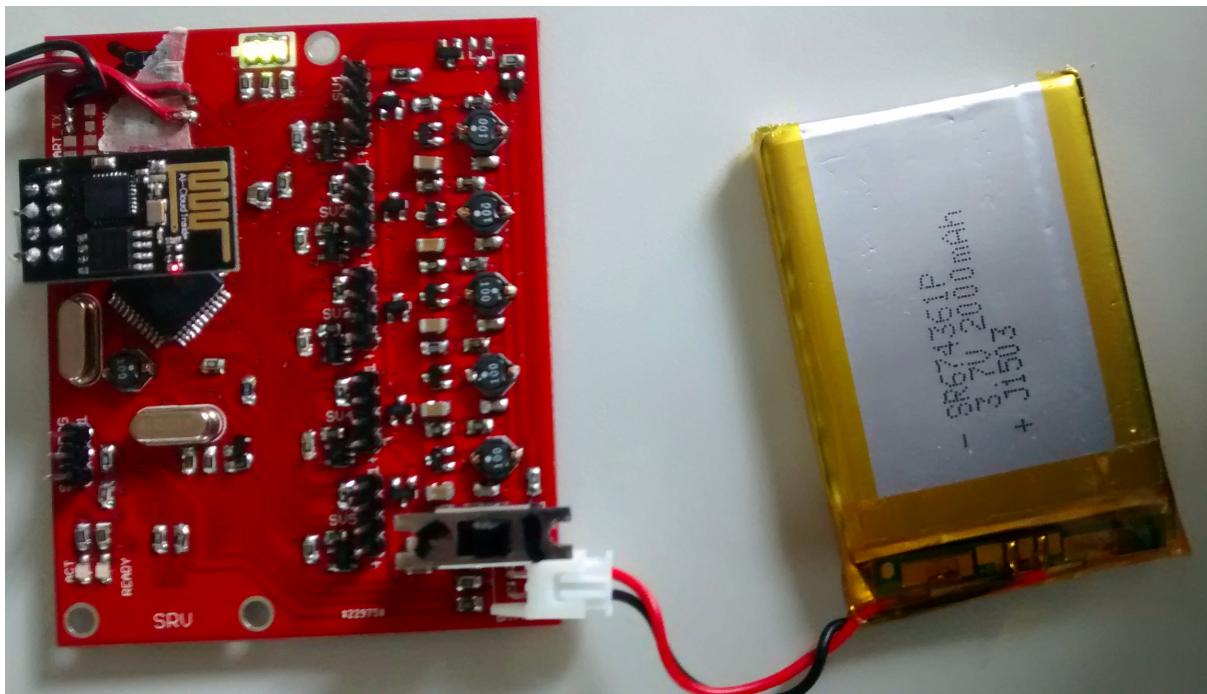


Progettazione di una scheda di controllo per mano robotica

Paolo Scaramuzza (1105663)

25 luglio 2016



Indice

1	Introduzione	3
1.1	Specifiche di progetto	5
2	Architettura hardware	6
2.1	Servomotori: controllo dell'angolo	7
2.2	Servomotori: lettura della corrente	8
2.3	Servomotori: lettura dell'angolo	10
2.4	ESP8266	12
3	Architettura software	13
3.1	esp_driver	14
3.2	serio_driver	15
3.3	adc_driver	16
3.4	battery_driver	17
3.5	servo_driver	17
4	Risultati	19
4.1	Sviluppi futuri	20

1 Introduzione

InMoov [1] è un robot realizzato a partire dal 2012 tramite l’uso di una stampante 3D dallo scultore francese Gael Langevin. Tutti i progetti sono pubblicati sotto licenza Creative Commons [2] e dunque rientrano nell’ambito della filosofia Open Source [3]. Proprio grazie alla disponibilità libera e gratuita dei *file* di progetto, negli anni sono state sviluppate diverse varianti dell’originale e ciascuna con il suo scopo specifico [4–6].

In generale InMoov è in grado di percepire suoni, vedere e muoversi (anche se non ancora autonomamente) nell’ambiente circostante. Il robot reagisce anche a comandi vocali che gli vengono impartiti dal proprietario.

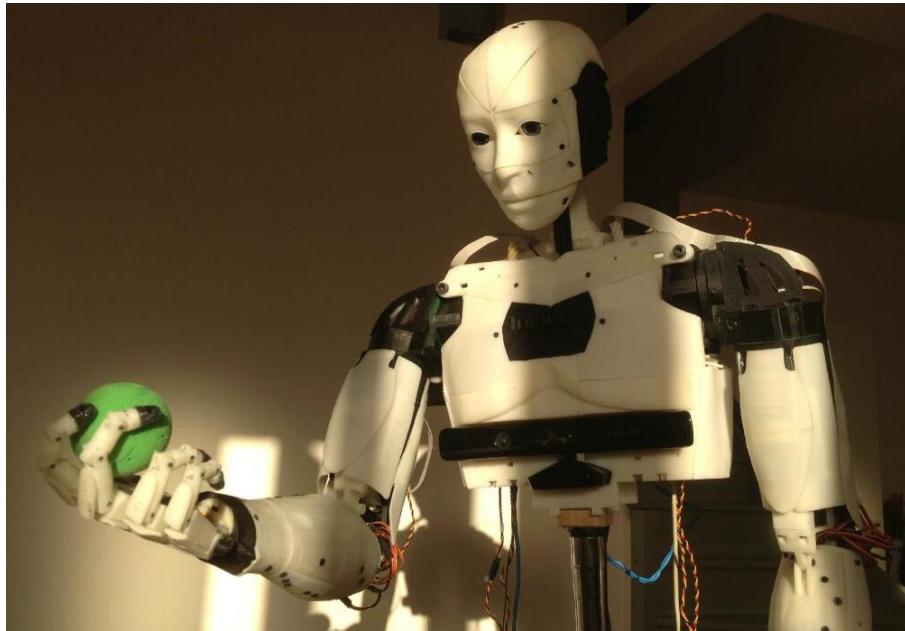


Figura 1: L’automa InMoov

Gli arti e gli attuatori dell’automa sono realizzati da servomotori controllati tramite la piattaforma Arduino. I servomotori sono dei particolari motori a corrente continua il cui alberino può essere posizionato in una specifica posizione angolare tramite un segnale di controllo. Finché il segnale di controllo è presente il motore mantiene la posizione richiesta. Vista la loro semplicità ed economicità questi motori trovano ampio impiego in ambito modellistico e robotico.

I servomotori sono un sistema in catena chiusa e nelle versioni più economiche non è possibile conoscere l’angolo effettivo a cui si trova l’alberino. Di conseguenza se, ad esempio, qualcosa intralcia il movimento di un arto o di un giunto il motore continuerà a sforzare cercando di raggiungere la posizione richiesta. In condizioni di stallo la corrente assorbita aumenta notevolmente e si rischia di danneggiare i componenti sia elettronici che meccanici del motore.

Un sistema di questo tipo può soddisfare i requisiti di un’applicazione modellistica. In ambito robotico e soprattutto per quanto riguarda un arto come la mano è invece di vitale importanza poter conoscere l’angolo effettivo dell’alberino (da cui è poi possibile ricavare la posizione nello spazio dell’arto) e la corrente assorbita dal motore (che è direttamente proporzionale alla forza applicata sul giunto).



Figura 2: Un tipico servo motore per applicazioni modellistiche

Nel caso particolare di una mano robotica, rendere disponibili queste informazioni ed impiegarle per realizzare un anello di controllo attorno al servomotore permette di realizzare movimenti e manipolazioni molto sensibili e complesse. Ad esempio è possibile stringere un oggetto con una determinata forza evitando ad esempio di romperlo se è delicato oppure muovere le dita ad una determinata velocità, fermandole quando si è raggiunto l'oggetto da manipolare.

