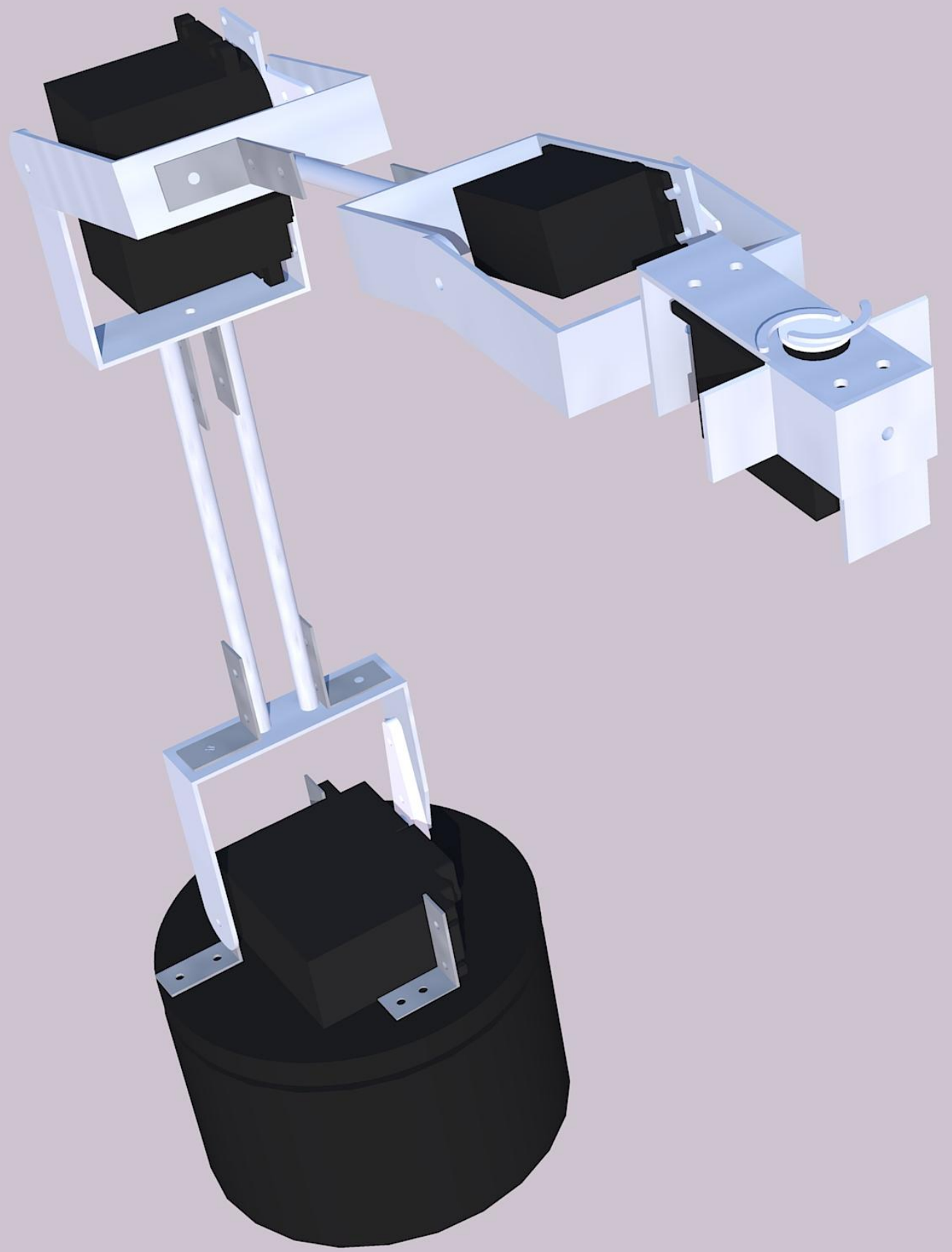
Braccio Robotizzato

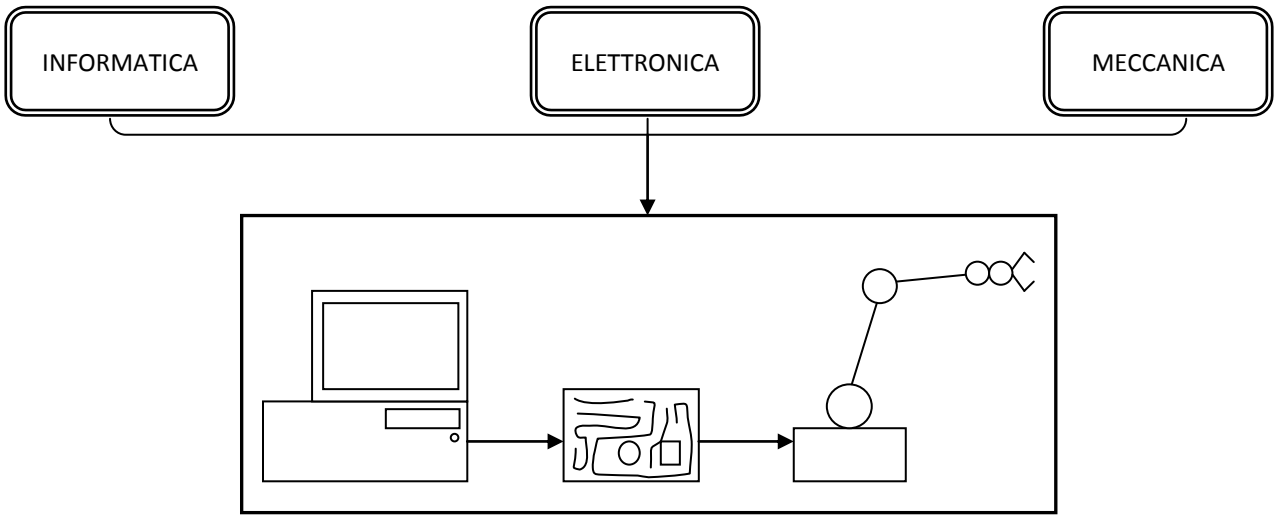


di Mattia Marcon
Cl. VAET A.S. 2009-2010

Braccio Robotizzato

Introduzione:

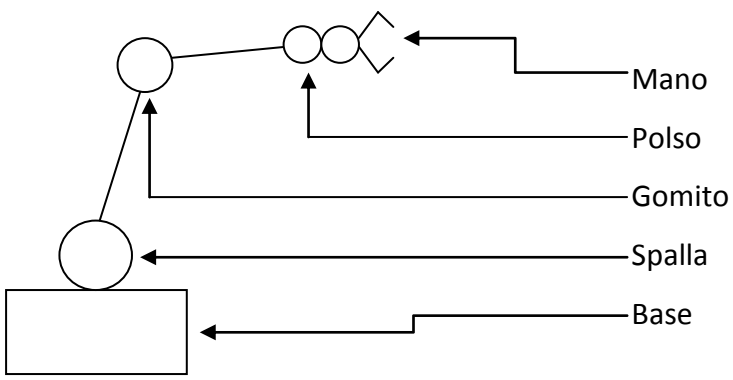
Il progetto nasce dall'idea di combinare meccanica, elettronica e informatica per costruire un braccio robotico a 3 assi con 2 gradi di libertà, capace di riprodurre in piccola scala quello che accade tutti i giorni nelle catene di montaggio. La spiegazione del progetto si dividerà in tre parti: la parte meccanica, la parte elettronica e infine la parte informatica. Le parti a loro volta saranno suddivise ulteriormente per una migliore comprensione del progetto.



Parte Meccanica

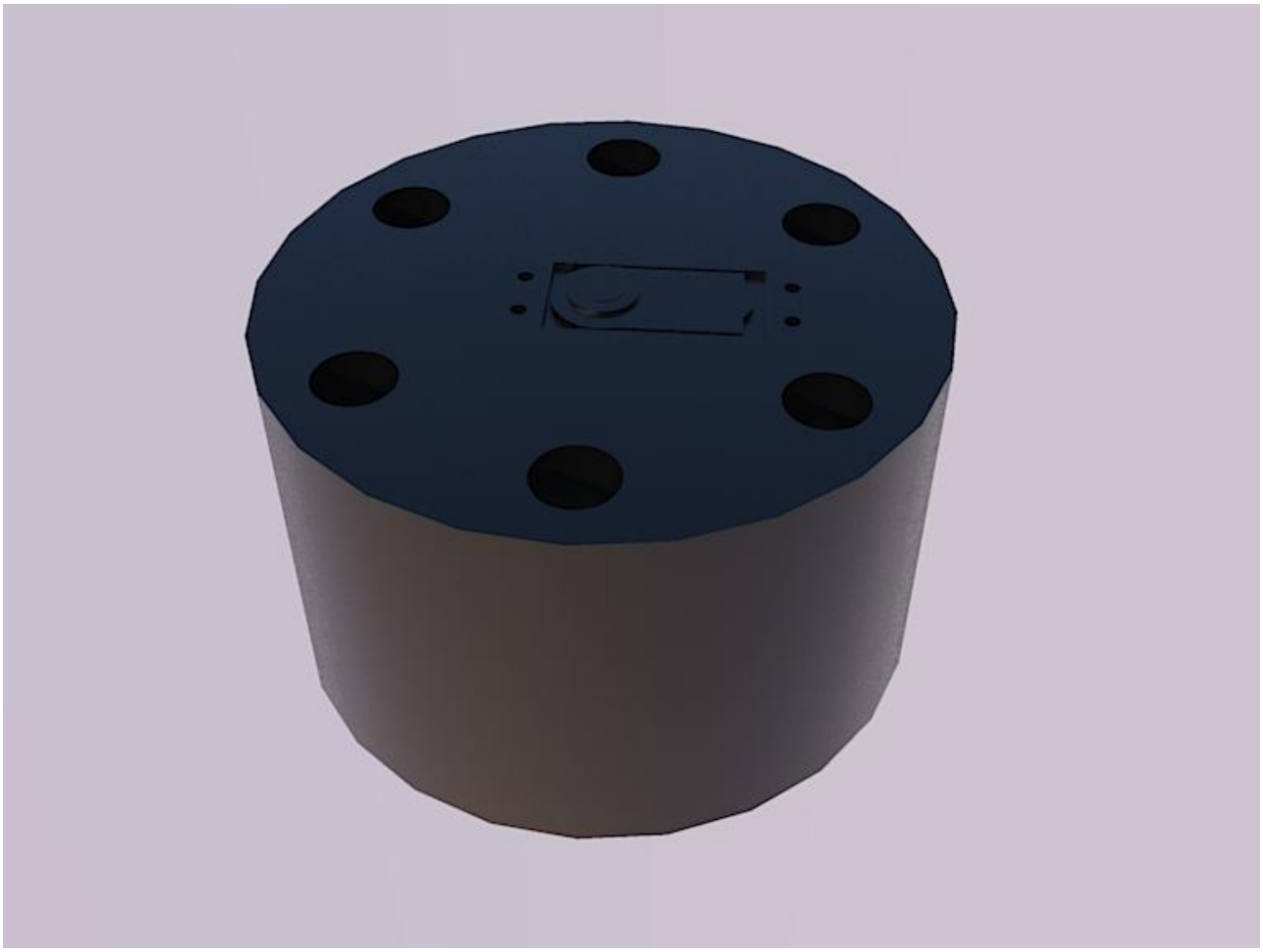
Inizialmente ho pensato alla parte meccanica, cominciando con alcuni schizzi a matita e a PC per poi riprodurre il tutto con materiali metallici e plastici.

Questa parte di progetto consiste nel fissare a dei bracci di metallo i servocomandi (mi occuperò di spiegare il funzionamento di questi nella parte di elettronica) e successivamente bloccare alla parte rotante dei motori un altro braccio per permettere la rotazione di quest'ultimo. Per il fissaggio della varie parti meccaniche sono stati utilizzati, quasi sempre, bulloni a testa esagonale e dadi M3. Per la realizzazione e la spiegazione della meccanica ho suddiviso il progetto in 5 parti, creando un analogia tra il braccio da realizzare e il braccio umano:



Base

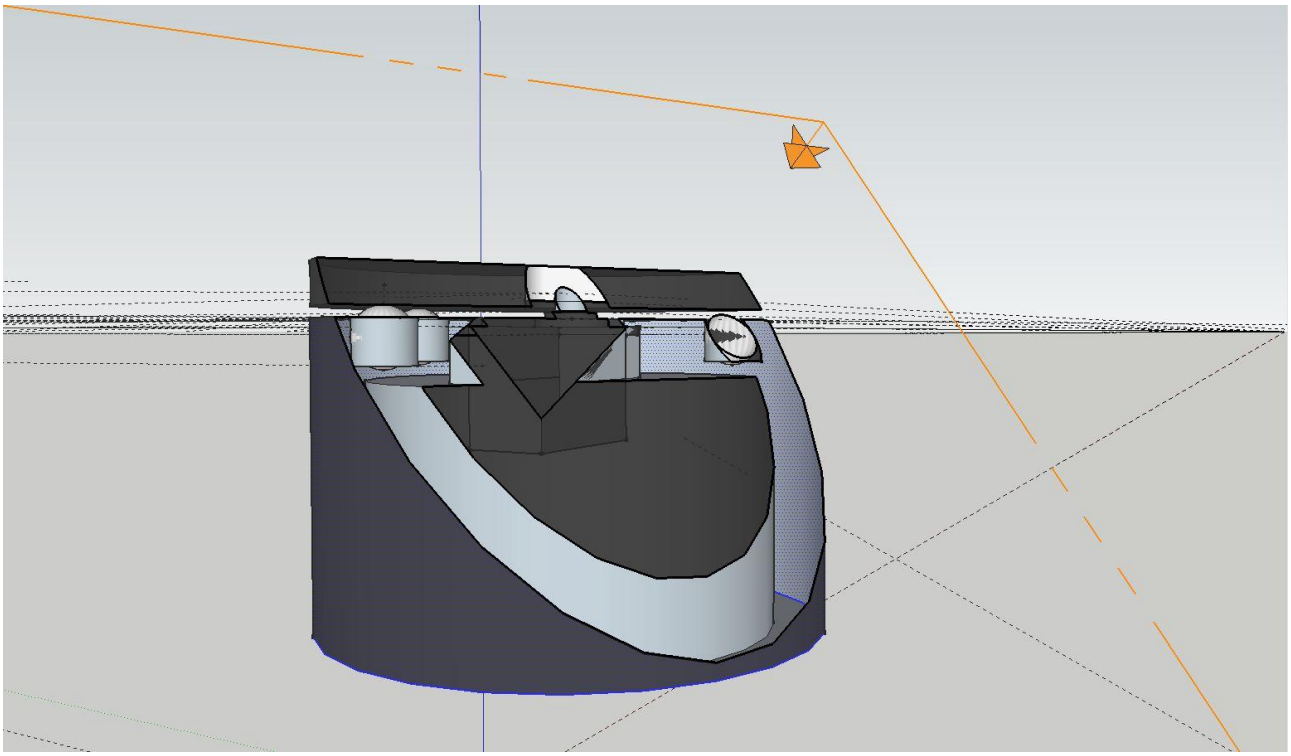
La base è stata realizzata da mio zio, spiegatogli il problema ha dato vita alle mie idee. Il sostegno è stato creato partendo da un cilindro di teflon nero successivamente incavato con il tornio per permettere l'inserimento del servo HS-475, e poi forato per far passare la parte rotante del motore al di fuori, al fine di agganciare un cilindro alto 10mm e di diametro 120mm (lo stesso della prima parte) alla testa della parte rotante. Tra i 2 pezzi di teflon sono state inserite, praticando dei fori 6 sfere di diametro 13 mm che permettono lo scorrimento dei 2 blocchi, riducendo gli attriti e la possibile rottura del servo non capace i sostenere l'intero peso del braccio.



Per riepilogare, la base è formata da 3 pezzi:

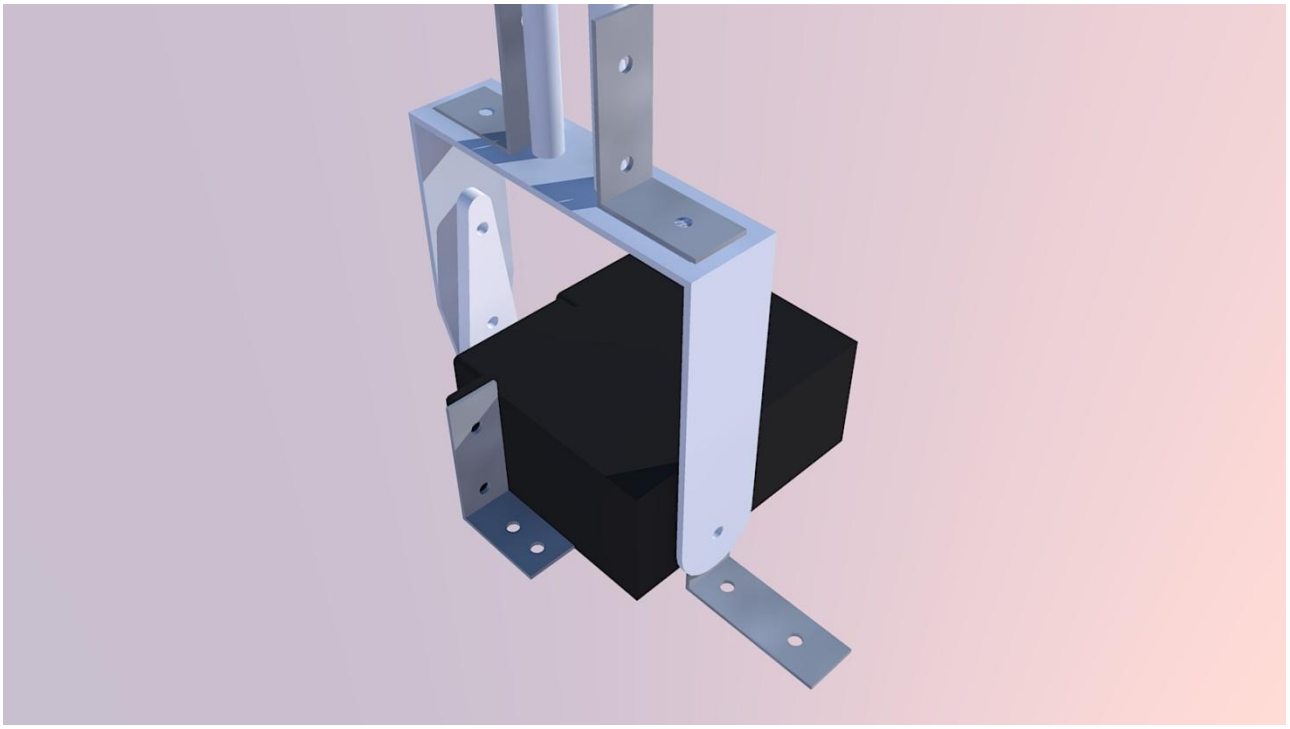
1) Un cilindro incavato (può essere pensato come la rotazione sull'asse centrale di una "U" squadrata) scavato e forato per permettere il fissaggio del servo.
2) Il servo con la parte esterna del rotore che presenta una testina a forma di croce.
3) Un piatto sempre dello stesso materiale con l'apposito scavo per l'aggancio della testa a X del motore.

