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

Laboratorio di Robotica Avanzata
Advanced Robotics Laboratory

Corso di Robotica Mobile
(Prof. Riccardo Cassinis)

Riprogrammazione del robot "Ruttolo"

Elaborato di esame di:

Michele Guerrini, Igor Zanetti

Consegnato il: **18 giugno 2009**

Sommario

In questo progetto ci siamo trovati di fronte alla programmazione di un robot: "Ruttolo". Prima di preoccuparci del linguaggio di programmazione è stato necessario comprendere gli strumenti ed i meccanismi di funzionamento del robot attraverso i quali è possibile creare il codice sorgente, eseguirne la compilazione ed inviare tale programma a "Ruttolo". Con questa relazione cercheremo di rendere chiari tali meccanismi in modo da facilitare il lavoro a chi in futuro utilizzerà il robot.

1. Introduzione

L'obiettivo principale del progetto è quello di creare un ambiente di sviluppo per il robot "Ruttolo" ed una relativa documentazione. L'obiettivo secondario è invece realizzare un piccolo programma dimostrativo del funzionamento del robot.

1.1. Il robot

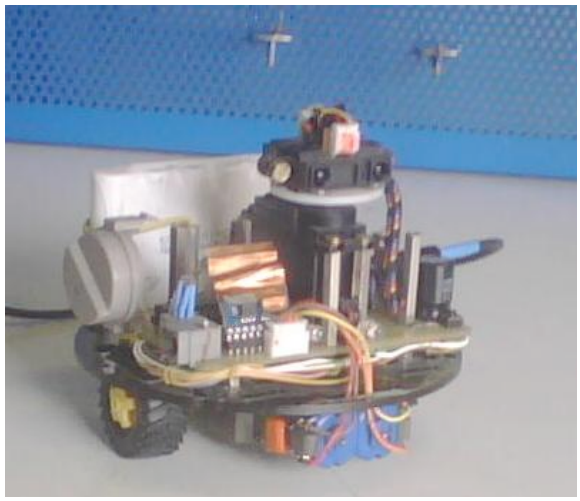

Fig. 1 - Il robot "Ruttolo"

Ruttolo monta un microcontrollore PIC versione 16F876. Il linguaggio di programmazione dei PICmicro è l'assembly ma nel tempo sono stati implementati dei compilatori per semplificarne la programmazione. Sono disponibili infatti molti compilatori di linguaggi con sintassi simili al Basic oppure compilatori di C o Pascal.

Per quanto riguarda il movimento, Ruttolo utilizza una base mobile di tipo Differential Drive. Questa è costituita da:

1. Due ruote con asse di rotazione parallelo e coincidente guidate indipendentemente da due motori elettrici;
2. Una terza ruota passiva, in questa caso una sfera metallica, necessaria per mantenere il robot in equilibrio.

In questo modo il robot può procedere lungo una linea, ruotare sul posto e muoversi lungo un arco di circonferenza.

I motori sono pilotati in corrente continua dal Dual Serial Motor Controller della Pololu attraverso una porta seriale RS-232.

Il raffreddamento del controller avviene per dissipazione di calore attraverso una sottile lamina di rame. Questo è il maggior problema del robot in quanto tende a scaldarsi troppo velocemente, andando a compromettere il suo corretto funzionamento e, di conseguenza, a limitare notevolmente la durata dei test effettuabili.

L'alimentazione è fornita da un pacco batterie da 7.2 V che viene a sua volta caricato da un alimentatore esterno.

Ruttolo è dotato di due sensori a raggi infrarosso montati su un servomotore. Il motivo che ha reso necessario avere due sensori è che il servomotore ha una corsa massima di 180°.

Oltre a questo Ruttolo ha anche una porta d'ingresso per il cavo seriale (necessario per caricare il file di programmazione), un pulsante di accensione ed uno di reset.

1.1.1. Microcontrollore PIC 16F876

Un microcontrollore è un dispositivo elettronico che, opportunamente programmato, è in grado di svolgere diverse funzioni in modo autonomo. Essenzialmente un PIC gestisce delle linee di input e di output in relazione al programma memorizzato.

Questi dispositivi implementano su un unico chip:

1. Una cpu RISC;
2. Una memoria di programma (EPROM-EEPROM);
3. Una memoria di lavoro RAM;
4. Porte di I/O;
5. Contatori, timer, convertitore A/D;
6. Pwm ed interfacce di comunicazione di vari tipi.

Col termine RISC, acronimo di Reduced Instruction Set Computing, si intende che le elaborazioni della cpu utilizzano un set ridotto di istruzioni (poche decine) eseguite molto velocemente.

