
**UNIVERSITÀ DI BRESCIA**
FACOLTÀ DI INGEGNERIA
Dipartimento di Elettronica per l'Automazione

**Laboratorio di Robotica Avanzata**
**Advanced Robotics Laboratory**

Corso di Robotica
(Prof. Riccardo Cassinis)

# Survivor 2

Elaborato di esame di: **Marco Negrini, Alessandro Zanacchi**

Consegnato il: **12 febbraio 2003**

# Sommario

*Lo scopo del nostro elaborato è quello di realizzare una nuova versione di un prototipo di sedia robot. Abbiamo sviluppato una nuova scheda basata sul microcontrollore PIC16F876A in grado di gestire i segnali provenienti da otto sensori all'infrarosso e dalla batteria,comunica con un pc tramite la seriale; inoltre fornisce i comandi ai motori collegati alle zampe della sedia. Abbiamo scritto il programma tramite un linguaggio assembly dedicato che realizza i comportamenti desiderati.*

# 1. Introduzione

Survivor 2 è la seconda versione di una creazione artistica che vuole esprimere l'orrore inflitto dalle mine antiuomo su persone innocenti. La sedia robot è una normale sedia a cui sono state tagliate le "gambe" anteriori per permettere l'aggiunta di due articolazioni, collegate a due motori, che consentono il movimento. Questa in pratica si trascina ed il movimento è reso più scorrevole con l'aggiunta di rotelle all'estremità delle gambe posteriori. Gestendo la sequenza dei segnali di comando è possibile far camminare la sedia avanti (una successione di passi alternati), a destra (una successione di passi con la zampa sinistra) e a sinistra (passi con la zampa destra) al fine di evitare ostacoli; può inoltre simulare un pianto tramite un cicalino piezoelettrico. Tutto questo per emulare sentimenti di: paura,vergogna, diffidenza ecc…

# 2. Problemi affrontati

Come prima modifica dell'hardware della sedia ci siamo occupati del comando dei motori. Dovevamo pilotare due motori in corrente continua, i motori dei tergicristalli delle automobili, alimentati con una tensione di 12 Volt con un assorbimento di circa 2-3 Ampere a vuoto, quindi abbiamo deciso di utilizzare due mosfet di potenza pilotati direttamente dal pic.

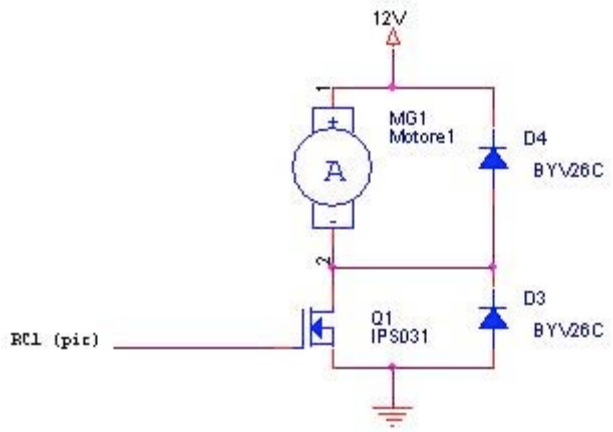



**Fig. 1: Comando motore**

In questo modo possiamo controllare la velocità dei motori con un pwm generato dal pic, cambiando semplicemente il suo duty-cycle, avendo così la possibilità di modificare la velocità di esecuzione dei passi della sedia robot.

I diodi in parallelo al motore e al mosfet servono per scaricare le extracorrenti generate all'accensione e allo spegnimento del motore, dovute all'alta induttanza; servono due dei diodi molto veloci.

Utilizzando il pwm è sorto però un problema, perché sollecitando il motore con un'onda quadra con una frequenza di circa 4KHz si generano extra correnti di notevole intensità, e quindi i diodi veloci che abbiamo messo per protezione in parallelo al motore si scaldavano parecchio, quindi siamo stati costretti a sostituirli con diodi aventi un case più robusto e grande per dissipare con più facilità il calore, con la possibilità di montarli su una aletta di raffreddamento.

I motori dei tergicristalli sono provvisti di un interruttore normalmente chiuso che si apre in un solo punto lungo il giro, per poter conoscere un giro esatto del motore; questo per noi è importante poiché dobbiamo fare eseguire alla sedia robot dei passi, ma un passo corrisponde ad un giro di un motore.

Abbiamo deciso quindi di collegare i due interruttori, rispettivamente corrispondenti ai due motori, con due ingressi "interrupt on change" del pic, secondo questo schema:

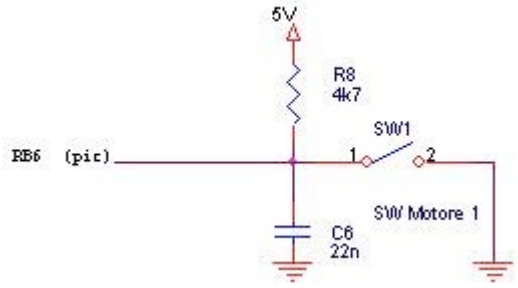



**Fig. 2: Interruttore motore**

in questo modo quando il motore è in moto l'interruttore è aperto e l'ingresso del pic è alto, appena il motore ha terminato un giro l'interruttore si chiude e l'ingresso del pic diviene basso, generando un interrupt che causerà l'arresto del motore.

Il circuito R-C serve per rallentare la dinamica dell'interruttore ed evitare quindi che il pic catturi anche i suoi rimbalzi.

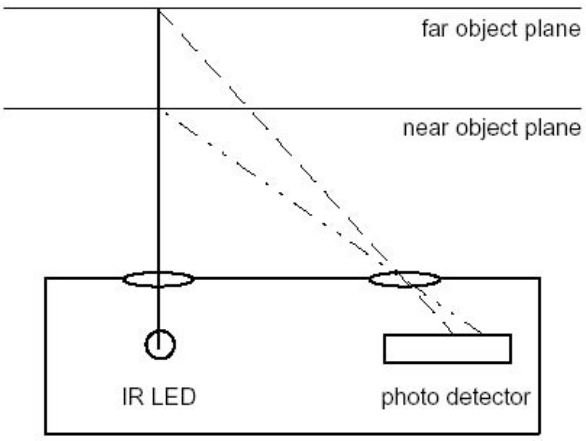
Avendo utilizzato due ingressi "interrupt on change" il motore genera un interrupt sia quando si ferma, l'interruttore si chiude, sia quando si accende, l'interruttore si apre, ma l'interruzione effettivamente utile è solo quella del motore che si ferma, quindi usiamo uno stratagemma software per non considerare l'altro interrupt.

Abbiamo poi affrontato il problema di far comunicare il microcontrollore con un pc, utilizzando la seriale presente nel microcontrollore stesso, in modo da poter comunicare con la sedia per operare azioni di settaggio e di controllo.

Per quanto riguarda il software abbiamo dovuto costruire tutte le procedure per il controllo base della sedia, come il controllo dei motori, l'acquisizione dei sensori esterni, la comunicazione seriale, ecc. in modo da poterle collegare fra loro in modo semplice per poter dare alla sedia vari comportamenti.

## 2.1. Prove sui sensori

La sedia robot ha come input dal mondo esterno delle misure di distanza dagli oggetti che le stanno intorno, queste misure sono effettuate da sensori di distanza ad infrarossi GP2D02, che si basano sulla triangolazione: un led a infrarossi proietta un raggio luminoso che incidendo contro un oggetto torna verso un fotoricevitore, e quindi la distanza dell'oggetto è proporzionale all'inclinazione del raggio luminoso.

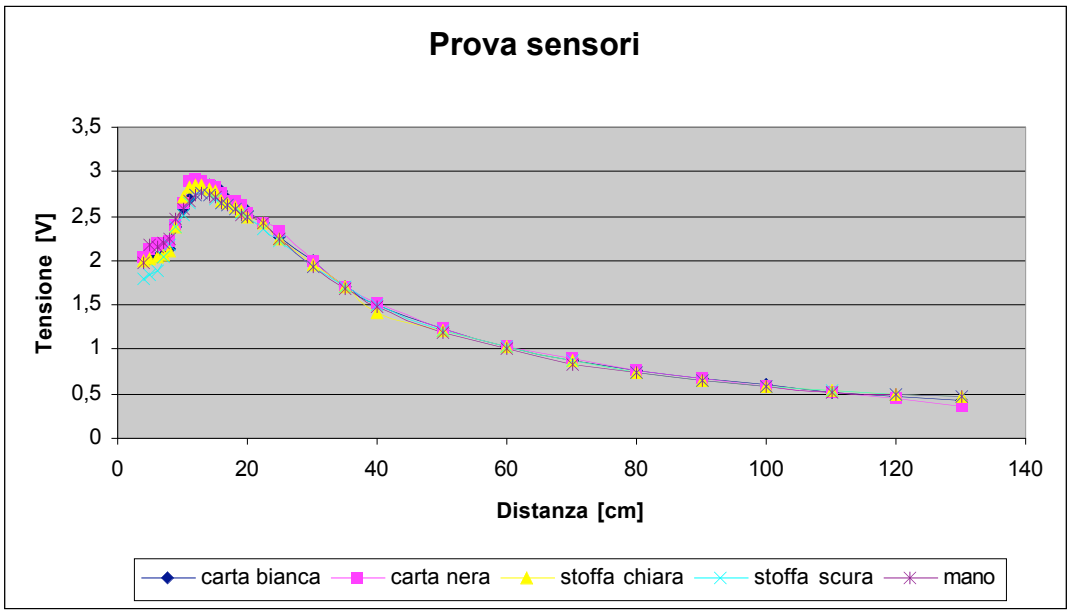



Il sensore è alimentato con una tensione di 5 Volt, e restituisce una tensione inversamente proporzionale alla distanza dell'ostacolo che ha di fronte, più l'ostacolo è distante più piccola è la tensione in uscita, con un andamento non lineare.