In questa immagine, una sezione della base, si può notare la funzione delle sfere e la parte superiore dove verrà fissato il servo della spalla.



Spalla

La spalla è stata realizzata fissando il servo HS-805BB, quello con la coppia più grande, alla base con 2 squadrette con angolazione di piegatura 90°, e come scritto prima con bulloni M3.



Alla parte rotante del motore è stato applicato l'accessorio più lungo che permetteva in più facile aggancio alla U metallica. In linea con il centro della parte mobile del motore è stata fissata una L metallica forata allo stesso livello della U con un perno inserito al fine di avere un rotazione più lineare. Per realizzare la parte metallica da attaccare al servo (la cosiddetta U) è stato piegato un listello di metallo argentato.

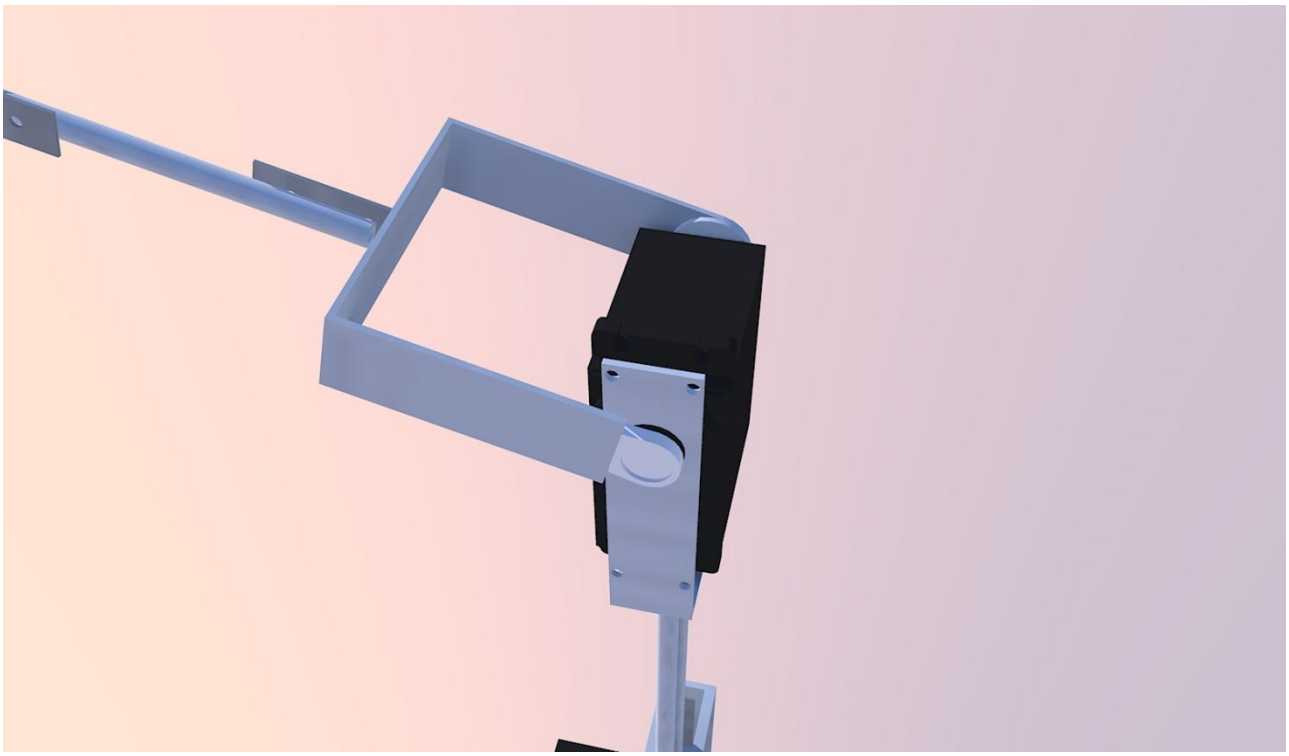
A questa U sono stati fissati sempre con delle squadrette, poiché è risultato il miglior modo per fissarle, non essere in possesso macchinari specializzati per minuterie meccaniche, due tubetti dello stesso materiale della lunghezza di 140 mm e diametro 7mm.

A capo di questi due prolungamenti è stata agganciata sempre allo stesso modo un'ulteriore U per agganciare il servo gomito del braccio.

Gomito

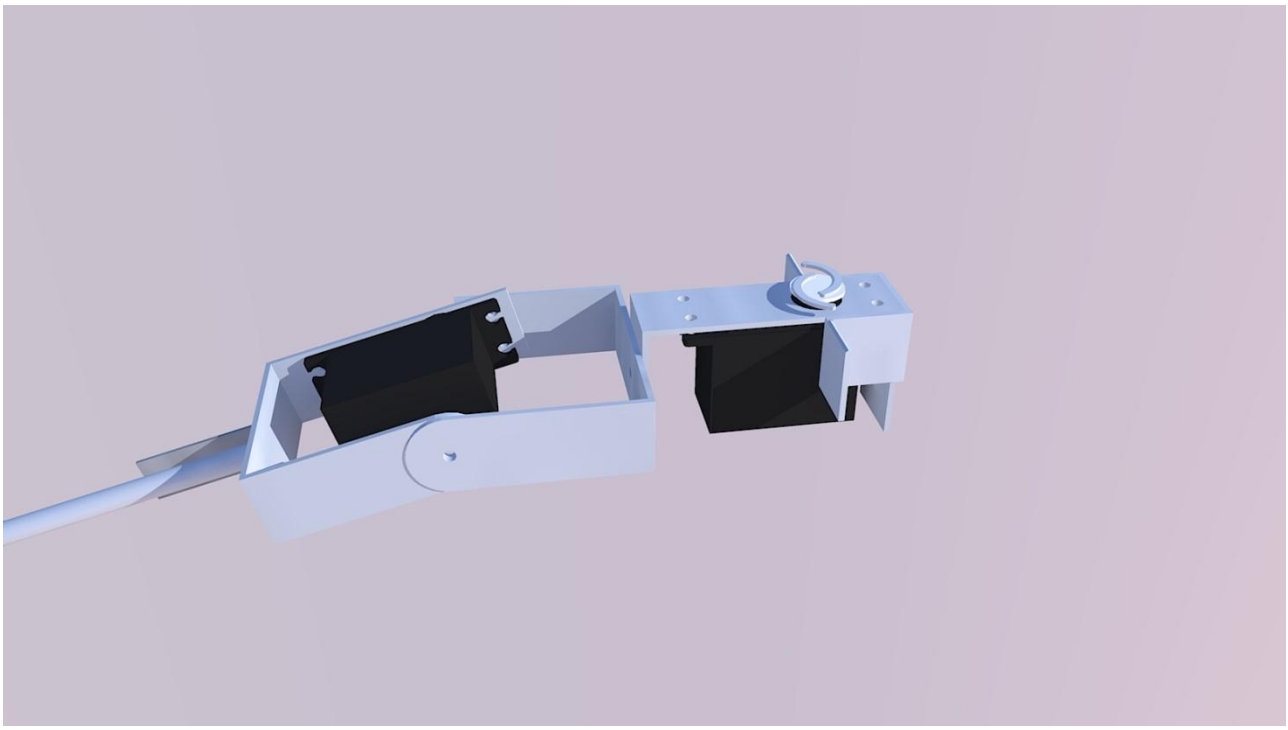
Una volta che è stato spiegata la meccanica della spalla, le altre parti risulteranno quasi uguali, quindi non mi soffermerò più di tanto spiegando in modo dettagliato come per la parte precedente.

Alla fine del prolungamento della spalla, alla U, è stato agganciato il servo motore HS-755HB. Utilizzando la medesima tecnica è stato effettuato il prolungamento dell'asse rotante del motore predisponendo il supporto per il "polso", unica differenza è l'utilizzo di un tubetto come supporto anziché due per ridurre il peso che dovrà essere alzato dai motori.



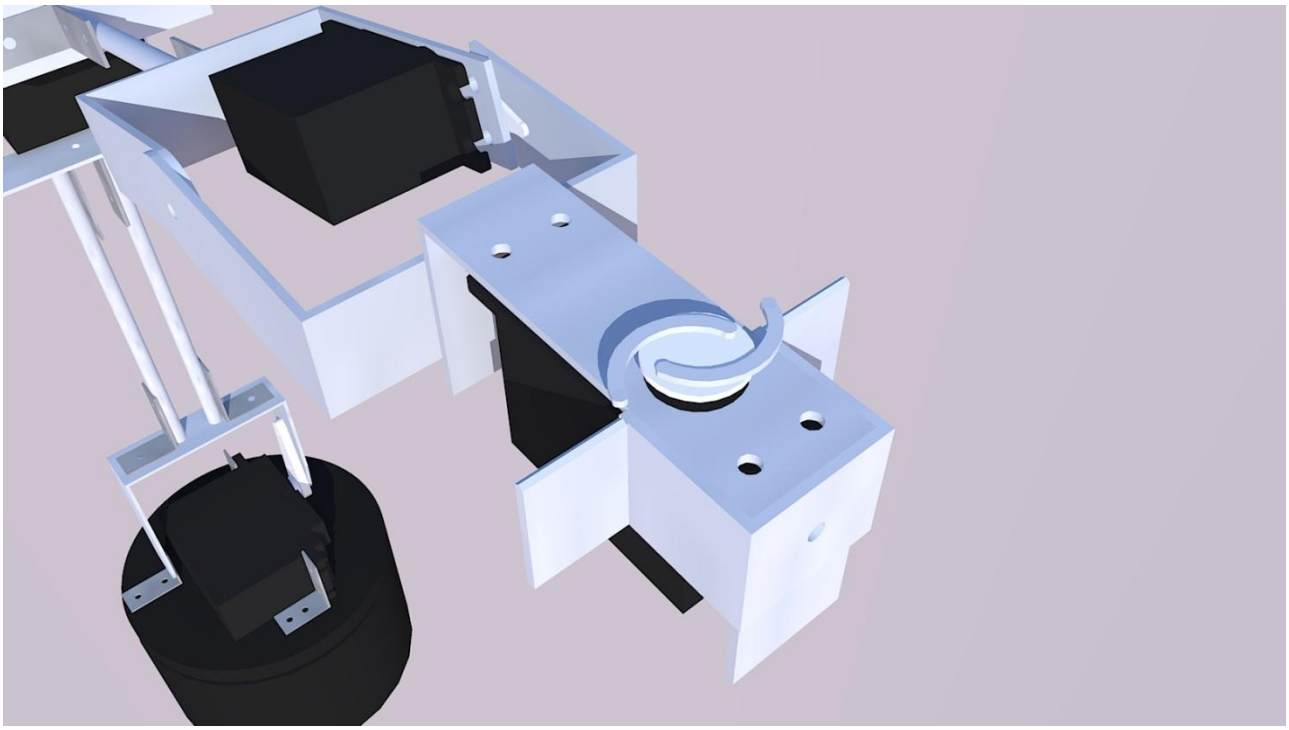
Polso

Medesima tecnica utilizzata precedentemente, anche in questa sezione l'unica differenza è il supporto di prolungamento, questa volta è stato subito attaccato il supporto successivo per la mano anche perchè la coppia del HS-475HB non è molto elevata.



Mano

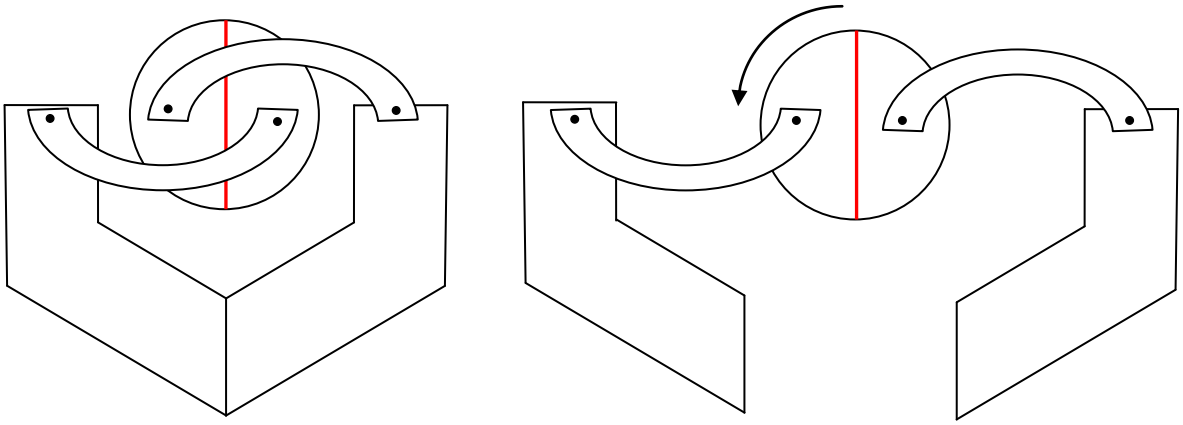
La mano del robot è stata pensata e progettata, ma per mancanza di tempo non realizzata. Qui sotto riporto il progetto della mano e in breve descriverò il suo funzionamento.



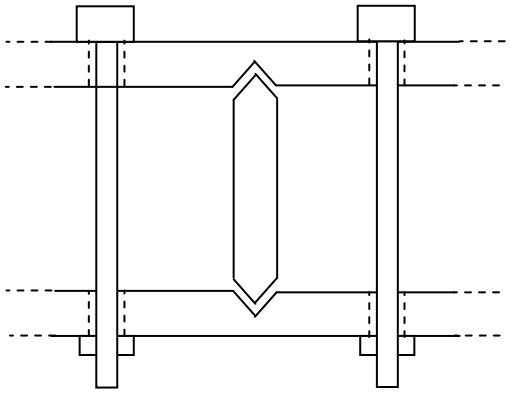
In sostanza la funzione della mano è di aprire e chiudere una pinza, per fare questo si utilizza un ulteriore servo motore posizionato però non rispetto lo stesso asse di rotazione degli altri motori, ma ruotato di 90°, anziché creare un piano y creerà un piano x.

Al rotore del servo motore HS-422, un servo motore con una coppia ridotta poiché la forza che deve esercitare è minima, sono stati fissati due piccoli archetti di metallo il loro obbiettivo è di tirare e spingere alla rotazione del motore. Nel disegno in 3d mancano i pezzi in metallo che dovrebbe spostare per una comprensione più facile del loro funzionamento.

Qui sotto ho voluto semplificare con 2 schemi il meccanismo (pinza aperta e chiusa).



I due pezzi di metallo che formano la mano sono fissati con dei bulloni ad altri 2 pezzi metallici identici, tra i quali è presente un ulteriore sbarretta di metallo, limata a 45° alle estremità, da entrambe le parti , che ha la funzione di binario.



Un messa a punto rilevante poteva essere l'implementazione di 2 sensori di forza posti all'interno della pinza. Il sensore in foto, qui affianco, è il: 1106 - Force Sensor della Phidgets. Questo sensore legge 0 quando nessuna forza e' applicata. Quando la forza, applicata sul bottone circolare, aumenta, il valore incrementa fino a 1000.
E' stato progettato come dispositivo di Input (es. per riconoscere quando

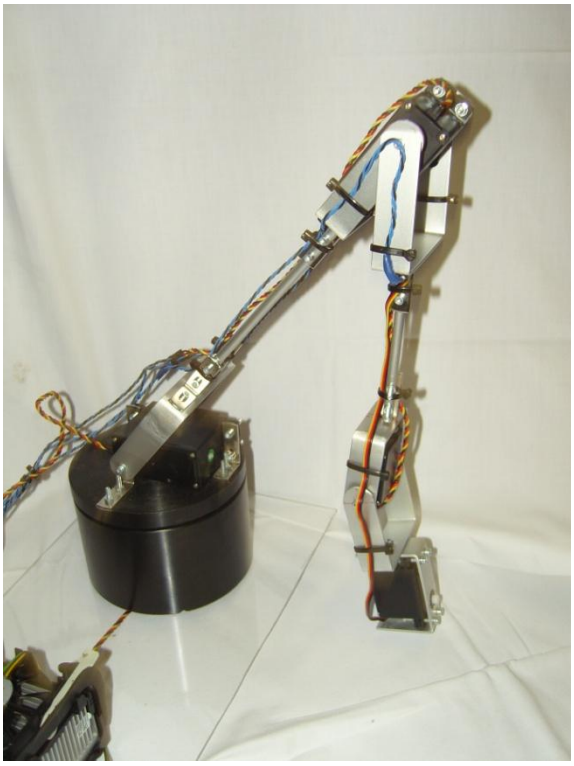
qualcuno spinge un bottone). Non e' accurato abbastanza per funzionare da dispositivo di misurazione, e non e' stato progettato per avere una forza applicata continuamente.