In commercio si trovano diversi modelli di PIC a seconda della complessità e delle funzioni implementate; i dispositivi per esempio si differenziano per la quantità di memoria disponibile o per la quantità di Timer a disposizione. A bordo del robot Ruttolo è presente un PIC modello 16F876 che ha le caratteristiche mostrate in figura 2:

Key Features PICmicro™ Mid-Range Reference Manual (DS33023)	PIC16F876
Operating Frequency	DC - 20 MHz
RESETS (and Delays)	POR, BOR (PWRT, OST)
FLASH Program Memory (14-bit words)	8K
Data Memory (bytes)	368
EEPROM Data Memory	256
Interrupts	13
I/O Ports	Ports A,B,C
Timers	3
Capture/Compare/PWM Modules	2
Serial Communications	MSSP, USART
Parallel Communications	—
10-bit Analog-to-Digital Module	5 input channels
Instruction Set	35 instructions

Fig. 2 - Dati tecnici PIC16F876

Nelle figure 3 e 4 vengono mostrate la piedinatura del circuito integrato e una tabella di descrizione dei vari pin presa direttamente dal datasheet del pic [2]:

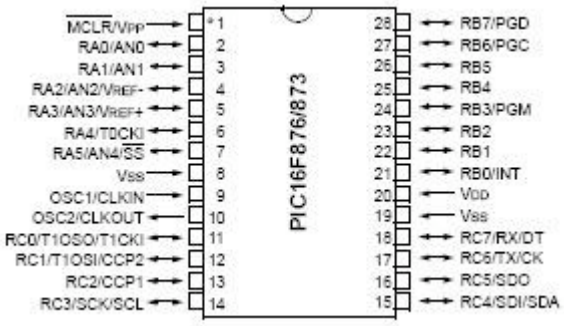

Fig. 3 - Piedinatura PIC16F876

Pin Name	DIP Pin#	SOIC Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	9	9	I	ST/CMOS(3)	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	10	10	O	—	Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, the OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
MCLR/VPP	1	1	I/P	ST	Master Clear (Reset) input or programming voltage input. This pin is an active low RESET to the device.
					PORTA is a bi-directional I/O port.
RA0/AN0	2	2	I/O	TTL	RA0 can also be analog input0.
RA1/AN1	3	3	I/O	TTL	RA1 can also be analog input1.
RA2/AN2/VREF-	4	4	I/O	TTL	RA2 can also be analog input2 or negative analog reference voltage.
RA3/AN3/VREF+	5	5	I/O	TTL	RA3 can also be analog input3 or positive analog reference voltage.
RA4/T0CKI	6	6	I/O	ST	RA4 can also be the clock input to the Timer0 module. Output is open drain type.
RA5/SS/AN4	7	7	I/O	TTL	RA5 can also be analog input4 or the slave select for the synchronous serial port.
					PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.
RB0/INT	21	21	I/O	TTL/ST(1)	RB0 can also be the external interrupt pin.
RB1	22	22	I/O	TTL	
RB2	23	23	I/O	TTL	
RB3/PGM	24	24	I/O	TTL	RB3 can also be the low voltage programming input.
RB4	25	25	I/O	TTL	Interrupt-on-change pin.
RB5	26	26	I/O	TTL	Interrupt-on-change pin.
RB6/PGC	27	27	I/O	TTL/ST(2)	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming clock.
RB7/PGD	28	28	I/O	TTL/ST(2)	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming data.
					PORTC is a bi-directional I/O port.
RC0/T1OSO/T1CKI	11	11	I/O	ST	RC0 can also be the Timer1 oscillator output or Timer1 clock input.
RC1/T1OSI/CCP2	12	12	I/O	ST	RC1 can also be the Timer1 oscillator input or Capture2 input/Compare2 output/PWM2 output.
RC2/CCP1	13	13	I/O	ST	RC2 can also be the Capture1 input/Compare1 output/ PWM1 output.
RC3/SCK/SCL	14	14	I/O	ST	RC3 can also be the synchronous serial clock input/output for both SPI and I2C modes.
RC4/SDI/SDA	15	15	I/O	ST	RC4 can also be the SPI Data In (SPI mode) or data I/O (I2C mode).
RC5/SDO	16	16	I/O	ST	RC5 can also be the SPI Data Out (SPI mode).
RC6/TX/CK	17	17	I/O	ST	RC6 can also be the USART Asynchronous Transmit or Synchronous Clock.
RC7/RX/DT	18	18	I/O	ST	RC7 can also be the USART Asynchronous Receive or Synchronous Data.
VSS	8, 19	8, 19	P	—	Ground reference for logic and I/O pins.
VDD	20	20	P	—	Positive supply for logic and I/O pins.

Legend: I = input O = output — = Not used I/O = input/output TTL = TTL input P = power ST = Schmitt Trigger input

Fig. 4 - Tabella piedinatura PIC

La scrittura nella memoria del PIC avviene tramite programmazione seriale o parallela sfruttando due pin (tipicamente RC7 e RC6). Un pin è necessario per il segnale di clock, l’altro per il transito dei dati (che possono essere bidirezionali).

Esistono due tipi di programmazione: “Out of circuit” e “In-circuit”. La prima tipologia è valida per tutti i PIC mentre la seconda è supportata solo da alcune famiglie e verrà trattata nel capitolo successivo.

La programmazione Out of circuit, ossia col microcontrollore separato dal circuito finale, avviene tramite l’utilizzo di un apposito dispositivo elettronico (chiamato “programmatore”), nel quale viene inserito il PIC, che si interfaccia al calcolatore tramite porta seriale.

Per effettuare questa scrittura è quindi necessario scrivere il codice sorgente, compilarlo per ottenere un file di estensione .hex, inserire il PIC nel programmatore ed avviare la scrittura utilizzando un

programma in grado di gestire i segnali tra calcolatore e programmatore leggendo il file ottenuto dalla compilazione.

