Giuseppe Romeo (giurom88)

ROBOTTINO C88

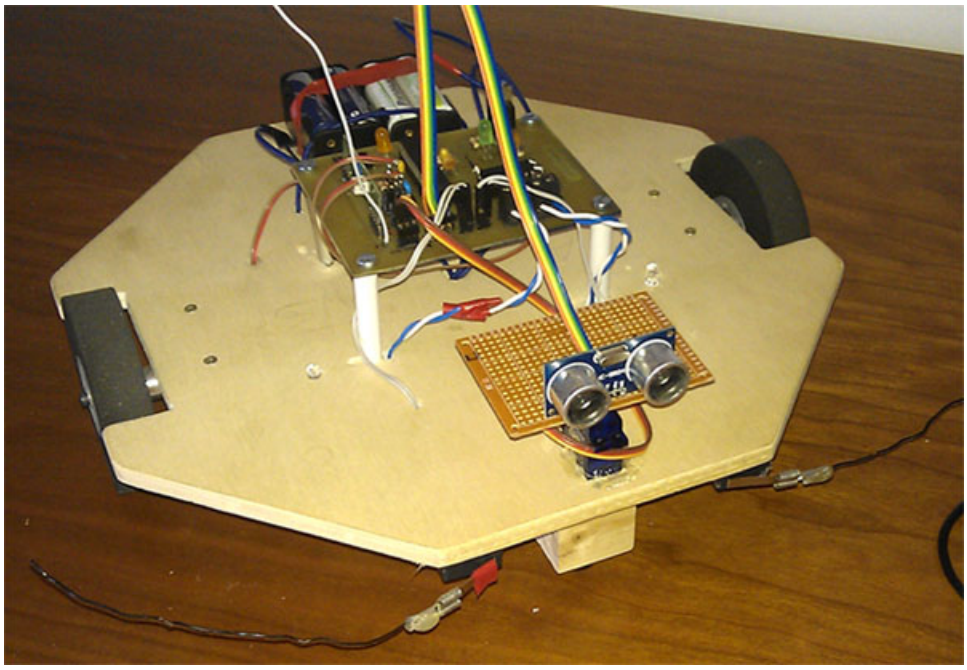
31 May 2013

Presentazione

Salve a tutti!
Già da bambino ero fortemente affascinato dall'elettronica,stravedendo soprattutto per le macchinine radio-comandate (che puntualmente demolivo per vedere com'erano fatte internamente), per i robot e per gli automatismi in generale.

L'ammirazione che avevo per il mondo dell'elettronica e della tecnologia in generale, mi hanno spinto ad intraprendere un percorso di studio universitario nel mondo dell'ingegneria elettronica. C'è da dire che rimpiango un po' il fatto di non aver frequentato prima un istituto industriale elettronico che sicuramente mi avrebbe dato già in principio delle basi che purtroppo prima non avevo. Ma alla fine, grazie soprattutto a libri, riviste ed internet (con siti e forum come electroyou), sono riuscito lo stesso ad acquisire delle buone conoscenze.

Finalmente, in procinto di concludere il mio percorso di studi, sfruttando pure un corso universitario che prevedeva la realizzazione di un progetto basato sull'utilizzo di un **microcontrollore**, sono riuscito a realizzare il mio tanto desiderato Robot!



Robot C88

Siccome per la realizzazione del robot mi sono avvalso molto delle informazioni della rete, sia come una forma di ringraziamento che come forma di aiuto per altri ragazzi, ho deciso di condividere in maniera libera questo progetto con la community di Electroyou.

Il Robot realizzato, basato su una struttura a 2 ruote motrici più una terza ruota di sostegno, è in grado di lavorare in 2 modalità:

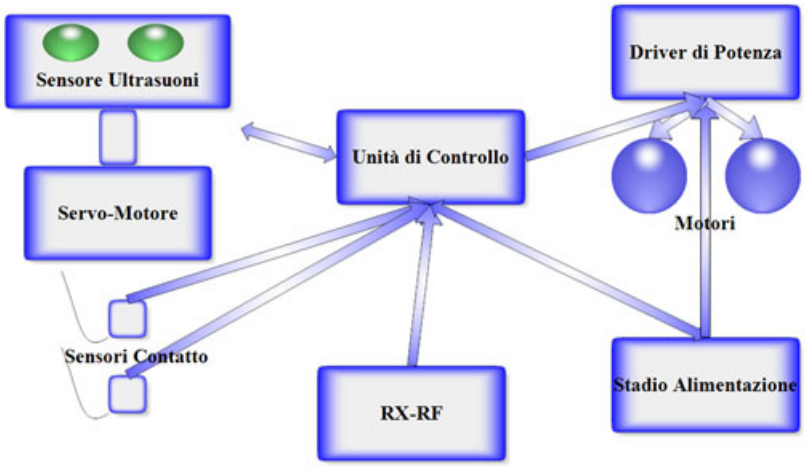
1. **Modalità autonoma**: Il robot si gestisce da solo, scansando gli ostacoli sfruttando un modulo ad ultrasuoni e 2 interruttori di fine-corsa;
2. **Modalità pilotata**: Il robot viene gestito dall'utente mediante un radiocomando.

Il progettino che avevo realizzato in precedenza per l'università prevedeva l'impiego di un ATmega8535 programmato in assembly (in modo da essere forzati a studiare per bene l'architettura interna del micro). Non avendo più disponibile l'ATmega8535, ho deciso di riaggiornare il progetto ricorrendo ad un **ATmega88**, sistemando anche su PCB auto-costruito tutta la centralina progettata (il progetto precedente era invece su millefori).
Ho modificato anche il radio-comando: in precedenza l'avevo realizzato su breadboard sfruttando un micro ATmega8515 (era l'unico che avevo in quel momento). Siccome quel micro era totalmente sprecato per la funzione che doveva svolgere, ho deciso di creare il nuovo radiocomando ricorrendo ad un ***PIC16F628A*** che avevo disponibile (con la scusa ho quindi imparato anche qualcosina sui PIC).

Specifiche del progetto:

- Velocità di movimento di almeno 0,1m/s;
- Capacità di rilevamento degli ostacoli ad una distanza ≤ 20cm;
- In presenza di ostacoli, vi deve essere la capacità di scansionare l'ambiente alla ricerca della direzione più libera;
- Possibilità di un controllo remoto mediante un link radio a 433MHz

Blocchi del progetto:



Schema a blocchi

Gli elementi più importanti del sistema realizzato sono:

- 2 motori per la locomozione del robot;
- 1 driver di potenza per il controllo dei motori (verso e velocità di rotazione);
- 1 modulo ad ultrasuoni (un emettitore ed un sensore) per il controllo della presenza di ostacoli;
- 1 servomotore da modellismo per ruotare il modulo ad ultrasuoni per uno scanning ambientale degli ostacoli;
- 2 interruttori stile “finecorsa” per notare la presenza di ostacoli in zone non coperte dal modulo ad ultrasuoni;
- 1 modulo radio ricevente a 433MHz per la ricezione dei comandi da remoto;
- 1 microcontrollore per gestire e coordinare il funzionamento di tutto il sistema;
- 1 pacco batterie come fonte di energia per tutto il sistema;
- 1 stadio di regolazione per alimentare l’elettronica a “bassa potenza”.

Adesso illustrerò brevemente gli elementi utilizzati.

Motori

I motori da scegliere devono avere una coppia motrice tale da muovere la massa del robot (contenuta in 3Kg massimi) ed un numero di giri al minuto tale da far muovere il robot alla velocità desiderata (almeno 0,1m/s).

Per il lavoro effettuato forse è un po' esagerato ricorrere a calcoli matematici per il dimensionamento dei motori del robot (anche perché si deve muovere in un ambiente casalingo ed il peso e la velocità desiderata sono tali da permetterel'impiego di un motorino DC qualsiasi), però, visto che in svariati forum sono molte le richieste di questi calcoli, li metterò per completezza (anche per vedere se i ragionamenti che ho fatto sono corretti).

Fissando un'accelerazione alla partenza di $0,1\frac{m}{s^2}$, la coppia motrice necessaria al moto sarà:

$$C = m \cdot a \cdot r$$

Con *m* = massa del robot; *a* = accelerazione del robot; *r* = raggio delle ruote

Siccome le ruote a disposizione presentano un raggio di $0.035\,\mathrm{m}$:

$$C = 3\mathrm{kg} \cdot 0.1\frac{m}{s^2} \cdot 0.035\,\mathrm{m} = 0.0105\,\mathrm{N\ m}$$

Per un dimensionamento più accurato, soprattutto nel caso in cui il robot abbia un peso rilevante e debba muoversi all'esterno, sarebbe necessario tenere conto pure dell'azione dissipativa dovuta all'attrito volvente. Questo tipo di attrito si manifesta quando un corpo sferico rotola senza strisciare su di una superficie. La resistenza opposta al moto rotativo è dovuta al fatto che, in corrispondenza della zona di contatto, i corpi si deformano in modo non perfettamente elastico. Ciò ha ripercussioni sulla distribuzione delle pressioni sulla ruota che non sono più simmetriche rispetto al punto di contatto: si vengono quindi ad instaurare delle forze di pressione di diversa intensità che tendono a riequilibrare il peso a discapito del rallentamento del moto.

Lo squilibrio tra le 2 forze di reazione vincolare viene espresso mediante una risultante *N* perpendicolare al suolo e opposta alla forza peso. In caso di corpo non deformabile questa forza dovrebbe passare per il punto di contatto.Invece, per via degli squilibri di pressione, questa forza si dispone anteriormente rispetto al centro della ruota di una distanza indicata con "*b*" che esprime il **parametro di attrito volvente**.

Per mantenere uniforme il moto rotativo è necessario quindi applicare una Forza uguale e opposta a quella dell'attrito pari a:

$$F_{att} = \mu_v \cdot F_p$$

con μ_v = b/r detto **coefficiente dell'attrito volvente**: diminuisce all'aumentare del raggio della ruota.

Non trovando alcun parametro relativo al contatto tra neopropilene e marmo (tipico dell'ambiente casalingo), scelgo (esagerando) come parametro di attrito volvente quello tra pneumatico e asfalto, pari a 0.005.

La coppia motrice necessaria a muovere il robot dovrà quindi essere maggiore o uguale a:

$C = C_{robot} + C_{att} = 0.0105Nm + 0.174Nm = 0.185Nm \leftrightarrow 1.886\,\mathrm{kg\ cm}$

Essendoci 2 motori a muovere il robot, il peso si ripartirà abbastanza equamente fra di essi, quindi ogni motore dovrà avere una coppia motrice minima di $Cr/2 = 0.943\,\mathrm{kg\ cm}$.

