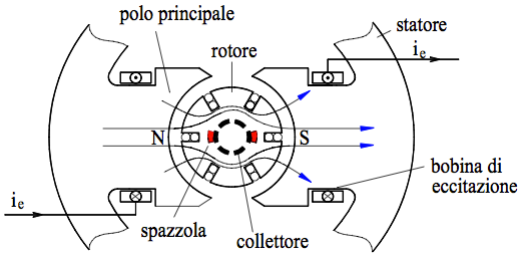


Figura 3.16 Rappresentazione schematica di un motore in CC

Una rappresentazione schematica della struttura di un motore a corrente continua a due poli è mostrata in Fig.3.16. Essa comprende una parte fissa (statore) che costituisce l'induttore della macchina (la struttura che produce il campo magnetico principale), dotata dei poli induttori su cui sono avvolte le bobine di eccitazione che, nell'insieme, formano l'avvolgimento di eccitazione (o induttore o di campo). Esiste quindi una parte rotante (rotore) che rappresenta l'indotto della macchina il cui avvolgimento (armatura) ha i suoi conduttori, detti conduttori attivi, collocati entro canali (cave) ricavati lungo le generatrici del cilindro rotorico. L'armatura è alimentata dalla corrente *ia* inviata attraverso contatti fissi (spazzole) striscianti su un sistema di lamelle solidale con il rotore (collettore). Statore e rotore sono separati da una sottile corona d'aria che prende il nome di traferro. Quando una corrente *ie* è inviata negli avvolgimenti di eccitazione, mentre i conduttori di indotto non sono sede di correnti (funzionamento a vuoto), si instaura nella macchina un campo magnetico principale, le cui linee di campo sono esemplificate in Fig.3.16, in cui il motore è rappresentato sviluppato e disteso su un piano.
Tale campo attraversa il traferro ed il rotore e si richiude nello statore; ad esso corrisponde una distribuzione lungo il traferro della componente radiale dell'induzione che presenta, in virtù della sagomatura dei poli stessi, l'andamento rappresentato in Fig. 3.17.

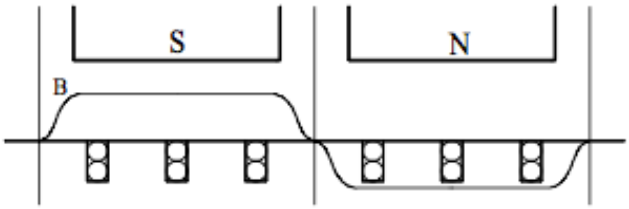


Figura 3.17 Andamento del campo magnetico nello statore

Assumendo la convenzione di segno degli utilizzatori, l'avvolgimento di eccitazione è retto dall'equazione dinamica:

$$u_e = R_e i_e + \frac{d\lambda_e}{dt} \tag{3.2}$$

Per descrivere il funzionamento del sistema indotto, si faccia riferimento alla Fig.3.18, ove il semplice rotore con 6 cave e 12 conduttori è rappresentato con maggior dettaglio ed è supposto, per il momento, fermo.

[1] Op. cit. Fondamenti di Macchine ed Azionamenti Elettrici Professor Mauro Zigliotto

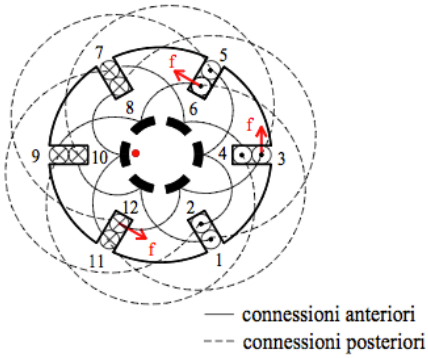


Figura 3.18 Funzionamento dell' indotto

Si immagini che il conduttore attivo 3 di figura sia sede di una corrente positiva i con il verso indicato. Se U è l'induzione radiale al traferro nella posizione occupata dal conduttore in esame, su di esso verrà ad agire la forza tangenziale mostrata in figura, di intensità $f = l\,(i\ x\ \beta)$ essendo l lo sviluppo assiale del conduttore. Se la stessa corrente *i* percorre anche tutti gli altri conduttori sotto il polo S e, con il verso contrario come evidenziato in Fig. 3.18, anche quelli sotto il polo N, su tutti verranno ad agire forze tangenziali di versi concordi (alcune indicate in figura), proporzionali al valore dell'induzione radiale al traferro nella posizione occupata dai conduttori stessi. In queste condizioni il rotore è sottoposto ad una coppia risultante diversa da zero che tende a metterlo in rotazione. Opportune connessioni fra i diversi conduttori e l'alimentazione degli stessi attraverso contatti striscianti spazzole-lamelle fanno sì che la distribuzione delle correnti rispetto ai poli sia sempre quella di Fig.3.18 qualunque sia la posizione del rotore sicché la coppia prodotta per una data corrente mantiene sempre lo stesso valore e segno. Per comprendere come ciò sia possibile si faccia riferimento alla schematizzazione di Fig.3.19 che illustra, sviluppato e disteso su un piano, l'avvolgimento indotto della macchina di Fig.3.18.

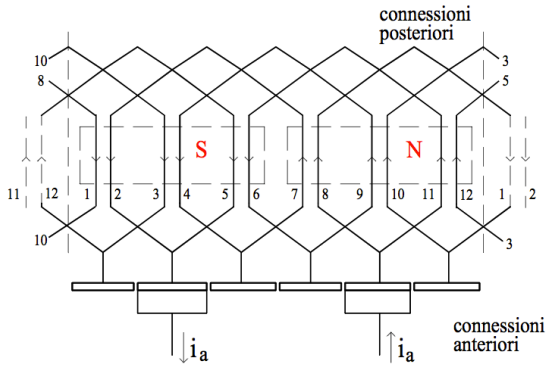


Figura 3.19 Avvolgimento dell'indotto sviluppato in un piano

Sulla stessa figura sono pure mostrati i collegamenti fra le lamelle del collettore e l'avvolgimento indotto, nonché la posizione che devono avere le spazzole. Si riconosce che l'avvolgimento d'indotto collega tra loro i diversi conduttori attivi così da formare un avvolgimento chiuso, condizione essenziale per garantire la necessaria simmetria rotazionale all'avvolgimento. Alcune di queste connessioni sono mostrate anche in Fig.3.18. La corrente di armatura *ia* inviata e prelevata attraverso le spazzole si suddivide nelle due vie interne identiche, ciascuna formata da sei conduttori, che si possono individuare percorrendo l'indotto da una spazzola all'altra. Si riconosce altresì che tutti i conduttori che giacciono sotto lo stesso polo sono percorsi da correnti equiverse, anche se essi appartengono per metà ad una delle due vie interne e per i rimanenti all'altra.
Spostando il rotore di un sesto di giro (per esempio in senso antiorario in Fig.3.18, verso de-

stra in Fig.3.19, corrispondente ad un movimento verso sinistra delle spazzole e della proiezione dei poli) rimane inalterata la distribuzione delle correnti sotto ogni polo. In particolare nelle coppie di conduttori 11,12 e 5,6 si ha un'inversione della corrente ed essi prendono il posto che già era, rispettivamente, delle coppie 1,2 e 7,8.
Durante ogni spostamento di questo tipo si ha una commutazione della corrente delle spazzole da una lamella alla successiva, che riporta le condizioni operative della macchina del tutto equivalenti a quelle di partenza, con conseguente costanza della coppia prodotta.
Posizionando invece il rotore in una posizione intermedia, come rappresentato in Fig.3.20, ogni spazzola alimenta due lamelle adiacenti, con correnti proporzionali alle relative superfici di contatto. Durante la commutazione si ha, allora, una graduale riduzione della corrente nelle lamelle uscenti dalle spazzole, una graduale crescita in quelle entranti mentre nelle due spire in commutazione (formate dai conduttori 11,6 e 12,5) e un graduale rovesciamento di corrente la quale passa per lo zero nel momento in cui le spire in questione transitano per il piano interpolare (piano di commutazione o delle spazzole) ove l'induzione del campo induttore è nulla.

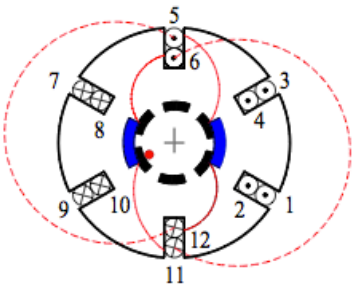


Figura 3.20 Avvolgimento dell'indotto della macchina

Se il numero di cave è sufficientemente alto, si può senz'altro assumere che anche durante la commutazione si venga a produrre sempre la stessa coppia. Essa sarà proporzionale all'intensità delle correnti nei conduttori, e quindi ad *ia*, e all'induzione media sotto ciascun polo, e quindi a ϕ, e potrà in ultima analisi essere espressa con la (3.3)

$$\tau = K_\tau \phi i_a \tag{3.3}$$

B. **Data Sheet**

Figura 3.20 Immagine del motore DC in dotazione con il kit Freescale

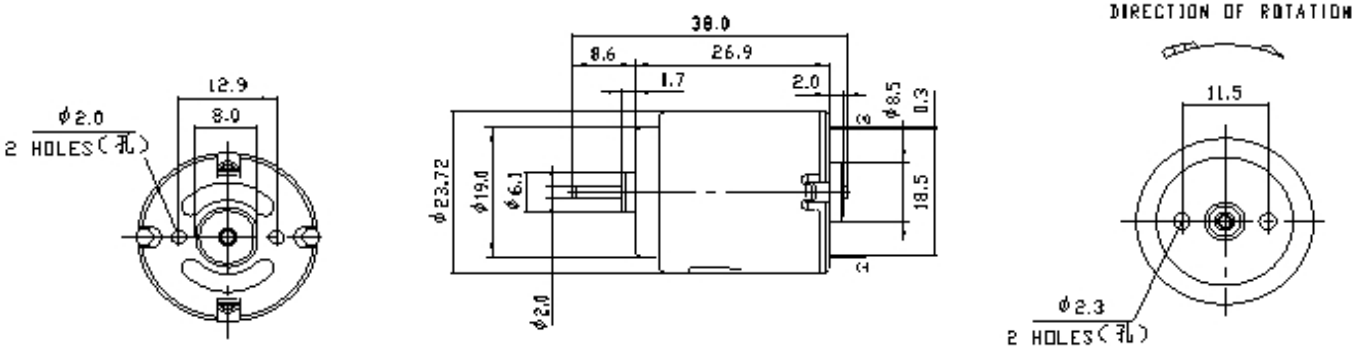


Figura 3.21 Dimensione dei motori in CC in dotazione

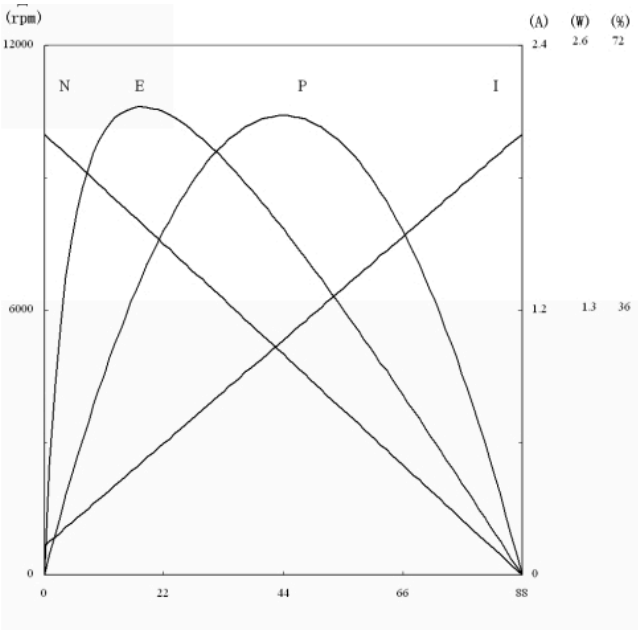


Figura 3.22 Transcaratteristica del motore

No Load		Max Efficiency			Max Output			Stall	
Current	Speed	Current	Speed	Torque	Current	Speed	Torque	Current	Torque
A	rpm	A	rpm	g.cm	A	rpm	g.cm	A	g.cm
0.13	10000	0.51	7950	18	1.07	5000	44	2	88

Tabella 3.3 Specifiche tecniche dei motori

Diamo ora una breve descrizione dei termini emersi dai *data sheet* del motore *RN260-C winding 18130* così da poterli comprendere meglio.
In Tab.3.3 notiamo i termini:
No load: implica che i dati di quella specifica colonna sono stati misurati dal costruttore in assenza di carico facendo ruotare il rotore con la sua tensione di armatura nominale, infatti si ottiene una velocità di *10000*2*π/60 =1047* [rad/s] e una corrente di armatura di 0.13 [A].
Utilizzando invece un freno esterno si può incrementare la coppia resistente fino a bloccare la rotazione dell'albero ottenendo così una *stall torque* di 88 [gcm] = 88*9.81*10^-2 [Nm] = 8,6 [Nm] e una *stall current* di 2 [A].
Max Efficiency rappresenta la colonna delle caratteristiche del motore nel punto di funzionamento nominale le così dette *Rated Torque, Rated Speed* e *Rated Current*.
I punti di massimo del motore sono visualizzabili sia dalla Tab.3.3 sotto il nome di *Max Output* sia dalla caratteristica meccanica di Fig.3.22, infatti i punti di massimo della curva *rpm-torque* sono esattamente quelli della tabella .