Quello che occorre (Figura 5) è quindi:

- Calcolatore
- Programmatore
- Software di gestione del programmatore

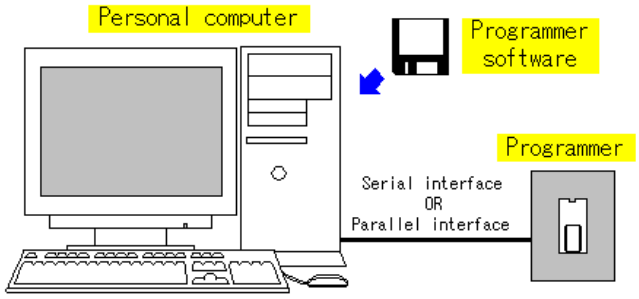

Fig. 5 - Componenti necessari per la programmazione

1.1.2. Bootloader e programmazione On-board

Per alcune famiglie di PIC, tra cui quella usata in questo progetto, è possibile utilizzare una diversa tipologia di programmazione detta In-ciruit o anche On-board. Come si può intuire dal termine, questa modalità offre il vantaggio di poter programmare il microcontrollore senza doverlo togliere dal circuito in cui viene utilizzato. Per effettuare la scrittura in memoria si utilizza un'interfaccia di programmazione che gestisce la comunicazione tra PIC e calcolatore provvedendo a convertire i segnali provenienti dalla porta seriale RS-232 del calcolatore.

Questa tipologia di scrittura viene realizzata attraverso l'utilizzo di un software chiamato Bootloader, il quale è composto da due parti: una parte risiede sul PIC ed è normalmente chiamata Bootloader, l'altra si trova sul calcolatore e viene chiamata programma di caricamento.

La parte che risiede sul PIC deve essere scritta in memoria attraverso l'utilizzo del programmatore. Allo start-up o al reset del PIC questo software, per un breve periodo di tempo, verifica la presenza di attività sulla porta seriale generata dal programma di caricamento e, in caso positivo, avvia il trasferimento del file di programma .hex dal PC e lo scrive nella memoria flash del PIC.

Nel caso non trovasse attività sulla porta seriale, dopo un periodo di tempo predefinito (qualche centinaio di millisecondi), il bootloader va in time out e viene eseguito il programma residente nella memoria del PIC.

Il bootloader che risiede sul PIC utilizza uno spazio di memoria riservato (256 bytes) che non deve essere sovrascritto dal codice utente. Per proteggere tale spazio di memoria è necessario inserire nel programma una particolare istruzione. Per il PIC usato in questo progetto tale riga di codice è:

```
#org 0x1F00, 0x1FFF void loader16F876(void) {}
```

In figura 6 mostriamo come si presenta la memoria del PIC con e senza bootloader:

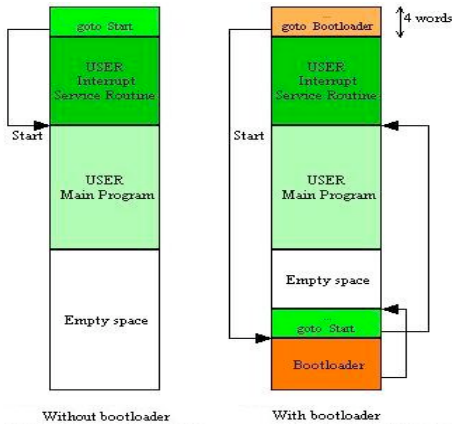

Fig. 6 - Memoria senza e con Bootloader

Dalla figura si capisce graficamente ciò che è stato spiegato precedentemente. Senza il bootloader viene eseguito direttamente il codice sorgente. Nel secondo caso, invece, l’esecuzione del codice utente avviene successivamente all’esecuzione del bootloader che controllerà prima la presenza di attività sulla porta seriale e si comporterà come spiegato in precedenza.

1.1.3. Micro Dual Serial Motor Controller

Per il controllo dei motori Ruttolo utilizza un micro controller della Pololu modello SMC02B. Questo controller consente di pilotare due motori in corrente continua fino ad 1 A di assorbimento, con 127 passi di velocità in due direzioni diverse. Con tale dispositivo è possibile regolare, attraverso l’uso di semplici comandi, la velocità dei singoli motori per ottenere le stesse velocità di rotazione e permettere al robot di percorrere un movimento rettilineo. In figura 7 è possibile vedere il circuito di collegamento della configurazione a 2 motori:

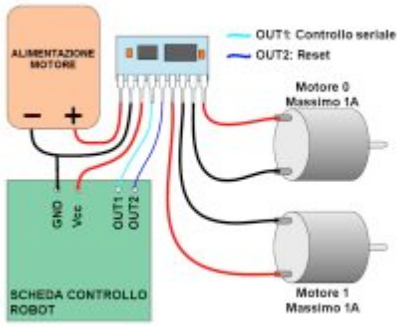

Fig. 7 - Schema collegamento controller e motori

Il pilotaggio dei motori avviene inviando al controller una sequenza di 4 bytes così strutturata :