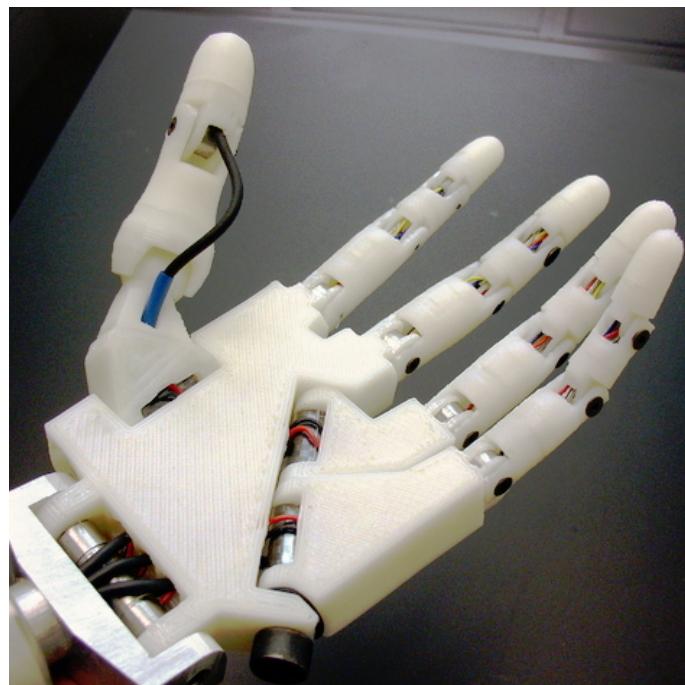


Figura 3: Mano di InMoov

1.1 Specifiche di progetto

A partire da quanto precedentemente descritto si sono evidenziati i seguenti problemi per quanto riguarda il controllo del semplice angolo dei servomotori che realizzano il movimento delle dita di una mano proposto da InMoov:

- Mancanza di informazioni riguardanti angolo e velocità effettiva del motore. Questo fa sì che le dita raggiungano la posizione indicata nel minimo tempo possibile, soluzione non sempre desiderabile.
- Mancanza di informazioni riguardanti la corrente assorbita. Di conseguenza non è possibile controllare la forza della stretta, né sapere se il motore è in stallo.
- Mancanza di un pacchetto ROS che permetta alla mano di interfacciarsi con altri progetti di robotica esistenti.
- Scarsa mobilità. La mano richiede infatti una sorgente di alimentazione esterna che spesso non è disponibile o non fornisce la tensione o la corrente richiesta.

Alla scheda che controlla i servomotori è dunque richiesto:

- Controllo indipendente di cinque servomotori, ciascuno con la propria informazione di angolo (incertezza $\pm 1^\circ$) e corrente assorbita ($\pm 1mA$).
- Controllo tramite WiFi e porta USB per *debug* e condizioni di scarsa ricezione radio.
- Alimentazione a batteria con durata superiore ai trenta minuti
- Pacchetto ROS in grado di interfacciarsi con la scheda e inviare e ricevere i dati con un *rate* di almeno 100Hz.
- Facilità d'uso e programmazione al fine di avere un'architettura chiara ed estendibile che funga da base per nuovi progetti.

2 Architettura hardware

Dalle specifiche risulta chiaro che non è possibile utilizzare un'architettura analogica ma è necessario ricorrere ad un microcontrollore. In particolare quest'ultimo deve soddisfare i seguenti requisiti:

- Ampia connettività per poter comunicare con moduli esterni e via USB
- Almeno 10 canali analogici per poter leggere l'angolo e la corrente assorbita di ciascun servomotore
- Almeno 5 canali PWM per comandare i servomotori
- Ampia memoria RAM e flash per poter contenere programmi anche complessi
- Frequenza operativa elevata
- Basso consumo per garantire un'elevata durata della batteria
- Facile utilizzo per rendere facilmente estendibile il progetto

Visto e considerato quanto sopra la scelta è ricaduta sulla famiglia di microcontrollori ATxmega [7] prodotta dalla Atmel. Questi prodotti rappresentano, dal punto di vista della potenza di calcolo e dimensione della memoria, un'ottima via di mezzo tra Arduino e soluzioni basate su *core* ARM a 32bit.

Per quanto riguarda la periferiche che corredano l'unità di elaborazione, questi microcontrollori sono caratterizzati da una dotazione completa e performante, che non ha pari tra gli altri produttori.

La scelta è ricaduta in particolare sull'ATxmega128D4 le cui caratteristiche salienti sono:

- 128kByte di memoria flash e 8kByte di RAM
- 48 pin di I/O
- 20 uscite PWM a 16bit
- Due USART (UART, SPI, I2C)
- Un convertitore analogico-digitale (ADC) a 12bit e 16 canali
- Massima frequenza operativa 32MHz

Scelto il microcontrollore l'hardware è stato organizzato come è mostrato in Figura 4. L'impiego di una batteria al litio da 3.7V rende necessario innalzare la tensione fino a 5V per pilotare i servomotori. Sono stati dunque inclusi sulla scheda cinque convertitori *boost* (uno per ogni dito della mano) che sono abilitati e disabilitati dal microcontrollore. In particolare essi sono spenti quando è presente l'alimentazione via USB oppure quando la batteria è scarica. Il modulo WiFi (ESP8266) si interfaccia tramite una porta seriale, così come la connettività USB è fornita tramite un adattatore da seriale a USB. I servomotori sono comandati dal modulo PWM, il loro angolo e la corrente assorbita sono letti dall'ADC.

Sulla scheda è stato inoltre incluso un programmatore USB (non mostrato in figura) per l'ATxmega in modo da semplificare la programmazione della scheda evitando l'acquisto di costose soluzioni proprietarie.

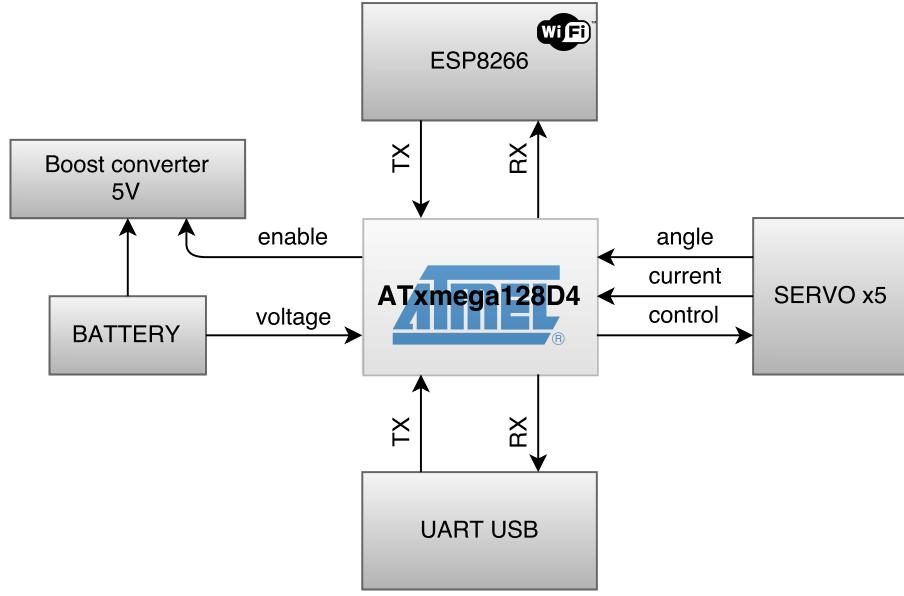


Figura 4: Schema a blocchi dell’architettura hardware

2.1 Servomotori: controllo dell’angolo

I servomotori sono controllati tramite un segnale a modulazione dell’ampiezza dell’impulso (PWM) la cui frequenza è tipicamente 50Hz. Il tempo durante il quale l’impulso si trova ad un valore logico alto (da 3 a 5V) determina l’angolo desiderato.