Volendo far muovere il robot ad una velocità media di 0.1m/s (6m/min), ogni motore dovrà avere un numero di giri pari a:

$$RPM = \frac{metri/minuto}{2r} = \frac{6\mathrm{m/min}}{0,22} \approx 27 giri/minuto$$

I motorini DC scelti sono i *Sanyo Solarbotics* che rispettano le specifiche imposte.Inoltre ingombrano pochissimo, sono leggerissimi, includono la riduzione meccanica e richiedono correnti e tensioni di lavoro inferiori rispetto ad altri modelli. Hanno anche un costo abbastanza contenuto.

Caratteristiche:	
Rapporto Riduzione:	100:1
Velocita a Vuoto:	85 RPM (3V) - 169 RPM (6V)
Corrente a Vuoto (mA):	18 (3V) - 32 (6V)
Corrente in Stallo (mA):	290 (3V) - 456 (6V)
Coppia Max (gr/cm):	1120 (3V) - 1960 (6V)
Dimensioni (mm):	12 x 10 x L30
Diametro Asse (mm):	3.00
Peso (gr):	10.00

Motori Sanyo Solarbotics

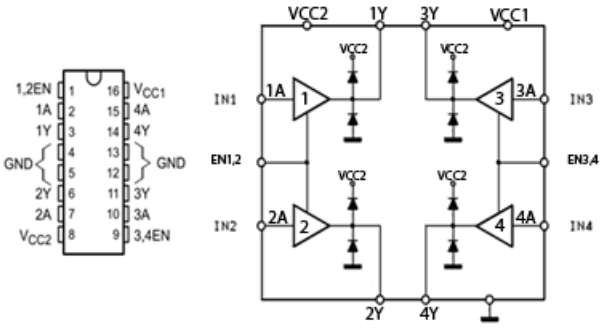
Driver di Potenza

Siccome il microcontrollore non è in grado di erogare una potenza adeguata al controllo dei motori (ogni pin digitale fornisce una tensione pari a VDD_{micro} (5V nel nostro caso) con corrente max erogabile di 25mA), è necessario ricorrere ad una qualche interfaccia adibita a tale compito. In particolare, per il controllo della potenza dei motori e del loro verso di rotazione è stato sfruttato l'integrato **L293D**. Esso è dotato di 4 buffer di potenza (degli half bridge) gestiti a coppie da 2 segnali di *ENABLE/INHIBIT*.

Tramite questo chip è possibile fornire una corrente media di 600mA ad ogni motore, con picchi massimi consentiti da 1.2 A.

Da notare che vi sono poi due ingressi di alimentazione: uno a "bassa potenza" (VCC1 - pin16) per la gestione della logica, ed uno ad "alta potenza" (VCC2 - pin8) per prelevare l'energia da fornire ai motori.

Per il controllo dei motori è stata usata la tecnica *Sign-Magnitude PWM* che prevede di controllare la potenza fornita ai motori mediante 2 segnali PWM applicati ai pin di ENABLE dell'integrato, mentre prevede la gestione del verso di rotazione mediante i 4 pin di input.



In1	In2	In3	In4	Verso
0	1	0	1	Avanti
1	0	1	0	Indietro
0	1	1	0	Destra
1	0	0	1	Sinistra
1	1	1	1	Stop

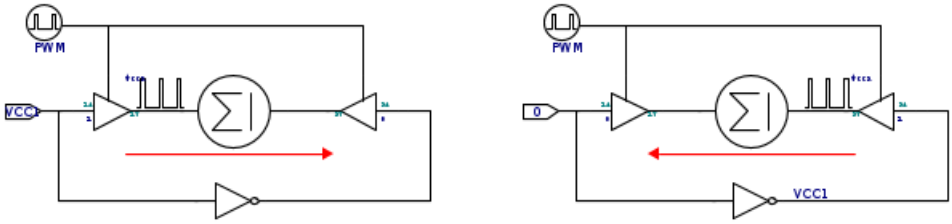
L293

Riferendoci al *buffer 1*, quando IN1 (*1A*) è alto, l'uscita *1Y* dipenderà dall'ENABLE:

- se l'*EN1,2* è alto (VCC1), l'uscita del buffer sarà sempre alta (VCC2)
- se l'*EN1,2* è basso (0V), l'uscita del buffer sarà bassa (0V).

Di conseguenza, un PWM a “bassa potenza” applicato sul pin di ENABLE verrà convertito dal driver L293D in un segnale PWM ad “alta potenza” contenuto nel range [0;VCC2]V.

Sfruttando 2 buffer per il controllo di un motore, si riesce ad ottenere un full-bridge completo con il quale è possibile gestire il verso di rotazione del robot.



Collegando per esempio il motore tra i 2 buffer gestiti dall’ *EN1,2*, se *1A* è alto mentre *2A* è basso, la corrente attraverso il motore scorrerà dal buffer_1 al buffer_2, facendo ruotare il motore in un verso.

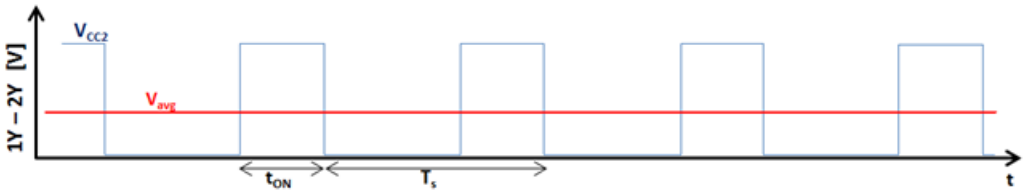
Invertendo invece il valore degli ingressi, la corrente scorrerà dal buffer 2 al buffer 1, facendo ruotare il motore nel verso opposto. Tenendo ambi gli ingressi alti o bassi, la d.d.p. ai capi del motore sarà nulla, e quindi resterà fermo.

Per la gestione della velocità invece si ricorre al segnale PWM sulla linea di *ENABLE* (che attiva/disattiva l’uscita del buffer indipendentemente dallo stato dell’ingresso). Il segnale PWM ('Pulse Width Modulation'), è un’onda rettangolare periodica che subisce una variazione del duty cycle in base alla variazione di un certo segnale di riferimento. Il microcontrollore ATmega88A implementa al suo interno una parte hardware dedicata proprio alla generazione di questo segnale.

Ma perché ricorrere al PWM?
L’applicazione di un segnale PWM sulla linea di ENABLE del driver L293D permette una “parzializzazione” della tensione fornita ai motori: in questo modo si riesce a gestire anche la potenza e quindi, mantenendo costante la coppia, a regolare la velocità del robot.

In dettaglio, per via dell’inerzia meccanica, i motori non reagiranno istantaneamente al cambiamento dell’alimentazione loro fornita dunque, pur parzializzando la tensione della loro alimentazione, essendo la frequenza di variazione della tensione molto più grande rispetto ai tempi di reazione del motore, il motore stesso funzionerà

come se ai suoi capi vi fosse una tensione continua: è come se vi fosse un filtro sul segnale PWM offerto dal driver che fa vedere al motore soltanto il valore medio della tensione PWM.



Segnale PWM e valor medio

Indicando con VCC2 la tensione di picco superiore del buffer di potenza dell'integrato L293D, il valore medio del segnale PWM in uscita sarà:

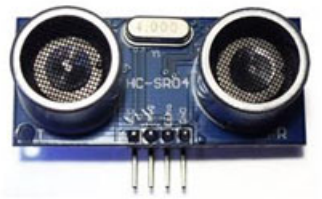
$$V_{avg} = VCC2 \cdot \frac{t_{on}}{T} = VCC2 \cdot Dutycycle$$

Aumentando t_{on} aumenterà il valore medio della tensione vista dai motori, quindi aumenterà la loro velocità. Riducendolo invece si ridurrà la potenza fornita al motore e di conseguenza la sua velocità.

Modulo ad ultrasuoni

Per la rilevazione di ostacoli presenti sul percorso del robot è stato sfruttato un modulo ad ultrasuoni **HC-SR04**. Tale modulo è costituito da 2 capsule:

- Una capsula trasmittente che emette un’onda ultrasonica a 40 kHz;
- un sensore che capta le onde ultrasoniche

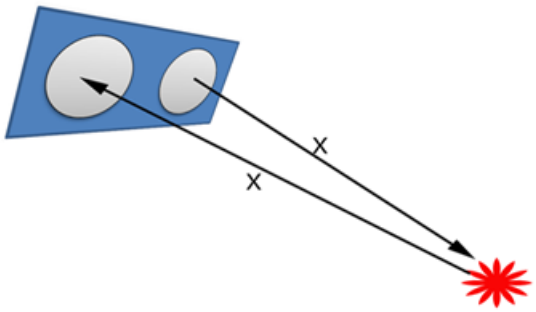


Caratteristiche HC-SR04:

- *Power Supply: 5V*
- *Quiescent current: < 2mA*
- *Effectual angle: = 15°*
- *Ranging distance: 2cm – 4m*
- *Risoluzione: 0.3 cm*

HC-SR04

Tale modulo permette di scoprire la distanza alla quale si trova un oggetto analizzando il tempo impiegato dall'onda sonora a ritornare (dopo la riflessione con un corpo) alla capsula ricevente.



misura della distanza

Il principio dovrebbe essere più o meno questo: noto che il suono si propaga a circa *434m/s*, indicando con *X*[m] la distanza tra un ostacolo ed il modulo, e con *t*[s] il tempo che intercorre tra l'emissione dell'onda sonora e la ricezione della parte riflessa, si trova che:

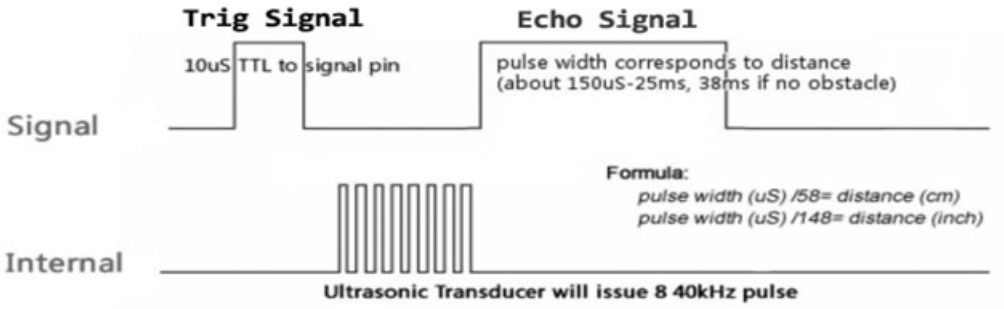
$$2X[m] = 434[m/s] \cdot t[s] \rightarrow X[m] = 0,5 \cdot 434[m/s] \cdot t[s]$$

Tutte le operazioni sono comunque svolte dalla logica di controllo presente a bordo del modulo stesso: ciò facilita notevolmente anche l'interfacciamento con il microcontrollore del robot.

In pratica, forzando ad 1 logico per almeno 10µs il pin di *TRIG* del modulo, verrà trasmesso un segnale a 40Khz (in particolare vengono trasmessi solo 8 impulsi). Dopo l'emissione degli 8 impulsi, viene forzato ad 1 logico il pin *ECHO* (normalmente

basso) del modulo: lo stato di questo pin ritorna basso solo quando viene captato il segnale trasmesso in precedenza, che viene riflesso da qualche ostacolo.

