UNIVERSITA' DEGLI STUDI DI ROMA "TOR VERGATA"

FACOLTA' DI INGEGNERIA

Tesi di Laurea in Ingegneria Informatica

"PROGETTAZIONE E REALIZZAZIONE DI ROBOT PER LO STUDIO DEL MOTO DI FORMAZIONI"

Relatore
Ing. Francesco Martinelli

Laureando
Danilo Luzi

Anno Accademico 2005/2006

ai miei genitori..

Indice

Capitolo 1

Introduzione

1 Introduzione

1.1 Generalità

Lo scopo del presente lavoro è quello di creare una generazione di Robot a basso costo e con le seguenti caratteristiche: versatilità, espandibilità, semplici da riprodurre e manutenere ,elevata precisione ed autonomia. Tali robot, denominati Octagon, sono creati con lo scopo di rappresentare uno strumento di elevato valore didattico e di consentire l'approccio al mondo della robotica anche a figure senza conoscenze specifiche in tale campo, rappresentando così un valido canale di comunicazione interdisciplinare. L'intelligenza di queste macchine infatti non risiede nelle MCU degli stessi Robot, ma potrà essere progettata e testata nella macchina dell'utente e nel linguaggio che a lui è più congeniale. Tale scelta è stata effettuata non solo per abbassare i costi dei robot evitando l'installazione di potenti processori con relative memorie centrali e di massa che ne avrebbero abbassato notevolmente l'autonomia, ma anche per non costringere l'utente finale ad acquisire le conoscenze necessarie a modificare il firmware dei robot. Inoltre come in tutti i sistemi sottoposti a particolari sollecitazioni, un aumento della complessità avrebbe comportato un probabile incremento degli interventi di manutenzione. Per i motivi di cui sopra i robot sono in grado di comunicare con la macchina dell'utente tramite un modulo radio con una portata di 250 metri in linea d'aria ed un consumo molto ridotto. La capacità di comunicazione è un requisito essenziale nello studio di tutti quegli algoritmi di intelligenza distribuita multi agente in cui si prevede la capacità degli agenti di comunicare fra di loro per tenere traccia delle altre unità e realizzare comportamenti di gruppo. Il basso costo dell'hardware necessario alla realizzazione dei robot influisce in modo molto positivo sulla riproducibilità e li qualifica come un'ottima piattaforma per l'immissione sul mercato di agenti basati su di essi e specializzati in un dato campo applicativo. La totale modularità con cui gli Octagon sono stati progettati li rende totalmente aperti a varie ed eventuali modifiche ed inoltre anche se nella loro versione di

base dispongono solo di sei sensori ad infrarosso, sono in grado di utilizzare fino a 18 sensori ad ultrasuoni , una vasta gamma di sensori ad infrarosso,sensori pirometrici per il rilevamento di fonti di calore e periferiche per il controllo di servocomandi. Gli Octagon sono infatti in grado di costituire una piattaforma mobile per bracci robotici, ed una volta equipaggiati di un sensore di visione costituiranno un ottimo strumento di ricerca , per studi in campo robotico di elevata complessità come ad esempio la Swarm Robotics. La robotica degli sciami (Swarm Robotics) è una recente disciplina che trae le sue origini dall'osservazione dei comportamenti di alcuni insetti sociali , organizzati intorno ad un individuo, la regina, che manifestano quella che spesso viene chiamata "intelligenza della colonia" che dà luogo ad un comportamento collettivo auto-organizzante . L'aspetto più interessante di questi sistemi sta nell'emergere di un comportamento globale grazie allo scambio di messaggi tra i singoli individui e senza alcuna azione di supervisione . Gli insetti sociali hanno ispirato anche gli ingegneri della Icosystems, un'azienda di software del Massachusetts che sta sviluppando per conto del Dipartimento della Difesa americano una strategia di controllo per sciami di robot, che nel caso specifico sono aerei a motore senza equipaggio, da impiegare in missioni di recupero e ricognizione, in ambienti rischiosi per l'essere umano. L'idea che sta alla base della robotica degli sciami si basa sull'emulazione e l'adattamento dei complessi comportamenti collettivi che gli insetti manifestano in assenza di controllo centralizzato e senza informazioni globali sull'ambiente. Gli algoritmi di controllo centralizzato a cui si è ricorso fino ad ora per gestire un singolo robot, non sono più validi per gli sciami di robot in quanto ci vorrebbe un software troppo pesante su ogni singola macchina ed il tempo di reazione crescerebbe a dismisura. In questo contesto, alcuni insetti sociali come le formiche e le api forniscono ai progettisti preziose indicazioni sulle strategie per generare un comportamento collettivo complesso. Nel caso degli insetti sociali precedentemente menzionati, uno dei fini ultimi della colonia è trovare la strada più breve per raggiungere il cibo. Swarm-bots è inoltre il nome del progetto realizzato con la collaborazione di diverse istituzioni: il

Politecnico federale di Losanna (EPFL), l'Università libera di Bruxelles , l'Istituto dalle Molle di studi sull'intelligenza artificiale (IDSIA) di Manno e la Commissione europea che ha stanziato la metà del costo del progetto. «L'idea di realizzare piccoli robot "a sciame" viene dallo studio del comportamento di insetti organizzati in società come le formiche: per questo, ci siamo valsi della collaborazione di uno studioso del comportamento degli insetti sociali, il professor Jean Louis Deneburg dell'Università di Bruxelles» dicono Giovanni Pettinaro e Ivo Kwee, ricercatori dell'IDSIA che si occupano della simulazione al computer degli Swarm-bots (o S-bots). «Ciascuno di quegli insetti - spiegano - ha un compito ben preciso da svolgere: ogni individuo è piccolo, ha limitate capacità, ma il lavoro coordinato di uno sciame di formiche permette la costruzione e il funzionamento di una struttura complessa come un formicaio. Ogni S-bot pesa circa un chilo e ha un diametro di dodici centimetri: si muovono da soli, ma possono cooperare quando le condizioni lo richiedono, esattamente come gli insetti sociali. Gli S-bots possono essere equipaggiati con diversi strumenti: sensori per rilevare la presenza di gas o di umidità e termometri; raggi infrarossi per riconoscere le caratteristiche del terreno su cui si muovono; sistemi di comunicazione per tenersi in contatto e videocamere per inviare immagini. Le piccole dimensioni permettono agli S-bots di muoversi in ambienti angusti. I cingoli sono adatti a zone accidentate e se un ostacolo è troppo arduo per un solo robot, l'unione fa la forza: agganciati tra loro, un gradino o una buca si possono superare agevolmente.» Anche i nostri Octagon sono nati per supportare la ricerca nel campo della Swarm Robotic ed anche se per ora non dispongono di cingoli hanno un'eccellente precisione di movimento ed offrono la possibilità di effettuare con la massima semplicità manutenzione adattiva al fine di adeguarsi in brevissimo tempo all'impiego in campi come :controllo di zone pericolose per l'uomo, perché contaminate da inquinanti o radiazioni, ispezionare tubature, essere trasportati su un pianeta del sistema solare per esplorarlo ,bonificare un campo minato o dipingere grandissime superfici orizzontali.

Capitolo 2

Descrizione Hardware

2 Descrizione Hardware

2.1 Generalità

Al fine di realizzare i robot Octagon nel minor tempo possibile ,con i fondi a disposizione e con le desiderate caratteristiche di performance, si è scelto di seguire il modello del ciclo di vita Rapid Prototyping Model . Tale scelta è stata effettuata anche in funzione del fatto che modelli più avanzati o meglio adatti a team di sviluppo e non ad un singolo individuo avrebbero introdotto un overhead eccessivo per una sola persona. Con gli Octagon si è cercato di creare un robot che potesse asservire con la massima versatilità alle attività didattiche e che proprio in funzione di questo fosse svincolato dalle componenti di carico rappresentate dal software utente.

*Figura 2.1: Robot **Octagon***

Questa caratteristica è essenziale per non predestinare gli Octagon all'obsolescenza e garantire brevissimi tempi di apprendimento finalizzati all'uso dello strumento. Tutto ciò è stato possibile grazie alla possibilità di controllare da remoto i robot e di non avere quindi l'obbligo di adeguare il proprio software alla potenza di calcolo degli stessi. Per arrivare al risultato finale si è passati attraverso 3 prototipi, che hanno permesso di adattare l'hardware degli Octagon al software di gestione fino al raggiungimento dei livelli di performance desiderati.

2.2 Sensori

Gli Octagon utilizzano i sensori ad infrarosso GP2D12, che consentono la misurazione della distanza dagli ostacoli che si trovano all'interno del loro volume di scansione. Questi sensori sono stati impiegati al posto dei sensori ad ultrasuoni perchè nonostante offrano una grande precisione a scapito della portata massima di rilevazione, rispetto ai secondi consentono tuttavia a più robot di operare simultaneamente nello stesso ambiente senza interferire tra di loro. I sensori GP2D12 hanno un range di scansione che varia tra 10 e 80 cm e nel caso in cui un oggetto, le cui superfici siano in grado di riflettere la luce generata dall'emettitore ad infrarosso posto sul sensore si trovi all'interno di tale range, si ha una variazione del segnale analogico (generato dal sensore) che è funzione non lineare della distanza del GP2D12 dall'ostacolo. La scelta dei sensori è stata effettuata anche in funzione del fatto che il microcontrollore impiegato nel presente progetto e preposto all'interpretazione dei segnali generati dai GP2D12 è dotato di otto ADCs, che gli permettono di interfacciarsi ad un numero di sensori più che sufficienti alla scansione del perimetro del robot. Nonostante siano stati impiegati solo sei sensori ad infrarosso, gli Octagon sono equipaggiati di un bus I^2C che consentirà tramite una una semplice operazione di plug-in il collegamento al robot di ben sedici sensori ad ultrasuoni. I sensori ad ultrasuoni permetteranno scansioni di range compreso tra 30mm e 600mm (nell'ipotesi che i robot non vengano messi in funzione contemporaneamente nello stesso ambiente).

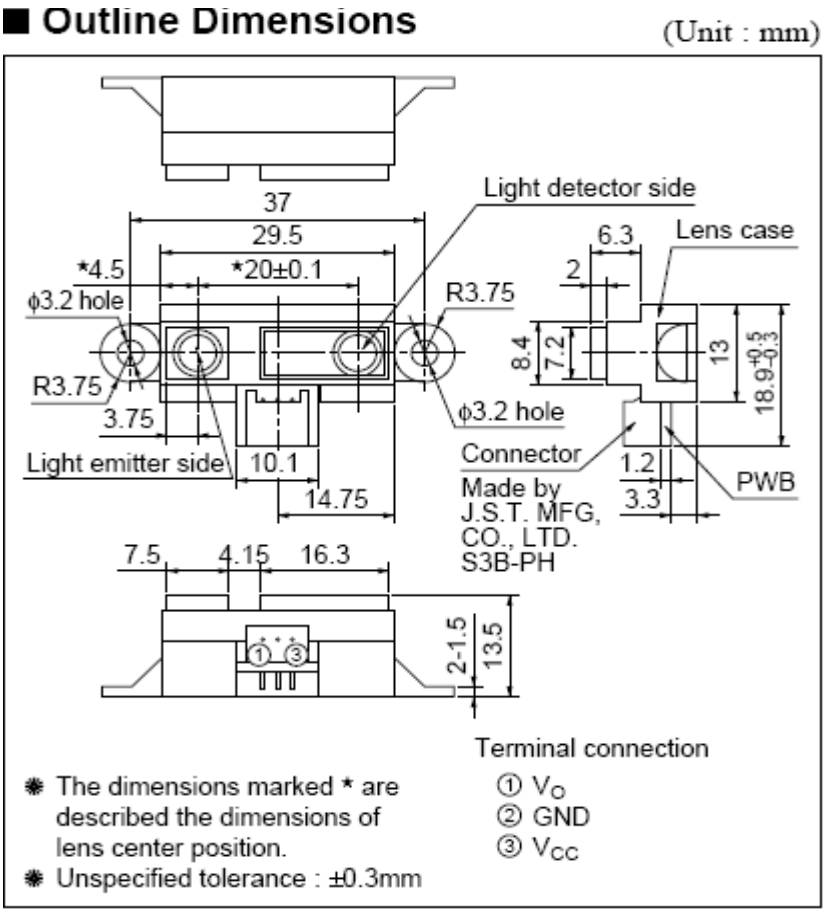


Figura 2.2: Sensore GP2D12

-tensione di alimentazione: [4.5V-5.5V]
-assorbimento : max 50 mA

2.2.1 Descrizione dei sensori IR GP2D12

I sensori GP2D12 effettuano le loro misurazioni tramite l'emissione di un raggio ad infrarosso pulsato con una lunghezza d'onda di 850 nm (+-8%). Se un oggetto è all'interno del volume di lettura del sensore la luce infrarossa che l'ostacolo riflette genera un'immagine sull'array CCD presente nel ricevitore ad infrarosso posto nel sensore. Il voltaggio del segnale analogico emesso dal GP2D12 è una funzione non lineare della distanza che intercorre tra il sensore e l'ostacolo (Figura 2.3).

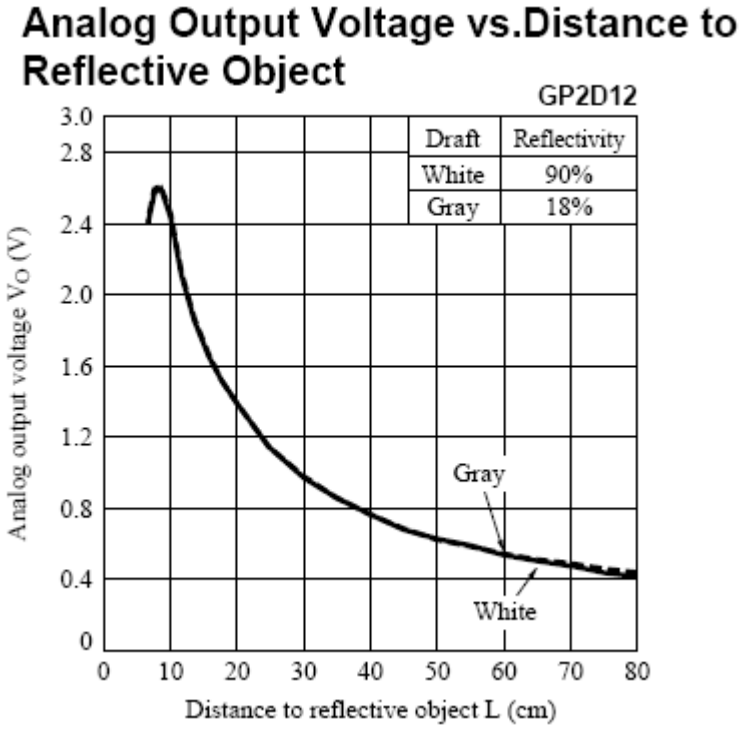


Figura 2.3: Segnale generato dai sensori GP2D12

I sensori GP2D12 hanno il pregio di essere poco sensibili alla luce ambientale, ma anche il difetto di non riuscire a rilevare oggetti a distanze inferiori a 10 cm. Questa caratteristica comporta non pochi problemi nella definizione del collocamento dei GP2D12 sul telaio del robot e in funzione di questo i sensori dovranno essere collocati il più possibile all'interno degli Octagon, dato che segnali analogici analoghi a quelli che vengono generati per oggetti non rilevabili (perchè troppo lontani) , vengono generati anche per oggetti prossimi ad urtare il robot. La funzione distanza di cui abbiamo bisogno è l'inversa di quella mostrata in Figura 2.3, che sarà invertita e approssimata via software tramite cinque segmenti ottenuti dall'interpolazione lineare di sei punti caratteristici della curva (Figura 2.4).

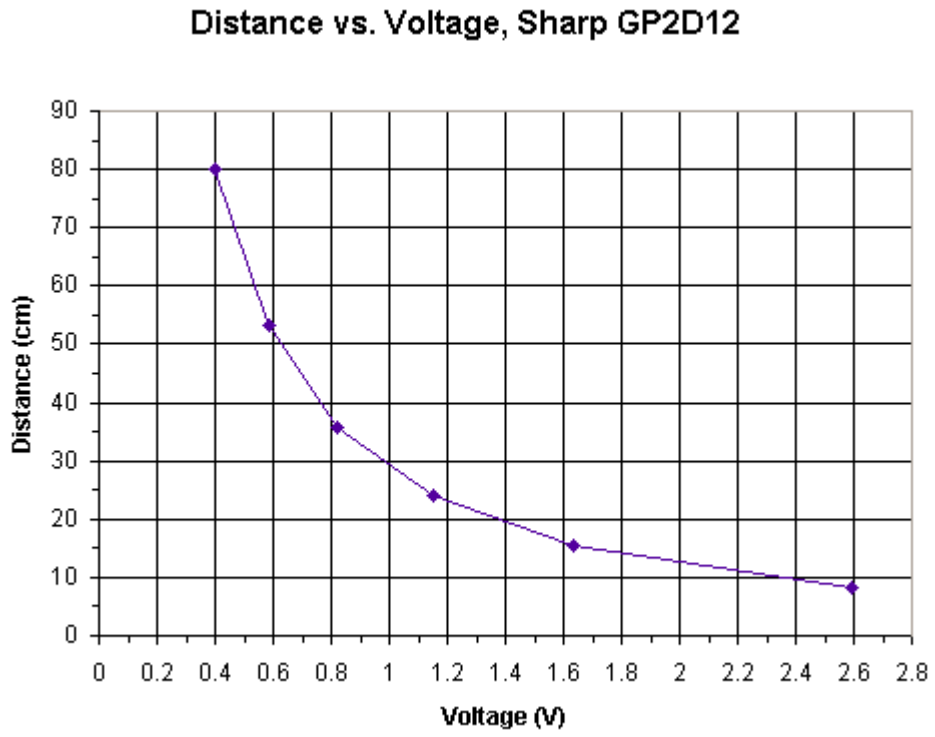
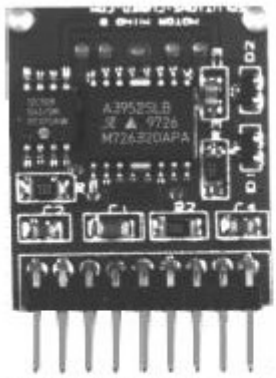


Figura 2.4

L' approssimazione della curva è stata effettuata in funzione dei punti di massima variazione del coefficiente angolare della stessa.

2.3 Controller motori

Il Motor Mind B enhanced (***M.M.B.e.***) è il controller per motori DC a 12V che verrà impiegato negli Octagon. La comunicazione tra l'MCU principale e il *M.M.B.e.* è implementata attraverso una interfaccia seriale TTL.