Per questo motivo abbiamo eseguito delle prove con cinque tipi di ostacoli diversi, misurando l'uscita del sensore, per ottenere la curva dell'andamento della tensione in relazione alla distanza per renderci conto della precisione sui diversi materiali.

Ecco le prove effettuate con il relativo grafico:

| Distanza [cm] | carta bianca [V] | carta nera [V] | stoffa chiara [V] | stoffa scura [V] | Mano [V] |
|---|---|---|---|---|---|
| 130 | 0,426 | 0,35 | 0,465 | 0,465 | 0,464 |
| 120 | 0,468 | 0,44 | 0,5 | 0,504 | 0,484 |
| 110 | 0,523 | 0,51 | 0,54 | 0,542 | 0,523 |
| 100 | 0,6 | 0,58 | 0,581 | 0,581 | 0,581 |
| 90 | 0,67 | 0,67 | 0,65 | 0,659 | 0,64 |
| 80 | 0,763 | 0,77 | 0,74 | 0,737 | 0,73 |
| 70 | 0,881 | 0,9 | 0,874 | 0,875 | 0,84 |
| 60 | 1,037 | 1,037 | 1,025 | 1,029 | 1,01 |
| 50 | 1,23 | 1,23 | 1,208 | 1,209 | 1,19 |
| 40 | 1,51 | 1,529 | 1,419 | 1,51 | 1,473 |
| 35 | 1,713 | 1,713 | 1,713 | 1,714 | 1,688 |
| 30 | 2,004 | 2 | 1,954 | 1,933 | 1,933 |
| 25 | 2,265 | 2,34 | 2,235 | 2,224 | 2,235 |
| 22,5 | 2,429 | 2,429 | 2,412 | 2,36 | 2,412 |
| 20 | 2,565 | 2,534 | 2,48 | 2,48 | 2,48 |
| 19 | 2,603 | 2,62 | 2,552 | 2,535 | 2,518 |
| 18 | 2,655 | 2,66 | 2,603 | 2,587 | 2,569 |
| 17 | 2,703 | 2,672 | 2,651 | 2,621 | 2,622 |
| 16 | 2,79 | 2,757 | 2,672 | 2,656 | 2,638 |
| 15 | 2,823 | 2,824 | 2,775 | 2,69 | 2,706 |
| 14 | 2,857 | 2,84 | 2,807 | 2,741 | 2,741 |
| 13 | 2,87 | 2,89 | 2,84 | 2,759 | 2,758 |
| 12 | 2,84 | 2,907 | 2,857 | 2,742 | 2,741 |
| 11 | 2,7 | 2,89 | 2,824 | 2,656 | 2,673 |
| 10 | 2,58 | 2,64 | 2,706 | 2,519 | 2,587 |
| 9 | 2,374 | 2,4 | 2,38 | 2,377 | 2,463 |
| 8 | 2,123 | 2,23 | 2,1 | 2,171 | 2,236 |
| 7 | 2,06 | 2,18 | 2,06 | 2,049 | 2,206 |
| 6 | 2,04 | 2,2 | 2,044 | 1,879 | 2,153 |
| 5 | 2,023 | 2,13 | 2,011 | 1,842 | 2,171 |
| 4 | 1,99 | 2,04 | 1,99 | 1,787 | 1,973 |




Per prima cosa notiamo che il sensore ha un comportamento abbastanza indipendente dal tipo di materiale da cui è costituito l'ostacolo, perché le curve risultano molto sovrapposte, ma dopo questo fatto positivo osserviamo che la curva non è monotona, ma presenta un picco di tensione in corrispondenza di una distanza di circa 13-14 cm.

A causa di questa non monotonia della curva non possiamo utilizzare il sensore per misurare distanze inferiori ai 13-14 cm, poiché la tensione fornitaci non è più caratterizzante la distanza.

Per ovviare a questo inconveniente si potrebbero montare i sensori non sull'estremità della sedia ma internamente alla sua superficie in modo che la misura accettabile più piccola del sensore risulti la distanza più piccola a cui vogliamo poter rilevare gli ostacoli dalla sedia.

# 3. Soluzione adottata

Il circuito completo della scheda per il controllo della sedia robot è il seguente:

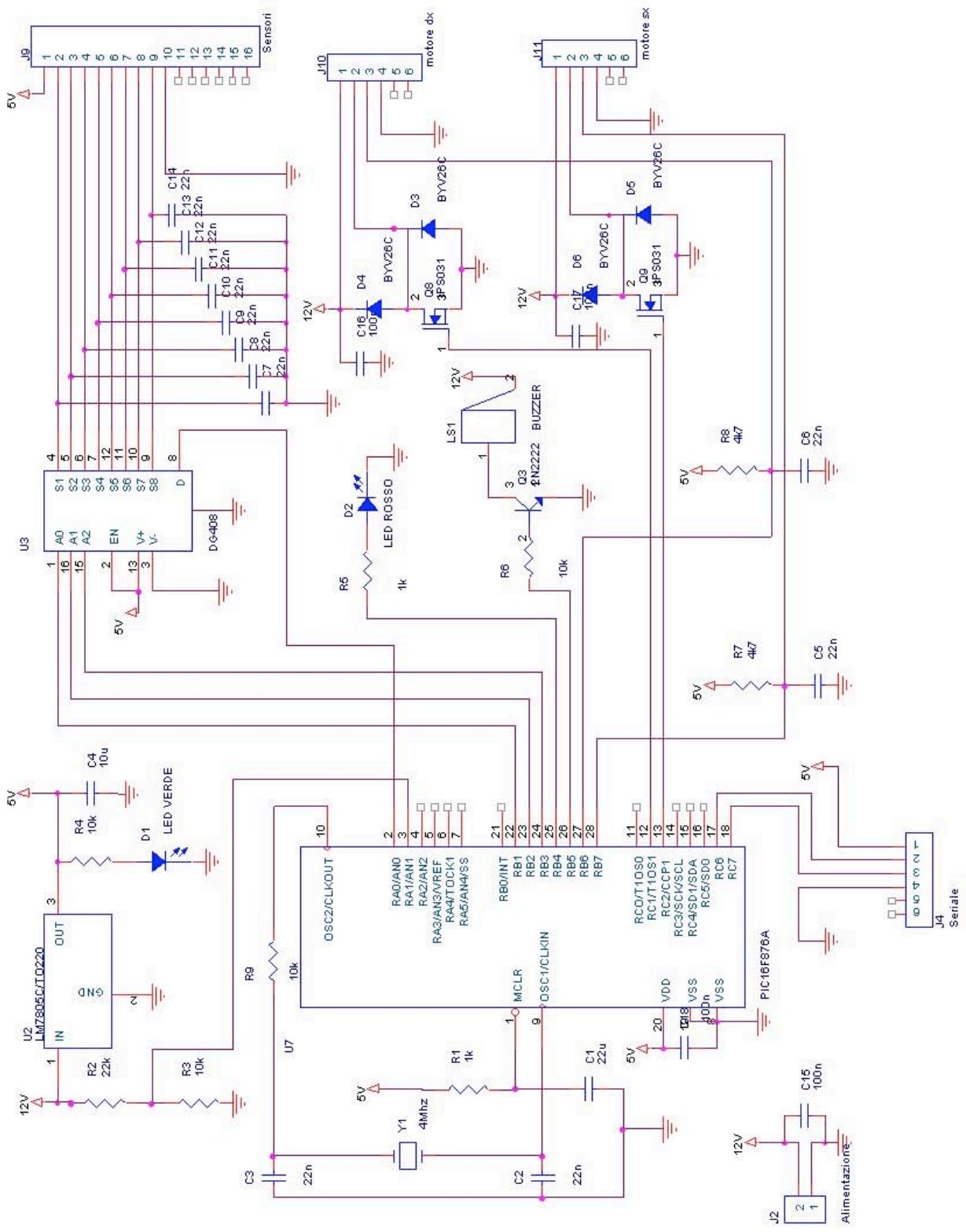



**Fig. 3: Scheda di controllo**

È basato sul microcontrollore PIC16F876A, scelto perché dotato di una memoria programma flash, e una memoria dati eeprom, 5 ingressi analogici con convertitore a/d a 10 bit, tre timer e due uscite pwm.