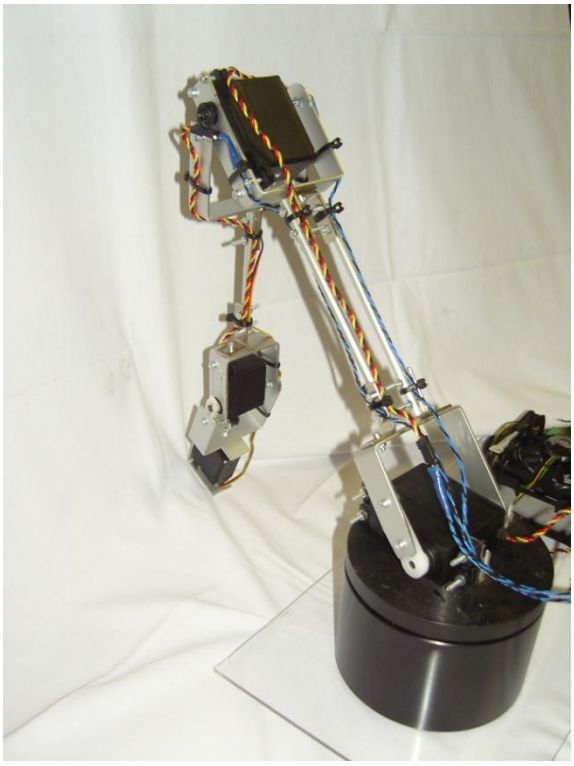
Note

Per la piegatura ho utilizzato la morsa e un martello con la testa in materiale plastico per evitare che un battente in ferro creasse ammaccature nel metallo. I fori nei listelli di metallo sono stati fatti con un trapano e una punta di diametro 3 mm, appoggiati ad un listello di legno perché è più facile non rovinare il materiale metallico. Per il taglio delle U e delle L metalliche, e per i bulloni eccessivamente lunghi è stato utilizzato un utensile della DREMEL, il DREMEL® Serie 300 (300-30), con in testa l'accessorio 456 (un disco da taglio rinforzato in fibra di vetro, vedi foto a lato). Per smussare e regolare il metallo è stato utilizzato sempre il medesimo strumento ma con un accessorio diverso, il 932 (una mollettina abrasiva all'ossido di alluminio, vedi foto a lato).

I cavi di alimentazione e di segnale di controllo dei servo sono stati allungati, se troppo corti, con delle prolunghe, create intrecciando dei cavi mediante l'utilizzo di un trapano e ponendo alle estremità i relativi connettori. Questi fili elettrici sono stati fissati alla parte meccanica con delle fascette, per rendere il robot più ordinato.

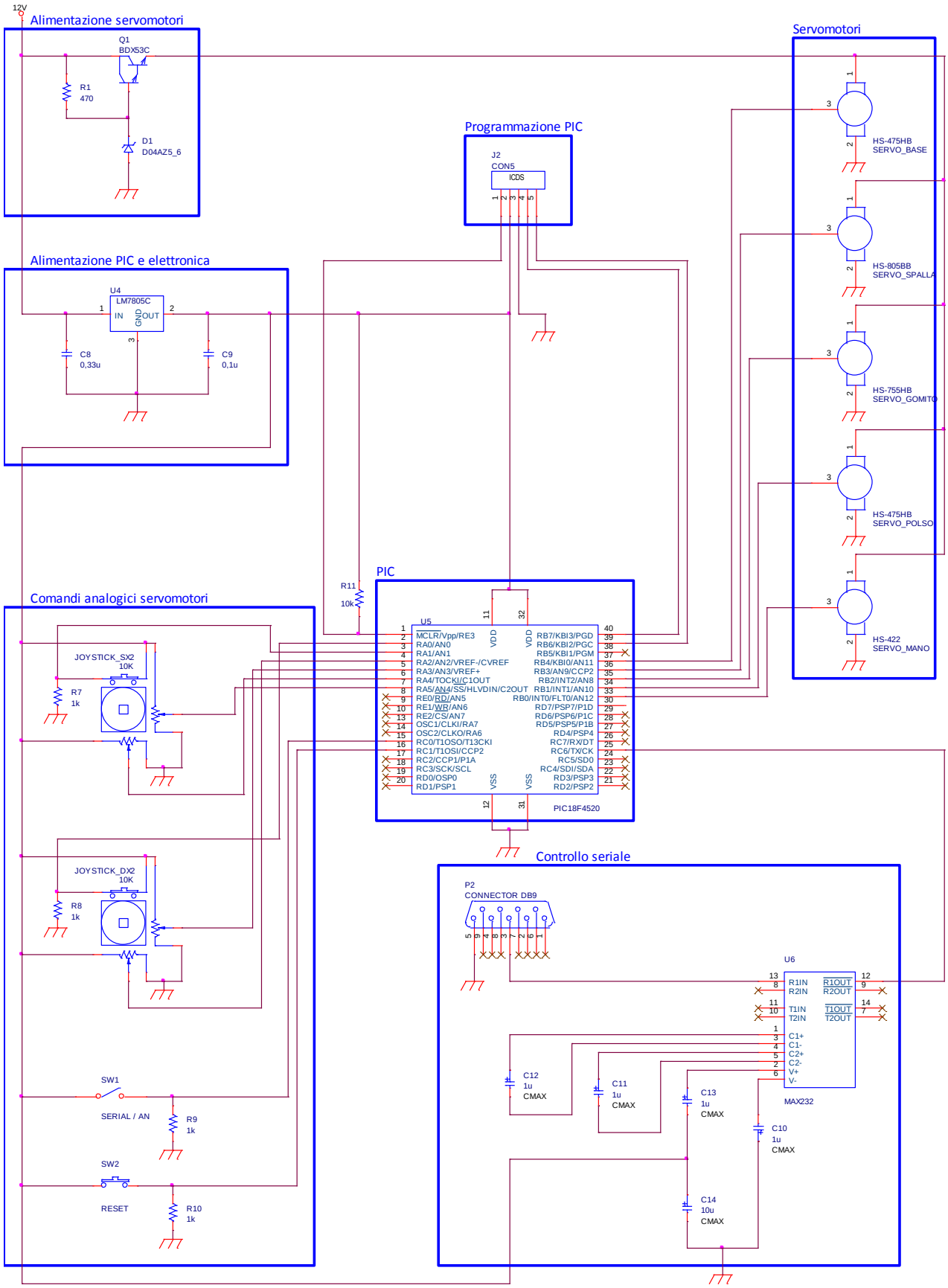
Foto



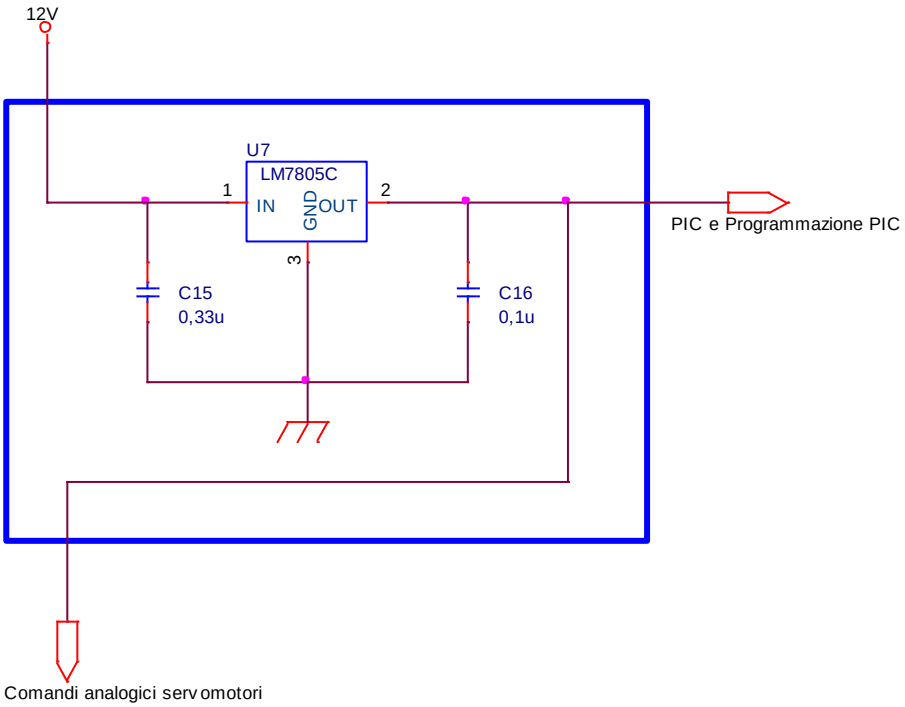


Parte Elettronica

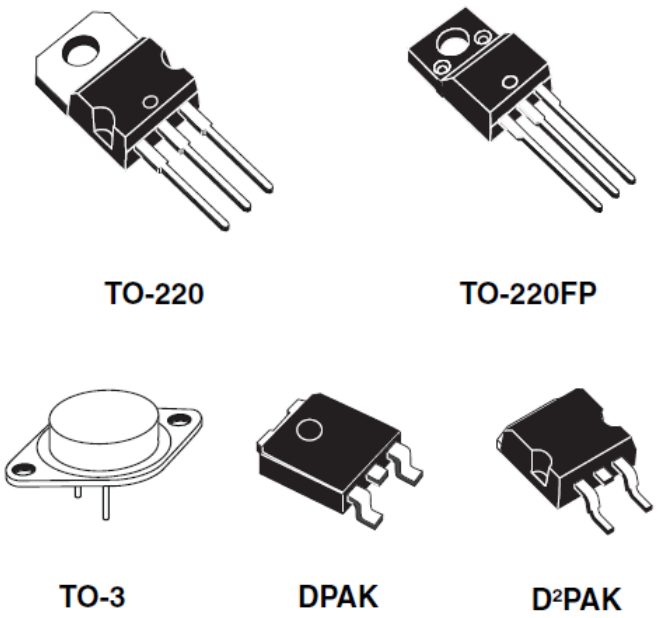
Il circuito:



Alimentazione PIC ed elettronica (7085)



L'alimentazione del microprocessore è stata stabilizzata a 5V, grazie all'integrato LM7805, uno stabilizzatore di tensione.



Gli stabilizzatori sono integrati che riescono a livellare delle tensioni sia positive che negative. Il nome stabilizzatore viene appunto dato a questo integrato perchè stabilizza la tensione ad un valore fisso. I piu comuni li possiamo trovare in case TO-220 e in TO-92 ma ci sono anche in case molto piu grandi, in grado di sopportare correnti maggiori e quindi dissipare potenze più elevate. La differenza di case è collegata con la corrente massima erogata. Gli stabilizzatori con case TO-220 sono quelli che al massimo erogano 1 A e sono predisposti per essere muniti di dissipatore. In caso contrario una protezione interna limita la corrente erogata a 0,5 A , ma per poco perchè perseverando si arriva alla distruzione dell'integrato. Invece quelli in case TO-92 hanno una corrente massima erogata di 0,1 A e vengono utilizzati spesso per stabilizzare tensioni di riferimento in applicazioni con operazionali ad esempio.

Gli stabilizzatori di tensioni positive 1A sono chiamati 78XX. Il 78 sta per tensioni positive, il numero che segue (xx sta per indicare un numero) è il valore della tensione di uscita espresso in volt (V). Ad esempio un 7812 è uno stabilizzatore da Max 1 A che stabilizza la tensione di uscita a 12 V. La versione da 100mA si chiama 78L12 .

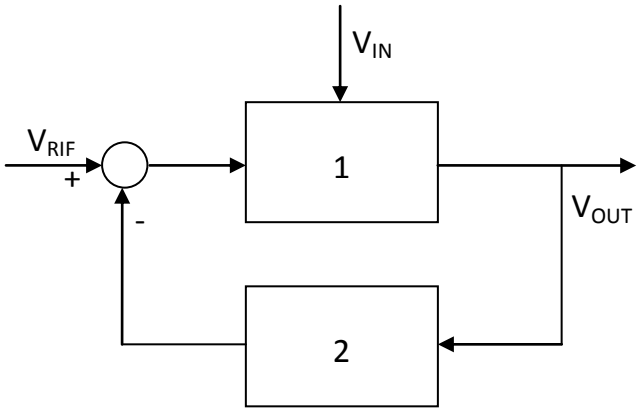
Gli stabilizzatori di tensioni negative 1A sono chiamati 79XX. Il 79 sta per tensioni negative, il numero che segue (xx sta per indicare un numero) è il valore della tensione di uscita espresso in volt (V). Ad esempio un 7912 è uno stabilizzatore da Max 1 A che stabilizza la tensione di uscita a 12 V negativi. La versione da 100mA si chiama 79L12.

Questi stabilizzatori di norma hanno una tensione d'ingresso massima ed una di uscita con la sua relativa tolleranza. Premettendo che ogni produttore ha dei valori, anche se poi sono quasi tutti simili, sono da rispettare per il funzionamento base del prodotto.

Perché il dispositivo funzioni occorre che la tensione di ingresso sia sempre superiore alla tensione di uscita. La differenza fra queste due tensioni, detta tensione di drop-out,

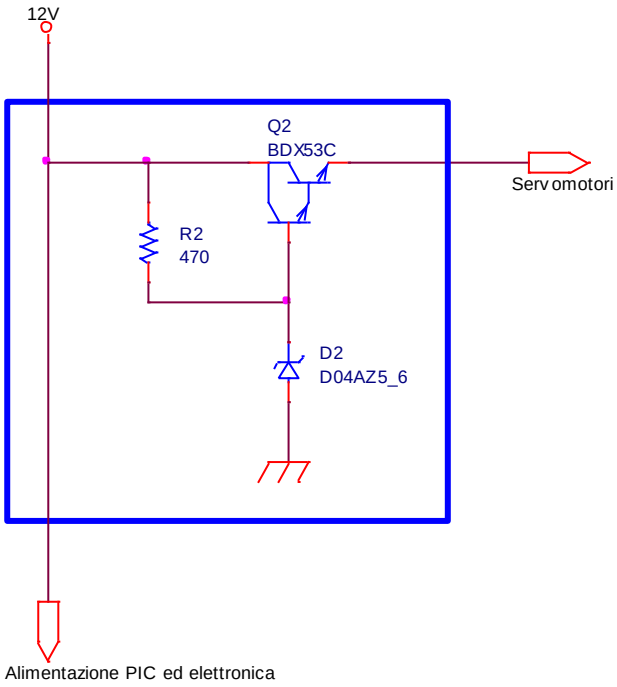
$$V_{DROP\ OUT} = V_{IN} - V_{OUT}$$

deve essere dunque positiva, ed almeno di 2 volt.
l dispositivo contiene un circuito complesso basato su BJT integrati. Che usa il concetto di retroazione, cioè, usando la logica degli schemi a blocchi.



In sostanza la tensione da stabilizzare è l'ingresso V_{IN} del nostro sistema, mentre la tensione stabilizzata è l'uscita V_{OUT}. Per mantenere la uscita stabile essa viene, attraverso il blocco 2, riportata in ingresso e confrontata con una tensione di riferimento V_{REF}. Se per qualche motivo la tensione di ingresso diminuisce, attraverso il blocco 1 diminuisce anche l'uscita , ma, fatta la differenza con la tensione di riferimento si ha un segnale positivo in ingresso al blocco 1 che controbilancia la diminuzione della tensione V_{IN}, il viceversa avviene se la tensione di ingresso tende ad aumentare. Come si può osservare, il sistema funziona anche se la tensione di uscita cambia non a causa della tensione di ingresso ma a causa di variazioni del carico.

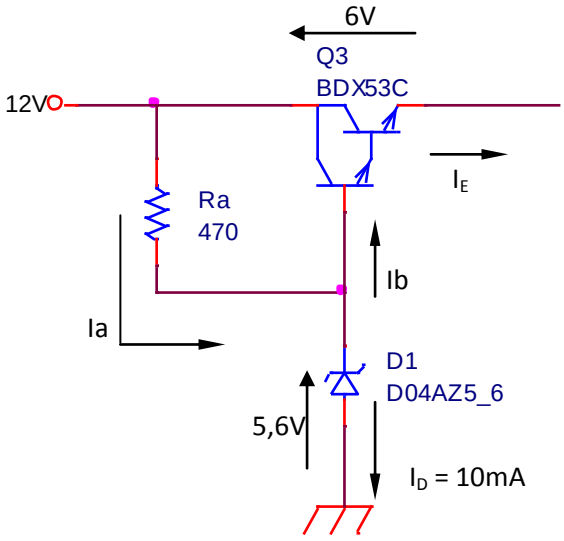
Alimentazione Servomotori



Per i servomotori è stato necessaria un'alimentazione differente da quella del PIC, poiché i i motori hanno bisogno di un'elevata corrente di funzionamento.

Per risolvere questo problema ho utilizzato un transistor BDX53C NPN darlington che presenta un hfe piuttosto elevato (750).