(Shown without heat sink)

SOLUTIONS CUBED

Solutions³

Motor Mind B

DC Motor Control Module
*Adjust Speed/Direction
*Easy Serial Interface
*Tachometer/Counter Input

Figura 2.5: Controller impiegato per la gestione dei motori

IL *M.M.B.e.* supporta correnti di alimentazione elevate, è protetto contro i picchi di corrente, di tensione ed il surriscaldamento. Le caratteristiche più importanti di questo controller si possono riassumere come segue:

- Fino a 1.75 A di corrente continua (6A di picco)
- Supporto per motori 6-36 vdc
- Frequenza PWM selezionabile tra 242 e 15.5 KHZ
- Interfaccia seriale TTL con velocità fino a 9.6 kbps
- Led per la comunicazione visuale delle condizioni di errore e dello stato della trasmissione dati.
- Filtro PI per implementare il controllo a ciclo chiuso della velocità

Il *M.M.B.e.* consente attraverso le sue funzioni tachimetriche, di autoregolazione della velocità e di conteggio degli impulsi generati dagli encoder dei motori, di implementare un rudimentale controllo di posizione del robot. In reltà nel presente lavoro i controller *M.M.B.e.*

verranno impiegati solo per la correzione della velocità di rotazione

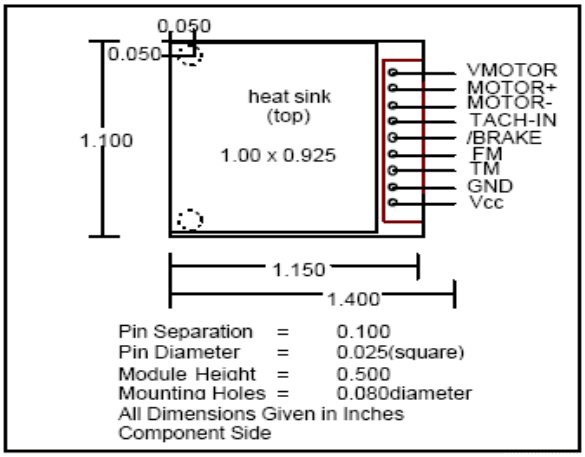


Figura 2.6: Motor MindB enhanced

delle ruote del robot, mentre il compito di gestire gli impulsi generati dagli encoder dei motori al fine di determinare la posizione del robot è stato delegato alla master MCU.

2.3.1 Led di segnalazione

D2 (Verde), Lampeggia brevemente appena il controller viene alimentato e durante la ricezione dei dati dalla master MCU.

D3 (Rosso), Lampeggia nei seguenti casi:

- la temperatura sul PCB supera i 175°
- la corrente assorbita supera i 6 A
- la tensione di alimentazione dei motori scende sotto i 5.6 V

Il led rosso viene disattivato non appena la condizione di errore scompare, anche se il bit di FAULT nello STAUS Register rimane attivo fino al reset del controller.

2.3.2 Filtro PI

La modalità di gestione della velocità può essere usata con uno schema di controllo a incremento/decremento o con filtro PI.
Nella modalità standard di incremento/decremento il controller genererà la frequenza desiderata con il comando SPDCON e la velocità

di rotazione del motore è gestita nel modo seguente:

- Viene incrementata se il tachimetro interno al controller segnala una velocità inferiore a quella richiesta.
- Viene decrementata se il tachimetro segnala una velocità superiore a quella desiderata.

Nel momento in cui viene selezionata la modalità PI, il segnale di errore:

$$\Delta fe = frequenza\_desiderata - frequenza\_attuale$$

viene moltiplicato per un termine proporzionale. Successivamente la somma degli errori calcolati su una data finestra temporale è moltiplicata per un termine Integrale ed infine la somma tra quest'ultimo valore ed il precedente è registrata in un registro a 32 bit, di cui verrano impiegati gli 8 bit più significativi. Gli 8 bit cosi ottenuti verranno utilizzati per la generazione del nuovo duty cycle del segnale PWM di controllo dei motori. Se la modalità PI è attiva è inoltre raccomandata una frequenza del PWM non inferiore a 15 KHZ.
Se settato correttamente il filtro PI è in grado di ridurre significativamente gli effetti negativi che un eccessivo carico del robot apporta alla dinamica di accelerazione dello stesso. In questo progetto si è scelto tuttavia di delegare alla master MCU il compito di conteggiare gli impulsi generati dagli encoder proprio al fine di ridurre per quanto possibile gli errori nel calcolo dell'entità degli spostamenti effettuati dal robot a seguito di variazioni della velocità di rotazione dei motori.
I parametri del filtro PI si possono modificare tramite il comando WRITE_PI, inoltre il bit STATUS.PIMODE deve essere settato a 1 prima che il filtro PI venga usato nella modalità SPDCON.
Grazie al ***M.M.B.e.*** sarà possibile apportare correzioni alla velocità con frequenza di 8HZ.

Gate Time Value	Gate Time(time spent reading tachometer, T_{TACH})	Duty Cycle Updates per second(SPDCON)	Resolution (TACH)
'80'h	2000ms	1 every 2 seconds	1Hz
'40'h	1000ms	1 every second	2Hz
'20'h	500ms	2 every second	4Hz
'10'h	250ms	4 every second	8Hz
'08'h	125ms	8 every second	16Hz
'04'h	62.5ms	16 every second	32Hz
'02'h	31.25ms	32 every second	64Hz
'01'h	15.625ms	64 every second	128Hz

Table of gate time values for TACH and SPDCON firmware revision 298

Figura 2.7

2.3.3 Il termine proporzionale PTERM

Il PTERM viene moltiplicato per il segnale d'errore ***Δfe,*** ed avrà l'effetto maggioritario sul cambiamento della velocità dei motori, essendo molto maggiore del termine integrale. Nonostante il PTERM venga calcolato su registri a 32 bit, per ragioni d'efficienza verrà troncato dei suoi 24 bit meno significativi.

2.3.4 Il termine integrale ITERM

Fintanto che la modalità PI è operativa, i segnali di errore vengono sommati in un registro. Il termine integrale è poi moltiplicato per la somma degli errori ed il risultato è sommato al PTERM di cui sopra ed al prodotto dei segnali di errore. Il termine integrale a differenza del proporzionale, influenzerà solo in piccola parte le modifiche apportate al PWM.

2.3.5 Il PISCALER

Il risultato delle operazioni effettuate sul termine integrale sarà molto maggiore di 8 bit, pertanto si effettuerà il troncamento delle cifre meno significative tramite divisione per 2^n con n=PISCALER.

2.3.6 STATUS.PIMODE

Lo STATUS.PIMODE *è* il bit dello STATUS REGISTER che va settato per abilitare la modalità PI ed il set di comandi SPDCON che consentiranno di impostare i parametri per la correzione della velocità di rotazione dei motori.

2.4 Modulo Wireless

Figura 2.8: Modulo ER400TRS

Il modulo radio ER400TRS è un ricetrasmettitore che incorpora la tecnologia Easy Radio, la quale ci consentirà di traferire i dati in modalità half duplex con una portata di 250 metri in linea d'aria.
L'ER400TRS è un sottosistema completo a basso consumo costituito da un microcontroller, un regolatore di tensione ed un trasmettitore FM.

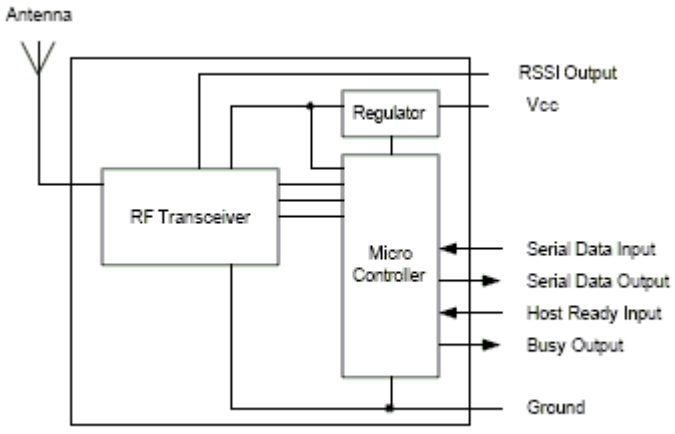


Figura 2.9: Diagramma a blocchi del modulo ER400TRS

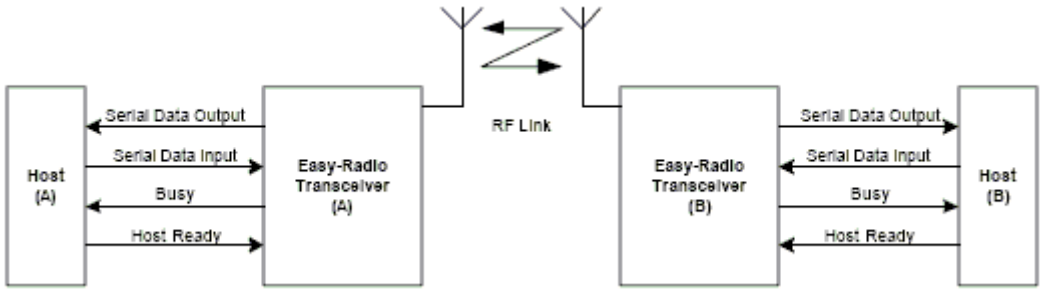


Figura 2.10: Schema di connessione con i moduli ER400TRS

Il microcontroller esplica la funzione di ricetrasmettitore RF e realizza l'interfaccia di trasferimento dati da e verso la sorgente tramite I/O seriale. Il presente modulo contiene inoltre una memoria EEPROM programmabile, per la registrazione dei dati relativi alle varie configurazioni delle modalità operative del sistema ricetrasmettitore. La MCU del modulo ER400TRS solleva inoltre il dispositivo host dal carico di lavoro derivante dall'ottimizzazione del segnale radio, rilevamento e correzione dei dati ricevuti in modo errato, trasmissione dei dati in un formato adatto al canale . Il Modulo EasyRadio consente inoltre di rilevare l'intensità del segnale ricevuto al fine di consentire ai robot di determinare il modulo della distanza dalla stazione di controllo, nella quale risiederà l'intelligenza artificiale dei robot e da altre unità Octagon. Il modulo wireless opera ad una velocità di 19200 baud ed è in grado di accodare 192 byte di dati nei suoi buffer interni sia in ingresso che in uscita, in attesa che vengano trasmessi o utilizzati dall'host. Purtroppo i moduli EasyRadio non consentono una comunicazione full duplex, in quanto operando a coppie sulla stessa frequenza sia in trasmissione che in ricezione sono costretti a condividere uno dei canali di livello fisico: l'etere. Per quanto detto in precedenza, non avendo la possibilità di trasmettere e ricevere su frequenze differenti il canale di comunicazione wireless sarà half-duplex. Una ovvia soluzione al problema sempre utilizzando i moduli ER400TRS, sarebbe impiegare due moduli su ciascun Host ed utilizzarli in modalità simplex, ma per contenere i costi e non ridurre l'autonomia del robot si è scelto di

equipaggiare gli Octagon con un solo modulo EasyRadio e di limitare la dimensione dei frame che transitano nel canale radio.

2.4.1 RSSI output

Il moduli ER400TRS sono in grado di determinare l'intensità del segnale ricevuto e forniscono un segnale analogico il cui valore di tensione è inversamente proporzionale all'energia RF presente nella banda passante del ricevitore. L'intensità di tale segnale analogico può variare tra 0 Volt (massimo segnale -50 dbm) , e 1,2 Volts (minimo segnale , -105 dbm) ed ha una pendenza di approssimativamente 50 db/volt.

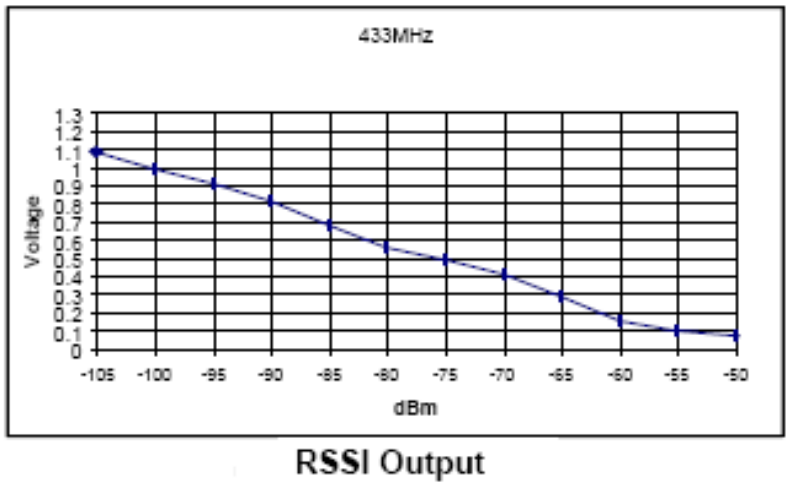


Figura 2.11: Segnale analogico RSSI

2.5 Unità di controllo

I microcontroller sono dei sistemi a microprocessore integrati su di un singolo chip equipaggiato di una memoria di programma non volatile e di sola lettura (solitamente una PROM,EPROM,EEPROM), una memoria RAM che spesso e purtroppo ai fini applicativi è di dimensioni molto ridotte (nel nostro caso è di 400 bytes) e periferici di I/O di vario tipo (convertitori DAC o ADC, porte seriali e/o parallele, contatori/timer, watchdog timer ecc). Da questa breve introduzione risulterà evidente

che i microcontrollori siano stati progettati soprattutto per applicazioni industriali di controllo, in cui una volta messo a punto il software di gestione dell'impianto non c'è più bisogno di modificarlo, oppure le modifiche sono molto rare. Questo tipo di applicazioni hanno trovato posto praticamente ovunque nella vita quotidiana, basti pensare che in ogni elettrodomestico , automobile o ciclomotore, molto probabilmente c'è almeno un microcontroller. Tra i numerosi tipi di micontrollori disponibili sul mercato , troviamo le famose MCU della famiglia PIC a 8 bit, prodotte dalla Microchip Technology. I PIC negli ultimi anni hanno guadagnato molta popolarità grazie alla loro relativa semplicità di programmazione, dotazione hardware, potenza di calcolo e soprattutto anche grazie ai loro costi estremamente contenuti.

2.5.1 PIC (Programmable Integrated Controller)

I microcontrollori della famiglia Pic si differenziano da un microprocessore classico essenzialmente perchè sono equipaggiati di CPU RISC (Reduced istruction set computing). La filosofia RISC consiste sostanzialmente nel prevedere un set di istruzioni di numero molto ridotto ed a basso livello. Tali istruzioni inoltre devono essere tutte della stessa lunghezza e richiedere possibilmente lo stesso tempo sia per il numero di cicli macchina necessari alla loro esecuzione che per il fetch delle stesse. Quest'ultima caratteristica unita al fatto che la filosofia RISC comporta anche una suddivisione fisica del canale attraverso cui fluiscono le istruzioni dal canale dati offre la possibilità di sovrapporre le fasi di fetch (pipelining) delle istruzioni. Non essendoci "tempi morti" (cicli in cui si deve attendere la terminazione dell'istruzione attuale per liberare le risorse necessarie all'esecuzione dell'istruzione successiva), si ha un miglioramento della velocità complessiva con cui viene eseguito il programma. Attualmente i PIC possono lavorare alla frequenza massima di 40MHZ e tenendo conto che riescono ad eseguire una istruzione tipicamente in un ciclo macchina (lungo quattro cicli di clock) sono caratterizzati da un instruction cycle di 100 ns. Nonostante i costi molto ridotti e le elevate prestazioni di cui tali microcontrollori sono capaci , non sono sempre la scelta migliore per la prototipazione , dato

che il loro ridotto set di istruzioni comporta tempi di sviluppo del software molto lunghi ed una pessima manutenibilità del codice. Per tale motivo nel presente progetto si è preferito restringerne l'impiego alla gestione dei controller dei motori , all'interfaccia radio ed ai sensori, mentre ad un microcontrollere BX24P prodotto dalla NetMedia sono stati affidati la gestione e il coordinamento delle attività espletate dalle MCU della famiglia PIC. Lo sviluppo del software di gestione del robot con un microcontrollore programmabile ad alto livello ha favorito lo sviluppo in tempi brevi del software di controllo degli Octagon, lasciando tuttavia la possibilità futura di sostituire il BX24P con un microcontrollore della famiglia PIC, abbattendo i costi di produzione d'almeno il 25% ed incrementando notevolmente le capacità di calcolo del robot.

2.5.2 Scelta del Master controller

Gli Octagon sono dei robot le cui ridotte dimensioni sono un requisito indispensabile per popolare di un gruppo consistente di tali macchine piccoli ambienti, quali ad esempio un laboratorio di robotica. La specifica progettuale relativa al volume occupato dai robot influenza in modo considerevole la scelta dell'hardware che li costituisce, tagliando inevitabilmente fuori tutti quegli elementi caratterizzati da un ingombro e/o assorbimento eccessivo. L'unità di processamento centrale che si occupa del controllo delle periferiche del robot è un microcontrollore della NetMedia. La scelta di questa classe di MCU non è dovuta all'assorbimento di corrente o all'ingombro che si discosta solo minimamente da quello della famiglia Pic ed altri modelli, ma bensì è la possibilità di programmarli con un linguaggio ad alto livello. Le MCU della famiglia PIC, nonostante l'ottimo rapporto qualità/costo del componente, richiedono un lungo periodo di apprendimento ai fini dell'acquisizione delle conoscenze necessarie alla loro programmazione. Questa loro caratteristica non li qualifica come la scelta migliore in tutti quegli ambienti di ricerca in cui è di fondamentale importanza la creazione rapida di prototipi. Il passaggio da un linguaggio di programmazione ad alto livelo a cui spesso è abituata la media degli utenti, ad uno molto

più vicino alla macchina come quello dei microcontroller PIC non è immediato e può richiedere tempi d'apprendimento abbastanza lunghi. Per le motivazione di cui sopra si è preferito impiegare il microcontrollore BX24P prodotto dalla Netmedia.

