

UNIVERSITÀ DEGLI STUDI DI PADOVA



FACOLTÀ DI INGEGNERIA

**Corso di Laurea in
Ingegneria Meccanica e Meccatronica**

Tesi di Laurea

**Sensori di visione lineari e
relativo utilizzo nel controllo di
una smart car**

ANNO ACCADEMICO 2012 – 2013

Laureando
Menara Tommaso

Relatore
Ch.mo prof. **Oboe Roberto**

1002027 - IMM

INDICE

SOMMARIO	5
INTRODUZIONE: IL PROGETTO FREESCALE CUP	7
1. PANORAMICA SUI COMPONENTI DELLA SMART CAR.....	10
1.1 Il Kit fornito	10
1.2 Componenti meccanici.....	11
1.2.1 Telaio.....	11
1.2.2 Trasmissione.....	12
1.3 Azionamenti elettrici	13
1.3.1 Servomotore	13
1.3.2 Motori DC per la propulsione.....	15
1.4 Componenti elettronici	18
1.4.1 Sensore di visione	18
1.4.2 Freescale Cup Motor Drive Board	19
A. Analisi della scheda.....	19
B. La tecnica di <i>Pulse Width Modulation</i>	20
1.4.3 Microcontrollore TRK-MPC5604B	23
2. ACQUISIZIONE DEI SEGNALI MEDIANTE SENSORE DI VISIONE.....	27
2.1 Sensore TSL1401CL e relativo montaggio	27
2.1.1 Caratteristiche tecniche	27
2.1.2 Principi di funzionamento	28
2.1.3 Montaggio della camera	33
2.2 Introduzione al software Codewarrior ed al linguaggio C.....	35
2.3 Setup dei sensori di visione.....	37
2.3.1 Idea iniziale con una camera	37
2.3.2 Aggiunta della seconda camera	39

2.4	Faro a LED	42
2.4.1	Prima versione.....	42
2.4.2	Seconda versione.....	44
2.5	Analisi del codice relativo alle tecniche di acquisizione della linea.....	45
2.5.1	Tecnica della media.....	45
2.5.2	Metodo derivativo	47
2.5	Disturbi nell'acquisizione e soluzioni adottate	49
3.	ELABORAZIONE ED UTILIZZO DEI SEGNALI ACQUISITI	52
3.1	Conversione dei segnali da parte dell'ADC.....	52
3.2	Stima della velocità mediante <i>feedback current</i>	55
3.3	Controllori PID	58
	CONCLUSIONI	62
	BIBLIOGRAFIA e SITOGRAFIA.....	64

SOMMARIO

Il seguente elaborato è il risultato del lavoro svolto nell'ambito della competizione *Freescale Cup*, svoltasi per la prima volta anche in Italia a partire dall'anno accademico 2012/2013. Si tratta di una gara tra auto-modelli autonomi forniti dall'azienda *Freescale Semiconductors* a gruppi composti da 3 studenti, i quali devono assemblarli e programmarli. Quello che gli studenti si trovano ad affrontare è una sfida di tipo mecatronico, oltre che di ingegno. I modelli, in scala 1/18, devono correre su pannelli bianchi segnati da una linea nera che si trova al centro di essi seguendo così un determinato percorso, nel quale sono presenti vari tipi di ostacoli, in modo completamente autonomo. La gara è vinta dal veicolo che effettua nel minor tempo un giro della pista.

La stesura del seguente documento ha come obiettivi quello di analizzare le modalità di acquisizione dei dati utili dalla pista e di esemplificare un possibile modo di utilizzo del segnale ottenuto mediante i sensori di visione.

INTRODUZIONE

Il progetto Freescale Cup



Figura 0.1: Logo della Freescale Cup

La sfida che gli studenti scelgono di affrontare nasce in Corea nel 2003 con 80 squadre, per poi vedere una crescita esponenziale sino ad arrivare a migliaia di squadre in tutto il mondo (in particolar modo in Cina) che competono per ottenere la vittoria ogni anno. Si deve attendere però il 2011 per vedere partecipare le prime squadre dell' EMEA (Europe, Middle East, Africa). È poi nell'anno 2011/2012 che i tre migliori team di Cina, USA e Messico sono stati invitati a competere in una finale mondiale svoltasi al *Freescale Technology Forum* a San Antonio, in Texas.

Segue uno schema a blocchi che sintetizza l'oggetto del lavoro delle varie squadre:

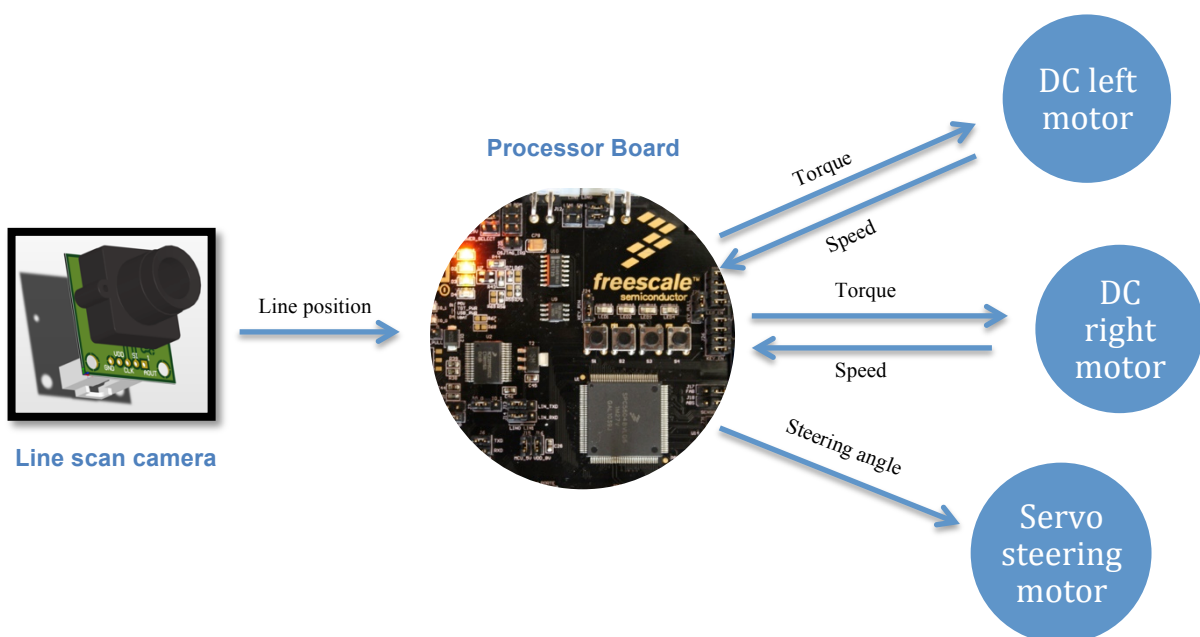


Figura 0.2: Schema a blocchi del funzionamento del veicolo

L'obiettivo principale della gara risiede nell'elaborazione dei dati, raccolti dal blocco sensori, con l'utilizzo della *processor board* che va a regolare i vari organi di sterzo e propulsione in relazione ad una eventuale variazione del percorso che il veicolo è in procinto di affrontare, la quale può essere una curva, un tunnel o un dosso. Tutto deve avvenire in maniera autonoma, senza interventi esterni. Dal sensore di visione si acquisisce un riferimento relativo al centro della linea nera, ai due motori DC che forniscono trazione di tipo posteriore viene mandata una tensione di armatura con tecnica PWM (*Pulse Width Modulation*), mentre al servomotore si manda un riferimento di posizione, permettendo così al veicolo di sterzare. Il controllo di questi parametri è affidato ad un controllore di tipo PID (*Proportional, Integral, Derivative*). L'interazione tra i parametri appena citati e quelli che verranno analizzati in seguito, è di tipo non lineare. È necessario quindi cercare di semplificare il più possibile la gestione del flusso di dati trovando il giusto compromesso tra prove di tipo empirico e analisi di tipo teorico.

Il percorso è su un fondo bianco in *abs* della larghezza di 600 mm al centro del quale si trova una linea nera di 25 mm di larghezza.



Figura 0.3: Esempio di tracciato ufficiale

Nell'arco di un semestre accademico ad ogni squadra è fornito un modello ed ha quindi il tempo di sviluppare hardware e software in previsione delle selezioni locali che avvengono a livello universitario. Per l'occasione è stato costruito un tracciato il più simile possibile ad un tracciato ufficiale in un laboratorio del Dipartimento di Tecnica e Gestione dei Sistemi Industriali dell'Università degli studi di Padova, sito in Vicenza.



Figura 0.4: Porzione del tracciato dell'università

La linea di partenza, la stessa linea che delimita il traguardo, è opportunatamente segnata, mediante due linee nere trasversali disegnate ai lati della linea nera.

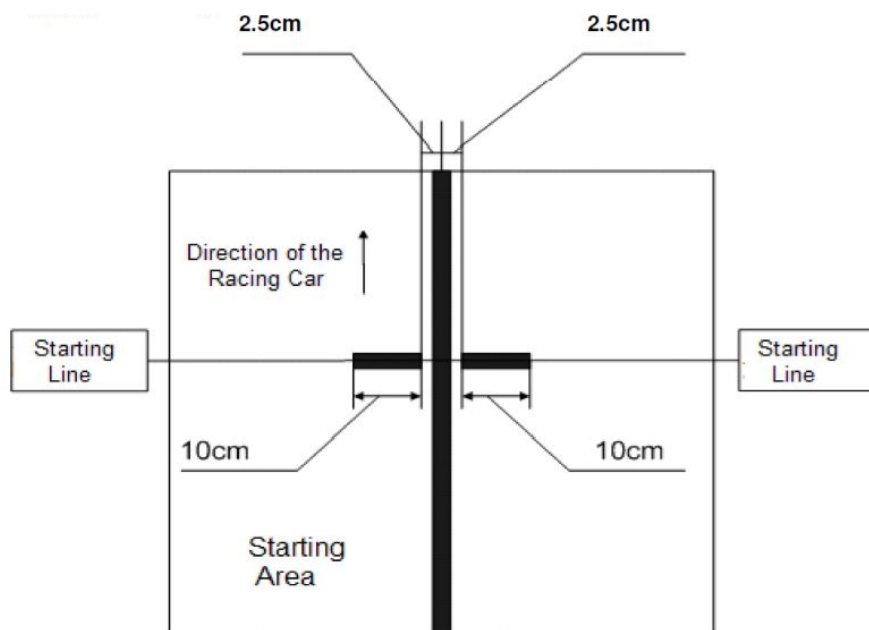


Figura 0.5: Start/Stop line

CAPITOLO 1

Panoramica sui componenti della smart car

1.1 Il kit fornito

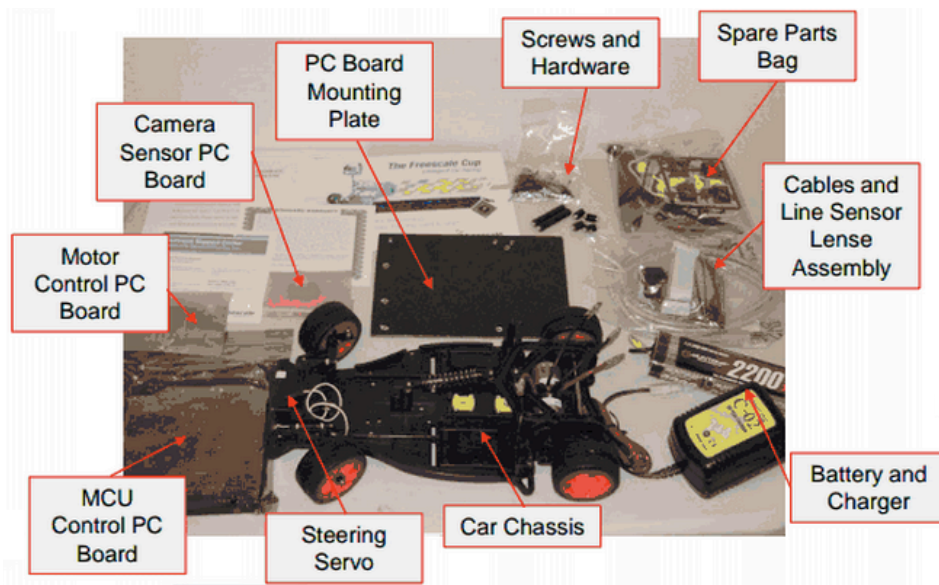


Figura 1.1: Il Kit per la "Freescale Cup"

Viene fornito all'università un kit per ogni squadra, essenzialmente composto da:

- 1) Microcontrollore MPC5604B
- 2) Scheda denominata *Motor Drive* (o *Motor Board*) per il controllo e alimentazione di attuatori e sensori
- 3) Telaio in materiale plastico in scala 1/18
- 4) Ruote e pneumatici
- 5) Alimentatore e pacchetto batterie
- 6) Una camera lineare che funge da sensore di visione
- 7) Coppia di motori DC a 7,2V collegati alle ruote posteriori tramite ingranaggi
- 8) Servomotore con relative astine regolabili per il collegamento alle ruote anteriori
- 9) Cablaggi e connettori vari

1.2 Componenti Meccanici

1.2.1 Telaio

Il telaio del veicolo è costituito da due parti plastiche, collegate tra loro attraverso una giunzione regolabile fissata tramite una vite. La giunzione collega il retrotreno alla parte principale mediante una sospensione a ponte rigido. È stata scelta questa soluzione anche per rendere leggermente snodabile la parte posteriore, nella quale trovano alloggio i due motori collegati alle due ruote posteriori.

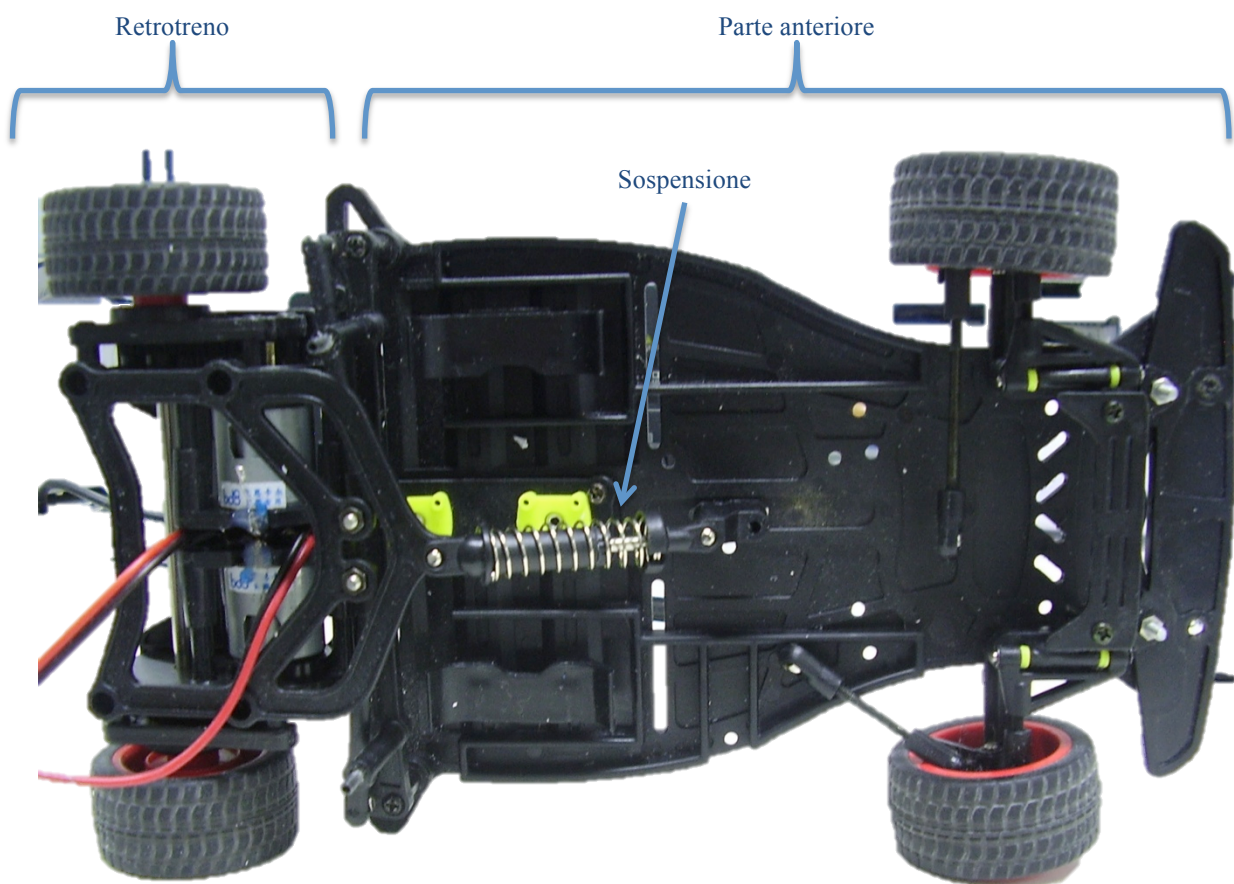


Figura 1.2: Top view del telaio del veicolo

Nel retrotreno trovano alloggio i due motori ed il rotismo che consente di collegarli alle ruote posteriori per la trasmissione della potenza.

Sotto alla sospensione a ponte rigido vi è l'alloggiamento per la batteria, ed infine nella parte anteriore troviamo, oltre alle ruote anteriori (già attaccate), uno spoiler con doppia funzione:

- 1) Funzione paraurti nel caso in cui il veicolo andasse a sbattere contro qualche ostacolo

2) Possibile base per il fissaggio dei supporti per i sensori di visione.

Per quanto riguarda le sospensioni anteriori, esse sono costituite da una molla coassiale. Le ruote presentano un problema dovuto al modo in cui sono montate: sono fissate al mozzo tramite un cuscinetto che presenta un gioco non trascurabile. Questo fatto provoca difficoltà nella gestione dell'angolo di sterzo, che difficilmente sarà sempre uguale a parità di angolo di sterzo impresso al servomotore. Un altro problema riscontrato dovuto al telaio è la ridotta altezza da terra che in caso di curva ad elevata velocità può causare il contatto, in alcuni punti, dei bordi dell'avantreno con il fondo della pista.

1.2.2 Trasmissione

La trasmissione del moto alle ruote posteriori avviene tramite un rotismo.