3.4 Servomotore

A. **Struttura**

Figura 3.23 Immagine del servomotore

Così come l'*H-bridge* anche il servomotore fa parte della famiglia dell'elettronica di potenza in grado però di gestire il movimento di organi adatti a tale scopo.
La peculiarità di questo dispositivo è di poter essere controllato da un segnale logico a bassa potenza.
Solitamente è costituito da un motore elettrico completo di riduzione meccanica e un sistema di *feedback* per la posizione dell'asse di uscita.
Optando per una logica di controllo piuttosto che per un'altra, è possibile far ruotare l'asse di uscita (la parte gialla in Fig.3.23) in una qualsiasi posizione.
In Fig.3.24 è possibile vedere il collegamento elettrico del servo al suo sistema di controllo e di alimentazione.

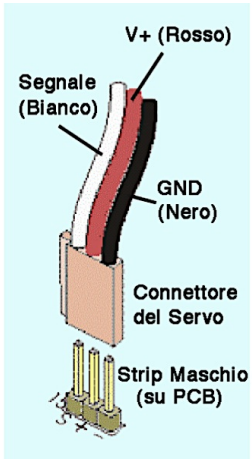


Figura 3.24 Collegamento del servomotore

In condizioni di normale funzionamento è possibile far ruotare il servo solo di ±180° rispetto al centro del suo asse. Inoltre solitamente i servomotori vengono controllati in *PCM (Pulse Code Modulation)* come si può notare da un' attenta analisi all'oscilloscopio.
Per capire come funzioni questo dispositivo è sufficiente considerare lo schema riportato in Fig.3.25:

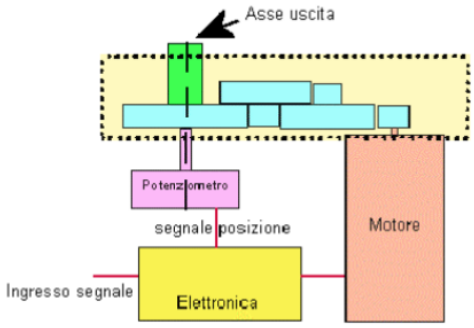


Figura 3.25 Schema funzionale del servo

Il segnale di controllo arriva all'elettronica, preposta per interpretarlo, che lo trasforma in un comando inviato al potenziometro che ruoterà il motore dell'angolo adatto.
Tramite una serie d'impulsi si può riuscire a determinare la posizione dell'asse di uscita; aumentando o diminuendo la durata di un impulso si avrà una riduzione o un aumento dell'angolo di rotazione del servo.
Un esempio di relazione tra durata d'impulso e rotazione solidale al potenziometro è mostrata in Fig.3.26.

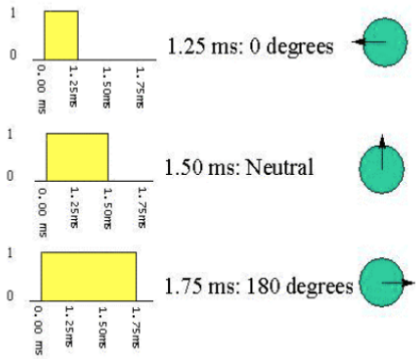


Figura 3.26 Esempio di relazione tra durata dell'impulso e rotazione del servo

B. **Data Sheet**

Vediamo ora le caratteristiche specifiche del servomotore *Futaba S3010*:

Basic Information

Modulation:	Analog
Torque:	**4.8V:** 72.0 oz-in (5.18 kg-cm) **6.0V:** 90.0 oz-in (6.48 kg-cm)
Speed:	**4.8V:** 0.20 sec/60° **6.0V:** 0.16 sec/60°
Weight:	1.45 oz (41.0 g)
Dimensions:	Length: 1.57 in (39.9 mm) Width: 0.79 in (20.1 mm) Height: 1.50 in (38.1 mm)
Motor Type:	3-pole
Gear Type:	Plastic
Rotation/Support:	Single Bearing

Figura 3.27 Informazioni base sul servomotore

Capitolo 4

Schema a Blocchi

Nei capitoli precedenti si è data importanza al funzionamento e all'analisi dei singoli componenti che caratterizzano la *Smart Car*, nel presente capitolo si vogliono prendere in considerazione le nozioni fin qui esposte e darne una visione d'insieme analizzando nel dettaglio le iterazioni tra i vari componenti del veicolo e le modifiche *hardware* che sono state ritenute necessarie per la *" messa in pista "* della *"macchinina"*.
Si utilizzerà una suddivisione in blocchi funzionali, così da creare una sorta di mappa concettuale del progetto.

4.1 Schema a Blocchi del Veicolo Freescale

A. **Struttura**

Vediamo ora una possibile rappresentazione in blocchi funzionali del processo che rende attuabile il movimento e il conseguimento dell'obiettivo Freescale:

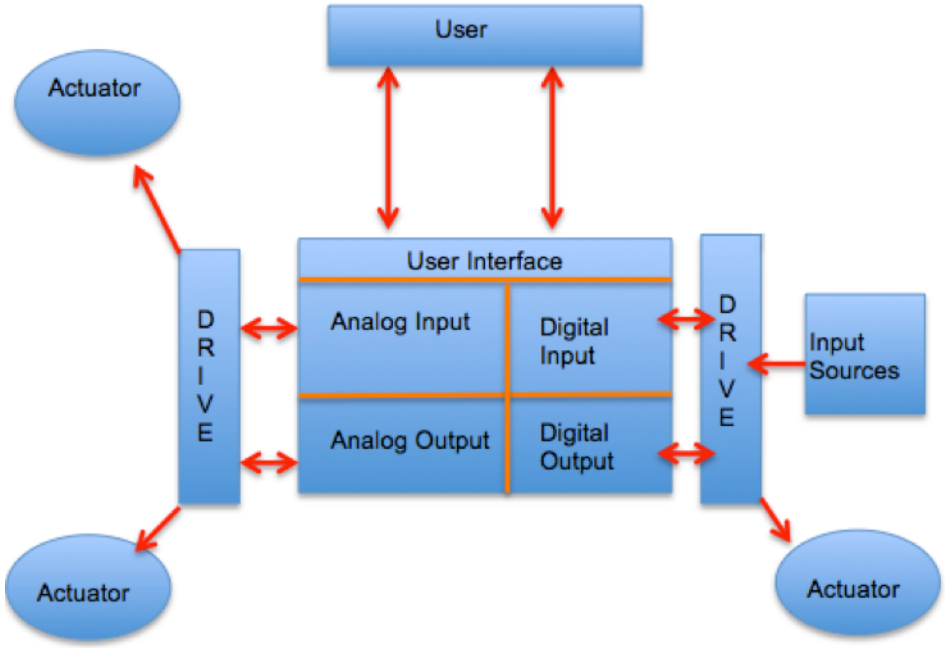


Figura 4.1 Rappresentazione in blocchi funzionali del sistema Freescale

Come si può notare da Fig.4.1, ciò che all'apparenza può sembrare un semplice "*veicoletto*" risulta un complesso sistema con svariate interazioni con l'ambiente circostante e con i propri componenti, sarà scopo dei prossimi capitoli esaminare dettagliatamente ognuno dei 8 blocchi funzionali presenti in questo schema; per ora ne daremo solamente una breve descrizione così da capire come si possa costituire un sistema che controlli la maggior parte delle variabili in gioco:

Actuator: Sono gli attuatori (Motori, servomotori) che se comandati nel modo corretto consentono al veicolo di aumentare o diminuire la velocità, di curvare e di stabilizzarsi in rettilineo o in curva.

Analog Input: Sono le sorgenti e i segnali di ingressi analogici
Analog Output: Sono le sorgenti e i segnali di uscita analogici
Digital Input: Sono i segnali e le fonti d'ingresso digitali
Digital Output: Sono i segnali e le fonti d'uscita digitali
Drive: Sono le schede che ricevono gli ingressi e le istruzioni per controllare gli attuatori
Input Sources: Sono i sensori che interfacciano il sistema con l'ambiente esterno
User Interface: è l'interfaccia che permette all'utente esterno di comunicare con il sistema
User: è l'utilizzatore finale del Veicolo.

4.2 Input Sources

A**. Caratterizzazione**

Ciò che caratterizza principalmente il sistema in esame è sicuramente l'insieme degli eventi esterni che interagiscono con il veicolo. Una *Smart Car*, per poter far fronte alle variazioni ambientali del tracciato durante la gara (*tunnel*, dossi, curve, curve ad "S", ecc…), deve potersi interfacciare con l'esterno tramite dei sensori (denominati *Input Sources*) ed analizzare determinati fenomeni per poter agire di conseguenza.
Come si è fatto presente nel regolamento esposto al capitolo 1, ogni gruppo concorrente può dotare il proprio mezzo di qualsiasi tipo di sensore, purché non si ecceda oltre un certo numero.
Gli unici sensori che abbiamo deciso di adottare sono state due telecamere; al paragrafo 4.2.D vedremo le motivazioni di tale scelta. Prima, però, è utile analizzare come l'interfaccia della macchinina dovrebbe comportarsi e quali siano le caratteristiche da implementare necessariamente via *software* (del corrispettivo codice si tratterà dettagliatamente nel capitolo 5).
Ora che abbiamo visto come funziona il sensore TSL1401CL, possiamo concentrarci a capire come interpretare le informazioni che questo ci invia. Come detto il sensore composto da 128 fotodiodi invia informazioni in funzione dell'intensità luminosa che ciascuna cella che lo compone riceve. Queste informazioni sono tradotte dall'ADC e dal sistema di elaborazione dati sotto forma di 128 numeri che variano da 0 a 255, in cui 0 equivale ad assenza perfetta di esposizione e 255 equivale a massima esposizione possibile; ovviamente questi due valori limite non sono quasi mai raggiunti durante il movimento del veicolo. Per poter seguire la linea sarà dunque necessario interpretare questi valori fornitici dalla nostra sorgente di *input* e distinguere quali dati appartengono alla linea nera e quali allo sfondo bianco.
Vi sono due metodologie di approccio diverso a tale problema:

1) Scansione mediante *threshold*

2) Scansione mediante metodo derivativo.