1. Byte 1. Byte iniziale: rappresenta l’inizio del comando e deve sempre essere 0x80 (128 in decimale);
2. Byte 2. Tipo di dispositivo: Identifica il tipo di dispositivo al quale si riferisce il comando. Per il controller dei motori deve essere 0x00;
3. Byte 3. Numero motore e direzione: Questo byte identifica il motore da controllare e la direzione di rotazione del motore. Per ottenere il valore si può utilizzare una semplice regola: si prende il numero del motore (per esempio 1), si moltiplica per 2 e si somma 1 nel caso di rotazione in avanti (quindi: 1x2 + 1 = 3. Il byte sarà quindi 0x03);
4. Byte 4. Velocità: Imposta la velocità di rotazione del motore. Il possibile range di valori va da 0x00 a 0x7F (da 0 a 127 in decimale). 0x00 spegne il motore mentre 0x7F rappresenta la velocità massima.

Nella seguente tabella sono visualizzate le caratteristiche tecniche del Micro Controller:

Caratteristiche tecniche del Micro Dual Serial Motor Controller	
Velocità seriale	1.200 – 19.200 baud (auto detected)
Tensione motori	1.8 – 9.0 V
Corrente motori	1 A con 2 motori, 2 A con singolo motore
Tensione logica	2.6 – 5.5 V
Frequenza PWM	600 Hz con 2 motori, 750 Hz con singolo motore
Passi velocità	127 avanti, 127 indietro, off
Motori	1 o 2
Dimensioni	22.8 x 11.4 mm

1.1.4. Servomotore

Il servo è un motore con delle caratteristiche particolari. Quello montato sul robot è un servo da modellismo, solitamente usato per azionare acceleratore, sterzo, vele delle barche ecc. Un servo è costituito da un motore completo di riduzione meccanica, da un sistema di feedback per la posizione dell'asse di uscita e dall'elettronica di controllo, il tutto racchiuso in un unico contenitore. Da questa scatola escono unicamente l'asse della riduzione meccanica, al quale va collegato il dispositivo da controllare (nel nostro caso i sensori), i due fili di alimentazione e un filo per il controllo. In figura 8 si può vedere lo schema funzionale di un servo generico.

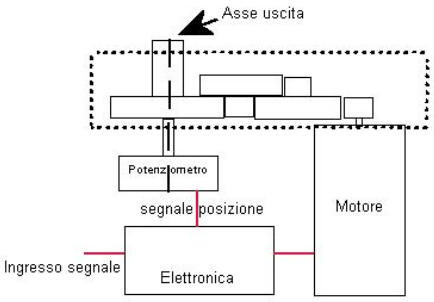

Fig. 8 - Schema funzionale di un servo motore

L'albero del servo si posiziona a diversi angolazioni (comprese tra 0° e 180°) a seconda della durata dell'impulso inviato all'ingresso di controllo. Il segnale che si utilizza è un PWM standard. L'impulso viene generato ogni 20 ms, il che vuol dire con una frequenza di 50 Hz, rimanendo alto tipicamente per valori di tempo compresi tra un minimo di 1 ms ed un massimo di 2 ms.

1.2. Ambiente di sviluppo

Per questo progetto abbiamo scelto Windows come sistema operativo perché a noi era più familiare e di facile utilizzo. Per sviluppare il software del robot è necessario utilizzare due componenti:

- PCW C Compiler : tool di sviluppo e compilazione dei file di programmazione in linguaggio C, da caricare successivamente su Ruttolo. Per il progetto abbiamo utilizzato la versione del software che ci è stata consegnata assieme al robot, ma è ovviamente possibile utilizzare versioni più aggiornate reperibili via web. [4]
- Pic bootloader (per windows) : tool per caricare il software in memoria al robot. I file necessari sono quelli con estensione ".hex" (ottenuti attraverso la compilazione del file C). Per caricare un file è sufficiente selezionarlo attraverso la funzione Search, indicare la porta seriale da

utilizzare e dare il comando Write. Successivamente per poter avviare la scrittura è necessario resettare il robot.

1.3. Materiale utile

Nel caso in cui il calcolatore non sia dotato di una porta seriale (come nel nostro caso) è possibile utilizzare un convertitore USB Serial Converter che andrà collegato all’interfaccia di programmazione di Ruttolo. In questo caso è ovviamente necessario installare i driver della periferica. Se si utilizza il convertitore bisogna prestare attenzione a quando si utilizza il PIC bootloader, in quanto la porta USB viene comunque riconosciuta dal programma come una porta seriale.

2. Il problema affrontato

Il problema che abbiamo dovuto affrontare è stato quello di riprogrammare il robot “Ruttolo” in modo che si potesse muovere in modo casuale all’interno di un ambiente non privo di ostacoli schivandoli opportunamente.

3. La soluzione adottata

La soluzione è stata quella di creare del codice in linguaggio C che permettesse al robot di muoversi nell’ambiente, individuare gli ostacoli attraverso i sensori di cui è dotato ed evitarli prima di entrarne in collisione. Tale programma verrà spiegato più nel dettaglio nel capitolo successivo, ma qui ci teniamo a precisare che ci possono essere vari modi per ottenere una valida soluzione e che noi ve ne mostreremo solo un semplice esempio.