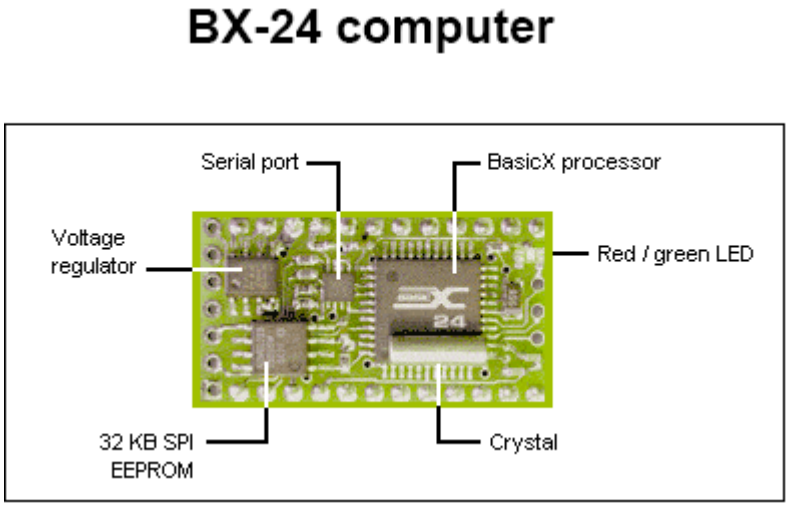


Figura 2.12: Microcontrollore BX24P

2.5.3 Il microcontroller BX24P

I microcontroller BX24P sono sistemi di controllo completi cablati su di un singolo chip, combinati con un IDE fornito gratuitamente dalla casa produttrice. Uno dei motivi principali per cui è stato impiegato questo tipo di microcontroller è la possibilità di poter programmare gli Octagon nel linguaggio ad alto livello BasicX. Una volta compilato il codice sorgente BasicX otteniamo un linguaggio binario intermedio che l'interprete BasicX residente nella ROM del core processor Atmel (su cui sono basati i BX24P) è in grado di eseguire. I file generati dalla compilazione del codice sorgente hanno estensione .BXP. Il compilatore produce inoltre un file con estensione .PRF, nel quale sono contenuti i parametri di configurazione iniziali del microcontroller BX24P. L'ambiente di sviluppo fornito con tali microcontrollori si occupa inoltre del download tramite interfaccia seriale RS232 del codice eseguibile dal microcontroller e di settare sulla base dei dati contenuti nel file .PRF i

parametri di startup. Il BX24P è equipaggiato a sua volta di un microcontroller ATMEL (AT90S8535), che ne costituisce il core processor. Quest'ultimo è caratterizzato da una ROM in cui risiede il sistema operativo e l'interprete BASICX, 400 byte di RAM, 32 Kbyte di EEPROM e molti devices come timers,Uarts,ADCS,DACS, digital I/O pin, bus periferici SPI e molto altro. Il sistema operativo BOS (Basic Operativ System) residente nella ROM del core processor, si occupa di offrire allo sviluppatore un ambiente multi-tasking ed inoltre contiene un motore di esecuzione BasicX. La possibilità di poter operare in un ambiente multi-tasking è una caratteristica molto importante che permette di ottimizzare l'utilizzo del microcontroller e ci consentirà di abilitare un task a rispondere agli eventi asincroni generati dall'hardware del BX24P. Se non avessimo avuto questa possibilità saremmo stati costretti a impegnare il microcontroller in cicli di attesa attiva per catturare gli eventi generati dagli encoder dei motori. Questo ci avrebbe costretto ad aumentare il numero delle MCU per una corretta allocazione delle componenti di carico, che a quel punto non sarebbero più state supportate da un solo microcontroller BX24P.

2.5.4 Descrizione hardware del BX24P

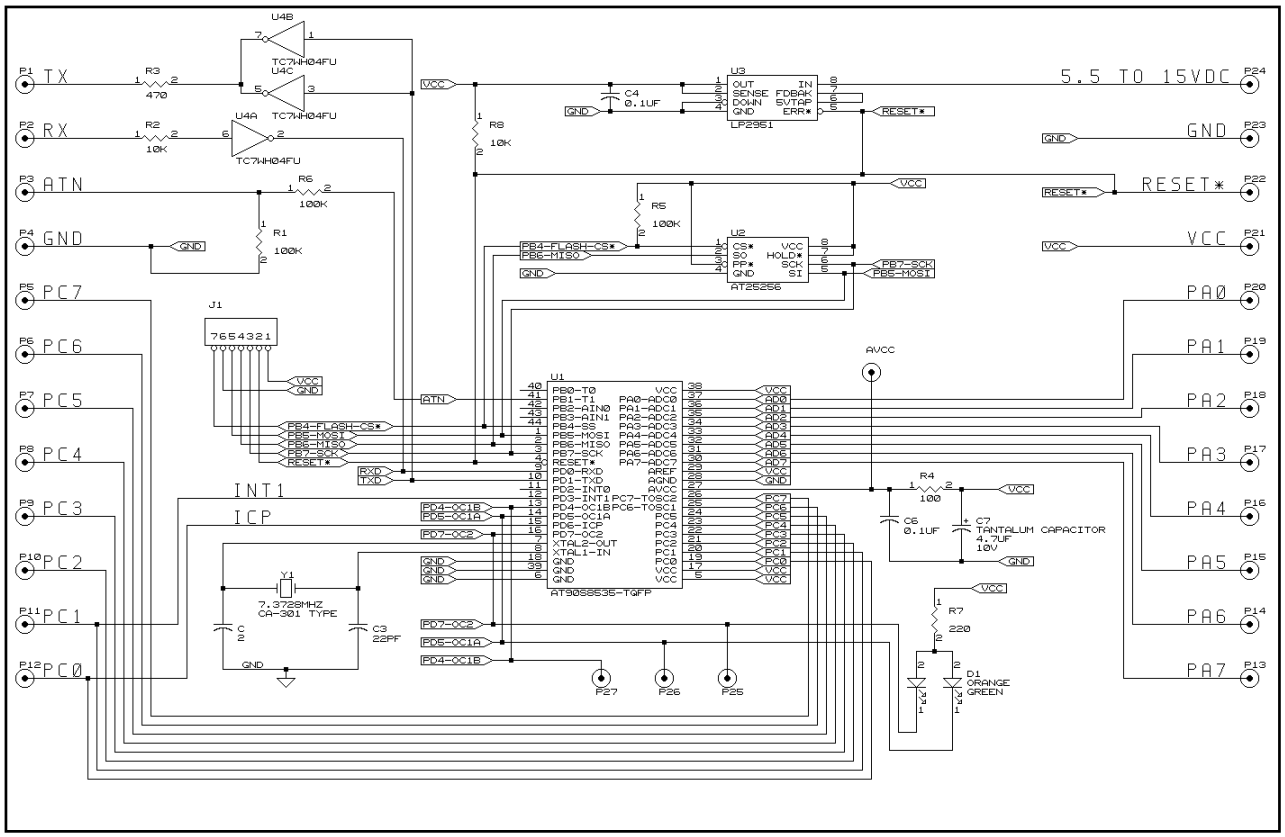


Figura 2.13: Schema elettrico del BX24P

- ***BasicX Processor***

Il *BasicX Processor* è un processore virtuale basato sul core processor AT9058535. Questa MCU si occupa dell'esecuzione delle istruzioni di programma utente registrate nell' EEPROM da 32 KB presente sul chip stesso. Il BX24P è inoltre dotato di ben 16 linee di I/O , compatibili TTL/CMOS e configurabili dall'utente per interfacciare l'hardware interno del core processor con l' esterno. Nel caso specifico tutte le 16 linee possono essere configurate come I/O digitali ed inoltre 8 di esse possono essere impiegate come convertitori analogici digitali.

- ***Porta Seriale***

Il BX24P è equipaggiato con una porta seriale che consente una velocità di comunicazione massima di 460800 baud. Tale porta seriale sarà impiegata nel presente lavoro per connettere il BX24P al modulo wireless che si occuperà del trasferimento dati da e verso il Robot e per consentire la programmazione del microcontroller stesso. Nonostante il modulo wireless ER400TRS sia TTL compatibile con la nostra MCU, se collegassimo il microcontroller AT9058535 direttamente alla seriale tramite i suoi pin I/O lo danneggeremmo irreversibilmente, dato che lo standard RS232 opera con tensioni di +-12V. Il microcontrollore è pertanto equipaggiato di un Hex Inverter (TC7WH04) che si occupa di invertire i segnali da e verso l'interfaccia RS232 attraverso cui verrà programmato il microcontroller e di interfacciare la logica TTL del BX24P con i valori di tensione dello standard RS232.

- ***32K SPI EEPROM***

L' SPI (Serial Peripheral Interface) EEPROM è il chip che ospiterà la memoria di programma utente. Tale memoria di programma è basata su un chip A25656, e può immagazzinare approssimativamente fino a 8000 linee di codice BasicX.

- ***Convertitori analogici digitali***

Il BX24P include 8 canali ADC a 10 bit che possono essere alternativamente configurati come I/O digitali. I canali ADC sono in grado di convertire valori di tensione compresi tra 0 e 5V, con una precisione di +-5 mV.

2.6 Motori Dc

I motori impiegati per la propulsione del robot sono dei comuni motori a corrente continua a 12V, dotati di motoriduttore a ingranaggi metallici

ed encoder integrato ad effetto Hall.

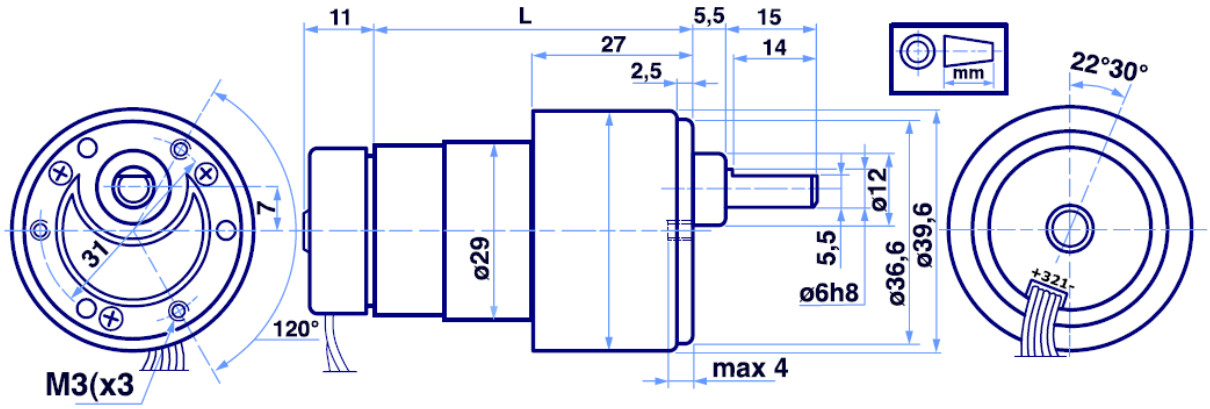


Figura 2.14: Motori DC

Per ogni giro del motore sono disponibili 3 impulsi, ed essendo il rapporto di riduzione pari a 198,5:1 si avranno 595 impulsi generati dall'encoder per ogni giro dell'asse esterno del motoriduttore.
Di seguito verranno riportate le caratteristiche più importanti dei motori impiegati:

Caratteristiche Tecniche:	
Tensione Nominale Vdc	12
Consumo a vuoto mA	140
Consumo Max mA	440
Coppia Max Kg/cm	10.2
Rapporto Riduzione	198.5:1
Velocita' a vuoto RPM	33
Velocita' a coppia Max RPM	23
Diametro Asse mm	6
Lunghezza mm	69
Peso gr	190

Gli encoder ad effetto Hall sono sensori molto stabili termicamente e resistenti alle sollecitazioni meccaniche e pertanto nonostante la bassa risoluzione che li caratterizza sono stati impiegati nel presente lavoro per l'implementazione del controllo di posizione del robot.

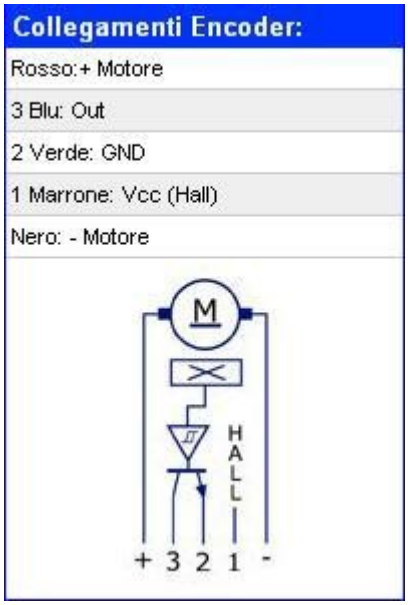


Figura 15: Schema di collegamento degli encoder

Il controllo della velocità e della potenza nei motori a corrente continua verrà effettuato dai controller **Motor Mind B enhanced**, che genereranno un' onda quadra modulata in ampiezza (PWM). L'onda quadra che verrà generata avrà frequenza costante e ne verrà variato il tempo della parte attiva (duty) all'interno del periodo fisso dell'impulso. Il pilotaggio PWM permetterà inoltre di modificare la velocità pur assicurando un rendimento energetico elevato e garantendo al robot una grande autonomia. Essendo i motori DC dei carichi induttivi, la corrente media è sostanzialmente costante e proporzionale al duty cycle del segnale di controllo degli stessi che permetterà quindi di variare la velocità di rotazione dei motori.

2.7 Motorizzazione della torretta

L'apparato sensorio degli Octagon verrà interamente cablato su di un supporto in allumino progettato per permettere la totale riconfigurazione dei sensori. Inoltre quest'ultimo verrà reso mobile per mezzo di un servomotore che può compiere escursioni di 180° con una precisione di 1/4 di grado. Di seguito verranno riportate le caratteristiche

tecniche del servomotore impiegato e lo schema elettrico di collegamento dello stesso.

Caratteristiche Tecniche:	
Cuscinetti	2
Coppia Kg*cm	9.6Kg (6Vdc)
Velocita' sec/60°	0.20 (6Vdc)
Peso	55.2 gr
Tipo Ingranaggi	Alumite/MP
Dimensioni	41 x 20 x 38 mm

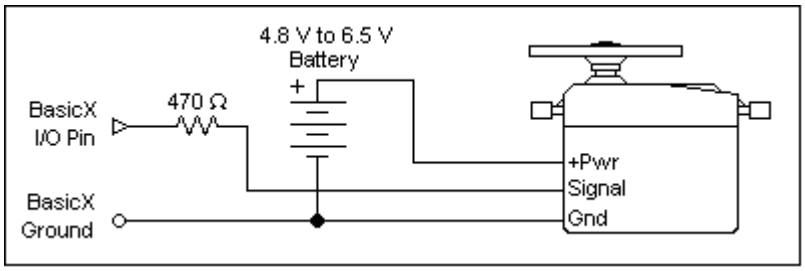


Figura 2.16: Schema di collegamento del Servomotore

La torretta verrà controllata direttamente dal BX24P che genererà un treno di impulsi d'opportuna ampiezza le cui caratteristiche di controllo del servo sono riportate nella figura 2.17. Per un controllo ottimale il segnale di controllo generato dovrebbe avere una frequenza di 50HZ , ma a causa della ridotta potenza di calcolo del microcontrollore impiegato si è scelto di limitarla a 33HZ (in modo da ridurre il tempo di CPU richiesto dal Task di gestione della torretta).

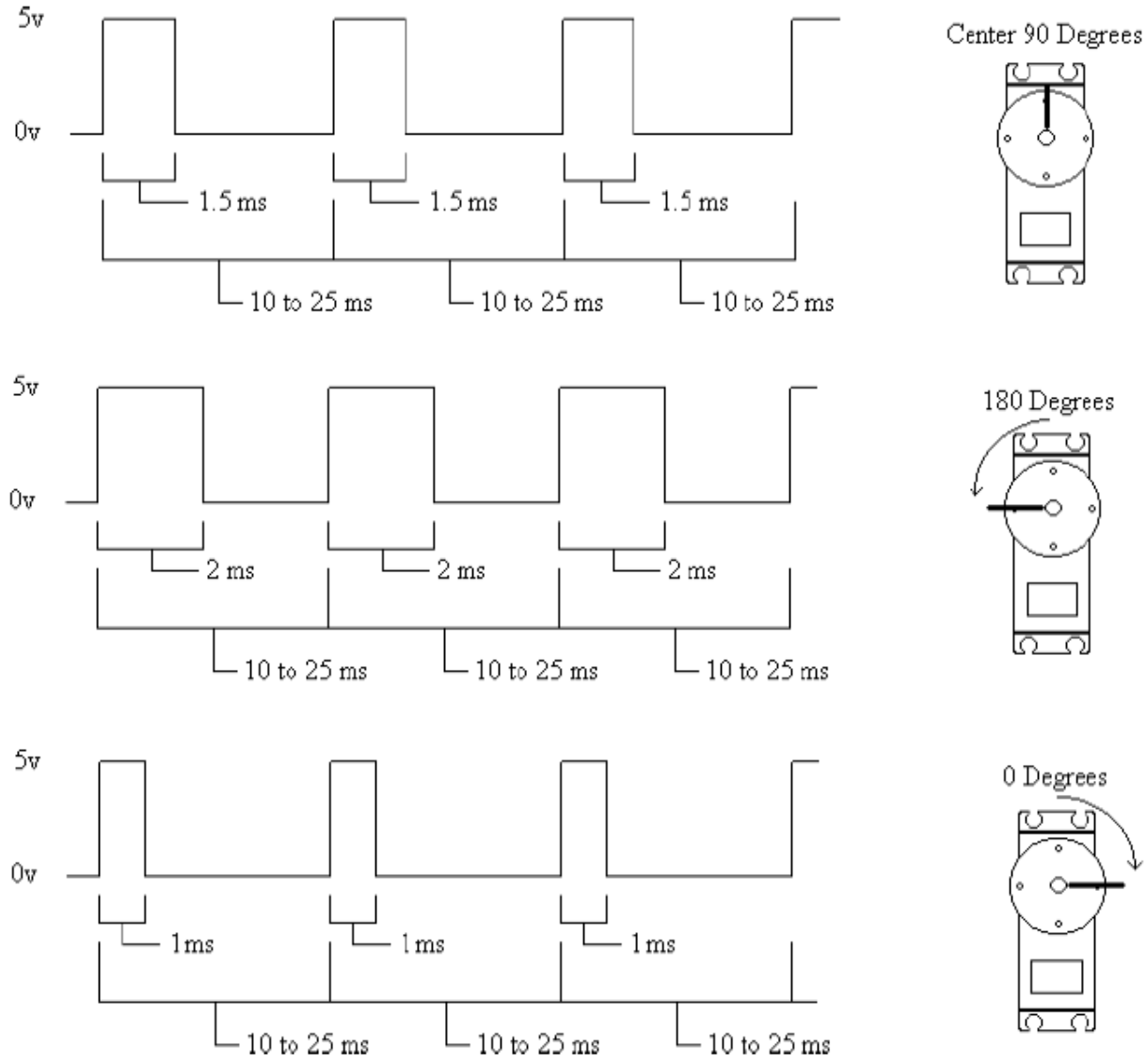


Figura 2.17: Segnale di controllo del Servomotore

Come è possibile notare dalla figura 2.3 il periodo dell'onda quadra del segnale di controllo può variare da 10 ms a 25 ms , anche se per carichi pesanti o sbilanciati, per evitare comportamenti instabili e dannosi per il servomotore è meglio non scendere sotto i 50 HZ. La posizione angolare dell'asse della torretta sarà invece determinata dall'ampiezza dell'onda quadra come riportato sempre in figura 2.17.