La relazione che lega tempo e angolo è lineare. Sapendo che un impulso di 1ms corrisponde ad un angolo di 0° gradi, mentre la massima rotazione pari a 180° è raggiunta con un impulso di 2ms è possibile ricavare la seguente caratteristica tra t_{ON} , tempo in millisecondi durante il quale l’impulso è a livello logico alto e α_{shaft} , angolo dell’alberino in gradi.

$$\alpha_{shaft} = 180 \cdot (t_{ON} - 1)$$

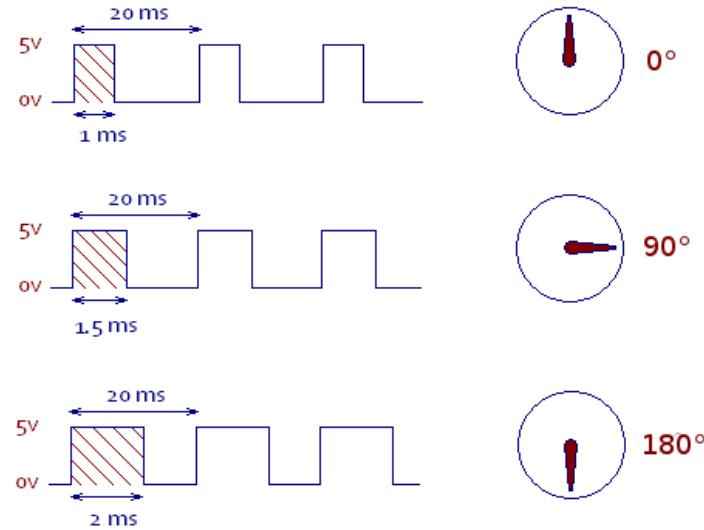


Figura 5: Forme d’onda per il segnale di controllo di alcuni angoli notevoli

Per un segnale PWM si definisce *duty cycle* o *duty ratio*, indicato con D , il rapporto tra il tempo durante il quale la forma d'onda assume il valore logico alto e il periodo del segnale: $D = \frac{t_{ON}}{T}$ [8]. Spesso il *duty cycle* è espresso come una percentuale.

Impiegando il *duty cycle* al posto di t_{ON} nella precedente relazione, ovvero dividendo a destra per il periodo (20ms) si ottiene una formula indipendente dal tempo. Tale relazione sarà utile per impostare i parametri operativi del modulo del microcontrollore che genera i segnali PWM per i servomotori.

$$\alpha_{shaft} = 180 \cdot (D - 0.05)$$

La relazione sopraindicata fornisce anche un'indicazione della risoluzione minima necessaria per esprimere il valore di D . Come da specifiche si vuole controllare la posizione dei servomotori con una precisione almeno di $\pm 1^\circ$.

Dalle forme d'onda si ricava che $0.05 \leq D \geq 0.1$ da cui $\Delta_D = 0.05$ perciò:

$$N_{PWM} = -\log_2 \left(\frac{\Delta_D}{180} \right) = 11.8bit$$

Sono quindi necessari almeno 12bit di risoluzione, ampiamente forniti dal modulo PWM del microcontrollore scelto che garantisce fino a 16bit.

2.2 Servomotori: lettura della corrente

Per la lettura della corrente assorbita dai vari motori si è scelto di impiegare un resistore di valore noto e preciso posto tra massa e il pin negativo del connettore dei servomotori.

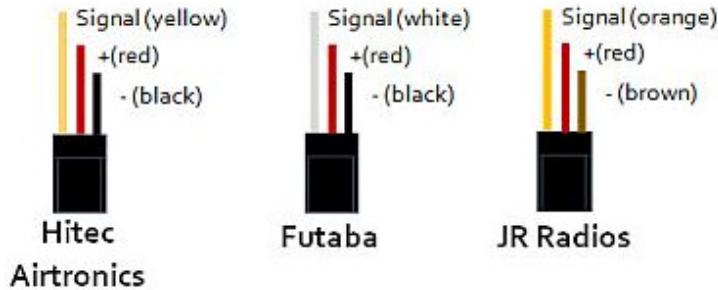


Figura 6: Tipico *pinout* dei servomotori più comuni

Dalla legge di Ohm si ha che $V_c = R_s I_s$, avendo indicato con V_c la tensione ai capi della resistenza di sensing R_s e con I_s la corrente assorbita dal servo motore.

Si vuole la R_s più piccola possibile, al fine di evitare un'eccessiva dissipazione di potenza. Tuttavia minore è la resistenza minore risulta il valore di tensione ai suoi capi, riducendo il rapporto segnale-rumore e rendendo difficile l'acquisizione tramite il convertitore analogico-digitale.

Particolarità del convertitore degli ATxmega è la possibilità di inserire uno stadio di guadagno prima di effettuare la conversione, minimizzando così il numero di componenti richiesti per la misura della corrente.

Avendo indicato con G il guadagno di tale modulo la tensione letta dall'ADC vale: $V_{ADC} = G \cdot V_c = G \cdot R_s I_s$. Usando una tensione di riferimento per la conversione pari ad 1V e indicando con N_{ADC} la risoluzione del convertitore si ha che il risultato dell'acquisizione è pari a:

$$d = 2^{N_{ADC}} V_{ADC} = 2^{N_{ADC}} \cdot G \cdot R_s I_s$$

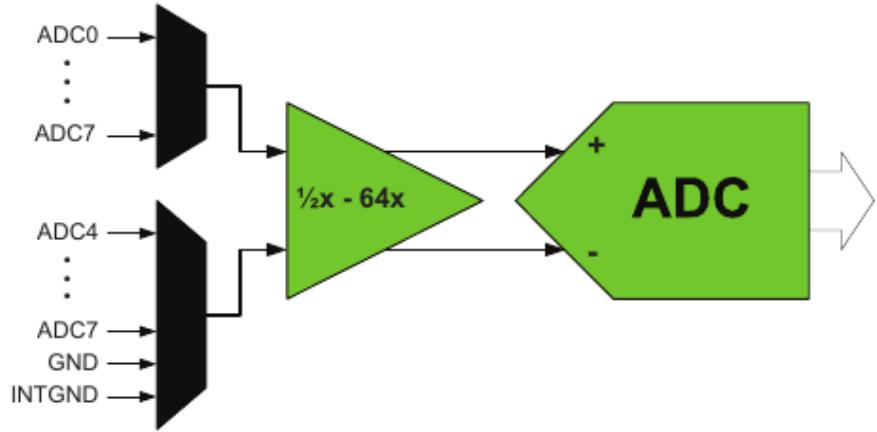


Figura 7: Configurazione con guadagno dell'ADC per un ATxmega. Il guadagno è selezionabile tra 0.5, 1, 2, 4, 8, 16, 32, 64.

Per ricavare una stima della corrente assorbita, in mA , a partire dal valore di lettura del convertitore analogico digitale è necessario invertire la relazione di cui sopra:

$$\hat{I}_s^{(mA)} = d \cdot \frac{1000}{2^{N_{ADC}} \cdot GR_S}$$

In particolare per la scelta di $R_s = 0.24\Omega$, $G = 16$ e $N_{ADC} = 8$ si ottiene che il rapporto $\frac{1000}{2^{N_{ADC}} \cdot GR_S}$ risulta 1.017 che può essere ben approssimato da 1, compiendo un errore dell'1.54%.