A questo punto, valutando la durata temporale in cui l'ECHO resta alto, sfruttando una formula data dal datasheet del componente, si può risalire alla distanza in cm dell'ostacolo:
d[cm] = width_echo[µs] / 58



Segnali gestiti dal sensore

Per risalire al tempo impiegato dall'ECHO per ritornare basso, si ricorre al timer/counter 0 dell'ATmega88 che viene avviato appena l'ECHO diventa alto, e stoppato quando l'ECHO ritorna basso.

Un ostacolo a 20cm dovrebbe causare un ECHO di durata pari a 1160µs, quindi settando il T/C0 con prescaler di 256 (periodo del timer di 32µs a frequenza di lavoro di 8MHz), il robot dovrà aggirare un ostacolo quando il timer fornirà un valore inferiore a 37.

Sinceramente, con una soglia di 37 il robot arrivava quasi a 10cm dall'ostacolo invece di fermarsi a 20cm. Una soglia più soddisfacente è di 55.

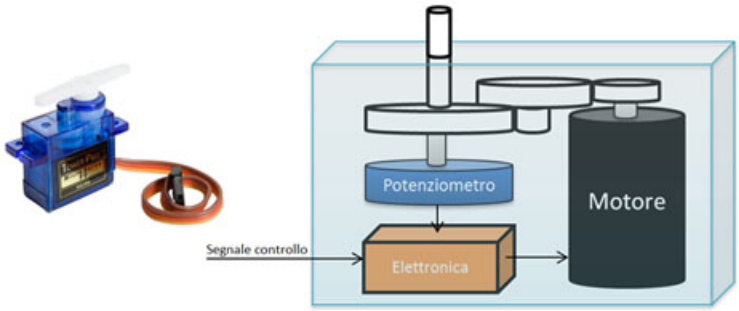
Servocomando

Il modulo ad ultrasuoni rappresenta la **"testa"** dotata di "occhi" del robot.

Quando viene rilevato un ostacolo a meno di circa 20cm, il robot si deve fermare per evitare l'impatto. A questo punto però deve valutare che strada prendere per aggirare l'ostacolo. A riguardo è quindi necessario valutare le altre direzioni per scegliere la strada più libera.

Per far ciò è necessario ruotare il sensore ad ultrasuoni in modo da effettuare una scansione ambientale quindi, proprio per questo, il modulo ad ultrasuoni viene montato sopra un piccolo servomotore da modellismo che rappresenta il **"collo"** del robot e che permette quindi la rotazione della "testa".

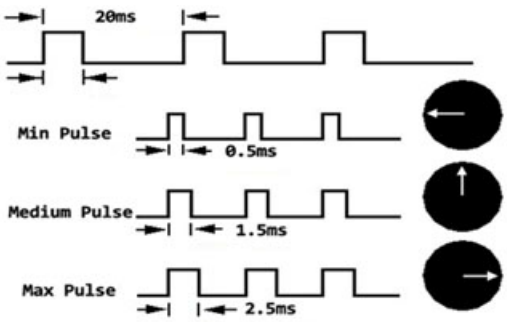
Un servomotore è un organo meccanico di potenza in grado di gestire un movimento comandato mediante un semplice segnale digitale. È generalmente costituito da un motorino DC completo di riduzione meccanica, un sistema di feedback per laposizione dell'asse d'uscita, e di tutta l'elettronica di controllo. Tramite un opportuno sistema di comando è quindi possibile far ruotare l'asse di uscita e posizionarlo in una specifica e precisa posizione.



Servomotore

Un generico servomotore da modellismo è costituito da 3 fili: uno di *alimentazione* (rosso), uno di *ground* (nero o marrone) ed uno di *controllo* (giallo, bianco o arancione).

Il segnale di controllo è di tipo PCM (Pulse-Code Modulation: Modulazione a codifica di impulsi): in pratica è costituito da una serie di impulsi TTL in cui la durata del singolo impulso determina la posizione dell'asse d'uscita. La durata di un impulso può variare tra 0.5ms e 2.5ms. Il periodo tra un impulso e l'altro può variare invece tra 10ms e 40ms (oltre 40ms l'albero del servo potrebbe ritornare nella posizione di riposo).



Segnale di controllo di un servo

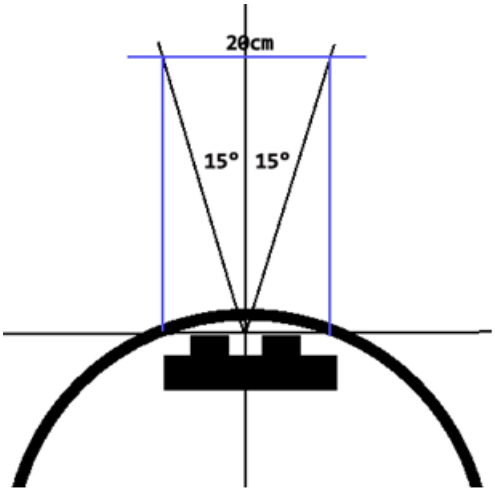
Il servo scelto per il progetto del robot è un **Turnigy TG9** funzionante tra 3V-6V, con coppia di 1.4Kg*cm e velocità di 0.10s/60°.

In posizione di riposo (0° - impulso di durata 1.5ms), il modulo ultrasuoni scansiona la direzione davanti al robot. In presenza di ostacoli, il robot si ferma ed il servo viene ruotato in maniera tale da scansionare le direzioni -90°, -45°, +45°, +90°. Individuata la direzione più libera, il robot ruota verso quella direzione e riprende poi il normale funzionamento.

Interruttori fine-corsa

Siccome il sensore ad ultrasuoni presenta un angolo di divergenza abbastanza ristretto (±15°), essendo la base del robot abbastanza larga (28cm), alcuni ostacoli che fuorisceono dal raggio di controllo potrebbero urtare lateralmente con il robot.

Rispetto al punto di partenza del sensore, rilevando gli ostacoli a 20cm di distanza, il sensora nota gli ostacoli nel raggio trasversale di 20cm*cos(75°) = ±5.2cm.



Zona di controllo del modulo ultrasuoni

Per evitare urti sugli 8.8cm scoperti da un lato e dall'altro, vengono aggiunti dei "baffi" connessi a degli switches che fungono da fine_corsa: quando gli switches vengono premuti, portano a 0V il pin dell'interrupt esterno (INT0) del micro ATmega88: ciò fa scattare una ISR (*Interrupt Service Routine*) che fa tornare indietro il robot di una decina di cm. A questo punto viene avviata una procedura di scanning ambientale per trovare la zona più libera.

Dispositivi Radio

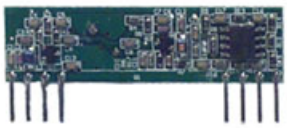
Per la comunicazione radio sono stati sfruttati dei moduli abbastanza economici dell'azienda QUASAR funzionanti a 433.92Mhz con modulazione ASK.

In particolare il trasmettitore è il **QAM-TX1**, mentre il ricevitore è il modulo super-rigenerativo **QAM-RX2**.

Caratteristiche:

- *Ampio range di tensioni (1.5 – 5V)*
- *Modulazione ASK*
- *Data Rate Max: 3Khz*
- *Basso assorbimento di corrente (11mA)*
- *Cmos/TTL input*

QAM - TX1



Caratteristiche:

- *Tensione operativa di 5V*
- *Demodulazione ASK*
- *Data Rate max : 3Khz*
- *Basso assorbimento di corrente (3.5mA)*
- *Cmos/TTL output*
- *VOH = 0.7 VCC*
- *VOL = 0.3 VCC*

QAM - RX2

Descriverò meglio più avanti come avviene la comunicazione radio fra i due moduli.

Stadio di Alimentazione

Tutto il sistema riceve energia da un pacco di 8 batterie ricaribili NI-MH. Ogni batteria presenta una tensione nominale di 1.2V, una tensione a pila scarica di circa 0.9V e una tensione massima subito dopo la carica di 1.37V.

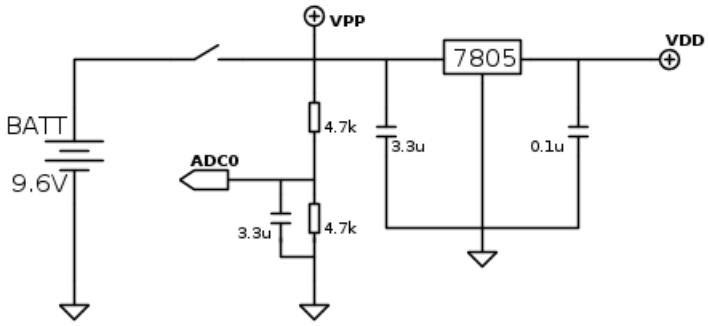
La tensione di tutto il pacco batterie (indicata con **VPP**) viene portata al pin *VCC2* dell'L293D.

Vengono poi ottenuti 5V stabilizzati da un *LM7805*: questi 5V andranno ad alimentare il microcontrollore, la parte a bassa potenza dell'L293D, il modulo ad ultrasuoni, il servomotore ed il ricevitore radio.

Per il monitoraggio dello stato di carica delle batterie, viene sfruttato il convertitore Analogico-Digitale del microcontrollore: tramite un partitore resistivo (che funge da rete di condizionamento), viene suddivisa la tensione totale del pacco batterie e mandata al canale 0 dell'ADC. La tensione che giunge su ADC0 si troverà nel range compreso tra 5.4V (subito dopo una piena ricarica) e 3.6V (carica minima).

Quando il microcontrollore riconosce che la tensione è minore o uguale a 3.6V, deduce che il pacco batterie è scarico ed accende un led rosso di segnalazione. Il robot comunque continuerà a muoversi finchè vi sarà energia a sufficienza.

C'è da dire che la velocità dei motori viene regolata in base allo stato della carica delle batterie, in modo da mantenere costante a circa 6V il valore medio del segnale PWM fornito ai motori (e mantenere quindi costante la velocità del moto del robot).



Microcontrollore

Finalmente arriviamo al nostro ATmega88A!

Esso è un buon giocattolino ad 8 bit fornito dalla Atmel che possiede tutto ciò di cui necessitiamo:

- Almeno 13 pin digitali general purpose (GPIO)(alcuni dei quali dotati di pull-up interno) per la gestione del servomotore, del modulo ad ultrasuoni, del driver L293D, del modulo di ricezione radio e del controllo di alcuni led ed uno switch di start;
- Almeno un GPIO capace di innescare un interrupt esterno alla pressione degli switches fine-corsa;
- Una periferica adibita alla generazione di 2 segnali PWM a 9 bit per il controllo dei motori;
- Un timer/counter necessario alla temporizzazione di alcune routines;
- Un ADC per monitorare lo stato della batteria.

