Ceriani Simone e Marco Dalli

Robotizzazione carrozzina elettrica per disabili

Introduzione al progetto

Nell'ambito del corso di “Laboratorio di Intelligenza Artificiale eRrobotica” si sono studiate varie possibilità per interfacciare ad un computer una carrozzina elettrica per disabili, al fine di controllarne il moto. Dopo aver verificato quale di queste erano effettivamente percorribili, si è proceduto a realizzare tale interfaccia.

La carrozzina è denominata “Rabbit” e le sue caratteristiche la rendono adatta ad ambienti interni ed esterni. Il sistema di motorizzazione e controllo è prodotto dalla DynamicControls [1] ed è chiamato DXSystem. Tale sistema comprende sia il comando utente (con il joystick DX Dolphin Master Remote – DX-REM34) che gli attuatori e controllori, di cui però non ci siamo occupati.

Ricerche e prove effettuate

Inizialmente abbiamo cercato di reperire quante più informazioni possibili sul sistema di guida DXSystem. Il produttore non ha acconsentito a fornirci le specifiche del protocollo di comunicazione tra il joystick e gli attuatori (chiamato DXBus). Cercando su internet ci è stato possibile trovare documenti prodotti da due università che avevano già lavorato con il DXBus, che purtroppo non ci hanno aiutato a capire come poterci interfacciare direttamente ad esso con un PC. Le informazioni reperite dai documenti [2] e [3] ci hanno indicato che il DXBus è sostanzialmente basato sul protocollo standard CAN, ma la DynamicControls ha apportato modifiche ad alcuni valori di tensione. I dispositivi CAN sono costosi e, non conoscendo precisamente se un dispositivo standard avrebbe potuto funzionare (e soprattutto non si sarebbe danneggiato) se collegato al DXBus, abbiamo preferito abbandonare questa soluzione.

Smontando il joystick abbiamo però appreso, con l'uso di un semplice tester, che la leva del joystick genera dei valori di tensione, riferiti alla massa presente nel joystick stesso, abbastanza semplici. Abbiamo quindi pensato di sostituire la leva del joystick con un circuito che potesse “simulare” tali valori di tensione, ottenendo così il controllo della carrozzina. Nello schema che segue è riportato il dettaglio delle tensioni rilevate sul connettore che collega la leva del joystick alla scheda di comando dell'intero sistema joystick.

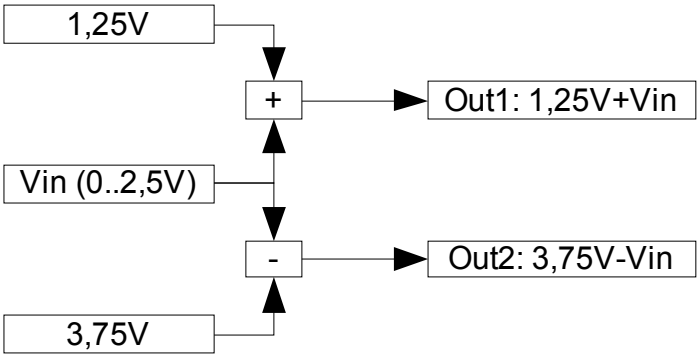
(Connettore visto dall'alto)
1- Alimentazione (+5V)
2- Massa di riferimento
3- Comando marcia 1
4- Comando marcia 2
5- Comando sterzo 1
6- Comando sterzo 2
X- Pin non connesso

Il connettore è a 7 piedini, di cui uno non utilizzato. I colori riportati nel disegno rispettano la colorazione dei fili collegati al connettore della leva. I primi due portano l'alimentazione a 5 Volt e la massa alla leva. I pin 3 e 4 quando la leva è in posizione di riposo hanno una tensione di 2,5 Volt. Quando si porta la leva avanti la tensione sul pin 3 cresce fino a 3,75V e sul pin 4 diminuisce fino a 1,25V. Portando invece la leva indietro si ottiene il comportamento opposto, ovvero il pin 3 diminuisce il valore di tensione fino ad un minimo di 1,25V e il pin 4 aumenta fino a 3,75V. I pin 5 e 6 hanno lo stesso comportamento ma si riferiscono al movimento della leva a destra (pin 5 aumenta e pin 6 diminuisce) e sinistra.

Si noti che i termini “comando marcia” e “comando sterzo” non sono del tutto appropriati in quanto la carrozzina non è dotata di un vero sistema di sterzo, ma di due motori che comandano le ruote posteriori in modo indipendente. Le ruote anteriori sono libere di muoversi e non sono controllate in alcun modo.

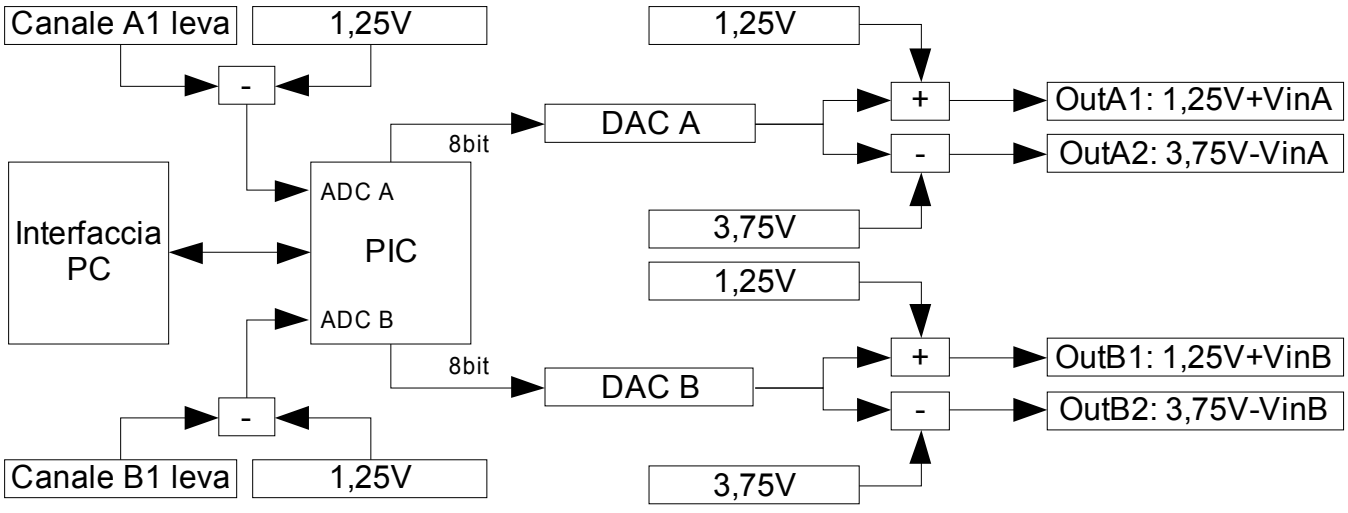
Studio del comando ad alto livello

Per pilotare ogni singolo canale di comando della carrozzina è necessario simulare i valori di tensione presenti sui due pin del connettore che lo riguardano. Supponendo di ottenere una tensione di ingresso (che chiameremo Vin) variabile tra 0V e 2,5V, sarà necessario sommarle 1,25V per ottenere il primo valore di riferimento, e sottrarla da una tensione di riferimento di 3,75V per ottenere il secondo comando del moto. Nel seguente schema mostriamo uno schema a blocchi di alto livello del circuito da realizzare per il comando di ogni singolo canale:



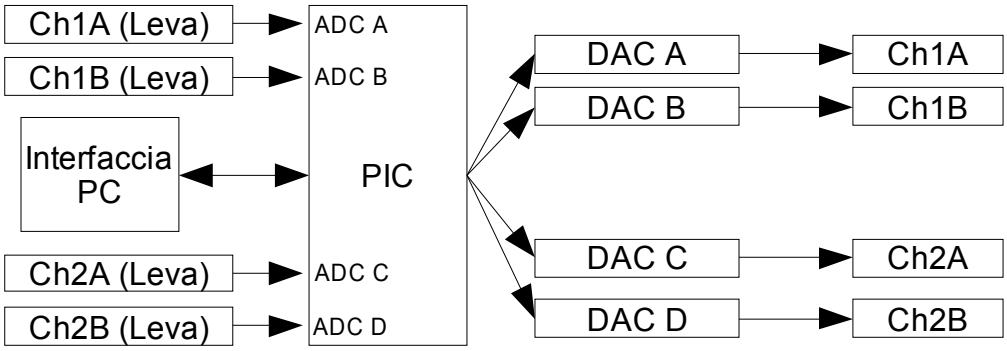
Riferendoci ad esempio al comando per la marcia, l'uscita Out1 sarà da collegare al pin3 e la Out2 al pin4, di modo che con Vin massima (2,5V) si ottiene il massimo valore di avanzamento e con Vin a 0V si ottiene la retromarcia alla massima velocità. Il valore Vin=1,25V permetterà di mantenere ferma la carrozzina, in quanto Out1 e Out2 saranno contemporaneamente a 2,5V. Analogamente per lo sterzo dovremo collegare Out1 al pin5 e Out2 al pin6.

Per permettere la connessione della carrozzina ad un computer è necessario che la tensione Vin sia generabile da un convertitore DAC (Digital Analogic Converter), che riceverà un valore digitale e produrrà in uscita un valore di tensione analogico compreso tra 0V e 2,5V. Inoltre, ritenendo opportuno non scollegare completamente la leva del joystick, ma permettere un controllo da parte dell'utente, è necessario che la tensione prodotta dalla leva sia "letta" da un ADC (Analogic Digital Converter) e comunicata al computer che potrà, con opportune politiche, comunicare il valore del comando da impostare alla carrozzina. Per fare ciò ci è sembrato opportuno utilizzare un microprocessore general purpose che si occupasse della trasmissione e ricezione dei dati dal computer, della lettura dei valori di tensione dalla leva del joystick e della produzione del valore digitale che servirà poi al comando della carrozzina. Dalla ricerca tra i prodotti della Microchip [4], in particolare tra i microprocessori PIC, abbiamo trovato molti prodotti che integrano già la conversione analogica digitale, ma non quella digitale analogica. Infatti tutti i dispositivi che avevano al loro interno un DAC non erano adatti ai nostri scopi, in quanto era presente un solo DAC, mentre per il nostro lavoro ne servivano almeno due (uno per ogni canale). L'interfaccia con il PC dovrà essere realizzata tramite l'uso della porta seriale. Lo schema ad alto livello risulta il seguente:



Dopo una più approfondita ricerca dei componenti da utilizzare, ci si è resi conto che era possibile utilizzare un PIC con (almeno) 4 ingressi muniti di convertitori analogico/digitale ed effettuare la differenza tra i valori di tensione di ogni singolo canale direttamente a livello software. Anche per lo stadio di uscita si è preferito usare direttamente 4 convertitori digitale/analogici che, opportunamente comandati a livello software, generassero le due coppie di tensioni differenziali. Quest'ultima scelta è stata favorita dalla disponibilità di chip DAC ad ingresso seriale, comandabili quindi tramite l'uso di pochi collegamenti.

Lo schema ad alto livello che si è effettivamente utilizzato risulta quindi essere il seguente:



Studio del software necessario

Il software da realizzare si divide in due parte che risiederà sul PIC e quella sul PC. Studiamo ad alto livello le due parti in modo separato e le comunicazioni che interverranno.

Innanzitutto bisogna considerare che il sistema deve essere in grado di funzionare anche quando il PC non è acceso o, per qualche motivo, non comunica nulla al PIC. Le uscite digitali del PIC dovranno fare in modo che la carrozzina risponda ai comandi della leva, senza interventi del sistema automatico. Il sistema di comunicazione farà in modo che il PC riceva in ingresso la posizione (convertita in digitale) che l'utente impone sulla leva e comunichi in uscita i valori calcolati dal sistema automatico.

Si dovrà provvedere a inserire un campo di controllo (es. checksum) per validare i dati scambiati e un campo di comando che permetta di cambiare la modalità di funzionamento della carrozzina da automatico a manuale e viceversa.

All'accensione della carrozzina il modo di funzionamento dovrà essere manuale, al fine di evitare possibili movimenti indesiderati. Solo successivamente sarà possibile passare alla modalità automatica, su esplicito intervento dell'utente (tramite pressione di un pulsante). Il passaggio da modalità automatica a manuale sarà invece possibile in tre modi:

- esplicita richiesta dell'utente tramite pressione del pulsante
- richiesta del PC di abbandonare la modalità automatica
- timeout nel caso il PC non comunichi nessun comando valido entro un tempo prestabilito

Comunicazione tra microprocessore PIC e PC

I dati che vanno dal PIC al PC sono i due valori di posizione letti dalla leva. Possiamo dunque strutturare una trama molto semplice che prevede l'invio ogni volta di quattro byte, il primo usato come sincronizzazione e controllo della modalità di comando in uso (automatico o manuale). Utilizziamo il valore F0h per indicare l'inizio della trama con modalità manuale, mentre F1h per l'inizio della trama con modalità automatica. Il secondo byte della trama rappresenta il valore letto dal comando della marcia e il terzo quello dello sterzo. Il valore del quarto byte viene calcolato come la somma dei tre byte precedenti a modulo 255. Riportiamo qui sotto lo schema della trama:

F[0|1]h	Leva1	Leva2	Chk

Il formato della trama comunicata dal PC al PIC è identico a quello sopra indicato, dove ovviamente i valori del secondo e terzo byte non rappresentano la lettura della leva ma il valore da mettere in uscita per il controllo automatico della carrozzina.

Qualora la modalità di funzionamento fosse manuale, i valori trasmessi dal PC saranno ignorati.

Software per il microprocessore PIC

Il software del microprocessore dovrà svolgere le seguenti funzioni:

1. Leggere i valori in ingresso agli ADC e sottrarli a coppie per ottenere due valori assoluti.
2. Comunicare al PC i valori calcolati al punto 1
3. Se la modalità è automatica, porre in uscita l'ultimo comando ricevuto dal PC

4. Se la modalità è manuale, porre in uscita i valori calcolati al punto 1
5. Leggere i dati eventualmente comunicati dal PC
6. Controllare la scadenza del timeout di validità dei comandi
7. Controllare eventuali pressioni del pulsante
8. Aggiornare la modalità di funzionamento in base agli eventi rilevati ai punti 6 e 7
9. Ripetere dal punto 1

Software per il PC

In questo momento analizziamo il software di basso livello, che dovrà fornire l'interfaccia con la carrozzina per un sistema di più alto livello.

Le funzioni da svolgere saranno le seguenti

1. Leggere le informazioni provenienti dal PIC e renderle disponibili al livello superiore
2. Accettare valori dal livello superiore e trasmetterli al PIC

E' opportuno che la prima funzionalità sia disponibile come thread che continuamente controlla gli ingressi.