Un tale errore risulta tollerabile dalle specifiche e rappresenta comunque un errore sistematico che porta a sovrastimare la misura della corrente garantendo quindi un certo margine di sicurezza.

Il circuito utilizzato per la misura della corrente assorbita è rappresentato in Figura 8.

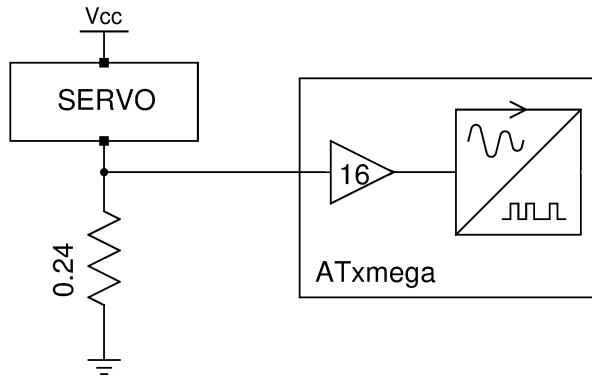


Figura 8: Circuito per la misura della corrente

2.3 Servomotori: lettura dell'angolo

Il circuito di controllo dei servomotori fa uso di un potenziometro lineare per ricavare l'angolo effettivo dell'alberino. Tramite questa informazione è realizzato un anello di retroazione che porta il motore all'angolo desiderato. Poiché il segnale di errore sull'angolo pilota direttamente il motore, maggiore è la differenza tra l'angolo richiesto e quello attuale, più rapida sarà la rotazione. La Figura 9 mostra lo schema a blocchi che realizza un servomotore. Si noti come non sia presente alcun controllo sulla corrente assorbita o sulla velocità di rotazione.

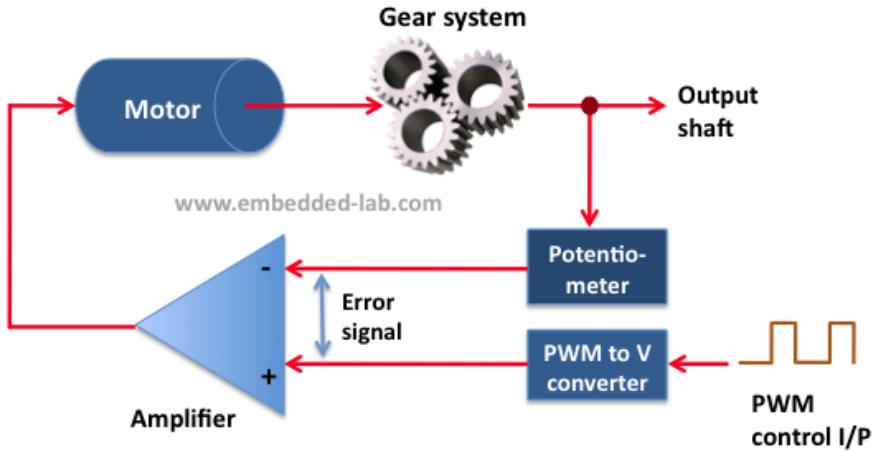


Figura 9: Schema a blocchi di un servomotore

Per avere un'informazione sull'angolo effettivo e sulla velocità di rotazione è possibile ricorrere all'acquisto di servomotori per impieghi industriali, dotati di *encoder* rotativi. Questa soluzione, oltre a far aumentare di un ordine di grandezza il costo, avrebbe richiesto un nuovo progetto per la struttura della mano che ospita i motori e avrebbe complicato notevolmente la soluzione circuitale.

La soluzione proposta nel presente lavoro consiste in una modifica dei servomotori per uso modellistico già presenti sulla mano, andando a prelevare direttamente il segnale al potenziometro rendendolo disponibile per microcontrollore (Figura 11). Tale segnale sarà acquisito, come l'informazione sulla corrente, tramite il convertitore analogico-digitale. La Figura 10 mostra lo schema elettrico che realizza il condizionamento del segnale del potenziometro del servomotore (indicato con R).

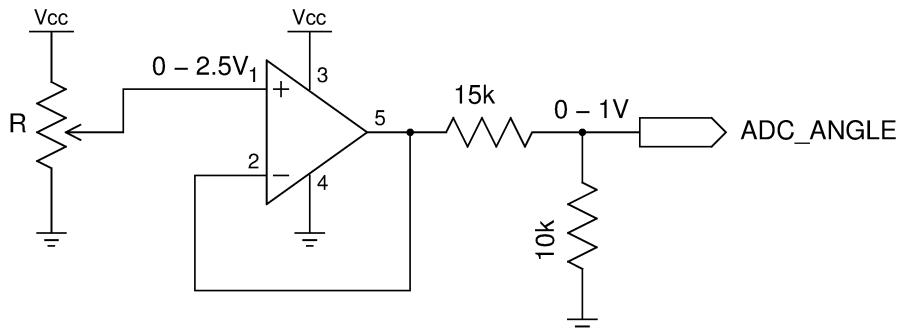


Figura 10: Circuito per la misura dell'angolo

Avendo indicato con V_{CC} la tensione di alimentazione del servomotore, la tensione sulla presa centrale del potenziometro segue la relazione:

$$V_{pot} = V_{CC} \cdot \frac{\alpha_{shaft}}{360^\circ}$$

Di conseguenza per una rotazione compresa tra 0° e 180° e una tensione di alimentazione di $5V$ ci si aspetta $0V \leq V_{pot} \leq 2.5V$.

Il convertitore analogico-digitale del microcontrollore usa una tensione di riferimento pari ad $1V$ di conseguenza è necessario scalare la tensione del potenziometro tramite un partitore prima di immetterla nell'ADC. Prima del partitore tuttavia è necessario interporre uno stadio di *buffer* in modo da non assorbire corrente dalla presa centrale del potenziometro.

La tensione su quel nodo è direttamente connessa al circuito di controllo del servo motore e dunque assorbendo corrente si degrada la stima del valore dell'angolo.

Con i valori scelti per il partitore di tensione come mostrato in Figura 10 ($10k\Omega$ e $15k\Omega$) la tensione letta dall'ADC risulta:

$$V_{ADC} = V_{pot} \cdot \frac{10k}{10k + 15k} = 5V \cdot \frac{\alpha_{shaft}}{360^\circ} \cdot \frac{1}{2.5} = \frac{\alpha_{shaft}}{180^\circ}$$



Figura 11: Servo motore con modifica al potenziometro

2.4 ESP8266

Il modulo ESP8266 è responsabile della connettività tramite WiFi. La scheda, di dimensioni ridotte, consiste in un *chipset* WiFi a basso costo completo di un intero *stack* TCP/IP con possibilità di essere usato autonomamente grazie al potente microcontrollore integrato. L'hardware è piuttosto performante. Il cuore del modulo è rappresentato dal processore RISC a 32bit Xtensa LX106 prodotto da Tensilica ed operante a 80MHz.

Il modulo è prodotto da Espressif, produttore cinese con sede a Shanghai.

È molto diffuso tra i *makers* per il suo prezzo, per la facilità d'uso e per la possibilità di estendere e modificare il *firmware* del dispositivo. Questo fa sì che esista un'ampia documentazione in rete riguardo al funzionamento del modulo e ai comandi che accetta. Sono inoltre disponibili molti esempi di software già testato per facilitarne l'utilizzo.