Per la progettazione di questa parte di circuito sono stati eseguiti i seguenti calcoli:



$$I_E = I_b \cdot h_{FE} \rightarrow I_b = \frac{I_E}{h_{FE}} = \frac{3}{750} = 4mA$$

$$I_A = I_b + I_D = 4mA + 10mA = 14mA$$

$$I_A = \frac{12V - V_{CE}}{R_a} \rightarrow R_a = \frac{12V - V_{CE}}{I_A} = 457\Omega[470\Omega]$$

Successivamente ho fatto i calcoli per la dissipazione dei componenti:

Dissipazione Resistenza: $W_{RA} = R_a \cdot I_A^2 = 470 \cdot (13{,}7 \cdot 10^{-3})^2 \cong 0{,}1W$

Dissipazione Diodo: $W_{DZ} = I_A \cdot V_D = 14 \cdot 10^{-3} \cdot 5{,}6 = 0{,}08W$

Dissipazione Transistor: $W_T = I_E \cdot V_{CE} = 3 \cdot 6 = 18W$

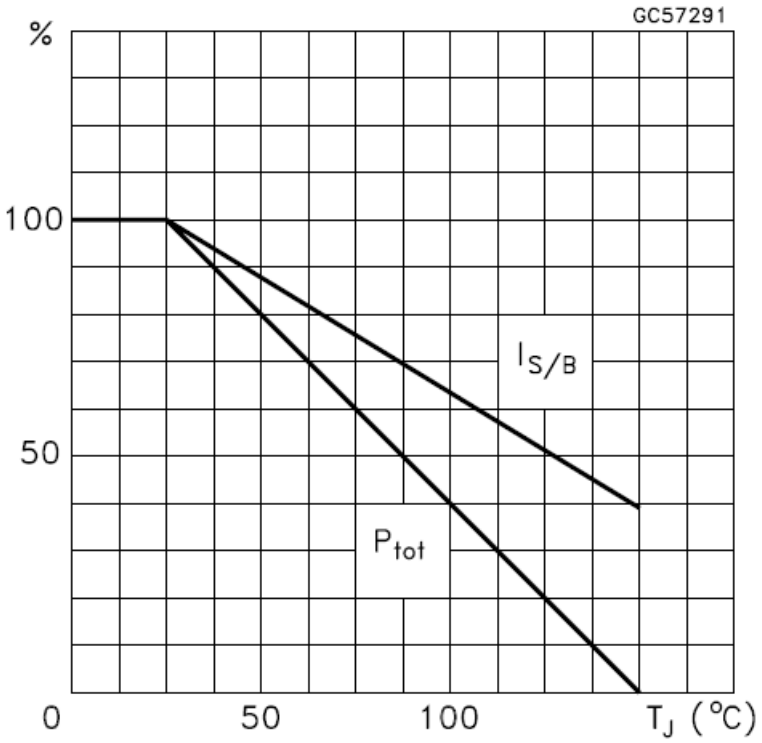
Dissipatore per il transistor:

Nei componenti passivi (resistori) e a semiconduttore (sia discreti che integrati) il passaggio della corrente elettrica provoca - per effetto Joule - un innalzamento della temperatura dei dispositivi, che può essere calcolato attraverso la seguente relazione:

(1) Tw - Ta = Pd · Rth

dove Tw è la temperatura raggiunta dal dispositivo a causa della dissipazione della potenza applicata Pd, Ta è la temperatura dell'ambiente circostante ed Rth è una costante che viene chiamata "resistenza termica", che rappresenta la difficoltà di smaltimento del calore verso l'ambiente circostante ed è l'inverso della "conduttanza termica" dei materiali e dei fluidi che smaltiscono il calore prodotto. Dalla (1) si ricava che l'unità di misura della resistenza termica è il grado/Watt (°C/W).
Poiché il costruttore del dispositivo indica nel foglio tecnico una temperatura massima di funzionamento Tmax, è possibile calcolare la massima potenza Pmax dissipabile dal dispositivo stesso ad una temperatura ambiente Tamax sfruttando la relazione (1), e riscrivendola nel modo seguente:

(2) Tmax - Tamax = Pmax · Rth



La relazione ora vista ci spiega inoltre perché, una volta raggiunta la massima temperatura specificata, un'ulteriore incremento della potenza applicata porterebbe ad un ulteriore aumento della temperatura, con conseguente possibile danneggiamento del dispositivo. Dovrebbe essere chiaro, quindi, che i limiti di potenza specificati per un certo dispositivo dipendono strettamente dai limiti di temperatura dello stesso.

← curva di derating del BDX53C

Come si vede dalla curva di derating, è possibile dissipare la potenza nominale fino ad una temperatura ambiente di 25 °C, superata la quale è necessario ridurre progressivamente la potenza dissipata fino a che - raggiunta la temperatura massima di lavoro di 150 °C - la potenza dissipabile diviene nulla come già osservato a proposito della relazione (2).

Calcoli dissipatore:

$$P_D = I_E \cdot V_{CE} = 3 \cdot 6 = 18W$$

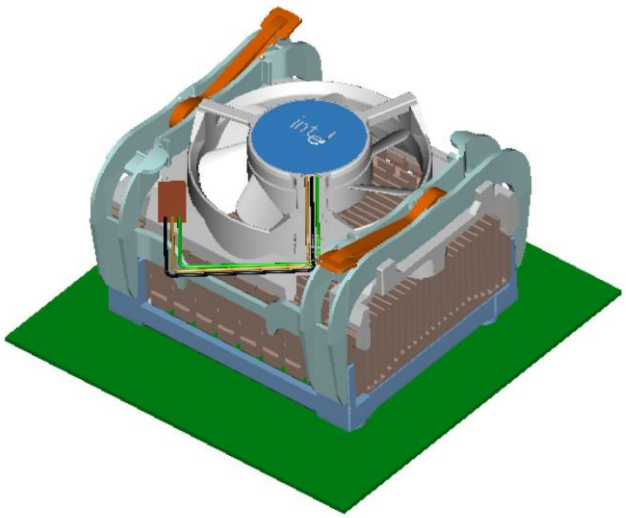
$$T_{jMAX} - T_{aMAX} = P_{DMAX} \cdot (R_{tjc} + R_{thcd} + R_{thda})$$

$$R_{thda} = \frac{T_{jMAX} - T_{aMAX}}{P_{DMAX}} - R_{tjc} - R_{thcd} = \frac{150 - 50}{18} - 2{,}08 - 1 = 2{,}48\ C°/W$$

Essendo questo il valore massimo ammesso, occorre scegliere sul catalogo un dissipatore con una resistenza termica inferiore ai 2,48 °C/W.

Avendo disponibile un dissipatore smontato da una vecchia CPU intel, ho preferito optare per l'utilizzo dello stesso che presenta una notevole capacità di dissipazione del calore.

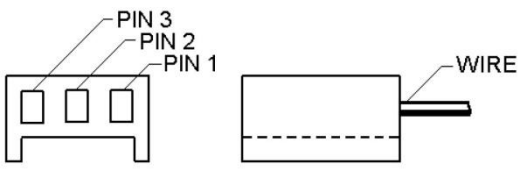
$$\Psi_{CA} = \frac{T_{CMAX}\,^{\circ}C - T_{LA}\,^{\circ}C}{TDP(W)} = \frac{69^{\circ}C - 45^{\circ}C}{54.3W} = 0.442\frac{^{\circ}C}{W}$$



Il dissipatore in questione presenta nella sommità una ventole con le seguenti caratteristiche:

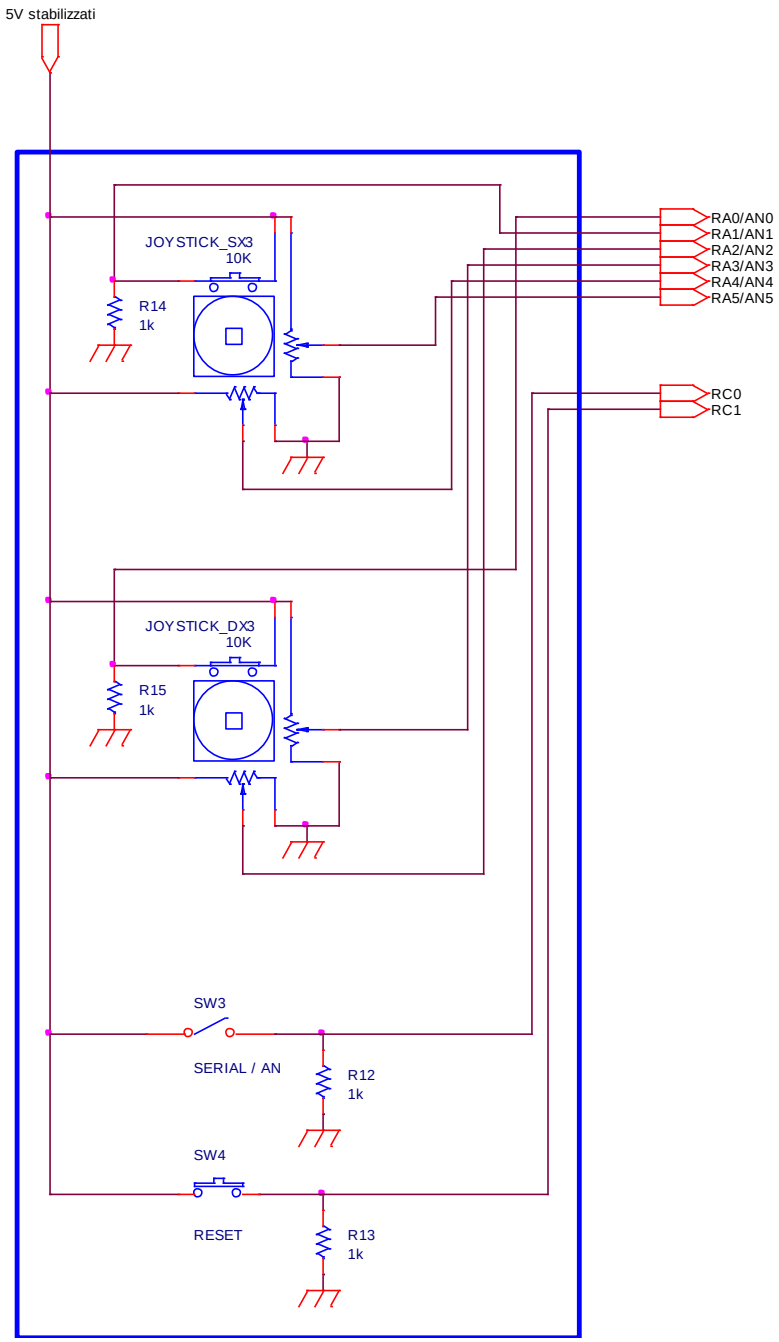
Requisiti elettrici:
- *Minimo: 9 V*
- *Tipico: 12 V*
- *Massimo: 13.8 V*
- *Massimo assorbimento di corrente all'avviamento della ventola (IC): 740 mA*

Fili di collegamento della ventola:



Pin1: Massa, filo nero.
Pin2: Alimentazione +12V, filo giallo
Pin3: Uscita tachimetrica della ventola, 2 impulsi a giro, filo verde

Comandi analogici servomotori



Le porte del pic sono state impostate nel seguente modo:

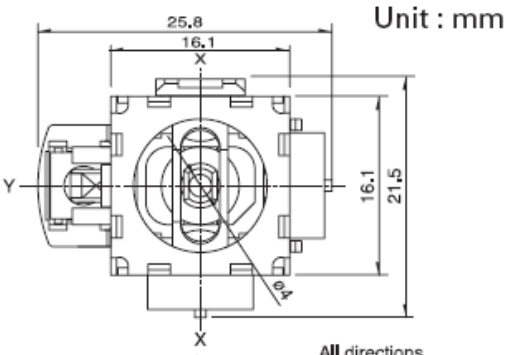
porte **A** [0; 3] – 4 ingressi analogici (joysick) (0-5V).

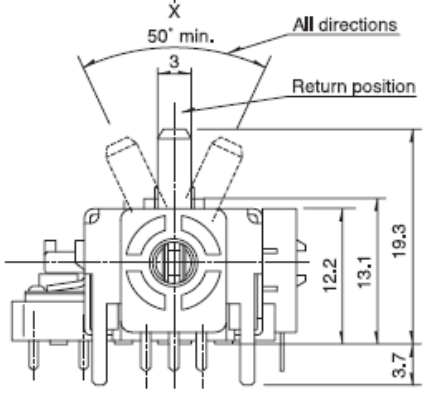
[4; 5] – 2 ingressi digitali (apri, chiudi pinza).

porte **C** [0; 1] – le porte 0 e 1 sono controlli digitali che in ordine servono per impostare il reset del braccio e l'ingresso di controllo seriale o analogico

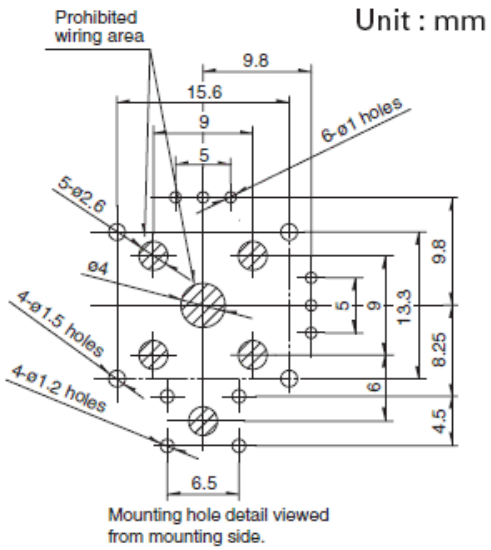
Alle porte A impostate come ingressi sono stati collegati 2 joystick, ricavati da un vecchio joypad della playstation 2, i componenti presentano queste specifiche:

Type		Variable resistor
Number of operating shafts		Single-shaft
Shaft material		Resin
Lever return mechanism		With
Potentiometer Part	Maximum operating voltage	50V AC, 5V DC
	Resistance taper	B (0B)
	Total resistance	10kΩ
Center push part	Center push	With
	Ratings (max.)	50mA 12V DC
	Travel (mm)	0.5(+0.5, -0.4)
Minimum order unit (pcs.)		1,400



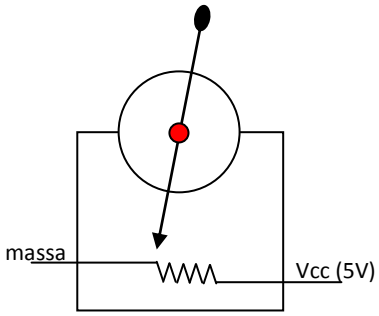
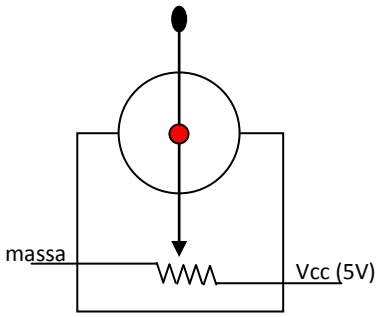
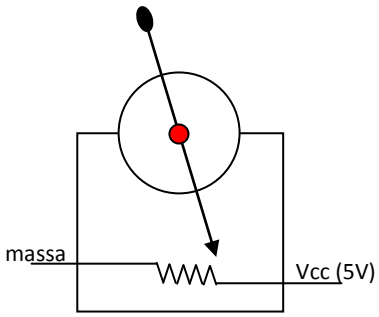


Operating temperature range			-10℃ to +70℃
Electrical performance	Directional resolution		Continuous
	Insulation resistance		100MΩ min. 250V DC
	Voltage proof		250V AC for 1 minute
	Rated power		0.0125W
Mechanical performance	Directional operating force		17±10mN·m
	Push operating force		6.4(+3.4, -2.6)N
Environmental performance	Cold		-30±2℃ for 96h
	Dry heat		80±2℃ for 96h
	Damp heat		60±2℃, 90 to 95%RH for 96h
Mechanical performance	Solder heat resistance	Manual soldering	300℃ max. 3s max.
		Dip soldering	260±5℃, 5±1s
Slider noise			300mV max. by JIS method
Lever play			5° max.
Lever strength			98N min. (Push), 50N min. (Pull)
Durability	Operating life	Directions	2,000,000 cycles
		Center push	1,000,000 cycles



Il loro funzionamento è di facile comprensione, i joystick sono dei resistori variabili (10KΩ), se sottoposti a tensione, nel nostro caso 5V ai capi dei resistori presentano un potenziale proporzionale al valore della resistenza.