La scheda viene alimentata da una batteria Ni/Cd ricaricabile a 12V, e al suo interno vengono generati i 5V con l'integrato stabilizzatore LM7805, il clock del microcontrollore viene generato al suo interno con l'ausilio del quarzo a 4MHz.

Tra la batteria e la scheda abbiamo inserito un fusibile da 10Ampere di protezione ed un interruttore generale.

Avendo a disposizione solo 5 ingressi analogici, ma dovendo acquisire i segnali di 8 sensori di distanza, utilizziamo un multiplexer analogico MAXIM DG408DJ "8 a 1" ad alimentazione singola e con uscita compresa in un range tra 0 e 5 Volt.

Comandiamo il multiplexer con tre bit del portB pel pic, RB1, RB2 e RB3, e la sua uscita viene portata al pin RA0 del portA che è il primo ingresso analogico; abbiamo inoltre collegato un condensatore ad ogni sensore per eliminare i possibili disturbi.

Nel secondo ingresso analogico, RA1 portiamo la tensione della batteria, opportunamente ridotta ad un valore inferiore ai 5V tramite un partitore, per tenere sotto controllo il livello di carica della batteria.

Comandiamo un led rosso tramite il pic avendolo collegato tramite una resistenza al pin RB4 del portB, mentre comandiamo un cicalino piezoelettrico avendolo collegato al pin RB5 dello stesso port tramite un transistor 2N222, infatti il cicalino va alimentato con una tensione di 12V e assorbe una corrente nell'ordine delle centinaia di mA, e il microcontrollore non è in grado di fornire in uscita questa corrente.

Con i pin RC1 e RC2 del portC, che sono le uscite predisposte per il pwm, comandiamo i due motori tramite due mosfet di potenza, con l'alimentazione di 12V e due diodi veloci di protezione; questa parte di circuito è quella che assorbe più corrente.

Gli interruttori dei motori, che si aprono a fine giro, sono collegati ai piedini RB6 e RB7 del pic, che sono ingressi ad "interrupt on change", cioè generano un interrupt ogni volta che cambiano stato, in questo modo possiamo gestire la fine di un passo attraverso un interrupt invece di aspettare con un polling.

I pin RC6 e RC7 sono rispettivamente il piedino di trasmissione ed il piedino di ricezione della porta seriale asincrona contenuta nel microcontrollore, che utilizziamo per scambiare informazioni con un emulatore di terminale da pc.

La seriale del microcontrollore lavora però con la logica TTL, con i valori di tensione 5:0 Volt, mentre lo standard 232 del pc lavora con la tensione +12:-12 Volt, per questo motivo utilizziamo l'integrato MAX232 per interfacciare i due standard.

Costruiamo questa interfaccia in una scheda a parte, che andrà collegata alla scheda di controllo della sedia e alla porta seriale di un pc.

Lo schema è il seguente:

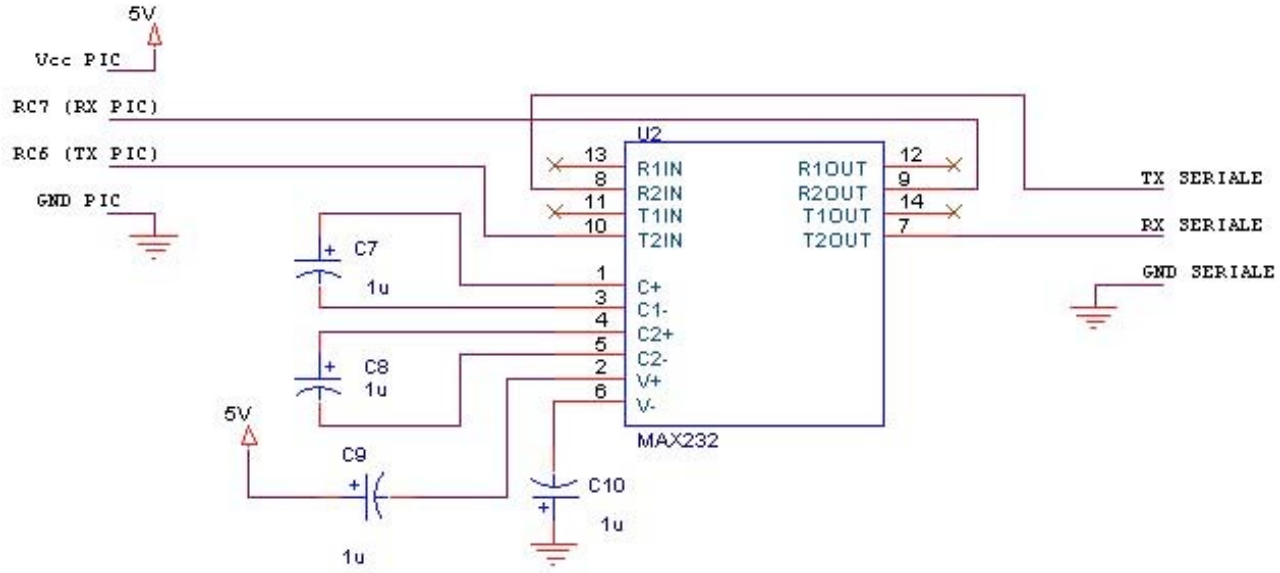



**Fig. 4: Interfaccia seriale**

## 3.1. Programma del pic

È stato scelto di costruire un programma basato completamente su interrupt, cioè ogni compito da svolgere è richiamato da un'interruzione, che può essere causata internamente al microcontrollore oppure esternamente dai motori e dalla seriale.

Il programma principale infatti oltre ad effettuare una serie di configurazioni, non è altro che un ciclo infinito in attesa di interrupt; l'azione che compie più frequentemente è quella di convertire il valore dei sensori, mentre l'azione principale è quella che viene richiamata ogni overflow del timer1 (circa ogni 63msec) che esegue un controllo sullo stato decidendo quali azioni effettuare.

Le variabili di controllo utilizzate sono:

- **STATO**:
  - bit 0. (stato trasmissione), se a 0 indica trasmissione seriale terminata, se a 1 indica trasmissione seriale in corso
  - bit 1. (stato motore destro), se a 0 indica motore dx spento, se a 1 indica motore dx acceso
  - bit 2. (stato motore sinistro), se a 0 indica motore sx spento, se a 1 indica motore sx acceso
  - bit 3. (ultimo passo), se a 0 indica che l'ultimo passo effettuato è quello destro, se a 1 indica ultimo passo quello sinistro
  - bit 4. (davanti), se a 0 indica nessun ostacolo davanti, se a 1 indica ostacolo davanti
  - bit 5. (destra), se a 0 indica nessun ostacolo a destra, se a 1 indica ostacolo a destra

bit 6. (sinistra), se a 0 indica nessun ostacolo sinistra, se a 1 indica ostacolo a sinistra

bit 7. (dietro), se a 0 indica nessun ostacolo dietro, se a 1 indica ostacolo dietro

- **ORDINE:**

bit 0. se a 1 ordina di fare un passo con motore destro

bit 1. se a 1 ordina di fare un passo con motore sinistro

bit 2. se a 1 ordina di accendere il cicalino

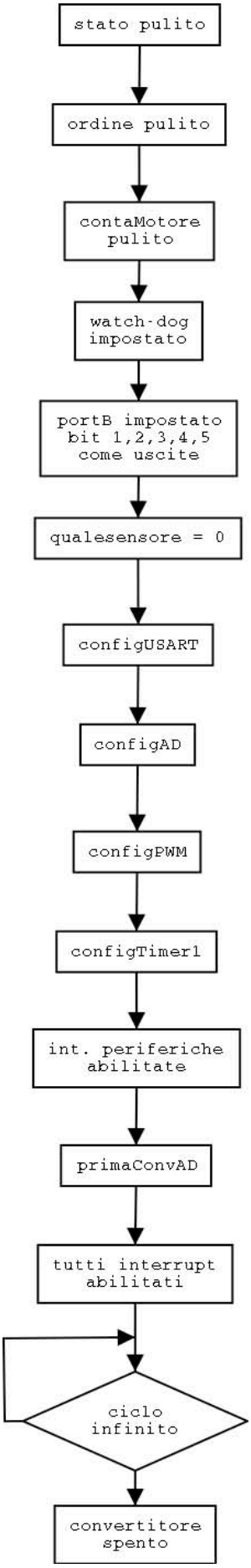
bit 3. se a 1 ordina di accendere il led

bit 4. se a 1 ordina di scrivere il valore dei sensori in eeprom

### 3.1.1. Programma principale

Il corpo principale del programma compie le seguenti azioni:

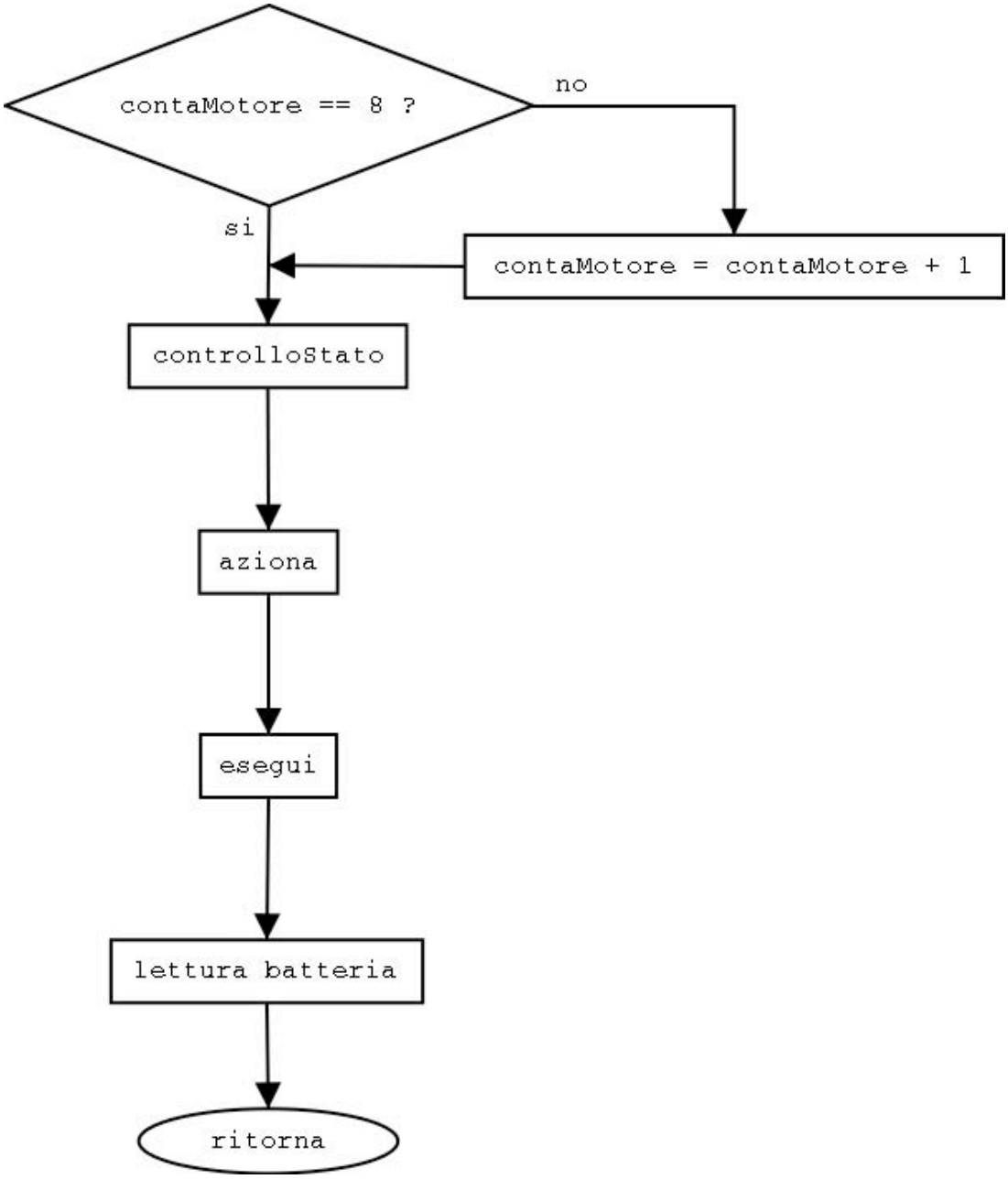
- Inizializzazione delle variabili di stato.
- Configurazione del watch-dog.
- Configurazione delle porte.
- Configurazione delle periferiche:
- Porta seriale: configurata asincrona a 9600 baud senza parità, impostata in modo tale che quando si è completata la ricezione di un byte viene generata un'interruzione.
- Convertitore analogico-digitale: vengono impostate le tensioni di riferimento 0-5V e si imposta come canale di ingresso RA0 che è quello proveniente dal multiplexer analogico. Anche in questo caso si imposta in modo che quando ha finito una conversione generi un'interruzione.
- Modulo PWM: Viene settato la frequenza a circa 4 KHz ed il duty cycle. Una volta che questo modulo è stato impostato, per accendere l'uscita basterà configurare un piedino del TRISC.
- Viene impostato il TIMER1, in modo che vada in overflow circa ogni 63ms. Quando va in overflow genera un'interruzione.
- Viene abilitata la maschera delle interruzioni delle periferiche
- Viene fatta partire la prima conversione di un sensore (richiamando la procedura **primaConvAD**), in questo modo quando questa avrà finito genererà un'interruzione, la quale farà partire la conversione di un altro sensore.
- Viene abilitata la maschera globale di tutte le interruzioni.
- Il programma rimane in attesa di interruzioni compiendo un ciclo infinito.


stato pulito
ordine pulito
contaMotore pulito
watch-dog impostato
portB impostato bit 1,2,3,4,5 come uscite
qualesensore = 0
configUSART
configAD
configPWM
configTimer1
int. periferiche abilitate
primaConvAD
tutti interrupt abilitati
ciclo infinito
convertitore spento

### 3.1.2. Procedura intTimer1

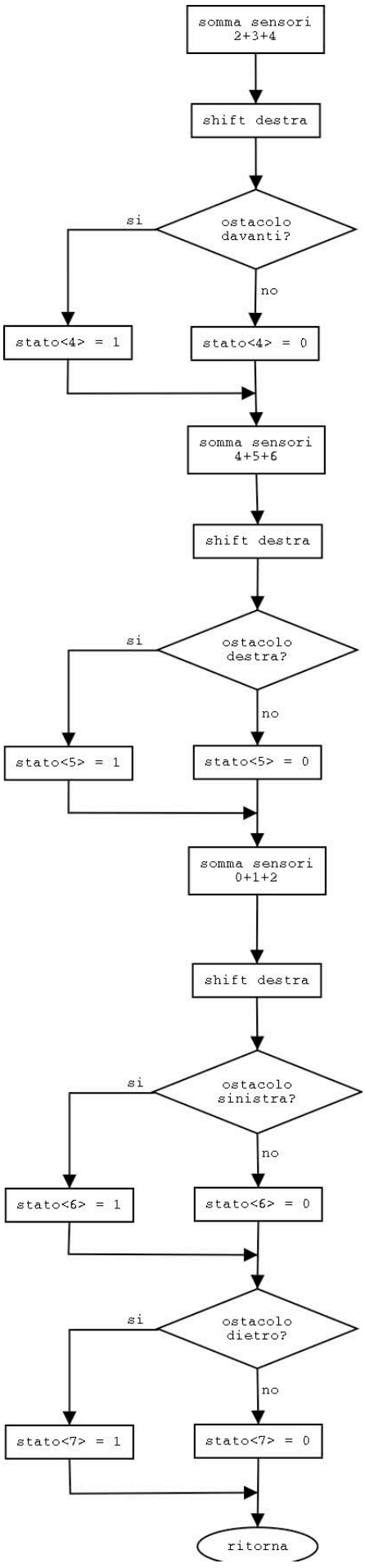
Questa procedura viene richiamata ogni volta che il TIMER1 va in overflow ossia circa ogni 63ms, ed ha il compito di effettuare dei controlli periodici sullo stato del sistema per decidere le azioni da eseguire.

- Incremento delle variabile *contaMotore*, fino a quando non ha raggiunto il valore 8, questa variabile viene annullata quando viene fatto partire e fermare un motore, in questo modo deve passare 8 volte il tempo di overflow del timer1 prima che il quarto bit di questa variabile torni ad essere ad uno. In questo lasso di tempo le interruzione dei motori non vengono prese in considerazione, in quanto l'interruzione utile è solo quella generata quando il motore si ferma.
- Viene richiamata la procedura **controlloStato**, questa ha in ingresso i valori dei sensore e deve verificare se c'è qualcosa attorno.
- Viene richiamata la procedura **aziona**, essa ha in ingresso la variabile *stato* e decide le azioni da compiersi andandole a specificare nella variabile *ordine*.
- A questo punto viene cambiato il canale del convertitore analogico-digitale in modo da poter acquisire il valore proveniente dalla batteria; poi si fa partire la conversione.
- Viene richiamata la procedura **esegui**, questa ha in ingresso la variabile *ordine*, deve semplicemente mettere in funzione gli attuatori.