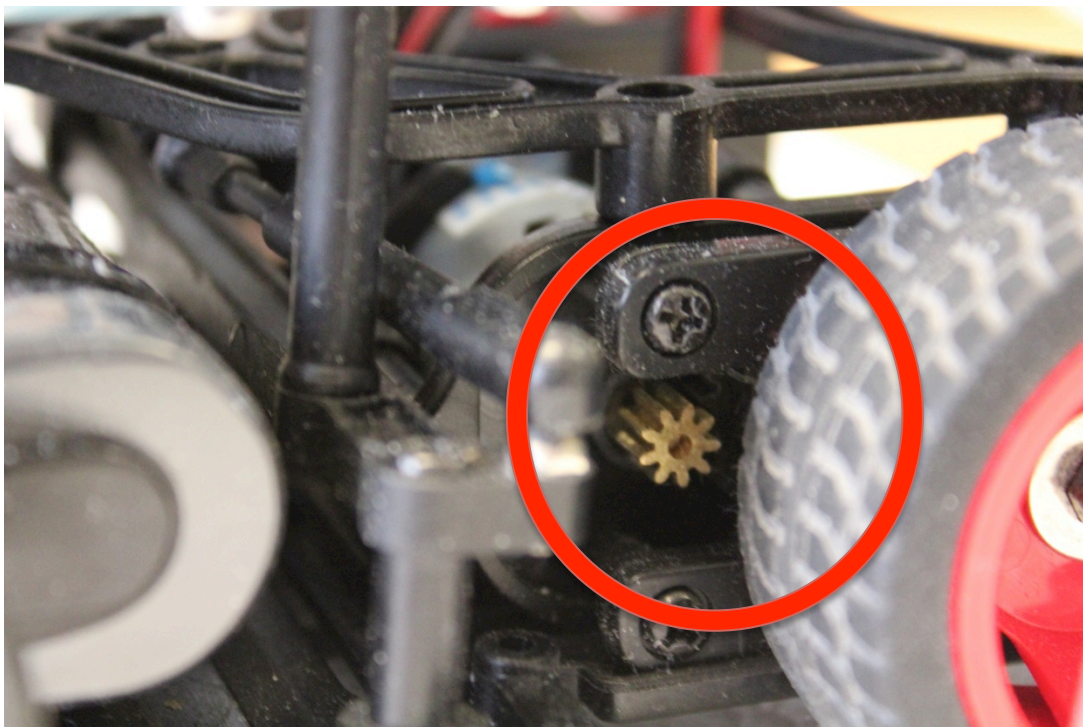


Figura 1.3: Dettaglio degli ingranaggi per la ruota posteriore sinistra

Il rotismo è di tipo ordinario a 4 ruote dentate ed il numero di denti di esse è:

- Denti ruota dentata calettata su albero motore: 9
- Denti prima ruota dentata albero centrale: 28
- Denti seconda ruota dentata albero centrale: 10
- Denti ruota dentata calettata sull'albero della ruota: 24

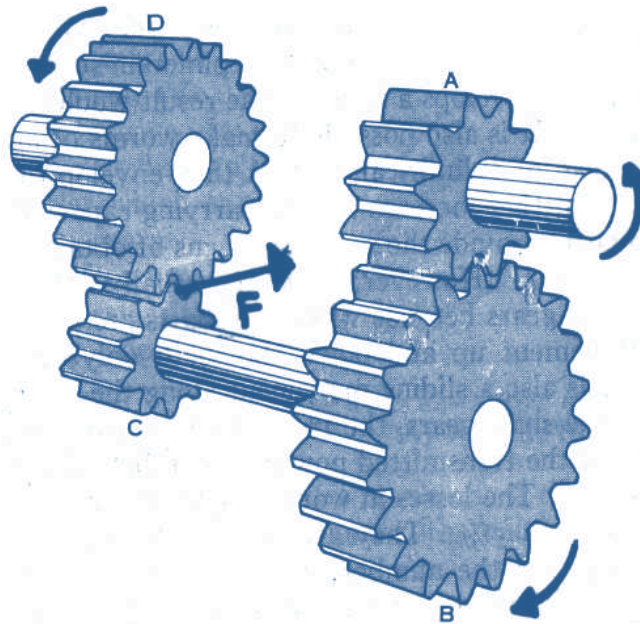


Figura 1.4: Rappresentazione di un rotismo ordinario

Dal numero di denti possiamo così ricavare il rapporto di trasmissione, dato da:

$$\frac{9}{28} * \frac{10}{24} = 0,1339$$

1.3 Azionamenti Elettrici

1.3.1 Servomotore

L'organo utilizzato per imprimere l'angolo di sterzo alle ruote anteriori è un servomotore in continua. In particolare il modello presente nel kit è il *"Futaba S3010"* (Figura 1.5).

Questo modello viene alimentato a 5V dalla scheda *Motor Drive*, ed il comando per il controllo dell'angolo di rotazione è di tipo PWM. Si è notato come i riferimenti PWM relativi all'angolo di sterzo massimo imprimibile, sia da un lato che dall'altro, fossero diversi da un gruppo all'altro di studenti. Sarà quindi necessario procedere con una taratura di tipo sperimentale partendo dal valore centrale di posizione neutra del

rotore e variando un poco alla volta l'angolo di rotazione via software, sino a trovare dei valori che permettano di sterzare senza sforzare troppo il servo comando ed evitando che gli pneumatici tocchino lo chassis del veicolo.



Figura 1.5: Servomotore "Futaba S3010"

ATTENZIONE: È necessario prestare molta attenzione a non sforzare troppo il servomotore imprimendogli angoli di rotazione troppo elevati, poichè questo può portare al lasciarsi di esso e quindi ad un funzionamento scorretto, da cui una imprevedibilità non trascurabile di tutta la parte di controllo relativa alla sterzata.

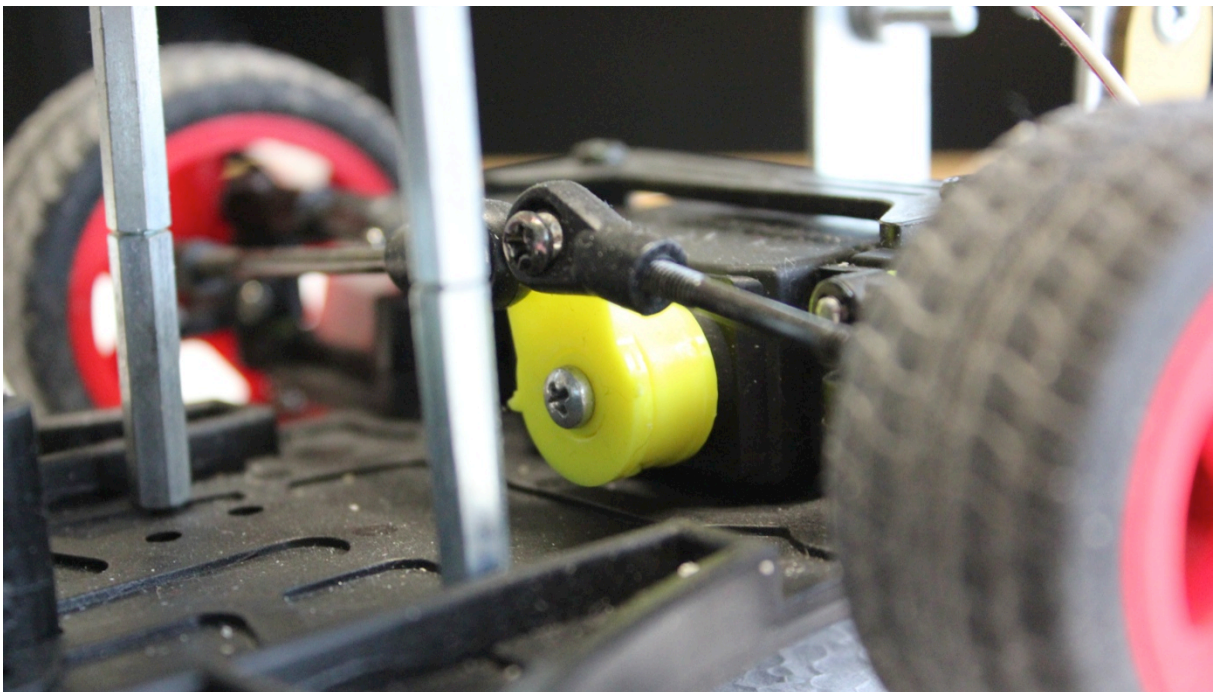


Figura 1.6: Collegamenti fra servomotore e ruote

Il collegamento con le ruote avviene mediante dei braccetti regolabili in materiale plastico, come visibile in figura 1.6. Questo semplifica la convergenza delle ruote, che

viene decisa durante il montaggio. Si è scelta una convergenza leggermente chiusa per un miglior controllo del veicolo in curva.

Da figura 1.7 si possono osservare alcune caratteristiche tecniche:

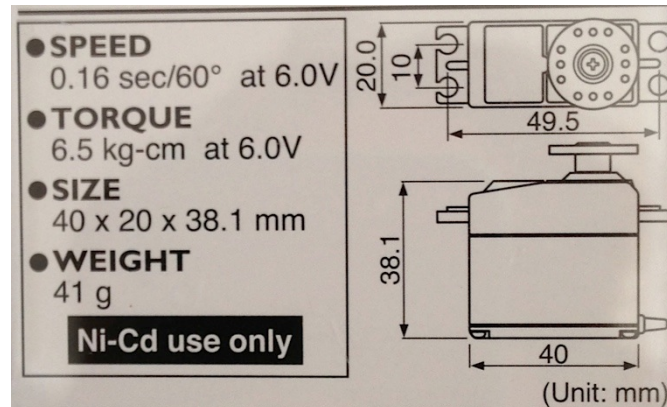


Figura 1.7: Retro della confezione del servomotore

Infine, il cablaggio del servomotore è costituito da tre fili:

- Bianco (1) → PWM
- Rosso (2) → alimentazione
- Nero (3) → massa

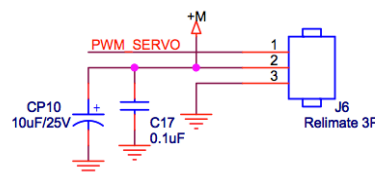


Figura 1.8: Schema del cablaggio del servomotore

1.3.2 Motori DC per la propulsione

Per la propulsione sono forniti due motori a corrente continua del tipo *Standard Motor: "rn 260 c" winding 18130*.

Segue una brevissima spiegazione del funzionamento di un motore DC. Per una spiegazione più approfondita si rimanda a testi specifici sull'argomento disponibili in letteratura.

Le parti fondamentali da prendere in considerazione per capire il funzionamento di questa tipologia di motori sono essenzialmente due: l'induttore e l'indotto.

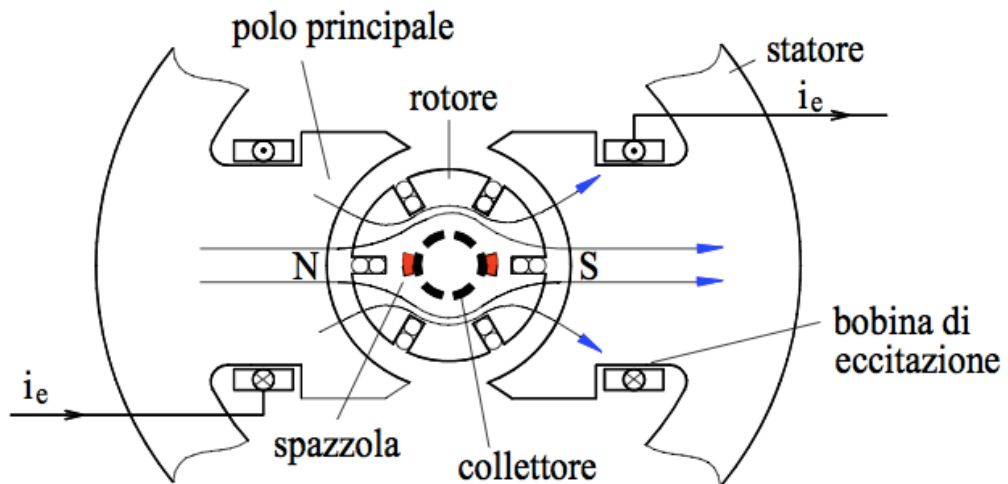


Figura 1.9: Schema di un motore DC

L'induttore consiste in una parte fissa detta *statore*, nel quale verrà prodotto il campo magnetico principale da un apposito circuito di eccitazione (nel quale in figura 1.9 viene fatta scorrere la corrente i_e) e che verrà incanalato nelle espansioni polari (o *poli*) e fatto passare nel traferro (la parte vuota, di aria), nel quale si trova l'indotto.

L'indotto consiste in una parte mobile detta *rotore*, nel quale verrà concatenato, dagli avvolgimenti presenti in esso, il campo magnetico principale. Gli avvolgimenti di rotore sono detti *conduttori attivi* e si trovano in particolari scanalature dette *cave*. Come si può dedurre dal loro nome essi vengono eccitati mediante un sistema spazzole-collettore, che consiste in un cilindro solidale al rotore sulla superficie del quale vi sono delle lamelle metalliche (il *collettore*) insieme a dei contatti metallici fissi (le *spazzole*) sui quali viene fatta scorrere la corrente di armatura i_a . Le lamelle sono collegate mediante dei fili conduttori agli avvolgimenti di rotore, come illustrato in un esempio in figura 1.10.

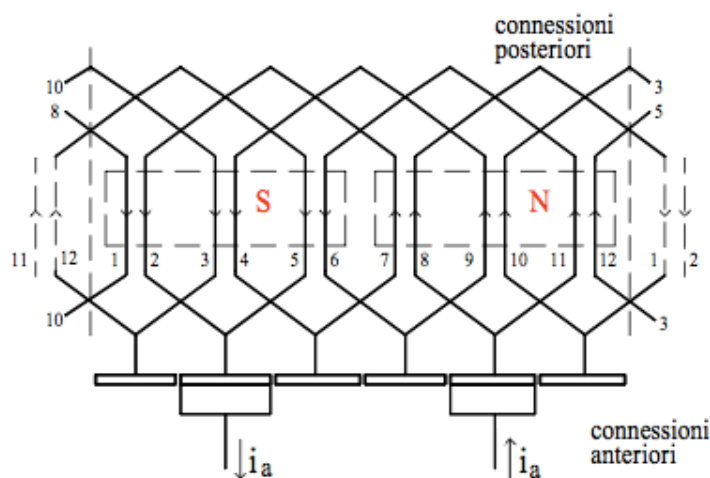


Figura 1.10: Esempio di connessioni tra collettore e conduttori del rotore

L'interazione tra i campi magnetici di statore e di rotore produce il moto rotatorio del sistema rotorico.

Sempre in figura 1.8 si nota come la corrente di armatura venga inviata attraverso le spazzole e si suddivida nelle due vie interne (identiche tra loro), ciascuna formata da sei conduttori, che si possono individuare percorrendo l'indotto da una spazzola all'altra. Si riconosce altresì che correnti dotate dello stesso verso percorrono tutti i conduttori che giacciono sotto la stessa espansione polare, 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 rimane inalterata la distribuzione delle correnti sotto ogni polo. In particolare in alcune coppie si osserverà un'inversione della corrente.

La costanza della coppia prodotta è dovuta al fatto che 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. Posizionando invece il rotore in una posizione intermedia ogni spazzola alimenta due lamelle adiacenti, con correnti proporzionali alle relative superfici di contatto.

Come si può notare da figura 1.11 i motori sono indipendenti. Questo fatto è fondamentale per la progettazione del controllo del veicolo, in quanto è opportuno che un eventuale differenziale andrà programmato via software, data la mancanza di un differenziale di tipo meccanico.

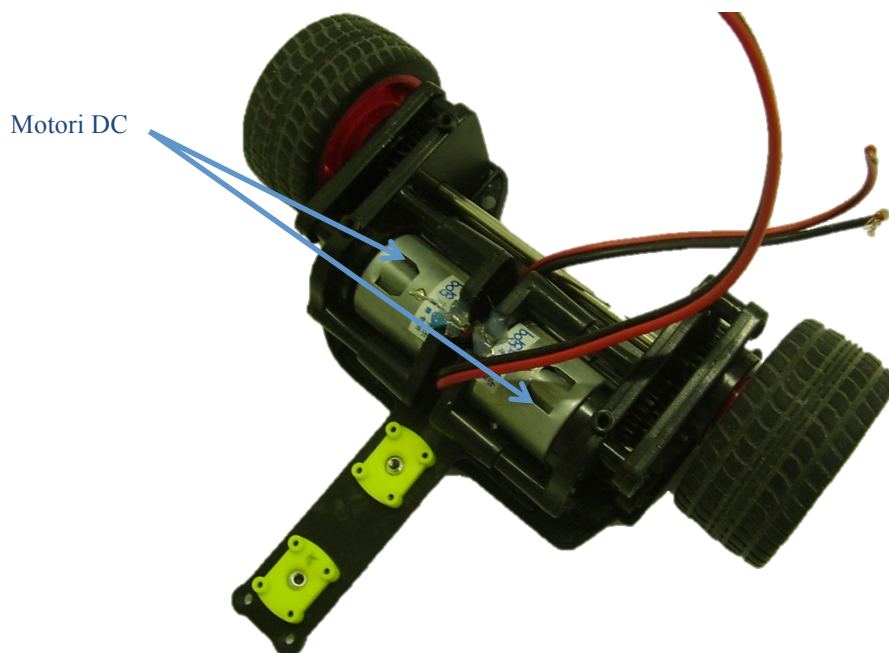


Figura 1.11: Fotografia del retrotreno dell'auto

1.4 Componenti elettronici

Meritano una trattazione più dettagliata rispetto ai precedenti i componenti elettronici, in quanto organi fondamentali per l'analisi su cui verte questo elaborato.

1.4.1 Sensore di visione

Nel kit è compreso anche un modulo con una camera lineare detto *Parallax TSL 1401-DB Line Scan Camera Module*. È il sensore più importante presente nella parte di sensing del veicolo e permette la visione anteriore ed il riconoscimento della linea nera.

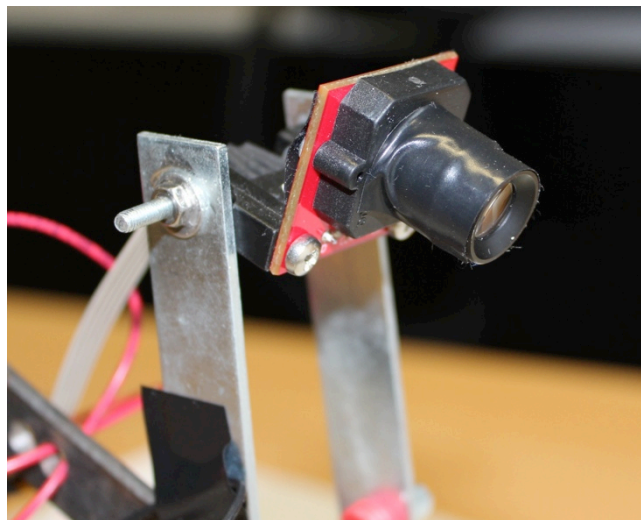


Figura 1.12: Fotografia della camera montata su supporti metallici

Al sensore di luminosità, visibile in figura 1.13, viene abbinata una lente, regolabile tramite avvitamento, da 7.9 mm per mettere a fuoco l' "immagine".

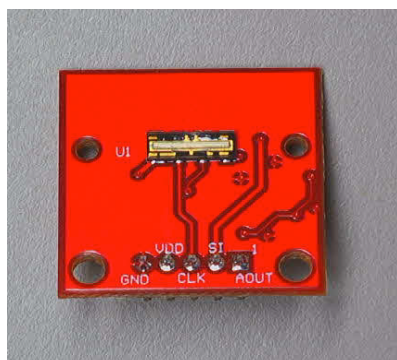


Figura 1.13: Dettaglio del sensore di luce

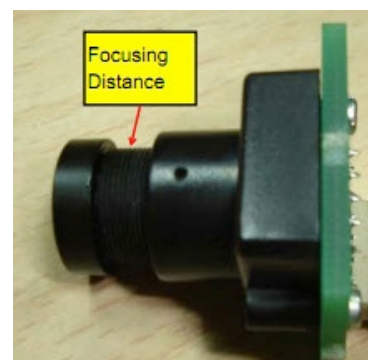


Figura 1.14: La lente montata sul sensore

Questo sensore è costituito da un *array* lineare CMOS di 128 fotodiodi, accoppiati a degli amplificatori operazionali per la carica di piccoli condensatori e a delle funzioni che comandano l'integrazione di ogni pixel.