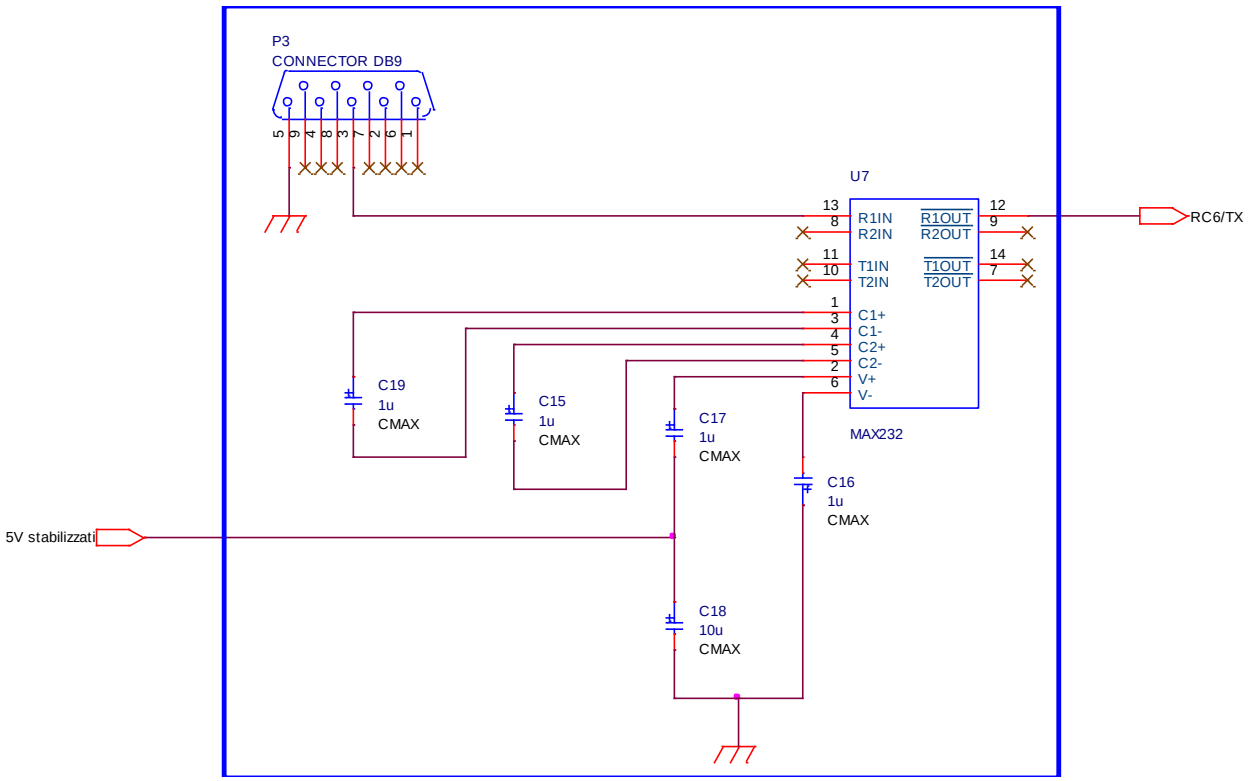
Per esemplificare il loro funzionamento nella seguente tabella viene descritta la funzione del movimento orizzontale della levetta del joystick :

Posizione	Immagine	Valore resistenza	Tensione
destra		10K	0 V
centrale		5K	2,5 V
sinistra		0K	5 V

Le porte B uscite digitali, il loro comportamento verrà spiegato nelle pagine successive quando tratterò dei servomotori.

Le porte C, ovvero C0,C1 sono utilizzate in questo modo: C0 è collegata a un dip-swich che permette la selezione della modalità di controllo analogica o seriale, se a 0 è impostato il controllo con i joystick, se a 1 tramite la porta seriale del PC; C1 è connessa a un push-button che se posto alto (1) il braccio si posizionerà in posizione, scusate il gioco di parole, standard;

Controllo seriale



Questa parte di circuito permette il controllo del braccio robotizzato tramite un programma in Visual Basic, che invierà tramite collegamento seriale dei dati (impulsi di 1 e 0) al max232, il quale convertirà il segnale perché sia leggibile dal PIC.

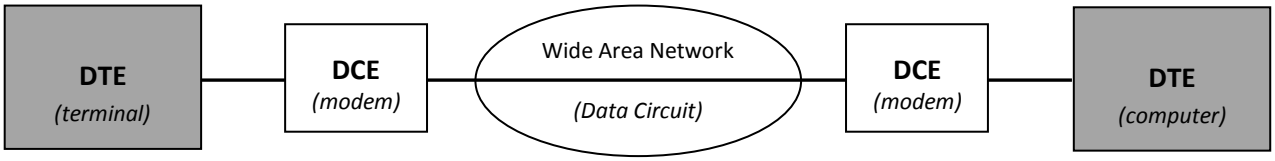
L'integrato **U7**, un **MAX232** prodotto dalla **Maxim**, si occupa di convertire i segnali RS232 dai +/-12 volt necessari per trasmettere e ricevere dati sulla porta seriale ai 0/5 volt TTL gestibili direttamente dalle porte del PIC.

Ma vediamo in dettaglio come funziona la comunicazione seriale in RS232.

Cos'é e a cosa serve l'RS232

Lo standard RS232 definisce una serie di specifiche per la trasmissione seriale di dati tra due dispositivi denominati **DTE** (Data Terminal Equipment) e **DCE** (Data Communication Equipment). Come si può vagamente intuire dal nome, il Data Communication Equipment è un dispositivo che si occupa di gestire una comunicazione dati mentre il Data Terminal Equipment è un dispositivo che si occupa di generare o ricevere dati. In pratica l'RS232 è stata creata per connettere tra loro un terminale dati (nel nostro caso un computer) con un modem per la trasmissione a distanza dei dati generati. Per avere una connessione tra due computer è quindi necessario disporre di quattro dispositivi come visibile in figura (pagina sucessiva): un computer (DTE) collegato al suo modem (DCE) ed un altro modem (DCE) collegato al suo computer (DTE). In questo modo qualsiasi dato generato dal primo computer e trasmesso tramite RS232 al relativo modem verrà trasmesso da

questo al modem remoto che a sua volta provvederà ad inviarlo al suo computer tramite RS232. Lo stesso vale per il percorso a ritroso.



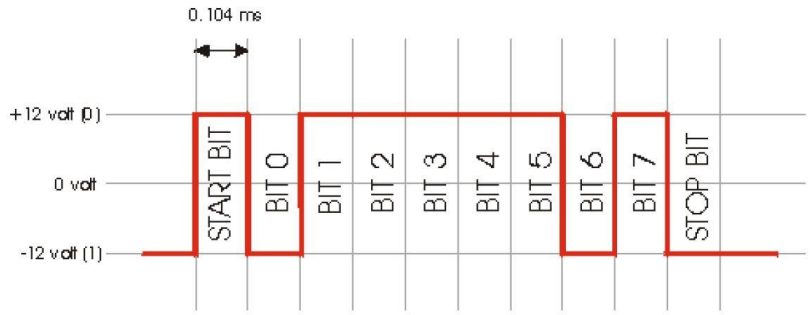
Per usare la RS232 per collegare tra loro due computer vicini senza interporre tra loro alcun modem dobbiamo simulare in qualche modo le connessioni intermedie realizzando un cavo **NULL MODEM** o cavo invertente, ovvero un cavo in grado di far scambiare direttamente tra loro i segnali dai due DTE come se tra loro ci fossero effettivamente i DCE.

Per connettere il PC al nostro circuito simuleremo invece direttamente un DCE facendo credere al PC di essere collegato ad un modem. Prima di fare questo diamo uno sguardo in dettaglio al principio di funzionamento di una comunicazione seriale.

La comunicazione seriale asincrona

Per consentire la trasmissione di dati tra il PC ed il modem, lo standard RS232 definisce una serie di specifiche elettriche e meccaniche. Una di queste riguarda il tipo di comunicazione seriale che si vuole implementare che può essere sincrona o asincrona.
Nel nostro caso analizzeremo solo la comunicazione seriale asincrona ignorando completamente quella sincrona in quanto più complessa e non disponibile sui normali PC.
Una comunicazione seriale consiste in genere nella trasmissione e ricezione di dati da un punto ad un altro usando una sola linea elettrica. In pratica se desideriamo trasmettere un intero byte dobbiamo prendere ogni singolo bit in esso contenuto ed inviarlo in sequenza sulla stessa linea elettrica, un pò come avviene per la trasmissione in codice morse. La differenza sostanziale stà nel fatto che a generare e ricevere dati non c'é il telegrafista ma un computer per cui le velocità di trasmissione raggiungibili sono molto superiori. Facciamo subito un esempio pratico e vediamo come fa un PC a trasmettere, ad esempio, il carattere **A** usando la RS232. Non è necessario ovviamente realizzare gli esempi riportati di seguito in quanto presuppongono l'uso di una coppia di PC ed un oscilloscopio non sempre disponibili nei nostri mini-laboratori da hobbysta. Per comprendere il funzionamento di quanto esposto è sufficiente fare riferimento alle figure a corredo.
Se prendiamo una coppia di fili e colleghiamo tra loro le porte seriali di due PC (che denomineremo PC trasmittente e PC ricevente) otterremo la più semplice delle connessioni in RS232.
La linea **Transmit Data** (TXD) presente sul pin 3 del connettore DB9 maschio di cui il vostro PC è dotato è connessa alla linea **Receive Data** (RXD) presente sul pin 2 del secondo PC. Le masse (GND) presenti sul pin 5 di entrambe i PC sono connesse tra loro.
Per osservare i segnali generati dal PC trasmittente durante la trasmissione seriale colleghiamo tra la linea TXD e la linea GND un oscilloscopio e lanciamo in esecuzione su entrambe i PC un

programma di emulazione terminale (tipo Hyperterminal o simili). Configuriamo le porte seriali di entrambe i PC a 9600 baud, 8 data bit, 1 stop bit, no parity e disabilitiamo il controllo di flusso (handshake) sia hardware che xon/xoff. In questo stato qualsiasi cosa digiteremo sul PC trasmittente verrà inviata immediatamente sulla porta seriale. Assicuriamoci inoltre che il programma di emulazione terminale scelto sia opportunamente configurato per usare la porta seriale su cui siamo connessi (COM1 o COM2). Proviamo a digitare la lettera **A** maiuscola e verifichiamo se è stata correttamente ricevuta sul PC ricevente. Fatto questo controllo andiamo a vedere sull'oscilloscopio che tipo di segnali sono stati generati per effettuare la trasmissione.
Quando non c'é nessuna trasmissione in corso la tensione sulla linea TXD è di -12 volt corrispondente alla condizione logica 1. Per indicare al PC ricevente che la trasmissione ha inizio, il PC trasmittente porta a +12 volt la linea TXD per un tempo pari all'inverso della frequenza di trasmissione ovvero al tempo di trasmissione di un singolo bit. Nel nostro caso, avendo scelto di trasmettere a 9600 bit per secondo, la tensione di alimentazione rimarà a +12 volt per: 1/9600=0.104 mS. Questo segnale viene denominato **START BIT** ed è sempre presente all'inizio di trasmissione di ogni singolo byte. Dopo lo start bit vengono trasmessi in sequenza gli otto bit componenti il codice ASCII del carattere trasmesso partendo dal bit meno significativo. Nel nostro caso la lettera A maiuscola corrisponde al valore binario **01000001** per cui la
sequenza di trasmissione sarà la seguente:



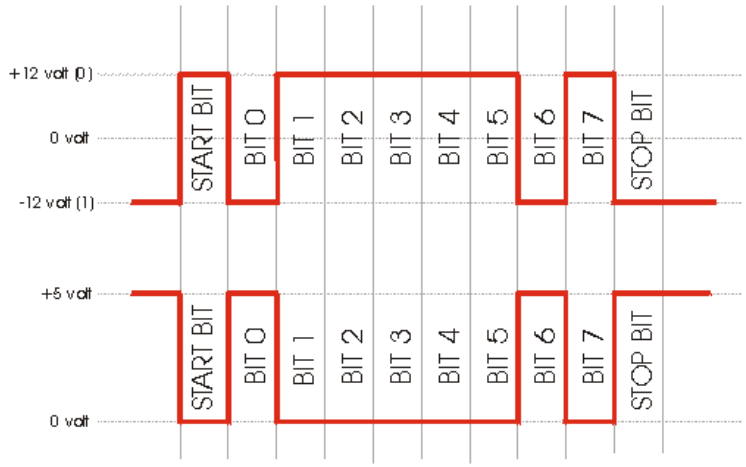
Una volta trasmesso l'ottavo bit (bit 7), il PC aggiunge automaticamente un ultimo bit a 1 denominato **STOP BIT** ad indicare l'avvenuta trasmissione dell'intero byte. La stessa sequenza viene ripetuta per ogni byte trasmesso sulla linea. Aggiungendo al nostro cavo seriale una connessione tra il pin TXD (pin 3) del PC ricevente con il pin RXD (pin 2) del PC trasmittente, potremo effettuare una trasmissione RS232 bidirezionale. Il cavo che abbiamo ottenuto è il più semplice cavo NULL MODEM in grado di mettere in collegamento tra loro due DTE.

Come collegare il nostro circuito

Come accennato prima il nostro circuito simula un dispositivo DCE. Questo significa che il cavo che dovremo realizzare non dovrà essere di tipo NULL MODEM o INVERTENTE ma **DRITTO** ovvero con i pin numerati allo stesso modo connessi tra loro. Questo tipo di cavo è identico a quelli che vengono usati per connettere al PC un modem esterno.
Dato che i dispositivi DTE sono sempre dotati di connettore DB9 maschio, il nostro circuito, essendo un DCE, avrà un connettore DB9 femmina. In alcuni casi i PC sono dotati di connettori

DB25 anziché DB9 per cui per le equivalenze occorre consultare la piedinatura dei connettori RS232.
Il cavo di collegamento tra il PC ed il nostro circuito dovrà essere intestato a sua volta con un connettore femmina da un lato per poter essere inserito nella seriale del PC ed un connettore maschio dall'altro per poter essere inserito nel connettore del nostro circuito.

Funzionamento del MAX232
Come accennato prima, nel nostro circuito d'esempio useremo un driver RS232, ovvero un integrato in grado di convertire i segnali a +/- 12 volt tipici della RS232 in segnali a 0/5 volt gestibili dalle porte del PIC.
Il segnale di trasmissione proveniente dal PC entra dal pin 3 del connettore DB9 femmina sul pin 13 di U3. Sul pin 12 di U3 sarà presente un segnale a 0 volt quando sul pin 13 ci saranno +12 volt e 5 volt quando sul pin 13 ci saranno -12 volt.
Sul pin 18 del PIC (RA1) avremo quindi la seguente corrispondenza di segnali con la linea TXD del PC.



Viceversa sul pin 17 (RA0) il PIC genera i segnali da inviare al PC a livello TTL che vengono convertiti in segnali RS232 da U3 tramine i pin 11 (ingresso TTL) e 14 (uscita RS232) e quindi inviati al PC tramite il pin 2 del connettore SERIALE.

La seconda parte descritta, ovvero la trasmissione dal PIC al PC non è stata utilizzata poiché non sono necessarie risposte di effettivo arrivo del comando, perché il comando dal PC al PIC viene inviato in modo continuo. Ad ogni aggiornamento del programma.
I collegamenti per lo schema sono stati fatti seguendo gli schemi tipici del costruttore.

Per mancanza di tempo dovuta a problemi riscontrati durante la stesura del listato del programma della parte analogica non è stato possibile realizzare la parte di programma relativa al controllo da PC, mi sono soffermato soprattutto sul controllo analogico.
Per la spiegazione di questa parte di circuito mi sono servito del corso Pix By Example del professore Tanzilli, adattata al mio circuito.

Servomotori

I servomotori (o servo) sono dei dispositivi molto utili per chi si diletta nella costruzione di robot e più in generale per tutte le realizzazioni che uniscono meccanica ed elettronica (spesso chiamata meccatronica o cibernetica). Si tratta in effetti di motori di precisione controllabili elettronicamente. Vediamo in questo articolo come funzionano e come fare per controllarli tramite microcontrollori.

Cosa sono i servomotori

In un piccolissimo contenitore racchiudono un motore, un riduttore ed il circuito che lo controlla. Il riduttore serve per aumentare la potenza disponibile. Il circuito permette di controllare la posizione del servo tramite impulsi inviati su un singolo filo.
I servo hanno infatti tre soli fili di collegamento. Si tratta di positivo e negativo di alimentazione e del filo di controllo (spesso erroneamente indicato come PWM).
Il rotore che esce dal corpo del servo (che può essere di plastica o metallico) ha collegata una croce che permette di utilizzarlo nelle costruzioni. Hanno solitamente liberta di movimento limitata a 180 gradi (mezzo giro). Smontandoli è possibile modificarli per eliminare questa limitazione e renderlo libero di girare 360 gradi come un motore normale.

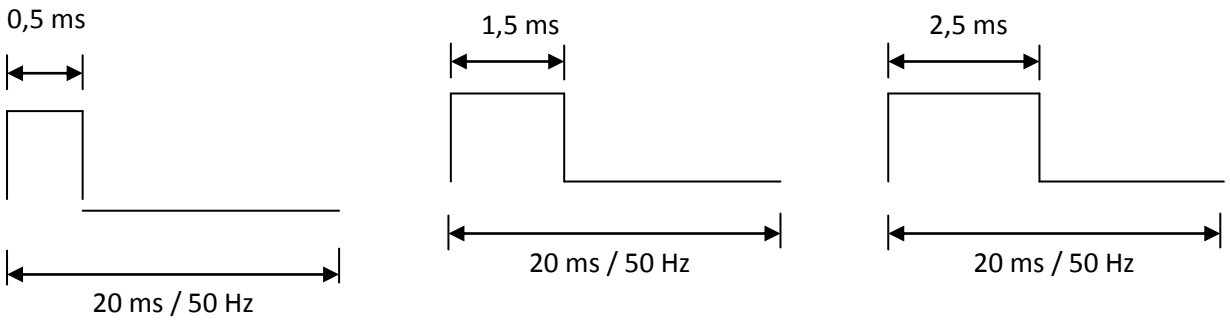
Come si controllano

L'impulso di controllo serve per controllare la posizione, cioè per far muovere il rotore fino a che non raggiunge l'angolazione desiderata.Il controllo è un segnale alto che si ripete a distanza di 20 ms, seguito dal livello basso. In base alla durata dell'impulso 1, il servo si posiziona tra 0 e 180 gradi.