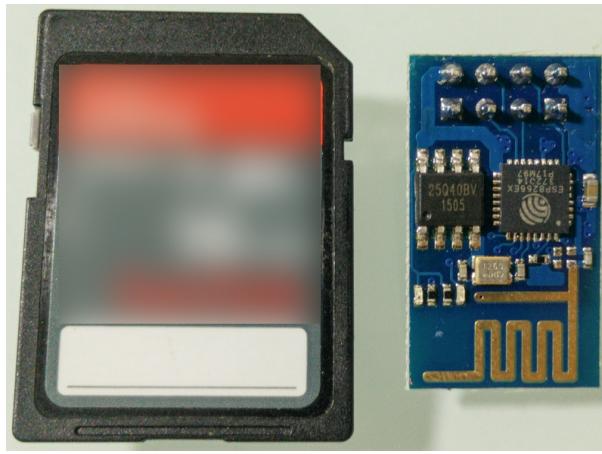


Figura 12: Modulo ESP8266 a fianco ad una scheda SD

Il modulo si interfaccia al microcontrollore tramite una porta seriale operante a 115200baud ed è comandato tramite comandi di tipo AT (Hayes) tipici dei *modem* su porta seriale [9].

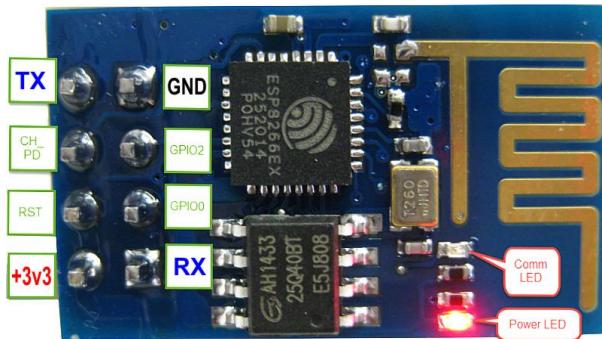


Figura 13: Connessioni per il modulo ESP8266

All'avvio del sistema il modulo produce una rete WiFi *ad hoc* a cui va a connettersi il dispositivo che vuole controllare la mano. I comandi sono trasmessi tramite una connessione TCP all'indirizzo 192.168.4.1 e porta 333.

Il modulo funge in questo caso solo da ponte tra WiFi e porta seriale. Il *parsing* e l'invio dei comandi Hayes è eseguito dall'ATxmega.

3 Architettura software

Il firmware del microcontrollore è stato scritto in C ed il progetto è fornito di un Makefile che si occupa di compilare i vari *file* sorgente e creare l’immagine da scaricare sul dispositivo. È inoltre prevista la possibilità di lanciare una suite di test per il firmware e di eseguire il *flash* della memoria dell’ATxmega.

Nello sviluppare il software si è cercato di suddividere il codice in vari blocchi, il più ortogonali possibili tra loro. I vari blocchi comunicano tra di loro e con l’hardware tramite interfacce ben definite. La Figura 14 mostra come è stato organizzato il codice sorgente. Si sono costruiti livelli di astrazione sempre più elevati a partire dall’hardware e da *application notes* fornite dalla Atmel.

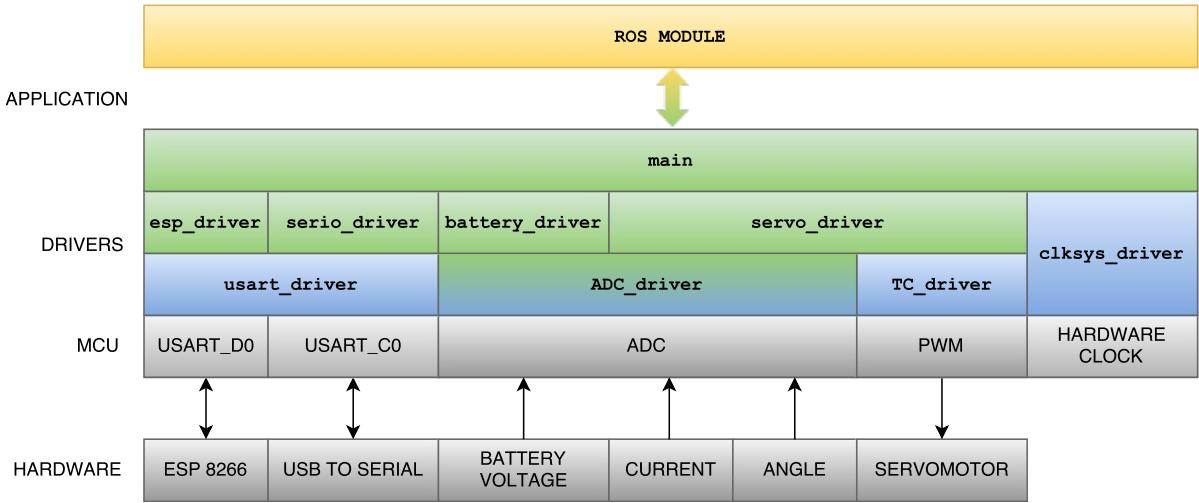


Figura 14: Architettura software. In grigio è indicato l’hardware, in blu il *software* ricavato da *application notes* della Atmel, in verde i contributi del presente lavoro. Il modulo **ADC_driver** nasce come *application note* ma è stato pesantemente adattato al presente progetto.

A fare da collante tra i vari moduli è il blocco **main**. Questo modulo è il punto di avvio del firmware. È responsabile dell’inizializzazione dell’hardware e del *parsing* e dell’esecuzione dei comandi ricevuti via WiFi o porta seriale. Dopo l’inizializzazione il codice entra in un loop infinito in cui attende tramite *polling* la ricezione di un comando, lo esegue e invia una risposta. L’esecuzione di tale loop viene interrotto nel momento in cui viene ricevuto un *interrupt* da parte dell’hardware. Una volta eseguita la richiesta della periferica l’esecuzione riprende dove era stata interrotta. Un ulteriore vantaggio derivante dalla scelta di questa famiglia di microcontrollori è dato dalla possibilità di assegnare una priorità ai vari *interrupt*, garantendo un’efficace distribuzione della potenza di calcolo tra i vari *task* in esecuzione.

Il modulo **main** comunica con un pacchetto ROS installato su un calcolatore, da cui riceve i comandi. Tale pacchetto non è ancora stato scritto, sono tuttavia disponibili degli applicativi *stand-alone* scritti in C che realizzano le funzionalità richieste. Visto il linguaggio di programmazione utilizzato, per integrare il presente lavoro con ROS è sufficiente progettare e realizzare i *topic* ed eventuali servizi per gestire la mano.

Segue una disamina dei vari blocchi che sono stati scritti per l’applicazione qui discussa. Per i blocchi non elencati si rimanda alle *application notes* fornite dal produttore del microcontrollore [10].

3.1 esp_driver

Questo blocco è responsabile dell'inizializzazione, della ricezione e dell'invio dei dati tramite il modulo ESP8266. È qui che sono gestiti i comandi AT del modulo.