Per quanto riguarda il nostro obiettivo, possiamo dire di ritenerci discretamente soddisfatti dell’esito del nostro programma. Ruttolo è in grado di muoversi all’interno di una stanza evitando pressoché ogni tipo di ostacolo, a patto che questi siano individuabili dai sensori sia in termini di altezza che di distanza minima (circa 80 mm) nel caso in cui venga messo un ostacolo improvvisamente sul suo percorso.

La soluzione presenta in ogni caso alcuni problemi, dovuti soprattutto alla struttura hardware del robot, che verranno comunque trattati con maggior dettaglio nel capitolo successivo.

4. Modalità operative

In questo paragrafo andremo a spiegare il funzionamento del robot in esame, i vari passi per metterlo in funzione, gli strumenti utilizzati e i problemi riscontrati.

4.1. Componenti necessari

Per prima cosa forniamo un elenco di tutti componenti che sono risultati utili e necessari per portare a termine i nostri obiettivi :

1. Il Robot “Ruttolo”;
2. Un alimentatore : le batterie hanno lunghi tempi di ricarica e durano relativamente poco, quindi bisogna cercare di effettuare test di breve durata e, quando possibile, cercare di effettuarli con l’alimentatore inserito;
3. L’interfaccia di programmazione : collegata direttamente alla porta serial del pc o attraverso un convertitore;
4. Un adattatore USB Serial Converter : nel caso in cui, come detto precedentemente, il calcolatore non abbia una porta seriale;

5. Un calcolatore: il calcolatore utilizzato non deve avere particolari caratteristiche. L'unica richiesta è che deve utilizzare windows per essere sicuri del funzionamento dei software presi in esame in questo progetto;
6. Il software necessario per la programmazione del robot;
7. Cacciaviti e attrezzi vari per gestire la parte hardware del robot.

4.2. Passi per la programmazione

Per poter sviluppare un programma per il robot "Ruttolo" abbiamo prima dovuto affrontare e risolvere alcuni problemi e difficoltà.

La situazione che più ci ha messo a disagio è stata la totale mancanza di documentazione del robot, il che ci ha obbligato a dover procedere per tentativi. Questo ci ha costretto sia a dover fare delle prove per individuare quali fossero i programmi da utilizzare per lo sviluppo, che studiare il software di esempio caricato precedentemente sul robot per capire le modalità base di programmazione.

A questo punto, si è passati a sviluppare il codice: come compilatore abbiamo deciso di utilizzare direttamente PCW senza appoggiarci a tool esterni (come per esempio Dev C++) cosa che, secondo noi, risulta giustificabile unicamente se è necessario sviluppare software relativamente complessi. Il file compilato è stato poi caricato nella memoria di Ruttolo, attraverso il PIC bootloader, ed immediatamente ne è iniziata l'esecuzione sul robot.

Per riassumere facciamo un breve elenco che indichi, in maniera molto semplice e schematica, la sequenza di passi da compiere per sviluppare del codice e mettere in funzione Ruttolo:

1. Scrivere il file C con PCW compiler;
2. Compilare il file C sempre con PCW compiler;
3. Selezionare con PIC bootloader il file compilato (estensione .hex)
4. Selezionare la porta COM;
5. Premere Write ed accendere o resettare il robot;
6. Scollegare l'interfaccia di programmazione da Ruttolo;
7. Resettare il robot.

Questi fondamentalmente sono i passi necessari per svolgere correttamente il lavoro. Per i vari problemi riscontrati e i suggerimenti sull'utilizzo rimandiamo alla consultazione del relativo capitolo "Avvertenze d'utilizzo"; ora nel paragrafo successivo andiamo a mostrare il codice che abbiamo prodotto fornendo una breve spiegazione sui metodi utilizzati.

4.2.1. Il codice realizzato

In questo paragrafo mostriamo il programma dimostrativo creato per Ruttolo. Essendo una semplice dimostrazione non ci siamo addentrati molto nei particolari di programmazione, ma riteniamo che dare una spiegazione di alcune funzioni utilizzate e sulla struttura del codice possa essere di aiuto per un progetto futuro.