Questo microcontrollore possiede inoltre una memoria FLASH da 8KBytes, una EEPROM da 512Bytes, una memoria RAM da 1Kbyte, una periferica di comunicazione USART, SPI, I2C e altro ancora.

Non mi dilungo sulla descrizione del micro in questione in quanto sarebbe tutto una ripetizione del datasheet (devo ammettere che i datasheets Atmel sono molto chiari al contrario di quelli forniti da Microchip che all'inizio mi hannofatto dannare un po').

Una peculiarità dei micro AVR è quella di avere *Throughput*(cioè numero di istruzioni al secondo) massimo, e quindi un ciclo di lavoro pari proprio al periodo di clock del sistema. Facendo dunque oscillare il clock del sistema ad una frequenza F=8MHz (periodo T = 125ns) mediante l'utilizzo per esempio di un quarzo o di un risuonatore ceramico (come quello usato in questo progetto), il throughput sarà di 8MIPS (MIPS sta' per "Milion Istruction Per Second") ed un ciclo macchina sarà proprio di 125ns. La maggior parte delle istruzioni richiedono un solo ciclo macchina, mentre le restanti (come le istruzioni di salto o di accesso alle memorie) possono richiedere da 2 a 3 cicli.

Dico tutto ciò per non entrare in confusione con i PIC della Microchip nei quali un ciclo macchina richiede 4 cicli di clock.

Il microcontrollore è stato programmato interamente in *Assembly* sfruttando come ambiente di sviluppo *AVR Studio 4* (che ingloba anche un buon simulatore per il debug del firmware).

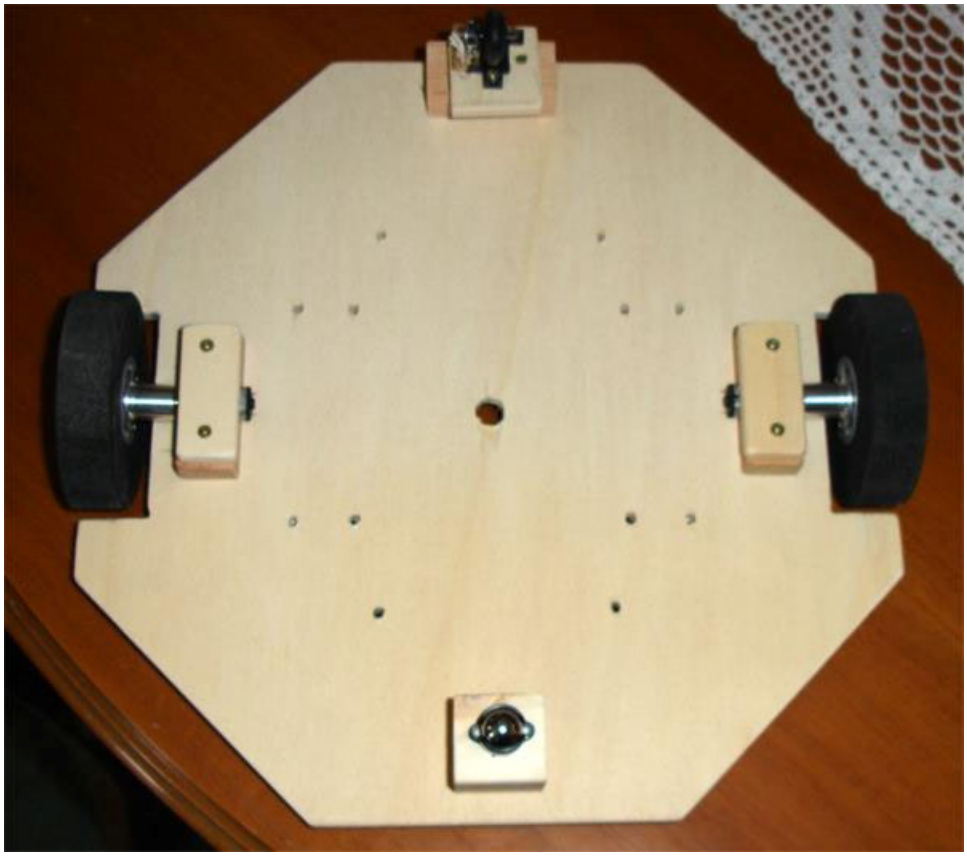
La programmazione fisica del microcontrollore è invece avvenuta ricorrendo alla scheda di sviluppo *STK-500* (ma vanno benissimo anche i programmatori SPI che si trovano in giro su internet a pochi €).

Da notare che, per impostazioni di fabbrica, l'ATmega88a viene fornito con i fuses settati in modo tale da sfruttare l'oscillatore interno. Inoltre è settato anche il fuse relativo alla suddivisione per 8 della frequenza del clock disistema. Quando si va a programmare il micro, siccome ho usato un risuonatore ceramico esterno ad 8MHz per cadenzare il clock del micro, ho disabilitato l'oscillatore interno e ho inoltre disabilitato il prescaler di 8bit in modo da sfruttare la piena frequenza di lavoro.

Realizzazione del Robot

Passiamo adesso ad analizzare il robot nel complesso :)!
La base del robot, realizzata con del compensato, è di tipo ottagonale, con 2 ruote ai

lati ed una ruota di supporto dietro dove vi è il peso maggiore dovuto alle batterie.



Base di C88

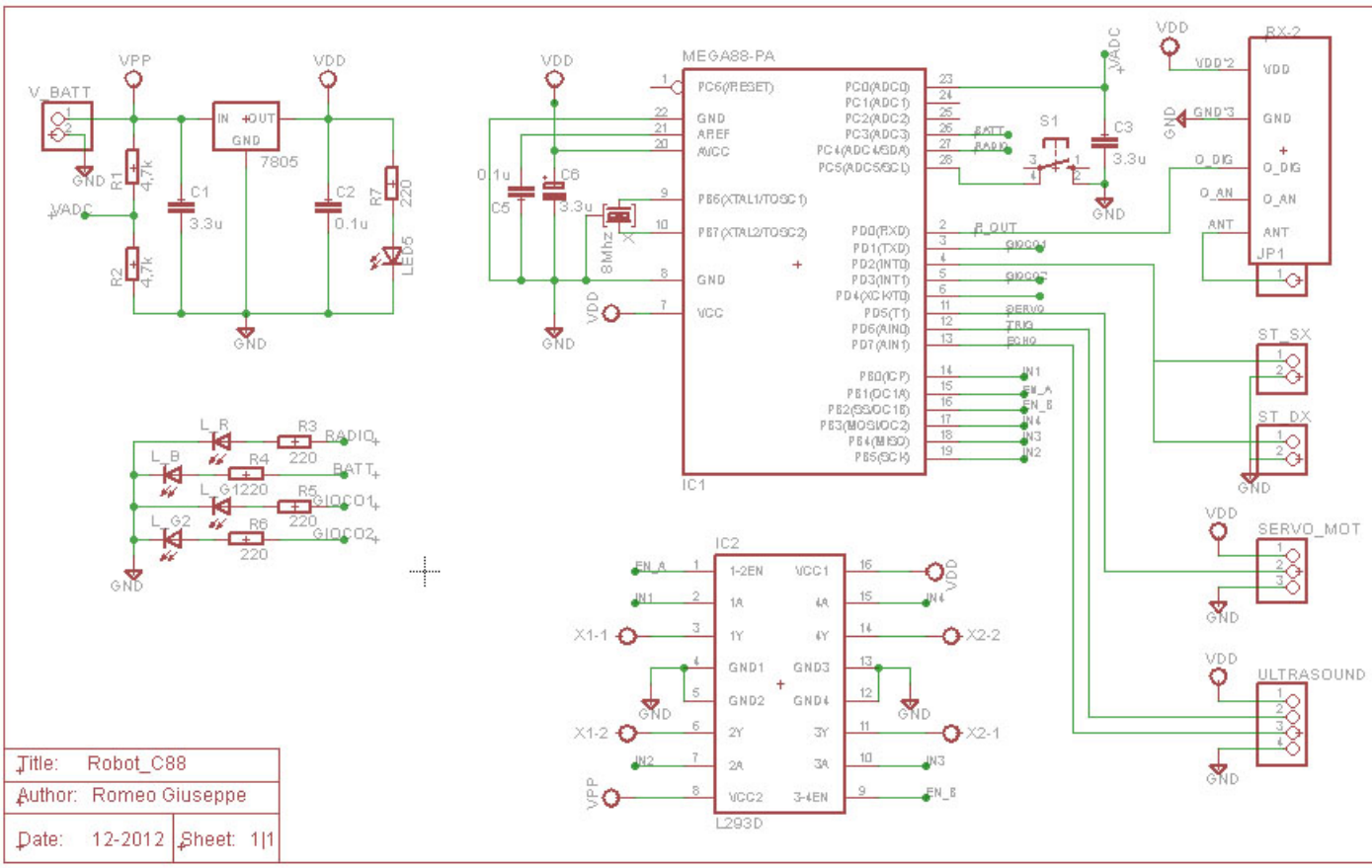
In precedenza la ruota posteriore era data da una sfera omnidirezionali. Siccome sul pavimento di casa faceva un attrito tutt'altro che trascurabile (anziché ruotare, strisciava...) ed un rumore assordante, l'ho rimpiazzata con unarotellina del mouse (di quelle che servono per scorrere le pagine su e giu).

La ruota omnidirezionale l'ho spostata poi avanti per ulteriore supporto (ma non tocca praticamente mai a terra visto che il baricentro del robot è parecchio spostato posteriormente).

Le ruote sono in neopropilene e sono connesse all'albero dei motori per mezzo di mozzi da modellismo (devo dire che costano più i mozzi e le ruote che tutto il robot con l'elettronica a bordo... costavano parecchio anche le steffette persostenere i motorini, ma per fortuna sono riuscito ad arrangiarmi con dei pezzi di legno nei quai

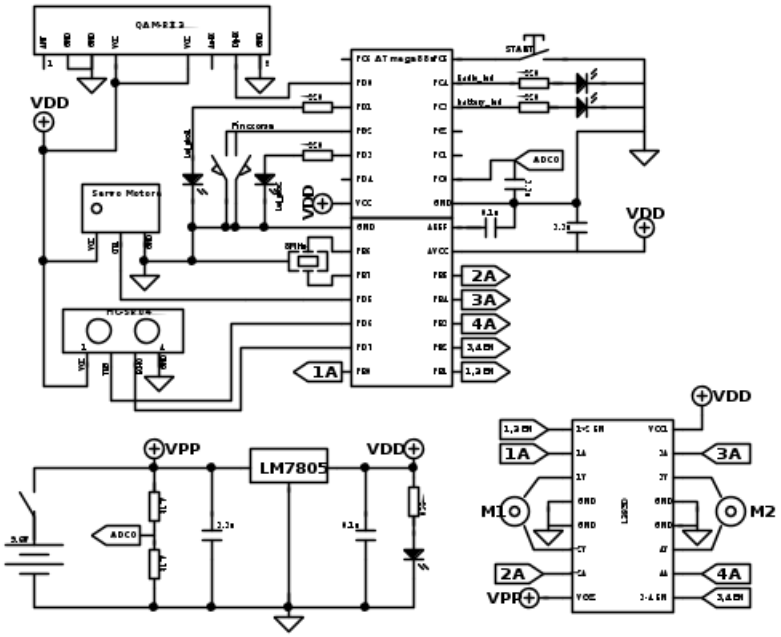
ho incassato per benino i motorini :D).