Si rimanda al capitolo 2 per un'analisi più approfondita delle caratteristiche tecniche e del funzionamento della camera.

1.4.2 Freescale Cup Motor Drive Board

A. Analisi della scheda

Lo stadio di potenza è separato dal processore, e si trova nella scheda *Motor Drive*, insieme all'hardware necessario ad interfacciarsi con servomotore e sensore di visione.

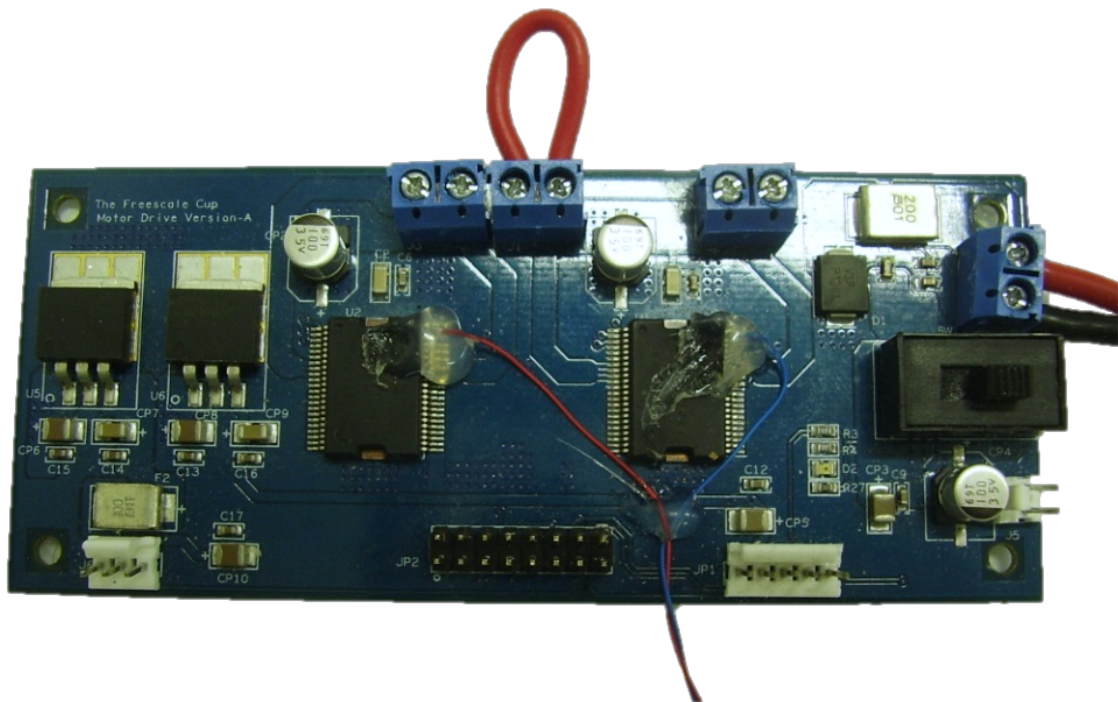


Figura 1.15: Fotografia del Freescale Cup Motor Drive

Nella scheda sono presenti 2 *H-bridge* per l'alimentazione dei motori DC tramite tecnica PWM ed i connettori per il cablaggio che interconnette il servomotore, nonché per il cablaggio relativo alla camera. È inoltre presente un interruttore che controlla l'alimentazione proveniente dalla batteria. Come e dove fissare questa scheda sullo *chassis* del veicolo è lasciato decidere ai concorrenti.

La connessione tra questa scheda ed il processore avviene tramite una piattina a 16

pin, che trasporta diversi segnali, tra cui quelli di comando per gli attuatori e quelli di lettura dei segnali provenienti dal sensore di visione o come il feedback di corrente dei motori DC usati per la propulsione.

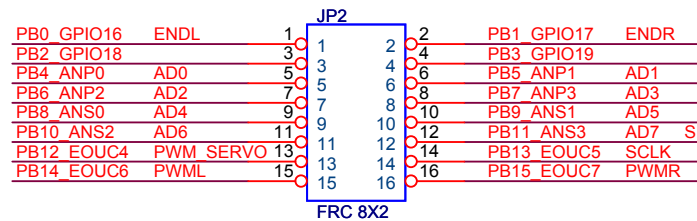


Figura 1.16: Schema dei pin della piastrina che interconnette *Motor Drive* e *Processor Board*

In figura 1.17 sono rappresentati i principali rapporti con le altre parti della *smart car*. Alla camera vengono mandati i segnali di *clock* e *serial input* ed in risposta si riceve l'output analogico. Il servomotore viene controllato mediante tecnica PWM, il cui valore arriva dal pin 13 della piastrina (vedi figura 1.16). Attraverso i pin 15 e 16 arrivano i valori di PWM da dare ai motori DC di sinistra e destra, rispettivamente, e dai ponti ad H si riceve il feedback di corrente, utilizzato mediante analisi via software per calcolare una stima della velocità del veicolo (si rimanda al paragrafo 3.2 per un approfondimento relativo a questa analisi).

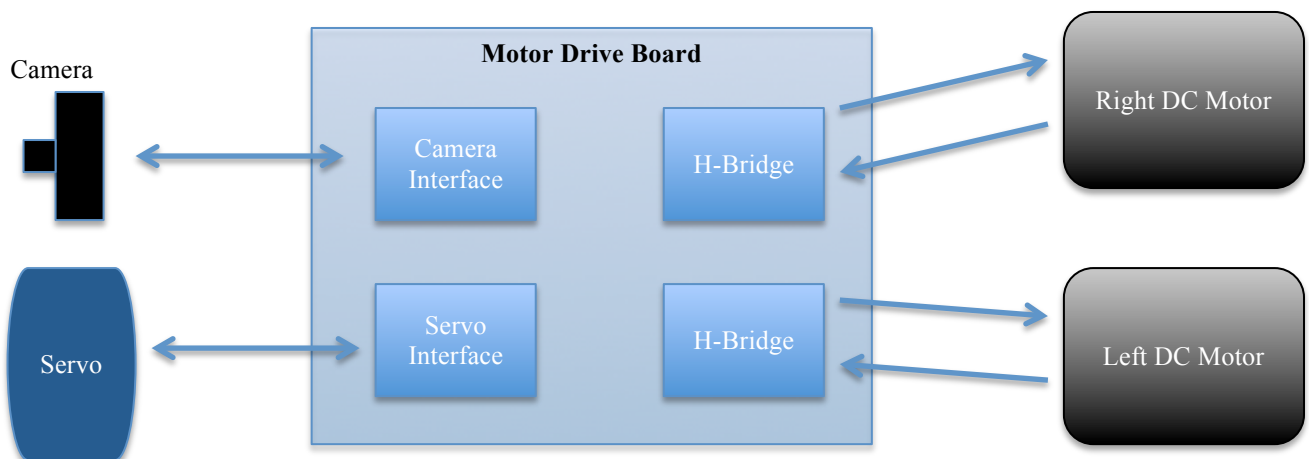


Figura 1.17: Schema delle interfacce tra *Motor Drive Board* e altre parti della *Smart Car*

B. La tecnica di *Pulse Width Modulation*

La tecnica denominata PWM, ovvero *Pulse Width Modulation* (modulazione della larghezza di impulso) consiste in una modulazione di tipo digitale che permette di ottenere una tensione media variabile che dipende dal rapporto tra la durata di un

impulso positivo e la durata di un impulso negativo. Grazie ai moderni microcontrollori è possibile attivare o disattivare un interruttore ad alta frequenza e allo stesso modo rilevare lo stato e il periodo di un impulso.

Questa tecnica è largamente utilizzata per la regolazione della potenza elettrica inviata ad un carico, e questo avviene attraverso la regolazione del *duty cycle*. Il *duty cycle* è la frazione di tempo che un'entità passa in uno stato attivo in relazione al tempo totale considerato. Per il suo calcolo si può usare questa formula, dove " ∂ " è il *duty cycle*:

$$\partial = \frac{T_{on}}{T_{tot}}$$

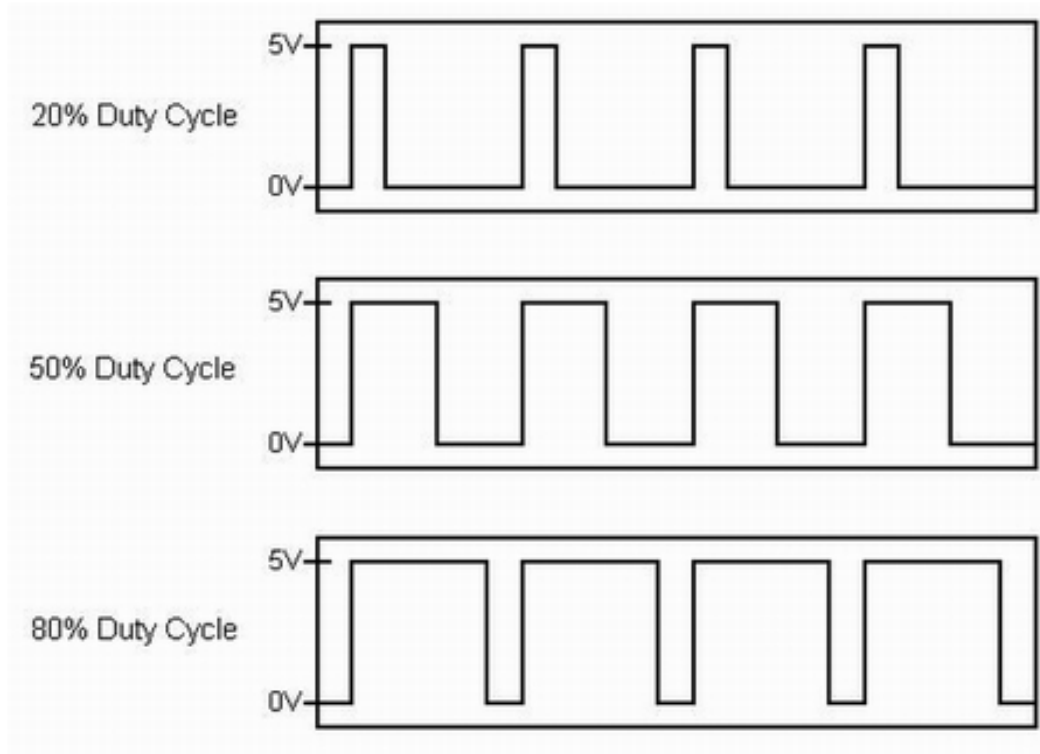


Figura 1.18: Diversi *duty cycle*

È immediato rendersi conto del fatto che un duty cycle pari a 0% implica che la potenza trasferita sia nulla, mentre al 100% si trasferisce la massima potenza disponibile. Tutti i valori intermedi servono per ottenere una corrispondente fornitura di potenza.

Tra i vantaggi della tecnica di modulazione PWM vi sono sicuramente una semplificazione ed un minor ritardo della sezione di potenza dell'azionamento, con benefici sulle prestazioni dinamiche di quest'ultimo.

Gli svantaggi invece riguardano le perdite per commutazione: maggiore è il numero delle commutazioni degli interruttori di potenza, maggiori sono le perdite. Si parla comunque di perdite accettabili a fronte dei vantaggi appena visti.

Per quanto riguarda il servomotore, l'alimentazione PWM avviene grazie ad un convertitore statico all'interno di esso che permette un'amplificazione del segnale in arrivo dalla scheda *Motor Drive*. Il servo attende un impulso ogni 20 ms, la durata dell'impulso determinerà la posizione angolare che dovrà assumere il rotore.

L'alimentazione dei motori posteriori invece avviene tramite amplificazione del segnale PWM che passa per il *H-Bridge* (ponte ad H o *chopper* a 4 quadranti) del relativo motore. Il ponte ad H è un circuito elettronico in grado di poter funzionare nei quattro quadranti del piano corrente-tensione sul carico.

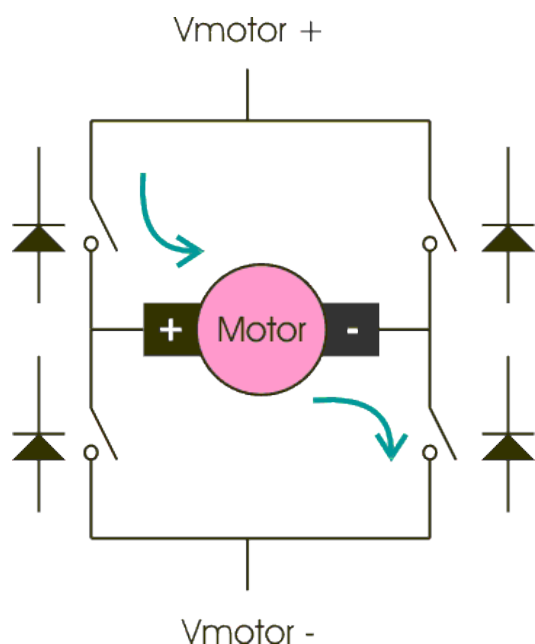


Figura 1.19: Schematizzazione concettuale di un *H-Bridge*

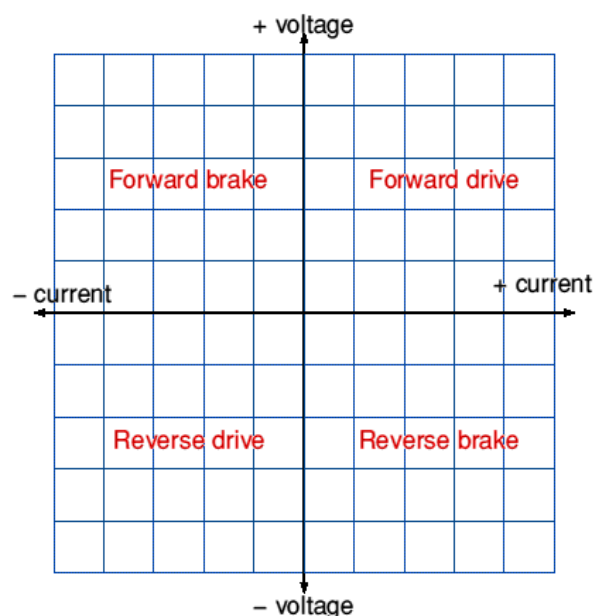


Figura 1.20: Quadranti di funzionamento

I quattro *switch* del circuito sono generalmente realizzati con dei *transistor*, accoppiati a dei diodi di ricircolo. Chiudendo e aprendo i vari "interruttori" si possono ottenere varie forme d'onda e modi di funzionare del motore. Ad esempio chiudendo il transistor in alto a sinistra e quello in basso a destra di figura 1.19 si ottiene che la corrente circola nel verso indicato dalla freccia verde, ed il motore lavora nel quadrante di *Forward Drive* di figura 1.20. Da evitare è la chiusura contemporanea di entrambi i transistor dello stesso braccio, in quanto la corrente di cortocircuito potrebbe andare a rovinare l'integrato o l'alimentazione. Per un'analisi più approfondita della tecnica PWM e del ponte ad H si rimanda a numerosi testi presenti in letteratura riguardanti l'argomento.

I ponti ad H forniti nella scheda sono il modello MC33931, dotato di numerose protezioni e della possibilità di controllare coppia, frenata e modulazione PWM (con frequenze fino a 11 kHz). Di particolare interesse è il feedback analogico in output

della corrente circolante sui rami del ponte ad H, che può essere letto tramite l' ADC (*analog-to-digital converter*) presente sulla scheda del microcontrollore.

In conclusione è utile far notare come il modello della scheda di potenza fornito agli studenti abbia un pin (denominato IN1) connesso a massa, che di fatto impedisce il pilotaggio inverso del motore e quindi l'accesso alla frenata. È stata necessaria una modifica hardware: alzare il pin numero 43 dell'integrato per connettervi un conduttore da collegare alla scheda del processore, dal quale controllarlo attraverso il software. La modifica è visibile nella foto di figura 1.15 (i fili rosso e blu uscenti dai due ponti ad H).

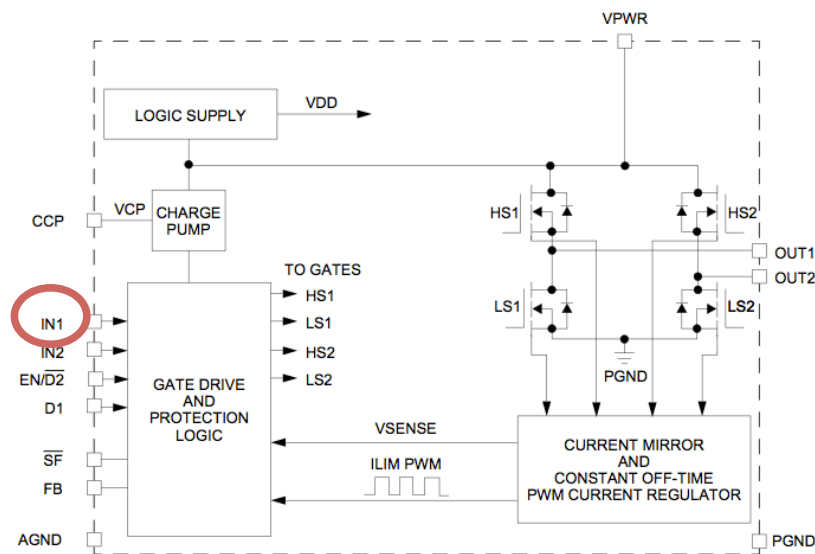


Figura 1.21: Schema a blocchi dell'integrato MC33931

1.4.3 Microprocessore TRK-MPC5604B

La MCU (*Micro Controller Unit*) TRK-MPC5604B è un controllore a 32 bit basato su architettura *Power Architecture®*. È parte di una famiglia più ampia di processori ideati per l'automotive.

Il processore che troviamo nella scheda, denominato e200z0, è progettato per applicazioni di controllo embedded che non puntano ad elevate performance, ma piuttosto a bassi costi e consumi.

Molto utili in fase di test sono i 4 pulsanti (più uno di reset) di cui è dotata la scheda per la selezione di diversi programmi da eseguire senza caricare nuovamente per via USB il software.

Di interesse non trascurabile è sicuramente l' ADC a 10 bit che troviamo nella scheda, il quale permette la conversione di tutti i segnali analogici provenienti dalla

Motor Drive Board in segnali digitali, permettendo quindi l'accesso a tali informazioni da parte del processore, e quindi una lettura con conseguente interpretazione da parte degli studenti, che si interfacciano ad esso con il PC attraverso un cavo USB. Si rimanda al capitolo 3 per una analisi più approfondita dell' ADC.

Prima di elencare le caratteristiche tecniche della MCU è utile anche una panoramica sull' eMIOS ed i suoi blocchi.