Prima di procedere alla descrizione formale dei due algoritmi di visione, risulta necessario evidenziare un punto debole dell'utilizzo delle telecamere come unica fonte di *input*: i *tunnel*.
Il tracciato di gara proposto dalla *Freescale*, infatti, prevede la presenza di *tunnel* in cui non vi sia luce e sotto cui la macchina dovrà sfrecciare senza fermarsi.
Per ovviare a tale inconveniente si possono adottare 2 metodologie:

1) Utilizzare dei sensori ad infrarossi che permettono la visione del tracciato indipendentemente dalle condizioni di luce ambientale;

2) Utilizzare una scheda *led* stabilizzata in modo da ovviare alla mancanza di luce naturale con una luce artificiale che, anche durante il percorso fuori dal *tunnel*, aiuti a rendere più nitido il contrasto tra linea nera e sfondo bianco del percorso di gara.

Essendo per noi studenti del secondo anno troppo complesso e, in termini di tempo, troppo lo studio per l'utilizzo dei sensori infrarossi, abbiamo deciso di optare per una semplice scheda *led* posta sotto la camera ed alimentata dalla batteria della vettura (come prevede il regolamento) in modo da illuminare il tracciato in qualunque condizione.
Riferendosi ad uno schema a blocchi classico dello studio dei controlli come quella rappresentato in Fig.4.2, possiamo affermare che, in questo caso, il nostro dev'essere un sistema a catena aperta dove il processo P è la lettura da parte della camera della luce del tracciato, i disturbi D sono i tunnel e le varie ombre proiettate sulla pista, il nostro regolatore R è uno dei due processi che andremo ad analizzare nei prossimi due paragrafi, l'input rappresenta la luce L mentre l'output O è il segnale da inviare al microcontrollore per identificare il tracciato.

Figura 4.2 Schema a blocchi del processo di acquisizione del tracciato mediante telecamera

B. **Scansione Mediante Threshold**

La prima tecnica di identificazione della linea è probabilmente quella più intuitiva ma non per questo la meno efficace.
Come detto precedentemente, la funzione rilevata mediante *DSO* proveniente dalle telecamere ha un andamento del tipo rappresentato in Fig.4.3.

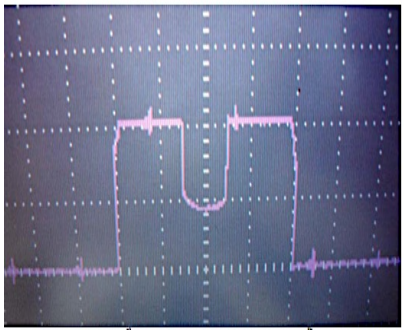


Figura 4.3 Segnale analogico proveniente dalla telecamera

La parte nera del tracciato, identificante la linea è il punto di minimo della funzione. Si potrebbe dunque pensare di inserire un valore limite (*threshold*) al di sotto del quale considerare ogni valore come nero e al di sopra del quale considerare tutti i valori bianchi. Il metodo, infatti, comincia tagliando l'inizio e la fine dell'*array* (arbitrariamente si è scelto di non considerare i primi 20 *bit* a sinistra e gli ultimi 20 *bit* a destra) così da evitare eventuali interferenze con lo sfondo del tracciato che potrebbe essere di qualsiasi colore; poi si prendono tutti i valori acquisiti dalla telecamera e si calcola la media. A questo punto viene settata una variabile temporanea, chiamata appunto *threshold*, contenete tale valore e si confrontano tutti gli 88 *bit* con detto valore partendo dal *bit* centrale (*bit* 44 che teoricamente, se la telecamera è stata centrata, dovrebbe corrispondere ad un punto della linea nera) e verificandone la superiorità o l'inferiorità numerica con il *threshold*.
A questo punto l'ultimo valore sotto il *threshold* a destra e a sinistra rappresenta univocamente il margine destro e sinistro della linea del tracciato.

Per comprendere meglio tale metodo, è utile ricorrere ad un esempio numerico:
Ipotizzando che la telecamera sia stata centrata con l'asse della vettura e assumendo per semplicità 10 *bit*, abbiamo i seguenti valori memorizzati nell'*array* acquisizione:

a[0]=14;a[1]=18;a[2]=19; a[3]=4;a[4]=5;a[5]=7; a[6]=12;a[7]=17;a[8]=21; a[9]=20

Il *threshold* risulterà quindi per definizione:

$$\frac{14+18+19+4+5+7+12+17+21+20}{10} = 13{,}7$$

impostato il valore del threshold, si parte dal *bit* centrale 4, si verifica che sia minore del *threshold* e si confrontano i valori a destra e a sinistra di tale *bit* con il *threshold* come segue :
a[3] <= threshold -> true; a[2] <= threshold -> false;
si nota che il *bit* 3 è il limite sinistro della linea.
a[5] <=threshold -> true; a[6] <= threshold -> true; a[7] <= threshold -> false
quindi il *bit* 6 rappresenta il limite destro della linea nera.
Questo metodo ha varie limitazioni in quanto molti valori sotto la media potrebbero non corrispondere a reali punti della linea del tracciato ma a ombre (create magari dalla macchina stessa).
Il pregio di tale algoritmo di controllo della posizione della linea è di essere adattativo, ossia indipendente dalle condizioni di luminosità in cui si sono fatte le acquisizioni.

C. **Scansione Mediante Metodo Derivativo.**

Il secondo metodo è probabilmente quello più complicato concettualmente e da implementare in C, però risulta il metodo più affidabile e corretto.
Tutto consiste nel pensare alla forma d'onda generata dall'acquisizione dei segnali della telecamera (Fig.4.4.a):

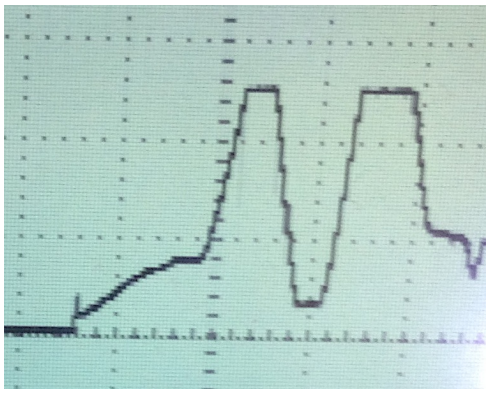


Figura 4.4.a Segnale analogico proveniente dalla telecamera

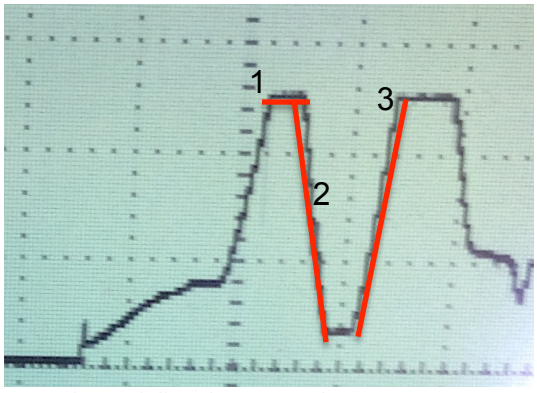


Figura 4.4.b Segnale analogico proveniente dalla telecamera, in rosso sono state evidenziate le derivate della funzione nei punti caratteristici

Come si può notare dalla Fig.4.4.b un modo del tutto equivalente al ragionamento precedente è quello di considerare le derivate puntuali della funzione (evidenziate in rosso). Basterà, infatti, (dopo aver tagliato i *bit* iniziali e finali per gli stessi motivi spiegati nel paragrafo precedente) individuare dove la derivata è nulla (1), dove è negativa (2) e dove è positiva (3), secondo questo pattern per trovare gli estremi e il centro della linea.
Seppure molto efficace sulla carta, perché permette di tollerare un maggior rumore sulla funzione e permette di non essere influenzato dalle condizioni luminose del tracciato, questo metodo presenta un difetto intrinseco: le derivate. Quando si deve implementare tale strumento matematico tramite *software*, infatti, occorre ricorrere ad un'approssimazione della derivata tramite la sua definizione: limite del rapporto incrementale. Così facendo i valori che si ottengono, e di conseguenza la visione del tracciato, risentono molto delle brusche variazioni luminose e dà un grande peso anche alle piccole quantità di rumore.
Per applicare questo metodo, dunque, occorre ricorrere ad un filtraggio ad esempio ad una media pesata di tre valori.
Si può quindi facilmente notare che usando questo algoritmo visivo si perderanno molte informazioni dal punto di vista del segnale.

D. **La Scelta delle Due Telecamere**

Concludiamo il paragrafo dedicato alle "*Input Sources* " spiegando la principale aggiunta sensoristica fatta durante la preparazione alla competizione.
Uno dei problemi derivanti dal fatto di avere una telecamera lineare come sensore di visione è certamente quello di doverla obbligatoriamente puntare in un range abbastanza vicino al veicolo in modo da poter seguire la linea e le sue variazioni in maniera quasi istantanea. Ciò pregiudica il fatto di poter prepararsi tempestivamente ad un'eventuale curva e quindi limita moltissimo le prestazioni in fase di gara del veicolo. Uno dei metodi per ovviare a questo inconveniente è quello di aggiungere un'altra telecamera in modo da usarne una per la visione ravvicinata e quindi l'inseguimento preciso della linea e una per fare dei modelli previsionali sull'andamento del tracciato. In questo modo si potranno calcolare: raggi di curvatura, velocità di curvatura, velocità in rettilineo frenature ecc. Unico inconveniente della scelta delle due telecamere è l'aumento del tempo di ciclo; infatti essendo tale sensore il più lento a livello *hardware*, è proprio quest'ultimo a determinare la velocità di esecuzione del programma. Facendo alcune prove, però, si è notato che fortunatamente l'aggiunta di un'altra telecamera non aggravava di molto la situazione.

4.3 Output

A**. Caratterizzazione**

Come detto più volte durante i paragrafi precedenti, un sistema di controllo è caratterizzato da un segnale di *input* proveniente dal mondo esterno che viene elaborato e interpretato dal sistema che, successivamente, agirà su dei dispositivi interagenti con l'esterno.
Questi dispositivi su cui il sistema di controllo agisce in risposta agli ingressi vengono chiamati in questo elaborato *Output*.
Un *output* dunque non è altro che un dispositivo (motore, servomotore) che riceve un *input* dal controllore e reagisce in un determinato modo (aumento/diminuzione di velocità, sterzata a destra/sinistra).
Per visualizzare tali concetti risulta utile ricorrere ancora una volta ad uno schema a blocchi di questo tipo:

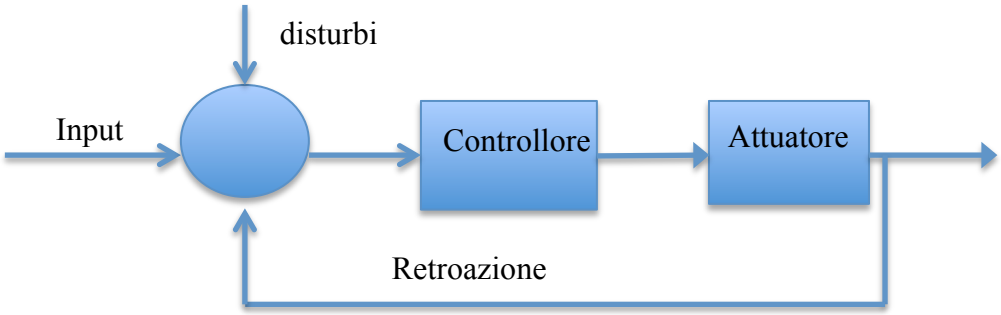


Figura 4.5 Schema a blocchi dell'interazione input/output nel veicolo

- per *input* si intendono i segnali di ingresso provenienti dai sensori descritti nel paragrafo precedente (telecamera);
- i disturbi sono quei segnali provenienti dall'ambiente esterno ed interno al sistema (rumore luminoso, rumore elettromagnetico, rumore termico, rumore *flicker*) che disturbano il segnale in ingresso;
- il controllore è il cuore del sistema, è ciò che interpreta i segnali di *input* e decide come reagire a tale sollecitazione esterna.
- gli attuatori sono gli organi preposti ad attuare le direttive del controllore.