2.8 Schema elettrico

Per interconnettere le periferiche dei robot Octagon sono stati creati due PCB: la Sensor-Board e la Motor-Board. La Sensor-Board è il PCB che ospiterà sia i regolatori di tensione per l'alimentazione dei sensori che il modulo radio ER400TRS. La Sensor-Board inoltre ospiterà anche i connettori del bus I^2C e dei canali ADC. Il PCB che si occuperà di ospitare i controller per i motori M.M.B.e. ed il master controller è la Motor-Board. A causa dei fortissimi accoppiamenti induttivi causati dalla compresenza sulla Motor-Board sia della logica di controllo che degli stadi di potenza necessari a pilotare i motori, sono stati utilizzati dei cavi schermati per il trasferimento dei dati da e verso i controller M.M.B.e..

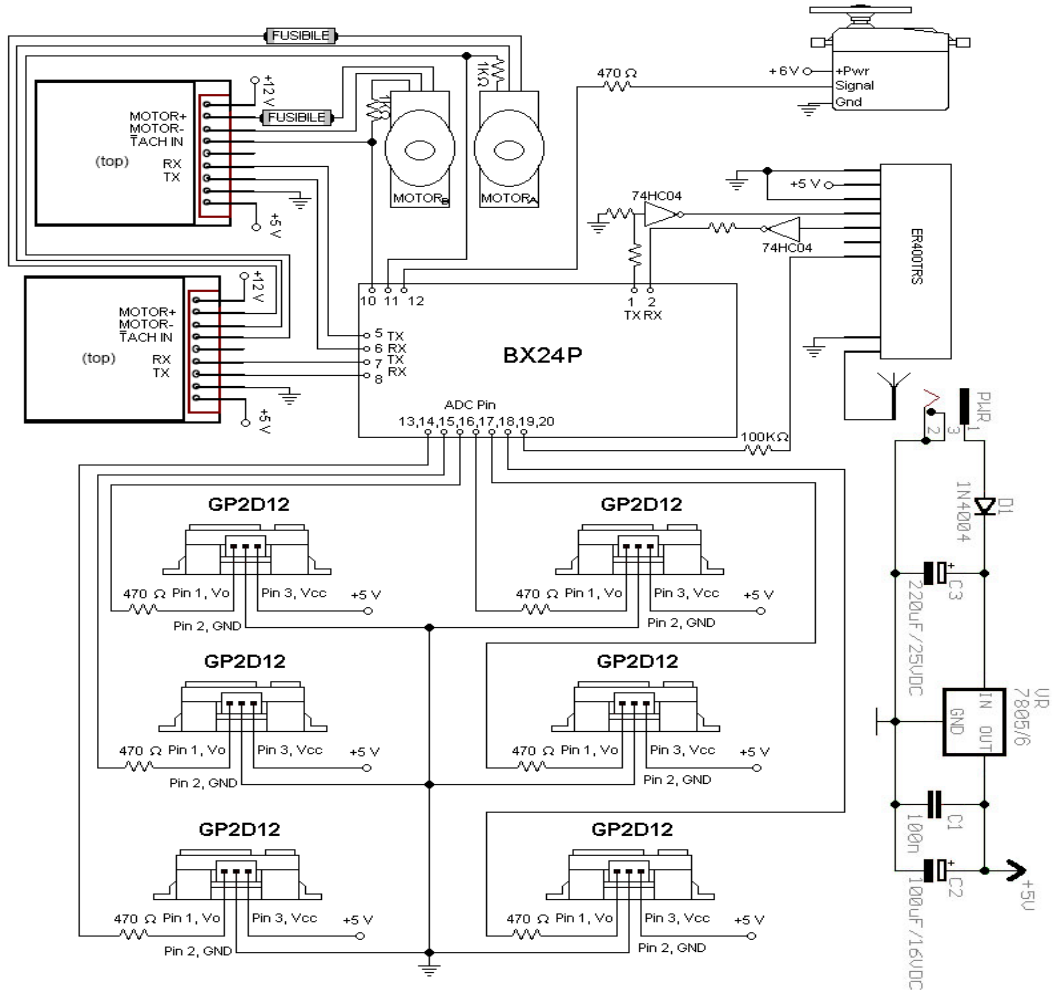


Figura 2.18: Schema elettrico dei robot Octagon

2.8.1 Sensor-Board

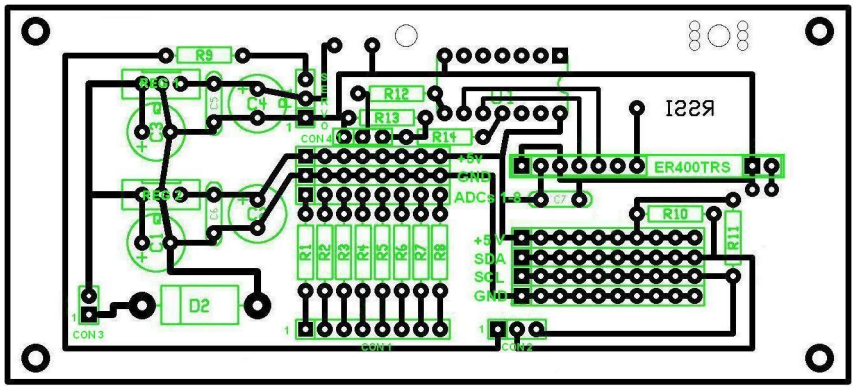


Figura 2.19: Sensor-Board lato componenti

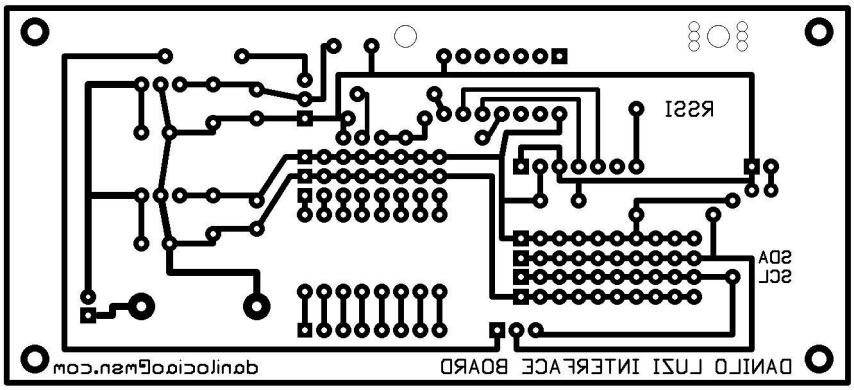


Figura 2.20: Sensor-Board lato rame

Elenco componenti:

- REG 1 Regolatore di tensione L7806
- REG 2 Regolatore di tensione L7805
- C1,C3 220 uF ,25 VDC
- C2,C4 100 uF,16VDC
- C5,C6 ,C7 100 nF
- ER400TRS
- U1 74HC04
- R1-R9 470 Ω di precisione
- R10,R11 4,7KΩ di precisione
- R12 330 Ω
- R13,R14 10 KΩ
- CON1 linee ADC 1-8

- CON2 1= Servomotore; 2=SDA; 3=SCL
- CON3 1=GND; 2=VCC (7-12)V
- CON4 1=GND; 2=TX; 3=RX

2.8.2 Motor-Board

Elenco componenti:

- 1X BX24P
- C1,C3 470 uF ,35 VDC
- 2X M.M.B.e.
- R1,R4 1K Ω di precisione
- R2,R3 10K Ω di precisione
- 2X fusibile 450 mA
- CON1=CON2 1=-V motori; 2=+V motori
- CON3=CON4 1=+5V encoder; 2=Gnd encoder; 3=TACH IN

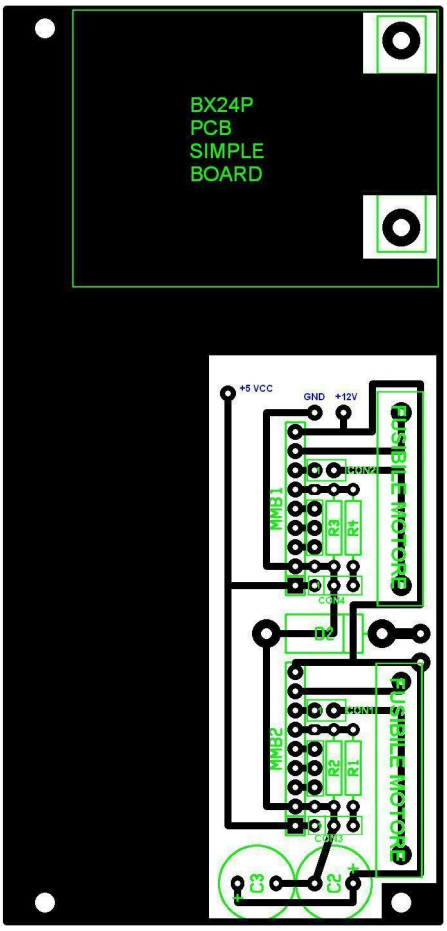


Figura 2.21: Motor-Board lato componenti

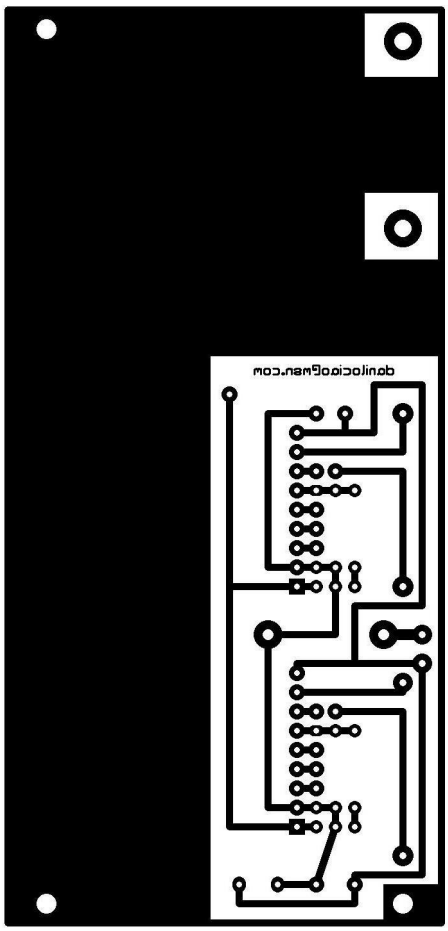


Figura 2.22: Motor-Board lato rame

2.9 Caratteristiche tecniche del Robot

Le caratteristiche più rilevanti degli Octagon possono essere riassunte come segue:

- Autonomia : 14 ore in moto continuo e 63 ore con sola elettronica di controllo attiva (sensori, master-controller, modulo radio)
- Peso :3,5 Kgr. di cui 2,8 costituiti dalla batteria
- Costo componenti : 344 euro
- Capacità di carico: 7,5 Kgr.
- Precisione nei movimenti traslatori: 0,25 mm
- Precisione nei movimenti rotatori : 0,15°
- Velocità media del master controller:(85000 istruzioni BasicX)/Secondo
- Potenza segnale modulo wireless: 10mW
- Canale wireless half-duplex da 19200 baud
- 8 canali ADC
- Bus I^2C
- Velocità massima : 9.9 cm/sec
- Escursione torretta :180°
- Velocità angolare torretta :60°/sec
- Coppia torretta: 9.6 Kg/cm
- Coppia motori: 10.2 Kg/cm
- Controllo closed-loop della velocità dei motori a cicli di 8HZ
- Gestione e controllo indipendente della velocità dei motori

2.10 Upgrade supportati

Grazie ai suoi 8 canali ADC ed al bus I^2C, che è fisicamente accessibile per la connessione di nuove periferiche dalla Sensor-Board gli Octagon sono in grado di supportare moltissimi upgrade di cui se ne elenca solo una parte :

- TPA81 ,8 Pixel Thermal Array Sensor
- CMPS03 Bussola elettronica per navigazione
- Giroscopio Piezoelettrico GWS PG-03
- ELTEC 442-3 Sensore Piroelettrico
- Sensore ad ultrasuoni ad alta frequenza SRF235
- Sensore di distanza ad Ultrasuoni SRF04
- Sensore di distanza ad Ultrasuoni SRF05
- Sensore di distanza ad Ultrasuoni SRF08
- Sensore di distanza ad Ultrasuoni SRF10
- 20A H-Bridge
- Scheda controllo per servi R/C SD21
- Lynx 5 Robotic Arm

Capitolo 3

Software di controllo del robot

3 Software di controllo del robot

3.1 Generalità

Lo scopo del software di controllo è quello di gestire le periferiche del robot e realizzare il protocollo di trasferimento dei dati da e verso l'impianto che ospiterà l'intelligenza artificiale che gestirà da remoto le attività degli Octagon. Il software sviluppato svolge le seguenti funzioni:

- Controllo della torretta
- Gestione dei sensori
- Gestione del bus I2C
- Gestione dei motori
- Gestione del canale radio

Come detto nel paragrafo 2.5.3 il sistema operativo BOS del microcontrollore BX24P permette di operare in un ambiente multitasking, mettendo a disposizione dello sviluppatore le primitive e le strutture dati tipiche della comunicazione intertask. Il software descritto in questo paragrafo è stato scritto con il linguaggio BasicX fornitoci gratuitamente dalla stessa azienda produttrice del BX24P insieme all' IDE per lo sviluppo del software e lo scaricamento dello stesso sul microcontroller alloggiato negli Octagon. Per consentire una migliore manutenibilità e leggibilità del codice ove possibile si è cercato di creare tutte le funzioni di interfaccia necessarie alla diminuzione del grado di accoppiamento dei moduli. Purtroppo, date le ridottissime risorse del microcontrollore, ciò non è stato sempre possibile e l'aspetto monolitico del codice è spesso dovuto alla necessità di minimizzare l'occupazione di memoria da parte degli stack dei task e di non intaccare la performance del sistema. A seguito dell'overhead introdotto dal contest switching implementato in modo non ottimale ,il multitasking è purtroppo

estremamente penalizzante sui sistemi operativi BOS. In funzione di questo si è cercato di ridurre il più possibile il numero stesso dei task che coordinano le attività degli Octagon anche se a scapito del grado di coesione ed accoppiamento dei moduli.

3.2 Gestione dei sensori

I sensori GP2D12 generano un segnale analogico compreso tra 0.399V e 2.60V che è funzione non lineare della distanza che intercorre tra lo stesso e l'eventuale ostacolo. Il segnale analogico verrà convertito dagli ADC di cui è equipaggiato il core processor AT9058535 e trasformato nel corrispettivo valore di distanza tramite inversione software della funzione non lineare presentata in figura 2.3. Il sistema operativo BasicX mette a disposizione dello sviluppatore la chiamata di sistema GetADC() che permette di convertire in digitale il valore di tensione applicato ai pin del BX24P che hanno funzione ADC. La funzione GetADC() restituisce un valore intero senza segno a 16 bit con range da 0 a 1023 e con risoluzione di 4,98 mV/unità. La funzione che si occuperà della inversione della funzione in figura 2.4 è :

```
Private Function voltageToRange( _
ByVal voltage As Single) As Single
```

mentre la procedura preposta al controllo del segnale analogico generato dai sensori GP2D12 è la seguente :

```
Private Sub GetRange( _
ByRef distance As Byte, _
ByRef success As Boolean, _
ByVal inputPin As Byte)
```

Tale controllo è essenziale ai fini della corretta interpretazione del segnale emesso dai sensori, in quanto essi genereranno segnali analoghi sia per oggetti troppo lontani, per i quali non si ha rifrazione di sufficiente intensità all'eccitazione del ricettore posto nel sensore, che per oggetti troppo vicini per i quali invece l'angolo di rifrazione della luce generata non è adeguata.

3.3 Controllo dei motori

Gli Octagon sono equipaggiati di due controller **M.M.B.e**., già discussi nel paragrafo 2.3. La comunicazione tra la master MCU ed i controller dei motori avviene tramite due bus seriali gestiti dal seguente task:

Public Sub EngineTask()

che inoltre si occuperà di gestire gli interrupt generati dagli encoder dei motori (si tratta di un task event-driven) come verrà discusso di seguito. Al livello più basso l'interrupt handler gestisce gli eventi innescati dall'hardware, con il core processor che agisce da dispatcher. Al livello superiore il sistema operativo BOS svolge la funzione di dispatcher per i task preposti alla gestione degli eventi generati dall'hardware. La chiamata di sistema che sospende il task di gestione dei motori in attesa della generazione di interrupt da parte degli encoder è la seguente:

Call WaitForInterrupt(28)

In particolare l' *EngineTask()* si attiverà ogni volta che sui pin del BX24P a cui sono collegati gli encoder si ha una transizione High-Low ed in generale ogni volta che il task principale lo attivi a seguito della chiamata di sistema:

CallTask "EngineTask", engineStack

che attiverà l' *EngineTask* assegnandogli lo stack *engineStack.* Il task terminerà invece non appena avrà inviato con successo il comando per la disattivazione dei motori. Tale comando verrà generato ogni volta che i parametri relativi alla velocità dei motori hanno valore nullo, oppure

quando lo spostamento previsto in fase di attivazione del task è stato effettuato. Nonostante la struttura monolitica dell'*EngineTask()* abbia portato ad un pessimo stile di programmazione, si è rivelata indispensabile ai fini della reattività del sistema stesso. Infatti il tempo di CPU richiesto dal task in fase di conteggio degli impulsi è estremamente ridotto e fa uso delle sole istruzioni :

```
Do
  if ((pulseCounter>0) and (motorSpeed<>0)) then
   Call WaitForInterrupt(28)
   pulseCounter=pulseCounter-1
  Else
   .....
   .....
Loop
```

Il tempo necessario al task switching è anch'esso ottimizzato mediante l'uso di variabili globali al modulo che hanno permesso di ridurre considerevolmente anche lo stack allocato staticamente per il task stesso. Non essendo stato possibile avvalerci dell'hardware USART del master controller per ambedue i canali di comunicazioni con i M.M.B.e. si è stati costretti a gestire la comunicazione seriale via software. Questo purtroppo ha portato alla necessità di disciplinare l'accesso alla CPU del master controller nelle fasi di gestione dei canali seriali attraverso cui il BX24P si interfaccerà con le altre MCU negli Octagon. Indicheremo con ASI1,ASI2,ASI3 (Asynchronous Serial Interface) i tre canali seriali degli Octagon. La gestione di ASI1 ed ASI2 è affidata all'*EngineTask()* che li attiva sequenzialmente, mentre ASI3 viene gestito dal *MainTask()* che si occuperà del modulo wireless. Durante le fasi di comunicazione seriale la risorsa CPU del BX24P dovrà essere acceduta in mutua esclusione dai due task contendenti.