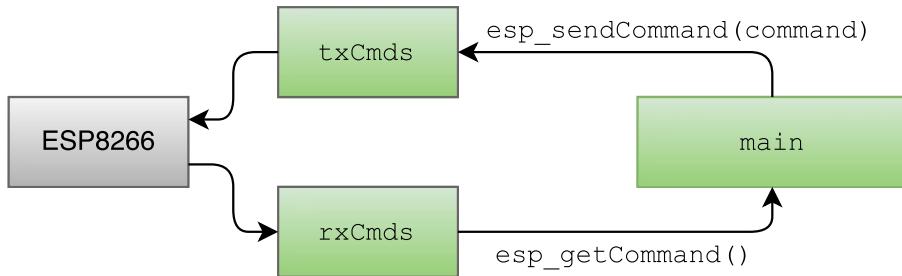


Figura 15: Schema a blocchi per il driver dell'ESP8266

In fase di ricezione tramite una macchina a stati finiti i comandi ricevuti sono estratti a partire dall'*output* del modulo. I dati così ottenuti sono accumulati in un *buffer* circolare chiamato **rxCmds**.

Quando il **main** chiama la funzione `esp_getCommand()` il comando più vecchio viene estratto da tale buffer. Tale funzione può operare sia in modalità *blocking* che *non-blocking* sulla base di un *flag* booleano che riceve come argomento. In modalità *blocking* la funzione controlla se il *buffer* in ricezione è vuoto e in caso affermativo resta in attesa di un nuovo comando, bloccando la funzione chiamante. In modalità *non-blocking* invece quando non ci sono più nuovi comandi nel buffer viene comunque restituito l'ultimo comando ricevuto, evitando così di fermare l'esecuzione del programma.

La funzione `esp_sendCommand(command)` deposita il comando ricevuto come parametro nel *buffer*, anch'esso circolare, **txCmds**. Sarà poi una *routine* di *interrupt* ad incapsulare i dati all'interno di comandi AT da inviare al modulo WiFi.

La dimensione del *buffer* è selezionabile in fase di compilazione agendo sul parametro `COMMAND_QUEUE_SIZE` presente nel file `include/esp_driver.h`. Attualmente tale dimensione è impostata ad 8 ma può essere aumentata ulteriormente visto la dimensione generosa della RAM del microcontrollore.

Si è scelto di usare un *buffer* circolare dove i comandi più vecchi vengono scartati per minimizzare l'uso di risorse hardware e perché non è di particolare importanza l'esecuzione di tutti i comandi ricevuti. Vista la grande disponibilità di banda offerta dalla connessione WiFi e l'esigua dimensione dei comandi è possibile ripetere l'invio finché dovesse rivelarsi necessario. Per confermare l'esecuzione di un comando la scheda risponde con un ACK rendendo noto all'applicazione se il comando è stato o meno scartato.

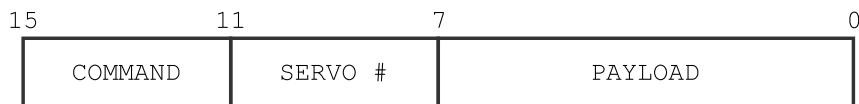


Figura 16: *bitfields* per i comandi inviati via WiFi

Per facilitare il *parsing* i comandi sono lunghi 2Byte così suddivisi:

- I 4bit più significativi identificano il comando da eseguire
- I 4bit successivi identificano il servomotore a cui il comando deve essere applicato (sono ignorati se il comando non prevede di identificare un servomotore)
- Gli 8bit rimanenti sono il *payload*, il cui significato cambia in relazione al comando

La struttura dati che descrive un comando è contenuta nel file `include/board.h` e la sua dichiarazione è la seguente:

```
union wifiCommand
{
    struct
    {
        uint8_t command : 4;
        uint8_t servo : 4;
        uint8_t data;
    } field;
    uint16_t raw;
};
```

Avendo organizzato così la struttura dati la lettura e la creazione dei comandi è immediata, garantendo la leggibilità del codice. I dati ricevuti tramite WiFi sono depositati nella variabile `raw` e grazie all'uso di una `union` per leggere il numero di servomotore si può scrivere `cmd.field.servo`.

I comandi attualmente implementati sono:

- **SET_MODE** per impostare la modalità operativa del controllo dei servo motori (maggiori dettagli a riguardo saranno forniti nella sezione dedicata)
- **SET_ANGLE** per impostare l'angolo desiderato di un certo servo motore
- **SET_CURRENT** per impostare la massima corrente assorbita da un dato servomotore
- **SET_SPEED** per impostare la velocità di rotazione del servo motore
- **GET_ANGLE** per ricevere l'informazione dell'angolo effettivo in cui si trova l'alberino del motore
- **GET_CURRENT** per conoscere la corrente attualmente assorbita
- **GET_SPEED** per conoscere se il servo motore si sta muovendo o meno e a che velocità. Questo comando fornisce anche un'indicazione della direzione dello spostamento

3.2 serio_driver

Questo componente gestisce la comunicazione tramite la porta seriale che a sua volta è connessa via USB al calcolatore che andrà ad inviare comandi alla scheda.

L'obiettivo di questo driver è quello di essere compatibile con le funzioni della libreria standard di C `printf()` e `scanf()` per gestire l'*input* e l'*output* del programma. È Stato anche uno dei primi moduli scritti vista la sua grande utilità in fase di *debug*.

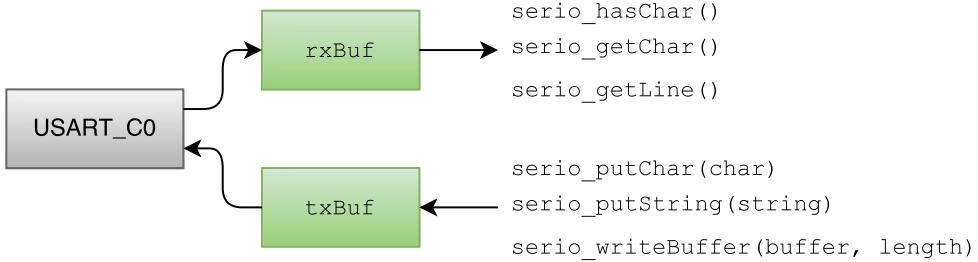


Figura 17: Schema a blocchi per il driver della porta seriale

Anche in questo caso per comunicare con la porta seriale si sono usati dei *buffer* circolari: **txBuf** per i dati da trasmettere e **rxBuf** per i dati ricevuti.

Rispetto alla soluzione fornita da `esp_driver` in questo caso il tipo di dati memorizzato nel buffer è `char` anziché `wificommand` e il driver cerca quanto più possibile di non scartare vecchi dati quando i buffer sono pieni. Questo comportamento è possibile solo in fase di trasmissione dove l'esecuzione si blocca finché **txBuf** non è vuoto. In fase di ricezione se **rxBuf** è pieno sarà il modulo hardware stesso a scartare i nuovi caratteri in arrivo.

La dimensione dei buffer è selezionata modificando il valore di `BUFFER_SIZE` all'interno del file `include/serio_driver.h`. Attualmente il valore di 16 si è rivelato un ottimo compromesso tra utilizzo di memoria e dati persi. In condizioni di funzionamento normale nessun carattere viene scartato.

Il modulo include alcune funzioni utili per scrivere direttamente sulla porta seriale e funzioni utili per interfacciarsi con `printf()` e `scanf()`.

3.3 adc_driver

Questo blocco gestisce le acquisizioni tramite il convertitore analogico digitale del microcontrollore. Nasce come *application note* fornita dalla Atmel, è stato pesantemente modificato per permettere di acquisire da più ingressi in sequenza.