L'acronimo eMIOS è usato per indicare l' *enhanced Modular IO Subsystem*, ed i moduli eMIOS offrono funzionalità specifiche per generare o misurare eventi temporali. È utilizzato per inviare i comandi di controllo per gli attuatori della smart car che il processore genera dal codice. Ogni canale dispone di un *counter* al suo interno, mentre per la sincronizzazione fra vari canali si hanno a disposizione un certo numero di *counter buses* per avere un riferimento comune di tempo. Il modulo eMIOS è ciò che gestisce l'invio del segnale PWM ai motori che regolano trazione e sterzo.

Caratteristiche dei moduli eMIOS:

- 2 blocchi eMIOS da 28 canali ciascuno
- 50 canali dispongono di funzionalità di tipo Output PWM Triggered (OPWMT)
- 6 canali con funzioni esclusivamente di tipo IC/OC
- Un Prescaler globale
- Registri a 16 bit
- 10 counter buses a 16 bit
- I contatori B,C,D,E possono essere comandati solo dai Unified Channel 0, 8, 16 e 24 rispettivamente
- Il contatore A può essere comandato solo dal Canale 23

Inoltre i canali eMIOS possono essere configurati per operare nelle seguenti modalità:

- Single Action Input Capture (SAIC)
- Single Action Output Compare (SAOC)
- Input Pulse Width Measurement (IPWM)
- Input Period Measurement (IPM)
- Double Action Output Compare (DAOC)
- Modulus Counter (MC)
- Modulus Counter Buffered (MCB)
- Output Pulse Width and Frequency Modulation Buffered (OPWFMB)
- Output Pulse Width Modulation Buffered (OPWMB)

- Output Pulse Width Modulation with Trigger (OPWMT)
- Center Aligned Output Pulse Width Modulation Buffered (OPWMCB)

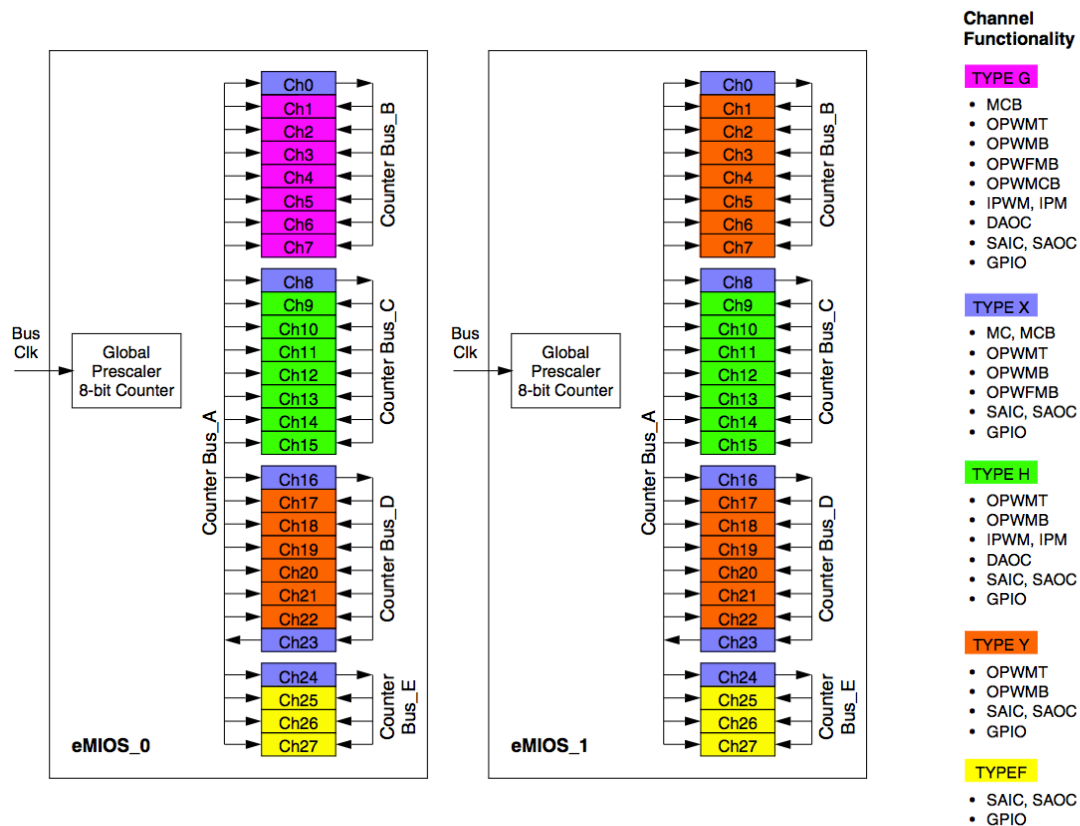


Figura 1.22: Configurazione dei blocchi eMIO

Le caratteristiche del processore fornito nel kit sono le seguenti:

CORE

- PowerPC e200z0 core running 48-64 MHz
- VLE ISA instruction set for superior code density
- Vectored 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)

COMMUNICATIONS

- 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
- Debug: Nexus 2+__
- 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

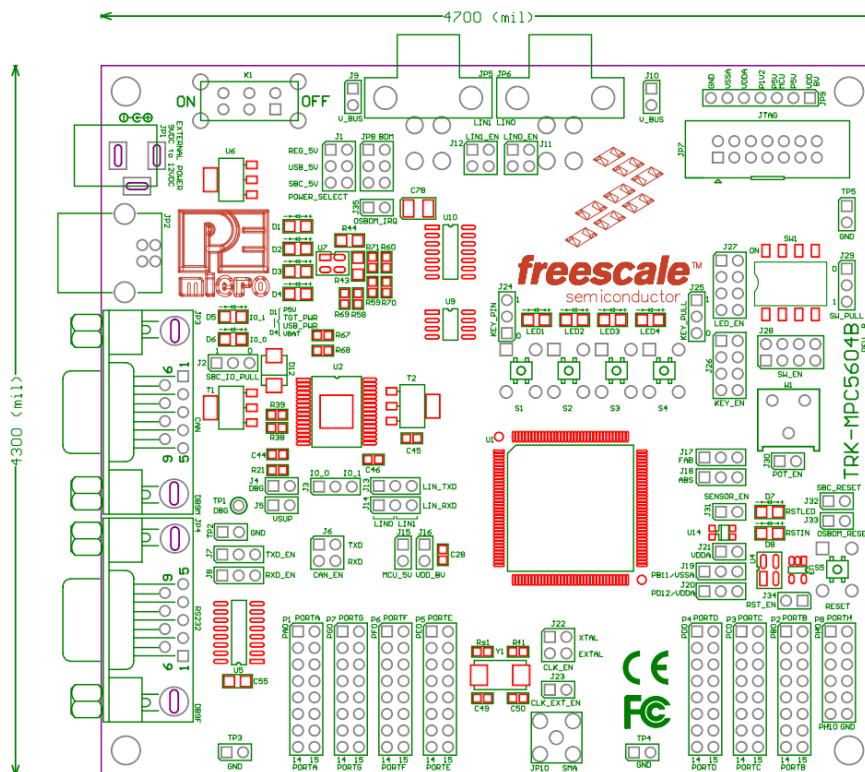


Figura 1.22: TRK-MPC5604B top view

CAPITOLO 2

Tecniche di acquisizione dei segnali mediante sensore di visione

2.1 Analisi del sensore TSL1401CL e relativo montaggio

2.1.1 Caratteristiche Tecniche

La camera per la visione utilizzata nella Freescale Cup si basa su un sensore prodotto dall'azienda *TAOS®*, il cui modello in questione è denominato "TSL1401CL". È sostanzialmente un convertitore luce-frequenza o luce-tensione (è il nostro caso). Segue uno schema che ne illustra il concetto alla base funzionamento, analizzato approfonditamente in questo paragrafo.

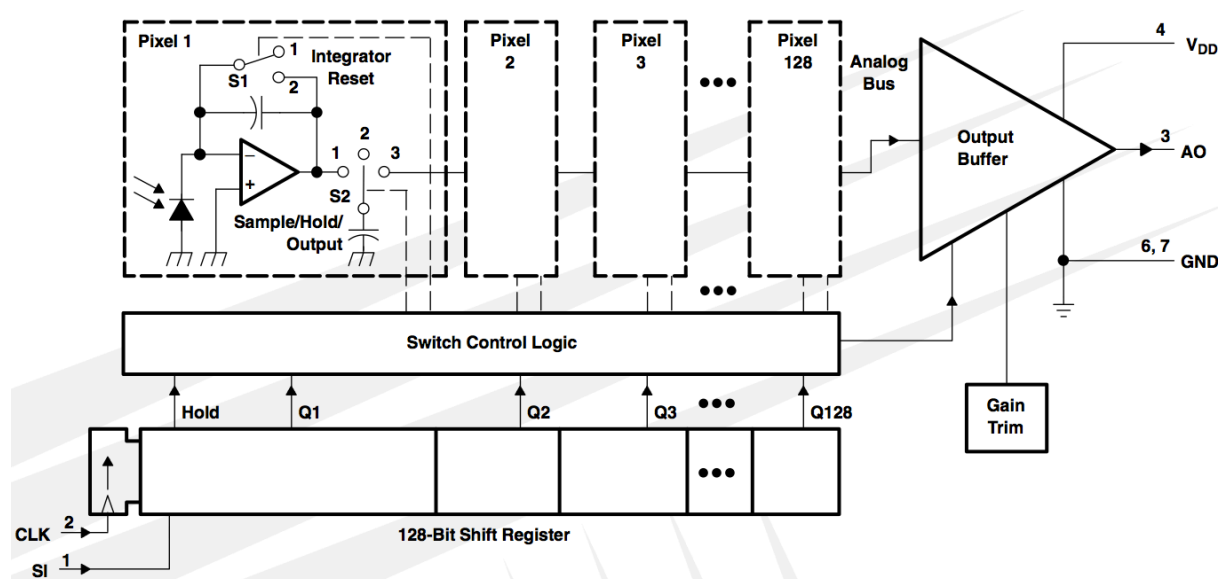


Figura 2.1: Schema del funzionamento del sensore di luminosità

Alcuni dettagli tecnici ricavabili dal *datasheet*:

- Risoluzione di 400 DPI
- Basso ritardo di immagine
- Frequenza di lavoro fino a 8 MHz

- Alimentazione unica a 3 V o 5 V
- Nessuna resistenza di carico richiesta per il funzionamento
- Ampio Dynamic Range 4000:1 (72 dB)
- Fotodiodi con superficie di $3524.3 \mu\text{m}^2$ ciascuno
- Distanza fra un fotodiodo ed un altro $8 \mu\text{m}$
- Tempo di esposizione: $267 \mu\text{s}$ a 68 ms

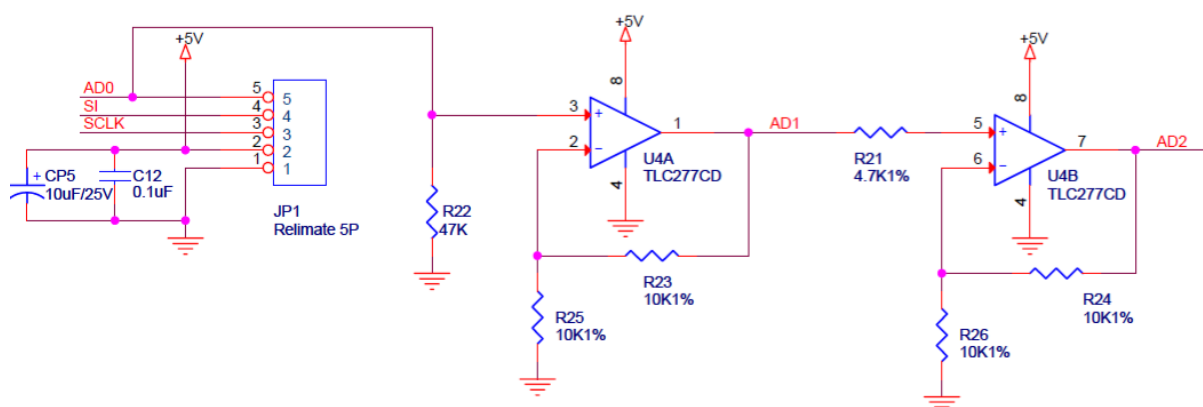


Figura 2.2: Schema dell'interfaccia elettronica presente tra *motor board* e telecamera

Dallo schema di figura 2.2 è possibile osservare come il segnale di output (AD0) sia amplificato due volte da amplificatori operazionali e quindi ci sia data la possibilità di scelta tra due segnali: AD1 o AD2. Configurando le porte attraverso il codice sarà possibile scegliere uno dei due o anche entrambi.

2.1.2 Principi di funzionamento

Come vengono interpretati i diversi livelli di luminosità ambientale? La luce che rimbalza sull'ambiente circostante passa attraverso le lenti, e l'ultima di esse la indirizza sul sensore. Essendo quest'ultimo dotato di un'array di piccoli capacitori che si caricano relativamente all'intensità luminosa, il processo di carica avviene

simultaneamente. In seguito il sensore rilascia il valore di *output* mediante un segnale analogico un pixel per volta.

Dal momento che la camera è di tipo lineare sarà impossibile ottenere una visione completa dell'ambiente di fronte ad essa, pertanto ci sarà fornita solo una "linea" di esso. Questa linea è completamente dipendente dalla distanza dalle lenti e dalle capacità riflettenti dell'oggetto puntato.

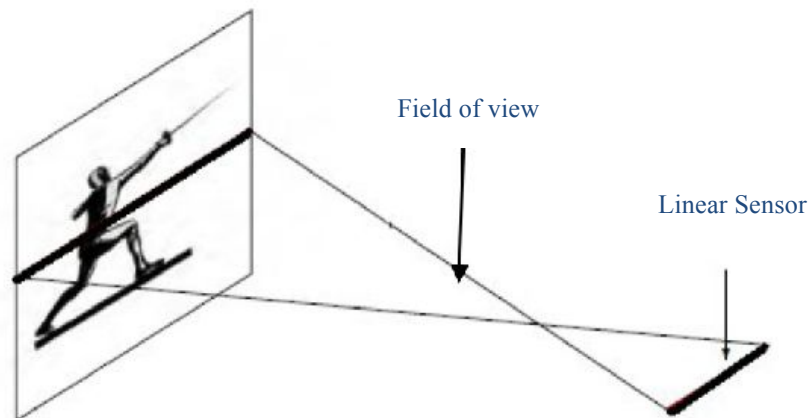


Figura 2.3: Esempio di acquisizione immagine di un sensore lineare

Per un utilizzo standard della camera, l'utente dovrebbe interessarsi solamente ai seguenti tre segnali:

- CLK (clock)
- SI (serial input)
- AO (analog output)

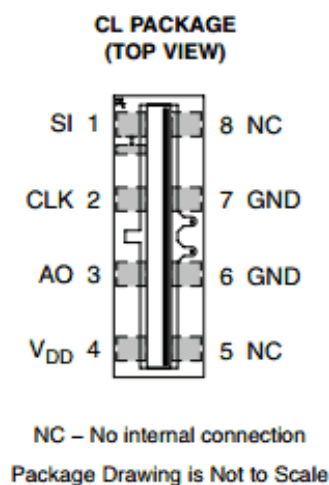


Figura 2.4: Segnali di input e output del sensore TSL1401CL

Nel nostro caso i segnali di input sono provenienti dalla *motor board* (vedi cap. 1.4.2) e sono originati dal microcontrollore. È di facile intuizione il fatto che all'aumentare della frequenza dei segnali di input, osserveremo una decrescita proporzionale del tempo di integrazione.

Il tempo di integrazione è il tempo durante il quale la carica generata va ad accumularsi su un condensatore. È un parametro molto importante, in quanto:

- un lungo periodo di integrazione implica una saturazione dei pixel anche in presenza di scarsa illuminazione ambientale
- un periodo di integrazione corto invece non fa guadagnare abbastanza carica ai vari pixel anche se l'illuminazione ambientale è elevata.

Il periodo di integrazione dell' array è determinato dal tempo impiegato per scorrere tutti i pixel e da quello per trasferire la carica, ma essendo quest'ultimo costante, si può calcolare il tempo minimo di integrazione solo in funzione del periodo, del numero di pixel dell'array e della frequenza di clock:

$$T_{int(min)} = \frac{1}{\text{maximum clock frequency}} \cdot (n - 18)\text{pixel} + 20 \mu s$$

dove n è il numero di pixel di cui è composto l' array.

Nel caso del sensore TSL1401CL, è possibile calcolare il valore del minimo periodo di integrazione grazie alla conoscenza di $n = 128$ e della frequenza massima di clock pari a 8 MHz.

$$T_{int(min)} = \frac{1}{8 \text{ MHz}} \cdot (128 - 18) + 20 \mu s = 33,75 \mu s$$

Il funzionamento del sensore è sintetizzabile in 3 fasi principali:



Durante la fase di **SAMPLE** inizia quando gli *switch* S1 e S2 di figura 2.1 si portano in posizione 1. L'energia luminosa incidente sul fotodiode genera una piccola corrente la quale, durante il periodo di integrazione, entra nel circuito integratore e va a caricare un condensatore collegato all'uscita mediante un interruttore analogico. La lettura della tensione analogica in uscita e l'operazione di reset degli integratori sono gestite dall'azione combinata di una logica di reset e da un registro di shift a 128 bit. Il ciclo di lettura inizia con la fase di **HOLD** quando il livello logico del segnale SI diventa "alto" e gli *switch* S2 passano in posizione 2. In corrispondenza dell' "1" nel SI viene generato un segnale di *Hold*, il quale viene subito trasmesso a tutti gli interruttori analogici che scollegano i condensatori di campionamento dai rispettivi circuiti di integrazione; nello stesso momento inizia anche il periodo di reset dei circuiti integratori (*switch* S1 in posizione 2). Segue la fase di **OUTPUT**, nella quale ciascun condensatore viene collegato in modo sequenziale all'amplificatore di uscita, il cui compito è generare la tensione analogica AO. Per fare questo gli *switch* S2 passano in posizione 3. Come è possibile osservare in figura 2.5, dopo 18 cicli si ha il completo reset dei circuiti integratori, ed i pixel iniziano a caricarsi nuovamente al 19° ciclo di clock, quando gli *switch* S1 dei primi 18 pixel tornano in posizione 1. Durante questi 18 cicli le prime 18 tensioni di *output* vengono lette mentre avviene la fase di *reset*. Dal diciannovesimo ciclo in poi i pixel successivi vengono letti sequenzialmente: per ognuno di essi S2 passa in posizione 3, viene letta la carica, poi S2 torna in posizione 1.

Dopo 129 cicli tutti i valori dei vari pixel sono stati rilasciati tramite il segnale AO, il quale acquisisce uno stato ad alta impedenza; da quel momento è possibile mandare un nuovo segnale di input seriale (SI) per iniziare il rilascio di un nuovo AO. Il segnale AO è dato dall'uscita di un amplificatore operazionale in modalità buffer tri-state (adattamento di impedenza).