Abbiamo già visto i segnali di *input* come vengono generati e come vengono trasmessi al controllore, non resta ora che concentrarsi su come il controllore debba regolare gli *output* ed in quali casi debba intervenire con delle correzioni.

Principalmente nel sistema *Freescale* abbiamo 2 tipologie di *output* :

1) I due motori: gli organi preposti alla variazione della velocità e alla frenatura dinamica del veicolo nonché al comportamento differenziale in curva.

2) Il servomotore: l'organo preposto ad aumentare o diminuire il raggio di curvatura del veicolo e alla direzione di tale azione.

Dallo schema a blocchi qui presentato, si può notare come il sistema sia in catena chiusa e presenti una retroazione.
Tale retroazione è dovuta ad una variazione degli attuatori che provoca una variazione della sorgente di *input*.
Ad esempio: se la sorgente di *input* invia un segnale al controllore in cui trasmette l'informazione che il veicolo si trova decentrato rispetto alla linea, il controllore agirà sugli attuatori preposti (servomotore in questo caso) in modo da centrare l'asse del veicolo alla linea, quindi agirà in modo da ridurre il più possibile l'errore tra la grandezza in ingresso e la sua corrispettiva uscita.
Il fatto che il sistema sia in catena chiusa è un aspetto fondamentale: infatti ciò consente di avere una maggiore stabilità nei transitori.
Vedremo ora gli *output* ed il loro controllo così da dare un'idea qualitativa e quantitativa di come si possano risolvere alcune problematiche legate alla loro regolazione.
Successivamente sarà analizzata una modifica *hardware* degli *output* motori essenziale per la diminuzione del rumore elettromagnetico in entrata e un'altra modifica essenziale per permettere le frenate attive al veicolo.

B. **Adattamento Potenza Motori**

Ciò che distingue maggiormente una *Smart Car* da un qualsiasi dispositivo *line follower* è sicuramente la velocità di esecuzione del tracciato.
Nel nostro caso il veicolo riesce a raggiungere una velocità massima di ben 3 [m/s] grazie ai due motori in corrente continua già analizzati nel capitolo 3.
Come in un qualsiasi veicolo da corsa che si rispetti, la velocità deve essere variata sia nel caso di un rettilineo, in cui l'auto dovrà andare a potenza massima, sia nel caso di curve in cui, non essendoci differenziali meccanici, la vettura dovrà essere in grado di aumentare la potenza ad un motore e diminuire quella dell'altro.
Per fare ciò è necessario adottare un controllore che regoli in *Pwm* la potenza erogata *dall' H-bridge* ai motori.
Ovviamente a seconda del tratto di tracciato individuato dalla telecamera, tale controllore dovrà dare la giusta *Pwm* a ciascun motore.
Ipotizzando, ad esempio, una struttura di questo tipo per il processo da controllare (trascurando i disturbi che verranno analizzati nel dettaglio al paragrafo 4.3.C) si avrà uno schema funzionale come il seguente:

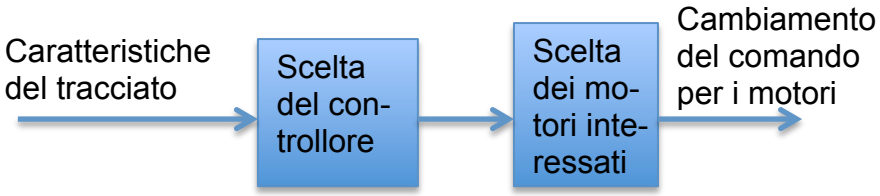


Figura 4.6 Schematizzazione del processo di adattamento della potenza motori

Si nota che si possono verificare principalmente tre casi ideali:

1) Caratteristiche del tracciato = Rettilineo
   Scelta del controllore = Tipo P
   Scelta dei motori interessati = Entrambi.
   Cambiamento potenza ai motori = Massima (nello specifico 999 valore oltre al quale si ha saturazione della potenza motrice.)

2) Caratteristiche del tracciato = Curva a sinistra
   Scelta del controllore = Tipo PID
   Scelta dei motori interessati = sinistro
   Sistema differenziale.
   Cambiamento potenza ai motori = Aumentare potenza del motore destro e mantenere o ridurre la potenza del motore sinistro.

3) Caratteristiche del tracciato = Curva a destra
   Scelta del controllore = Tipo PID
   Scelta dei motori interessati = sinistro
   Sistema differenziale.
   Cambiamento potenza ai motori = Aumentare potenza del motore sinistro e mantenere o ridurre la potenza del motore destro.

Sono stati trascurati come trattazione i disturbi e la possibilità di avere una frenata attiva in quanto questi due argomenti sono strettamente correlati a delle modifiche *hardware* che verranno trattate, per chiarezza espositiva, nei prossimi paragrafi.

Si nota, dunque, che il sistema *Freescale* è un sistema dinamico, cambia infatti con il variare della posizione del veicolo lungo il tracciato; un buon sistema di controllo, infatti, deve poter agire quasi istantaneamente, se non riuscire a predire, ad eventuali variazioni del tracciato così da evitare l'insorgere di spiacevoli instabilità o di ritardi nell'azione di contenimento dell'errore tra la potenza dei motori prevista e quella reale.

C. **Inserimento Capacitori tra i Motori**

Una delle principali modifiche *hardware* che si sono rivelate fondamentali durante la progettazione del veicolo è stata l'inserimento di tre capacitori da 100 [nF] tra i due motori per avere un effetto filtro.
Per spiegare cos'è stato fatto in sostanza possiamo iniziare con il vedere da cos'è nato il bisogno di questa modifica.
Come spiegato nei precedenti paragrafi, il sistema *Freescale* è un sistema basato su una sensoristica di tipo visivo, dove (nel nostro caso) gli unici sensori di interfaccia con il mondo esterno sono proprio le due telecamere montate; ogni disturbo ai segnali provenienti dalle telecamere risulta quindi fondamentale per la stabilità del sistema.
In fase di *test* di velocità del veicolo si è notato come per valori elevati di *Pwm*, la nitidezza del segnale proveniente dalla camera fosse inversamente proporzionale alla velocità.
Leggendo su vari forum, abbiamo scoperto che oltre un determinato valore di *Pwm* il campo magnetico generato dal sistema dei *DC motors* interferiva con i segnali provenienti dalla camera. Inizialmente abbiamo optato solo per l'utilizzo di cavi schermati così da ridurre il rumore il più possibile.
Questo ha sicuramente migliorato il segnale ma non in maniera così efficace come ci aspettavamo.
Siamo allora ricorsi all'inserimento di un filtro capacitivo di tipo passa basso nell'interfaccia motori ponte H come mostrato nelle Fig.4.7 e 4.8.

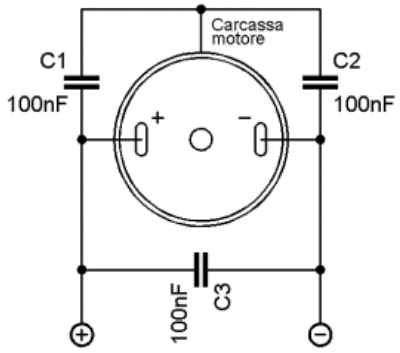


Figura 4.7 Schematizzazione del filtro applicato ai motori

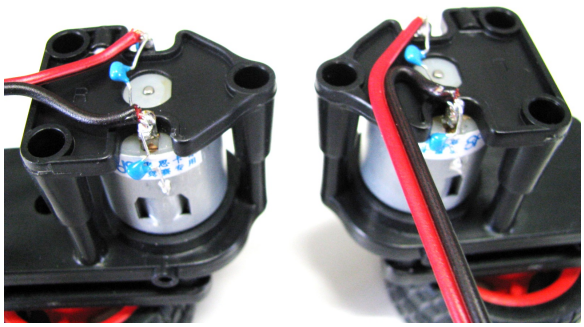


Figura 4.8 Collegamento effettivo del filtro

Questa modifica ha permesso di aumentare la *Pwm* massima attribuibile ai motori (specialmente in rettilineo), senza precludere la pulizia e la veridicità del segnale proveniente dalla telecamera.

D. **Modifica per Frenata Attiva**

Assieme alla precedente modifica *hardware* è stato necessario dotare il veicolo di un sistema di frenata.
Questo sistema risulta infatti fondamentale per affrontare il tracciato alla massima velocità in ciascun punto.

Infatti, implementando un buon algoritmo di visione e di previsione del tracciato si può sfruttare la frenata attiva per passare da una condizione di rettilineo, caratterizzata da *Pwm* uguale e massima su entrambi i motori, ad una condizione di curva, in cui si potrebbe idealmente arrivare in una situazione in cui i due motori forniscono potenza in direzioni opposte così da far sterzare il veicolo con più energia e con velocità maggiore.
Oltre a questo aspetto, risulta fondamentale adottare un sistema di frenatura per permettere al veicolo di rispettare una delle condizioni del regolamento: fermarsi entro 1000 $[mm]$ dalla linea d'inizio.
Apparentemente appena dotati del veicolo tale funzione sembrava già possibile, in quanto i *data sheets* fornitici indicavano che il *pin IN1* (responsabile assieme al *pin IN2* del verso di rotazione dei motori) fosse collegato al microcontrollore.
Dall' analisi di *data sheets* reperiti più recentemente si è notato come tale *pin* fosse stato collegato inavvertitamente a massa anziché al microcontrollore.
È stato dunque necessario sollevare i due *pin* ed inserirvi due fili (quello rosso e quello blu in Fig. 4.9) per consentire di invertire il moto indipendente dei motori con le tecniche illustrate nel capitolo 3.

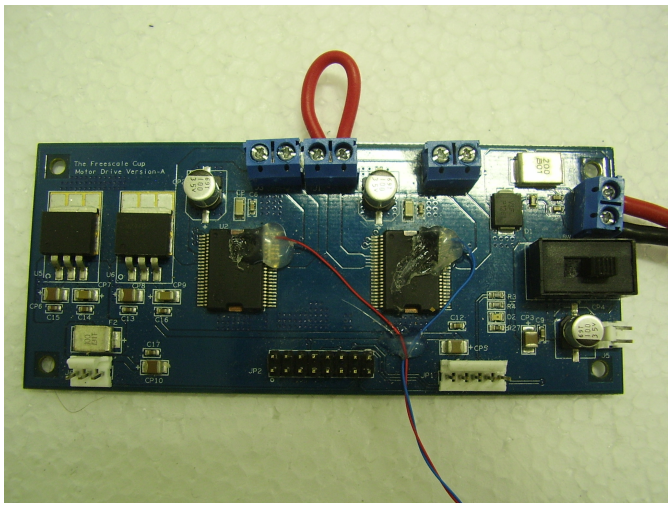


Figura 4.9 Scheda motori con relativa modifica

La combinazione di segnali alti e bassi nei *pin IN1 e IN2* (evidenziati anche in Fig.4.10) permette di controllare il verso di ciascun motore e quindi (tramite *Pwm*) consente di fermare e far decelerare il veicolo.