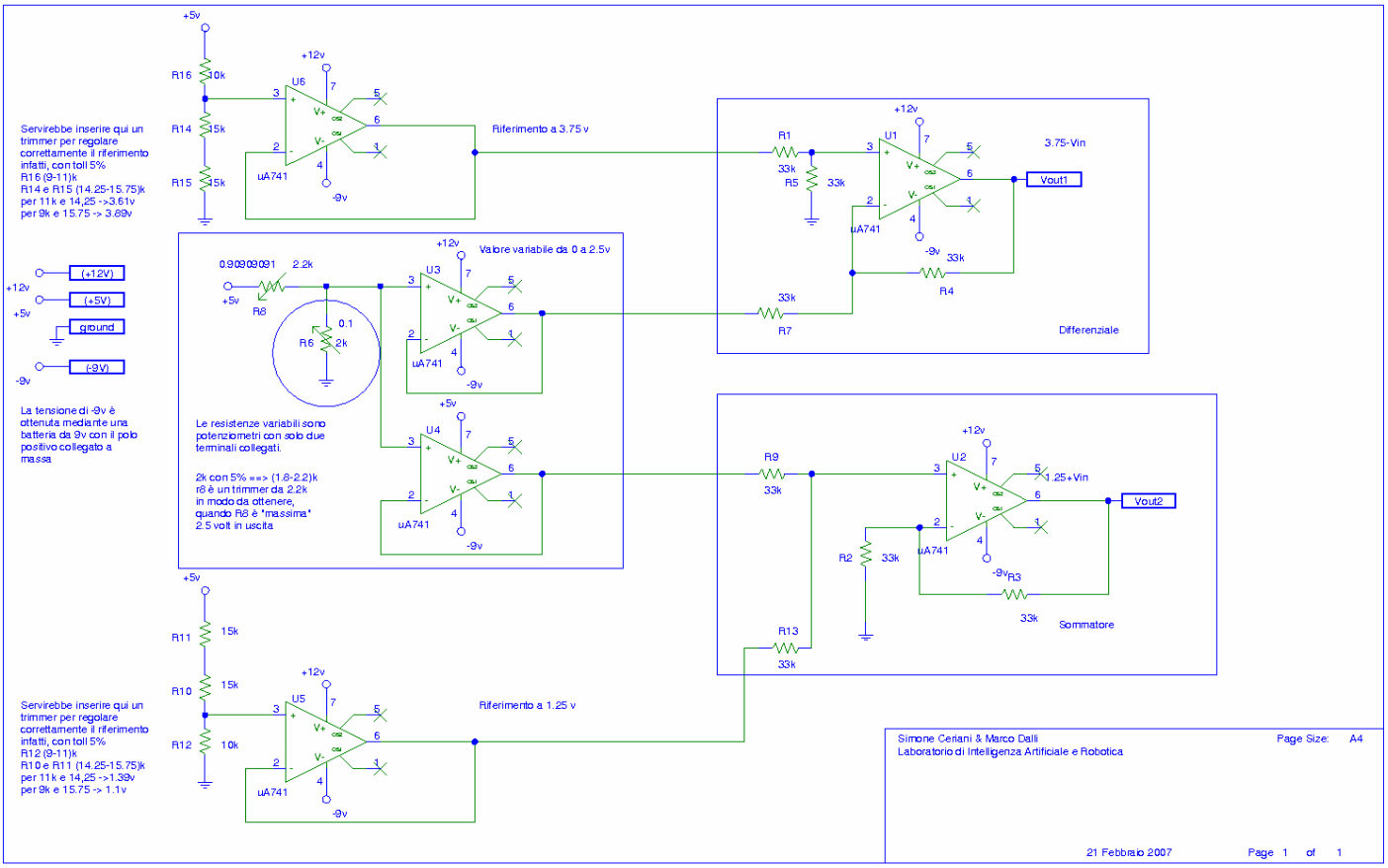
Primo circuito elettronico di comando

Analizziamo ora come è possibile creare un circuito che “simuli”, dato un valore di tensione in ingresso, le due tensioni differenziali necessarie a comandare un singolo canale del joystick. Il valore di tensione in ingresso viene per ora generato con l'uso di un potenziometro, allo scopo di verificare se è possibile percorrere questa strada per controllare la carrozzina.

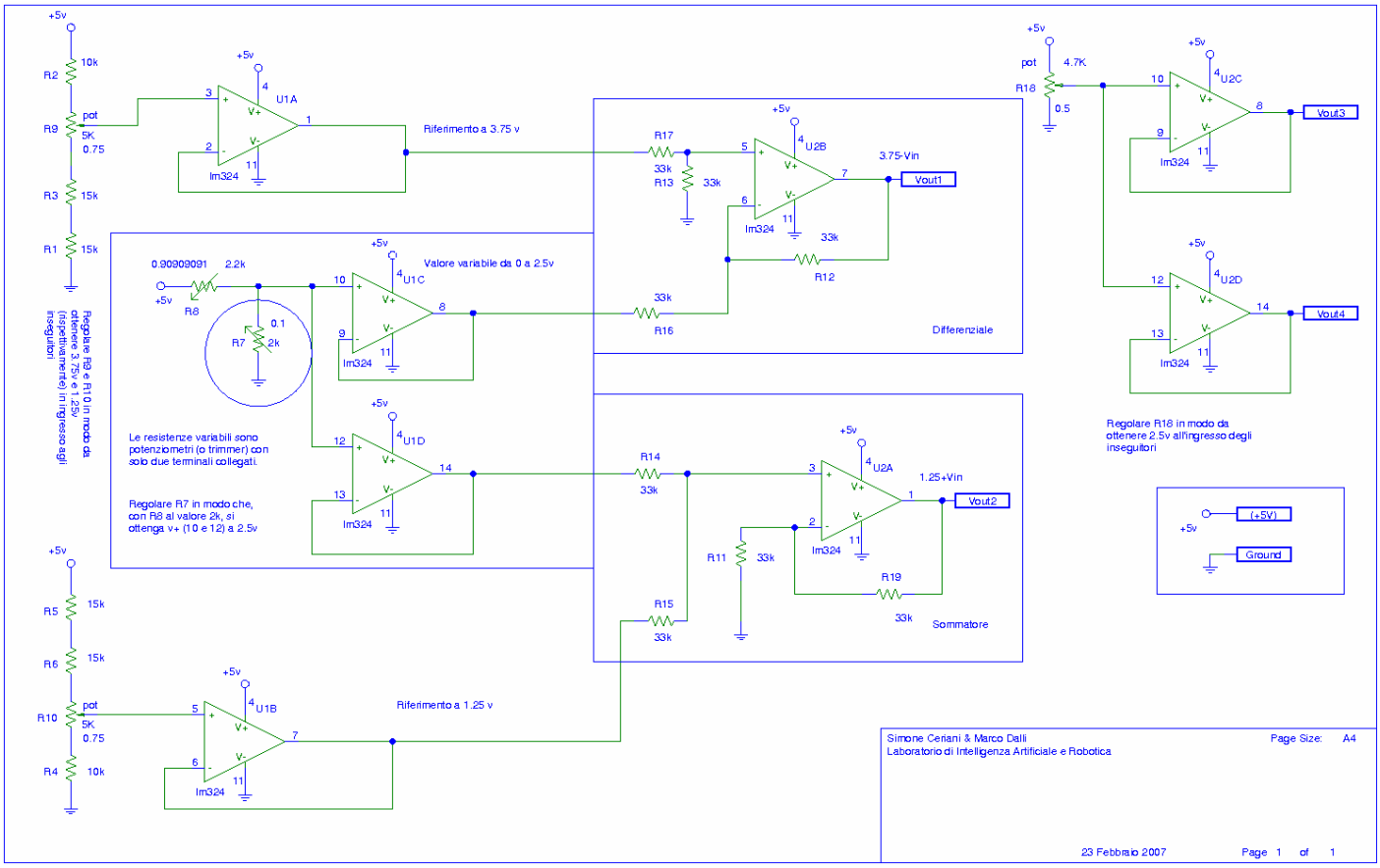
Il potenziometro ha lo scopo di simulare la “posizione” della leva generando una tensione variabile tra 0V e 2,5V e sarà sostituito in seguito con la tensione prelevabile in uscita dal DAC. Utilizzando un circuito sommatore e uno differenziale è possibile generare, dati due valori di riferimento a 1,25V e 3,75V, le due tensioni di uscita.