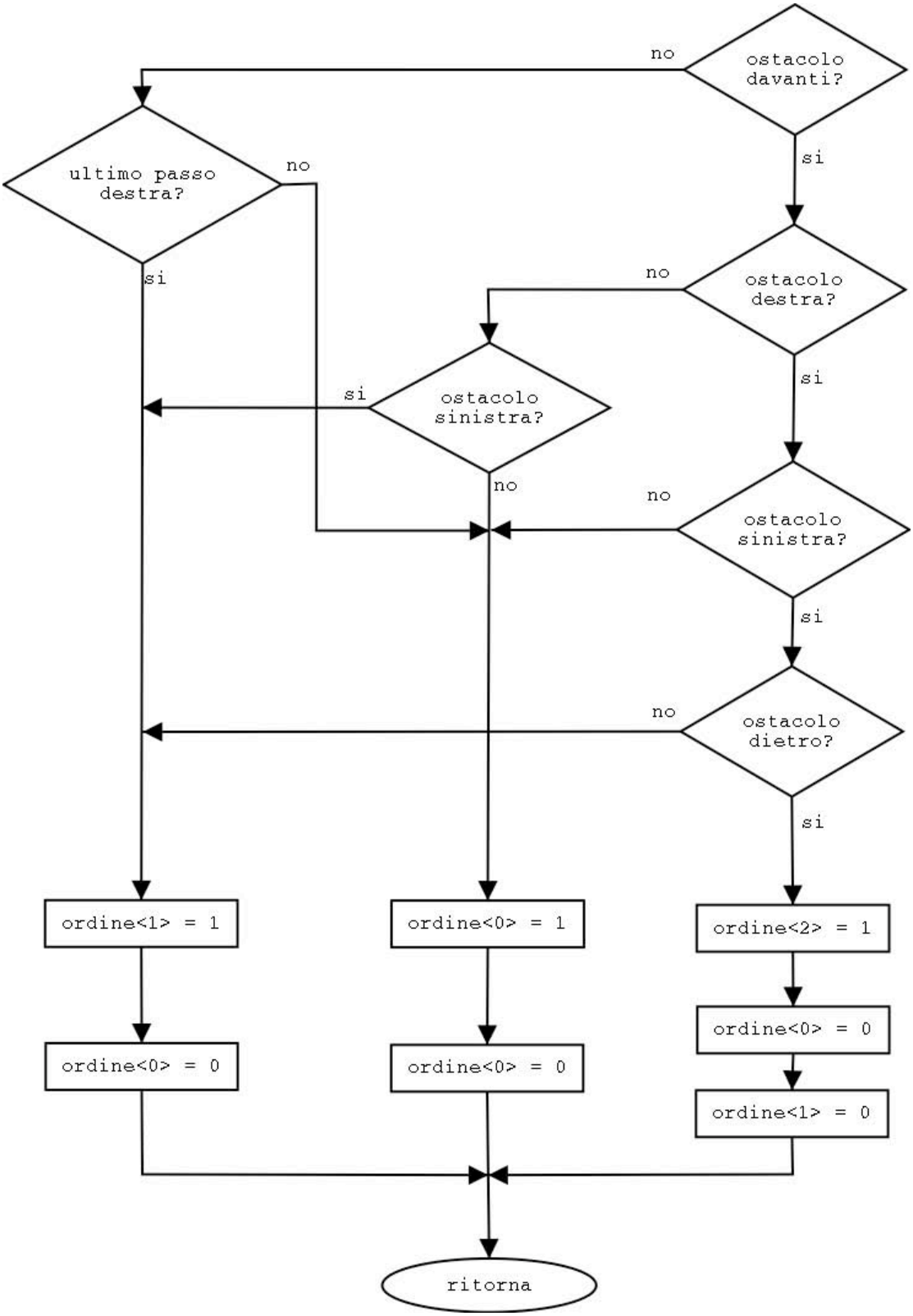

### 3.1.3. Procedura controlloStato

Controllando il valore restituito dai sensori verifica la presenza o meno di ostacoli nelle direzioni: davanti, destra, sinistra e dietro ed imposta la variabile stato di conseguenza.


somma sensori 2+3+4
shift destra
si
ostacolo davanti?
no
stato<4> = 1
stato<4> = 0
somma sensori 4+5+6
shift destra
si
ostacolo destra?
no
stato<5> = 1
stato<5> = 0
somma sensori 0+1+2
shift destra
si
ostacolo sinistra?
no
stato<6> = 1
stato<6> = 0
si
ostacolo dietro?
no
stato<7> = 1
stato<7> = 0
ritorna

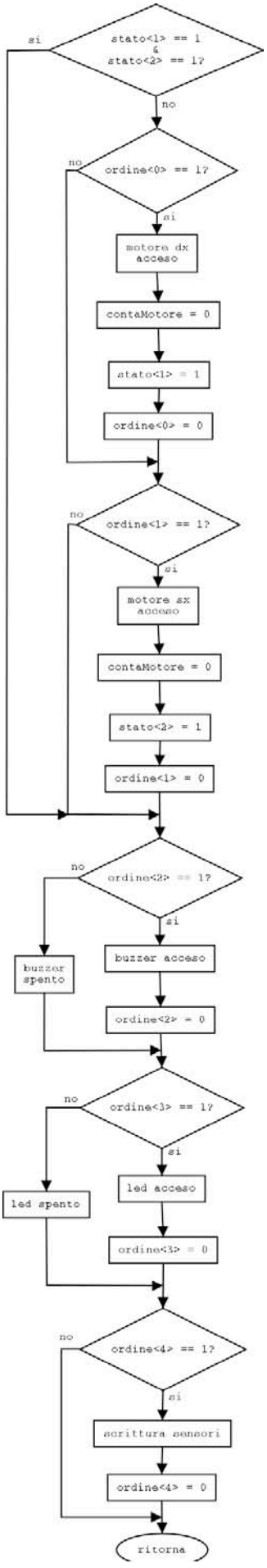
### 3.1.4. Procedura aziona

In base alla variabile stato decide quali azioni deve svolgere e imposta la variabile ordine di conseguenza.


ostacolo davanti?
no
si
ultimo passo destra?
no
si
ostacolo destra?
no
si
ostacolo sinistra?
si
no
ostacolo sinistra?
no
si
ostacolo dietro?
no
si
ordine<1> = 1
ordine<0> = 0
ordine<0> = 1
ordine<0> = 0
ordine<2> = 1
ordine<0> = 0
ordine<1> = 0
ritorna