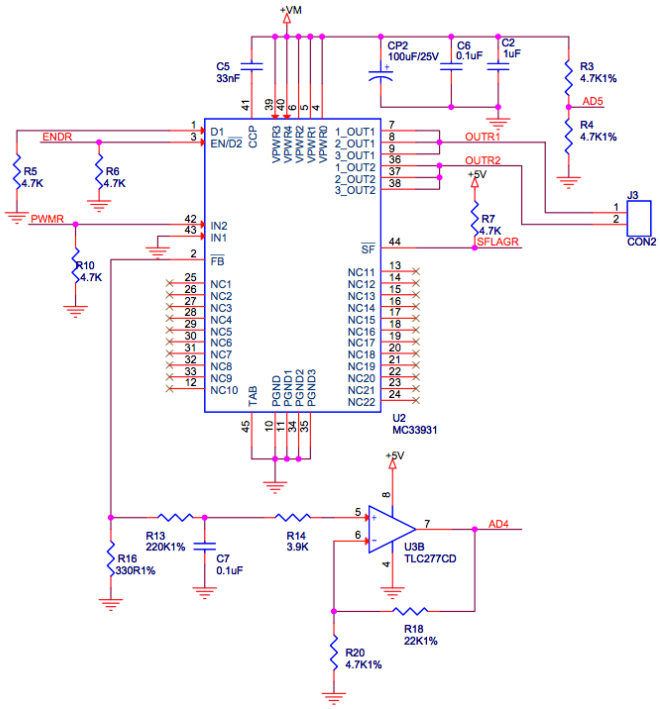


Figura 4.10 Schema di collegamento di IN1 e IN2

Le combinazioni dei segnali da inviare ai *pin* sono le seguenti :

1) Segnale *Low* come *tensione di* alimentazione: entrambi i motori rimangono fermi senza fare resistenza al movimento (possono essere usati per il trascinamento d'inerzia).

2) Segnale *High* di alimentazione e *High* agli *enable* 1,2 dando segnale alto all'*analog input Pa[0]* e basso al *Pa[1]* il motore destro muove nel verso frontale (avanti) della vettura.

3) Segnale *High* di alimentazione e *High* agli *enable* 1,2 dando segnale basso all'*analog input Pa[0]* e alto al *Pa[1]* il motore destro muove nel verso opposto alla vettura, si ha il così detto *reverse*.

4) Segnale *High* di alimentazione e High agli *enable* 3,4 dando segnale alto all'*analog input Pb[0]* e basso al *Pb[1]* il motore sinistro muove nel verso frontale (avanti) della vettura.

5) Segnale *High* di alimentazione e *High* agli *enable* 1,2 dando segnale basso all'*analog input Pb[0]* e alto al *Pb[1]* il sinistro muove nel verso opposto alla vettura, si ha il così detto *reverse*.

6) Segnale *High* di alimentazione e *High* agli *enable* 1,2 ed agli *enable* 3,4 dando segnale alto all'*analog input Pa[0]* e al *Pb[0*] e basso al *Pa[1] e al Pb[1]* entrambi i motori muovono nello stesso verso.

7) Segnale *High* di alimentazione e *High* agli *enable* 1,2 ed agli *enable* 3,4 dando segnale basso all'*analog input Pa[0]* e al *Pb[0]* e alto al *Pa[1]* e al *Pb[1]* entrambi i motori muovono nello stesso verso opposto al punto precedente.

8) Segnale *High* di alimentazione e *High* agli *enable* 1,2 ed agli *enable* 3,4 dando segnale basso all'*analog input Pa[0],Pb[0], Pa[1]* e *Pb[1*] si ha la stessa condizione del punto 1.

Tutte queste combinazioni vengono riassunte per semplicità in Tab.4.1.a e 4.1.b.

H	x	H	X	X	0	1	The left engine of the car began to move backwards which causes the car turn right in reverse without using the servo.
H	H	H	1	0	1	0	Both motors rotate in the same direction to generate the carriage moves in a straight direction, to make the car move perfectly in a straight line should be to calibrate the main shaft of the servomotor.
H	H	H	0	0	0	0	Both engines, the left and the right to remain static but no resistance to movement, that is, they can make use of inertia

Tabella 4.1.a Riassunto della logica di comando

Vcc1	Enable 1,2	Enable 3,4	1A (PA[0])	2A (PA[1])	3A (PB[0])	4A (PB[1])	Function
L	x	x	x	x	x	x	Both engines, the left and the right to remain static but no resistance to movement, that is, they can make use of inertia
H	H	X	1	0	X	X	The right engine moves in front of the car, this is to make a left turn without using a servomotor.
H	H	x	0	1	x	x	The right engine moves the car back, this causes a turn is made in reverse to the left
H	x	H	X	X	1	0	The left engine of the car began to move forward which causes the car turn right without using the servo

Tabella 4.1.b Riassunto della logica di comando

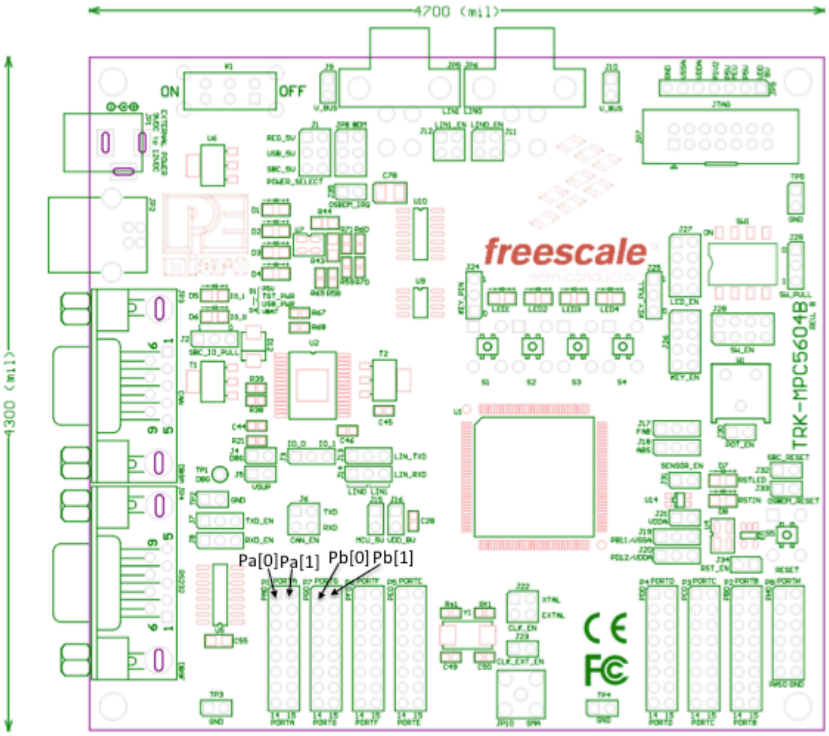


Figura 4.11 Collocazione dei pin di input Pa[0],Pa[1],Pb[0],Pb[1]

E. **Adattamento Sterzata in Funzione del Tracciato**

Dopo aver visto come un controllore dovrebbe adattare la potenza dei motori in funzione del tracciato, vediamo ora come dovrebbe essere strutturato un controllore per regolare l'angolo di sterzo della vettura.
Facciamo anzitutto riferimento al seguente schema del processo da controllare:

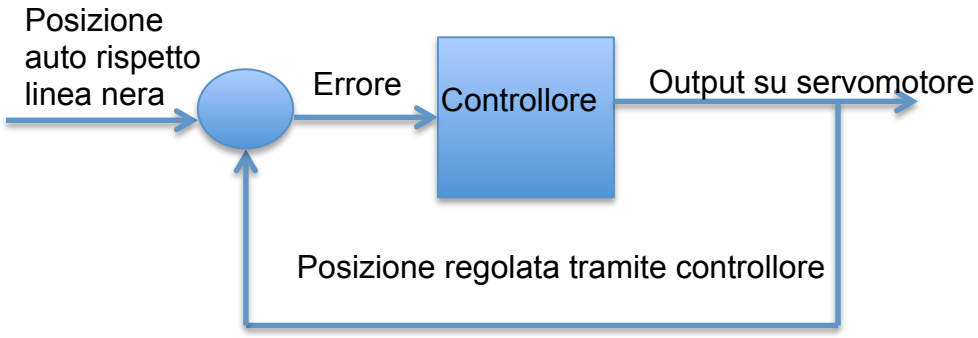


Figura 4.12 Schema del processo di sterzata

Notiamo anzitutto che il sistema da controllare è in retroazione.
Prima di procedere alla spiegazione di come dovrebbe agire il controllore, risulta utile osservare che il servomotore è un dispositivo meccanico altamente sensibile, anche un minimo "*gioco*" dei suoi componenti interni in una qualsiasi fase del processo potrebbe provocare una cattiva calibrazione dell'attuatore.

Tenendo presente ciò, da prove sperimentali, si è arrivati a ricavare i valori di *Pwm* da dare allo sterzo che sono:

Posizione centrale: 2200;

Posizione completamente a destra: 2315;

Posizione completamente a sinistra: 2085;

Il principio di funzionamento è abbastanza semplice ed è riassumibile come segue:

1) Si sceglie il *bit* centrale della telecamera (tenendo conto di eventuali tagli) come *bit* di riferimento per la posizione centrale del veicolo in rettilineo e si memorizzano i valori dei *bit* rappresentativi: il limite destro e sinistro della linea nera.

2) Il successivo ciclo si guarda dove sono posizionati i 3 *bit* :

   2.a) Stessa posizione allora l'errore è nullo, il controllore non deve intervenire

   2.b) *Bit* spostati di una certa quantità x positiva (supposto errore nullo veicolo centrato) auto decentrata a destra, il controllore (tipo PID) dà un segnale proporzionale all'errore allo sterzo per farlo ruotare di una certa quantità y verso sinistra.

   2.c) *Bit* spostati di una certa quantità x negativa (supposto errore nullo veicolo centrato) auto decentrata a sinistra, il controllore (tipo PID) dà un segnale proporzionale all'errore allo sterzo per farlo ruotare di una certa quantità y verso destra.

Oltre al già citato problema di *wind up* (capitolo 2), è da tenere presente il fatto che valori oltre i limiti (2315 e 2085) causerebbero danni irrimediabili allo sterzo.
Durante l'implementazione del controllore PID, infatti, risulta necessario ricorrere a delle condizioni di verifica sull'uscita in modo da evitare il superamento di tali limiti.
Assieme a queste condizioni di verifica, è necessario avere un algoritmo abbastanza stabile, per evitare eventuali oscillazioni continue dello sterzo che potrebbero causare l'uscita di strada del veicolo (instabilità).
Se ad esempio il controllore avesse bisogno di ruotare lo sterzo di un valore pari a 2800, dovremmo fare in modo che dia come valore 2315, in tal modo però l'errore non tornerebbe nullo nel tempo previsto ma probabilmente avremo bisogno di più cicli per tornare ad una condizione di stabilità.
Uno dei maggiori difetti di questo metodo è sicuramente il facile insorgere di instabilità causato dalla restrizione dell'*output* del PID appena descritta.
Infatti, tale metodo di restrizione rischia di far oscillare l'errore con ampi valori positivi e negativi.
Una soluzione possibile a tale problema risulta quella di contare il numero di sterzate sopra o sotto un determinato range (ad esempio sopra 2250 e sotto 2150) per cicli vicini.
Ad esempio se per 6 cicli consecutivi ci sono state 10 sterzate con valore 2300 e 10 sterzate con valore 2100 sicuramente la macchina è in fase di instabilità, a tal punto si può scegliere di azzerare manualmente l'errore settandolo a zero in modo da recuperare stabilità e dare un valore di sterzo neutro ad esempio quello della posizione centrale (2200).

Capitolo 5

Implementazione Software

Questo capitolo conclude la trattazione sull'esperienza *Freescale*; fino ad ora si è focalizzata l'attenzione sull'*hardware* che necessita di controllo e, in particolare nel precedente capitolo, si è analizzata la logica del controllore da implementare per ogni singolo componente. Si passerà, adesso, a esaminare da un punto di vista implementativo, tramite sintassi C, tali controllori analizzandone nel dettaglio le singole funzioni.
Si è scelto di riportare solamente le parti di *software* che interessano i controllori già citati nel capitolo 4 sia per chiarezza espositiva sia per non esulare dallo scopo di questo elaborato.
Eventuali funzioni chiamate all'interno dei controllori ma non dichiarate, verranno descritte brevemente nell'introduzione dei singoli paragrafi.
Prima di iniziare la trattazione ci si vuol soffermare un istante sul perché si è scelto di riservare un intero capitolo alla parte *software*. Tale decisione è stata presa perché in fase di stesura si è voluto dare più importanza alla parte concettuale e alle definizioni di concetti utili anche al di fuori dell'esperienza *Freescale*. La parte *software* è stata, forse, l'argomento principale di tutto il lavoro ma anche quella più soggettiva, che ogni gruppo ha implementato in modo diverso e come meglio ha creduto.