La presenza di sommatori e differenziali che operano su tensioni analogiche ci ha suggerito l'uso degli amplificatori operazionali per realizzare il circuito elettrico proposto. La scelta iniziale è stata quella di utilizzare come amplificatori operazionali gli ua741 [5], degli integrati molto comuni. Il problema di tali integrati è che necessitano di essere alimentati oltre che con una tensione positiva (ad esempio i 5V disponibili sul connettore) anche con una tensione negativa. Per superare tale problema si è provato a collegare il polo positivo di una comune pila a 9V alla massa di riferimento, ottenendo così una tensione di riferimento di -9V sul polo negativo. Utilizzando inoltre un relè è stato possibile fare in modo che la pila risultasse collegata solo quando la tensione di 5V era effettivamente presente. Tale soluzione, realizzata su una breadboard di prova e testata singolarmente ha permesso di verificare la correttezza del progetto del circuito, ma non è stata testata sulla carrozzina. Infatti è stata subito abbandonata in funzione di una più semplice che non necessita più dell'uso della pila, realizzata con gli integrati lm324 [6]. Essi presentano il vantaggio di non necessitare di un'alimentazione duale e di contenere al loro interno 4 amplificatori operazionali indipendenti.

Qui sotto sono riportati i due schemi che sono stati realizzati. Ci limiteremo a commentare il secondo, in quanto il primo è molto simile e inoltre privo di alcuni accorgimenti che ne hanno migliorato la precisione.



Per realizzare lo schema del primo circuito abbiamo utilizzato PSPICE nella versione studenti. La realizzazione del circuito è invece avvenuta, come già detto, su breadboard.



Il primo operazionale (in alto a sinistra, U1A) è collegato in modo da creare un inseguitore di tensione, di modo che produca, grazie al partitore collegato al morsetto positivo, una tensione di riferimento di 3,75V sull'uscita. Come si può notare, il partitore è realizzato con due resistenze da 15kΩ a valle del punto in cui si preleva la tensione e da una da 10kΩ collegata alla tensione di riferimento di 5V; inoltre è presente un trimmer da 5kΩ. Se le resistenze fossero ideali

(ovvero il loro valore nominale corrispondesse al valore reale) il trimmer non sarebbe necessario, in quanto la tensione prelevata in mezzo al partitore varrebbe $\frac{5V \cdot 30K}{(10k + 15k + 15k)} = 3,75V$. Analogamente anche l'operazionale U1B è un inseguitore che genera la tensione di riferimento a 1,25V. Gli inseguitori realizzati con U1C e U1D servono per disaccoppiare la tensione prelevata all'uscita del potenziometro R7 che, creando un partitore di tensione con R8, svolge la funzione di far variare da 0 a 2,5V il valore di riferimento che servirà per simulare il comando di un canale. Il blocco differenziale e sommatore creano, utilizzando le resistenze tutte di valore uguale, le due tensioni differenziali da collegare al comando della carrozzina.

Questo circuito sarebbe da replicare in alcune sue parti (blocco sommatore e differenziale più aggiunta di alcuni inseguitori) per permettere anche il comando del secondo canale. Per questo primo circuito ci limitiamo a “tenere fermo” il secondo canale di comando, impostando entrambi i valori di tensione che lo riguardano a 2,5V, come realizzato con il trimmer R18 e gli operazionali U2C e U2D.

In realtà questo circuito non è stato effettivamente realizzato nella sua versione “doppia”, in quanto la soluzione effettivamente adottata ha previsto l'uso di 4 DAC che, opportunamente comandati, provvedono alla generazione delle due coppie di tensioni differenziali. L'uso di questo circuito ci ha permesso però di verificare che è possibile controllare la carrozzina simulandone le tensioni della leva con un sistema costruito *ad hoc*.

Dimensionamento dei trimmer

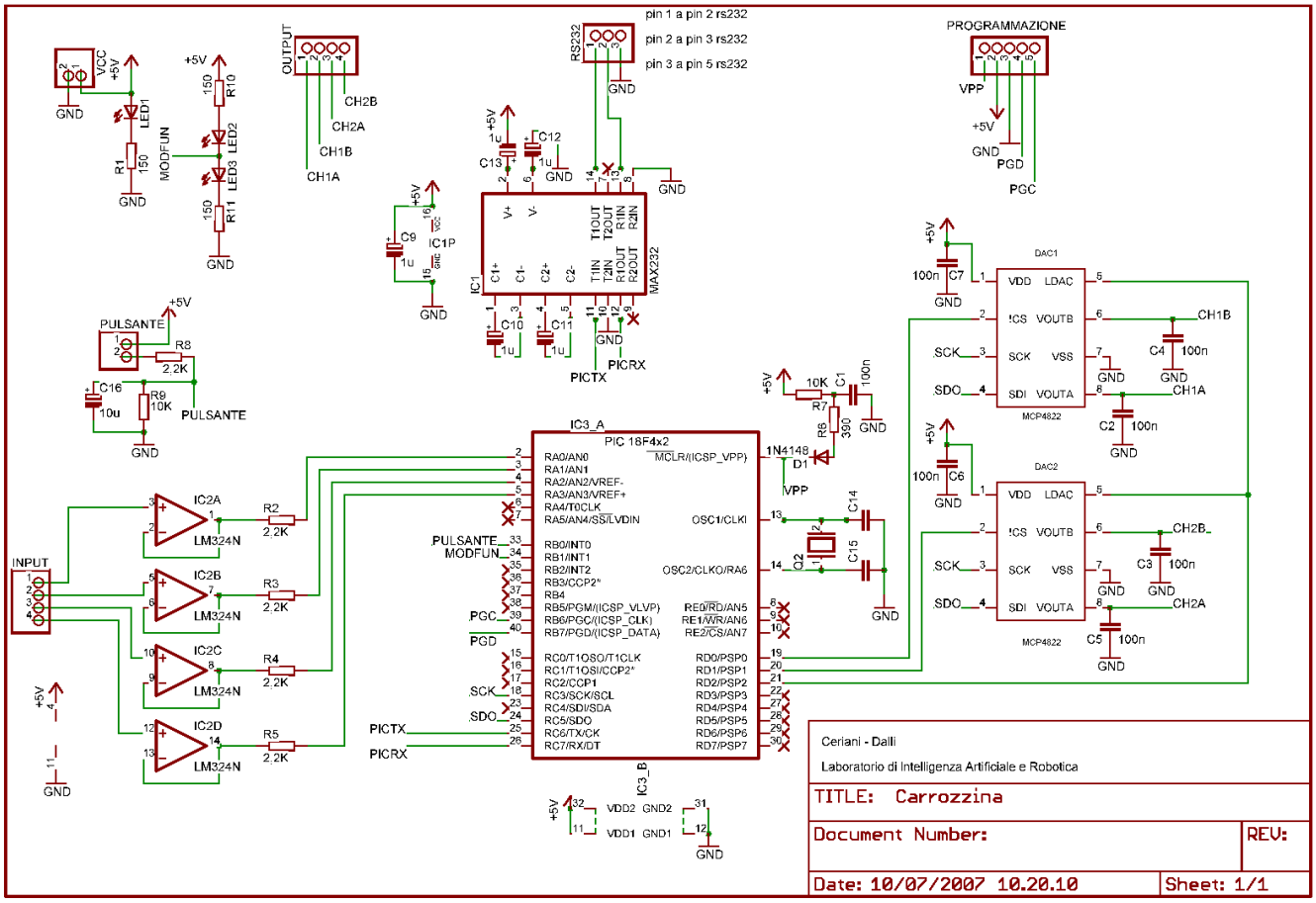
Consideriamo ora il dimensionamento del trimmer R9 che serve per generare correttamente il valore di tensione di 3,75V. Il dimensionamento di R10 è analogo e porta allo stesso risultato, data la “simmetria” della sua configurazione.