In basso riporto lo schema circuitale realizzato in Eagle della centralina del robot:

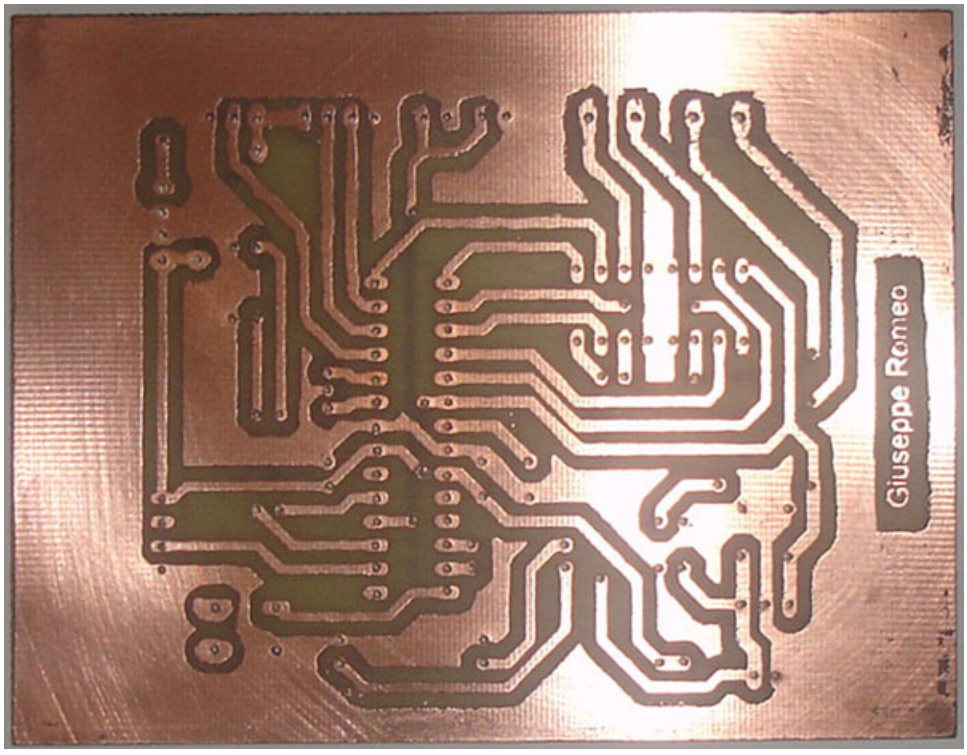


Schema circuitale in Eagle

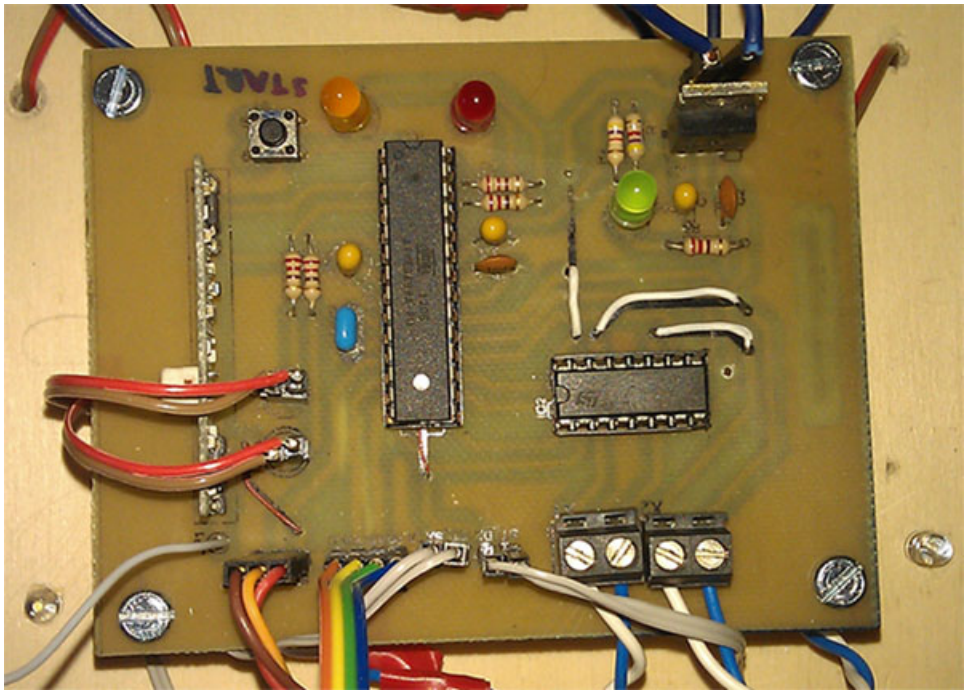
Riporto anche lo schema in FidoCad (anche se è un po' confusionario...):



Ecco adesso delle foto relative al PCB realizzato:

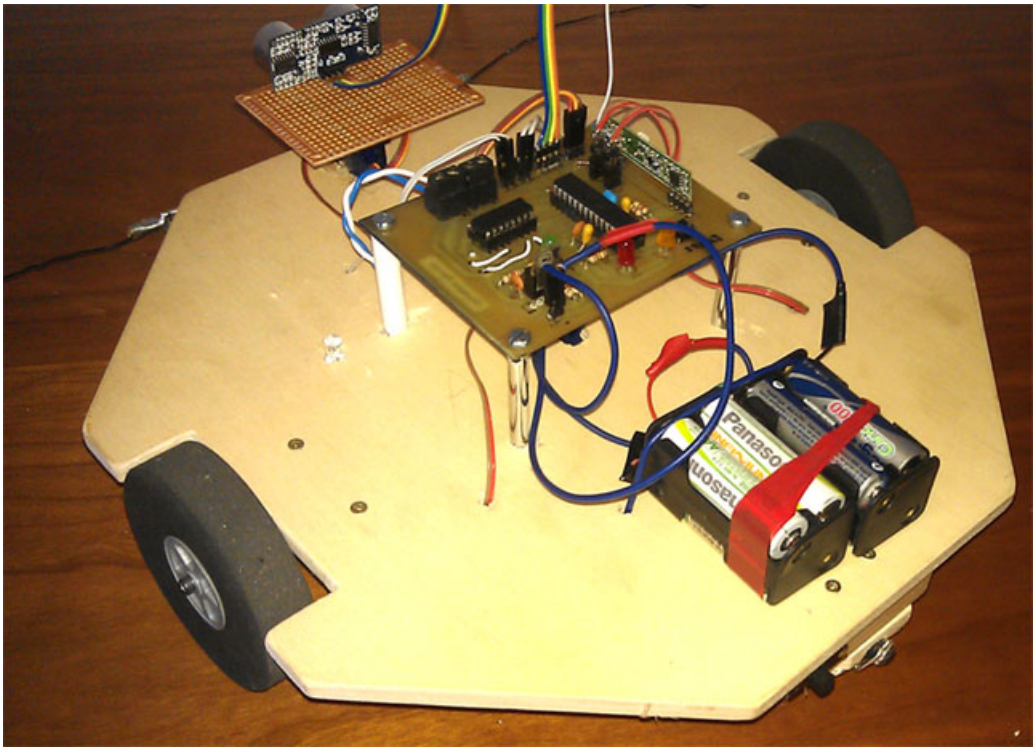


PCB su piastra in vetronite



Centralina C88

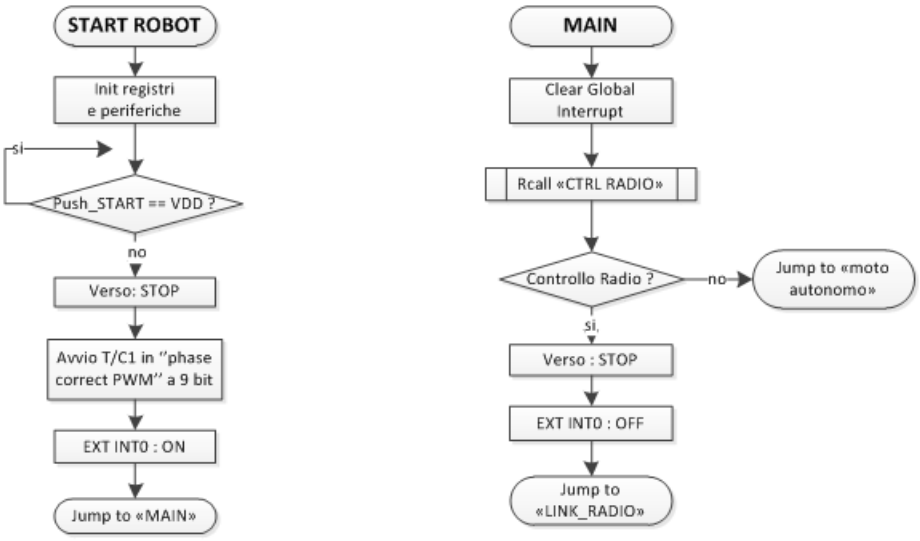
Ed ecco un'altra foto del robot nel complesso:



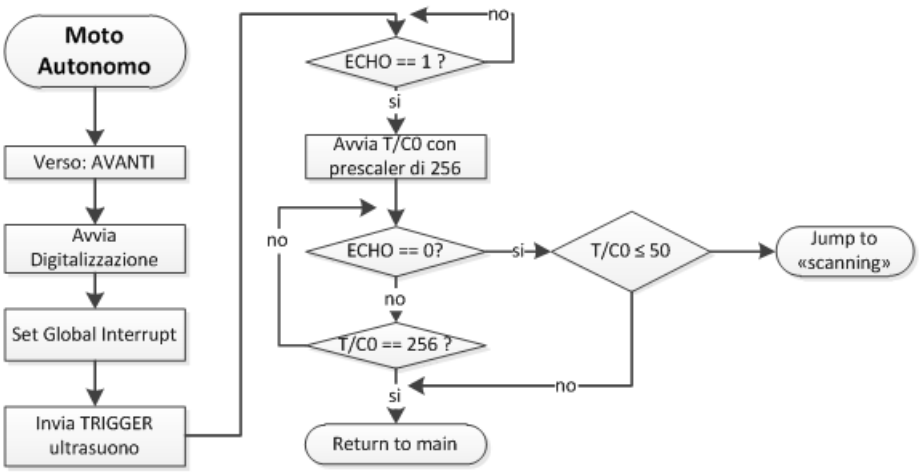
C88 - vista posteriore

Sulla parte posteriore del robot, alla sinistra della rotellina posteriore, vi è un interruttore posto tra le batterie e la scheda del robot: chiudendolo, si manderà energia a tutto il robot (sia scheda elettronica che motori).
Quando alimentato, il microcontrollore inizia a settare opportunamente tutti i registri interni e pone il servomotore in posizione centrale di riposo. Comunque il robot resterà ancora fermo: per avviarlo sarà necessario premere lo switch **"START"** presente sulla scheda.