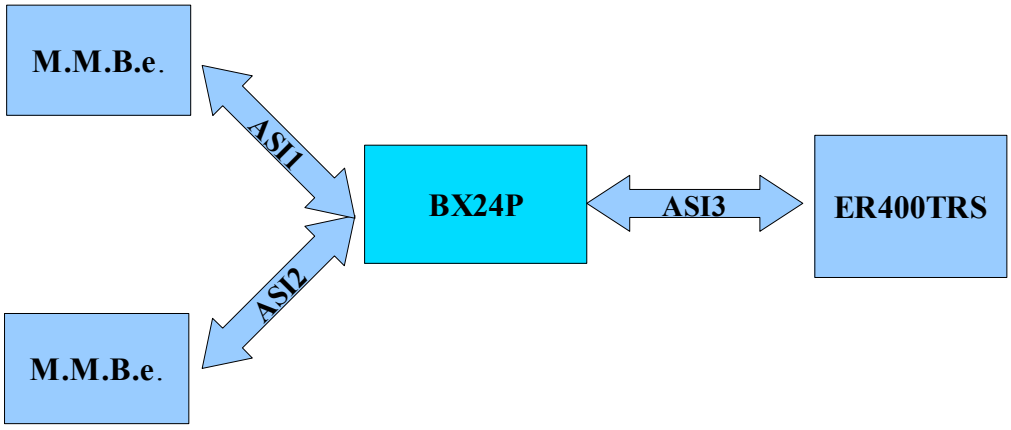


Figura 3.1

A tal fine è stato creato il flag "*spiIsNotFree"* utilizzato dalla chiamata di sistema:

Semaphore(spiIsNotFree)

che in un'operazione indivisibile permetterà di testare e settare (nel caso non sia già stato attivato) il flag *spiIsNotFree* con il quale sincronizzeremo i due task di cui sopra. Se un task trova la risorsa occupata si sospende in attesa che si liberi e che il BOS lo riattivi il prima possibile. Per impostare velocità,verso di rotazione ed ottenere un feedback dai controller dei motori si utilizzeranno i comandi: SETDC_DIR e STATUS. La dimensione di ogni elemento della sequenza di trasmissione riportata di seguito è di un byte ed impiega approssimativamente 4,97 ms ad essere trasmessa al M.M.B.e.. Per l'attivazione di ambedue i motori occorrono quindi circa 10 ms che è un tempo trascurabile rispetto alla velocità di movimento del robot e pertanto non introduce errori rilevanti nel calcolo della posizione.

SYNC	COMANDO **SETDC_DIR**	VELOCITA'/DIREZIONE	**SYNC**	COMANDO **STATUS**
0x55	0x08	0-0xff	0x55	5

A seguito della richiesta di conferma da parte del BX24P i controller **M.M.B.e.** genereranno la seguente sequenza di byte:

STATUS REGISTER								VELOCITA'
MOTDIR	FAULTON	BRAKEON	BAUD9600	FREQLOW	PIMODE	Non usato	Non usato	
1°bit	2°bit	3°bit	4°bit	5°bit	6°bit	7°bit	8°bit	0-0xff

L'unico byte che viene impostato nella sequenza di trasmissione è il terzo , in cui viene specificata la velocità di rotazione dei motori con valori compresi nell'intervallo [-127,+127] in complemento a 2 e codificati su 8 bit. Con i primi 6 bit della sequenza di ricezione si potranno verificare di volta in volta i parametri operativi dei M.M.B.e. , mentre con il secondo byte ci verrà comunicato il passo di velocità attuale con cui stanno operando i controller. Quest'ultimo non sarà rappresentato in complemento a due ed in generale se la fase di trasmissione è andata a buon fine sarà in rapporto 2:1 con il valore assoluto del 3° byte trasmesso. Dello STATUS register utilizzeremo in fase operativa solo il primo bit che ci fornirà il verso di rotazione del motore. I bit 3,4,5,6 verranno invece settati e testati solo all'atto della prima messa in opera dei controller M.M.B.e.. In questa fase infatti le impostazioni vengono registrate nella EEPROM di cui sono equipaggiati e non sarà più necessario modificarle.

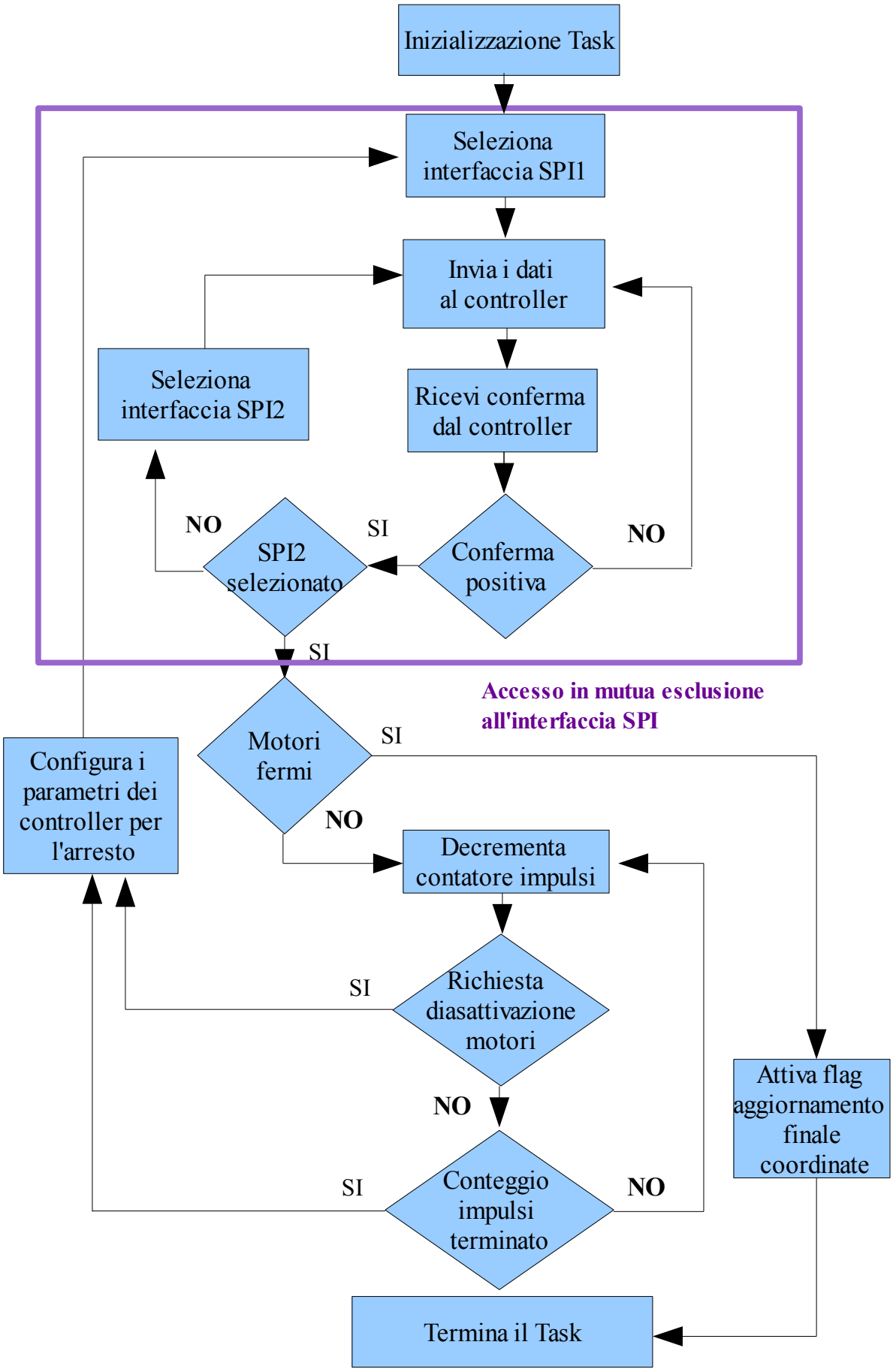


Figura 3.2: Schema di gestione dei motori

3.4 Controllo della torretta

La posizione angolare della torretta sulla quale sono posti i sensori degli Octagon verrà controllata dal micontroller BX24P che genererà un treno di impulsi a frequenza costante e modulati in ampiezza. Come discusso nel paragrafo 2.7 ,il servomotore che si occuperà della movimentazione della torretta può effettuare escursioni di 180° e la sua posizione angolare è funzione del tempo in cui il segnale di controllo rimane a livello alto. Per generare l'onda quadra è stata utilizzata la chiamata di sistema:

PulseOut()

che ci permetterà di ottenere impulsi dell'ampiezza desiderata con una risoluzione di 1.085µs. Il task che si occuperà di gestire la torretta è il ServoTask(), che si attiva con una frequenza di circa 33Hz per generare il segnale di cui sopra.

3.5 Gestione del Bus I^2C

I^2C è un protocollo di comunicazione seriale sviluppato dalla Philips per asservire alle sue necessità di interconnessione dei device su cui erano basati i prodotti che immetteva nel mercato. Uno dei principali vantaggi derivanti dall'impiego di questo protocollo risiede nel fatto che negli ultimi anni ha ottenuto una enorme diffusione e pertanto sono disponibili innumerevoli periferiche in grado di supportarlo. Il protocollo I^2C permetterà nel caso in cui se ne presenti la necessità, di connettere agli Octagon altri microcontrollori consentendo la riallocazione delle componenti di carico del software di controllo del robot nel caso in cui la potenza di calcolo del master controller risulti inadeguata al supporto delle future funzionalità dei robot Octagon. Il protocollo I^2C a livello fisico si avvale di due sole linee di comunicazione denominate rispettivamente SCA e SDL. La linea SDA è dedicata al flusso dati , mentre la linea SCL porta alle varie periferiche interconnesse al bus I^2C il segnale di clock per la sincronizzazione, che

viene generato dal master controller.

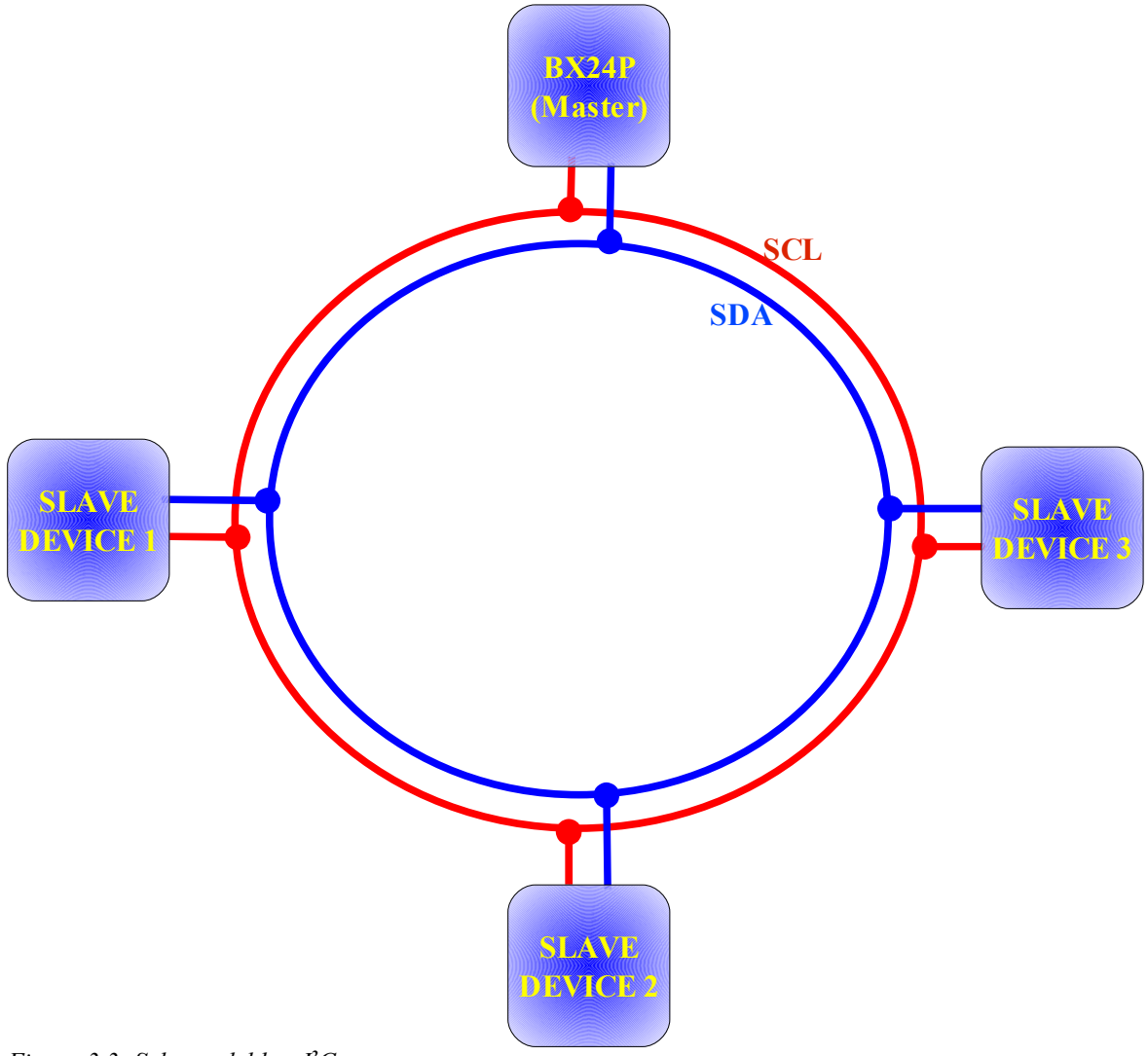


Figura 3.3: Schema del bus I²C

Nonostante il bus I²C sia multi-master, tratteremo di seguito solo il caso in cui l'unico master sia proprio il BX24P. Il master device deve per prima cosa generare la condizione d'inizio e successivamente indirizzare la periferica con cui desidera comunicare specificandone eventualmente l'operazione che quest'ultima deve espletare. Dopo aver inviato ed eventualmente ricevuto i dati dalla periferica il bus master termina il trasferimento segnalando la condizione di stop.

Figura 3.4

Tutti i dispositivi slave collegati al bus I^2C potranno quindi essere interrogati dal master controller degli Octagon prendendo in considerazione tutti e soli quei frame che dopo il bit di start riportano il loro indirizzo. Per la gestione del bus I^2C sono state create le seguenti primitive:

```
Sub I2cByteWrite(ByVal I2cAddr As Byte, _
                 ByVal I2cReg As Byte, _
                 ByVal I2cData As Byte)

Function I2CByteRead(ByVal I2cAddr As Byte, _
                     ByVal  I2cReg As Byte) As Byte

Function I2CWordRead(ByVal I2cAddr As Byte, _
                     ByVal I2cReg As Byte) As UnsignedInteger

Sub I2cOutByte(I2cData As Byte)
```

Function I2cInByte() As Byte

Sub I2cStart()

Sub I2cStop()

con le quali si potrà comunicare con i vari device presenti sul bus I^2C degli Octagon. Supponiamo ad esempio che si sia connesso al bus I^2C il sensore ad ultrasuoni SRF08. In questo caso gli indirizzi disponibili per questa famiglia di sensori sono : {0xe0, 0xe2, 0xe4, 0xe6, 0xe8, 0xea, 0xec, 0xee, 0xf0, 0xf2, 0xf4, 0xf6, 0xf8, 0xfa, 0xfc, 0xfe} e che il nostro sensore sia stato programmato all'indirizzo 0xe0. Dal Data Sheet dei sensori SRF08 otteniamo il numero di registro interno ai sensori che ci permetteranno di comandarne l'operato:

```
Const CmdReg As Byte = 0
Const LdrReg As Byte = 1
Const RangeReg As Byte = 2
Const RangeCmd As Byte = 81
```

che rappresentano rispettivamente: il registro di comando, il registro del sensore di luce ,il registro delle misurazioni distanziometriche ed il comando di misurazione in Cm . Per effettuare una misurazione con il sensore SRF08 sarà sufficiente impostare le linee SDA e SCL a livello alto ed utilizzare le seguenti primitive:

```
Call I2cByteWrite(&He0, CmdReg, RangeCmd)
Call Sleep(0.07)   ' attende per 70mS  che la misurazione sia conclusa
Range = I2CWordRead(&He0, RangeReg)
```

3.6 Gestione del modulo wireless

Per trasferire i dati da e verso la stazione di controllo il robot si avvale di un sistema di comunicazione digitale wireless. Tramite questo modulo gli Octagon sono inoltre in grado di captare le trasmissioni da e verso gli altri robot della loro stessa famiglia e di stimare la distanza a partire dall'intensità del segnale ricevuto. Tutto questo consentirà agli Octagon di comunicare in una LAN e di trasformarsi in ripetitori mobili per il recupero delle unità che per qualche motivo si spostano al di fuori della portata della stazione di controllo. Il protocollo di comunicazione degli Octagon si avvale del paradigma master-slave, in cui vi è una sola stazione master che disciplina l'assegnazione dei canali radio (10 canali half-duplex). Il task **Main()** coordina le attività del **ServoTask()** e **dell'EngineTask()** ed inoltre è preposto alla gestione del canale radio tramite un protocollo proprietario che offre un servizio connectionless non affidabile in fase di trasmissione ed affidabile in ricezione dalla stazione di controllo. I bit che transiteranno nel canale radio in trasmissione verranno raggruppati in frame composti da 15 bytes. La struttura del frame trasmesso alla stazione di controllo è riportata di seguito:

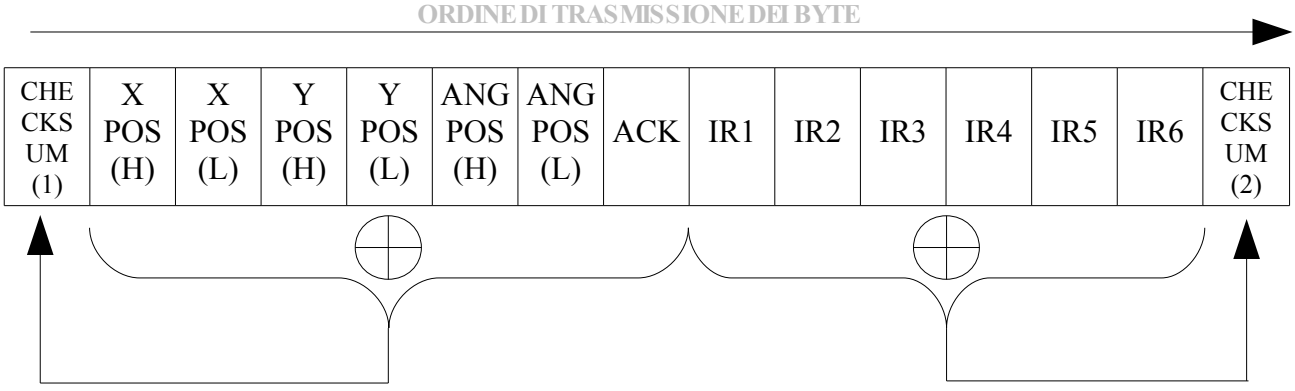


Figura 3.5: Sequenza trasmessa dall'Octagon

Il task Main() calcolerà in fase di trasmissione due byte di checksum per consentire alla stazione di controllo di verificare la validità dei dati ricevuti. Il frame è composto da 120 bit, di cui 16 sono di controllo (bit ridondanti) e 104 di dati. I frame inviati potranno contenere quindi 2^{104}

messaggi differenti, ma il numero totale delle parole di codice è strettamente minore di 2^{120} perchè alcune delle sequenze di 120 bit sono preposte alla segnalazione dell'errore. Gli errori verranno rilevati calcolando la parità sulle colonne della matrice le cui righe sono costituite dai byte del frame in figura 3.5. In sostanza verranno create di volta in volta due matrici costituite rispettivamente dai byte 2-8 e 9-14 dalle quali verranno calcolati colonna per colonna i bit di parità contenuti nei byte 1 e 15 del frame inviato. In questo modo un burst di al più 8 errori su ogni sotto sequenza di 7 e 6 byte è sempre rilevato perchè modifica al più un bit per ciascuna delle colonne di cui è costituita la matrice. Con questo sistema la probabilità che un burst di errori di numero maggiore di 8 non venga rilevato è di $1/(2^8)$ =0,00390625 su ciascuna sotto sequenze. La stazione di controllo attenderà la ricezione dei dati inviati dagli Octagon per inviare a sua volta il frame che conterrà i parametri di controllo del robot. I frame ricevuti dagli Octagon avranno una dimensione di 5 byte dei quali il quinto è il checksum che verrà calcolato con le stesse modalità di cui sopra. In questo caso la probabilità che un burst di errori di numero maggiore di 8 non venga rilevato dalla stazione di controllo è ancora di 0,00390625. Di seguito è riportata la struttura dei frame ricevuti dagli Octagon:

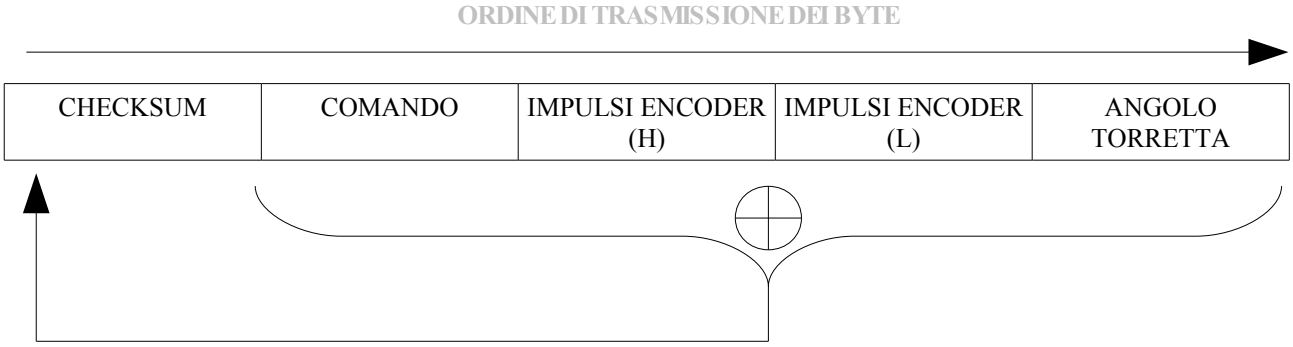


Figura 3.6: Sequenza ricevuta dall'Octagon

Nel caso in cui il checksum ricalcolato a partire dai byte ricevuti non coincida con il primo byte del frame in figura 3.6 verrà inviato un nack alla stazione di controllo che provvederà alla ritrasmissione, nel caso in cui invece i due checksum risultino uguali verrà inviato un ack e non ci

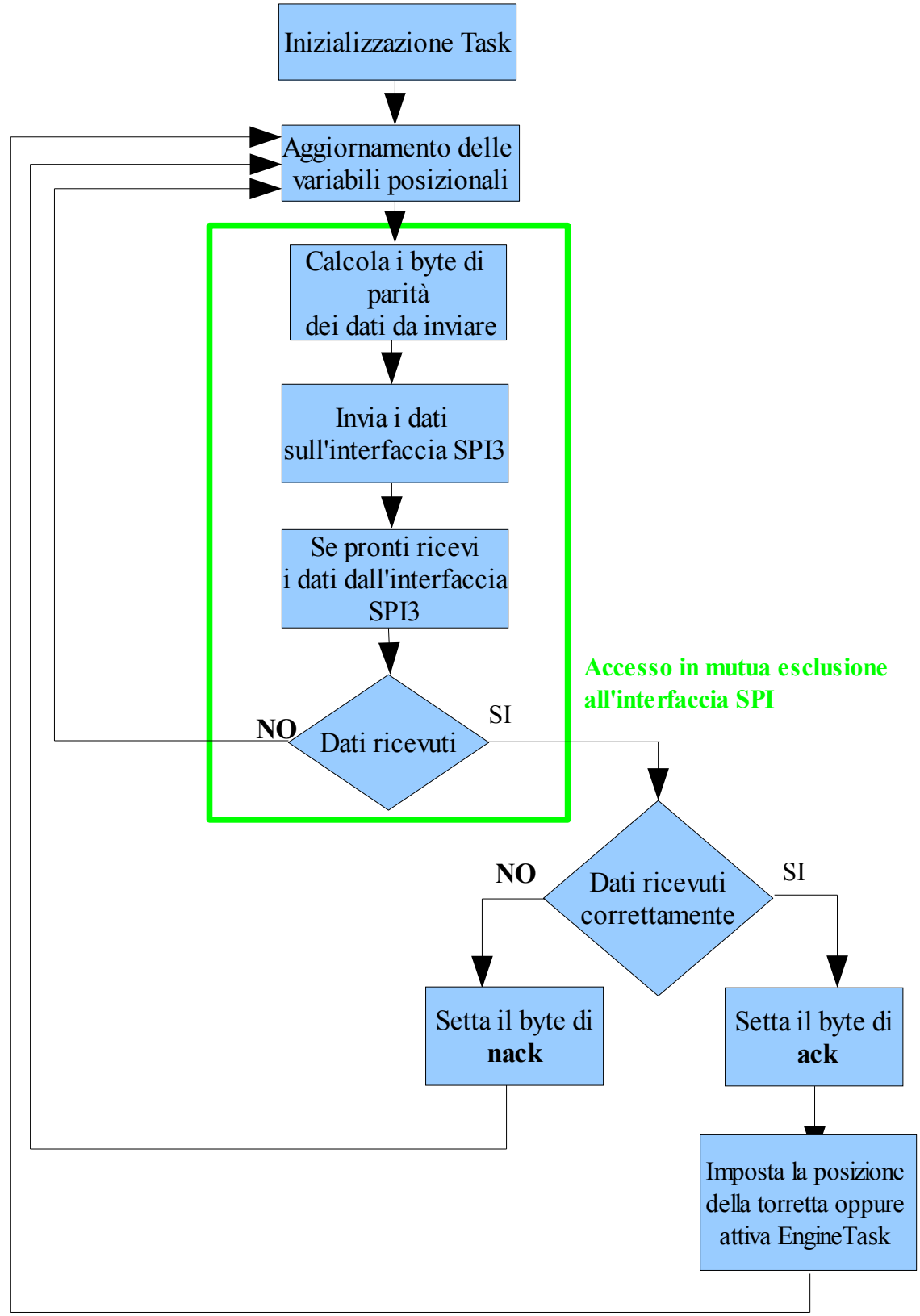


Figura 3.7: Protocollo di comunicazione degli Octagon

saranno ritrasmissioni. Il fatto che in trasmissione i dati scartati dalla

stazione di controllo non vengano recuperati con operazioni di ritrasmissione o correzione a partire dai bit ridondanti contenuti nei frame ricevuti è una conseguenza della estrema rumorosità del canale radio che purtroppo causa un numero di errori di trasmissione così elevato da rendere inefficiente qualsiasi operazione di recupero. Se venissero usate codifiche a correzione di errore a causa del ridotto baud rate del canale radio si ridurrebbero considerevolmente il numero di aggiornamenti dello stato del robot nella stazione di controllo e si è pertanto preferito scartare i dati ricevuti in modo errato in funzione del fatto che gli eventuali frame con errore al tempo T siano comunque da scartare rispetto a quelli pervenuti in istanti T' successivi a T. I byte trasmessi dagli Octagon hanno i seguenti significati:

CHECKSUM(1) = XPOSH ^ XPOSL ^ YPOSH ^ YPOSL ^ ANGPOSH ^ ANGPOSL ^ ACK

CHECKSUM(2) = IR1 ^ IR2 ^ IR3 ^ IR4 ^ IR5 ^ IR6

X_POS_H= Parte alta dei 16 bit della variabile posizionale *xPos*

X_POS_L= Parte bassa dei 16 bit della variabile posizionale *xPos*

Y_POS_H= Parte alta dei 16 bit della variabile posizionale *yPos*

Y_POS_L= Parte bassa dei 16 bit della variabile posizionale *yPos*

ANG_POS_H= Parte alta dei 16 bit della variabile posizionale *anglePos*

ANG_POS_L= Parte bassa dei 16 bit della variabile posizionale *anglePos*

mentre per quanto riguarda il significato dei byte facenti parte del frame ricevuto:

CHECKSUM = COMANDO^IMPULSI _ENCODER_H^IMPULSI_ENCODER_L^ANGOLO_TORRETTA

COMANDO = Codice del comando che i robot dovranno eseguire

IMPULSI _ENCODER_H= Parte alta dei 16 bit della variabile posizionale *pulseCounter*

IMPULSI _ENCODER_L= Parte bassa dei 16 bit della variabile posizionale *pulseCounter*

ANGOLO_TORRETTA= Variabile posizionale *servoAngle*

(Indichiamo con ^ l'operatore logico bit a bit XOR)

Capitolo 4

Software di controllo remoto

4 Software di controllo remoto

4.1 Generalità

Per connetterci ai robot Octagon è stato impiegato il modulo USB RF04 gestito tramite il componente *Microsoft Communications Control 6.0* che ci permetterà di gestire gli eventi generati dal modulo RF04.

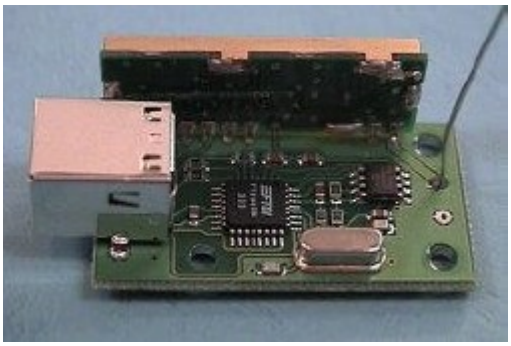


Figura 4.1: Modulo RF04

Il modulo RF04 viene fornito con i driver per sistemi operativi Microsoft e Mac (http://www.ftdichip.com/Drivers/VCP.htm) ma viene riconosciuto senza l'aggiunta di supporto alcuno da parte di sistemi Linux con kernel di versione successiva alla 2.6.9. La compatibilità con i più noti sistemi operativi è stata alla base della scelta del modulo RF04, che nel software dimostrativo riportato nel paragrafo 4.3 è stato gestito sotto il sistema operativo Microsoft Windows XP.

4.2 Descrizione della classe GenericRobot

Per comunicare con gli Octagon è stata progettata la classe astratta GenericRobot, che si occuperà di interpretare e codificare i dati da e verso i robot. Le istanze delle classi derivate da GenericRobot rappresenteranno oggetti nel mondo reale che a partire dagli Octagon verranno anch'essi specializzati in un particolare campo applicativo (ad esempio in Robot Artificieri). Da questa specializzazione conseguirà l'incremento delle funzionalità delle classi derivate (ad esempio con l'aggiunta dei metodi relativi alla gestione dell'hardware per il

disinnesco dell'esplosivo) e si è pertanto deciso di non rendere possibile la creazione di istanze appartenenti alla classe GenericRobot a meno di una derivazione della stessa.

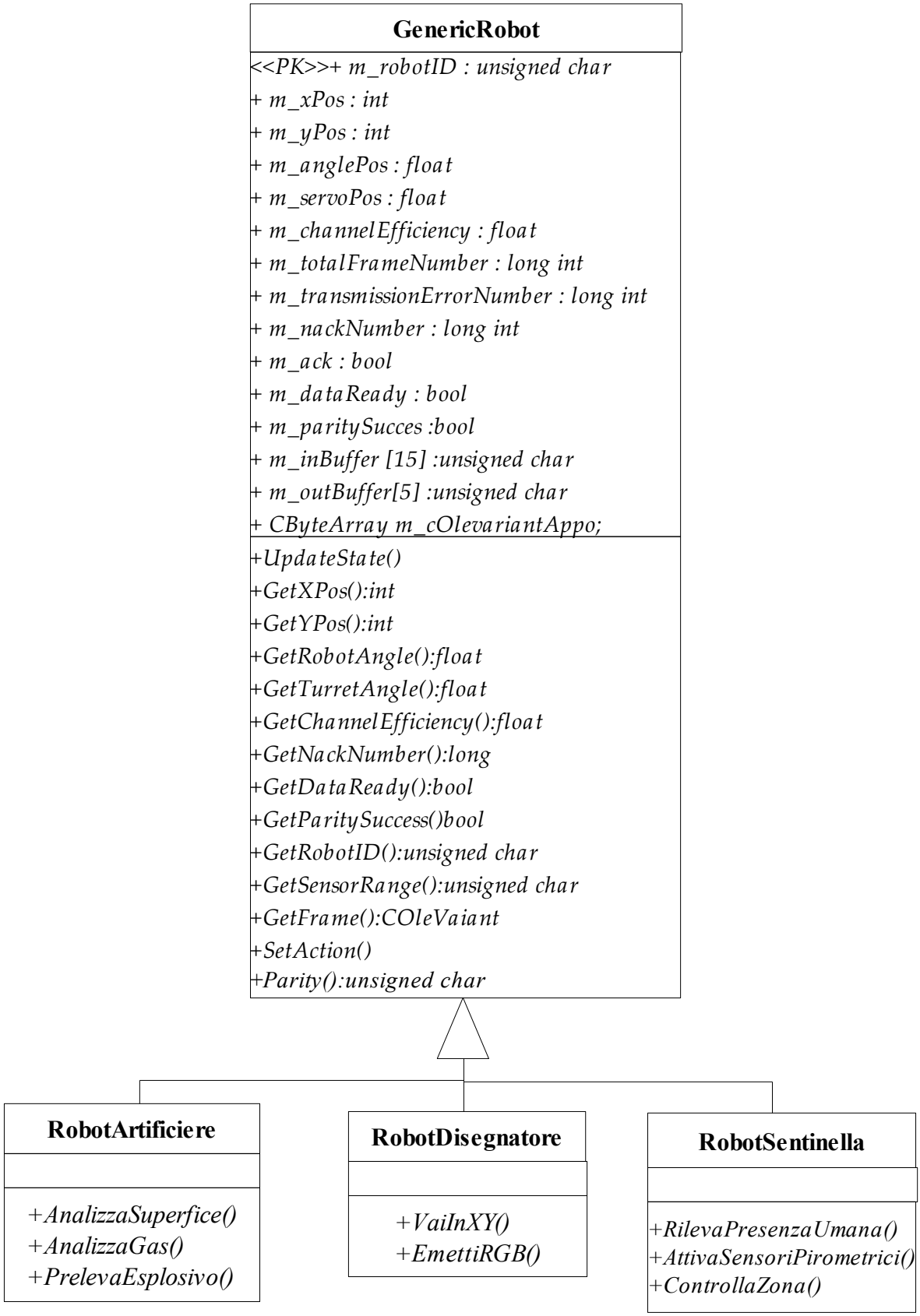


Figura 4.2: Possibili specializzazioni della classe GenericRobot

Di seguito è riportata la dichiarazione della classe GenericRobot:

```
class GenericRobot {
public:
        GenericRobot(unsigned char robotID);
        virtual ~GenericRobot()=0;
        int   GetXPos();
        int   GetYPos();
        float GetRobotAngle();
        float GetTurretAngle();
        float GetChannelEfficiency();
        long  GetNackNumber();
        bool  GetDataReady();
        bool  GetParitySuccess();
        unsigned char GetRobotID();
        unsigned char GetSensorRange(unsigned char sensorID);

        void UpdateState(COleVariant input);
        COleVariant GetFrame();
        void SetAction(unsigned int pulse,
                        unsigned char action,
                         unsigned char servoAngle);
private:
        unsigned char Parity(unsigned char *data,
                                unsigned short m_startByte,
                                unsigned short m_endByte);
```

```
    unsigned char m_inBuffer [15];
    unsigned char m_outBuffer[5];
    CByteArray m_cOlevariantAppo;

    int m_xPos;
    int m_yPos;
    float m_anglePos;
    float m_servoPos;
    float m_channelEfficiency;
    bool  m_ack;
    bool  m_dataReady;
    bool  m_paritySuccess;
    long int m_totalFrameNumber;
    long int m_transmissionErrorNumber;
    long int m_nackNumber;
    unsigned char m_robotID;
};
```

La classe *GenericRobot* implementa il protocollo di comunicazione utilizzato dalla stazione di controllo dei robot per interpretare e codificare i dati da e verso gli Octagon. La struttura dei frame che transitano nel canale fisico già discussa nel paragrafo 3.6 verrà elaborata dai metodi della classe GenericRobot nel modo seguente:

- *void UpdateState(COleVariant input)*

Il metodo *UpdateState* decodifica i dati contenuti nei frame ricevuti dai robot Octagon , che contengono le informazioni riguardanti le variabili posizionali, i sensori e la richiesta di ritrasmissione. Il risultato della

decodifica dei parametri posizionali verrà assegnato alle seguenti variabili membro:

- *int m_xPos; //coordinata assoluta X*
- *int m_yPos; //coordinata assoluta Y*
- *float m_anglePos; //posizione angolare robot*
- *float m_servoPos; //posizione angolare torretta*

UpdateState si avvale a sua volta del metodo *Parity* che permette di ricalcolare i byte di parità del frame ricevuto per poterli poi confrontare con quelli contenuti nel frame stesso. Nel caso in cui il confronto dia esito positivo il frame viene impiegato per aggiornare le variabili posizionali di cui sopra, in caso contrario il frame viene scartato e si incrementa la variabile membro *m_transmissionErrorNumber* . In ogni caso verrà incrementata la variabile *m_totalFrameNumber* che rappresenta il numero totale dei frame ricevuti. Il metodo UpdateState inoltre assegnerà il valore true alla variabile *m_ack* che rappresenta la conferma di ricezione dai robot Octagon se hanno ricevuto il comando senza errori, oppure il valore false che implica la ritrasmissione da parte della stazione di controllo del frame che non è stato ricevuto correttamente dai robot.