A causa della tolleranza delle resistenze che compongono il partitore di tensione (5%) si potrebbero ottenere, nei casi peggiori, valori di riferimento molto differenti. Infatti, con R2 da 9K, R1 e R3 entrambe da 15,75K, si ottiene una tensione di 3,89V, mentre con R2 da 11K, R1 e R3 entrambe da 14,25K, si ottiene una tensione di 3,61V. Il trimmer permette di tarare, con l'uso di un tester, il valore della tensione in ingresso all'inseguitore. Per il dimensionamento del trimmer si è seguita la seguente procedura. Si identifica il trimmer con due parametri: R, il valore di resistenza incognito (in KΩ) e b, un valore reale compreso tra 0 e 1 che indica qual è la regolazione del trimmer. Considerando il primo caso, abbiamo l'equazione $\frac{(28,5 + R \cdot b)}{(39,5 + R)} = 0,75$ da cui si ricava b in fuzione di R: $b = \frac{3 \cdot (2 \cdot R + 3)}{(8 \cdot R)}$. Si impone che valga $0 \le b \le 1$ e si ottiene $R < -1,5 \vee R > 4,5$, dove la prima condizione è assurda e la seconda verificata con una resistenza maggiore di 4,5K. Considerando invece il secondo caso, in cui si ha $\frac{(31,5 + R \cdot b)}{(40,5 + R)} = 0,75$, si ottiene $R < -4,5 \vee R > 1,5$. Combinando le due condizioni si nota come un trimmer da 5K soddisfa le due condizioni imposte.

Il trimmer R8 si dimensiona molto più semplicemente. Considerando infatti R7 di valore 2K e una tolleranza del 5%, si potrebbe ottenere un valore di resistenza variabile tra 1900 e 2100Ω. Il comportamento desiderato è che, quando R7 assume il suo massimo valore di resistenza, la tensione in ingresso all'inseguitore sia di 2,5V. Il valore di resistenza di R8 deve poter essere impostato allo stesso valore di R7, quindi deve valere almeno 2100Ω. Scegliendo un trimmer da 2200Ω si ottiene che, con una tolleranza del 5%, esso vale al minimo 2090Ω. Esso non sarebbe dunque adatto ai nostri scopi, ma sperimentalmente è risultato sufficiente ed è stato scelto perché già disponibile. Questa inesattezza di progetto non comporterà alcun problema in seguito, in quanto il potenziometro R7 sarà rimosso in favore del collegamento con il convertitore DAC.

Circuito di comando realizzato

Riportiamo, e di seguito commentiamo, lo schema del circuito elettronico effettivamente realizzato che rende disponibili tutte le funzionalità previste per il controllo della carrozzina.



Breve panoramica sui componenti utilizzati

Elenchiamo ora i principali componenti utilizzati e la funzionalità che svolgono; per ulteriori dettagli si rimanda ai datasheet delle case costruttrici.

- *LM324*: chip contenente 4 amplificatori operazionali a singola alimentazione (0 – 5V). E' un dispositivo standard, di basso costo e prodotto da numerose industrie. Ulteriori informazioni su [6].
- *MAX232*: interfaccia tra la seriale del PC e il dispositivo UART del PIC. Le due interfacce lavorano a livelli di tensioni differenti e grazie all'uso di questo componente è possibile realizzarne la connessione. Si veda [7] e [8].
- *PIC18F452*: microprocessore prodotto dalla Microchip [4] che integra al suo interno moduli per l'input-output digitale, per l'input analogico con conversione in digitale, timer per il controllo del tempo, modulo UART per la trasmissione e ricezione seriale, interfaccia SPI (Serial Peripheral Interface) per la comunicazione con altri dispositivi. Datasheet reperibile all'indirizzo [9].
- *MCP4822*: chip contenente 2 convertitori digitale-analogico a 12 bit. Ogni convertitore genera una tensione di uscita compresa tra 0 e 4,095 Volt. E' provvisto di un piedino di Chip Select e di uno che avvia la conversione digitale analogica. I valori digitali vengono comunicati al chip tramite l'interfaccia SPI. Datasheet su [10].

Alimentazione

In alto a sinistra nello schema è presente il connettore denominato VCC che alimenta il circuito con una tensione di +5V e un riferimento a massa. Ad esso è direttamente collegato un led (LED1, di colore rosso) indicante lo stato di accensione del circuito.

Input

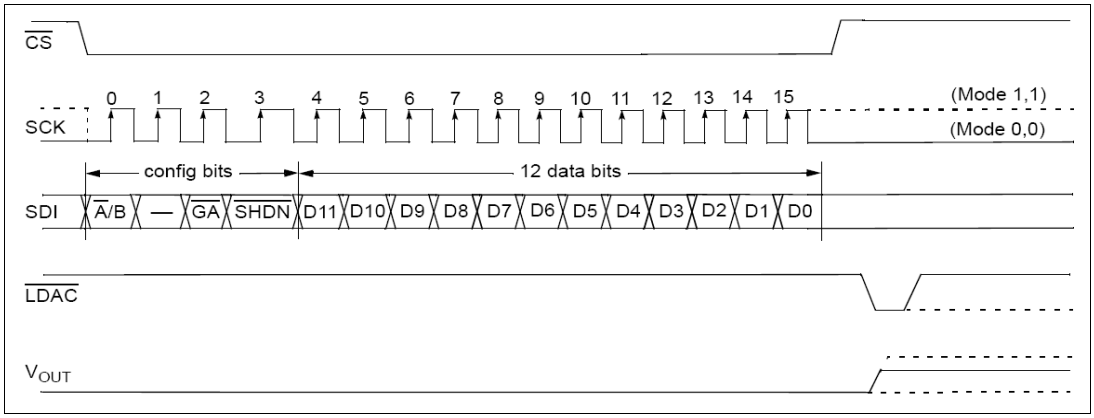
Il connettore di input, sulla sinistra dello schema, preleva i valori di tensione della leva. Per ognuno dei valori viene effettuato un disaccoppiamento del segnale tramite un amplificatore operazionale in configurazione inseguitore. L'uscita di ogni inseguitore viene collegata tramite una resistenza di carico da 2,2KΩ agli ingressi RA0..RA3 del PIC che, opportunamente configurati provvedono alla conversione analogica/digitale. Si noti che i quattro amplificatori sono tutti

integrati nel componente LM324.

Output

Alla destra del PIC si trovano due MCP4822 che si occupano di effettuare la conversione digitale analogica. Ognuno di essi contiene due DAC indipendenti che pilotano le due uscite VOUTA e VOUTB. Per comandare tali integrati è necessario utilizzare l'interfaccia SPI (Serial Peripherical Interface). Per i dettagli riguardo il protocollo e i byte di comando si faccia riferimento al datasheet. I pin del PIC denominati SCK (Serial Clock) e SDO (Serial Data Output) sono collegati rispettivamente a SCK e SDI (Serial Data Input) dei due integrati. E' inoltre necessario utilizzare un segnale di chip select (CS) per ciascun integrato, proveniente da RD0 per il primo chip e RD1 per il secondo. I DAC producono un'uscita corrispondente all'ultimo comando ricevuto solo quando si dà un impulso sul pin LDAC; tale segnale è generato dal PIC su RD2, collegato a LDAC di entrambi i chip, per garantire il sincronismo delle uscite. L'uso dei condensatori C2..C7 è consigliato dal datasheet del MCP4822 ai fini di filtrare le tensioni di alimentazione e quelle di uscita.

Lo schema sottostante, presentato sul datasheet, indica come dev'essere pilotato il DAC utilizzando i segnali sopra citati.



Interfaccia RS232

Nella zona centrale in alto si può vedere il connettore denominato RS232 con 3 pin per la comunicazione via seriale. Esso è collegato al chip MAX232 che si occupa delle conversioni delle tensioni dei segnali tra interfaccia seriale e PIC. I condensatori utilizzati sono specificati dal datasheet del componente. I PIN T1IN e R1OUT sono collegati a RC6 e RC7 del PIC.

Interfaccia Utente

L'interfaccia utente è costituita da 2 LED (oltre a quello di accensione) e da un pulsante. Il pulsante è rappresentato da un connettore a 2 pin collegato all'alimentazione e al pin RB0 del PIC mediante un circuito RC di carica e scarica che evita che i rimbalzi meccanici del pulsante vengano interpretati come pressioni. I 2 LED sono invece collegati al pin RB1 del PIC e la loro accensione è mutuamente esclusiva. Il LED2 è verde e indica la modalità di funzionamento della carrozzina manuale, mentre LED3 è blu e indica la modalità automatica.