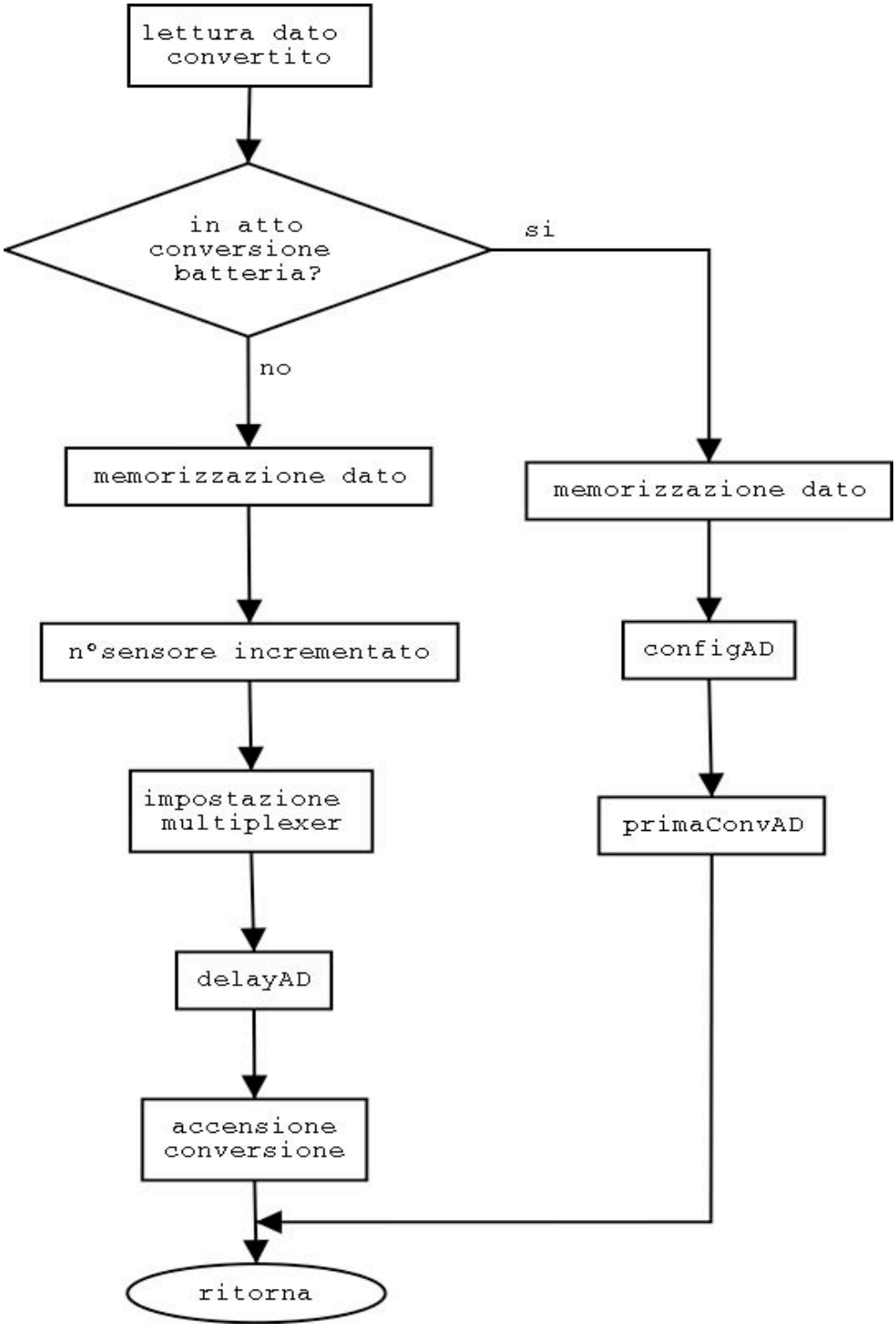
### 3.1.5. Procedura esegui

In base alla variabile ordine aziona le periferiche, cioè accende i motori, accende o spegne il cicalino ed il led e scrive in eeprom i valori letti dai sensori, in modo da poterli leggere da seriale.


stato<1> == 1
&
stato<2> == 1?
si
no
ordine<0> == 1?
no
si
motore dx acceso
contaMotore = 0
stato<1> = 1
ordine<0> = 0
ordine<1> == 1?
no
si
motore sx acceso
contaMotore = 0
stato<2> = 1
ordine<1> = 0
ordine<2> == 1?
no
si
buzzer spento
buzzer acceso
ordine<2> = 0
ordine<3> == 1?
no
si
led spento
led acceso
ordine<3> = 0
ordine<4> == 1?
no
si
scrittura sensori
ordine<4> = 0
ritorna

### 3.1.6. Procedura conversione

Per prima cosa viene controllato se questa procedura è stata richiamata per effettuare la lettura dei sensori oppure della batteria, per far questo basta controllare come è impostato il canale del convertitore. Se si tratta della batteria deve, oltre che a memorizzare il valore, richiamare la procedura di configurazione del convertitore analogico-digitale e far ripartire la conversione dei sensori. Se invece si tratta di una conversione proveniente dei sensori deve memorizzare il valore nel vettore *sensori*, deve incrementare la variabile *qualeSensore* e far partire in questo modo la conversione del prossimo sensore.




### 3.1.7. Procedura intPortB

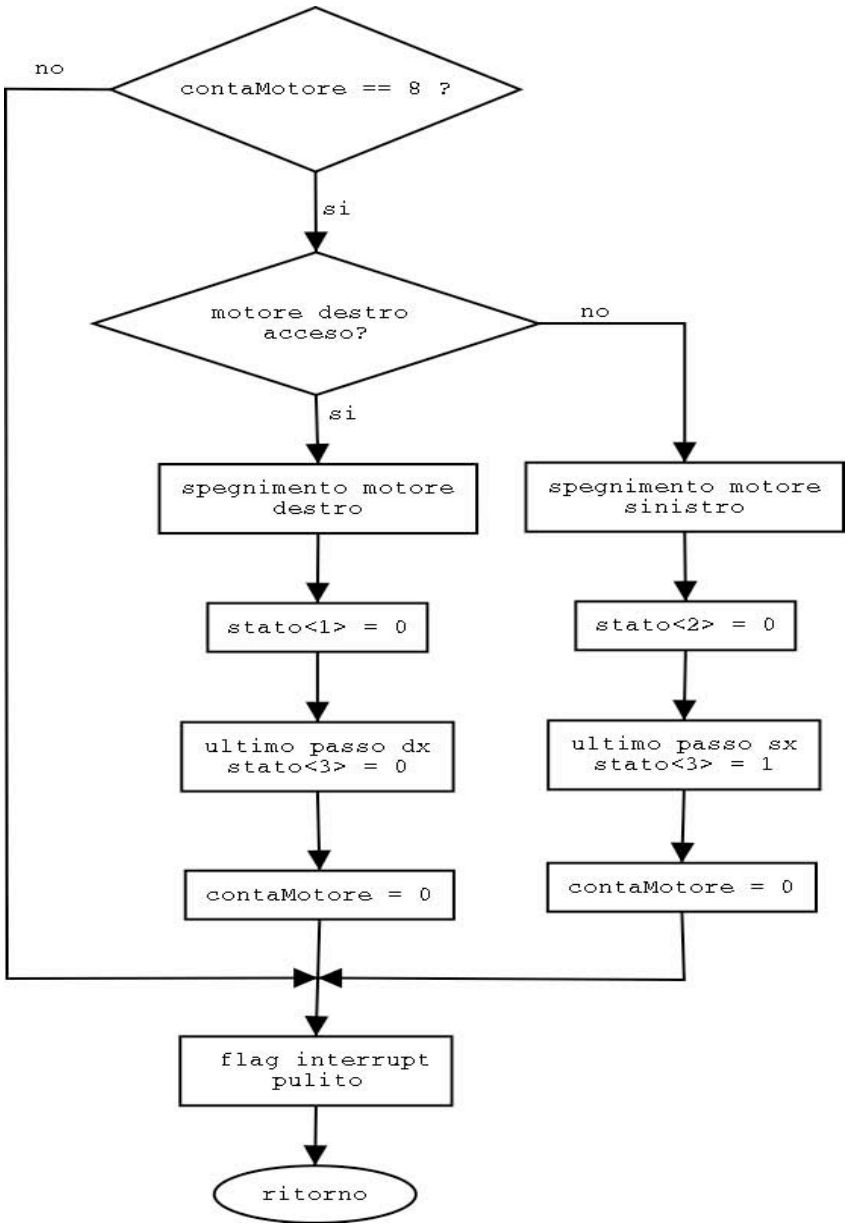
Questa procedura viene richiamata quando si verifica un'interruzione proveniente dai motori. Poiché tali interruzioni sono del tipo on-change, la prima interruzione che avviene, deve essere scartata perché avviene quando il motore è appena partito e

l'interruttore va ad aprirsi. Quindi l'interruzione da considerare è solo quella che avviene alla fine del giro del motore ossia quella che indica che l'interruttore si è chiuso.

Per accettare solo la seconda interruzione controlliamo se è ad 1 il quarto bit di *contaMotore*, in questo modo visto che la variabile viene pulita quando sono stati fatti partire i motori; ed è incrementata solo ad ogni overflow di timer1; siamo così sicuri che sono passati circa 400ms da quando il motore è partito.

A questo punto si può controllare quale dei due motori è acceso in questo modo sappiamo quale ha generato l'interruzione, così possiamo andare a spegnerlo.