- *bool GetDataReady();*

Il metodo *GetDataReady* permette di determinare se l'ultimo comando trasmesso è stato ricevuto (nel caso restituisca false), oppure se nel canale radio stiano transitando i dati relativi all'ultimo frame inviato (in questo caso restituisce true).

- *void SetAction(unsigned int pulse, unsigned char action, unsigned char servoAngle);*

Il metodo *SetAction()* permette di codificare il numero di impulsi che il robot dovrà conteggiare prima di fermarsi, l'angolo di posizionamento della torretta ed il comando che il robot dovrà eseguire. Nei movimenti

traslazionali i robot Octagon compiono uno spostamento di 0.3195mm/impulso, mentre in quelli rotazionali di 0.1145°/impulso. I robot Octagon possono eseguire i seguenti comandi: {Ruota a sinistra, Avanza, Retrocedi, Ruota a destra, Riposiziona torretta}.

- *COleVariant GetFrame();*

Il metodo GetFrame() restituisce il frame di comando da inviare ai robot Octagon ed imposta la variabile membro m_dataReady a false se il robot non ha fatto richiesta di ritrasmissione. Se invece il robot ha inviato un nack , GetFrame() incrementerà m_nackNumber e preparerà i dati per la ritrasmissione.

- *bool GetDataReady();*

Restituisce true se l'ultimo frame non è stato ancora ricevuto correttamente dai robot Octagon oppure false nel caso in cui la trasmissione sia andata a buon fine.

- *float GetChannelEfficiency();*

L'ultimo metodo che analizziamo è *GetChannelEfficiency()* che restituisce un numero adimensionale dato dal seguente rapporto:

$$\frac{\text{m\_totalFrameNumber} - \text{m\_transmissionErrorNumber}}{\text{m\_totalFrameNumber}}$$

e quindi fornisce un'indicazione sull'efficienza del canale radio.

4.3 Software dimostrativo

Il programma dimostrativo per testare l'hardware dei robot Octagon ed il canale radio è stato sviluppato utilizzando l'ambiente di sviluppo Microsoft Visual C++ 6.0 che ha permesso la creazione di una gradevole interfaccia grafica in tempi brevi. Per la gestione della porta seriale è

stato impiegato il controllo MSCOMM sfruttando la tecnologia ActiveX che si basa sulla tecnologia Microsoft COM.

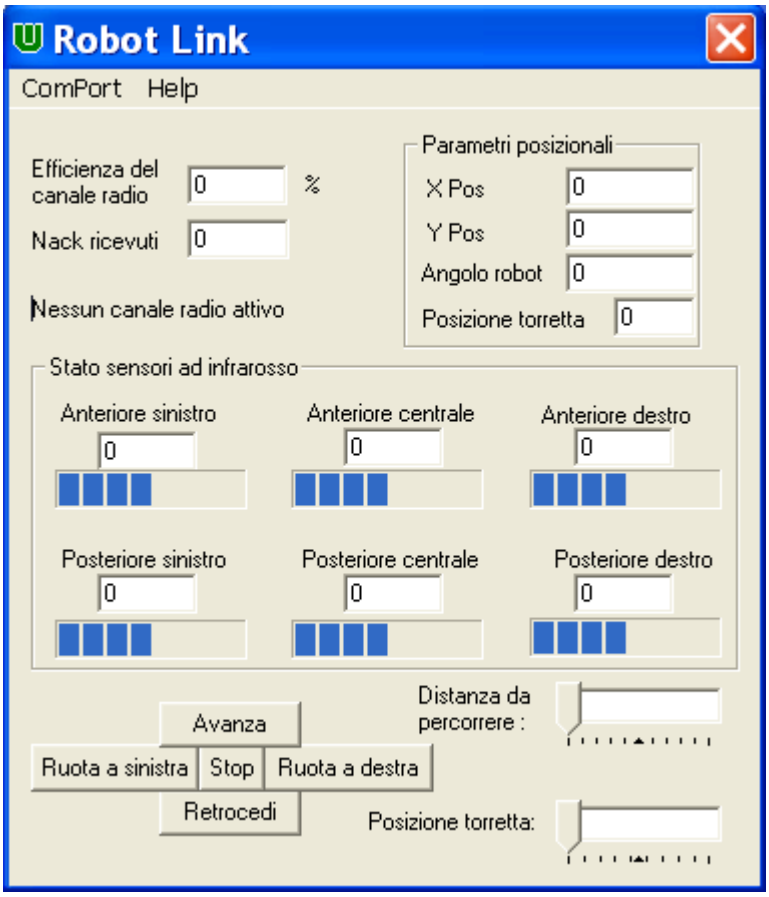


Figura 4.3: Software dimostrativo

Il controllo MSCOMM è stato inserito ed utilizzato nel nostro programma come se fosse una parte nativa dello stesso. Uno dei punti chiave dei controlli ActiveX è l'automazione che gli consente di attivarsi per controllare la propria parte di interfaccia , apportare le modifiche del caso e quindi disattivarsi quando il flusso di programma si trasferisce in un'altra parte dell'applicazione che non è controllata dal componente. Il funzionamento di questa automazione è dovuto ad una particolare interfaccia chiamata IDspatch la quale è costituita da un puntatore alla tabella dei metodi che possono essere eseguiti nel controllo ActiveX o in un modulo dell'applicazione che lo ospita. La tabella associa ad ogni metodo eseguibile dal controllo un ID chiamato

DISPID. Dalla conoscenza del DISPID del metodo desiderato lo si può richiamare tramite il metodo invoke() dell'interfaccia IDspatch a cui va passato il DISPID. La porta seriale è stata configurata con i seguenti parametri 19200,N,8,1 (19200 bps, nessuna parità, 8 bit di dati , 1 bit di stop) nel controllo MSCOMM che offre la possibilità di gestire la comunicazione in modalità Event-Driven, la quale ci consentirà di processare i frame in arrivo dagli Octagon con l'evento OnComm() . Tale evento viene generato ogni volta che la proprietà CommEvent cambia valore e nel caso specifico ogni volta che nel buffer di ricezione della USART ci sono un numero di byte maggiore o uguale a 15. La soglia di 15 byte è stata impostata con il metodo Rthreshold del controllo MSCOMM che setterà la proprietà ComEvent a ComEvReceive e genererà l'evento OnComm, gestito dal seguente codice :

```
void CRobotLinkDlg::OnOnCommMscomm1()
{
        m_Octagon.UpdateState(m_seriale.GetInput());

        UpdateData(true);

        if (m_Octagon.GetParitySuccess())
        {
           //Edit Box sensori frontali

            m_IR_Front_Left  =m_Octagon.GetSensorRange(0);
            m_IR_Front_Center=m_Octagon.GetSensorRange(1);
            m_IR_Front_Right =m_Octagon.GetSensorRange(2);

            //Edit Box sensori posteriori

             m_IR_Back_Left  =m_Octagon.GetSensorRange(3);
            m_IR_Back_Center=m_Octagon.GetSensorRange(4);
```

```
        m_IR_Back_Right =m_Octagon.GetSensorRange(5);

        //Progress Bar sensori frontali

        m_Progress_F_L.SetPos(m_Octagon.GetSensorRange(0));
        m_Progress_F_C.SetPos(m_Octagon.GetSensorRange(1));
        m_Progress_F_R.SetPos(m_Octagon.GetSensorRange(2));

        //Progress Bar sensori posteriori

        m_Progress_B_L.SetPos(m_Octagon.GetSensorRange(3));
        m_Progress_B_C.SetPos(m_Octagon.GetSensorRange(4));
        m_Progress_B_R.SetPos(m_Octagon.GetSensorRange(5));

        //Visualizzazione delle variabili posizionali

        m_X_Pos=m_Octagon.GetXPos();
        m_Y_Pos=m_Octagon.GetYPos();
        m_Robot_Angle=m_Octagon.GetRobotAngle();
        m_Servo_Angle=float(m_Servo_Control.GetValue());
        m_nack=m_Octagon.GetNackNumber();

    }

    m_Efficienza_Canale=m_Octagon.GetChannelEfficiency()*100;


    UpdateData(false);

    if (m_Octagon.GetDataReady())
      m_seriale.SetOutput(m_Octagon.GetFrame());

}
```

Dove con *m_seriale* indichiamo un'istanza della classe d'interfaccia per il controllo **MSCOMM** e con m_Octagon un'istanza della classe **RemoteControlledRobot,** derivata dalla classe **GenericRobot** e discussa nel paragrafo 4.2.

Capitolo 5

Considerazioni finali

5 *Considerazioni finali*

5.1 Considerazioni sul sistema

Nonostante i problemi derivanti dall'impossibilita di acquisire i moduli software/hardware adeguati nella fase di sviluppo dei robot , gli Octagon hanno rispettato tutte le specifiche imposte in ambito progettuale raggiungendo così i livelli di performance desiderati. La struttura monolitica del software di controllo dei robot, che ha portato in alcuni casi ad un pessimo stile di programmazione si è rivelata però indispensabile per adattare la complessità di tale componente di carico alla capacità di calcolo del microcontrollore impiegato. Per migliorare significativamente la reattività e la precisione degli Octagon è necessario lo sviluppo di un controller per i motori nelle cui MCU venga ripartita la componente di carico software che si occupa del conteggio degli impulsi generati dagli encoder di posizione ad effetto Hall. In questo modo saranno i controller dei motori ad occuparsi in totale parallelismo della gestione della posizione del robot e non il master controller, che potrà quindi dedicarsi alla gestione del canale radio, dell'apparato sensoriale e delle periferiche SPI(Serial Peripheral Interface). Scegliendo di delegare ai controller dei motori la gestione dei parametri di posizione, sarà possibile trattare molto più efficientemente il segnale generato dagli encoder grazie ad una significativa riduzione degli interrupt generati da quest'ultimi e non trattati. E' inoltre consigliabile portare all'esterno del vano, che negli Octagon ospita l'elettronica di controllo, gli stadi di potenza che si occupano dell'alimentazione dei motori. Questo permetterà di ridurre significativamente il rumore sulle linee dati causato dall'accoppiamento induttivo di quest'ultime con quelle che alimentano i motori. La scelta di radiocontrollare i robot apporta considerevoli vantaggi in ambiti quali la didattica e la ricerca favorendo lo sviluppo rapido di prototipi software, mentre non è la scelta ottimale in tutti quei casi in cui è invece richiesta l'autosufficienza dei robot (ad esempio quando le trasmissioni radio sono fortemente disturbate o non sono sicure). A tal proposito si segnalano i seguenti

dispositivi e la loro relativa applicazione negli Octagon finalizzata a rendere i robot indipendenti dalla stazione di controllo:

PIC 18F452 : rappresenta una valida alternativa al microcontroller BX24P e può essere sostituito al modulo radio per emulare la stazione di controllo.

Rabbit Semiconductor Core Modules: grazie alla notevole dotazione di hardware, il supporto ai protocolli TCP/IP ed un potente ambiente di sviluppo per il linguaggio C, i Core Modules della Rabbit Semiconductor rappresentano una delle più valide ed economiche soluzioni per l'emulazione della stazione di controllo.

I robot Octagon possono essere adattati ai più disparati casi d'uso come ad esempio: monitoraggio di aree pericolose per l'essere umano, sorveglianza, esplorazione, ed in particolare grazie alle loro ridotte dimensioni e semplicità d'uso rappresentano un valido strumento didattico.

Riferimenti bibliografici

[1]I. Sommerville; Software Engineering (7th ed.); [2004] Pearson, Addison Wesley

[2]NetMedia; Basic Express BX-24 Document; [2005] (http://www.basicx.com/transfer/bx_updates.htm)

[3]SHARP GP2D12 General Purpose Type Distance Measuring Sensors (http://sharp-world.com/products/device/lineup/selection/index.html#other)

[4]Motor Mind B Serial DC Motor Driver Module (http://www.solutions-cubed.com)

[5]M.D.Adams, H.Hu, P.J.Probert [1990], Toward a Real Time Architecture for Obstacle Avoidance and Path Planning in Mobile Robots in Proceedings of the 1990 IEEE International Conference on Robotics and Automation.

[6] J.Borenstein, L.Feng, H.R. Everett, D. Wehe; Mobile Robot Positioning - Sensors and Techniques, Journal of Robotic Systems, Special issue on mobile robots. Vol 14 No 4.

[7]Jiming Liu, Jianbing Wu; Multiagent Robotic Systems (International Series on Computational Intelligence) ; [May 30, 2001] CRC Press ISBN:084932288X

[8]Ulrich Nehmzow; Mobile Robotics: A pratical Introduction; [1999] Springer Verlag. ISBN 1-85233-173-9.

[9]Bruce Donald ,Kevin Lynch ,Daniela Rus; Algorithmic and Computational Robotics New Directions 2000 WAFR; [2001] ISBN: 1-56881-125-X

[10] Fred. G. Martin ;Robotic Exploration; [2001], Prentice Hall

[11] Jonathan H Connel; Minimalist Mobile Robotics; [July 28, 1990] Morgan Kaufmann ISBN: 012185230X

[12] Joseph L. Jones and Anita Flynn; Mobile robots (2nd ed.): inspiration to implementation ;[1998] A. K. Peters, Ltd. Natick, MA, USA ISBN:1-56881-097-0

[13]Penny Probert Smith, Penelope Probert Smith ;Active Sensors for Local Planning in Mobile Robotics (World Scientific Series in Robotics & Intelligent Systems); [2001] World Scientific Publishing ISBN: 9810246811

[14] Martin David Adams, Adams; Sensor Modelling, Design and Data Processing for Autonomous Navigation in Confined Environments;[1998] World Scientific Publishing Company ISBN: 9810234961