Mostriamo per prima cosa il codice nella sua versione iniziale :

```
#include "confPICC.h"
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

//  -----Funzione per impostare la velocità dei motori-------
void set_speed (signed int velsin, signed int veldes)
{
   putc (0x80);
   putc (0x00);
```

```
      if (velsin < 0)
         putc (0x01);
      else
         putc (0x00);
      putc (abs(velsin));

      putc (0x80);
      putc (0x00);
      if (veldes < 0)
         putc (0x03);
      else
         putc (0x02);
      putc (abs(veldes));
   }

   // -----Funzione per leggere la distanza dal sensore anteriore-------
   int get_front_ir ()
   {
      set_adc_channel(0);
      delay_ms(1);
      return Read_ADC();
   }

   // ----- Funzione per leggere la distanza dal sensore posteriore-------
   int get_rear_ir ()
   {
      set_adc_channel(1);
      delay_ms(1);
      return Read_ADC();
   }

   // ----- Funzione per direzionare la testa del robot ad una certa angolazione----
   void dir_ir (signed int angle)
   {
      angle +=90; //necessario per impostare l'angolazione 0° dritta davanti
      set_pwm1_duty(angle);
   }

   // ------ Funzione che fa un controllo della distanza di oggetti ad angolazioni
   // -20°, 0°, 20° dalla posizione attuale del servo
   int look_in_front()
   {
      int minimo, lettura;
      dir_ir(-20);
      delay_ms(300);
      minimo=get_front_ir();
      dir_ir (0);
      delay_ms(300);
      lettura=get_front_ir();
      if (lettura > minimo) minimo=lettura;
      dir_ir (20);
      delay_ms(300);
      lettura=get_front_ir();
      if (lettura > minimo) minimo=lettura;
      return minimo;
   }

   #define SIZE 25

   void main()
   {
      int rightdist,leftdist;
      int distanza;


       output_low(PIN_B2);
       setup_ccp1(CCP_PWM);   // Configure CCP1 as a PWM
       setup_timer_2(T2_DIV_BY_16, 255, 1);
       dir_ir(0); //Dirigilo al centro davanti


      setup_port_a( ALL_ANALOG );
```

```
        setup_adc( ADC_CLOCK_INTERNAL );

          while (true)
           {
              distanza = look_in_front();
              while (distanza < 100)
                 {
                    set_speed (50,50);
                    delay_ms(1000);

                  if(look_in_front() > 100)
                       {
                       set_speed(0,0);
                       }

                    delay_ms(1000);
                    set_speed(0,0);
                    distanza = look_in_front();

                 }//fine while interno

              set_speed (0,0);
              delay_ms(1000);

              dir_ir(-50);
              delay_ms(500);
              leftdist=get_front_ir();
              dir_ir(50);
              delay_ms(500);
              rightdist=get_front_ir();

              dir_ir(0);

              if (leftdist>rightdist)
                 set_speed (50,-50);
              else
                 set_speed (-50,50);

              delay_ms(1000);
              set_speed (0,0);
              delay_ms(1000);
          }//fine while esterno
    }
```

Per prima cosa facciamo notare l'inclusione del file "confPICC.h". Questo file contiene le righe di codice necessarie per configurare il microcontrollore:

```
    #include <16F876.h>
    #device *=16 //usa puntatore a 16 bit

    //imposta il convertitore A/D a 8 bit e così il numero di bit che read_adc()
    //dovrebbe ritornare
    #device adc=8
    #use delay(clock=4000000)//velocità processore

    //imposta i bit di configurazione del PIC
    #fuses NOWDT,XT, NOPUT, NOPROTECT, BROWNOUT, LVP, NOCPD, NOWRT, NODEBUG
    //codice di protezione del bootloader per 8k 16F876/7
    #org 0x1F00, 0x1FFF void loader16F876(void) {}
    #use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7, PARITY=N, BITS=8)
```

Facciamo soprattutto notare la presenza del frammento di codice necessario per la protezione del bootloader e le impostazioni della porta seriale.

A questo punto, per facilitare la lettura del codice realizzato, forniamo una breve spiegazione delle funzioni utilizzate:

- Set_speed : funzione che invia la sequenza di bytes necessaria per pilotare i motori al controller pololu. Facciamo notare che il senso di marcia del robot e il verso di rotazione dei motori sono l’uno l’opposto dell’altro;
- Get_front_ir : funzione utilizzata per ottenere la misurazione dal sensore ad infrarosso anteriore;
- Get_rear_ir : funzione utilizzata per ottenere la misurazione dal sensore ad infrarosso posteriore;
- Dir_ir : direziona la testa del robot ad una data angolazione attraverso una rotazione dell’asse del servo motore. Il parametro angle è usato per regolare il duty cycle dell’impulso utilizzato per comandare il servo.
- Look_in_front : Funzione che fa un controllo della distanza di oggetti ad angolazioni -20°, 0°, 20° dalla posizione attuale del servo.

Questi metodi fanno riferimento, al loro interno, a funzioni specifiche del microcontrollore PIC. Riteniamo quindi essere d’aiuto dare una breve spiegazione dei metodi più usati e utili :

- Set_adc_channel(n) : funzione necessario per selezionare i sensori ad infrarosso anteriori (0) o posteriori (1) montanti sulla testa del robot;
- Read_adc : funzione per effettuare la lettura del valore dal sensore selezionato;
- Set_pwm1_duty (angle) : funzione per ruotare la testa di Ruttolo di un certa angolazione;
- Setup_adc e Setup_port_a : funzioni necessari per configurare il convertitore analogico/digitale;
- Setup_ccp1(CCP_PWM) : inizializza il ccp come generatore di un segnale PWM necessario per controllare il servo motore;
- Setup_timer_2 : inizializza il timer di sistema.

Per realizzare programmi più complessi rimandiamo alla consultazione del manuale PIC che contiene un elenco dei comandi che è possibile utilizzare.

Tale codice ha subito successivamente una modifica in seguito ad alcuni test effettuati. Motivazioni e spiegazioni di tali cambiamenti sono illustrati nel paragrafo seguente.

4.3. Test sui sensori del robot

Nella fase di testing del robot ci siamo accorti di alcuni comportamenti strani dei quali non riuscivamo a capirne la motivazione.