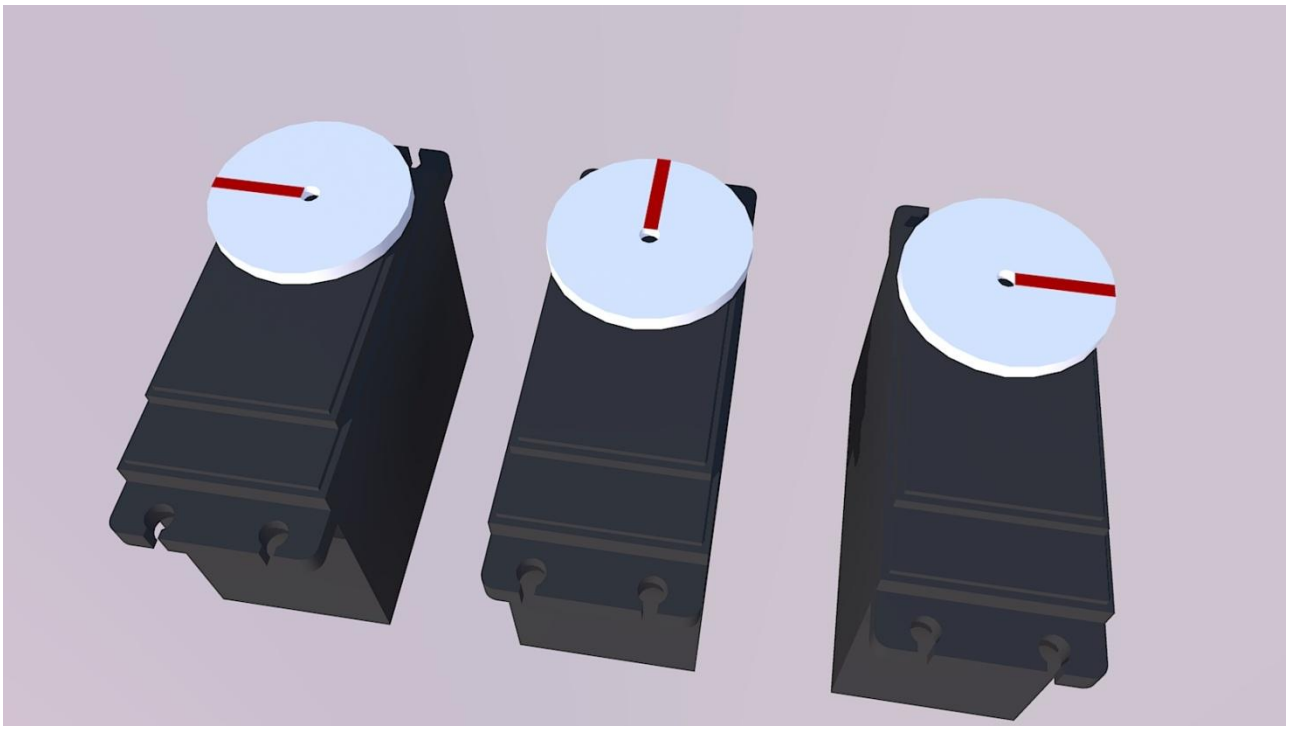
La tabella di seguito riporta alcuni esempi (ogni costruttore porta delle leggere differenze)

	Impulso di controllo				Cavi di collegamento		
Costruttore	**min.**	**centro.**	**max**	**Hz**	**+ batt**	**-batt**	**impulso**
Futaba	0.9	1.5	2.1	50	rosso	nero	bianco
Hitech	0.9	1.5	2.1	50	rosso	nero	giallo
Graupner/Jr	0.8	1.5	2.2	50	rosso	marrone	arancio
Multiplex	1.05	1.6	2.15	40	rosso	nero	giallo
Robbe	0.65	1.3	1.95	50	rosso	nero	bianco
Simprop	1.2	1.7	2.2	50	rosso	azzurro	nero

Nel disegno è illustrato il meccanismo:



0° **90°** **180°**



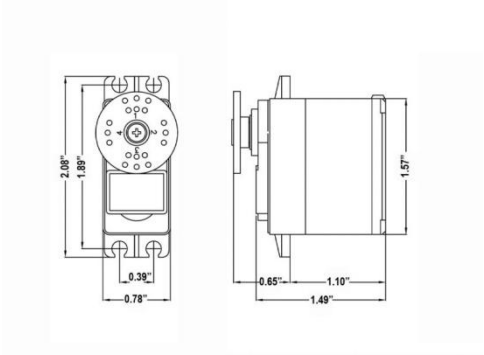
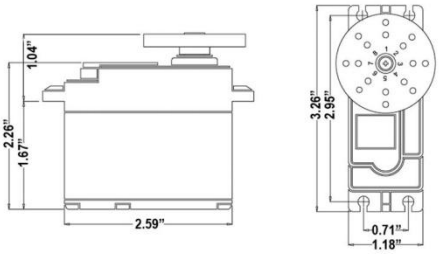
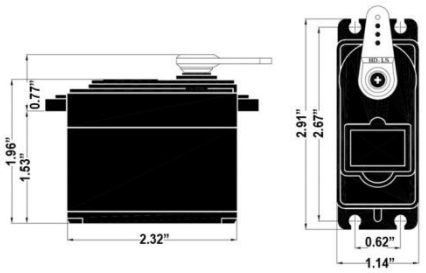
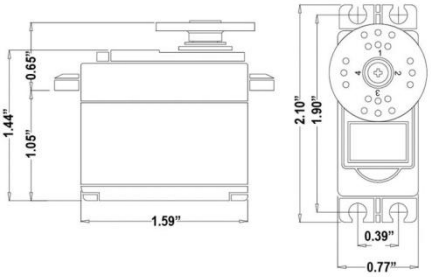
Le caratteristiche

Coppia: la forza che è in grado di esercitare. È espressa in kg/cm e indica quanti chili il servo è in grado di muovere per ogni cm di lunghezza del braccio (a partire dal perno di rotazione)

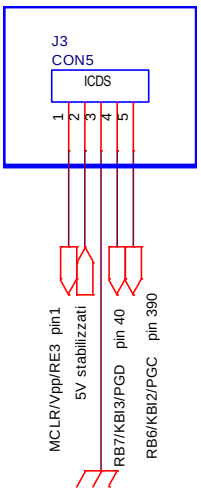
Velocità di rotazione. Espressa solitamente in gradi (o secondi) al secondo
Angolo di rotazione. Più diffusi servo che ruotano 180, ma ce ne sono che limitano a 90 o estendono a 270 il loro movimento.

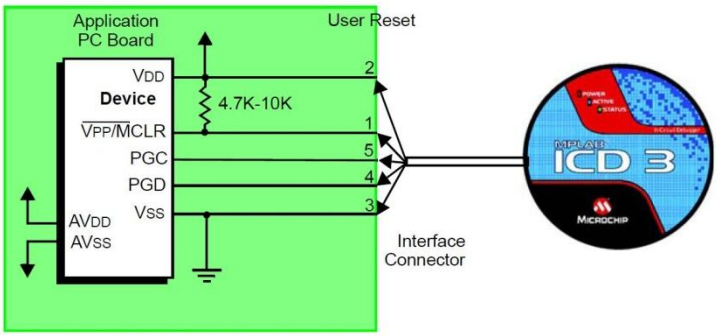
Tipo di ingranaggi: plastica o metallo. È indice della robustezza.

I servo motori da noi utilizzati sono i seguenti:

Servomotore	Dimensioni	Specifiche
Base e Polso: **HS-475HB** *(sia le immagini che le specifiche riguardano il servo motore dell'Hitec HS-485HB, si tratta dell'aggiornamento del 475, il quale non ha subito variazioni)* 		**Categoria:** Deluxe **Coppia (4,8V):** 4,4 kg*cm **Coppia (6,0V):** 5,5 kg*cm **Velocita' (4,8V):** 0,23 sec/60 **Velocita' (6,0V):** 0,18 sec/60 **Dimensioni:** 38,80x19,80x36 mm **Peso:** 40 gr **Ingranaggi:** Karbonite **Cuscinetti:** 2 **Motore:** 3 poli ferrite
Spalla: **HS-805BB** 		**Categoria:** Maxiservo **Coppia (4,8V):** 20,2 kg*cm **Coppia (6,0V):** 23,1 kg*cm **Velocita' (4,8V):** 0,19 sec/60 **Velocita' (6,0V):** 0,15 sec/60 **Dimensioni:** 66,0x30,4x58,4 mm **Peso:** 119,7 gr **Ingranaggi:** Nylon **Cuscinetti:** 2
Gomito: **HS-755HB** 		**Categoria:** Scala 1:4 **Coppia (4,8V):** 11,0 kg*cm **Coppia (6,0V):** 13,2 kg*cm **Velocita' (4,8V):** 0,28 sec/60 **Velocita' (6,0V):** 0,23 sec/60 **Dimensioni:** 59x29x50 mm **Peso:** 110 gr **Ingranaggi:** Karbonite **Cuscinetti:** 2 **Motore:** 3 poli ferrite
Mano: **HS-422** 		**Categoria:** Standard **Coppia (4,8V):** 3,1 kg*cm **Coppia (6,0V):** 3,7 kg*cm **Velocita' (4,8V):** 0,20 sec/60 **Velocita' (6,0V):** 0,17 sec/60 **Dimensioni:** 40,6x20,3x35,5 mm **Peso:** 45,6 gr **Ingranaggi:** Nylon **Cuscinetti:** Boccola autolubrificante

Programmazione PIC





La programmazione del PIC è stata eseguita con lo strumento di sviluppo della MicroChip MPLAB ICD3. Il quale presenta le seguenti caratteristiche (in inglese, dal sito del costruttore):

- **Real-time Debugging -** MPLAB ICD 3 In-Circuit Debugger is designed to support high-speed processors running at maximum speeds, allowing embedded engineers to debug applications on their own hardware in real time.
- **Ruggedized Probe Interface -** Protection circuitries are added to the probe drivers to guard the probe kit from power surges from the target. Vdd and Vpp voltage monitors protect against over-voltage conditions, and all lines have over-current protection. The unit can provide power to a target (up to 100 ma).
- **Microchip Standard Connectivity -** MPLAB ICD 3 In-Circuit Debugger employs a standard Microchip debugging connector (RJ-11).
- **Portable, USB-powered and RoHS-Compliant -** Housed in a small (3.7" x.8") and attractive enclosure, the MPLAB ICD 3 In-Circuit Debugger is powered by the USB port, so an external power adapter is not required. MPLAB ICD 3 In-Circuit Debugger is CE and RoHS-compliant.
- **High Speed Programming -** Fast programming allows both quick firmware reload for fast debugging and for in-circuit re-programming. Programming times are improved up to 15x over MPLAB ICD 2.
- **Low Voltage Emulation -** MPLAB ICD 3 supports target supply voltages from 2.0 to 5.5 volts.
- **Test Interface Module -** Included with every MPLAB ICD 3 is a test module to test I/O lines to confirm the unit is working properly.
- **Ease of Maintenance and Feature Upgrade -** Adding new device support and advanced features to MPLAB ICD 3 In-Circuit Debugger is as simple as installing later versions of the MPLAB IDE, downloadable free. MPLAB ICD 3 In-Circuit Debugger is field upgradeable through a firmware download from MPLAB IDE.
- **Low Cost -** MPLAB ICD 3 In-Circuit Debugger breaks the price barrier for a complete and advanced in-circuit debugger, offering new ways to interact with and debug applications at a fraction of the cost of traditional emulator systems.
- **Powerful Debugging -** High powered debugging with MPLAB IDE, supporting multiple breakpoints, stopwatch, source code file debugging in MPLAB's editor for quick program modification/debug.

Il PIC

Introduzione

Un microcontrollore è un dispositivo elettronico che opportunamente programmato è in grado di svolgere diverse funzioni in modo autonomo. Essenzialmente gestisce delle linee di input e di output in relazione al programma in esso implementato. Esistono diverse famiglie di dispositivi in grado di svolgere queste funzioni come ad esempio lo Z80, ST6 e il più evoluto 8088; I più semplici dispositivi oggi in commercio, i PIC prodotti e distribuiti dalla Microchip.

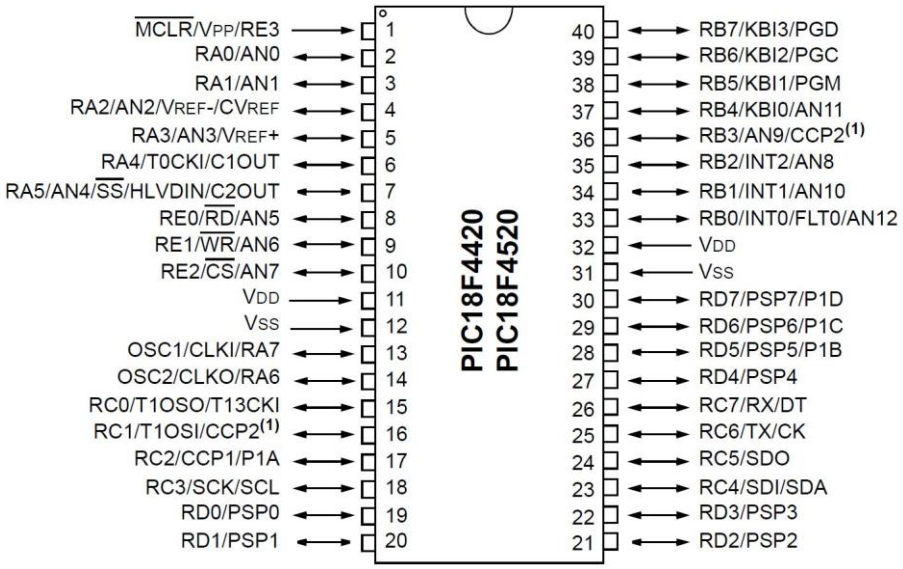
Questi dispositivi implementano al loro interno un vero e proprio microprocessore completo di CPU, RAM, Timer e numerose linee di IN/OUT. A differenza dei microprocessori più evoluti nei PIC il programma è contenuto all'interno in un'apposita area di memoria (non volatile) e viene eseguito ciclicamente, anche la RAM per i dati volatili è all'interno dello stesso dispositivo, alcuni dispongono di aree dati non volatili e riscrivibili (EPROM). Lo stadio di IN/OUT è già implementato all'interno ed alcuni dispongono già di interfacce per segnali analogici, per comparatori o per comunicazioni seriali.

I microcontrollori PIC dispongono di un numero ridotto di istruzioni e quindi sono dei dispositivi di tipo RISC e si programmano in assembly ovvero in codice macchina.

In commercio si trovano diversi modelli di PIC a seconda della complessità e delle funzioni implementate, alcuni dispositivi si differenziano anche per la quantità di memoria disponibile e per la quantità di Timer a disposizione. Le versioni con memoria Flash sono programmabili più volte (e quindi idonei alla sperimentazione) mentre le versioni OTP (One Time Programmable) si programmano una sola volta. Le versioni UV sono riprogrammabili dopo la cancellatura con raggi ultravioletti.

Il nostro PIC

Il microcontrollore è il cuore del sistema ed è da esso che parte ogni segnale di controllo verso le periferiche. Per il PIC la scelta è caduta su un integrato della Microchip ovvero un PIC18F4520.

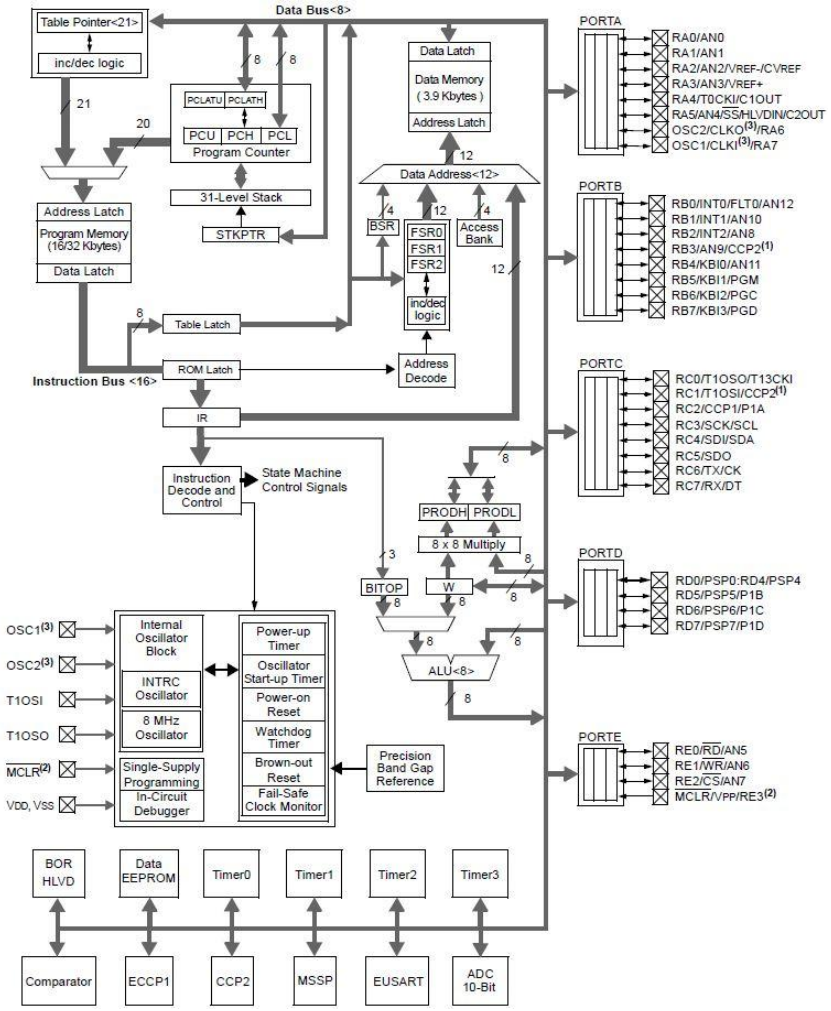


Le caratteristiche principali del microcontrollore sono di avere un'architettura di tipo RISC (reduce instruction set computer), e di possedere 32 Kb di memoria di programma di tipo flash e 1500 Byte di RAM più una memoria EEPROM di 256 byte, il clock è ottenuto da un oscillatore interno stabilizzato tramite un quarzo e due condensatori per realizzare il tipico circuito a pigreco per far oscillare la porta NOT CMOS interna al micro, esso ha una frequenza di 8 Mhz e si può arrivare a far funzionare il circuito fino a 40 Mhz attivando il PLL interno (moltiplicatore x 4) e utilizzando un quarzo esterno a 10 Mhz, tuttavia non sono richieste elevate velocità per il controllo del braccio per cui si è utilizzato l'oscillatore interno a 8Mhz.