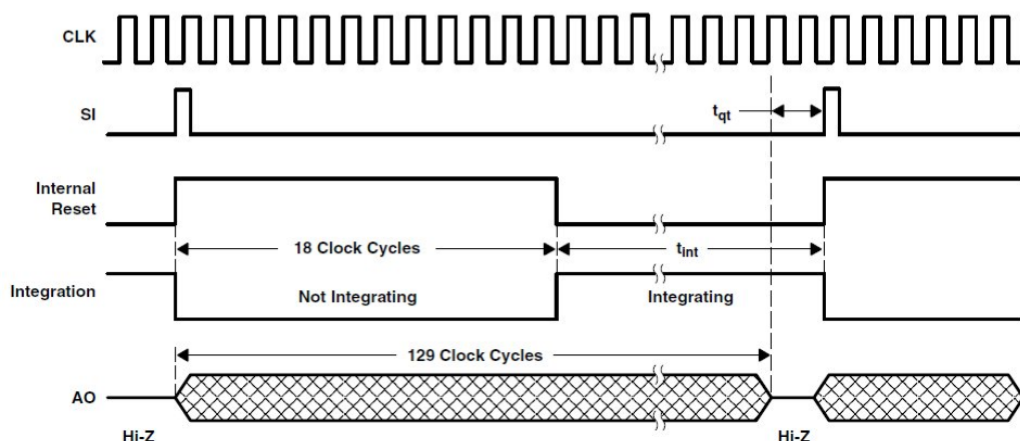


Figura 2.5: Relazioni temporali fra i segnali del sensore

Con alimentazione a 5 V il segnale di output può variare da 0 V in assenza di luce a 4,8 V in saturazione, cioè per la massima luce rilevabile. Una luminosità media si aggira intorno ai 2 V.

La tensione corrispondente all'intensità luminosa acquisita in un periodo sarà resa in output attraverso l' AO solo al periodo successivo, con la velocità di un pixel ogni ciclo di clock. In pratica viene fornita in output la tensione accumulata nel periodo di integrazione precedente finché il campione attuale si sta integrando.

Esiste una formula per calcolare il segnale di output:

$$V_{out} = V_{drk} + R_e E_e t_{int}$$

dove:

- V_{out} è la tensione di uscita analogica relativa al bianco
- V_{drk} è la tensione di uscita analogica relativa all'assenza di luce
- R_e è la risposta del sensore ad una data lunghezza d'onda della luce
- E_e consiste nella radiazione incidente
- t_{int} è il tempo di integrazione (in secondi)

Inoltre i tre segnali in analisi devono essere sincronizzati fra loro per consentire un corretto funzionamento della camera, quindi i segnali SI e CLK vengono sistemati in modo opportuno:

- entrambi devono essere periodici con lo stesso periodo
- entrambi devono essere perfettamente sfasati di mezzo periodo
- di conseguenza l' output AO sarà perfettamente in fase con CLK

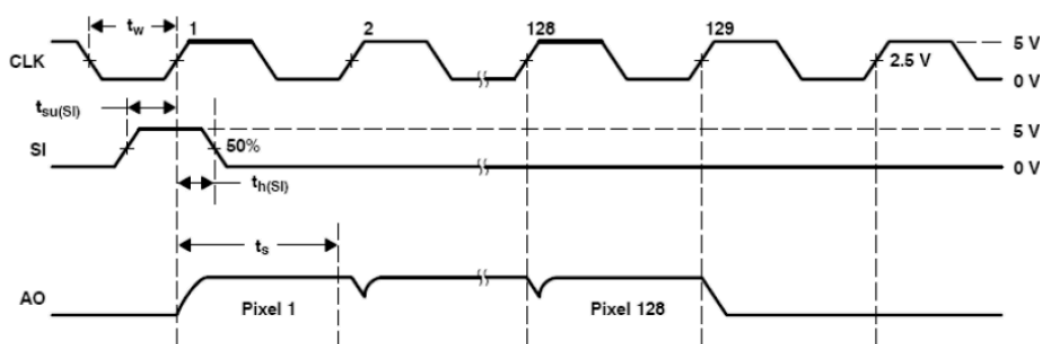


Figura 2.6: Sfasamento e periodi dei segnali CLK, SI e AO.

2.1.3 Montaggio della camera

Una questione che viene lasciata affrontare completamente agli studenti è quella relativa al montaggio del sensore di visione, in quanto nel kit non vengono forniti alcun supporto o basetta da usare per questo scopo. È infatti una scelta fra le più importanti da prendere, ed è bene ponderare su vantaggi e svantaggi che ogni possibile posizione può offrire, con particolare riguardo ai *trade-off* tra cui discernere, perchè non ci sarà una posizione perfetta in termini assoluti, ma solo alcune migliori di altre.

Tra i fattori che influenzano questa scelta troviamo sicuramente:

- la massima distanza focale raggiungibile dalla camera che si ha a disposizione
- la necessità di avere una visione della linea particolarmente nitida da vicino per approcci software di tipo *line-following*
- la necessità di vedere il più lontano possibile per approcci software di tipo predittivo
- la disponibilità di fonti luminose che illuminino il tracciato
- ecc. ecc.

I supporti utilizzati nel nostro caso sono di materiale ferroso forati in modo da permettere più altezze tra cui scegliere. È molto importante che il fissaggio avvenga nel modo più stabile possibile, poichè il veicolo è sottoposto ad un certo numero di vibrazioni durante il moto. Si è scelto un fissaggio come quello in figura per facilità di montaggio e stabilità del supporto.

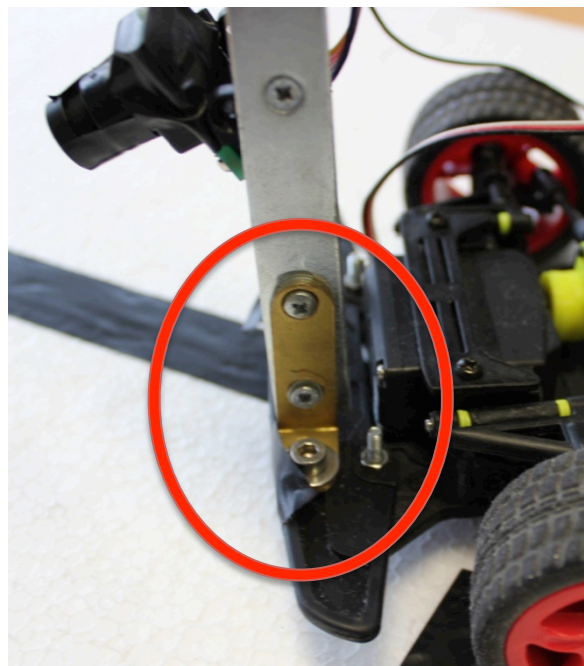


Figura 2.7: Dettaglio della base del fissaggio dei supporti

In seguito si è resa necessaria l'aggiunta di tiranti in quanto con l'aumentare della velocità le vibrazioni e le oscillazioni dei supporti disturbavano l'acquisizione del segnale soprattutto per quanto riguarda la seconda telecamera, posizionata il più in alto possibile (vedi paragrafo 2.2.2 per informazioni aggiuntive sull'utilizzo di due sensori di visione).

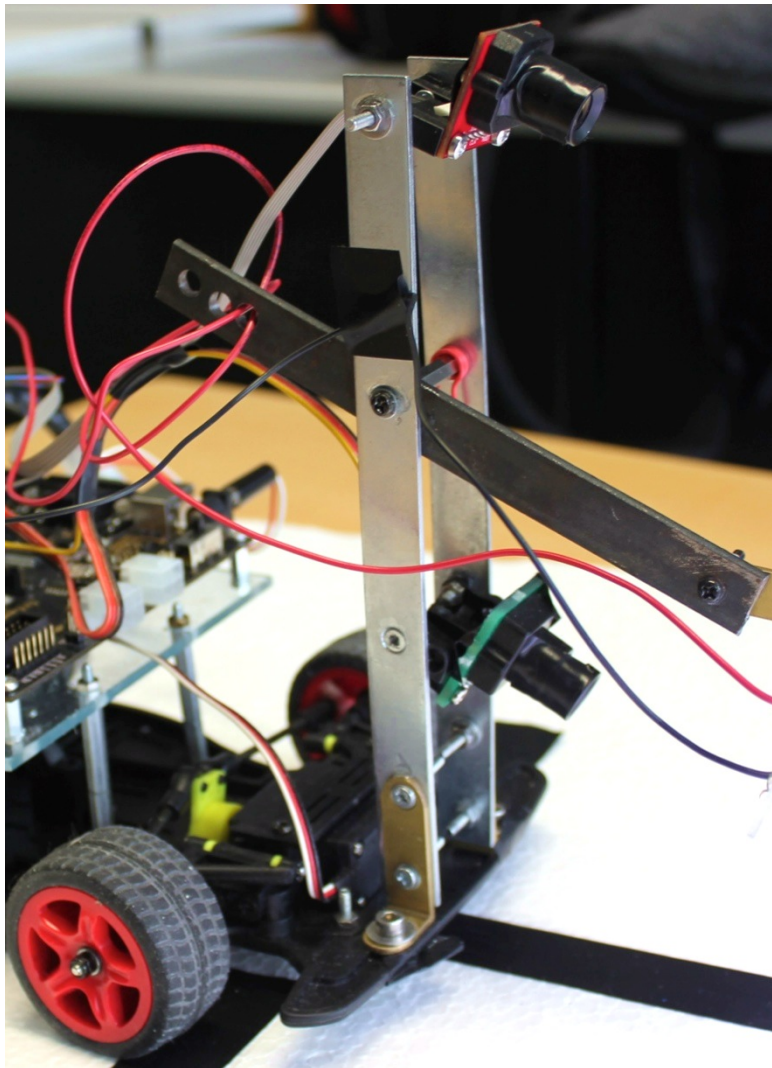


Figura 2.8: Supporto per il montaggio delle camere

La regolazione dell'angolo di visione è stata possibile grazie ad un fissaggio che avviene tramite viti e dadi. Grazie a questa scelta dei supporti è stato possibile scegliere tra diverse altezze e angolazioni.

Si consiglia di dedicare abbastanza tempo alla ricerca di una buona posizione dei sensori di visione, anche attraverso numerosi test su pezzi diversi di tracciato, essendo la questione del posizionamento e montaggio delle camere la base di tutte le analisi e strategie che verranno elaborate successivamente.

Inoltre per evitare che la luminosità ambientale disturbi il segnale è bene coprire il retro dei sensori di visione con dello scotch isolante scuro.

2.2 Introduzione al software Codewarrior ed al linguaggio C

Prima di affrontare i seguenti paragrafi dell'elaborato, contenenti estratti di codice in linguaggio C, è utile una panoramica su *Codewarrior* ed il linguaggio di programmazione utilizzato.

La programmazione del microcontrollore avviene tramite un IDE (Integrated Development Environment) attualmente sviluppato da *Freescale Semiconductors*, e inizialmente ideato da *Metrowerks*. Ne esiste una versione per C, una per C++ ed una per *Assembly*.

Codewarrior è stato fondamentale per *Apple inc.* nel passaggio dall'architettura 68K a *PowerPC®*. È ora utilizzato per sviluppare software da usare in applicazioni di tipo *embedded*.

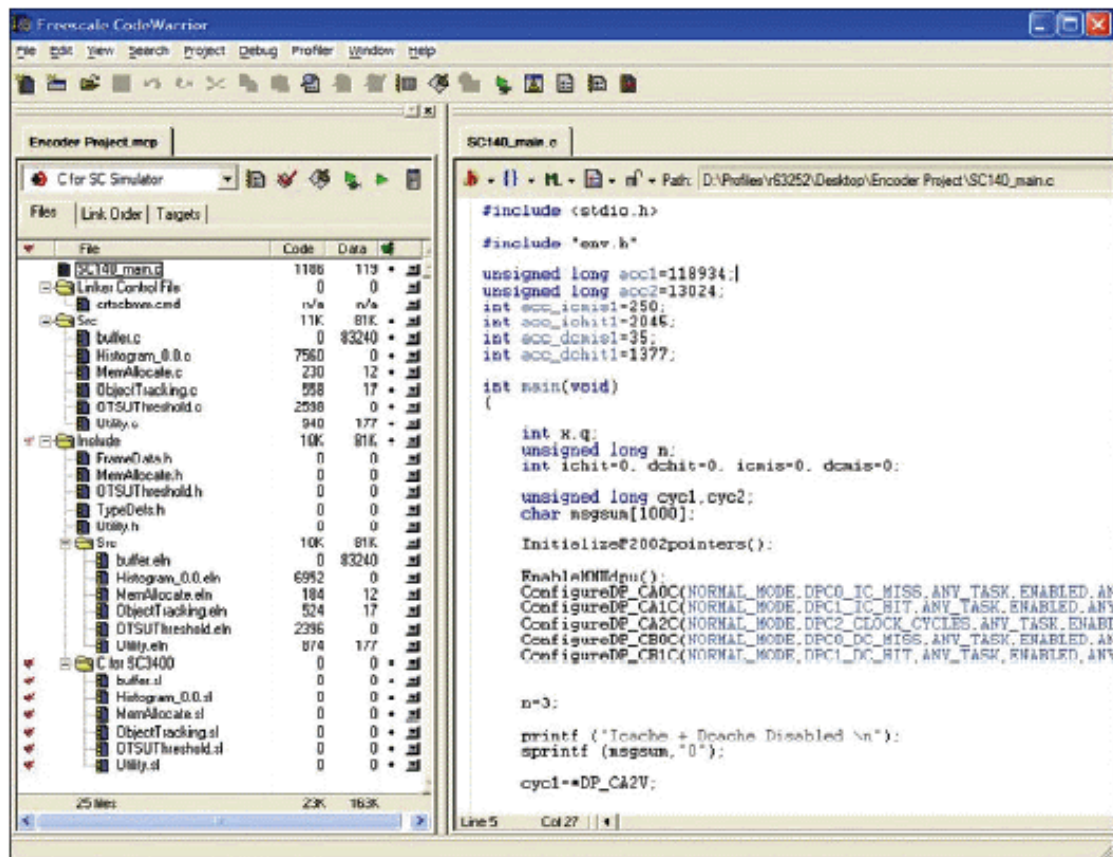


Figura 2.9: GUI di Codewarrior

Le istruzioni da seguire per poter "utilizzare" il software creato sono le seguenti. Innanzitutto è necessario compilarlo, e questo è possibile farlo tramite il menu *Project* → *Compile* oppure premendo *ctrl+F7*. Segue il *debug* del programma, attivabile tramite pressione del tasto *Debug* o premendo *F5*. Il programma può ora essere

avviato direttamente dal PC se salvato in *Ram*, oppure può essere salvato nel processore se salvato in *internal flash*. La scelta può essere effettuata da un menu a tendina. Ovviamente la prima delle due scelte ha come svantaggio quello di obbligare l'utente a mantenere attaccato il cavo USB che connette smart car e PC, ma offre d'altro canto la possibilità di poter leggere le variabili globali che sono salvate al momento dello stop manuale del programma. La modalità *internal flash* permette invece di caricare il programma nel processore del veicolo e quindi di renderlo autonomo dal PC.

Il linguaggio di programmazione utilizzato nel nostro caso è il linguaggio *C*. È un linguaggio general-purpose sviluppato da Dennis Ritchie agli *AT&T Bell Labs* negli anni intorno al 1970. È diventato in seguito uno dei più utilizzati di tutti i tempi.

C ha operatori per:

- arithmetic: +, -, *, /, %
- assignment: =
- augmented assignment: +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=
- bitwise logic: ~, &, |, ^
- bitwise shifts: <<, >>
- boolean logic: !, &&, ||
- conditional evaluation: ? :
- equality testing: ==, !=
- calling functions: ()
- increment and decrement: ++, --
- member selection: ., ->
- object size: sizeof
- order relations: <, <=, >, >=
- reference and dereference: &, *, []
- sequencing: ,
- subexpression grouping: ()
- type conversion: (typename)

La sintassi utilizzata nel programma prevede un uso massiccio di cicli *for*, ovvero di istruzioni ripetute continuamente per un determinato numero di volte a seconda di determinate condizioni.

2.3 Setup dei sensori di visione

Lo scopo primario della smart car è quello di seguire la linea nera al centro del percorso, ed una buona comprensione di come impostare i sensori al meglio può fare la differenza. Lo scopo di questa sezione dell'elaborato è quello di andare ad esporre i due diversi *setup hardware* utilizzati dal team, di capirne l'impostazione vi software e di analizzare i vantaggi derivanti dall'aggiunta di un secondo sensore di visione.

2.3.1 Idea iniziale con una camera

I primi risultati soddisfacenti riguardo ad una buona posizione della camera per acquisire la linea del tracciato sono stati trovati attraverso un approccio semplice ma efficace.

La camera è stata posizionata ad un'altezza di circa 30 cm dal tracciato, con una distanza di *look-ahead* di 30 cm. Questa soluzione è stata trovata sperimentalmente osservando il segnale di output della camera tramite un DSO (Digital Storage Oscilloscope) e provando varie combinazioni tra distanze focali e posizioni. Questo è stato possibile collegando il puntale della sonda all'uscita AO (Analog Output), segnalata sulla schedina, così come gli altri 4 segnali; una volta collegato il puntale basta semplicemente eseguire dal *main* del programma una chiamata alla funzione CAMERA() mettedola in un loop infinito.



Figura 2.10: Segnale AO della camera puntata verso la linea nera visto in un DSO.

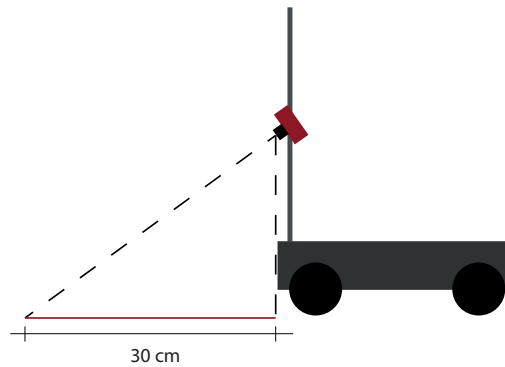


Figura 2.11: Look-ahead distance

Segue un estratto di codice in linguaggio C ed una breve spiegazione di esso.

```
void initPads (void)
{
    SIU.PCR[50].R = 0x2000;    /* MPC56xxB: Initialize PD[2] as GPI[6] */
}

void initADC(void)
{
    ADC.MCR.R = 0x00000000;    /* Initialize ADC one shot mode*/
    ADC.NCMR[0].R = 0x00000078; /* abilita la conversione canali 3,4,5,6 */
    ADC.CTR[0].R = 0x00008606; /* Conversion times for 32MHz ADClock */
}

void Delaycamera(void){
    for(dly=0;dly<20000;dly++);
}
```

Nella funzione `initPads` si inizializza il pin `PD[2]` come canale di precisione dell' ADC. Seguono poi l' abilitazione della modalità di funzionamento *One Shot*, la lettura del canale 6 e la scelta del clock a 32 MHz. Infine si definisce un *delay* pari a 20000 cicli di clock, che verrà usato nella funzione seguente.

```
void CAMERA(void)
{
    SIU.PCR[27].R = 0x0200;    /* Program the Sensor read start pin as output*/
    SIU.PCR[29].R = 0x0200;    /* Program the Sensor Clock pin as output*/
    SIU.PGPD0[0].R &= ~0x00000014; /* All port line low */
    SIU.PGPD0[0].R |= 0x00000010; /* Sensor read start High */
    Delay();
    SIU.PGPD0[0].R |= 0x00000004; /* Sensor Clock High */
    Delay();
    SIU.PGPD0[0].R &= ~0x00000010; /* Sensor read start Low */
    Delay();
    SIU.PGPD0[0].R &= ~0x00000004; /* Sensor Clock Low */
    Delay();
    for (i=0;i<128;i++)
    {
        Delay();
        SIU.PGPD0[0].R |= 0x00000004; /* Sensor Clock High */
        ADC.MCR.B.NSTART=1; /* Trigger normal conversions for ADC0 */
        while (ADC.MSR.B.NSTART == 1) {};
        adcddata = ADC.CDR[6].B.CDATA;
        Delay();
        SIU.PGPD0[0].R &= ~0x00000004; /* Sensor Clock Low */
    }
}
```

```

        Result[i] = (uint8_t)(adcdata >> 2);
    }
    Delaycamera();
}

void main (void)
{
    ...    /* Inizializzazione del clock e dei vari canali eMIOS */
    for(;;) /* Loop forever */
    {
        CAMERA();
    }
}

```

L'estratto di codice soprastante esemplifica il codice della funzione fondamentale da eseguire e la sua chiamata nel *main* del programma. Si è omessa l'inizializzazione dei vari canali dell' eMIOS poichè esula dalla trattazione di questo elaborato.