Durante la navigazione del robot all’interno del laboratorio, Ruttolo schivava correttamente solo alcuni ostacoli mentre altri sembrava non fosse in grado di vederli. Come diretta conseguenza abbiamo pensato a due possibili ipotesi di malfunzionamento :

- Errore nella programmazione;
- Malfunzionamento dei sensori.

Dopo aver ricontrollato il codice e testato meglio il suo effetto sul robot, ci siamo resi conto che non era quello il problema in questione per cui abbiamo deciso di spostare la nostra attenzione sui sensori.

Per poter capire il problema è necessario avere ben chiaro il principio di funzionamento fisico dei sensori ad infrarosso. Il sensore funziona sul principio del sonar (figura 9). Un led IR (Tx) collocato sul robot emette un raggio di luce infrarossa. Se il raggio intercetta un ostacolo, viene riflesso verso il robot, ed è captato da un fotodiodo (Rx):

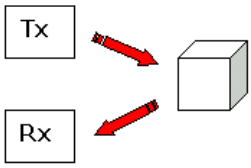

Fig. 9 - Principio funzionamento sensori

La particolarità dei raggi infrarosso sta nel fatto che il loro funzionamento non viene influenzato dall'illuminazione ambientale e dal colore degli oggetti. Per evitare di rilevare per errore eventuali sorgenti di calore la luce viene modulata attraverso l'utilizzo di un oscillatore. Questo tipo di sensori fornisce un valore di tensione che varia con la distanza dell'ostacolo: più questo è lontano e più la tensione fornita è piccola.

A questo punto, per verificarne il corretto funzionamento, l'idea che ci è venuta in mente è stata quella di farci passare dal robot le misurazioni effettuate dai sensori. Per fare questo è stato ovviamente necessario tenere Ruttolo collegato al calcolatore. Dopo alcuni tentativi, ricerche sul manuale e prove di alcuni metodi, abbiamo trovato un'utility di PCW C Compiler chiamata "Serial port monitor" che ci permetteva di visualizzare a video gli eventuali output del robot.

A questo punto abbiamo inserito nel codice degli appositi comandi che dessero come output le distanze misurate dai sensori.

Una precisazione importante è che per far funzionare lo strumento 'Serial port monitor' è stato necessario impostare nelle opzioni la porta e il baud-rate (nel nostro caso 9600) con cui avveniva la comunicazione.

Durante il test abbiamo verificato che i sensori misuravano una distanza minima a fondo scala che va via via crescendo più l'ostacolo si avvicina. Quando l'ostacolo raggiunge una distanza di 100 mm il sensore restituisce un valore di 120 e continua a crescere fino ad una distanza di 80 mm circa dove misura 130. Da questo punto in poi la misura ricomincia a scendere fino ad arrivare ad un valore minimo di 1 se l'ostacolo è contro il sensore. Nel grafico in figura 10 possiamo vedere il risultato dei valori medi misurati a varie distanze. Si può notare come più ci si allontana dal sensore più la granularità e la precisione diminuiscono, stesso problema si ha quando ci si avvicina troppo (sotto gli 80 mm).

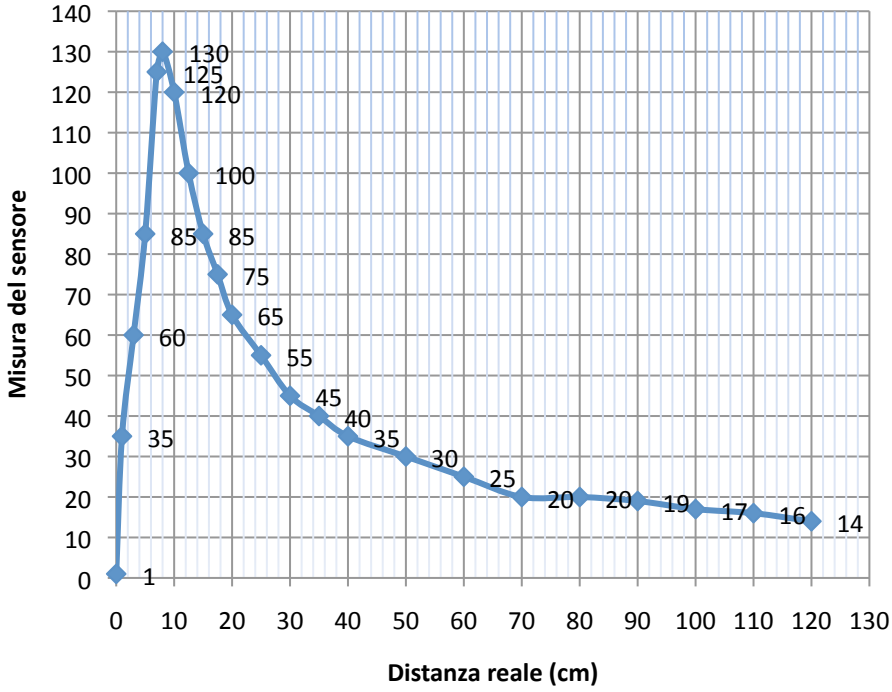

Fig. 10 - Grafico delle rilevazioni dei sensori del robot

Questo risultato poteva ovviamente essere ottenuto dalla consultazione della documentazione dei sensori. Il test ci ha però permesso di verificare anche il loro corretto funzionamento.