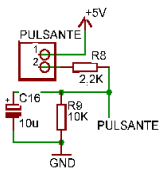
Altri componenti

Al PIN MCLR (Master Clear) del PIC è collegato un circuito che provvede a mantenere alto il valore di tale ingresso. I pin OSC1 e OSC2 sono collegati con un quarzo da 4 Mhz e due condensatori da 22pF, come specificato sul datasheet.

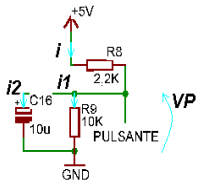
Si è inoltre previsto un connettore per la programmazione *in-circuit* del PIC, senza cioè la necessità di rimuovere il PIC dal suo zoccolino.

Dimensionamento del circuito di carica e scarica del pulsante

Al fine di evitare i rimbalzi meccanici del pulsante è stato realizzato un circuito di carica e scarica collegato al pulsante, utilizzando le resistenze R8, R9 ed il condensatore C16.



Il connettore denominato PULSANTE è collegato ad un interruttore normalmente aperto. Quando si chiude l'interruttore, la resistenza R8 risulterà dunque collegata alla tensione di alimentazione e il circuito, supponendo nulla la carica iniziale del condensatore, si caricherà nel seguente modo:



$$i = i_1 + i_2$$

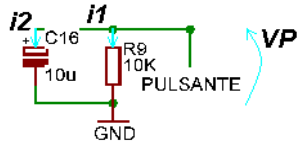
$$i = \frac{5\mathrm{V} - VP}{R8} \qquad i_1 = \frac{VP}{R9} \qquad i_2 = C\frac{dVP}{dt} = C\ \dot{VP}$$

$$\frac{5\mathrm{V} - VP}{R8} = \frac{VP}{R9} + C\ \dot{VP}$$

Risolvendo l'equazione differenziale e imponendo le condizioni iniziali si ottiene:

$VP(t) = \frac{5\mathrm{V} * R9}{R8 + R9}\left(1 - e^{-\frac{R8+R9}{R8\,R9\,C}}\right)$ e sostituendone i valori delle resistenze e dei condensatori si ottiene che la tensione raggiunta dopo un tempo pari a $5\tau = 5\frac{R8\,R9\,C}{R8+R9} \simeq 0.1\mathrm{sec}$ è di $4{,}1\,V$, sufficiente per essere considerata un "1 logico" dal piedino di ingresso del PIC.

La fase di scarica seguirà invece il seguente andamento:



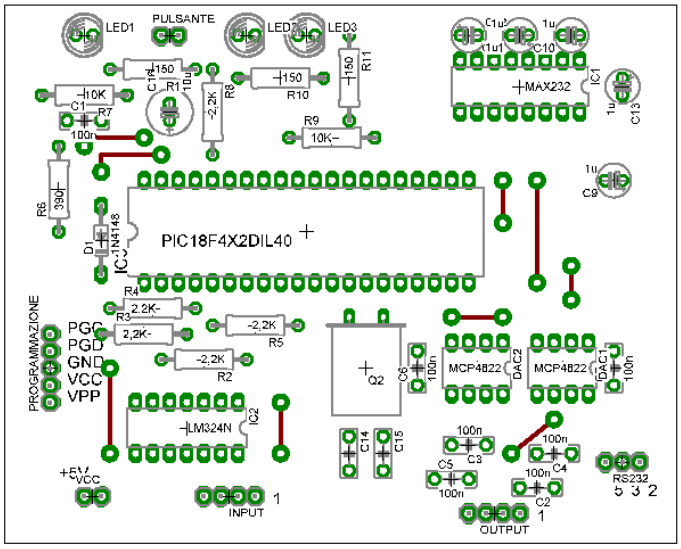
$i_1 = -i_2 \qquad \frac{VP}{R9} = -C\ \dot{VP}$ da cui $VP(t) = VP(0)e^{-\frac{1}{R9C}}$. Utilizzando come tensione di carica iniziale del condensatore $4{,}1\,V$ si può considerare che la scarica venga completata in $5\tau = 5\,R9\,C \simeq 0{,}5\,sec$.

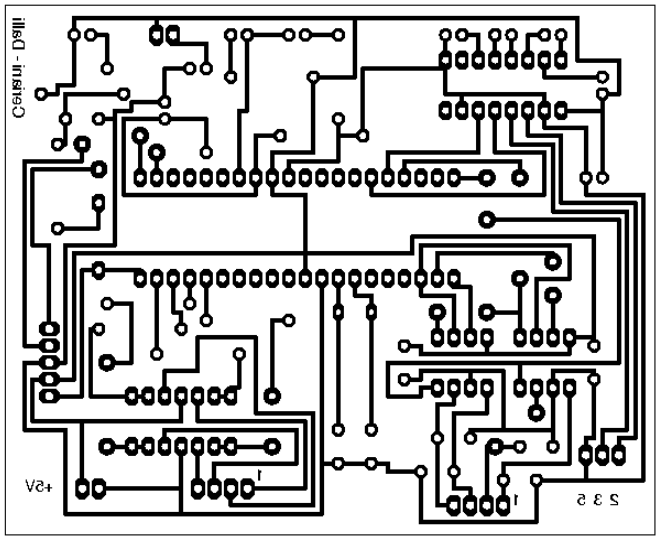
Si noti che per raggiungere il valore di "0 logico" sarà sufficiente un tempo inferiore a quello stimato.

La prova pratica del circuito ha permesso di verificare il corretto funzionamento e ha evidenziato la totale assenza dei problemi dei rimbalzi meccanici che si ottenevano collegando direttamente il pulsante al piedino di ingresso del PIC.

Realizzazione fisica del circuito di comando

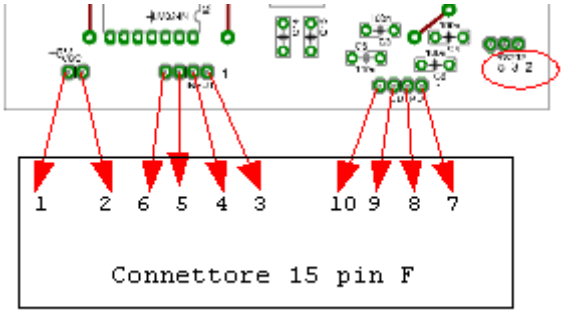
Dopo aver realizzato e testato lo schema precedentemente illustrato su breadboard, ne abbiamo realizzata una versione su basetta stampata tramite incisione fotochimica e successiva foratura e saldatura. In questa sezione riportiamo (non nelle dimensioni reali) il progetto del lato superiore (su cui vanno montati i componenti) e inferiore (su cui si trovano le piste). La basetta ha dimensioni sufficientemente contenute: 8 cm x 10 cm, anche se l'ingombro totale è maggiore a causa della necessità di installare alcuni connettori sulla scatola. In questa sezione commentiamo anche come sono collegati al circuito i vari connettori di interfaccia presenti sulla scatola del circuito.





I connettori chiamati *VCC, INPUT* e *OUTPUT* sono collegati ad un connettore da 15 poli femmina alloggiato sulla scatola che contiene il circuito. Con un cavo provvisto di connettore 15 poli maschio si effettua il collegamento di tale circuito alla scatola del joystic. In particolare, i due fili provenienti dal connettore *VCC* saranno collegati ai piedini 1 e 2 del connettore a 15 poli (il primo è quello a sinistra, indicato dalla scritta +5V). Il pin 1 di INPUT è collegato al pin 6 del connettore a 15 poli, il 2 (procedendo a sinistra dall'uno) al 5, il 3 al 4 e il 4 al 3. Il pin 1 dell'OUTPUT è collegato al pin 10, il 2 al 9, il 3 a 8 e il 4 a 7.