Si può notare l'inizializzazione delle porte PCR[27] e PCR[28], corrispondenti a PB[11], e a PB[13], rispettivamente. Questa abilitazione consente la generazione dei segnali SI e CLK, sfasati di mezzo periodo grazie alla chiamata di *Delay()*.

Dopo i 128 cicli del *for* presente in CAMERA() si avranno a disposizione dei valori numerici in uscita dall' ADC che vengono salvati temporaneamente in una variabile *adcdata* e, dopo un *cast* al tipo *uint_8* (*unsigned integer* a 8 bit), salvati nella variabile globale *array Result*. Ad ogni nuova chiamata della funzione questi valori verranno sovrascritti. È possibile, tramite il software Codewarrior (vedi capitolo 3), leggere i valori salvati nelle variabili globali (in questo caso la variabile *Result*), scoprendo così che variano da 0 (nero/assenza di luce) a 255 (massima luminosità).

2.3.2 Aggiunta della seconda camera

Una importante svolta dal punto di vista dell' efficienza e della consapevolezza dell'analisi del tracciato da parte della *smart car* è giunta grazie all' aggiunta di una seconda camera. Questo fatto ha reso necessario riadattare la posizione anche dell'altra camera. Dopo svariati test, l'idea che ha prevalso è stata quella di utilizzare la camera alta per "prevedere" la presenza di una curva imminente e quindi poter far accelerare il veicolo il più possibile in presenza di un rettilineo fino a quando il tracciato lo avesse permesso. Il compito assegnato al sensore di visione basso invece era quello di acquisire la linea con la maggior chiarezza possibile mentre la parte di software per il controllo dello sterzo cercava di non far uscire mai dal campo visivo del sensore la linea nera.

La distanza di *look-ahead* scelta è stata di 20 cm per la camera bassa, montata ad altezza di 20 cm, e 60 cm per quella alta, montata a 40 cm da terra. Ovviamente per poter avere una visione sufficientemente definita della linea a 60 cm dal veicolo si è resa necessaria l'introduzione di una fonte luminosa esterna, come spiegheremo nel prossimo paragrafo.

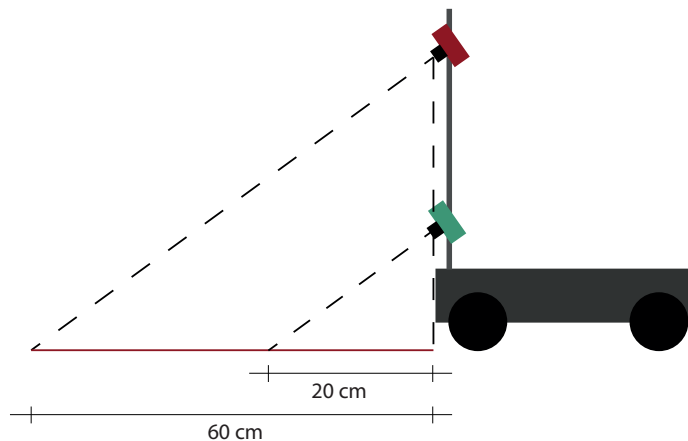


Figura 2.12: Look-ahead distance delle 2 camere

La seconda camera ricevuta, visibile in figura 2.12, differisce non poco da quella presente nel kit per qualità del sensore di visione. Infatti essa, sempre prodotta dalla TAOS, è il modello TSL1401R-LF, e nonostante il datasheet riporti gli stessi dati del modello TSL1401CL, si può ben notare come il datasheet sia datato “maggio 2006”, mentre per il sensore nel kit la data riportata sul datasheet è “luglio 2011”. È nella pagina di riferimento del sensore di visione presente nel *Freescale Cup Wiki* che la reale differenza tra i due sensori è spiegata: nel 2011 l'azienda ha provveduto ad una revisione, migliorando lo stadio di amplificazione dell'output analogico.

Il problema più grande incontrato con il sensore meno recente è stato quello di un picco di tensione costante a circa tre quarti dell'array di fotodiodi, che ha reso necessario il cambio dell'intero algoritmo di acquisizione della linea (vedi paragrafo 2.5).



a.



b.

Figura 2.13: a) Sensore TSL1401CL. b) Sensore TSL1401R-LF

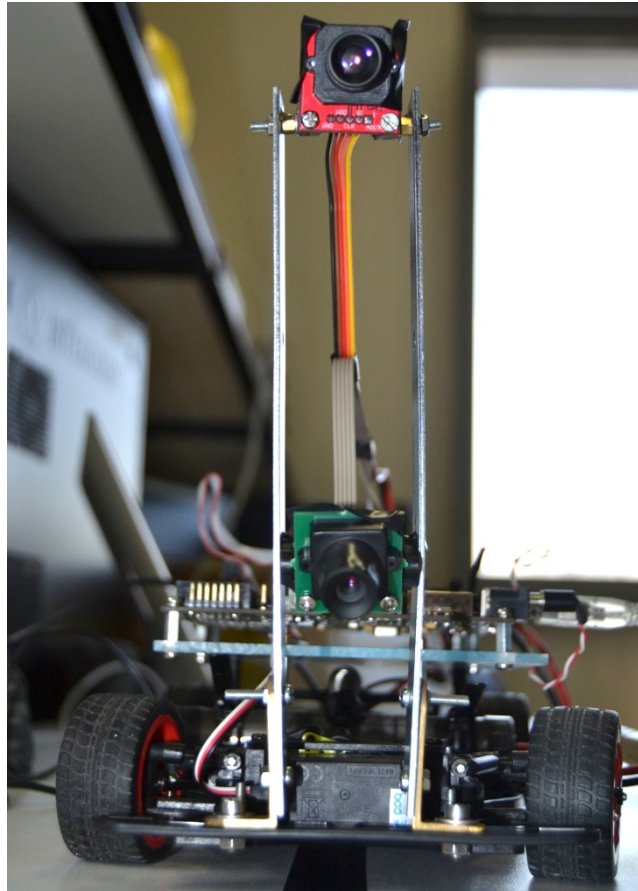


Figura 2.14: Le due camere montate

Gestire una seconda camera tramite la Motor Drive Board richiede però l'aggiunta di un connettore modificato, in grado di interconnettere due camere contemporaneamente. Tramite questo connettore è possibile mandare tutti i segnali di *input* ad entrambe le camere. La lettura dell' *Analog Output* della camera aggiuntiva viene però fatta direttamente dalla scheda del processore, alla quale è connesso il solo conduttore del segnale di AO ad un pin.

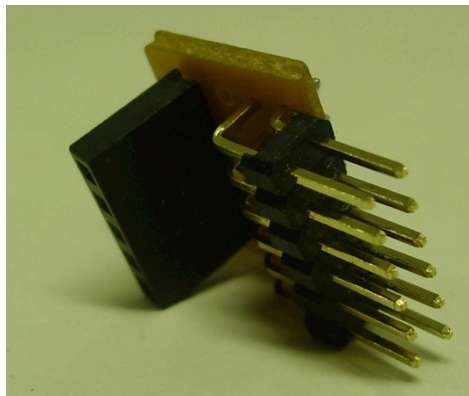


Figura 2.15: Connettore per le camere

Segue il codice utilizzato per abilitare i pin PD[0] e PD[2] ai quali sono stati collegati i cavetti per l'output.

```
void initPads (void) {
    SIU.PCR[48].R = 0x2000; /* MPC56xxB: Initialize PD[0] as GPI[4] : segnale
telecamera alta*/
    SIU.PCR[50].R = 0x2000; /* MPC56xxB: Initialize PD[2] as GPI[6] : sasegnale
telecamera bassa*/
}
```

È necessario aggiungere un paio di righe anche al codice del metodo CAMERA(), nella parte relativa al ciclo *for*, per andare a salvare i dati numerici dell' ADC provenienti dalle due diverse camere in due array differenti.

```
for (i=0;i<128;i++)
{
    Delay();
    SIU.PGPD0[0].R |= 0x00000004; /* Sensor Clock High */
    ADC.MCR.B.NSTART=1; /* Trigger normal conversions for ADC0 */
    while (ADC.MSR.B.NSTART == 1) {};
    adcddata = ADC.CDR[6].B.CDATA;
    adcddataSopra = ADC.CDR[4].B.CDATA;
    Delay();
    SIU.PGPD0[0].R &= ~0x00000004; /* Sensor Clock Low */
    Result[i] = (uint8_t)(adcddata >> 2);
    Result_Camera_Alta[i] = (uint8_t)(adcddataSopra >> 2);
}
```

2.4 Faro a LED

L'introduzione di un faro a LED (*Light-Emitting Diodes*) è consigliabile, in quanto migliora di molto la qualità dell' *Analog Output* dei sensori di visione e permette alla *smart car* di "vedere" nei vari tunnel posizionati sono lungo il percorso.

2.4.1 Prima versione

La prima versione ideata prevedeva di mantenere la tensione ai capi dei diodi costante in modo da avere luminosità costante. Alimentando la scheda a LED direttamente dalla batteria (per semplicità di collegamenti) si sarebbero presentati probemi poichè la tensione della batteria cala con l'utilizzo della smart car, e quindi

anche la tensione fornita alla scheda. Grazie allo schema circuitale utilizzato è possibile mantenere una alimentazione di 5 V sui LED fino a circa 5 V di tensione forniti dalla batteria.

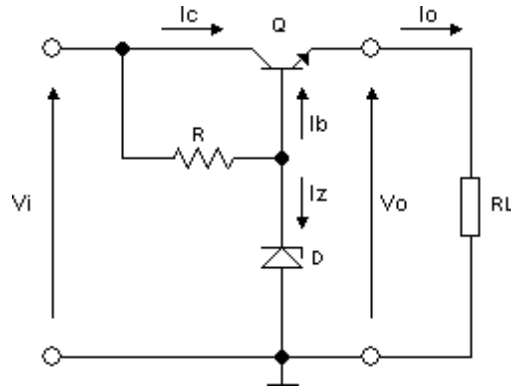


Figura 2.16: Schema circuitale di circuito stabilizzatore di tensione con diodo zener e BJT

$$I_0 \cong V_z - V_{BE}$$

dove V_{BE} è solitamente $\cong 0.7 V$;

$$I_B = \frac{I_C}{\beta}$$

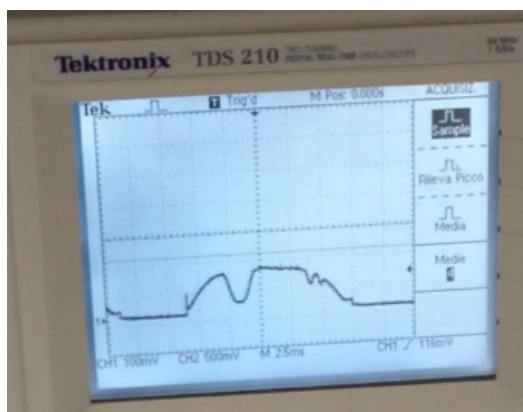
dove β è un parametro caratteristico dei BJT al silicio; inoltre, essendo I_B molto piccola, possiamo trascurarla e asserire che:

$$I_0 \cong I_C \rightarrow I_B = \frac{I_0}{\beta}$$

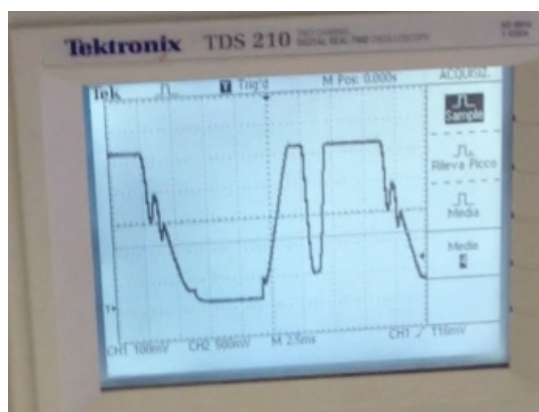
Il calcolo della resistenza R è allora ottenibile da:

$$R = \frac{V_{i,min} - V_z}{I_B + I_z}$$

È possibile osservare in figura 2.15 come il segnale per la camera bassa migliori sensibilmente all'accensione della scheda a led. Grazie ad una migliore luminosità è possibile diminuire i *delay* presenti nel metodo CAMERA() per velocizzare l'acquisizione del segnale diminuendo il tempo di integrazione delle camere.



a.



b.

Figura 2.17: a) Segnale AO senza illuminazione; b) Segnale AO con illuminazione

2.4.2 Seconda versione

A causa di un cortocircuito accidentale si sono bruciati alcuni componenti del circuito stabilizzatore a qualche giorno dalla gara per la selezione. Si è resa quindi necessaria qualche modifica alla scheda LED, data l'impossibilità di recuperare i componenti nell'immediato. Sono stati cambiati i LED con un modello più luminoso (Nichia White Led, mod.NSPW500GS-K, 33 cd di luminosità), andando a potenziare quindi il fascio luminoso ma *bypassando* la parte di circuito per la stabilizzazione. Ne è conseguito un semplice circuito a più rami paralleli di resistenze e diodi in serie.

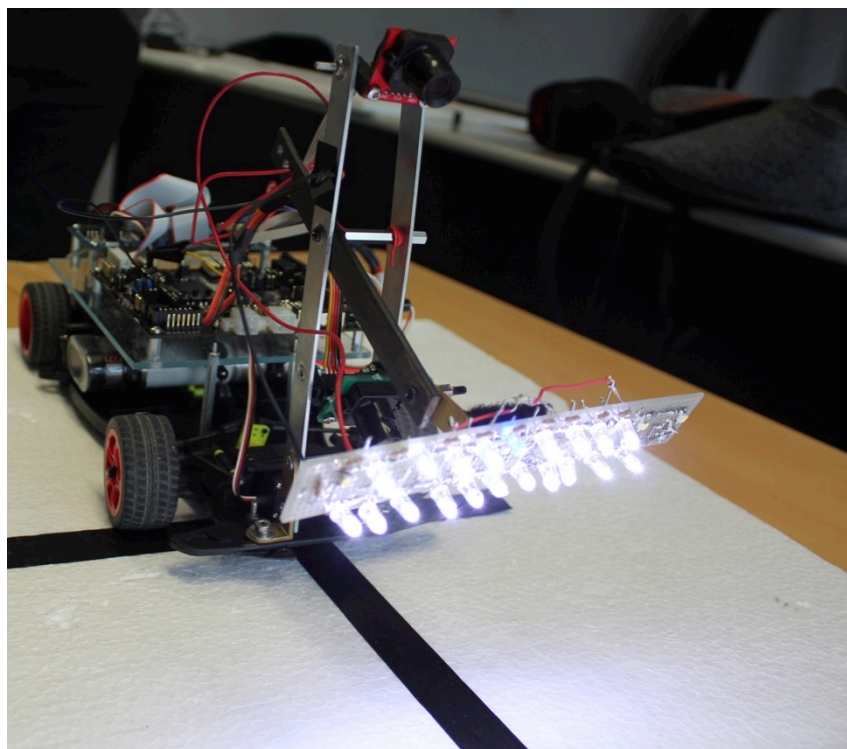


Figura 2.18: Scheda a LED in funzione

2.5 Analisi del codice relativo alle tecniche di acquisizione della linea

Analizziamo ora i due diversi approcci utilizzati per il riconoscimento della linea. Il primo approccio è molto macchinoso, meno semplice da comprendere, ma concettualmente più facile. Il secondo, quello che utilizza un metodo derivativo discreto, è più rigoroso ma maggiormente comprensibile.

2.5.1 Tecnica della media

Come analizzeremo nel codice riportato di seguito, questo metodo va a modificare delle variabili globali, in particolare la variabile *posizione_centro_linea_camera_alta*, che restituisce un numero tra 26 e 102 compresi, corrispondente al pixel che punta proprio al centro della linea. Sono stati esclusi i primi e gli ultimi 25 pixel nella parte di codice riguardante la ricerca della linea poichè non possono restituire risultati affidabili, in quanto il segnale *Analog Output* è debole per quelle posizioni. I nomi delle variabili sono stati scritti qui senza abbreviazioni per chiarezza espositiva, ma ovviamente per una buona programmazione è bene utilizzare nomi più corti e sintetici.