5.1 Visione

A. **Introduzione**

Abbiamo già visto nel capitolo 3 che il sensore *TSL1401CL* è l'unico sensore di interfaccia con il mondo esterno e quello deputato alla visone del tracciato. Si ricorda che il sensore invia in modo digitale al microcontrollore un *array* di 128 numeri che variano da 0 a 255.
In questo paragrafo si farà riferimento al solo algoritmo di individuazione della linea nera e di tipologia di tracciato (rettilineo o curva), mentre nei successivi verrà descritta l'implementazione degli algoritmi per affrontarne le variazioni.
Prima di passare a questa parte è necessario definire brevemente una funzione che viene chiamata nell'algoritmo di visione ma che non verrà riportata per i motivi sopraelencati.
La funzione "*Camera(void)"* è una funzione di tipo *void* che quindi non restituisce alcun valore e, inoltre non richiede parametri di ingresso, in quanto attua ciò che è stato descritto nel paragrafo 3.1.A del capitolo 3 per 128 cicli riempiendo mano a mano due *array*: uno per la camera bassa ed uno per la camera alta, con tutti i valori ricevuti trasformati in numeri. I due *array* sono rispettivamente: *Result_Camera_Sotto[] e Result_Camera_Sopra[]*.
È da notare che il nome delle variabili è stato scelto appositamente più lungo del necessario per rendere più chiara la sua funzione all'interno del programma.
Riportiamo di seguito lo schema del processo, già visto nel capitolo 4, per chiarezza espositiva:

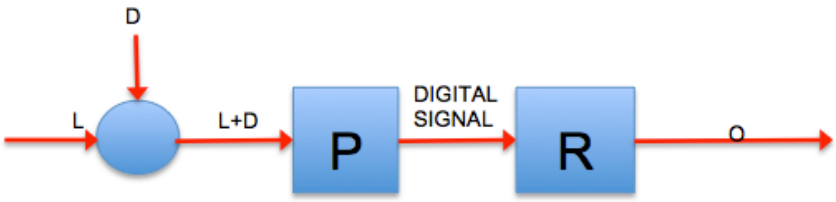


Figura 5.1 Schema a blocchi del processo di acquisizione del tracciato mediante telecamera

Si ricorda inoltre che le tipologie di controllore sono 2 :
-mediante derivazione
-mediante media
Si farà riferimento all'implementazione di entrambe così da evidenziarne dal punto di vista *software* pregi e difetti. Essendo lo stesso algoritmo di visione sia per la camera bassa, che per quella alta (tra le due cambierebbe solo l'*array* da *Result_Camera_Sotto[]* a *Result_Camera_Sopra[]*) ne esamineremo solo uno dei due.

B. **Implementazione dell'Algoritmo per la Visione tramite Metodo Derivativo:**

```
int Visione_Camera_Bassa (int posizione ,int* Result_Mobile) //definizione della funzione per la
                                                              //camera bassa che ritorna la
                                                              //posizione del centro
{
        double deltaY, deltaX, derivata, posizione_derivata_Positiva, posizione_derivata_Negativa;
        /*dichiarazione delle variabili statiche*/
        int centro = 0 ;
        int media =  0;
        int i;

        double derivata_massima_Positiva = 10;//Setto dei valori iniziali per la derivata positiva

        double derivata_massima_Negativa = -10; //Setto dei valori iniziali per la derivata negativa

       for(i=posizione; i<posizione+BANDA; i++)/*BANDA è una variabile globale di tipo int  che
                                                 indica quanti digit dei 128 acquisiti   sono
                                                 significativi ; è settato sperimentalmente a 97*/
        {
                //Trovo il valore massimo della derivata positiva e di quella negativa.

                    deltaY = Result_Camera_Sotto[i+3]-Result_Camera_Sotto[i];

/*uso la definizione di derivata come limite del rapporto incrementale su 3 elementi così da avere
miglior approssimazione della derivata */

                deltaX = 4;

                derivata = deltaY/deltaX;

                if(derivata> derivata_massima_Positiva)
                {

                        derivata_massima_Positiva = derivata;

                        posizione_derivata_Positiva = i+1;//Memorizzo la posizione dell'array in cui
                                                          //si trova l'elemento con la derivata
                                                          //massima positiva
                }
```

```
            else if(derivata< derivata_massima_Negativa)
            {

                derivata_massima_Negativa = derivata;

                posizione_derivata_Negativa = i; //Memorizzo la posizione dell'array in cui si
                                                 //trova l'elemento con la derivata massima
                                                 //negativa

            }

            media = media + Result_Camera_Sotto[i];

        }


        media = media/BANDA; /* uso media come una sorta di threshold per capire quali bit sono
                                neri (<media )e quali bianchi (>media)*/

            //Trovo il centro della linea

        centro = (int) (posizione_derivata_Negativa+ posizione_derivata_Positiva)/2;



if(centro > 112 || centro <15 || Result_Camera_Sotto[centro]>media || deriv-
ta_massima_Positiva==10 || derivata_massima_Negativa == -10)
/*verifico se la telecamera ha visto la linea oppure se (in curva ad esempio) sta puntando fuori dal
tracciato*/
        {

            centro = *Result_Mobile; //centro è una variabile globale che viene
                                     //aggiornata ad ogni ciclo

            nonCiVedo = nonCiVedo + 1;

/*nonCIVedo è una variabile globale che mi dice se la camera sta puntando la linea ( valore 1) oppu-
re no ( valore 0)*/

            ciVedo_Camera_Bassa = 0;/*La camera non sta vedendo la linea
                                      (nonCiVedo=1,ciVedo_Camera_Bassa =0)*/
        }

        Nuovo_Centro = centro;

centro = *Result_Mobile;

 /* aggiorno il valore del vecchio centro così in caso la telecamera non ci vedesse al prossimo ciclo
potrò usare questo valore per  determinare l'andamento del
tracciato.*/
        return centro; //ritorno la posizione dell'array in cui si trova il centro della linea.

}
```

C. **Implementazione dell'Algoritmo per la Visione tramite Metodo della Media:**

```
void Visione_Camera_Alta(void)// definizione funzione per algoritmo di visione tramite  media
{
        int max = Result_Camera_Alta[0];//setto il valore massimo al primo valore dell'array

        int min = Result_Camera_Alta[0];//setto il valore minimo al primo valore dell'array

        int threshold_Camera_Alta=0;//inizializzo il livello di threshold per la camera alta

        int k1, k2, j1;//inizializzo i contatori

        stato_precedente_di_nonCiVedo = nonCiVedo;//memorizzo in una variabile globale lo stato
                                                  //attuale della visione

        limite_destro_Camera_Alta=0;//inizializzo il valore iniziale del limite destro della linea

        limite_sinistro_Camera_Alta=0;//inizializzo il valore iniziale del limite sinistro della linea


        for(i=0; i<128; i++)//trovo il massimo e il minimo dei valori acquisiti dalla camera alta
        {
                if(max<Result_Camera_Alta[i])
                {
                        max =Result_Camera_Alta[i];
                }
                if(min>Result_Camera_Alta[i])
                {
                        min = Result_Camera_Alta[i];
                }
        }

threshold_Camera_Alta=(int)((max+min)/2);  /*setto il threshold (variabile globale) come la media del
                                             massimo e del minimo*/
          for(k1=52, k2=53; k1>25, k2<93; k1--, k2++) /*determino centro ed estremi della linea fa-
cendo partire dal centro dell' array le 2 variabili k1 e k2 (si sono esclusi i primi e gli ultimi 25 bit)*/
          {

if(Result_Camera_Alta[k1]<threshold_Camera_Alta&&Result_Camera_Alta[k2]<threshold_Camera
_Alta && (k1-k2)<larghezza_linea_Camera_Alta)
/* caso che entrambi siano neri subito, larghezza_linea_Camera_Alta è un valore misurato speri-
mentalmente che quantifica quante
posizioni dell'array dovrebbe occupare l'intera linea : 20 */

    {


//setto i confini della linea e li memorizzo su variabili globali accessibili a tutti gli altri metodi
```

```
                    for(j1=k1; Result_Camera_Alta[j1]<threshold_Camera_Alta; j1--)
                    {
                            limite_sinistro_Camera_Alta=j1;
                    }
                    for(j1=k2; Result_Camera_Alta[j1]<threshold_Camera_Alta; j1++)
                    {
                            limite_destro_Camera_Alta=j1;
                    }
                    goto LABEL_Settaggio_Visione;/*settati entrambi i limiti vado direttamente
alla parte di programma che setta il nonCiVedo e il CiVedo per la camera Alta evitando di ciclare
inutilmente*/
            }

            if(Result_Camera_Alta[k1]<threshold_Camera_Alta)
            {
                    limite_destro_Camera_Alta=k1;
                    for(j1=k1; Result_Camera_Alta[j1]<thr; j1--)
                    {
                            limite_sinistro_Camera_Alta=j1;
                    }
                    goto LABEL_Settaggio_Visione;
            }

            if(Result_Camera_Alta[k2]<threshold_Camera_Alta)
            {
                    limite_sinistro_Camera_Alta=k2;
                    for(j1=k2; Result_Camera_Alta[j1]<threshold_Camera_Alta; j1++)
                    {
                            limite_destro_Camera_Alta=j1;
                    }
                    goto LABEL_Settaggio_Visione;

            }

            limite_destro_Camera_Alta = 100; limite_sinistro_Camera_Alta = 1;
            /*solo nel caso in cui non riesca, ad entrare in nessuno degli if*/
          //facendo così non verrà modificato il valore centrale nel prossimo if
    }


LABEL_Settaggio_Visione:
if (limite_destro_Camera_Alta - limite_sinistro_Camera_Alta < larghezza_linea_Camera_Alta)
        {
posizione_centro_linea_camera_alta = (int)((limite_destro_Camera_Alta + limite_sinistro_Camera_
Alta)/2) +5;
/*+5 ( misurato sperimentalmente) perchè la camera legge il  centro con un certo offset a causa del-
la pendenza dei supporti/*
            nonCiVedo=0; // la setto a 0 perché ho visto la linea
            contatore = contatore + 1; /* contatore usato nella funzione di sterzata per evitare
                                          che in incroci o altro la macchina segua le linee errate*/
        }
/*essendo centrale una variabile globale nel caso in cui non entri in un if essa rimane quella del ciclo
precedente in quanto non viene modificata*/
```

```
        else
        {
                nonCiVedo=1; // non ho visto la linea
                contatore = 0;// riazzero il contatore quando smetto di vedere
        }
}
```

Notiamo, prima di concludere, che il secondo metodo (probabilmente più macchinoso) non restituisce un valore ma modifica solamente delle variabili globali.
Il primo metodo risulta molto più rapido e di facile comprensione tuttavia risente in modo maggiore della presenza di rumori e necessita, quindi, di un precedente filtraggio del segnale. Il secondo metodo non ha questo inconveniente ma risente delle imprecisioni e sollecitazioni meccaniche agenti sui supporti su cui si posizionano i sensori.