Caratteristiche principali PIC18F4520:

Parameter Name	**Value**
Program Memory Type	Flash
Program Memory (KB)	32
CPU Speed (MIPS)	10
RAM Bytes	1,536
Data EEPROM (bytes)	256
Digital Communication Peripherals	1-A/E/USART, 1-MSSP(SPI/I2C)
Capture/Compare/PWM Peripherals	1 CCP, 1 ECCP
Timers	1 x 8-bit, 3 x 16-bit
ADC	13 ch, 10-bit
Comparators	2
Temperature Range (C)	-40 to 125
Operating Voltage Range (V)	2 to 5.5
Pin Count	40

Il dispositivo è dotato internamente di vari moduli, molti dei quali non sono utilizzati per il progetto. di seguito è riportato lo schema interno e la descrizione dei moduli utilizzati.



Di tutti i moduli presenti quelli che interessano particolarmente sono le porte di interfaccia verso l'esterno, sono svariate e ad ognuna di esse è associata una lettera *per* identificarla PORTA, PORTB, PORTC, PORTD. PORTE, hanno diversi numeri di bit e possono essere impostate come ingresso o come uscita e alcuni bit di esse possono essere settate come ingressi analogici per il convertitore analogico digitale interno.

Oltre al modulo delle porte è stato utilizzato il modulo TIMER0 per creare un timer appunto di 5µs. le impostazioni dei moduli citati e dei registri collegati verranno spiegati nella parte di informatica.

Come scrivevo prima alcune porte possono essere impostate come ingressi analogici, per il progetto si è utilizzato l'ADC (lettura joystick) impostato tramite appositi registri per funzionare in modalità ingresso analogico per le porte da RA0 a RA5 il resto in modalità digitale 1 o 0.

I moduli ADC:

I moduli convertitori analogico a digitale (A\D) sono 13 ingressi per la versione con 40/44 pin. Questi moduli permettono la conversione di un ingresso analogico al corrispettivo numero digitale a 10 bit.

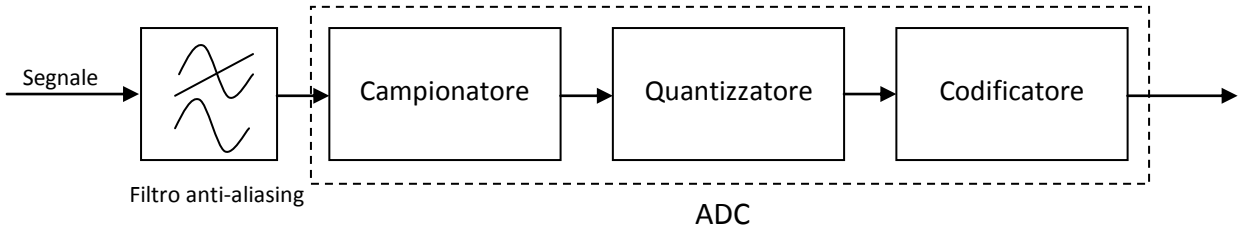
Il modulo ha 5 registri:

- Risultato A/D Registro alto (ADRESH)
- Risultato A/D Registro basso (ADRESL)
- A/D Registro di controllo 0 (ADCON0)
- A/D Registro di controllo 1 (ADCON1)
- A/D Registro di controllo 2 (ADCON2)

Gli ADC in teoria

La conversione analogico-digitale è un procedimento che associa ad un segnale analogico (tempo continuo e continuo nei valori) un segnale numerico (tempo discreto e discreto nei valori).

La conversione analogico-digitale si può suddividere in quattro parti principali:



1) Il filtro anti-alias è un filtro digitale utilizzato prima del campionamento di un segnale, al fine di restringere la banda del segnale stesso per soddisfare approssimativamente il teorema del campionamento di Nyquist-Shannon.
2) Il campionamento consiste nell'andare a "sentire" (misurare, registare) il valore del segnale analogico in diversi istanti di tempo. Il tempo T che intercorre tra una valutazione e l'altra si chiama periodo di campionamento. La frequenza di campionamento F=1/T è invece l'inverso del periodo di campionamento.

   Il teorema che stabilisce quale sia la frequenza minima di campionamento con una determinata caratterizzazione in frequenza (trasformata di Fourier) affinché il segnale analogico possa essere ricostruito a valle a partire da quello discreto in input è il teorema del campionamento di Nyquist, ovvero:

   $$f_c \geq 2 \cdot f_{MAX}$$

   Dove f_c è la frequenza di campionamento ed f_{MAX} è la massima frequenza dello spettro del segnale da campionare.
3) Perché una grandezza sia trasmissibile e codificabile con un numero finito di bit ovvero in forma numerica, è però necessario che essa possa assumere solo un numero finito di valori di codominio discreti; ciò avviene tramite un successivo processo di quantizzazione del valore in ordinata della grandezza in questione.

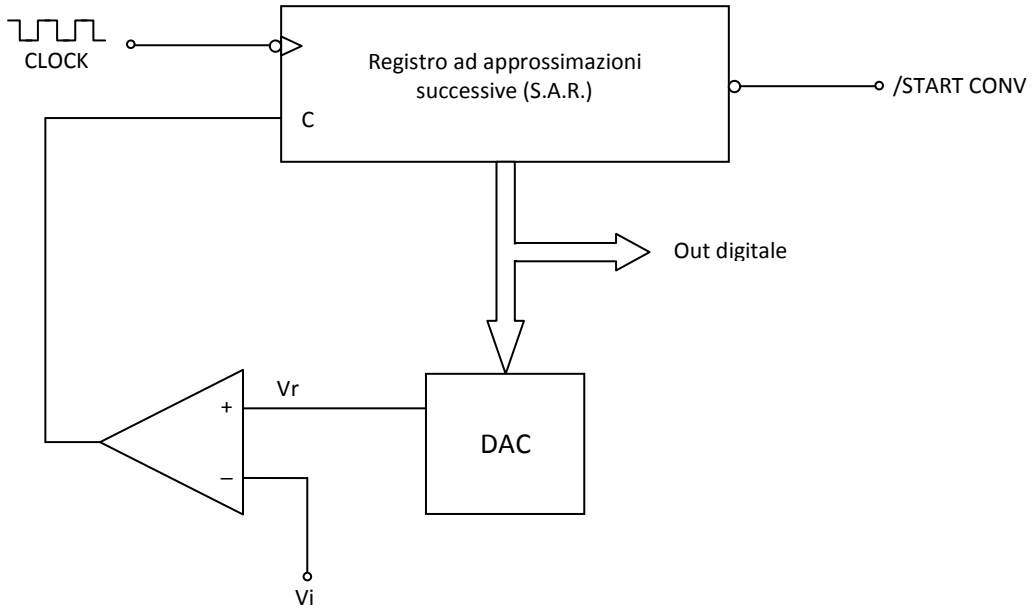
   Per ottenere ciò i valori possibili della grandezza in questione vengono innanzitutto limitati tra un massimo ed un minimo intorno a dei valori discreti preventivamente definiti definendo così le relative regioni di decisione e la dinamica del quantizzatore stesso: in tal modo il valore analogico della grandezza originaria, in corrispondenza del valore campionato in ascissa, verrà ricondotto al più prossimo dei valori discreti preventivamente definiti tramite il processo di decisione.

   Con la quantizzazione vengono però introdotti degli errori detti errori di quantizzazione pari alla differenza tra il valore quantizzato e il suo valore "reale" nel campo continuo. L'errore massimo possibile che potrà essere introdotto volta per volta sarà quindi

pari alla metà dell'intervallo discreto discriminabile o regione di decisione, nel caso limite in cui il valore di ingresso si collochi esattamente a metà tra due valori discreti di uscita ovvero sulla frontiera di due regioni di decisione contigue.

4) Nei convertitori analogico-digitali la codifica consiste nell'associare ad un livello di tensione un determinato numero binario.

Nel nostro microcontrollore la conversione analogica digitale è effettuata da un adc ad approssimazioni successive:



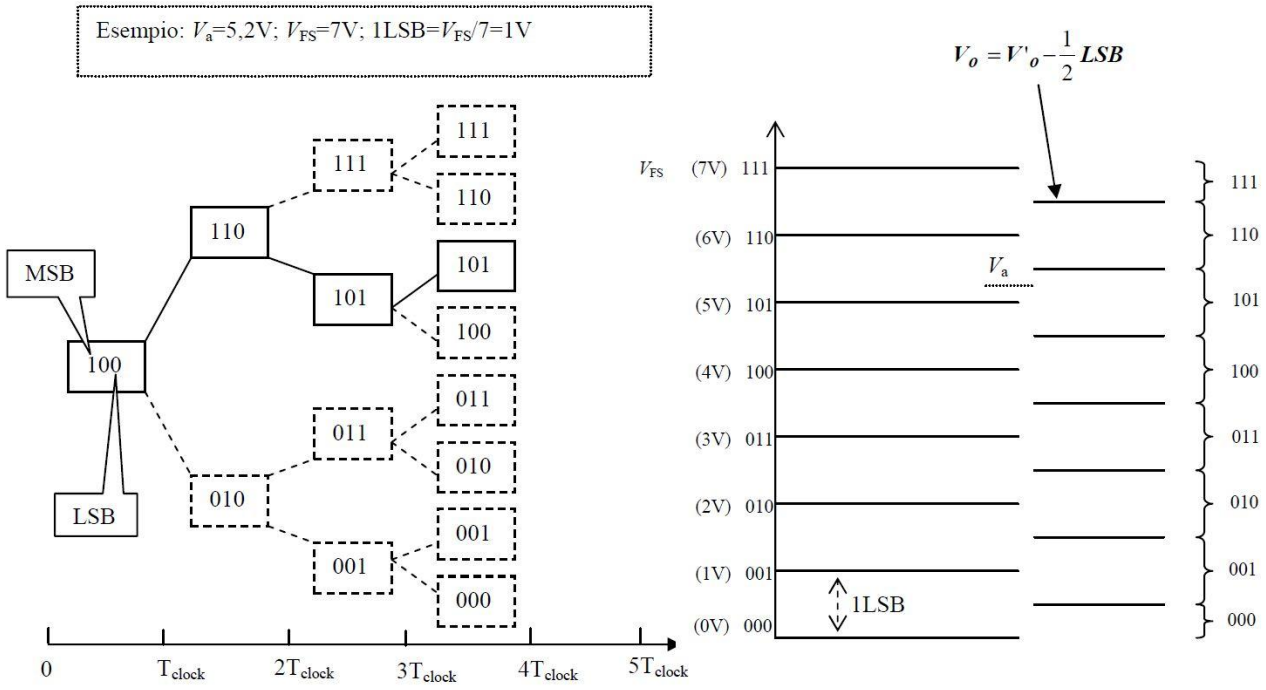
La logica sequenziale è costituita da un registro ad approssimazioni successive (S.A.R.) che implementa quello che in informatica è noto come algoritmo di ricerca dicotomica: si tratta di reperire un elemento in un insieme ordinato (il questo caso il quanto in cui cade la tensione incognita Vi) stabilendo in quale metà del range si trova, poi in quale metà della metà, poi in quale metà della metà della metà ecc.
Le operazioni che deve compiere il comparatore/SAR possono essere espresse mediante questo algoritmo:

```
inizio conversione
azzera tutti i bit
a partire da MSB fino a LSB ripeti:
        inizio ciclo
                poi a 1 il bit corrente
                converti i bit nella tensione Vr
                        se Vi < Vr
                                allora
                                        riporta a 0 il bit corrente
        fine ciclo
fine conversione
```

A ogni ciclo (corrispondente a un ciclo di clock) il S.A.R. "aggiusta" definitivamente un bit; quindi il tempo di conversione è dell'ordine nTck, dove n è la risoluzione in bit e Tck il periodo di CLOCK; più esattamente, considerando un ciclo di clock aggiuntivo per la lettura e la (re)inizializzazione, si ha:

$$T_{conv} = (n+1) \cdot T_{ck}$$

Esempio funzionamento ADC ad approssimazioni successive:



Riprendendo il discorso dei registri degli ADC, riporto qui alcune configurazioni dei bit per settare la conversione analogica digitale.

Il registro ADCON0, mostra nel registro 19-1, le operazioni di controllo del modulo A/D.

REGISTER 19-1: ADCON0: A/D CONTROL REGISTER 0

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CHS3	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	ADON
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-6 **Unimplemented:** Read as '0'

bit 5-2 **CHS<3:0>:** Analog Channel Select bits

0000 = Channel 0 (AN0)
0001 = Channel 1 (AN1)
0010 = Channel 2 (AN2)
0011 = Channel 3 (AN3)
0100 = Channel 4 (AN4)
0101 = Channel 5 (AN5)(1,2)
0110 = Channel 6 (AN6)(1,2)
0111 = Channel 7 (AN7)(1,2)
1000 = Channel 8 (AN8)
1001 = Channel 9 (AN9)
1010 = Channel 10 (AN10)
1011 = Channel 11 (AN11)
1100 = Channel 12 (AN12)
1101 = Unimplemented)(2)
1110 = Unimplemented)(2)
1111 = Unimplemented)(2)

bit 1 **GO/$\overline{\text{DONE}}$:** A/D Conversion Status bit
When ADON = 1:
1 = A/D conversion in progress
0 = A/D Idle

bit 0 **ADON:** A/D On bit
1 = A/D Converter module is enabled
0 = A/D Converter module is disabled

Note 1: These channels are not implemented on 28-pin devices.
2: Performing a conversion on unimplemented channels will return a floating input measurement.

Il registro ADCON1, mostra nel registro 19-2, le configurazioni delle funzioni delle porte.

REGISTER 19-2: ADCON1: A/D CONTROL REGISTER 1

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-q(1)	R/W-q(1)	R/W-q(1)
—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-6 **Unimplemented:** Read as '0'

bit 5 **VCFG1:** Voltage Reference Configuration bit (V_{REF-} source)
1 = V_{REF-} (AN2)
0 = V_{SS}

bit 4 **VCFG0:** Voltage Reference Configuration bit (V_{REF+} source)
1 = V_{REF+} (AN3)
0 = V_{DD}

bit 3-0 **PCFG<3:0>:** A/D Port Configuration Control bits:

PCFG3: PCFG0	AN12	AN11	AN10	AN9	AN8	AN7(2)	AN6(2)	AN5(2)	AN4	AN3	AN2	AN1	AN0
0000(1)	A	A	A	A	A	A	A	A	A	A	A	A	A
0001	A	A	A	A	A	A	A	A	A	A	A	A	A
0010	A	A	A	A	A	A	A	A	A	A	A	A	A
0011	D	A	A	A	A	A	A	A	A	A	A	A	A
0100	D	D	A	A	A	A	A	A	A	A	A	A	A
0101	D	D	D	A	A	A	A	A	A	A	A	A	A
0110	D	D	D	D	A	A	A	A	A	A	A	A	A
0111(1)	D	D	D	D	D	A	A	A	A	A	A	A	A
1000	D	D	D	D	D	D	A	A	A	A	A	A	A
1001	D	D	D	D	D	D	D	A	A	A	A	A	A
1010	D	D	D	D	D	D	D	D	A	A	A	A	A
1011	D	D	D	D	D	D	D	D	D	A	A	A	A
1100	D	D	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	D	D	A
1111	D	D	D	D	D	D	D	D	D	D	D	D	D

A = Analog input D = Digital I/O

Note 1: The POR value of the PCFG bits depends on the value of the PBADEN Configuration bit. When PBADEN = 1, PCFG<2:0> = 000; when PBADEN = 0, PCFG<2:0> = 111.
2: AN5 through AN7 are available only on 40/44-pin devices.

Il registro ADCON2, mostra nel registro 19-3, la configurazione del clock degli A/D, e la programmazione del tempo di acquisizione e la giustificazione.

REGISTER 19-3: ADCON2: A/D CONTROL REGISTER 2

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **ADFM:** A/D Result Format Select bit
1 = Right justified
0 = Left justified

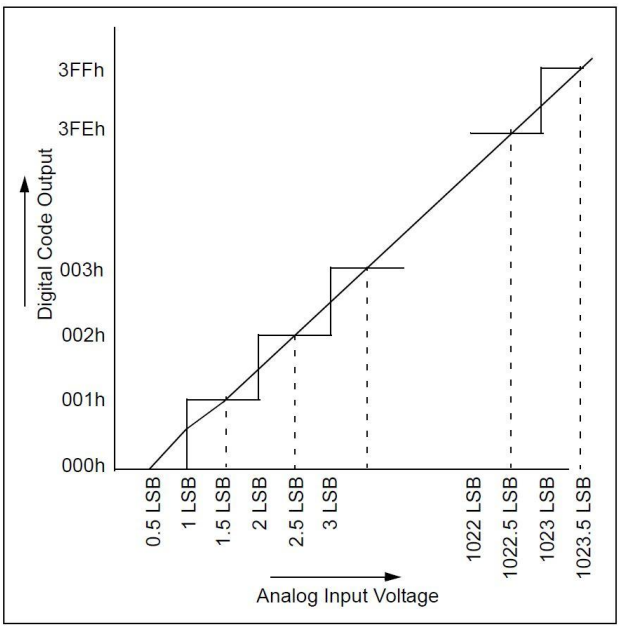
bit 6 **Unimplemented:** Read as '0'

bit 5-3 **ACQT<2:0>:** A/D Acquisition Time Select bits
111 = 20 TAD
110 = 16 TAD
101 = 12 TAD
100 = 8 TAD
011 = 6 TAD
010 = 4 TAD
001 = 2 TAD
000 = 0 TAD(1)

bit 2-0 **ADCS<2:0>:** A/D Conversion Clock Select bits
111 = FRC (clock derived from A/D RC oscillator)(1)
110 = FOSC/64
101 = FOSC/16
100 = FOSC/4
011 = FRC (clock derived from A/D RC oscillator)(1)
010 = FOSC/32
001 = FOSC/8
000 = FOSC/2

Note 1: If the A/D FRC clock source is selected, a delay of one TCY (instruction cycle) is added before the A/D clock starts. This allows the SLEEP instruction to be executed before starting a conversion.