Il codice sottostante riguarda la camera alta, ma il concetto rimane lo stesso per quanto riguarda la camera bassa.

```
void Visione_Camera_Alta(void) // Definizione del nome del metodo
{
    // inizializzazione delle variabili che utilizzeremo in seguito
    int max = Result_Camera_Alta[0];
    int min = Result_Camera_Alta[0];
    int threshold_Camera_Alta=0;
    int k1, k2, j1;
    stato_precedente_di_nonCiVedo = nonCiVedo; // vado ad archiviare lo stato
    precedente della visione, ovvero se la linea era stata trovata o no
    limite_destro_Camera_Alta=0;
    limite_sinistro_Camera_Alta=0;

    for(i=0; i<128; i++) // ciclo che setta i valori di massima luminosità e
    minima luminosità captati da quel ciclo di integrazione della camera
    {
        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); //viene settato il threshold
(variabile globale) tramite media aritmetica tra max e min

for(k1=63-11, k2=64-11; k1>25, k2<103-10; k1--, k2++) // ciclo for che
determina dove inizia e dove finisce la linea facendo partire dal centro dell'
array le 2 variabili k1 e k2, che lo scorreranno in versi opposti
{
    if(Result_Camera_Alta[k1]<threshold_Camera_Alta && Result_Camera_Alta[k2]<
threshold_Camera_Alta && (k1-k2)<larghezzaLineaAlto) // caso che entrambi siano
sotto al threshold già al primo ciclo (capita con il veicolo centrato sulla linea)
    {
        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; // goto usato per non ciclare
inutilmente
    }

    if(Result_Camera_Alta[k1]< threshold_Camera_Alta) // caso in cui venga
trovato per primo il limite destro della linea (quindi la linea è a sinistra
del centro della visione del veicolo)
    {
        limite_destro_Camera_Alta =k1;
        for(j1=k1; Result_Camera_Alta[j1]<threshold_Camera_Alta; j1--)
        {
            limite_sinistro_Camera_Alta =j1;
        }
        goto LABEL_Settaggio_Visione;
    }

    if(Result_Camera_Alta[k2]<threshold_Camera_Alta) // caso in cui venga trovato
per primo il limite sinistro della linea (quindi la linea è a destra del
centro della visione del veicolo)
    {
        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; // valori
arbitrari che vengono usati solo nel caso in cui non riesca ad entrare in
nessuno degli if, in modo da non andare a soddisfare le condizioni seguenti e
quindi non modificare il centro della linea visto al ciclo del main precedente
}

LABEL_Settaggio_Visione: if (limite_destro_Camera_Alta -
limite_sinistro_Camera_Alta < larghezza_linea_Camera_Alta) // dove
larghezza_linea_Camera_Alta è una variabile globale trovata sperimentalmente
indicante il numero di posizioni dell'array mediamente occupato dalla linea nera e
maggiorato di qualche valore per sicurezza
{
    posizione_centro_linea_camera_alta = (int)((limite_dx_sopra +
limite_sx_sopra)/2) +5; // il valore 5 è stato misurato sperimentalmente. È
utilizzato perchè per il centro della linea è letto dalla camera spostato

```

```

    rispetto alla reale posizione a causa della pendenza laterale dei supporti
    nonCiVedo=0; // variabile che viene settata a 0 perchè ho visto la linea
    contatore = contatore + 1; // contatore usato nel metodo "sterzata" per
    evitare che in incroci o curve strette la macchina segua "linee" che non
    c'entrano
}
// essendo posizione_centro_linea_camera_alta 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; // variabile che viene settata a 1 perchè non ho visto la linea
    contatore = 0; // riassetto il contatore quando smetto di vedere
}
}

```

2.5.2 Metodo derivativo

Il seguente metodo invece utilizza il concetto di derivata discreta e di pendenza della retta. Risente molto di più di eventuali disturbi, quindi il segnale salvato nell'array Result necessita di un filtraggio prima di poter essere analizzato. Essendo una funzione di tipo *integer* mi restituirà, una volta chiamata, il valore relativo al centro della linea nera calcolato. In ingresso riceve un puntatore ad un *buffer* "circolare" dei vari centri trovati (quelli relativi alla camera bassa in questo caso), cioè continuamente aggiornato in maniera *First In Last Out*.

```

void aggiornaArray(int nuovo_centro, int* result_Mobile, int *oldest)
{
    result_Mobile[*oldest] = nuovo_centro; // "nuovo_centro" è il centro della
    linea appena calcolato
    *oldest = (*oldest +1) % DIMENSIONE_BUFFER; // utilizzando il resto della
    divisione, dove DIMENSIONE_BUFFER è definita a priori (es.10), si ottiene
    l'archiviazione F.I.L.O dei dati
}

```

Segue il codice che restituisce il centro della linea col metodo derivativo:

```

int Visione_Camera_Bassa(int posizione, int* result_Mobile)
{
    double deltaY, deltaX, derivata, posizione_derivata_Positiva,
    posizione_derivata_Negativa;
    int centro;
    int media = 0;
    double derivata_massima_Positiva = 10; // valore iniziale derivata positiva
    double derivata_massima_Negativa = -10; // valore iniziale derivata negativa
    for(i=posizione; i<posizione+BANDA; i++) // BANDA è un valore definito
    globalmente indicante quanti pixel dell'array sono ritenuti significativi.
    Nel nostro caso sono 97. Sommato a 15 (il valore di posizione), otteniamo che
    il ciclo for va da i=15 a i=113
    {
        deltaY = Result[i+3]-Result[i]; // calcolo del deltaY su 4 valori per
    }
}

```

```

filtrare e bypassare eventuali valori dovuti a variazioni puntuali
deltaX = 4; // il deltaX corrisponde sempre a 4 posizioni dell'array
derivata = deltaY/deltaX; // derivata discreta
if(derivata > derivata_massima_Positiva)
{
    derivata_massima_Positiva = derivata;
    posizione_derivata_Positiva = i+1; // salvo la posizione dell'array
    corrispondente alla derivata massima positiva
}
else if(derivata < derivata_massima_Negativa)
{
    derivata_massima_Negativa = derivata;
    posizione_derivata_Negativa = i; // salvo la posizione dell'array
    corrispondente alla derivata massima negativa
}
media = media + Result_camera_sotto[i]; // accumulo nella variabile media
tutti i valori numerici salvati in Result_camera_sotto
}

media = media/BANDA; // faccio la media aritmetica dei valori presenti
nell'array Result_camera_sotto

centro = (int) (posizione_derivata_Negativa + posizione_derivata_Positiva)/2;
// calcolo del centro della linea

if(centro > 112 || centro <15 || Result[centro]>media ||
derivata_massima_Positiva==10 || derivata_massima_Negativa == -10)
// controllo nel caso in cui non venga trovato un valore accettabile del
centro
{
    centro = *result_Mobile; // non viene modificato il valore del centro letto
    al ciclo precedente
    nonCiVedo = nonCiVedo + 1;
    ciVedo_Camera_Bassa = 0;
}

nuovo_centro = centro; // aggiorno la variabile globale chiamata nel metodo
aggiornaArray mostrato sopra
return centro;
}

```

2.6 Disturbi nell'acquisizione e soluzioni adottate

L'acquisizione dei segnali non è sempre semplice da fare ed i risultati non sono sempre chiari e privi di non-idealità. I disturbi principali che influiscono sul segnale acquisito dai sensori di visione sono principalmente due: la variazione della luminosità ambientale e le interferenze elettromagnetiche provocate dai motori.

A causa di ombre non previste o illuminazione ambientale differente da quella in cui si sono regolate le lenti delle camere per una visione ottimale, può essere che il segnale del sensore di visione sia debole, poco chiaro o addirittura qualche ombra sia scambiata per la linea nera. La soluzione ritenuta migliore per questo problema è stata quella dell'aggiunta di illuminazione a LED, grazie alla quale, come detto precedentemente, anche il problema dei tunnel è stato risolto.

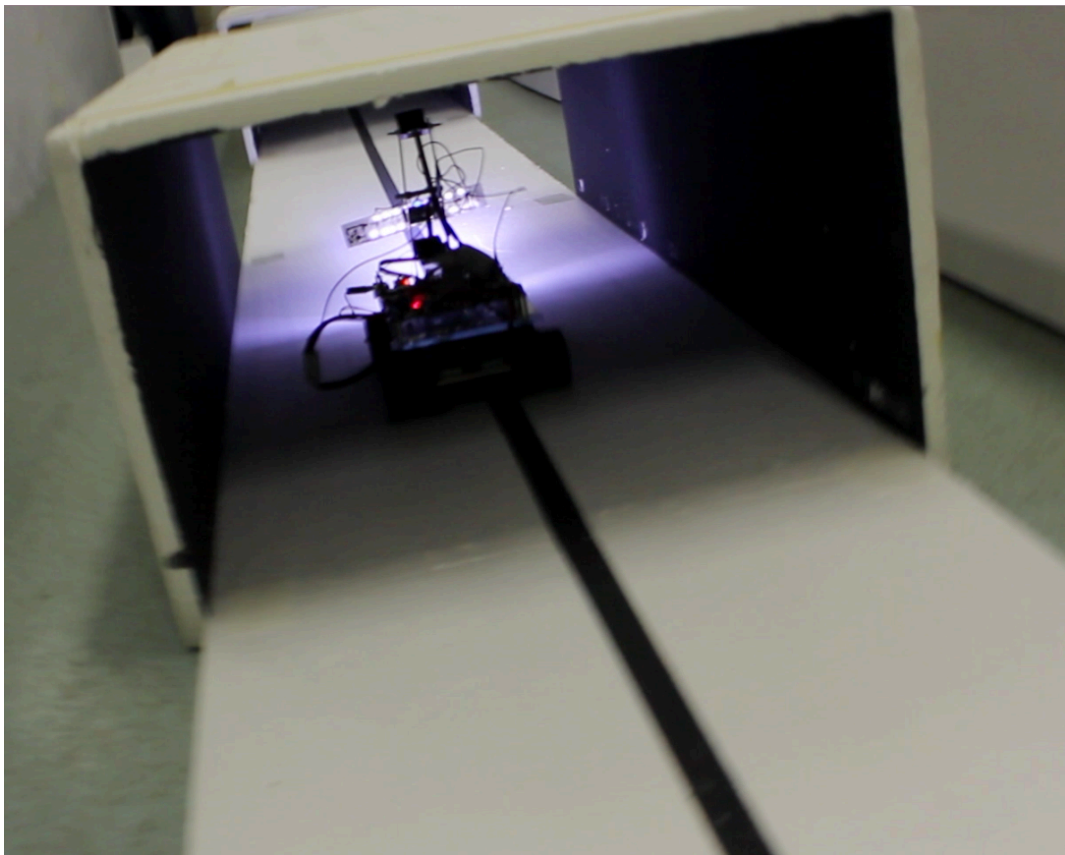


Figura 2.19: La smart car riesce ad acquisire un segnale sufficientemente chiaro da poter superare il tunnel

Un altro tipo di problema invece sorge all'aumentare della frequenza di PWM nei due motori usati per la propulsione. Il problema deriva dal fatto che sia i motori che la camera si collegano alla *Motor Drive Board*. All'aumentare quindi della PWM si ha che l'effetto induttivo dei motori aumenta a tal punto che l'interferenza elettromagnetica prodotta da essi disturba il segnale passante per i connettori delle camere. Questo anche perchè i cavetti utilizzati non sono ben schermati.

La soluzione a questo problema si è trovata saldando 3 capacitori da 100 nF ciascuno alla carcassa del motore secondo lo schema di figura 2.22. Il risultato è una attenuazione del rumore prodotto alle alte frequenze a causa del filtro passabasso che viene a formarsi con l'aggiunta dei capacitori, i quali vanno a formare un circuito RLC (resistivo induttivo capacitivo). Quest'ultimo è utile anche per ridurre l'effetto dello scintillio delle spazzole nel rotore.

Una ulteriore miglioria si è avuta con la sostituzione dei cavetti che connettono le camere alla *Motor Drive Board* presenti nel kit con dei cavetti meglio schermati.

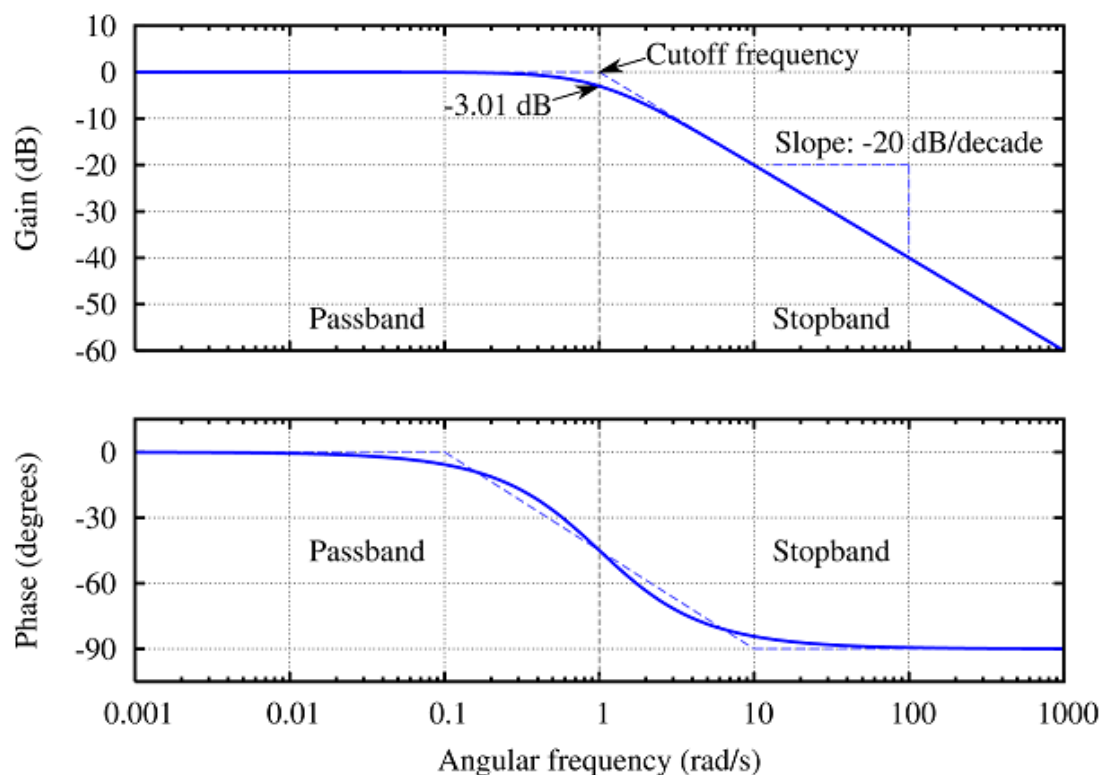


Figura 2.21: Diagrammi di Bode di un filtro passa basso ideale



Figura 2.21: Foto dei motori dopo la modifica hardware

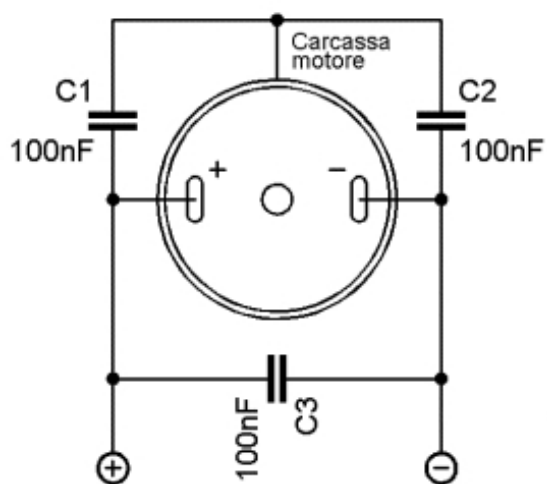


Figura 2.22: Schema per il collegamento dei capacitori

CAPITOLO 3

Elaborazione ed utilizzo dei segnali acquisiti

3.1 Conversione dei segnali da parte dell' ADC

Come accennato nel primo capitolo, l' ADC (*Analog-to-digital converter*) presente nella scheda del microcontrollore è un adc a 10 bit. Esso appartiene alla categoria dei DVM (*Digital Volt-Meters*).

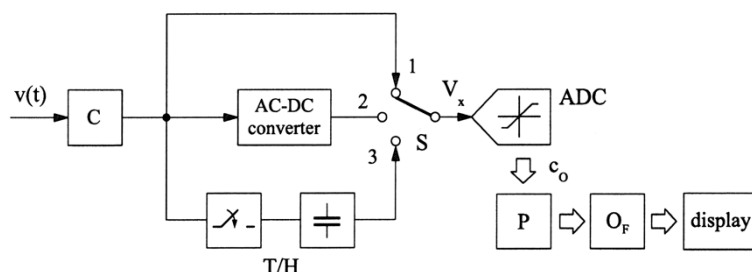


Figura 3.1: Schema generale di un DVM

Come è possibile osservare l' ADC è la parte centrale dello schema soprastante. In ingresso arriva un segnale in tensione, il quale viene fatto passare per un blocco C, detto di "condizionamento", dove viene amplificato (o attenuato) e pulito dal rumore. Seguono 3 vie possibili, la prima per i segnali DC (*direct current*), la seconda per segnali che richiedono una conversione AC-DC e la terza per tensioni in ingresso di tipo AC (*alternate current*), composta da un circuito di tipo *Track and Hold*.

La conversione del segnale analogico da parte dell' ADC avviene in 3 fasi principali. La prima è quella del "campionamento", ed in base ad una frequenza, detta appunto frequenza di campionamento (F_c), va a prendere un campione del segnale in ingresso ogni $T_c = \frac{1}{F_c}$ secondi. La seconda fase, detta "di tenuta" o *Sample ad Hold* consiste poi nel mantenere il valore campionato andando a caricare un condensatore al valore di tensione acquisito nella prima fase. Questo valore viene tradotto in un uscita digitale (un codice di B bit, 10 in questo caso) passando attraverso la caratteristica propria dell' ADC nella terza fase, detta "quantizzazione" delle ampiezze.

Il passo di quantizzazione Q è facilmente calcolabile:

$$R = 5 \text{ V}$$

$$B = 10 \text{ bit}$$

$$N = 2^B \text{ numero di livelli di quantizzazione}$$

$$Q = \frac{R}{N} = \frac{R}{2^B} = 4,88 \cdot 10^{-3} V$$

Seguono il blocco P di "*processing*" del segnale digitale, O_F di "*output formatting*" ed infine troviamo un display, nel caso di uno strumento il cui scopo sia una lettura ottica del risultato. Nel nostro caso non sono presenti gli ultimi due blocchi.

L' ADC integrata nel microcontrollore MPC5604B è di tipo SAR (*Successive Approximation Register*), di tipo differenziale, e le sue caratteristiche sono:

- Risoluzione a 10 bit
- Tempo di conversione supportato: fino a 650 ns
- Fino a 36 canali di input, espandibili a 64 con l'aggiunta di multiplexer esterni
- 16 canali di precisione
- 16 canali standard
- 4 canali standard per il collegamento di multiplexer esterni
- Multiplexer interno per acquisizioni multiple
- Generazione interna di un segnale di tipo address decoder per coordinare gli eventuali multiplexer esterni
- differenti registri per i tempi di campionamento e conversione CTR[0:2] per i canali interni di precisione, i canali standard e i canali esterni
- 64 registri dedicati per i risultati della conversione di ogni canale
- Precampionamento
- Funzionamento in modalità one-shot, scan, injection e triggered injected
- Configurazione indipendente dei parametri di ogni canale

La modalità **Normal Conversion** è la più utilizzata, ed essa accetta come ingresso un canale per l'acquisizione, il quale viene abilitato impostando un "1" nel campo corrispondente del NCMR (*Normal Conversion Mark Register*). La conversione avviene solo se il bit NSTART è settato a "1" (abilitazione via software) o se, oltre al bit NSTART alto, si riceve un segnale che supera quello di *trigger* (abilitazione via *trigger*).

Le altre modalità, come *One shot* e *Scan*, vengono abilitate programmando il MCR[MODE] e selezionando i vari MODE. La prima fa una conversione una sola volta al termine di una acquisizione e salva il valore numerico nel corrispondente registro; la seconda modalità invece, *Scan*, effettua una conversione continua al termine di ogni acquisizione.

Il funzionamento di un voltmetro di tipo SAR può essere esemplificato come segue. La conversione digitale del segnale in tensione avviene in un numero finito di passi pari al numero di bit B dell'ADC.