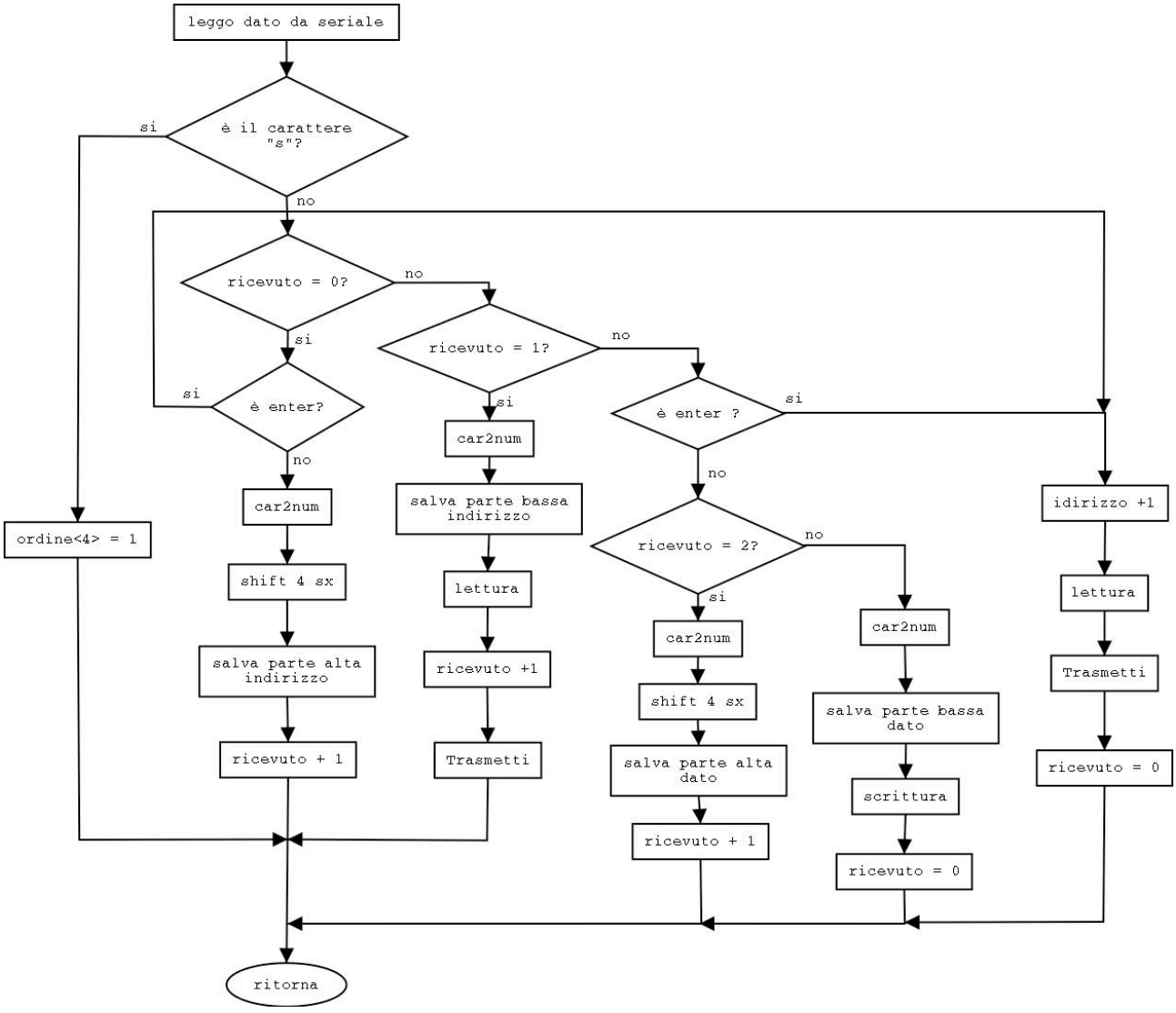
La variabile *contaMotore* viene azzerata anche quando vengono spenti i motori, perché anche in questo caso si verifica un interrupt, che potrebbe fare entrare nella procedura spegnendo un motore già spento ma cambiando la variabile stato modificando la sincronia dei passi.



### 3.1.8. Procedura ricezione

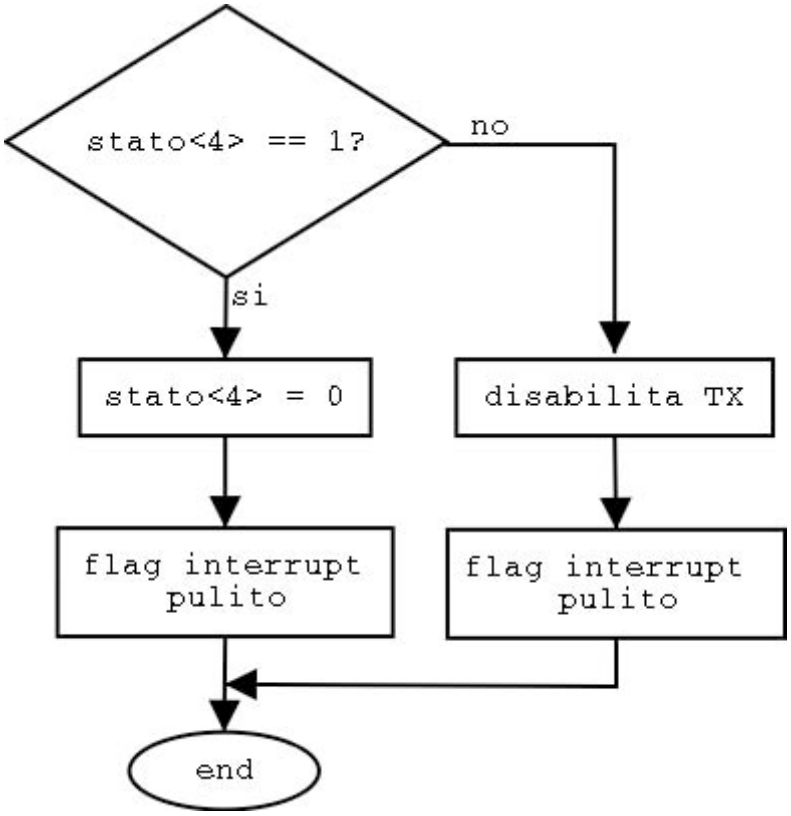
Questa procedura permette la scrittura e lettura della eeprom interna al microcontrollore. L'utente invia tramite un terminale due caratteri corrispondenti all'indirizzo che intende leggere, il pic trasmette in risposta due caratteri che rappresentano il contenuto della cella di memoria in esadecimale. A questo punto inviando il carattere corrispondente ad invio il pic restituisce il contenuto della cella successiva, mentre inviando due caratteri questi vengono scritti nella eeprom all'indirizzo precedentemente inviato.

Inoltre, se viene ricevuto il carattere ascii speciale "s", il programma darà l'ordine di scrivere il valore letto dai sensori in eeprom, modificando opportunamente la variabile *ordine.*



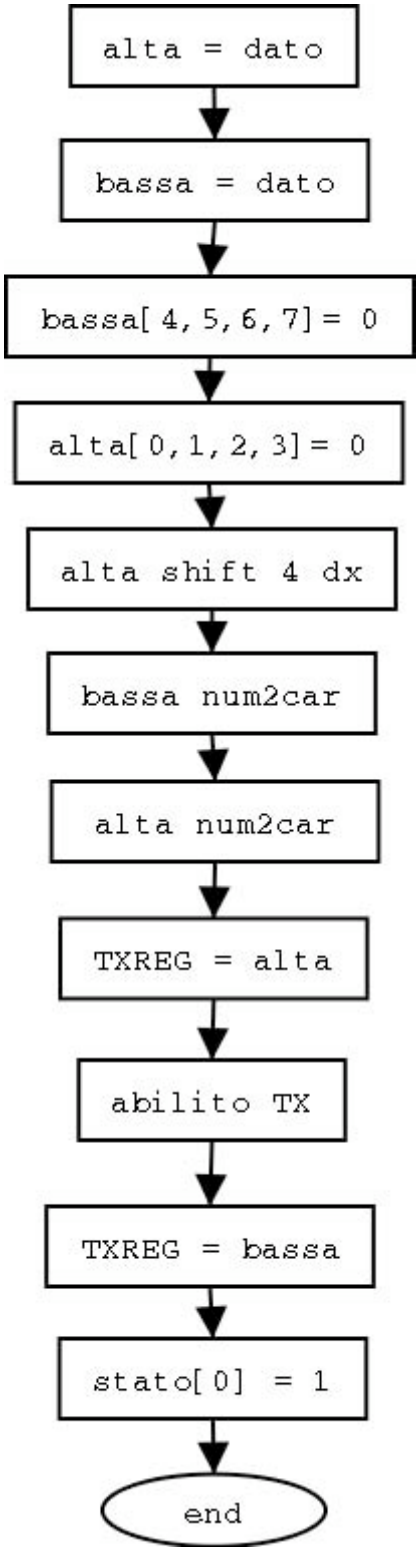
### 3.1.9. Procedura spedizione

Viene richiamata quando arriva un'interruzione dovuta alla fine della trasmissione di un carattere. Disabilita la trasmissione quando sono stati inviati i due caratteri corrispondenti alla parte alta e bassa del dato da trasmettere.




### 3.1.10. Procedura trasmetti

Trasmette il contenuto della eeprom attraverso due caratteri che rappresentano la parte alta e bassa del valore in esadecimale.