5.2 Servo

A. **Introduzione**

Completata la parte relativa alla visione, passiamo alla prima delle due parti che vengono maggiormente interessate dal problema del controllo : il Servo.
Come già descritto, il seguente algoritmo serve per calibrare e direzionare la sterzatura del veicolo in funzione dei parametri acquisiti con i sistemi di visione.
Si riporta ora uno schema esemplificativo del programma, così da poterne avere una più chiara visione una volta analizzato il codice:

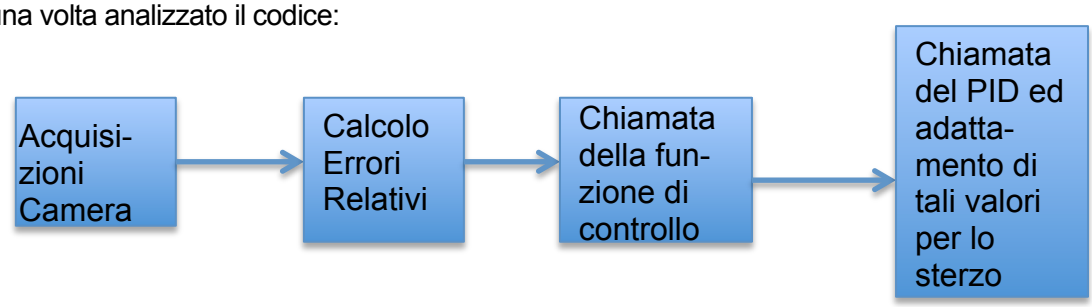


Figura 5.2 Visualizzazione del programma mediante blocchi funzionali

È fondamentale prima di leggere il codice tenere presente le seguenti osservazioni :

1) Eventuali parti del codice non significative per il controllo dello sterzo sono state volontariamente omesse;

2) Tutto il programma si basa sul fatto che si è scelta la posizione 63 degli *array* derivanti dall'algoritmo di visione come origine del sistema di riferimento. A tale posizione, infatti, si è scelto di dare la condizione di allineamento della macchina al tracciato;

3) Facendo riferimento allo schema a blocchi del capitolo 4.3 paragrafo E, l'errore utilizzato nel controllo è proprio la differenza tra il centro calcolato ad ogni ciclo dalla camera e la posizione di riferimento;

4) Nel calcolo del PID si è tenuto conto dell'effetto della saturazione della parte integrale;

5) Si è scelto di utilizzare un PID per le curvature e un PD per il rettilineo (tenendo comunque come variabile l'azione integrale ma attribuendoci valore nullo fisso a causa dell'insorgere di problemi dovuti alla non continuità del suo storico);

6) I valori del PID sono stati calcolati in modo sperimentale pertanto non sono privi di errore;

B. **Implementazione Software dell'Algoritmo per il Processo di Sterzatura**

```
void main (void)
{

        for(;;)
        {
                errore_di_posizione_camera_alta = 63 - centraleAlto;
                errore_di_posizione_camera_bassa = 63 - centraleBasso;
                Controllo_di_sterzata(centraleBasso,errore_di_posizione_camera_alta);
/* utilizzo come parametri di input  per la funzione Controllo_di_sterzata il centro visto dalla
camera bassa (che è quello più immediato ed affidabile) e l'errore della camera alta che mi
consente di capire se vi è l'inizio di una curva.*/
        }


}
double Pid_per_Camera_Bassa_in_Curva(int actualPosition)
/*viene passata come input  la posizione del centro istantaneo */
{
        /*è stato dato un DEFINE dt  0.020  in quanto il loop time risulta di 20ms*/
        double outputPid=0;
/*essendo la parte proporzionale e quella derivativa  istantanee non si necessita di
memorizzarne il valore ad ogni ciclo*/
          double derivative = 0;
          double errore;
        errore= carPosition - actualPosition;
/* carPosition è la posizione di un valore  di Result[i] che si è deciso di prendere come origi-
ne del sistema di riferimento: nel nostro caso la posizione 63 */
        integralBasso = integralBasso + errore*dt;
/*essendo che la parte integrativa necessita di uno storico è stata definita come variabile
globale. */
        derivative = (errore - pre_error)/dt;
        outputPid = Kp*errore + Ki*integralBasso + Kd*derivative;
/* i coefficienti tarati sperimentalmente sono : Kp = 0.3 kd = 0.01 ki = 2.22 */

        if(integralBasso > MAX)
/* Condizione contro la saturazione ell'integrale : MAX=16*/
        {
                outputPid = MAX;
                integralBasso=0;
        }
        else if(integralBasso < MIN)// Condizione contro il wind up : MIN=-16
        {
                outputPid = MIN;
                integralBasso=0;
```

```
            }
        pre_error = errore; /*Aggiornamento errore ( pre_error è una variabile globale) */
            return outputPid;
    }

double Pid_per_Camera_Bassa_in_rettilineo(int actualPosition)
    /*sostanzialmente è un controllore di tipo Pd però è necessario dare comunque un valore
    (anche se nullo) all'azione integrale per avere uno storico completo e senza buchi di tale va-
    riabile globale */
    {
            double outputPid=0;

            double derivative = 0;

            double errore;

            errore= carPosition - actualPosition;

            integralBasso = 0;

            derivative = (errore - pre_error)/dt;

            outputPid = Kp*errore + Ki*integralBasso + Kd*derivative;


        pre_error = errore; /*Aggiornamento errore*/

    return outputPid;
    }

    int adattaControllo(int input)//trasforma i valori di pid in valori per il servo.
    {
            int risultato= (2200 - input*(multi)); //multi =27
            return risultato;
    }

    double alternativaControlloRettilineo(int errore) /*alternativa all' utilizzo del PD ma si hanno i
                                                        problemi già descritti di discontinuità
                                                        dell'azione integrale.*/
    /*funzione più rapida rispetto al Pid , la si  potrebbe utilizzare solo nel caso di rettilineo*/
    {
            double risultato = (2200 - (errore * 3.465));
    /* in sostanza è una retta che restituisce il valore centrale del servo se l'errore è nullo e il va-
    lore massimo  dell'estremo destro del servo se l'errore è -115 mentre restituisce il valore
    massimo  dell'estremo sinistro del servo se l'errore è 115.*/
            return risultato;
    }

void Controllo_di_sterzata(int cenBass, int errore_di_posizione_camera_alta)
    /* richiede in ingresso la posizione del centro assoluto individuato dalla camera bassa e la
    posizione relativa del centro individuato dalla camera Alta (cioè rispetto quello alla posizione
    di riferimento 63 (errore_di_posizione_camera_altao = 63- centraleAlto)*/
```

```
{
        int Pid_del_rettilineo = 0;
/*variabile fittizia per continuare a chiamare il pid anche in rettilineo ed evitare problemi
nell'integrazione*/

        if(nonCiVedo > 3)
/*se è da più di tre cicli che no ci vedo ( condizione per evitare che momentanei punti cechi
della camera influiscano sul controllo)*/
        {

        Pid_del_rettilineo = adattaControllo(Pid_per_Camera_Bassa_in_Curva(cenBass));
/* chiamo la funzione adatta controllo utilizzando il centro dell'unica camera che ci vede
( quella bassa) */

                if(Pid_del_rettilineo> valore_massimo_destro_del_servo)
/*Se ho un valore oltre i limiti fisici dello sterzo */
                {

EMIOS_0.CH[4].CBDR.R = valore_massimo_destro_del_servo;
valore_di_correzione_per_il_servo_nella_condizione_di_nonCiVedo =
                                                        valore_massimo_destro_del_servo;
                }
                if(Pid_del_rettilineo < valore_massimo_sinistro_del_servo)
                {
                        EMIOS_0.CH[4].CBDR.R = valore_massimo_sinistro_del_servo;
valore_di_correzione_per_il_servo_nella_condizione_di_nonCiVedo =
                                                        valore_massimo_sinistro_del_servo;
                }
if(Pid_del_rettilineo<=valore_massimo_destro_del_servo
   &&Pid_del_rettilineo >=valore_massimo_sinistro_del_servo)
                {
                        EMIOS_0.CH[4].CBDR.R = Pid_del_rettilineo;
valore_di_correzione_per_il_servo_nella_condizione_di_nonCiVedo= Pid_del_rettilineo;
                }

        }
        else
        {

                Pid_del_rettilineo = Pid_per_Camera_Bassa_in_rettilineo(cenBass);
/*continuo a chiamare il pid basso così da accumulare valori ed evitare buchi di storico
dell'azione integrale del Pid*/
        Pid_del_rettilineo = adattaControlloRettilineo(errore_di_posozione_camera_alta);
/* chiamo la funzione adatta controllo utilizzando come input i dati ricevuti sul tracciato dalla
camera alta */
                EMIOS_0.CH[4].CBDR.R =  Pid_del_rettilineo;
//assegno allo sterzo i valori del Pid
valore_di_correzione_per_il_servo_nella_condizione_di_nonCiVedo = Pid_del_rettilineo

        }

}
```

5.3 Blocco Motori

A. **Introduzione**

L'ultima parte della *Smart Car* che è fortemente condizionata dal problema del controllo è quella relativa ai motori.
Anche qui, come fatto precedentemente, risulta utile fare alcune precisazioni e dare uno sguardo ad aspetti poco intuibili dal solo codice C.
Nel capitolo 3 abbiamo visto come tramite il ponte ad H fosse possibile comandare il verso di rotazione di ciascun singolo motore; nel codice che verrà proposto, tale possibilità è stata sfruttata per consentire una frenata attiva, differente cioè dalla semplice riduzione di velocità di rotazione dei motori, e quindi avere una maggiore rapidità di risposta. La calibrazione dei valori di *Pwm* inversi da dare al blocco motore è stata fatta per via sperimentale, seguendo un andamento lineare. Tale scelta è stata fatta, in quanto, i tempi ridotti e le scarse conoscenze dell'argomento hanno imposto un approccio facilmente controllabile e di facile implementazione al problema della frenata.
Un altro aspetto che per motivi di tempo, ma anche per mancanza di conoscenze non è stato sviluppato è sicuramente la regolazione della potenza dei motori tramite indicazioni di velocità. Nel kit fornito dall'azienda *Freescale* infatti, non erano inclusi *encoder* o altri strumenti per la misura della velocità del veicolo in movimento.
Le opzioni per inserire un tale controllo erano due :
1) inserire un *encoder* ottico di tipo rotativo (scelta subito scartata per mancanza di tempo e competenze)
2) utilizzare il sensore di corrente presente sul veicolo e calcolare mediante approssimazioni numeriche la velocità ideale del veicolo, trascurando attriti e altri fattori esterni.
Il metodo che si è cercato di adottare è stato il secondo ma i risultati ottenuti sono stati troppo scarsi e quindi si è abbandonato il controllo di velocità del veicolo.
Nel codice che si analizzerà al paragrafo 5.B saranno comunque riportati i parametri dei motori e la modalità di calcolo della velocità utilizzata, a scopo puramente esemplificativo del secondo metodo.
Altro aspetto molto utile, che si è rivelato di notevole comodità in fase di *test*, è stato l' utilizzo del potenziometro presente nel microcontrollore per regolare manualmente la potenza dei motori, evitando così di modificare volta per volta il codice (la funzione in questione è : CONTROLLO_MOTORI_TRIMMER).
Il programma segue la seguente logica esprimibile per semplicità mediante schema a blocchi:

Figura 5.3 Visualizzazione del programma mediante blocchi funzionali

Nell'implementazione *software* dei concetti presentati nel paragrafo 4.3.A ci si è trovati in difficolta nella taratura di un PID per la potenza dei motori, dato che l'errore da utilizzarsi in tal caso avrebbe dovuto tener conto della velocità che, per le ragioni sopra esplicitate, non è stato possibile calcolare. Si è scelto quindi di tenere un valore medio di Pwm per il rettilineo e per le curve, così da permettere al veicolo di percorrere l'intero tracciato.