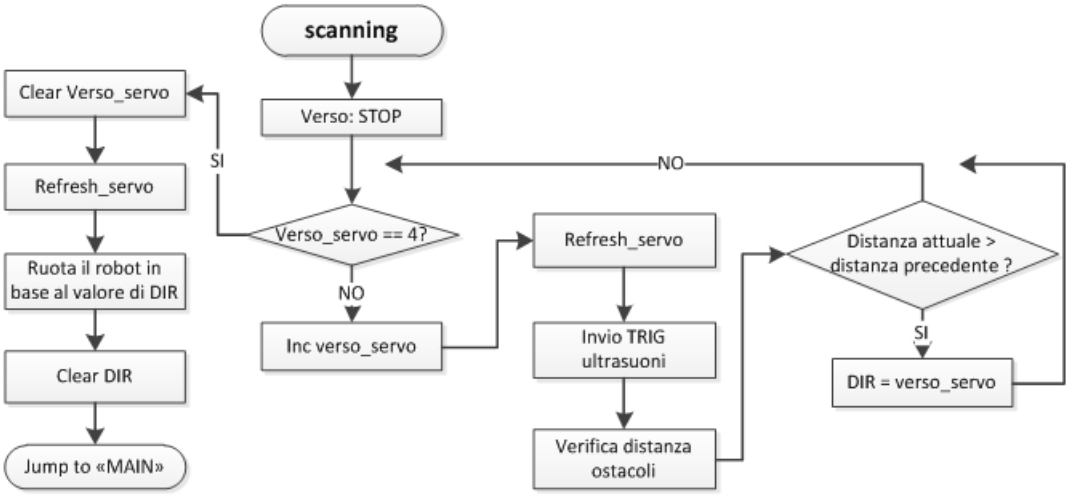
Premuto lo START, il microcontrollore entra nel *main* vero e proprio del progetto dove si va a valutare se è necessario avviare il robot in modalità autonoma o controllata.



Flow chart del main



Flow chart della routine di moto autonomo



Flow chart della routine di scansione ambientale

Descrizione Comunicazione Radio

Passiamo adesso a vedere un po' la parte radio.

I modulini a 433MHz della QUASAR che ho menzionato in precedenza, li avevo trovati qualche anno fa per pochi spiccioli. Pensavo di poterli gestire mediante protocollo

RS-232 (sfruttando le periferiche USART del micro ricevitore e trasmettitore) come possibile per tantissimi moduli radio più costosi a 433MHz o 2.4GHz.

Purtroppo, quando ho provato a mandare dati seriali secondo std. RS232 dal trasmettitore al ricevitore, il risultato è stato a dir poco pessimo…

In pratica avevo connesso il TX della USART di un ATmega8515 (il microcontrollore di transizione che usavo all'epoca per implementare il telecomando) al pin TX del trasmettitore QUASAR, ed avevo fatto un piccolo firmware che prevedeva l'invio di 8 bit sfruttando la USART dell'8515.

Dal lato ricevitore, mi ero preoccupato solo di valutare lo stato dell'uscita digitale (pin 7) del QAM-RX-2 ricorrendo ad un oscilloscopio.

Ciò che vedevo all'oscilloscopio era un segnale abbastanza diverso da quello inviato infatti, alcuni livelli alti TTL trasmessi scomparivano (sostituiti da 0 logico) o duravano pochissimo rispetto al baudrate imposto.

Probabilmente, sullo stadio d'ingresso del trasmettitore è presente un qualche accoppiamento AC che va a bloccare le componenti continue (e quindi i segnali che restano alti per un certo tempo). D'altronde, per quanto li avevo pagati, non potevo aspettarmi miracoli...

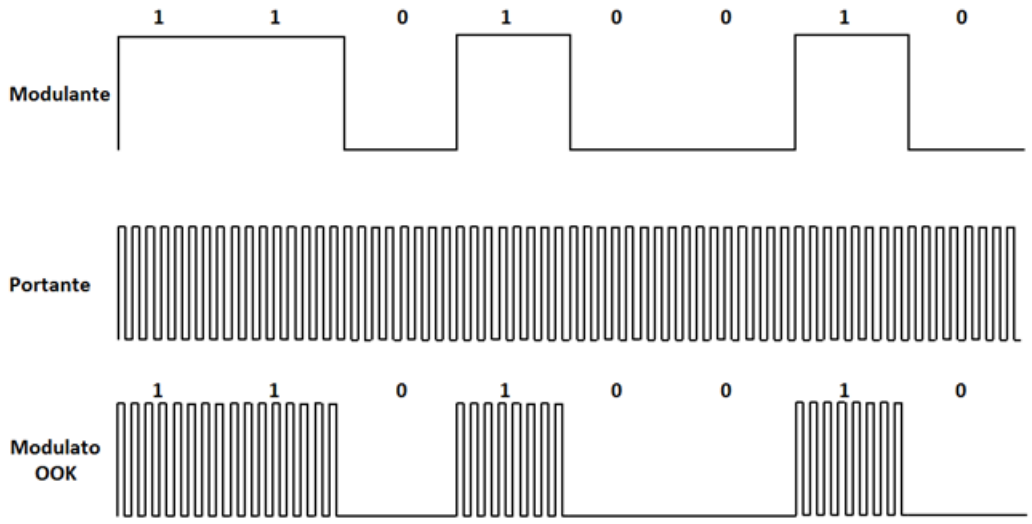
Non potendo ricorrere alla comunicazione secondo std. RS-232, si è reso necessario modulare il segnale da trasmettere in modo tale da renderlo a componente continua minima (quindi è necessario modulare l'1 logico in modo tale che abbia parecchie transizioni al suo interno che non vengano bloccate dall'ipotetico condensatore serie posto in ingresso).

Per quanto riguarda la modulazione del segnale da trasmettere, ho deciso di ricorrere ad una modulazione implementata via software molto simile all' *ASK-OOK*, con una portante prossima ai 3KHz (i moduli radio utilizzati non garantiscono il corretto funzionamento con segnali avente frequenze superiori ai 3KHz).

La modulazione ASK (*Amplitude Shift Keying*) può essere vista come la controparte digitale (modulante digitale) della modulazione analogica (quando la modulante è analogica) AM in quanto l'informazione è codificata nell'ampiezza della portante: il segnale modulato non sarà altro che il segnale portante la cui ampiezza viene "shiftata" (quindi spostata) tra 2 livelli differenti a seconda che il bit da trasmettere sia 1 logico o 0 logico. Generalmente il segnale modulato dell'1 logico presenta un'ampiezza più grande, mentre quello dello 0 logico un'ampiezza molto più attenuata.

Quando in corrispondenza dello 0 logico della modulante l'ampiezza della portante viene azzerata, si parla di modulazione **OOK** (*On-Off Keying*) in quanto il segnale

modulato assume due soli livelli: ON oppure OFF.



Modulazione OOK

Nell'implementazione software, la portante l'ho imposta di tipo digitale con una frequenza prossima a 2.86KHz (T=350µs), mentre la modulante è data da una sequenza binaria in cui ogni bit ha durata di 10ms (quindi un baudrate di 100bps).

In presenza di 1 logico, quindi, il segnale modulato presenterà una transizione da uno stato all'altro ogni 175µs: in ogni bit quindi vi dovranno essere 10000µs/175µs = 57 transizioni.

Come già detto in principio, utilizzare un ATmega8515 per realizzare il trasmettitore radio era uno spreco assurdo (in quanto mi servivano solo 7 GPIO ed un T/C ad 8 bit). Non avendo altri AVR di piccola taglia, ho sfruttato un *PIC16F628A*.

Ho comunque realizzato un firmware per *ATtiny2313*.

Devo ammettere che il passaggio dall'assembly degli AVRs a quello dei PIC16F è stato traumatico, anche perchè con i PIC, fino ad oggi, non ci avevo mai fatto granchè...

Ho trovato che anche i loro datasheet sono più ostici e scarni rispetto a quelli degli AVR (sarà forse pure la simpatia che avevo per gli AVR...).

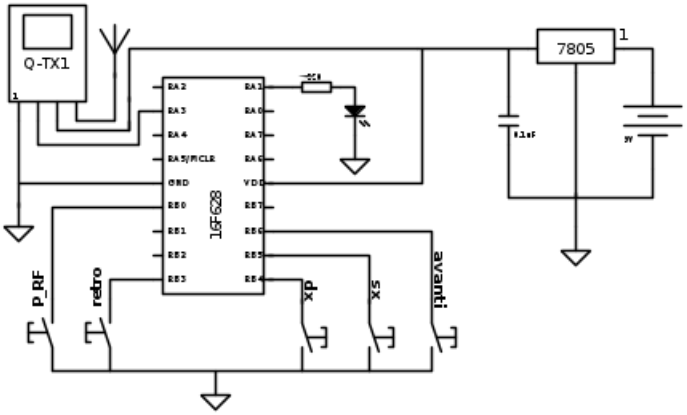
Per non parlare poi dei registri delle periferiche suddivisi in 4 banchi (alcuni registri persino ripetuti su più banchi) e dell'assenza dei 32 general purpose register con i quali ormai ero abituato sugli AVR.

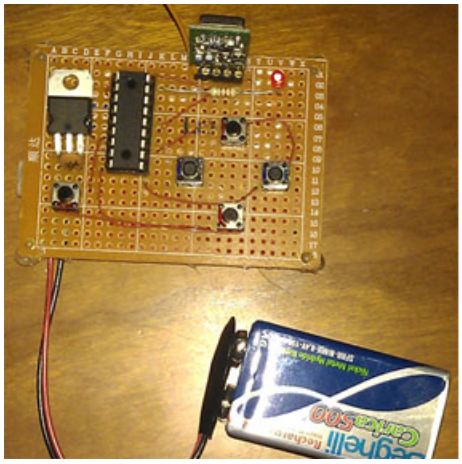
Comunque, dopo il primo impatto, non è stato male lavorare nemmeno con i PIC. :)

Per la loro programmazione software in *Assembly* ho sfruttato l'ambiente di sviluppo **MPLAB 8.73**, mentre per il trasferimento dell'eseguibile sulla flash del micro ho sfruttato un **PIC-Kit2**.

Per quanto riguarda la frequenza di lavoro, ho sfruttato l'oscillatore interno a 4MHz.

In basso riporto lo schema circuitale del radio-comando (non ho fatto alcun layer PCB in quanto ho trovato più comodo implementarlo su millefori:





Radiocomando

Dei 5 swithces, 4 sono utilizzati per scegliere la direzione del robot (vai avanti, ruota a sx etc), mentre 1 (P_RF) serve per instaurare/abbandonare il link RF.