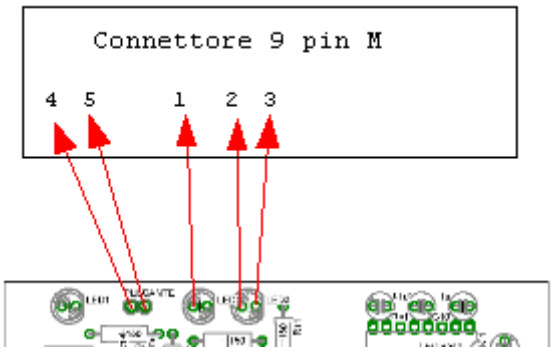
Le connessioni sono riportate nel seguente schema:



Il connettore RS232 è collegato al connettore a 9 pin femmina fissato alla scatola del circuito e utilizza solo i piedini 2, 3 e 5 di tale connettore. I pin da collegare al connettore sono indicati direttamente sul progetto, il 2 a destra, il 3 in mezzo, il 5 a sinistra.

Non è prevista nessuna interfaccia esterna per il connettore della programmazione, in quanto l'uso di questo connettore è piuttosto raro e la programmazione è un'operazione che va effettuata esclusivamente da personale qualificato.

Sulla scatola del circuito è presente un terzo connettore a 9 pin maschio che non è però esplicitamente presente sul circuito. A causa delle difficoltà incontrate nel reperire una scatola di dimensioni tali da poter essere alloggiata vicino all'utente della carrozzina e risultare comoda all'uso, si è deciso di spostare fisicamente i componenti identificati come LED2 e LED3 e collegare il pulsante (che già rimaneva esterno al circuito) in una seconda scatola alloggiata vicino al joystic della carrozzina e collegata tramite un cavo al circuito principale. Tale cavo è provvisto di un connettore 9 pin femmina da collegare al 9pin maschio presente sulla scatola del circuito. Il connettore da 9 poli sul circuito principale è quindi collegato all'anodo del led verde (pin 1 connettore a 9 pin), al catodo del led verde, comune anche all'anodo del led blu (pin 2), al catodo del led blu (pin 3) e ai due pin del connettore *PULSANTE* (pin 4 e 5 del connettore a 9 pin), come schematizzato:



Software realizzato per il microprocessore PIC e strumenti utilizzati

I compilatori presenti sul mercato per la programmazione dei microprocessori *PIC* permettono l'uso di diversi linguaggi di programmazione, diversi per l'astrazione che permettono di raggiungere. Alcuni linguaggi sono infatti simili al Basic, altri al C. Abbiamo scelto di utilizzare però l'assembler nativo del *PIC*, in quanto il software da realizzare non era particolarmente complesso. L'uso dell'assembler ha permesso inoltre di sviluppare un software “essenziale”, quindi molto rapido nei tempi di risposta. L'ambiente di sviluppo è *MPLAB IDE,* disponibile gratuitamente sul sito del costruttore.

Scelta del microprocessore PIC

La famiglia dei microprocessori *PIC* è molto vasta. Nello scegliere quale modello utilizzare abbiamo valutato i seguenti criteri:

1. Facilità di programmazione, dovendo utilizzare il linguaggio assembler
2. Moduli di interfaccia necessari (*UART*, *SPI*, convertitori *D-A*,numero *PIN I/O*)
3. Moduli interni implementati in hardware (Timer e gestione interrupt)

I *PIC* della famiglia *16x* presentano alcune difficoltà nella programmazione in assembler: suddividono infatti la memoria di lavoro in pagine ed è necessario selezionare sempre la pagina corretta prima di usare una variabile. Questo rappresenta una grossa fonte di errori e abbiamo optato quindi per i *PIC* della famiglia 18x, la cui memoria è interamente accessibile su un unica pagina.

La scelta è poi ricaduta sul *PIC18F452* con package *DIP* a 40 pin. Questo microprocessore presenta al suo interno i moduli prima citati necessari e un numero più che sufficiente di PIN di ingresso-uscita. In realtà le sue funzionalità sono anche sovrabbondanti rispetto al compito che deve svolgere, ma vista anche la nostra iniziale inesperienza si è preferito mantenere un cospicuo margine di sicurezza.

Strumenti utilizzati

L'ambiente di sviluppo utilizzato è *MPLAB IDE*, che permette di scrivere codice assembler per tutti i modelli di *PIC*, di simularne l'esecuzione, eseguirne il debug e, dopo aver effettuato le connessioni necessarie, programmare il microprocessore fisico con il software realizzato.

Un altro strumento molto utile, anch'esso fornito gratuitamente insieme all'ambiente di sviluppo *MPLAB IDE*, è *Application Maestro*. Questo applicativo permette di aggiungere all'applicazione che si sta sviluppando dei moduli assembler per l'uso delle componenti hardware del microprocessore. Nell'applicativo sarà possibile accedere alle funzionalità offerte utilizzando delle semplici chiamate a procedura. Ad esempio vengono rese disponibili le funzioni *UARTIntGetCh* e *UARTIntPutCh* per l'uso del modulo UART.

Nel nostro applicativo sono stati utilizzati i seguenti moduli, di cui riportiamo anche i parametri di configurazione impostati:

- *USART interrupt driven*, a 9600 baud e buffer di ricezione e trasmissione entrambi di 10 byte
- *10 bit ADC polled*, con tempo di acquisizione di 25ms, clock pari a metà di quello del processore e allineamento a sinistra dei 10 bit letti sui due byte del risultato (si rimanda al datasheet del PIC18F452 e al codice per ulteriori informazioni)
- *SPI Master (Polled)* con funzione bloccante (non restituisce il controllo all'applicazione finché non ha terminato di inviare i dati) e baud rate pari ad un quarto della frequenza del processore.

Panoramica sul software realizzato

Il software realizzato deve gestire il loop di controllo descritto precedentemente. In particolare, dopo aver svolto alcune operazioni di inizializzazione necessarie, come la corretta impostazione dell'uso dei piedini (input o output) o l'inizializzazione delle periferiche, si dovrà eseguire ciclicamente il loop di controllo. Abbiamo deciso di svolgere il ciclo di controllo ad una frequenza di 50Hz, sufficiente per un controllo preciso del moto della carrozzina e di molto superiore alle possibilità di comando manuale effettuate sulla leva del joystick. Per permettere che ogni ciclo duri il tempo desiderato, abbiamo utilizzato un timer (*Timer1*) opportunamente programmato per generare un interrupt ogni 20ms. Il *Timer1* viene dunque attivato ogni volta che il ciclo di controllo ricomincia da capo e ne viene attesa la scadenza prima di iniziare un nuovo ciclo. In ogni ciclo sarà necessario quindi leggere i valori presentati sui quattro convertitori analogici-digitali e sottrarli a due a due. I due valori così ottenuti rappresentano dunque le posizioni attuali della leva. A questo punto il ciclo di controllo si comporta diversamente se la modalità di funzionamento è manuale o automatica. Se la modalità è manuale, i valori appena letti dagli ADC andranno posti in uscita sui DAC, mentre, se la modalità è automatica, si dovranno impostare in uscita i valori letti dalla seriale (quindi comunicati dal PC). A questo punto il ciclo si conclude attendendo che scada il *Timer1* e in questo tempo di attesa si occupa di processare i dati provenienti dalla seriale.

La pressione del pulsante che permette di cambiare la modalità di funzionamento da manuale a automatica e viceversa è controllata da una routine di risposta ad interrupt. Quando la modalità passa ad automatica si attiva *Timer0* (impostato a 0,5 s). *Timer0* svolge una funzione di *watch-dog,* infatti ne viene resettato il conteggio quando un comando viene ricevuto correttamente dalla seriale. Se *Timer0* genera l'interrupt di fine conteggio significa che per 0,5 secondi non si sono ricevuti comandi corretti (ad esempio per checksum errato o proprio per mancanza del collegamento) e la modalità di funzionamento viene automaticamente riportata a manuale. Questo permette di garantire un tempo di validità di un comando, garantendo anche un minimo di sicurezza per l'utente in caso di malfunzionamento del software sul PC.