Il segnale in ingresso incognito V_x entra in un comparatore, dove viene sottratto ad una tensione di riferimento V_R . Il risultato, saturato dal comparatore, entra in una unità logica comandata da un *clock*, dove viene prodotto un codice binario a B cifre, che si aggiorna ad ogni nuovo passo. Oltre a costituire l'uscita numerica della conversione, questo codice è anche l'ingresso di un DAC (*Digital to Analog converter*), che tramite una formula produce una tensione di riferimento V_R' , la quale va a caricare un condensatore, la cui tensione andrà a costituire il riferimento V_R . I valori di V_R' , e quindi di V_R , variano ad ogni nuovo passo, approssimando sempre di più la tensione V_x .

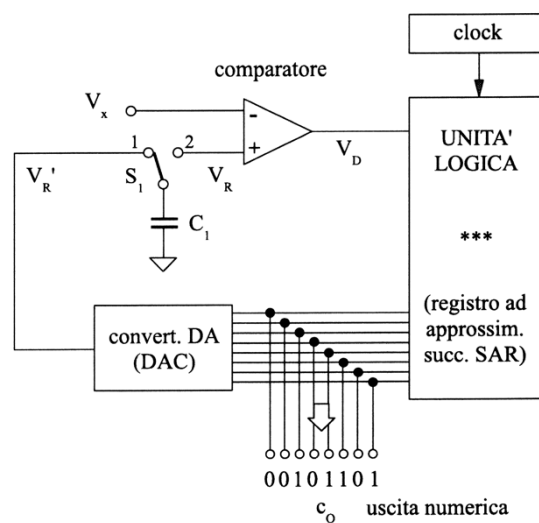


Figura 3.2: Schema di funzionamento di un voltmetro SAR

La formula per il calcolo di V_R' è la seguente:

$$V_R' = [b_{B-1}2^{B-1} + b_{B-2}2^{B-2} + \dots + b_12^1 + b_0] \cdot \frac{R}{2^B}$$

Dove b_i è l' i -esimo bit dell'uscita digitale.

Al completamento dei B passi possiamo leggere in uscita il codice C_0 corrispondente alla tensione convertita.

3.2 Stima della velocità tramite *feedback current*

Una strada percorsa con scarso successo a causa di svariati problemi è stata quella del calcolo della velocità in *real-time* del veicolo per un controllo in catena chiusa degli attuatori. Il maggior problema nell'usare la stima della velocità è stato quello di non poter avere un valore numerico attendibile in fasi di accelerazione e decelerazione del veicolo. Illustriamo comunque i procedimenti ed i calcoli effettuati.

Per stimare la velocità in base alla misura della corrente circolante nei motori è necessario prima ottenere la velocità angolare di rotazione del rotore:

$$\Omega(t) = \frac{U_a(t) + R \cdot i(t)}{k_e \Phi}$$

Dove:

- $\Omega(t)$ è la velocità angolare di rotazione del rotore [*rad/sec*]
- $U_a(t)$ è la tensione impressa di armatura [V]
- R è la resistenza di armatura [Ω]
- $i(t)$ è la corrente di armatura [A]
- $k_e \Phi$ è la costante di coppia [*V · sec/rad*]

La stima della velocità del veicolo è calcolata tramite la seguente formula:

$$v(t) = \Omega(t) \cdot k \cdot r$$

Dove:

- $v(t)$ è la velocità stimata del veicolo [m/s]
- k è il coefficiente di sensibilità (rapporto tra la velocità angolare del motore a vuoto e quella della ruota)
- r è il raggio della ruota [m]

Il raggio della ruota è misura 0,025 metri.

Il coefficiente di sensibilità vale 0,133929.

La resistenza di armatura è stata misurata sperimentalmente a causa della scarsità di dati relativi ai motori. Una prima misurazione tramite modalità "a quattro fili" non ha dato risultati soddisfacenti in quanto le correnti che venivano fatte scorrere dal multimetro nel motore non erano adatte (troppo basse) per la misura che si cercava di fare. Si è invece riusciti ad ottenere dati migliori andando a iniettare una corrente per mezzo di un alimentatore misurando la caduta di tensione ai capi degli

avvolgimenti.

I valori di resistenza ottenuti tramite media di una dozzina di misurazioni sono:

- $R_{sx} = 2,677 \, \Omega$
- $R_{dx} = 2,42 \, \Omega$

Per quanto riguarda il calcolo sperimentale della costante di coppia, questo è stato fatto tramite l'utilizzo di un trapano collegato all'albero del motore per imprimere un numero di giri costante. La conta dei giri (in *rpm*) è stata data da un misuratore tachimetrico digitale. Si è misurata quindi la forza controelettromotrice del motore grazie ad un voltmetro digitale collegato alle fasi e, dopo numerose prove a diversi *rpm*, si sono calcolate le due costanti:

- $k_e \Phi_{sx} = 0,032 \, \text{V} \cdot \text{s/rad}$
- $k_e \Phi_{dx} = 0,031 \, \text{V} \cdot \text{s/rad}$

Per verificare i risultati ottenuti come stima della velocità con questi dati, si è proceduto con ulteriori prove sperimentali. Questo è stato fatto per poter accoppiare valori di PWM a velocità corrispondenti. Si è preso un lungo rettilineo del tracciato nel quale far correre il veicolo, ed in due diversi punti sono stati posizionati dei *sonar* usati come fotocellule. Dopo il periodo di accelerazione il veicolo, a velocità costante, attraversava il primo *sonar*, percorreva una distanza fissa fino al secondo sonar e veniva fermato solo dopo aver attraversato quest'ultimo. Misurando il tempo di percorrenza tra i due sensori e sapendo la distanza che li separa è stato così possibile calcolare la velocità effettiva del veicolo corrispondente a diversi valori di PWM.

I valori ottenuti sono stati confrontati con quelli calcolati via software e sono state notate discrepanze non trascurabili. Analizzando più a fondo i dati si è trovata una sorta di relazione lineare, rappresentata nel seguente grafico, tra le velocità misurate (reali) e quelle calcolate (stimate via software), rispettivamente in ordinata e in ascissa.

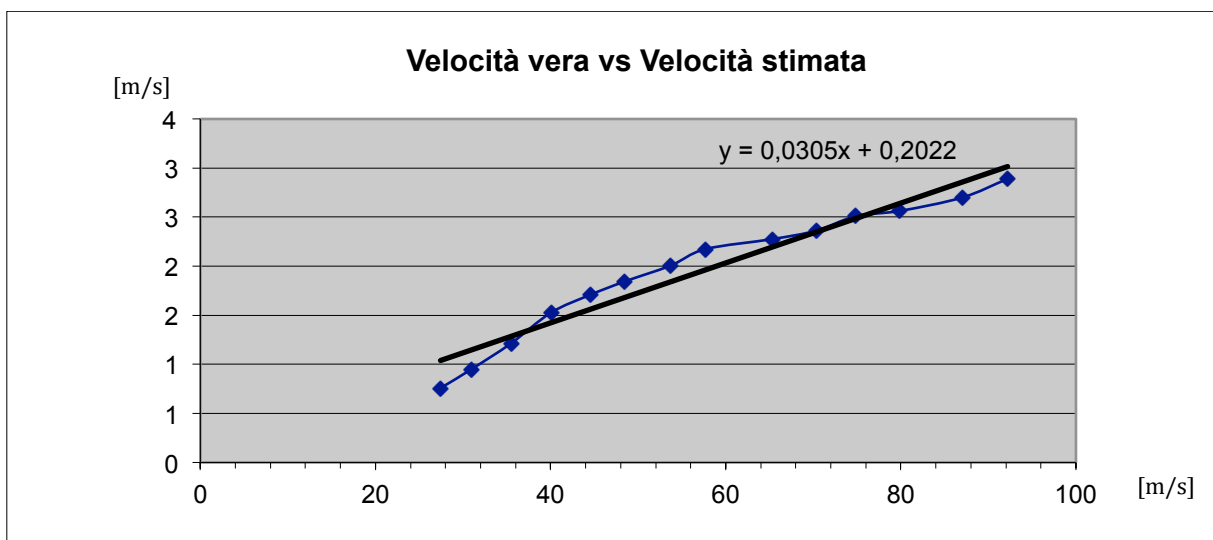


Figura 3.3: Interpolazione lineare del rapporto tra le velocità vere e quelle stimate leggendo la corrente del motore sinistro

La stima è errata anche perchè il pin FB di output dell' *H-Bridge* nel quale si va a leggere la corrente circolante all'interno dei rami porta ad una lettura di un riferimento di corrente pari allo 0,24% della reale corrente sul carico.

L'impossibilità di conoscere la velocità in accelerazione o in decelerazione dovuta alle variazioni repentine delle correnti che rendono inutilizzabili le formule viste sopra insieme ai dati ottenuti non del tutto attendibili hanno decretato l'eliminazione dall'algoritmo di controllo del veicolo di una retroazione data dalla lettura della velocità. Probabilmente l'utilizzo di un encoder montato sull'asse delle ruote avrebbe sicuramente prodotto risultati migliori, a fronte di un lavoro più impegnativo di montaggio ed interfacciamento.

Segue il codice utilizzato nella funzione che restituisce il valore della velocità stimato e corretto:

```
double correnteVelocitaSinistra(double current, int relativeVoltage) // in ingresso
vi sono il valore di corrente letto ed il valore di PWM assegnato al motore
{
    double Ua = relativeVoltage*voltageCoefficient; // voltageCoefficient serve
    per la conversione tra valori percentuali eMIOS (0..1000) in una grandezza di
    tensione in Volt.
    double Ia = current;
    double linearSpeedSx=0;
    double linearSpeedSx1=0;
    angularSpeedSx = (Ua + RaSx*Ia)/KeSx;
    linearSpeedSx = angularSpeedSx*sensibilityCoeff*rearWheelRadius; // calcolo
    teorico della velocità
    linearSpeedSx1 = linearSpeedSx*0.0305+0.2022; // correzione del valore
    return linearSpeedSx1;
}
```

3.3 Controllori PID

Il tipo di controllore utilizzato per regolare il servomotore ed il riferimento di tensione PWM da dare ai motori per la propulsione è di tipo PID, ovvero *proportional, integrative, derivative*. La funzione di trasferimento di un controllore PID è:

$$G_C(s) = K_p \left(1 + T_d s + \frac{1}{T_i s} \right)$$

La costante K_p è detta sensibilità proporzionale, T_d è la costante di tempo dell'azione derivativa e T_i la costante di tempo dell'azione integrale.

Il regolatore PID integra tre diverse azioni e consente, oltre ad una buona prontezza di risposta, anche un errore statico nullo.

$$K_d = K_p T_d$$

$$K_i = \frac{K_p}{T_i}$$

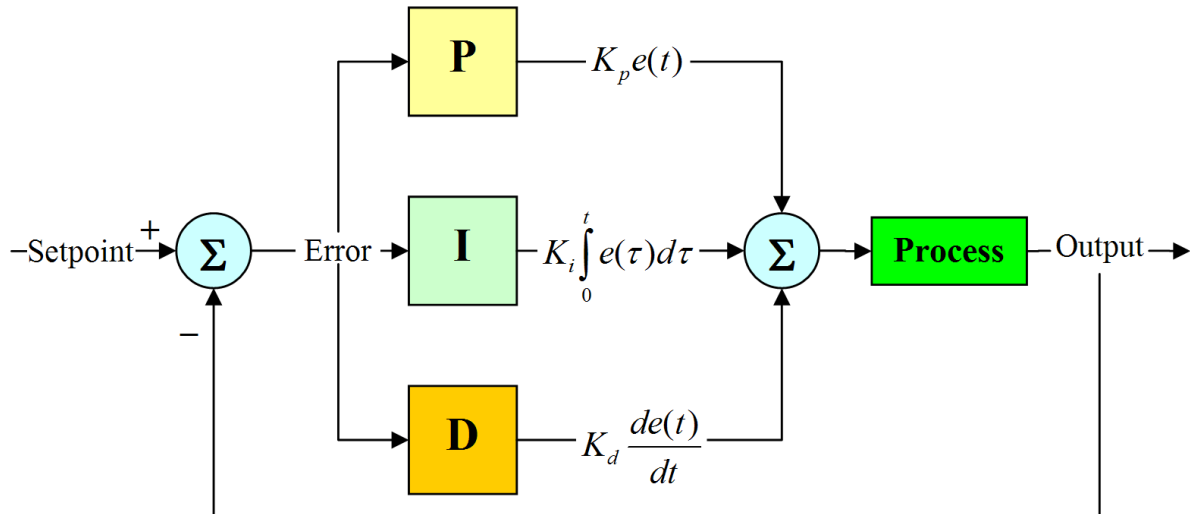


Figura 3.4: Schema a blocchi di un controllore PID in configurazione standard

La taratura del controllore è stata fatta tramite prove sperimentali e l'attuazione del **metodo di Ziegler e Nichols**. Per procedere a questa taratura è necessario determinare la "banda di pendolazione": questa è definita come il valore della banda proporzionale $1/K_0$ che, in assenza delle azioni integrale e derivativa, porta il sistema di regolazione in condizioni di stabilità limite, cioè in oscillazione permanente. È così possibile ottenere il periodo di oscillazione T_0 e calcolare le costanti grazie alla

seguente tabella:

	K_p	T_i	T_d
P	$0,5 K_0$		
PI	$0,45 K_0$	$T_0 / 1,2$	
PID	$0,6 K_0$	$0,5 T_0$	$T_0 / 8$

Tabella 3.1: Taratura tramite metodo di Ziegler e Nichols

Per il calcolo del periodo di oscillazione T_0 una volta fatta oscillare la smart car in rettilineo dopo svariate prove, si è usato un software di analisi audio per analizzare una registrazione del moto oscillatorio del servomotore del veicolo. I valori ottenuti sono stati ritenuti attendibili ad eccezione di quello relativo all'azione integrativa, che è stato abbassato e tarato con prove su pista. I valori ottenuti sono i seguenti:

- $K_p = 0,3$
- $K_d = 0,01$
- $K_i = 0,0095$

Questi valori consentono una buona risposta del veicolo alle diverse tipologie di curva.

Volendo migliorare la stabilità nei tratti di rettilineo, anche alla massima velocità raggiungibile dal veicolo, si è scelto di escludere l'azione integrale nei tratti in cui la camera alta vedeva la linea nera per più di un numero predefinito di cicli (i.e. 20). Questo fatto implica che si è in un tratto sicuramente rettilineo e quindi è necessario un controllo meno "forte" di quello richiesto per affrontare una curva, poichè sono necessarie solo piccole regolazioni per mantenere in traiettoria rettilinea il veicolo.

Mediante un controllo di tipo solamente PD si è riusciti quindi ad ottenere ottima stabilità nei rettilinei anche a tensione impressa sui motori massima.

La gestione dell'azione integrale è la più delicata, e necessita di alcune osservazioni. Innanzitutto è necessario imporre un limite negativo ed un limite positivo ad essa, in modo da evitare un *integrator windup*, cioè un "caricamento dell'integratore". Basta poi disconnettere l'azione integrale quando viene raggiunto uno dei due livelli di saturazione, massimo o minimo, e l'ingresso risulta rispettivamente positivo o negativo. Un ulteriore accorgimento è nel distacco dell'integratore anche quando si passa al controllo PD (i.e. in uscita da una curva seguita da un rettilineo), questo per evitare che l'integratore continui a caricarsi anche se non è utilizzato e quando si passa nuovamente al controllo PID non faccia sterzare bruscamente il veicolo, facendo perdere il controllo della traiettoria.

Un semplice esempio di realizzazione di un PID in codice C utilizzato come controllore per il veicolo nel quale è presente un *Anti Windup* per limitare la saturazione dell'azione integrativa.

```
double PID_Camera_Bassa(int actualPosition)
{
    double outputPid=0;
    double derivative=0;
    double errore;
    errore = carPosition - actualPosition; // dove carPosition è una variabile
    globale valida 63, cioè il centro dell'array.
    integralBasso = integralBasso + errore*dt; // il dt è il tempo di ciclo del
    main
    derivative = (errore - pre_error)/dt;
    outputPid = Kp*errore + Ki*integralBasso + Kd*derivative;

    if(integralBasso > MAX) // Filtro di saturazione
    {
        outputPid = MAX;
        integralBasso=0;
    }
    else if(integralBasso < MIN)
    {
        outputPid = MIN;
        integralBasso=0;
    }

    pre_error = errore; // Aggiornamento errore
    return outputPid;
}
```

La realizzazione del controllore PD per i rettilinei invece è la seguente.

```
double PD_Rettilineo(int actualPosition)
{
    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;
}
```


Conclusioni

Questo elaborato nasce per approfondire le questioni riguardanti l'acquisizione di segnali dal mondo analogico e la loro digitalizzazione nell'ambito del progetto *Freescale Cup*, ma la presentazione del lavoro svolto può essere uno strumento di supporto anche a chi si trova ad affrontare tematiche simili.

Ci si augura che la trattazione abbia soddisfatto le aspettative del lettore e possa essere stata d'aiuto per risolvere dubbi soprattutto a studenti che si rivolgono a questa sfida, magari per la prima volta.

Il concetto di maggior importanza che si vuole trasmettere è la difficile corrispondenza tra idealità di un concetto e realtà dei fatti, in quanto ci si trova di fronte ad una sfida di non poco conto, dove la progettazione teorica si deve scontrare con l'applicazione pratica. Si è infatti cercato di mostrare tutti problemi sorti nel rendere il veicolo in grado di riconoscere la pista di gara e i diversi elementi che la compongono, cercando al contempo di fornire delle possibili soluzioni.

Si sono inoltre volutamente esclusi ulteriori approfondimenti sulla parte di controllo del veicolo, preferendo un semplice accenno di carattere generale ad essi, poichè non erano materiale fondamentale di discussione della tesi.

BIBLIOGRAFIA E SITOGRAFIA

- M. Bertocco and A. Sona, *Introduzione alle misure elettroniche*, Lulu, 2nd ed., 2010
- Programmazione in C, Kim N. King , Apogeo
- Controlli Automatici – di Giovanni Marro – Ed. Zanichelli - 5° Edizione
- R. Oboe, *Dispensa dal corso di Controlli Automatici*, 2013
- M. Zigliotto, *Dispense dal corso di Fondamenti di macchine ed azionamenti elettrici*, 2013
- Freescale Semiconductor; Inc., *MPC5604B/C Microcontroller – Reference Manual, MPC5604BCRM*, Rev. 8.1, 05/2012
- Freescale Semiconductor; Inc., *Motor Drive Rev. A*
- TAOS inc., *TSL1401CL, 128 x 1 Linear Sensor Array With Hold, TAOS136*, 7/2011
- Steve Mihalik, *Getting Started with MPC560xB*, Freescale Cup, January 24, 2011
- Freescale Semiconductor, *MC33931 Data Sheet*, Rev. 3.0, June 2012
- Freescale Semiconductor, *MPC5604B Application Note: Using Parallax TSL1401-DB Linescan Camera Module for line detection*, Rev. 0, January 2011
- Freescale Semiconductor, *MPC5604B/C Microcontroller Reference Manual*, Rev. 8.1, May 2012
- Freescale University Programs
(<https://community.freescale.com/community/uvp>)

- Futaba (<http://www.futabarc.com>)
- Linescan camera use (<https://community.freescale.com/docs/DOC-1030>)
- Linear sensor array (<http://www.ams.com/eng/LinearSensorArray>)
- Line Scan Camera Board (<https://community.freescale.com/docs/DOC-1058/version/2>)