B. **Implementazione software dell'algoritmo per il processo di regolazione dei motori**

Nel leggere il seguente codice si tenga presente che le parti che non interessano il processo di controllo dei motori sono state omesse per chiarezza.

```
# define KeSx 0.03200864 /*costante di coppia del motore in volt / (rad/s)
# define KeDx 0.031463/*costante di coppia del motore in volt / (rad/s)
# define RaSx 2.42      /*resistenza motore sinistro in Ohm */
# define RaDx 2.677/* resistenza motore destro in Ohm*/
# define voltageCoeff 0.005/*conversione tra valori percentuali emios (0..1000) in una grandezza di
                            tensione in Volt*/
# define rearWhellRadius 0.025/*raggio ruota posteriore */
# define sensibilityCoeff 0.133929 /* rapporto tra la velocità angolare del motore e quella della ruota
*/

void RIGHT_MOTOR_CURRENT(void)
{
        for (i=0;i <10;i++)
        {
                ADC.MCR.B.NSTART=1; /* Trigger normal conversions for ADC0 */
                while (ADC.MSR.B.NSTART == 1) {};
                curdataDx = ADC.CDR[32].B.CDATA;
        }
        curdataDx = (curdataDx*1.8794 + 383.48)/1000; ;/*così in curdata c'è il valore della corrente
                                                        in Ampere*/
}

void LEFT_MOTOR_CURRENT(void)
{
        for(i=0; i<10;i++)
        {
                ADC.MCR.B.NSTART=1; /* Trigger normal conversions for ADC0 */
                while (ADC.MSR.B.NSTART == 1) {};
                curdataSx = ADC.CDR[3].B.CDATA;
        }
        curdataSx = (curdataSx*1.8794 + 383.48)/1000;/*così in curdata c'è il valore della corrente
                                                       in Ampere*/
}
double correnteVelocitaSinistra(double current, int relativeVoltage)
{
        double Ua = relativeVoltage*voltageCoeff;
        double Ia = current;
        double linearSpeedSx=0;
        double linearSpeedSx1=0;
        angularSpeedSx = (Ua + RaSx*Ia)/KeSx;
```

```
	linearSpeedSx = angularSpeedSx*sensibilityCoeff*rearWhellRadius;
	linearSpeedSx1 = linearSpeedSx*0.0305+0.2022;
	return linearSpeedSx1;
}

double correnteVelocitaDestra(double current, int relativeVoltage)
{
	double Ua = relativeVoltage*voltageCoeff;
	double Ia = current;
	double linearSpeedDx=0;
	double linearSpeedDx1=0;
	angularSpeedDx = (Ua + RaDx*Ia)/KeDx;
	linearSpeedDx = angularSpeedDx*sensibilityCoeff*rearWhellRadius;
	linearSpeedDx1 = linearSpeedDx*0.0449-0.4722;
	return linearSpeedDx1;
}


void CONTROLLO_MOTORI_TRIMMER(void)
{
		double trimmer;
		ADC.MCR.B.NSTART=1;/* Trigger normal conversions for ADC0 */
		while (ADC.MSR.B.NSTART == 1) {};
		trimmer = ADC.CDR[5].B.CDATA;
		trimmer=trimmer/100;
		pwmMotorRight=(uint32_t)((trimmer*2854)-28197);/*comando PWM motore destro
		                                                 */
		if(pwmMotorRight<0)/* Se va sotto la soglia ferma i motori*/
		pwmMotorRight=0;
		pwmMotorLeft=(uint32_t)((trimmer*2854)-28197);/*comando PWM motore sinistro
		                                                */
		if(pwmMotorLeft<0)/*Se  va sotto la soglia ferma i motori*/
		pwmMotorLeft=0;


}

void MotorOn(int pwmLeft, int pwmRight)/*assegno i valori di potenza calcolati mediante le funzioni
                                         apposite ai motori*/
{
		EMIOS_0.CH[6].CBDR.R = pwmLeft;
		EMIOS_0.CH[7].CBDR.R = pwmRight;
}

void Controllo_di_sterzata(int cenBass, int errore_di_posizione_camera_bassa)
{
	int freno = 0;/*inizializzo una variabile per quantificare la potenza di frenata*/


	if (rettilineo > 200)
	{
		freno = 50; /*valore arbitrario da convertire successivamente in comando per la
		              potenza inversa dei motori*/
	}
	if(rettilineo <= 200)
```

```
        {
                freno =(int) -3*rettilineo + 700;/*creo una funzione lineare per frenare.
                la potenza frenante deve aumentare proporzionalmente alla condizione di rettilineo*/
                if (freno>999)/*se per qualche motivo la variabile rettilineo fosse negativa devo
                                evitare di superare la potenza massima ammissibile dai motori*/
                freno = 999;
        }
  if(nonCiVedo > 3)
        {
                if((nonCiVedo == 4 && rettilineo > 100) | (frenatura<=15))
                {
                                SIU.PCR[6].R = 0x0000;// abilita motore sinistro per andare indietro
                                SIU.PCR[7].R = 0x0000;// abilita motore destro per andare indietro
                                frenatura = frenatura + 1;
/*aumento il numero di cicli in cui sto frenando a causa di una cattiva visione della camera*/
                                EMIOS_0.CH[6].CBDR.R = 1 + freno*(1 - (frenatura/15));
/*aumento la potenza frenante proporzionalmente alla condizione di rettilineo (grazie a freno) e in
funzione del numero di cicli in cui si è iniziato a frenare*/
                                EMIOS_0.CH[7].CBDR.R = 1 + freno*(1 - (frenatura/15));
/*aumento la potenza frenante proporzionalmente alla condizione di rettilineo (grazie a freno) e in
funzione del numero di cicli in cui si è iniziato a frenare*/

                }
     if (frenatura == 15)//15 è un valore arbitrario scelto sperimentalmente
      {
                        SIU.PCR[6].R = 0x0200;// abilita motore sinistro per andare avanti

                        SIU.PCR[7].R = 0x0200;// abilita motore destro per andare avanti
                        frenatura=0;/* azzero le condizioni di frenatura e rettilineo*/
                        rettilineo = 0;
                }
                if(errore_di_posizione_camera_bassa>0 && frenatura == 0)//curva a sinistra
                 {
                SIU.PGPDO[0].R = 0x00004000;
                EMIOS_0.CH[7].CBDR.R = 600;
                EMIOS_0.CH[6].CBDR.R = 300;

                 }
                 if(errore_di_posizione_camera_bassa<0 && frenatura == 0)//curva a destra
                   {
                SIU.PGPDO[0].R = 0x00008000;/
                EMIOS_0.CH[6].CBDR.R = 600;
                EMIOS_0.CH[7].CBDR.R = 300;
                }
        }
        else
        {
                SIU.PCR[6].R = 0x0200;// abilita motore sinistro per andare avanti
                SIU.PCR[7].R = 0x0200;//abilita motore destro per andare avanti
                SIU.PGPDO[0].R = 0x0000C000;//accendo entrambi i motori
                pwmMotorRight = 500;//assegno valore medio come potenza da dare al motore
                                    //destro
                pwmMotorLeft = 500;//assegno valore medio come potenza da dare al motore
                                   //sinistro
                rettilineo = rettilineo + 1;//aggiorno il contatore del rettilineo
```

```
        }

    }

void main (void)
{


        SIU.PCR[16].R = 0x0200;//abilita motore sinistro per andare avanti
        SIU.PCR[17].R = 0x0200;// abilita motore destro per andare avanti
        SIU.PGPDO[0].R = 0x0000C000;//Accendo entrambi i motori con la Pwm inizialmente
                                    //impostata
        rettilineo = 0;/*inizializzo la variabile rettilineo tale variabile è un contatore che aumenta ogni
ciclo che la camera vede il centro della posizione di riferimento ( 63) in questo modo ho la certezza
dell'andamento del tracciato*/
        frenatura = 0;/*inizializzo la variabile che quantifica il numero di cicli in cui sto frenando*/
        for(;;)
        {

            errore_di_posizione_camera_alta = 63 - centraleAlto;
          errore_di_posizione_camera_bassa = 63 - centraleBasso;
          Controllo_di_sterzata(centraleBasso,errore_di_posizione_camera_alta);
          MotorOn(pwmMotorLeft,pwmMotorRight); /*ad ogni ciclo assegna la Pwm
                                                (modificata dalla funzione
                                               "Controllo_di_sterzata")  ai motori*/
        }

}
```

Conclusioni

Lo scopo ultimo di questo elaborato è quello di dare una visione d'insieme del progetto *Freescale* agli studenti degli anni futuri che vorranno cimentarsi con questa esperienza unica. Si è voluto fornire una visione incentrata soprattutto sul problema del controllo in quanto, in esso si riassumono tutte le conoscenze e discipline che vengono affrontate durante l'intero ciclo della laurea triennale di Ingegneria Mecatronica dell'Università di Vicenza.
Sicuramente è stata un'occasione per mettere sul campo e collegare tutte le conoscenze base della meccatronica ma principalmente è stata utile per avere un modo diverso (più pratico e meno teorico) di approcciarsi ai problemi reali e una rara occasione per sperimentare il lavoro di *team*.
Da studente appena approdato al terzo anno di Ingegneria, mi è stato molto difficile capire il funzionamento di molte parti del progetto (ADC, utilizzo delle strumentazioni di laboratori, tecnica Pwm, controllori PID, ecc…) vista la complessità degli argomenti. Tuttavia affrontare simili problemi prima da un punto di vista pratico e poi da un punto di vista teorico, mi ha permesso di entrare più a fondo nel problema e di capirlo in modo più adeguato, facilitandone, poi, il successivo studio.
Ritengo, infine, che siano proprio esperienze formative come questa che permettono a noi aspiranti ingegneri di capire molti aspetti del lavoro che andremo a svolgere e di apprendere le modalità corrette di approccio ai problemi dei sistemi meccatronici.

Bibliografia e Sitografia

[1] M. Bertocco and A. Sona, Introduzione alle misure elettroniche, Lulu, 2nd ed., 2010

[2] T. N. Blalock and R. C. Jaeger, Microelectronic Circuit Design, McGraw-Hill, 3rd ed.,2008

[3] Freescale Semiconductor, MC33931 Data Sheet, Rev. 3.0, June 2012

[4] Freescale Semiconductor, MPC5604B Application Note: Using Parallax

[5] TSL1401-DB LinescanCamera Modulefor line detection, Rev. 0, January 2011

[6] Freescale Semiconductor, MPC5604B/C Microcontroller Reference Manual, Rev. 8.1, May 2012

[7] Taos, TSL1401CL 128x1 linear sensor array with hold, July 2011

[8] M. Zigliotto, Dispense dal corso di Fondamenti di macchine ed azionamenti elettrici, 2013

[9] R. Oboe, Dispensa dal corso di Controlli Automatici, 2013

[10] Freescale Semiconductor : Jason Lee – KLM MSG (2nd August 2008 - Smart Car Workshop)

[11] Programmazione in C, Kim N. King , Apogeo

[12] Controlli Automatici – di Giovanni Marro – Ed. Zanichelli - 5° Edizione

[13] Uninettuno - Corso di Azionamenti Elettrici 1

[14] Principi e applicazioni di elettrotecnica Massumo Guarnieri, Andrea Stella

[15] https://community.freescale.com/community/uvp

[16] Futaba (http://www.futabarc.com)

[17] Servo motor tutorial (http://www.seattlerobotics.org)

[18] Standard Motors (http://www.standardmotor.net/)

[19] http://en.wikipedia.org/wiki/Freescale_Semiconductor

[20] https://community.freescale.com/docs/DOC-93225

[21] http://www.ilmondodelletelecomunicazioni.it/argomento.php?id_lezione=22

[22] http://cache.freescale.com/files/32bit/doc/data_sheet/MPC5604BC.pdf?fr=gdc

[23] http://www.vincenzov.net/tutorial/motoridc/driver.htm

[24] http://www.grifo.it/CORSO/BASCOM_8051/Esempi_8051/Bas51_025

[25] http://www.standardmotor.net/sc_webcat/ecat/product_view.php?lang1&cat=1&id=50&page=1

[26] https://community.freescale.com/docs/DOC-1030

[27] http://www.ams.com/eng/LinearSensorArray