Quando è instaurata una comunicazione radio con il robot, si accende il led rosso del radiocomando e quello arancione della scheda del robot.

I dati da trasmettere fuoriescono dal pin RA3 del pic è presentano una struttura di questo tipo:

- 1 sequenza di START con 5 bit alti ed 1 basso;
- 4 bit dati per comandare il robot;
- 4 bit di controllo d'errore (sono i negati dei 4 bit precedenti);
- 2 bit di stop bassi.

Per avere delle temporizzazioni abbastanza accurate (alterare lo stato del segnale da trasmettere ogni 175µs), si ricorre al Timer0 ad 8 bit (usando l'oscillatore interno a 4MHz, la frequenza di lavoro è di 1MHz e quindi il timer viene incrementato ogni microsecondo) e si sfrutta l'interrupt sull'overflow (precaricando nel timer un valore di partenza).

In principio, il pin RA3 (TX) è basso. Quando sopraggiunge un'interrupt da overlow, la ISR va a valutare il bit da trasmettere: se tale bit è 0, non accade nulla, mentre se è 1, si va a negare RA3 invertendone lo Stato logico.

Siccome un bit ha una durata di 10ms, devono essere eseguite 57 ISRs prima di passare al bit successivo (o tornare eventualmente al MAIN del programma).

Nel caso in cui si prema *P_RF*, il PIC va a valutare lo stato del LED: se lo nota acceso, capisce che è richiesta la disconnessione dal robot, altrimenti capisce che si vuole comandare il robot da remoto.

Nel caso si voglia instaurare un link RF, la sequenza di bit trasmessa verso il Robot è un susseguirsi di uni e zeri (1010….0101…) che persiste per tutto il tempo in cui si mantiene premuto P_RF.

Nel caso in cui invece ci si voglia scollegare dal robot, facendolo muovere autonomamente, viene invece inviata una sequenza alternata di coppie di uni e zeri (11001100)preceduti da una sequenza di START (in quanto, in modalità controllata, il Robot filtra tutte le stringhe che non sono precedute dallo START).

Quando è instaurato il link RF, premendo uno qualsiasi dei 4 switches di controllo, verranno inviate 4 stringhe binarie che, una volta decodificate dall'ATmega88 in

ricezione, imporranno al robot di muoversi avanti/indietro, oppure di ruotare su se stesso a destra/sinistra.

L'imposizione continua per tutto il tempo in cui viene mantenuto premuto lo switch. Una volta rilasciato, il robot resta fermo.

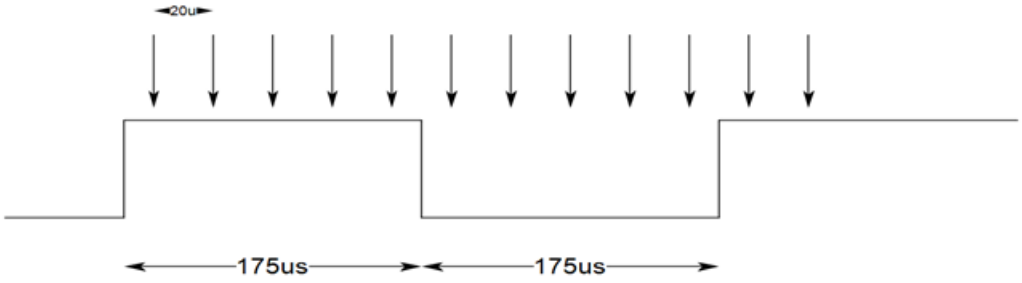
Sul Robot, una volta instaurato il link RF, l'ATmega88a verificherà continuamente (in polling) se vi sono dati in arrivo, cercando di captare la sequenza di START.

Se lo START viene agganciato, il microcontrollore capisce che sta' giungendo un codice da remoto, quindi campiona gli 8 bit e li decodifica: sarà verificata come prima cosa l'integrità del codice (i primi 4 bit ed i successivi 4 bit devono essere negati fra loro) e poi passerà all'interpretazione del codice.

La routine di ricezione campiona ad intervalli regolari di 20µs il pin PD0 connesso all'RX del QAM-RX2. 20µs è un buon periodo di campionamento in quanto è circa 18 volte maggiore della frequenza del segnale da ricevere.

Da notare che i campionamenti avvengono su porzioni di segnale da 1ms: quindi su un bit (10ms), suddivido il segnale in 10 porzioni (ognuna da 1ms) e campiono ogni porzione con un periodo di campionamento di 20µs. In questo modo, su 1ms vado a leggere 1000µs/20µs = 50 campioni.

In caso il bit in ricezione sia 1 logico, i campioni prelevati in 1ms dovrebbero essere per il 50% alti e per il restante 50% bassi (per via della modulazione OOK). Quindi idealmente al millisecondo letto si dovrebbe assegnare 1 logico nel caso in cui il numero di campioni alti risultasse 25.



Campionamento del segnale in ricezione

Per avere una certa tolleranza, assegno 1 logico al millisecondo letto se il numero di campioni alti è maggiore o uguale a 15, altrimenti assegno 0.

Acquisite tutte e 10 le porzioni di 1 bit (10ms), vado a valutare quante porzioni sono alte, e quante basse. In particolare, se le porzioni alle quali è stato assegnato 1 logico sono maggiori o uguali a 6, assegno al bit il valore di 1 logico, altrimenti di 0.

Comunque questi parametri di soglia sono gestibili a piacimento. Io mi sono trovato bene con questi valori!

Per come è strutturata la fase di campionamento, per permettere una buona sincronizzazione dei dati, il primo bit di ogni sequenza che sopraggiunge dopo lo START è stato scelto pari ad 1 logico: in questo modo, durante la lettura dello 0 logico dello START, quando sopraggiunge l'1 logico della sequenza, il microcontrollore si accorge quasi subito (o al max con uno scarto di 20us) dell'inizio del segnale di comando, passando immediatamente alla sua lettura.

Letti tutti gli 8 bit del segnale di comando, verrà effettuata un'operazione di XOR tra i primi 4 e gli ultimi 4: se l'operazione da "1111", allora il codice ricevuto è corretto e si passerà alla sua decodifica, altrimenti verrà scartato.

La decodifica dei codici non è altro che un confronto con delle stringhe memorizzate internamente: in base alla stringa riconosciuta, si dirà al robot come muoversi...

Stato delle batterie e regolazione della velocità

Altro discorso particolare è quello che lega la velocità dei motori allo stato delle batterie.

Come già detto, la velocità dei motori è legata al valor medio del segnale PWM offerto dai buffer dell'integrato L293D ai motori.

Il PWM in uscita dal driver è identico in forma al PWM generato dal microcontrollore (la differenza sta' nel range di tensioni: il PWM del driver di potenza oscillerà tra [0;VPP]Volt, mentre il PWM del micro oscillerà tra [0;VDD]Volt.

Se venisse lasciato un duty cycle sempre costante ed indipendente dallo stato delle batterie, a piena Carica il Robot andrebbe più veloce, per poi diminuire la sua corsa a man mano che la carica delle batterie decresce (questo perché VPP è proprio la tensione offerta dal pacco batterie, che passa da circa 10.9V a piena carica, ad una tensione di circa 7V a pacco batterie scarico).

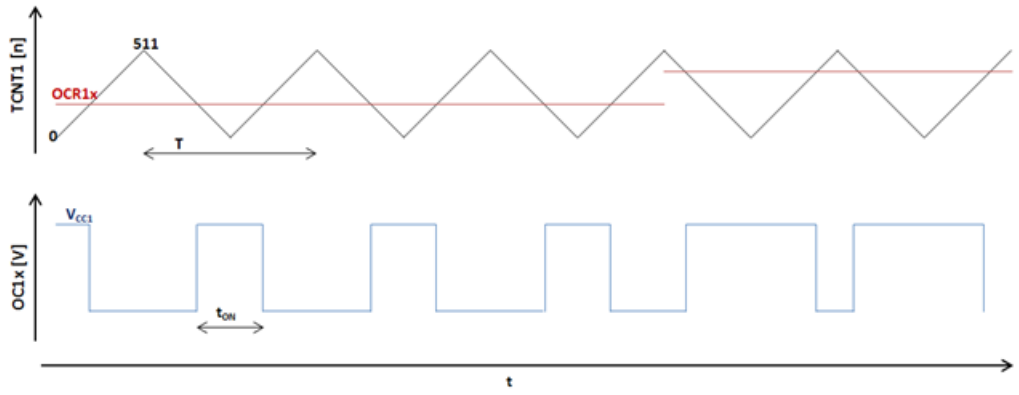
Siccome V_{motor} = VPP * D, se D è costante V_{motor} dipenderà linearmente da VPP.
Da notare che V_{motor} è il valor medio della tensione fornita ai motori.

Per ovviare al problema, l'unico modo per mantenere V_{motor} costante è quello di aumentare il duty cycle a man mano che VPP si riduce (a discapito logicamente di una più rapida scarica delle batterie).

Prima di addentrarci nel meccanismo di controllo, cerchiamo di vedere come il microcontrollore genera il segnale PWM sfruttando il T/C1 a 9 bit in modalità *phase correct*.

In modalità *"Phase Correct PWM"*, il T/C1 incrementa e decrementa continuamente il suo valore (contenuto nel registro **TCNT1**), contando da 0 a 511 (511 è il TOP del timer quando lavora a 9bit), e poi decrementando il valore del contatore da 511 a 0.

Questo salire e scendere periodico del valore del TCNT1, se graficato, mostrerebbe un'onda triangolare avente un periodo T_{TRI} pari a *2*dt*512*, dove *dt* è il tempo di incremento del T/C1: lavorando senza prescaler ad una frequenza di 8MHz, *dt* è pari a 125ns, quindi T_{TRI}=128µs.



PWM generato dal T/C0 dell'AVR

Per poter generare un segnale PWM, al T/C1 è associata una periferica detta di *"compare match"* che confronta di volta in volta il valore binario contenuto nel registro **OCR1x** con il valore assunto dal timer nel registro TCNT1.

Il valore di *OCR1x* serve per creare un riferimento per la generazione del PWM: quando il valore di *OCR1x* è maggiore del valore di *TCNT1*, l'uscita del pin **OC1x** sarà imposta ad 1 logico. Appena il valore di *OCR1x* diventa inferiore al valore di *TCNT1*,

allora l'uscita del pin *OC1x* sarà impostaa 0 logico.

Da una similitudine fra triangoli si nota che
$$\frac{t_{on}/2}{T/2} = \frac{OCR1x}{511} \rightarrow \frac{t_{on}}{T} = D = \frac{OCR1x}{511}$$

Di conseguenza si ha che $V_{avg} = VDD \cdot \frac{OCR1x}{511}$

La tensione in uscita dal driver sarà invce:

$$V_{motor} = VPP \cdot \frac{OCR1x}{511}$$

Volendo quindi una certa V_{motor}, si deve avere

$$OCR1x = \frac{V_{motor} \cdot 511}{VPP}$$

Quindi, giocando su *OCR1x*, si va a regolare V_{motor}.