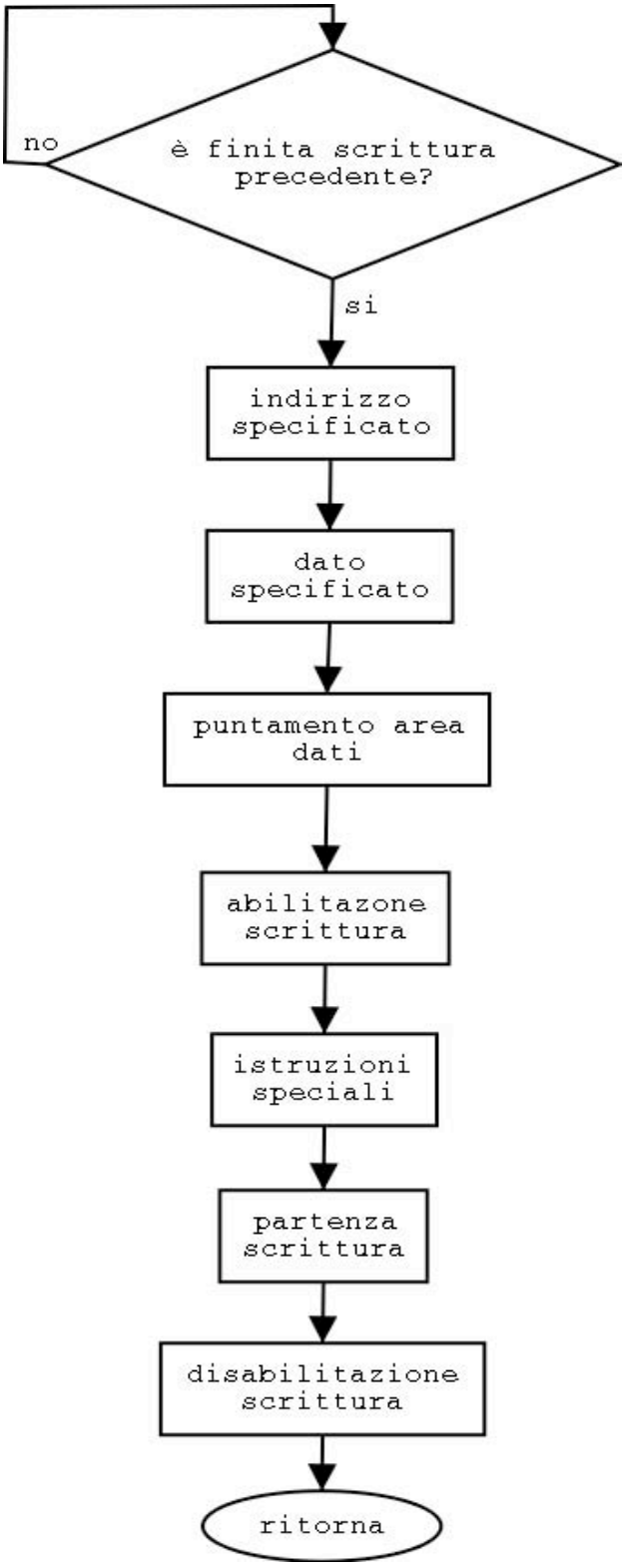

### 3.1.11. Procedura lettura

Si tratta di una funzione di utilizzo generale, serve per compiere una lettura in EEPROM, per farlo bisogna specificare l'indirizzo di lettura attraverso la variabile

*ind_eeprom*; dalla funzione saranno compiute alcune istruzioni che servono per compiere la lettura ed alla fine restituirà il risultato nella variabile *dato*.


### 3.1.12. Procedura scrittura

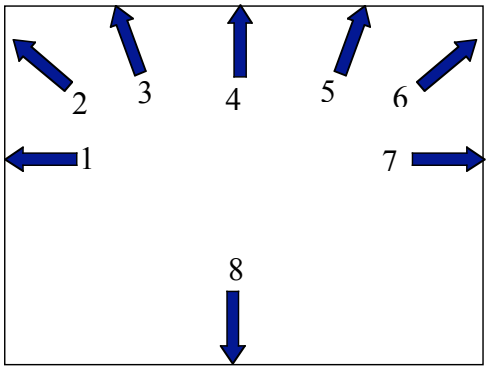
Si tratta di una procedura di utilizzo generale, serve per compiere una scrittura in EEPROM, per farlo è necessario specificare l'indirizzo nella variabile *ind_eeprom* ed il valore da scrivere nella variabile *dato*.


è finita scrittura precedente?
no
si
indirizzo specificato
dato specificato
puntamento area dati
abilitazone scrittura
istruzioni speciali
partenza scrittura
disabilitazione scrittura
ritorna

# 4. Modalità operative

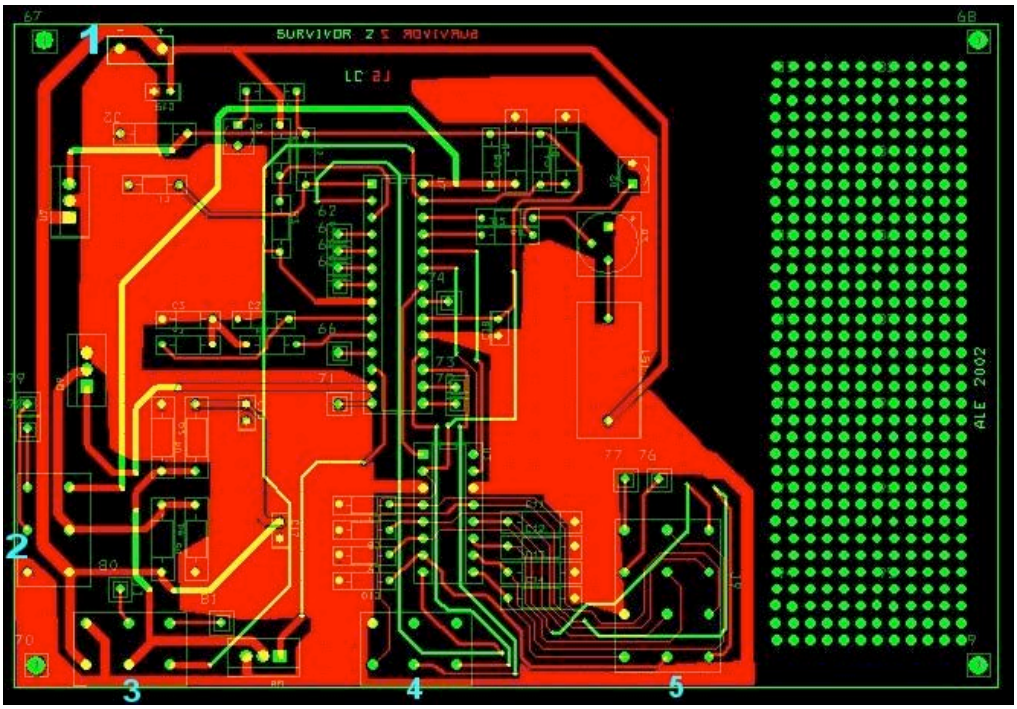
Per la comunicazione con il microcontrollore tramite la seriale è necessario utilizzare un emulatore di terminale, che permetta l'invio e la ricezione di singoli caratteri: in modalità asincrona a 9600 baud senza parità.

La disposizione dei sensori sulla sedia viene mostrata nella figura seguente:




Vengono usati 7 sensori per rilevare gli ostacoli presenti in un'area di 180° davanti alla sedia mentre viene usato un solo sensore per rilevare ostacoli dietro.

È stato costruito lo stampato della scheda elettronica per il controllo della sedia, nella figura seguente viene mostrato lo stampato a doppia faccia: in rosso il lato saldature in verde il lato componenti:



Sono stati numerati i connettori:

1. alimentazione
2. motore + interruttore sinistro
3. motore + interruttore destro
4. seriale
5. sensori

# 5. Conclusioni e sviluppi futuri

Sono state implementate tutte le funzionalità di base: come esecuzioni passi, emulazione pianto, comunicazione seriale, modifica velocità dei passi, acquisizione sensori…

Implementando delle procedure con logica fuzzy è possibile ottenere un comportamento più intelligente; che emula il comportamento di una persona. Il programma è organizzato in modo da facilitare l'aggiunta di nuove funzionalità.

# Bibliografia

Tutta la documentazione necessaria al progetto è reperibile sui datasheet dei vari componenti che si possono trovare sui siti internet sotto elencati:

[1] Microchip: "PIC16F87X Data Sheet 28/40-Pin 8-Bit CMOS FLASH Microcontrollers", www.microchip.com .

[2] Microchip: "PICmicro™ Mid-Range MCU Family Reference Manual", www.microchip.com

[3] Maxim: "Improved 8-Channel,CMOS Analog Multiplexer", www.maxim-ic.com

[4] Maxim: "Multichannel rs232 drivers/receiver", www.maxim-ic.com

[5] Sharp: "GP2Y0A02YK Long Distance Measuring Sensor", www.sharpsma.com

[6] International IOR rectifier: "IPS031 Fully protected power mosfet switch", www.irf.com

[7] Philips Semiconductors: "BYV26 series", www.semiconductors.philips.com

# Indice