La funzione di trasferimento degli ADC del PIC come riportato dal datashettes risulta la seguente:



Parte Informatica

La parte informatica risulta la parte più importante per il funzionamento del braccio robotico. Per parte informatica si intende principalmente il programma inserito nel PIC per dettare le regole del funzionamento delle periferiche.

Il programma si può dividere in alcuni fasi:

Definizione file #include:

le stringhe seguenti rappresentano i file da includere (include) nel programma, sono delle sottofunzioni, definizione dei registri, senza le quali il programma non potrebbe funzionare.

```
#include <P18F4520.h>

#include   <stdlib.h>
#include   <string.h>
#include   <adc.h>
#include   <portb.h>
#include   <timers.h>
#include   <pconfig.h>
#include   <math.h>
#include   <ctype.h>
#include   <delays.h>
#include   <usart.h>
#include   "def.h"
```

Definizione dei prototipi e Dichiarazione delle variabili:

Definizione delle variabili per assegnare il valore degli ADC, e tempi alti del segnale.

```
void adc();
void InterruptHandlerHigh (void);


int timer=0;
unsigned int adc_base;
unsigned int adc_spalla;
unsigned int adc_gomito;
unsigned int adc_polso;
unsigned int adc_mano;
int tempo_base=0;
int tempo_base_p=0;
unsigned int i=0;
```

Sub routine ADC:

Subroutine che permette di leggere il valore dei Joystick e assegnare un valore decimale. Inizialmente vengono dichiarate 4 variabili (1 per motore) e vengono poste a zero, successivamente si setta in canale (SetChanADC (Ch) ulteriore subroutine presente in adc.h), si

avvia la conversione, si aspetta che la conversione sia terminata e infine si assegna alla relativa variabile il valore letto.

```
void adc(){
adc_base=0;
adc_spalla=0;
adc_gomito=0;
adc_polso=0;

            SetChanADC(ADC_CH0);
            ConvertADC();                       // Start conversion
            while( BusyADC() );                 // Wait for completion
                  adc_base = ReadADC();         // Read result

            SetChanADC(ADC_CH1);
            ConvertADC();                       // Start conversion
            while( BusyADC() );                 // Wait for completion
                  adc_spalla = ReadADC();       // Read result

            SetChanADC(ADC_CH2);
            ConvertADC();                       // Start conversion
            while( BusyADC() );                 // Wait for completion
                  adc_gomito = ReadADC();       // Read result

            SetChanADC(ADC_CH3);
            ConvertADC();                       // Start conversion
            while( BusyADC() );                 // Wait for completion
                  adc_polso = ReadADC();        // Read result


CloseADC();                                     // Disable A/D converter

}
```

Void Main():

```
void main(){

      OSCCON=0x7C;
      OSCTUNE |=0x40;

      ADCON1=0xA;
      INTCON = 0x20          //disable global and enable TMR0
      INTCON2 = 0x84;        //TMR0 high priority
      RCONbits.IPEN = 1;     //enable priority levels

      INTCONbits.GIE=1;       // attivo gli tutti interrupt
      INTCONbits.TMR0IE=1;    // Enables the TMR0 overflow interrupt
      INTCONbits.PEIE=1;      // Disables all unmasked peripheral interrupts

//INTERRUPTS TMR0
//interrupt sul timer0 ogni 5u sec
//il quarzo è da 32 Mhz quindi:
//f_macchina=f_oscillatore/4 -->f_macchina=8M-->ciclo_macchina

      T0CONbits.TMR0ON=1;    //ATTIVO IL TIMER 0
```

```
      T0CONbits.T08BIT=0;    //Timer0 è configurato a 16-bit timer/counter
      T0CONbits.T0CS=0;      //Internal instruction cycle clock (CLKO)
      T0CONbits.T0SE=0;      //salita
      T0CONbits.PSA=0;       //Timer0 prescaler is assigned. Timer0 clock
                             //input comes from prescaler output.

      T0CONbits.T0PS2=0;     //carico il prescaler a 0
      T0CONbits.T0PS1=0;
      T0CONbits.T0PS0=0;

      TMR0L=0xF4;            //carico il valore FFF4 nel timer0 così:
      TMR0H=0xFF;            // per ottenere 5us

      TRISC=0;               //dichiaro come uscite le porte
      TRISB=0;
      TRISD=0;

//apro gli ADC

      OpenADC(ADC_FOSC_RC & ADC_RIGHT_JUST &  ADC_12_TAD,
      ADC_VREFPLUS_VDD & ADC_VREFMINUS_VSS & ADC_CH0 & ADC_INT_OFF,
      ADC_CH0 & ADC_INT_OFF);

      for(;;){               //ciclo for() infino del programma

//timer da 50ms 5us*10000=50ms
//generazione segnale per servomotori

           if (i>1000) {


                if (adc_spalla<=400){
                     tempo_spalla=tempo_spalla_p-((400-adc_spalla)/100);

                }
                if (adc_spalla>=600){
                     tempo_spalla=tempo_spalla_p+((adc_spalla-600)/100);
                }
                tempo_spalla_p=tempo_spalla;


                if (adc_gomito<=400){
                     tempo_gomito=tempo_gomito_p-((400-adc_gomito)/100);

                }
                if (adc_gomito>=600){
                     tempo_gomito=tempo_gomito_p+((adc_gomito-600)/100);
                }
                tempo_gomito_p=tempo_gomito;


                if (adc_polso<=400){
                     tempo_polso=tempo_polso_p-((400-adc_polso)/100);
                }
                if (adc_polso>=600){
                     tempo_polso=tempo_polso_p+((adc_polso-600)/100);
                }
                tempo_polso_p=tempo_polso;
```

```
      if (PORTAbits.RA4==1){
            tempo_mano=tempo_mano_p-50;
      }
      if (PORTAbits.RA5==1){
            tempo_mano=tempo_mano_p+50;
      }
      tempo_mano_p=tempo_mano;


      if (adc_base<=400){
            tempo_base=tempo_base_p-((400-adc_base)/100);
      }
      if (adc_base>=600){
            tempo_base=tempo_base_p+((adc_base-600)/100);
      }
      tempo_base_p=tempo_base;


      i=0;
}


if (tempo_spalla>=380) tempo_spalla=380;
if (tempo_spalla<=0)   tempo_spalla=0;
if (adc_spalla>=400 && adc_spalla<=600){
      tempo_spalla=tempo_spalla;
}
if (timer>=1 && timer<=(120+tempo_spalla))
      LATBbits.LATB0=1;
else
      LATBbits.LATB0=0;


if (tempo_gomito>=380) tempo_gomito=380;
if (tempo_gomito<=0)   tempo_gomito=0;
if (adc_gomito>=400 && adc_gomito<=600) {
      tempo_gomito=tempo_gomito;
}
if (timer>=500 && timer<=(620+tempo_gomito))
      LATBbits.LATB1=1;
else
      LATBbits.LATB1=0;


if (tempo_polso>=380)  tempo_polso=380;
if (tempo_polso<=0)    tempo_polso=0;
if (adc_polso>=400 && adc_polso<=600)   {
      tempo_polso=tempo_polso;
}
if (timer>=1000 && timer<=(1120+tempo_polso))
      LATBbits.LATB2=1;
else
      LATBbits.LATB2=0;


if (tempo_mano>=380)   tempo_mano=380;
if (tempo_mano<=0)     tempo_mano=0;
if (adc_mano>=400 && adc_mano<=600)     {
```

```
            tempo_mano=tempo_mano;
      }
      if (timer>=1500 && timer<=(1620+tempo_mano))
            LATBbits.LATB3=1;
      else
            LATBbits.LATB3=0;


      if (tempo_base>=380)   tempo_base=380;
      if (tempo_base<=0)     tempo_base=0;
      if (adc_base>=400 && adc_base<=600)    {
            tempo_base=tempo_base;
      }
      if (timer>=2000 && timer<=(2120+tempo_base))
            LATBbits.LATB4=1;
      else
            LATBbits.LATB4=0;


      if (timer>3000 && timer<3500)
            adc();
      if (timer>4000)
            timer=0;

      }
```

Si può sintetizzare il funzionamento delle stringhe precedenti, nel modo sotto descritto (la spiegazione verrà scritta solo per un motore poiché le altre sono praticamente identiche variano solo i range di funzionamento).

Sul canale RB dal quale esce il segnale di controllo del motore deve sempre essere presente un segnale a 50Hz che vari il suo tempo alto tra i 600us e i 2500us, questo segnale è dato dalla seguente stringa:

```
if (timer>=1 && timer<=(120+tempo_spalla))
            LATBbits.LATB0=1;
      else
            LATBbits.LATB0=0;
```

Avendo un timer da 5us è stato impostato che la porta deve rimanere alta (1) da 1 a 120 della variabile timer, ovvero deve sempre essere maggiore di 600us (5us*120us=600us). Dopodiché la porta verrà riportata bassa (0). L'incremento del tempo alto del segnale è dato dalla variabile tempo_spalla (si riferisce al servo che comanda la spalla del braccio robotizzato).

```
if (i>1000) {
      if (adc_spalla<=400){
            tempo_spalla=tempo_spalla_p-((400-adc_spalla)/100);
      }
      if (adc_spalla>=600){
            tempo_spalla=tempo_spalla_p+((adc_spalla-600)/100);
      }
      tempo_spalla_p=tempo_spalla;
```

tempo_spalla viene aggiornato ogni 50ms (5us*1000=50ms), solo se il valore dell'adc è inferiore a 400 o superiore a 600. Se il joystick posizionato in un certo modo fa leggere all'adc un valore maggiore di 600 aumenta il tempo alto di una variabile proporzionale al valore letto dall'adc, ovvero se lo sposto di poco il braccio si muoverà lentamente, se invece lo sposto fino al limite il braccio robotizzato si muoverà più velocemente. Succederà il contrario se invece l'adc misurerà un valore inferiore a 400, diminuirà il tempo alto.
Il range centrale che non incrementa o decrementa la variabile del tempo è stato impostato per permettere la rotazione di un motore alla volta senza influenzare anche gli altri, perché il joystick è molto sensibile.

Infine sono state scritte delle stringhe per non far superare in valori massimi di funzionamento dei motori, per azzerare la variabile timer e chiamare la funzione adc():

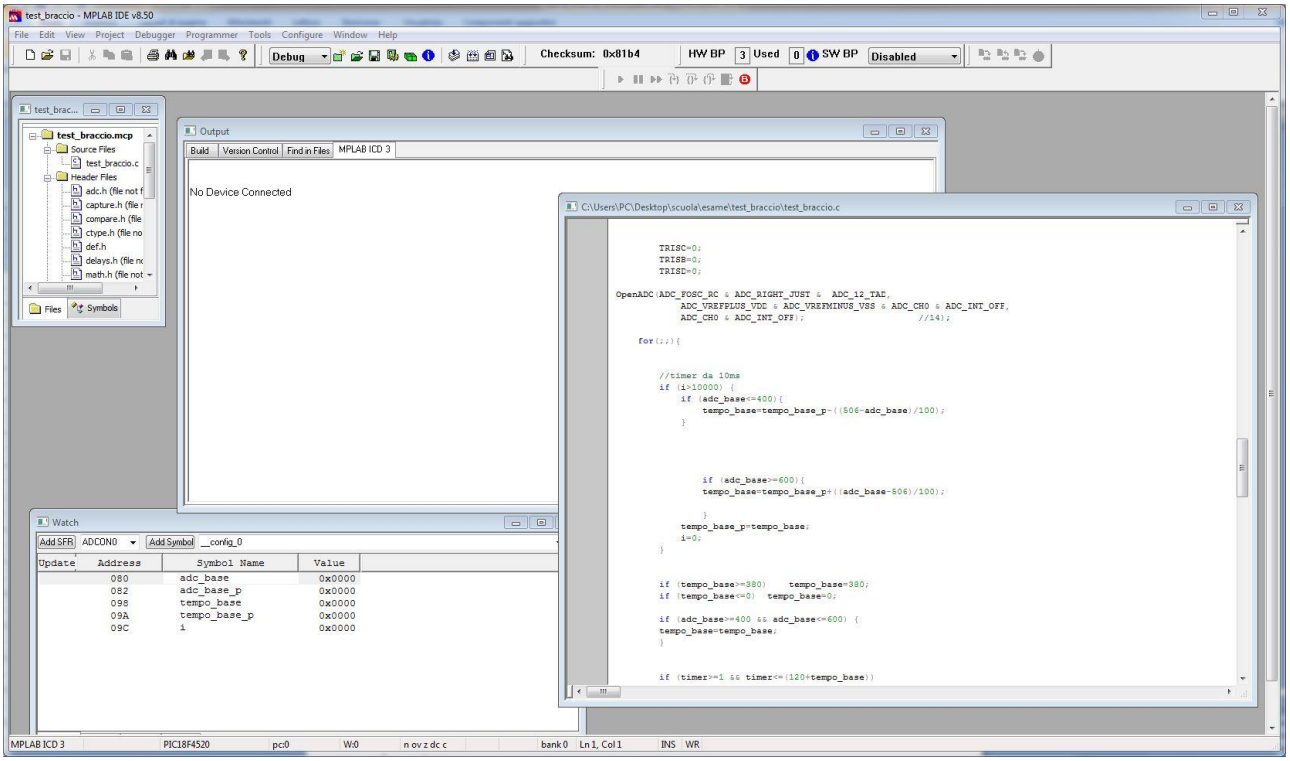
```
if (tempo_spalla>=380) tempo_spalla=380;
if (tempo_spalla<=0)   tempo_spalla=0;

if (timer>3000 && timer<3500)
      adc();
if (timer>4000)
      timer=0;
```

Interrupt e timer0:

```
// High priority interrupt vector

#pragma code InterruptVectorHigh = 0x08
void
InterruptVectorHigh (void)
{
  _asm
    goto InterruptHandlerHigh //jump to interrupt routine
  _endasm
}

// High priority interrupt routine

#pragma code
#pragma interrupt InterruptHandlerHigh

void
InterruptHandlerHigh ()
{

      if (INTCONbits.TMR0IF) {     //check for TMR0 overflow
      timer++;                     //incremento la variabile timer
            i++;                   //incremento la variabile i

            TMR0L=0xF4;            //carico il valore FFF4 nel timer0 così:
            TMR0H=0xFF;            //per ottenere 5us
            INTCONbits.TMR0IF = 0; //clear interrupt flag
      }


}
```

Il compilatore

Dato che il microcontrollore non è stato programmato in linguaggio assembly si è reso necessario l'utilizzo di un compilatore in linguaggio C visto che questo offre molte possibilità anche a livello piuttosto basso, si può agire sui singoli bit dei registri del microcontrollore ma svolgere anche operazioni più complesse grazie alle funzioni integrate del C.
Il compilatore impiegato è il MPLAB IDE v8.50 della Microchip (di seguito è riportata una videata del programma), con esso si crea un file .hex (formato esadecimale) pronto per essere trasferito sulla memoria del microcontrollore, PIC tramite il programmatore ICD3; l'operazione di trasferimento è molto veloce.



Il programmatore offre varie funzionalità tra cui un tool di debug che permette attraverso breakpoint di capire dove si trovano eventuali errori.

Conclusioni

Inizialmente ero preoccupato per la realizzazione della meccanica, abbastanza laboriosa, ma la parte che mi ha portato "via" più tempo è stata la stesura del programma (ancora incompleto)
ho riscontrato dei problemi nella dichiarazione del timer0, perché lavorando ad alte velocità anche una sola stringa nella subroutine di interrupt avrebbe cambiato l'intero programma, e anche perché i calcoli teorici non risultavano nell'oscilloscopio gli stessi.
All'inizio è stato perso del tempo perché non riuscivamo a capire perché alcune porte andassero un momento prima e un momento dopo no, siamo arrivati alla conclusione, che fossimo noi a causare questi malfunzionamenti poiché con la sonda dell'oscilloscopio andavamo a toccare, per verificare il segnale di controllo dei servo anche in pin affianco, e in quel periodo di progetto non

era ancora stata realizzata l'alimentazione dei motori, e quindi causavamo dei corti portando le porte dei PIC a 12V.
Purtroppo il progetto finale presenta alcuni problemi che per mancanza di tempo non siamo riusciti a risolvere, ovvero il segnale di controllo dei servomotori non è precisissimo e quindi il robot in fase di funzionamento presenta delle vibrazione dei bracci. La non precisione del segnale pensiamo sia dettata dal fatto che le velocità di funzionamento siano troppo elevate, i quindi che non riesca ad entrare nel ciclo if di commutazione della porta ogni volta nello stesso preciso momento. La parte della mano e del controllo seriale sempre per motivi di tempo, e perché ne avrebbero tolto allo studio, non è stata realizzata, ma come descritto in precedenza progettata.
Lo sviluppo di questo progetto è stato, molto utile, poiché è il primo progetto che si realizza da zero, senza delle spiegazioni del professore specifiche sul progetto.
Per la stesura di questa tesina mi sono servito oltre alle mie conoscenze, dei libri di testo di elettronica, telecomunicazioni, tdp e di internet, fondamentale è stato l'utilizzo della rete dalla quale ho ottenuto informazione che mi sembrava giusto riportare per la miglior comprensione del progetto.