A questo punto, ciò che è necessario fare, è legare il valore *OCR1x* in modo che vari in maniera inversa allo stato di VPP.

Il convertitore analogico digitale del microcontrollore, settato con un prescaler di 128 (Fs=62,5KHz) e risoluzione a 10bit, va ad acquisire la metà del valore di VPP (la metà in quanto vi è il partitore resistivo).

Il valore digitale del segnale analogico acquisito risulta essere:

$$V_{dig} = \frac{V_{an} \cdot 1023}{V_{ref}}$$

Siccome il riferimento è preso internamente ed è pari a 5V, e siccome V_{an} = VPP/2, si ha:

$$V_{dig} = \frac{VPP \cdot 1023}{10} \rightarrow VPP = \frac{10 \cdot V_{dig}}{1023}$$

Quindi:

$$OCR1x = \frac{V_{motor} \cdot 511}{2V_{an}} \rightarrow OCR1x = \frac{V_{motor} \cdot 511 \cdot 1023}{10 \cdot V_{dig}}$$

Se il firmwire fosse stato scritto con linguaggi a più alto livello come il C, ricorrendo a librerie matematiche l'equazione in alto sarebbe facilmente implementabile.

La risoluzione in assembly invece mi avrebbe sicuramente portato ad epilessie notturne: avrei dovuto creare una routine di divisione a 10 bit, un'altra di moltiplicazione a 16 bit, gestire poi i floating etc...

Una soluzione notevolmente più semplice è quella di ricorrere per esempio ad una look-up-table da sfruttare in dei confronti con il valore digitalizzato.

V_an	V_dig	V_dig in binario	OCR1 @ V_{AVG} = 6V
5,0	1023	11-1111-1111	**307**
4,9	1003	11-1110-1011	**313**
4,8	982	11-1101-0110	**319**
4,7	962	11-1100-0010	**326**
4,6	941	11-1010-1101	**333**
4,5	921	11-1001-1001	**341**
4,4	900	11-1000-0100	**348**
4,3	880	11-0111-0000	**357**
4,2	859	11-0101-1011	**365**
4,1	839	11-0100-0111	**374**
4,0	818	11-0011-0010	**383**
3,9	798	11-0001-1110	**393**
3,8	777	11-0000-1001	**403**
3,7	757	10-1111-0101	**414**
3,6	737	10-1110-0001	**426**
3,5	716	10-1100-1100	**438**

Valore di OCR in base al valore digitalizzato

Siccome era un po' noiosa la questione dei confronti, ho provato ad escogitare un'altra strategia che sfruttasse dei valori noti presi a riferimento e fosse in grado di adattarsi di volta in volta in base al valore digitalizzato.

Dopo vari tentativi sono riuscito a trovare qualcosa di accettabile.

In pratica, considero come elementi di riferimento V_{dig} = 1023, il suo corrispondente valore di OCR (307) e la differenza Z = V_{dig} - OCR (716).

Poi mi serve conoscere la distanza del valore digitalizzato (per esempio 900) dal valore di riferimento (1023).

Per capire l'algoritmo è comunque necessario osservare la seguente tabella:

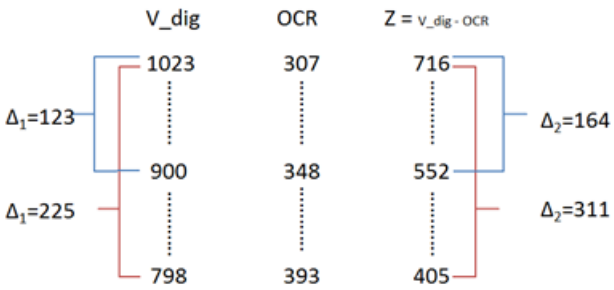


Tabella di riferimento

Ho i valori noti di riferimento (1023 e 716) e poi considero 2 casi separati: uno in cui il valore digitalizzato sia 900 (stato della batteria a 8,8V), l'altro in cui invece il valore digitalizzato sia 798 (stato della batteria a 7,8V).

Concentriamoci al caso in cui il valore digitalizzato sia 900:

900 dista **Δ=123** da 1023. Il corrispettivo valore dato dalla differenza tra V_{dig} e OCR (ossia **Z=552**) dista invece **Δ2=164** da 716.
Ciò che vogliamo ottenere è invece **OCR = 348** (o un qualcosa di molto vicino).

Quello che noi possiamo calcolare una volta ottenuto il valore digitalizzato è solamente "123"!

Se conoscessimo anche **Δ2** ("164"), l'OCR poteva essere calcolato come **900-(716 – Δ2)**.

Come trovare Δ2?

Ho notato che effettuando **Δ1+(Δ1/4) +(Δ1/6)** si ottiene un valore molto vicino a *Δ2*!!!

In binario basta effettuare delle divisioni per 2 in successione ricorrendo a degli shift a destra di una posizione.

Dimostrando:

123+(123/4)+(123/6) = 123+30+15 = **168** che è molto vicino a **164**

Quindi *OCR= 900-(716-168) = 900-548 =* ***352*** che è molto vicino al valore corretto **348**.

L'errore di 4 bit è più che tollerabile in quanto causa una discrepanza di V_{motor} di pochi mV. A ciò aggiungiamo che pure i valori in tabella sono arrotondati (infatti, realmente sono dei floating, ma nel registro OCR1x è necessario piazzare dei valori interi).

Verifichiamo l'algoritmo nel caso in cui il valore digitalizzato sia *798*:

225+(225/4)+(225/6) = 225+56+28 = ***309*** che è molto vicino a **311**.

Quindi:

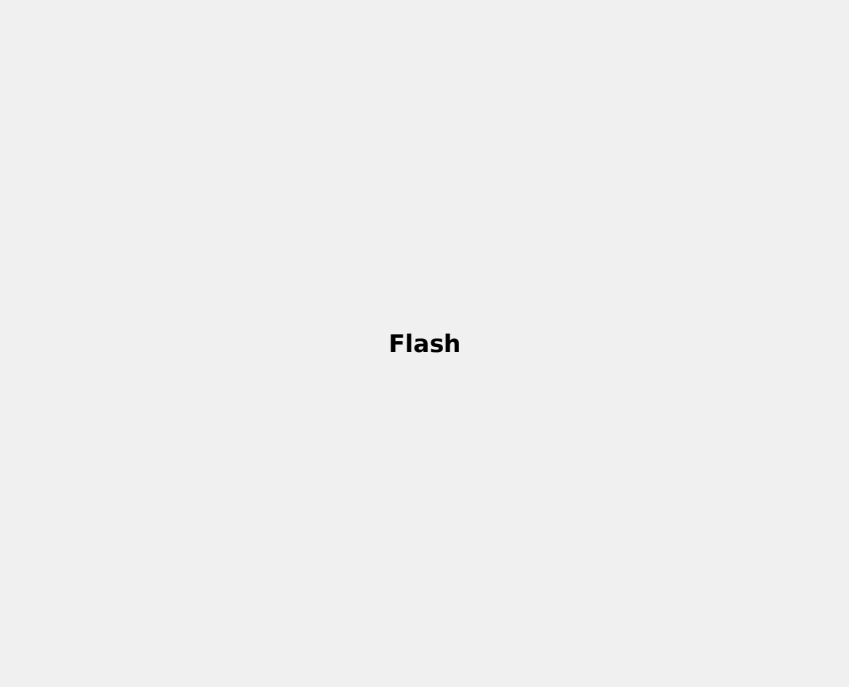
OCR = 798-(716-309) = 798 – 407 = ***391*** che è vicinissimo al valore corretto 393!

Da sottolineare che il micrcontrollore non esegue questo algoritmo ogni volta che avviene l'ISR dell'ADC, ma viene eseguito sul valore medio valutato di volta in volta su 10 digitalizzazioni.

Conclusioni

A questo punto penso di essere finalmente arrivato alla fine. Spero che il progetto vi piaccia.

Qui un piccolo video sul funzionamento del robottino!!!



Logicamente possono essere fatte parecchie modifiche e migliorie.

Ricorrendo a linguaggi più ad alto livello è possibile sicuramente implementare delle cose più complesse come un regolatore PID per il controllo della velocità delle 2 ruote, oppure aggiungere la funzionalità di “line follower” etc.

La comunicazione radio può essere estesa ricorrendo ai famosi moduli RN-XV o X-BEE che permettono una comunicazione non più unidirezionale da TX ad RX, ma bidirezionale (consentendo così anche al robot di mandare dati al trasmettitore, come informazioni sullo stato della batteria, eventuali dati acquisiti da un qualche sensore a bordo etc).

Le idee per proseguire sono molteplici (i soldi per comprare le cose un po’ meno :D :D).

Download allegati: Allegati C88 - Firmware e File Eagle

Riferimenti

Motori e pilotaggio
http://forum.roboitalia.com/showthread.php?t=8627
http://www.oocities.org/tommaso84/attrvolv_testo.htm
http://www.giovannitonzig.it/integrazioni/meccanica/meccanica_10_attrito.pdf
http://www.datasheetcatalog.org/datasheet/texasinstruments/l293d.pdf
http://digilander.libero.it/beamweb/ponte_h.htm
http://www.electroyou.it/vis_resource.php?section=Lezio&id=143
http://modularcircuits.tantosonline.com/blog/articles/h-bridge-secrets/sign-magnitude-drive/
http://modularcircuits.tantosonline.com/blog/articles/h-bridge-secrets/lock-anti-phase-drive/
http://www.electroyou.it/vis_resource.php?section=DomRisp&id=394

Servocomandi:
http://www.settorezero.com/wordpress/come-funziona-un-servocomando/
http://www.electroyou.it/vis_resource.php?section=ArtCorso&id=85
http://www.adrirobot.it/servotester/il_servomotore.htm

Modulo ad Ultrasuoni:
http://www.settorezero.com/wordpress/il-sensore-ad-ultrasuoni-hc-sr04/
http://www.electroyou.it/tipu91/wiki/iniziamo-con-arduino

Microcontrollori AVR:
http://www.electroyou.it/tardofreak/wiki/iniziare-con-avr-atmega
http://sd2cx1.webring.org/l/rd?ring=avr;id=71;url=http%3A%2F%2Favr-asm.tripod.com%2F
http://www.embedds.com/category/avr-projects/
http://extremeelectronics.co.in/
http://www.electroyou.it/crestus/wiki/tutorial-pic-guida-passo-passo-per-i-principianti
http://www.electroyou.it/davidde/wiki/pickit2

Comunicazione radio:
http://www.ettorepanella.com/dmdocuments/Modulazione-Demodulazione%20ASK-OOK.pdf
http://swapf.altervista.org/elettronica.php?read=2400_moduliradio&page=all
http://electronicdesign.com/communications/understanding-modern-digital-

modulation-techniques

Estratto da "http://www.electroyou.it/mediawiki/index.php?title=UsersPages:Giurom88:robottino-c88"