4.3.1. Modifiche al codice

In seguito all’esito del test ci siamo preoccupati di modificare il programma in modo che Ruttolo fosse davvero in grado di evitare gli ostacoli incontrati sul suo cammino.

L’unica parte del codice che è stata modificata è quella relativa al secondo while, all’interno del metodo main, dove ora la distanza viene continuamente misurata mentre il robot avanza e, quando si accorge di aver superato il valore 95, si ferma ed esce dal while per effettuare un cambio di direzione.

La chiave risolutiva è appunto quella di aver aumentato notevolmente la frequenza delle misurazioni in modo da essere sicuri che il robot rilevi l’ostacolo prima di esserne troppo vicino.

Qui sotto vi mostriamo il frammento di codice che ha subito modifiche :

```
while (distanza < 95)
{  set_speed (50,50);
   dir_ir(0);
   distanza = get_front_ir();
   if(distanza > 95)
   {
   set_speed(0,0);
   }
}
```

4.4. Avvertenze

In questo paragrafo andremo a creare un elenco dei vari problemi riscontrati con le relative soluzioni (se esistono) per facilitare chi in futuro dovrà lavorare con Ruttolo.

Per quanto riguarda i motori ci sono due diversi problemi:

1. Quello più fastidioso (e probabilmente di difficile soluzione) è il surriscaldamento del micro controller. Dopo un utilizzo del robot, anche relativamente breve, il sistema entra in protezione ed attua una sorta di limitazione sui motori: questo comporta che il robot tenderà a muoversi in maniera strana in quanto il primo motore a subire questa limitazione è quello destro.
2. Un altro problema è di origine elettrica: i motori e la porta seriale che si trovano a bordo di Ruttolo condividono la stessa linea di trasmissione dati. Ciò comporta che, quando il robot è collegato al calcolatore, questo non è in grado muoversi.

In relazione al problema precedente, facciamo notare che quando la copia del file in memoria su ruttolo termina, ha subito inizio l'esecuzione del codice stesso. Questo comporta che, se per caso ci si dimenticasse di scollegare il cavo seriale, Ruttolo non potrebbe muoversi nonostante la corretta esecuzione del codice. In più, per far partire tale esecuzione dall'inizio una volta che il robot viene scollegato, è necessario effettuare un reset.

Sempre in riferimento al gruppo motori/ruote è importante controllare sempre il loro corretto funzionamento. Per esempio a noi è successo che una ruota non funzionasse nonostante il motore girasse correttamente : il motivo è che si era spostato il pignone del motore e, di conseguenza, non entrava più in contatto con gli ingranaggi della ruota.

Altra avvertenza riguarda il tool di sviluppo PCW. Sia in caso di installazione ex-novo che di spostamento dalla cartella iniziale, è necessario indicare al software il percorso delle cartelle "Include" e "Driver" (le quali sono sottocartelle del percorso principale del software) per permettere al compilatore di trovare gli header file e i driver necessari. Tale operazione viene eseguita, in maniera molto intuitiva, attraverso il menù opzioni del tool stesso.

5. Conclusioni e sviluppi futuri

Come conclusione del progetto vogliamo effettuare alcune piccole considerazioni :

- In primo luogo ci è sembrato interessante potersi occupare della programmazione di un robot anche se molto piccolo e con degli evidenti limiti sia funzionali che di progettazione. Questo ci ha permesso di toccare con mano alcuni concetti che sono stati appresi durante le lezioni aiutandoci a comprendere meglio (anche se in piccola parte) la logica con cui un robot mobile può agire;
- Ci siamo resi conto di come sia importante l'ambiente in cui un robot si muove e come, in fase di progettazione, bisogna aver chiaro lo scopo per cui esso viene creato cosi da dotarlo delle funzionalità necessarie per raggiungere gli obiettivi prefissati. A volte può anche essere necessario effettuare delle modifiche all'ambiente per migliorare le funzionalità del robot stesso;
- Infine abbiamo trovato costruttiva la necessita di sistemare o sostituire alcuni componenti meccanici del robot a causa di malfunzionamenti apparsi nel corso del progetto.

Per quanto riguarda il futuro, basandoci sul lavoro compiuto, possiamo proporre alcuni sviluppi del progetto che potrebbe avere un certo interesse:

- Progettare un ambiente di programmazione adatto a funzionare su un sistema operativo molto usato e open source come "Linux";
- Sistemare alcuni problemi di dissipazione del dispositivo in modo da poterlo utilizzare, senza problemi, per tempi prolungati e senza continue interruzioni;
- Creare programmi più complessi di quello mostrato in questo progetto per poter far svolgere a Ruttolo compiti più vari ed interessanti.

Bibliografia

[1] Documentazione ufficiale del compilatore PWC “C compiler Reference Manual”.

[2] Data Sheet del PIC 16F87X
http://ww1.microchip.com/downloads/en/DeviceDoc/30292c.pdf

[3] Sito web dove reperire il pic bootloader con relativa spiegazione d’uso
http://www.microchipc.com/PIC16bootload/index.php

[4] Sito web contenente materiale aggiornato e supporto tecnico
http://www.ccsinfo.com/

Indice