Un problema all'apparenza semplice, ma la cui soluzione a causa dell'uso dell'assembler è risultata piuttosto macchinosa, è la corretta presentazione dei valori da comunicare ai DAC (mcp4822) per produrre le tensioni di uscita. I valori letti dagli ADC e opportunamente sottratti tra loro e i valori comunicati dal PC sono infatti dei numeri compresi tra 0 e 255. I DAC utilizzati usano comandi a 12bit e fanno corrispondere ad ogni digit 1 millivolt. Infatti il valore di fondoscala è 4,095 volt, rappresentato dal numero 4095. I valori da 0 a 255 devono essere convertiti in valori corrispondenti compresi tra 0 e 2500. Per fare ciò è sufficiente moltiplicare il valore considerato per 2500/255. Dato che il microprocessore non gestisce in modo nativo né i numeri float né le moltiplicazioni, abbiamo optato per effettuare tale operazione di scala utilizzando solo somme e shift di bit. Infatti, considerando che 2500/255 è un valore pari a 9,80 (arrotondato a due cifre decimali), si può seguire il seguente procedimento:

- Si moltiplica il valore in input per 10, utilizzando il valore stesso shiftato a sinistra di una posizione sommato a quello shiftato a sinistra di tre posizioni.

- Si moltiplica il valore di input per 0,2, utilizzando la somma degli shift a destra di 3,4,6,7, corrispondenti a $\frac{1}{8}+\frac{1}{16}+\frac{1}{64}+\frac{1}{128}=\frac{27}{128}\simeq 0{,}21$.
- Si sottrae dal valore trovato al primo punto quello trovato al secondo, ottenendo così il valore iniziale moltiplicato per 9,8.

A questo punto, dovendo produrre l'uscita differenziale con valori compresi tra 1,25 e 3,75 Volt, sarà sufficiente sommare e sottrarre il valore numerico appena calcolato a 1250 e 3750. I valori così ottenuto potranno essere presentati ai DAC.

Software realizzato su PC

Sul PC abbiamo realizzato dei semplici applicativi dimostrativi che permettono di verificare il corretto funzionamento del circuito di comando è l'interfaccia con il computer. Alcuni applicativi sono stati realizzati in matlab e successivamente abbiamo sviluppato una classe C++ e un'applicazione dimostrativa.

Applicativi Matlab

- *provaVel.m* è una funzione che riceve come parametro di ingresso il nome della porta su cui si è collegato il circuito della carrozzina (es. 'COM1') e provvede a inviare i comandi alla carrozzina in modo da far partire la marcia indietro alla massima velocità, rallentare progressivamente fino a fermarsi e accelerare in marcia avanti fino alla massima velocità. Alla fine della sequenza viene impostata la modalità di funzionamento manuale.
- *serialeInv.m* "inverte" i comandi impostati sulla leva. Il programma infatti legge i movimenti imposti dall'utente sulla leva del joystick e comunica alla carrozzina il comando opposto: ad esempio, quando la leva è avanti la carrozzina procederà all'indietro.
- *drive.m* permette di guidare la carrozzina con i tasti "*wasd*".

Applicativi C++

L'applicativo è stato sviluppato in ambiente linux (Ubuntu 7.04) e permette, a livello dimostrativo, le stesse funzionalità di *serialeInv.m* e d*rive.m.* I due comportamenti dimostrativi sono stati realizzati semplicemente dopo aver implementato una classe che permette di lanciare un thread che "ascolta" la seriale in attesa dei valori comunicati dalla carrozzina al PC ed espone all'utente un metodo per mandare comandi alla carrozzina. La classe realizzata è contenuta nel file *"GestioneComandi.cpp"* (e nel relativo header *"GestioneComandi.h"*), di cui ne riportiamo e commentiamo i metodi pubblici.

```
class GestioneComandi{
[...]
public:
          GestioneComandi(char *dev);
          ~GestioneComandi();
          int isReady();

          void startReader();
          void updateStatus();
          void endWork();
          StateReader * accessLocalCopy();
          int sendCommand(unsigned char automatic,unsigned char ch1,unsigned char ch1);
};
```

Il costruttore accetta come parametro una stringa del tipo *"/dev/ttySx"*, dove *x* è un numero, e apre il file corrispondente al terminale della porta seriale scelta a 9600 baud (come impostato sul PIC). Per controllare che il costruttore abbia eseguito correttamente il proprio compito si può utilizzare il metodo *isReady(),* che restituisce un valore diverso da 0 se l'oggetto è pronto per essere usato. Per avviare il thread che legge in modo continuo i dati comunicati dalla carrozzina al PC si usa il metodo *startReader*. Per inviare un nuovo comando alla carrozzina si usa invece il metodo *sendCommand*, i cui parametri specificano la modalità di funzionamento (1 automatica, 0 manuale) e i due valori (tra 0 e 255) da imporre sui canali di comando. Quando si desidera accedere alle informazioni "collezionate" dal thread è necessario prima utilizzare il metodo *updateStatus,* che accede in modo mutuamente esclusivo alle variabili condivise con il thread e ne crea una copia locale, i cui campi potranno essere letti ottenendo il puntatore a questa risorsa con il metodo *accessLocalCopy*. Quando si desidera fermare il thread di lettura è necessario invocare il metodo *endWork*, che viene anche automaticamente invocato dal distruttore della classe.

Presentiamo ora la classe *StateReader*, che è utilizzata per memorizzare le informazioni raccolte dal thread durante la sua esecuzione.

```
class StateReader
{
[...]
public:
           [...]
           int isAutomatic();
           int getCh1();
           int getCh2();
           unsigned long  int getCommandRead();
           unsigned long  int getCommandErr();
           char *toString();
};
```

Commentiamo solo i metodi a cui si accede per reperire le informazioni raccolte dal thread; altri metodi sono utilizzati invece dal thread stesso, ma l'accesso dall'esterno è sconsigliato e inutile ai fini delle applicazioni da realizzare. Il metodo *isAutomatic* ci comunica se nell'ultima trama correttamente ricevuta la modalità di funzionamento della carrozzina era automatica (1) o manuale (0). I metodi *getCh1* e *getCh2* permettono di conoscere la posizione della leva nell'ultima trasmissione corretta. I metodi *getCommandRead* e *getCommandErr* permettono di conoscere il numero totale di trame ricevute e il numero di trame errate. Il metodo *toString* restituisce un puntatore ad una stringa allocata nello heap che contiene le informazioni in forma sintetica. Lo spazio di memoria utilizzato da questa stringa deve essere liberato dall'applicazione che ne fa uso.

Bibliografia e Link Utili

[1] http://www.dynamiccontrols.com
[2] C. Amaya, F. Díaz del Río, J. L. Sevillano, G. Jiménez, S. Vicente, A. Civit Balcells *Schedulability Analysis of CAN based Systems withPrecedence Constraints.* http://www.can-cia.org/icc/7/amaya.pdf
[3] Mallet *Wad project.* http://www.laps.univ-mrs.fr/~mallet/WADProjectIROS021.pdf
[4] http://www.microchip.com
[5] http://www.national.com/ds/LM/LM741.pdf
[6] http://www.national.com/mpf/LM/LM324.html
[7] http://www.maxim-ic.com/
[8] http://www.maxim-ic.com/quick_view2.cfm/qv_pk/1798
[9] http://www.microchip.com/stellent/idcplgidcplg?IdcService=SS_GET_PAGE&nodeId=1335&dDocName=en010296
[10] http://www.microchip.com/stellent/idcplgidcplg?IdcService=SS_GET_PAGE&nodeId=1335&dDocName=en024016