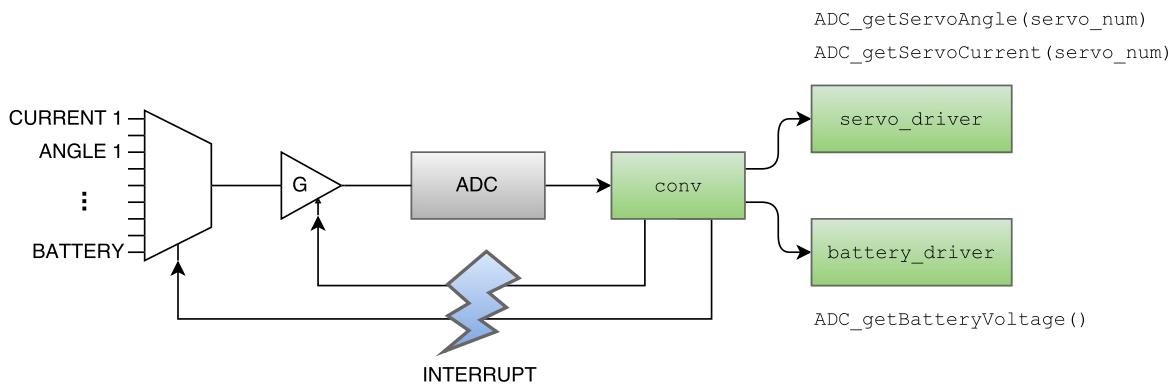


Figura 18: Schema a blocchi per il driver del convertitore analogico-digitale

Essendo infatti l'ADC unico mentre gli *input* sono undici (5 per la corrente, 5 per l'angolo e uno per la tensione della batteria) è necessario che la periferica esegua una scansione dei vari input.

Il convertitore è impostato in modo tale da generare un *interrupt* al termine di ogni conversione. Nella routine di *interrupt* il valore acquisito è salvato in memoria e sono

cambiate le impostazioni del convertitore (ingressi e guadagno) in modo da preparare una nuova conversione per l'input successivo. In questo modo un singolo convertitore è in grado di emulare l'effetto di undici convertitori operanti in parallelo sui vari ingressi. Quello che si riduce è la frequenza di acquisizione che viene suddivisa tra gli *input*. Tuttavia non si hanno particolari requisiti sulla frequenza di campionamento perciò questa modalità permette di operare senza un convertitore esterno, contenendo costi e l'area occupata sulla scheda.

Le funzioni `ADC_getServoAngle()`, `ADC_getServoCurrent()`, `ADC_getBatteryVoltage()` prelevano dalla memoria l'ultima acquisizione corrispondente al dato richiesto ed effettuano una conversione tra unità di misura ove necessario.

3.4 battery_driver

Questo modulo è responsabile della gestione di tutto ciò che concerne la gestione dell'alimentazione e della potenza. In fase di inizializzazione crea una *routine* di *interrupt* a bassa priorità che viene eseguita ogni secondo. Questa routine ha la funzione di monitorare il livello di carica della batteria letto tramite il convertitore analogico-digitale, di informare l'utente sul suo stato di carica accendendo o meno tre LED presenti sulla scheda e di intervenire se il livello di carica è troppo basso spegnendo i servomotori per non danneggiare la batteria.

La routine di *interrupt* controlla anche la presenza o meno dell'alimentazione esterna. Se l'alimentazione è presente tutti i sistemi sono alimentati a partire da quella tensione e i convertitori *boost* sono spenti. In questo modo la carica della batteria procede indisturbata, senza che i vari sottosistemi ne riducano la corrente di carica.

Finchè la batteria è in carica i tre LED sono accesi e spenti ad ogni esecuzione dell'*interrupt*. Il numero di LED accesi rappresenta lo stato della carica. Il lampeggio si ferma quando la batteria è piena.

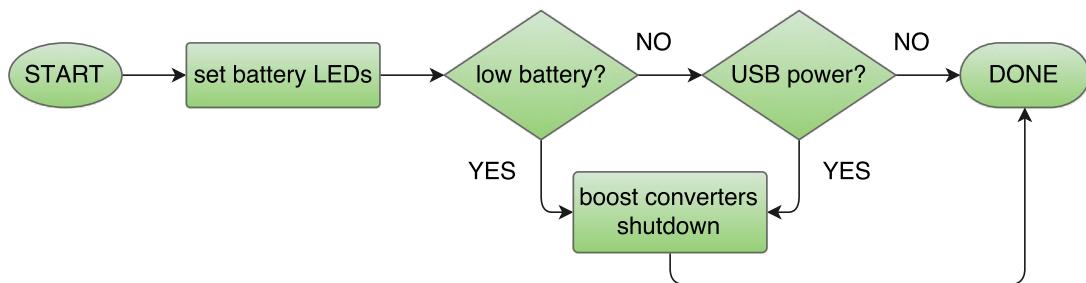


Figura 19: Diagramma di flusso per `battery_driver`

3.5 servo_driver

Questo *driver* va a chiudere l'anello di controllo attorno ai servomotori. Ha come ingresso l'informazione su angolo e corrente forniti da `adc_driver` e da in uscita un segnale PWM che controlla la posizione dell'alberino.

Sulla base del valore fornito alla funzione `servo_setMode()` si individuano tre modalità operative:

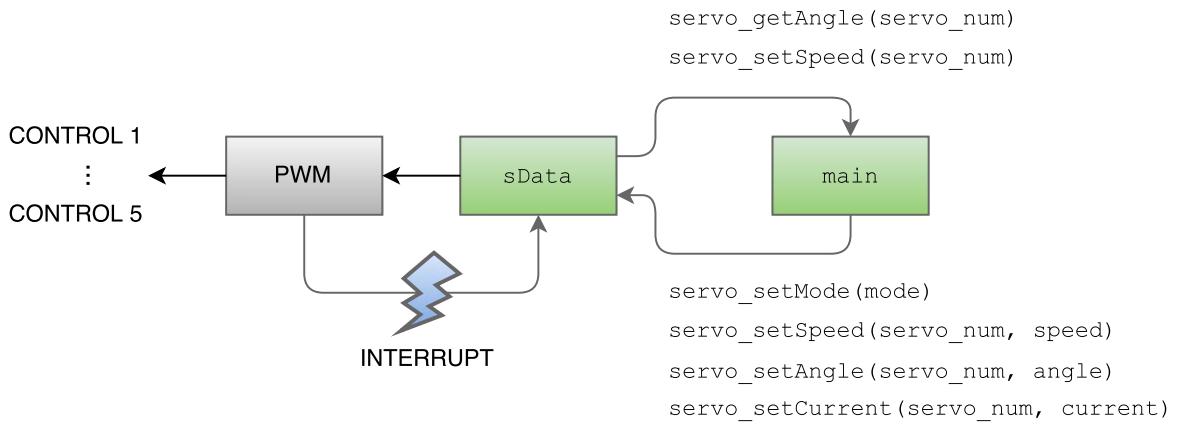


Figura 20: Schema a blocchi per il driver dei servomotori

- **FOLLOW** in cui i servo motori, pur essendo alimentati non ricevono alcun segnale di controllo. L'operatore è libero di muovere le dita della mano che asseggiano lo spostamento richiesto.
- **ANGLE** in cui i servo motori si portano ad un certo angolo con una determinata velocità. Questi dati sono ricevuti come parametri da `servo_setAngle()` e `servo_setSpeed()`. Quando l'angolo è raggiunto la posizione viene tenuta. Se la corrente assorbita supera la soglia impostata con `servo_setCurrent()` il servo motore procede leggermente in direzione opposta alla precedente, nel tentativo di ridurre l'assorbimento di corrente probabilmente dovuto alla presenza di un ostacolo lungo il percorso.
- **HOLD** dove la mano cerca di stringere un oggetto. In questa modalità la velocità di rotazione è ridotta al minimo. A dominare è l'impostazione della massima corrente. Il servo motore cerca di chiudere la mano finché la corrente assorbita non raggiunge la soglia, nel qual caso torna leggermente indietro. Così facendo l'oggetto da manipolare è stretto con una ben determinata forza.

Oltre ai metodi per impostare corrente, angolo e velocità il driver fornisce i metodi `servo_getSpeed()` che fornisce un'indicazione della velocità effettiva a cui il servo motore sta ruotando e `servo_getAngle()` che fornisce l'angolo che il *driver* sta richiedendo al motore in questione. Si noti che l'informazione fornita da `servo_getAngle()` differisce da quella fornita da `adc.getServoCurrent()` nel fatto che la prima restituisce l'angolo richiesto al servo il quale può essere molto diverso dalla posizione effettiva dell'alberino. Questo è dovuto alla presenza di errori di offset oppure al movimento stesso del motore che si porta da una posizione alla successiva in un tempo finito.

4 Risultati

A partire dallo schema elettrico realizzato in fase di progettazione è stato fatto il *layout* della scheda elettrica che realizza il circuito. La scheda è realizzata su doppia faccia, utilizzando il maggior numero di componenti a montaggio superficiale possibile in modo da rendere compatto il risultato finale.

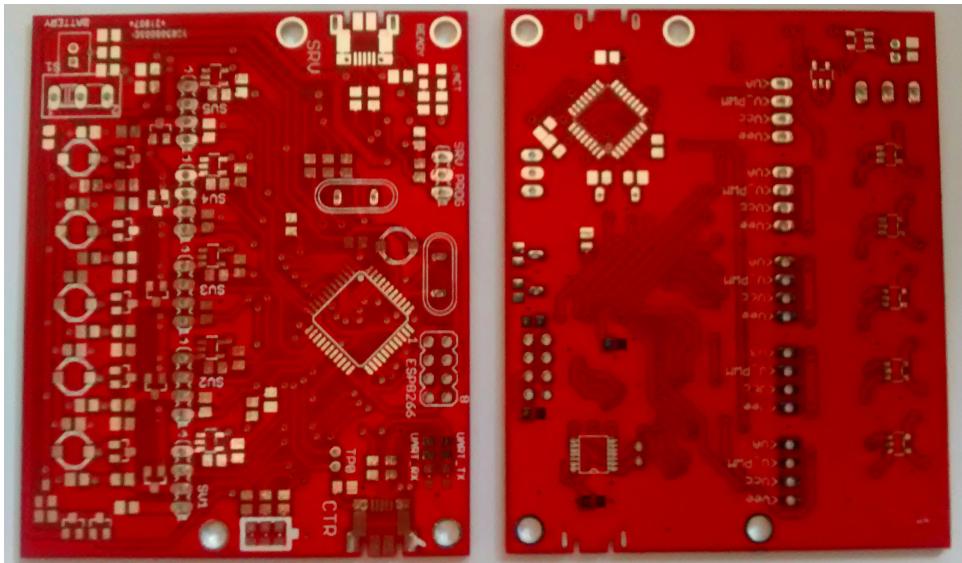


Figura 21: Lato *top* (sinistra) e *bottom* (destra) della scheda elettrica non popolata dai componenti

Dopo aver saldato i componenti è stato programmato il microcontrollore ed è iniziata la fase di *debug* del *firmware*. A tale scopo sono stati scritti in C due eseguibili il cui scopo è quello di inviare comandi via WiFi alla scheda per muovere i servo motori (`tests/testwifi.c`) e per leggere i valori di angolo e corrente (`tests/wifimon.c`).

Il codice sorgente di questi eseguibili andrà a costituire il nucleo centrale del pacchetto ROS per controllare la mano. I test coprono infatti tutti i comandi contemplati dalla scheda per cui è sufficiente integrare il tutto con *topic* e servizi.

Complessivamente il movimento dei servo motori si è rivelato soddisfacente. L'angolo, la posizione e la velocità sono raggiunti e tenuti in maniera corretta.

Per quanto riguarda le letture, soprattutto per quelle di corrente, i valori letti sono caratterizzati dalla presenza di un forte rumore. Ciò è dovuto al fatto che l'assorbimento di corrente dei motori non è costante ma presenta delle variazioni piuttosto consistenti attorno al valore medio. Le variazioni di corrente si riflettono anche sulla tensione di alimentazione che a loro volta si propagano, tramite il potenziometro dell'alberino, sulla lettura dell'angolo.

In conclusione quindi, partendo da una mano robotica in grado di compiere semplici movimenti come l'apertura e la chiusura delle dita, si è ottenuta una piattaforma hardware completa ed estendibile, in grado di permettere manipolazioni complesse senza richiedere costosi servo motori per applicazioni industriali.

La compatibilità con il *framework* ROS garantisce una facile integrazione all'interno di progetti robotici già esistenti oppure nuovi.

4.1 Sviluppi futuri

Come già spiegato la principale estensione da affrontare per il presente progetto è la scrittura di un modulo ROS per controllare la mano. Tutto il codice necessario è già presente all'interno del *repository* del progetto. Quello che manca è l'adattamento di tale codice al *framework*.

Un altro aspetto da affrontare è il miglioramento della qualità dei dati acquisiti tramite l'ADC. Sarebbe interessante valutare l'impatto di un filtro passa-basso per estrarre il valore medio del segnale. Tale filtro può essere realizzato sulla scheda in ambito analogico oppure nel dominio digitale. Le due soluzioni possono essere confrontate per qualità dei risultati, area occupata sulla scheda e potenza di calcolo richiesta.

Al momento della scrittura del presente elaborato inoltre, collegando la porta USB ad un calcolatore, il *prompt* che la scheda fornisce è molto semplice e non è ancora possibile inviare comandi con questo metodo. Sarebbe interessante valutare la scrittura di un'interfaccia partendo da zero o adattare progetti preesistenti di interpreti di lisp e i suoi dialetti, Lua oppure javascript.

La scheda attualmente realizzata infine rappresenta solo un prototipo. È necessario consolidare il progetto ad esempio rendendolo più robusto a urti e vibrazioni e rimuovendo il programmatore sulla scheda sostituendolo con una qualche forma di *bootloader*.

Riferimenti bibliografici

- [1] Inmoov website. <http://inmoov.fr/>. Ultimo accesso: 25 luglio 2016.
- [2] Creative commons. <https://creativecommons.org/>. Ultimo accesso: 25 luglio 2016.
- [3] Open source definition. <https://opensource.org/docs/osd>. Ultimo accesso: 25 luglio 2016.
- [4] Inmoov fablab pisa. <http://www.fablabpisa.org/?p=855#more-855>. Ultimo accesso: 25 luglio 2016.
- [5] Inmoov. <http://robinhsieh.com/>. Ultimo accesso: 25 luglio 2016.
- [6] Meet the robot giving hospitalised children superpowers. <https://www.theguardian.com/sustainable-business/2015/feb/06/robots-for-good-hospitalised-children-superpowers>. Ultimo accesso: 25 luglio 2016.
- [7] Avr xmega microcontrollers. http://www.atmel.com/products/microcontrollers/avr/AVR_XMEGA.aspx. Ultimo accesso: 25 luglio 2016.
- [8] R.W. Erickson and D. Maksimovic. *Fundamentals of Power Electronics*. Power electronics. Springer US, 2001.
- [9] Esp8266 at wiki. https://github.com/espressif/ESP8266_AT/wiki. Ultimo accesso: 25 luglio 2016.
- [10] Avr xmega microcontrollers - documents. http://www.atmel.com/products/microcontrollers/avr/avr_xmega.aspx?tab=documents. Ultimo accesso: 25 luglio 2016.