Capitolo 6

Listato

6 Listato

6.1 Codice sorgente per il software di controllo del robot

```
Option Explicit


'/*-------------------------------------------------------
'PARAMETRI DEL MODULO WIRELESS
'------------------------------------------------------- ----*/

Private Const serialDataInBufferSize  As Integer = 15              'dimensione del buffer di ingresso
Private Const serialDataOutBufferSize As Integer = 25              'dimensione del buffer di uscita

Private serialDataIn (1 To serialDataInBufferSize )  As Byte       'Buffer di ricezione   dal modulo wirelsss
Private serialDataOut(1 To serialDataOutBufferSize)  As Byte       'Buffer di trasmissione al modulo wireless

Private generalPurposeBuffer(1 To 7) As Byte



'/*---------------------------------------
'COSTANTI DI CONVERSIONE
'---------------------------------------*/

Private Const PULSE_TO_LINEAR_DISTANCE       As Single = 0.3195   'Millimetri/Impulso
Private Const PULSE_TO_ANGOLAR_DISTANCE   As Single = 0.0027   'Radianti/Impulso
Private Const PI_GRECO                                  As Single = 3.1415   'Radianti



'/*---------------------------------------
' PARAMETRI DEI SENSORI IR
'---------------------------------------*/

Private Const irNumber   As Byte = 6             'numero di sensori IR presenti sulle linee ADCs
Private irRange(1 to irNumber)  As Byte          'misurazioni effettuate dai sensori (in Cm)
Private irSucces(1 to irNumber) As Boolean    'esito delle misurazioni dei sensori



'/*-------------------------------------------------------------------------------------------------------
'DICHIARAZIONE DELLO STACK PER IL TASK DI GESTIONE DELLA TORRETTA
'-------------------------------------------------------------------------------------------------------*/

Private Const servoStackSize                    As Integer = 35
Private servoStack(1 To servoStackSize)    As Byte



'/*-------------------------------------------------------------------------------------------------------
'DICHIARAZIONE DELLO STACK  PER  IL TASK  DI  GESTIONE DEI MOTORI
'-------------------------------------------------------------------------------------------------------*/

Private Const engineStackSize                   As Integer = 35
Private engineStack (1 To engineStackSize) As Byte



'/*-----------------------------------------------------------------
'DICHIARAZIONE DEI PARAMETRI POSIZIONALI
'-----------------------------------------------------------------*/

Private deltaPulse          As Integer  'distanza da percorrere in impulsi
Private anglePos            As Single   'posizione angolare del robot   (radianti)
Private xPos                As Single   'posizione X assoluta del Robot (millimetri)
Private yPos                As Single   'posizione Y assoluta del Robot (millimetri)
```

```
'/*-------------------------------------------------------------------------------------------
'DICHIARAZIONE DEI PARAMETRI PER LA COMUNICAZIONE INTER-TASCK
'-------------------------------------------------------------------------------------------*/

Public semaphorePositionRefresh   As Boolean
Public turnOffEngine              As Boolean
Public spiIsNotFree               As Boolean
Public inversePulseCounter        As Integer
Public pulseCounter               As Integer


'/*-------------------------------------------------------------------------------------------
' PROCEDURA DI INIZIALIZZAZIONE DEGLI INTERRUPT  "InitializeBx24"
'-------------------------------------------------------------------------------------------
' Configurazione dell'INT0 come Input-pullup per prevenire
' la generazione di interrupt indesiderati
' L' INT0 viene attivato dal pin 11 del chip 8535 ,
' e non  è connesso a pin esterni del BX24P.
'-------------------------------------------------------------------------------------------*/

Private Sub InitializeBx24()

   Register.DDRD  = Register.DDRD And bx1111_1011          'configurazione dei registri di interrupt
   Register.PORTD = Register.PORTD Or bx0000_0100          'sul Port D del core processor

End Sub


'/*-------------------------------------------------------------------------------------------------------------------
'PROCEDURA PER AGGIORNARE I PARAMETRI POSIZIONALI "SetPosition"
'-------------------------------------------------------------------------------------------------------------------
'moveSense : direzione del movimento: {rotazione a sinistra, avanza, retrocedi, rotazione a destra}.
'pulse          : numero di impulsi conteggiati.
'-------------------------------------------------------------------------------------------------------------------*/

Public Sub SetPosition( ByVal moveSense As Byte, _
                        ByVal pulse         As Integer)

        Select Case moveSense
                Case 0    'rotazione a sinistra
                      anglePos=((CSng(pulse)* PULSE_TO_ANGOLAR_DISTANCE)+ anglePos)
                Case 1     'avanza
                     xPos=xPos+(CSng(pulse)* Cos(anglePos)* (PULSE_TO_LINEAR_DISTANCE))
                      yPos=yPos+ (CSng(pulse)* Sin(anglePos)* (PULSE_TO_LINEAR_DISTANCE))
                Case 2    'retrocedi
                       xPos=xPos-(CSng(pulse)* Cos(anglePos)* (PULSE_TO_LINEAR_DISTANCE))
                       yPos=yPos-(CSng(pulse)* Sin(anglePos)* (PULSE_TO_LINEAR_DISTANCE))
                Case 3      'rotazione a destra
                       anglePos=(anglePos-(CSng(pulse)* PULSE_TO_ANGOLAR_DISTANCE))
        End Select


End Sub



'/*-------------------------------------------------------------------------------------------------------------
' PROCEDURA PER LA CODIFICA DEI PARAMETRI DA TRASMETTERE  "SingleSplit"
'-------------------------------------------------------------------------------------------------------------
'highPart      :parte alta della variabile singleToSplit.
'lowPart       :parte bassa della variabile singleToSplit.
'singleToSplit :variabile da codificare per la trasmissione.
'-------------------------------------------------------------------------------------------------------------*/

Private Sub SingleSplit( ByRef highPart      As Byte, _
                         ByRef lowPart       As Byte, _
                         ByVal singleToSplit As Single)

Dim Appo As Integer

        Appo=FixI(Abs(singleToSplit))                   'conversione e troncamento della parte decimale di singleToSplit

        highPart=CByte(Appo\128) And bx0011_1111   'determinazione della parte alta da trasmettere
```

```
        If (singleToSplit<0.0) Then                     'assegnamento del bit di segno nella parte alta
          Call PutBit(highPart,6,1)
        End If

        lowPart=bx0111_1111                 'inizializzazione della parte bassa del numero da trasmettere

        'NON UTILIZZARE CICLI PER LE SEGUENTI ISTRUZIONI_____________________

        Call PutBit(lowPart,0,GetBit(Appo,0)) 'inizio trasferimento della parte bassa di singleToSplit
        Call PutBit(lowPart,1,GetBit(Appo,1))
        Call PutBit(lowPart,2,GetBit(Appo,2))
        Call PutBit(lowPart,3,GetBit(Appo,3))
        Call PutBit(lowPart,4,GetBit(Appo,4))
        Call PutBit(lowPart,5,GetBit(Appo,5))
        Call PutBit(lowPart,6,GetBit(Appo,6)) 'termine trasferimento della parte bassa di singleToSplit

End Sub



'/*-------------------------------------------------------------------------
' FUNZIONE PER IL COMPUTO DEI BYTE DI PARITA'
'-------------------------------------------------------------------------
'buffer    : dati su cui calcolare la parità.
'endByte   : indice finale dei dati su cui calcolare  la  parità.
'startByte : indice iniziale dei dati su cui calcolare la parità.
'-------------------------------------------------------------------------*/

Private Function SetParity( _
    ByRef buffer()   As Byte, _
    ByVal endByte    As Byte, _
    ByVal startByte  As Byte   ) As Byte

Dim      i As Byte
Dim parAppo As Byte


        parAppo=buffer(startByte)

        For i=(startByte+1) To endByte
          parAppo=buffer(i) Xor parAppo
        Next

        SetParity=parAppo

End Function



Public Sub Main()


'INIZIO DEFINIZIONE ED INIZIALIZZAZIONE DEI PARAMETRI DEL MASTER CONTROLLER_____________________________

Dim i              As Byte          'variabile generica di conteggio
Dim appo1      As Byte          'variabile generica di appoggio
Dim appo2      As Byte          'variabile generica di appoggio
Dim appo3      As Byte          'variabile generica di appoggio
Dim dataParityOk As Boolean     'flag di validazione parità

        i=0                                 'inizializzazione della variabile generica di conteggio
        xPos=0.0                            'inizializzazione della coordinata assoluta X
        yPos=0.0                            'inizializzazione della coordinata assoluta Y
        appo3=0                             'inizializzazione della variabile generica di appoggio
        anglePos=0.0                        'inizializzazione della posizione angolare iniziale dl robot
        servoAngle=0.52                     'inizializzazione della posizione iniziale della torretta (90°)
        pulseCounter=0                      'inizializzazione dell'accumulatore per conteggio impulsi
        inversePulseCounter=0               'inizializzazione dell'accumulatore per conteggio impulsi
        spiIsNotFree=False                  'inizializzazione del flag di sincronizzazione su SPI
        turnOffEngine=False                 'inizializzazione del flag di gestione motori
        semaphorePositionRefresh=True       'inizializzazione del flag di gestione aggiornamento variabili posizionali


        Call PutPin(25,1)       'disattiva il led di segnalazione: "Task gestione motori attivo"
```

```
        Call  InitializeBx24      'inizializza i registri di gestione degli interrupt del Core Processor
        Call  EngineInizialize  'inizializza i parametri di gestione dei motori

        CallTask "ServoTask", servoStack      'attivazione del Task di gestione della torretta sullo stack "servoStack".

        Call OpenQueue(serialDataIn,  serialDataInBufferSize)       'apertura buffer di ingresso della porta wireless
        Call OpenQueue(serialDataOut, serialDataOutBufferSize)   'apertura buffer di uscita   della porta wireless
        Call OpenCom(1, 19200, serialDataIn, serialDataOut)         'attivazione della porta seriale wireless a 19200 bps sui buffer:serialDataIn,
                                                                    'serialDataOut.

'FINE DEFINIZIONE E INIZIALIZZAZIONE DEI PARAMETRI DEL MASTER CONTROLLER_____________________________

 Do

        Call PutPin(26,1)                                   'inizia segnalazione di invio dati

        deltaPulse=inversePulseCounter-pulseCounter       'ottieni gli impulsi conteggiati dagli encoder fino a questo momento.
        inversePulseCounter=pulseCounter                  'aggiorna il contatore di impulsi
        Call SetPosition(GetMotorRotSense(),deltaPulse)  'aggiorna le variabili posizionali del robot
        Call IrRefresh(irRange,irSucces,irNumber)          'ottieni le misurazioni dai sensori IR


 'CONVERSIONE DEI PARAMETRI DI RILOCAZIONE PER LA TRASMISSIONE_____________________________________

        Call SingleSplit(generalPurposeBuffer(1),generalPurposeBuffer(2),xPos)                                  'codifica la coordinata X
        Call SingleSplit(generalPurposeBuffer(3),generalPurposeBuffer(4),yPos)                                  'codifica la coordinata Y
        Call SingleSplit(generalPurposeBuffer(5),generalPurposeBuffer(6), ((anglePos*180.0)/PI_GRECO))  'codifica la posizione angolare

        If (dataParityOk) Then
          generalPurposeBuffer(7)=bx0000_0000   'ricezione andata a buon fine
        Else
           generalPurposeBuffer(7)=bx0111_1111   'richiesta di ritrasmissione
        End If

        Do While (Not(Semaphore(spiIsNotFree)))  'impossessati della CPU per la seriale software

          Call Delay(0.01)                  'attendi 10 ms

        Loop

'INVIO DEI DATI TRAMITE MODULO WIRELESS_____________________________________________________________

          appo1=SetParity(generalPurposeBuffer,7,1)  'calcola il byte di parità delle coordinate del robot
          appo2=SetParity(irRange,6,1)                  'calcola il byte di parità delle letture  dei sensori

          Call PutQueue(serialDataOut,appo1,1)                    'Trasmetti il primo byte di parità
          Call PutQueue(serialDataOut,generalPurposeBuffer,7)   'Trasmetti le coordinate del robot
          Call PutQueue(serialDataOut,irRange,6)                  'Trasmetti le letture dei sensori
          Call PutQueue(serialDataOut,appo2,1)                    'Trasmetti l'ultimo byte di parità

          spiIsNotFree=False

'TERMINE DELL'INVIO DEI DATI_____________________________________________________________________

        Call Delay (0.05)                          'attendi che gli eventuali dati in arrivo siano pronti nel buffer di ricezione
        Call PutPin(26,0)                          'disattiva il led rosso per la segnalazione di trasmissione dati
        generalPurposeBuffer(2)=bx0111_1111 'elimina il vecchio comando ricevuto con una configurazione non valida


'INIZIO DELL'EVENTUALE RICEZIONE DATI_____________________________________________________________

        i=0

        Do While (StatusQueue(serialDataIn))   'finchè ci sono dati nel buffer di ricezione
          i=i+1
          Call GetQueue(serialDataIn,appo1, 1) 'estrai i byte dalla coda di ricezione


          If (i<=5) then
            generalPurposeBuffer(i)=appo1      'copia i dati in "generalPurposeBuffer"
          End If
        Loop
```

```
'TERMINE RICEZIONE_____________________________________________________________________

            if (i>0) then

              appo1=SetParity(generalPurposeBuffer,5,2)                    'calcola il byte di parità dei dati ricevuti

              dataParityOk=((appo1=generalPurposeBuffer(1)))               'verifica la parità


             If (generalPurposeBuffer(2)=5) and  (dataParityOk) then       'gestione comando "solo torretta"

               servoAngle=CSng(generalPurposeBuffer(5))/255.0              'aggiornamento della posizione della torretta

             Elseif (generalPurposeBuffer(2)<=4) and  (dataParityOk) then  'gestione comando "solo motori"

              Call SetEngineStop()                                         'arresto dei motori

            Do While (Not(semaphorePositionRefresh))         'inizio attesa per arresto motori

              Call Delay(0.01)                                'attendi 10 ms

              Loop

'TERMINE ATTESA PER ARRESTO MOTORI_____________________________________________________

 'AGGIORNAMENTO DELLE VARIBILI POSIZIONALI_____________________________________________

              deltaPulse=inversePulseCounter-pulseCounter          'ottieni gli impulsi conteggiati dagli encoder nella fase finale di arresto motori.

              Call SetPosition(GetMotorRotSense(),deltaPulse)      'aggiorna le variabili posizionali

              pulseCounter=CInt(generalPurposeBuffer(3))*127 + _
                                CInt(generalPurposeBuffer(4))      'aggiorna la variabile di conteggio impulsi
              inversePulseCounter=pulseCounter                     'aggiorna la variabile di conteggio impulsi

              Call SetEngine(100,98,generalPurposeBuffer(2))       'setta i parametri dei motori

              If (pulseCounter<>0) then                            'esegui se la distanza da percorrere non è nulla

                CallTask "EngineTask", engineStack                 'attiva Task per la gestione dei motori
                appo3=6

              End If

             End If

            End If

            If (appo3>=5) Then
              Call StopEngine()
             appo3=0
            End If

            appo3=appo3+1
 Loop

End Sub


Attribute VB_Name = "Engine_Protocol"
'-----------------------------------------------
Option Explicit

'/*-------------------------------------------------------------------------------------------------------------------
'PARAMETRI DEL CANALE SERIALE PER LA COMUNICAZIONE CON I CONTROLLER M.M.B.e.
'-------------------------------------------------------------------------------------------------------------------*/

Private motorData(1 to 7) As Byte                  'array di appoggio per dialogo con controller motori
Private motorRotSense     As Byte                  'verso di rotazione dei motori

Private Const inputBufferSize  As Integer = 12     'dimensione del buffer di ingresso (canale seriale motori)
Private Const outputBufferSize As Integer = 15     'dimensione del buffer di uscita   (canale seriale motori)
```

```
Private inputBuffer (1 To inputBufferSize)    As Byte       'Buffer di ricezione dai controller motori
Private outputBuffer(1 To outputBufferSize) As Byte       'Buffer di trasmissione verso controller motori


'/*-----------------------------------
'PARAMETRI MOTORE A
'-----------------------------------*/

Private Const MOTOR_A_Tx_PIN     As Byte = 10  'pin di trasmissione dati verso controller motore A
Private Const MOTOR_A_Rx_PIN     As Byte = 9   'pin di ricezione dati da controller motore A

Private motorSpeedA              As Byte   'velocità di rotazione del motore A (in passi da 0-125)
Private motorRotSenseA           As Byte   'verso di rotazione del motore A



'/*-----------------------------------
"PARAMETRI MOTORE B
'-----------------------------------*/

Private Const MOTOR_B_Tx_PIN     As Byte = 8 'pin di trasmissione dati verso controller motore B
Private Const MOTOR_B_Rx_PIN     As Byte = 7 'pin di ricezione dati da controller motore B

Private motorSpeedB                 As Byte       'velocità di rotazione del motore B (in passi da 0-125)
Private motorRotSenseB              As Byte       'verso di rotazione del motore B



'/*-------------------------------------------------------------------------
'-------------------------------------------------------------------------
'Il seguente Task si occupa di gestire la comunicazione con i
'controller dei motori e del conteggio degli impulsi generati
'dagli encoder
'-------------------------------------------------------------------------*/

Public Sub EngineTask()

Dim status            As Byte      'Status byte ricevuto dai controller
Dim engineSelect  As Byte      'numero del motore con cui si sta comunicando
Dim byteMarker    As Byte      'contatore per la numerazione dei byte in arrivo dai controller
Dim speedChek     As Byte      'velocità di rotazione dei motori comunicata dagli encoder

        semaphorePositionRefresh=False   'flag di sincronizzazione con il Main Task
        turnOffEngine=false              'flag di stato dei motori

        Call PutPin(25,0)                'segnalazione di motori attivi

        label_stopCycle:

'INIZIO ATTIVAZIONE DISATTIVAZIONE DEI MOTORI_____________________________________________________________

        For engineSelect=1 to 2

          speedChek=0

          Select Case engineSelect
                Case 1
                      motorData(3)=motorSpeedA*(1+(-2*motorRotSenseA))    'generazione del byte di controllo velocità e direzione in
                                                                           'complemento a due (range da -125 a 125).
                      motorData(6)=motorRotSenseA                          'verso di rotazione del motore A
                      motorData(7)=motorSpeedA                             'velocità di rotazione del motore A
                      Call DefineCom3(MOTOR_A_Tx_PIN, MOTOR_A_Rx_PIN, bx0000_1000)   'selezione Pin Tx/RX
                Case 2
                      motorData(3)=motorSpeedB*(1+(-2*motorRotSenseB))     'generazione del byte di controllo velocità e direzione in
                                                                           'complemento a due (range da -125 a 125).
                      motorData(6)=motorRotSenseB                          'verso di rotazione del motore B
                      motorData(7)=motorSpeedB                             'velocità di rotazione del motore B
                      Call DefineCom3(MOTOR_B_Tx_PIN, MOTOR_B_Rx_PIN, bx0000_1000)  'selezione Pin Tx/RX
          End Select


          Do

            byteMarker=0
```

```
            Call PutQueue(outputBuffer,motorData, 5)   'invia i dati ai controller
            Call Sleep(0.015)                          'attendi 15 ms

          Do While (StatusQueue(inputBuffer))          'analisi dei dati ricevuti dai controller
            status=speedchek
            byteMarker=byteMarker+1
            Call GetQueue(inputBuffer,speedChek, 1)    'leggi uno alla volta di dati nel buffer di ricezione
          Loop

            if ((speedChek= motorData(7)*2) and  (byteMarker=2) and (GetBit(status,0)= motorData(6))) then     'verifica che i dati ricevuti dal_
                                                                                                               'controller siano corretti.
             Exit Do
            ElseIf ((speedChek=0) and  (motorData(7)=0)) then
              Exit Do
            End If

          Loop

        Next

'TERMINE ATTIVAZIONE DISATTIVAZIONE DEI MOTORI_____________________________________________________________


      Do

        if ((pulseCounter>0) and (motorSpeedA<>0)) then   'se condizione vera continua il conteggio degli impulsi
          Call WaitForInterrupt(28)       'abilita il Task a rispondere agli interrupt generati dagli encoder dei motori (sospendi il Task fino al
                                          'prossimo impulso)
          pulseCounter=pulseCounter-1 'decrementa il contatore degli impulsi
        Else

          if (turnOffEngine) then                     'controlla che i motori siano spenti e se lo sono termina il task
            Exit Do
          End If

          turnOffEngine=True                          'abilita il flag per la disattivazione dei motori e la terminazione del Task

          motorSpeedA=0                               'azzera velocità motore A
          motorSpeedB=0                               'azzera velocità motore B
          motorRotSenseA=0                            'resetta il verso di rotazione motore A
          motorRotSenseB=0                            'resetta il verso di rotazione del motore B

          Do While (Not(Semaphore(spiIsNotFree)))  'controlla che SPI non sia occupato
            Call Sleep(0.0)
          Loop

          goto label_stopCycle                              'vai a disattivare i motori
        End if

      Loop


        Call PutPin(25,1)                             'disattiva il led di segnalazione motori attivi
        spiIsNotFree=False                            'libera SPI
        turnOffEngine=false                           'setta il flag di turnOffEngine per sincronizzazione con Main Task
        semaphorePositionRefresh=true                 'setta il flag di semaphorePositionRefresh  per sincronizzazione con Main Task

End Sub


'/*-------------------------------------------------------------------------------------------------------------------
' PROCEDURA PER L'INIZZIALIZZAZIONE DEI PARAMETRI DEI MOTORI   "EngineInizialize"
'-------------------------------------------------------------------------------------------------------------------*/

Public Sub EngineInizialize()

        motorData(1)=85 'byte di sincronizzazione con i controller
        motorData(2)=8  'comando SPDCON.SETDIR 'REVISIONE
        motorData(3)=0  'Velocità di rotazione dei motori

        motorData(4)=85 'byte di sincronizzazione con i controller
        motorData(5)=5  'comando di richiesta feedback dai controller
```

```
            motorData(6)=0  'verso di rotazione dei motori


             motorSpeedA=0 'azzera velocità motore A
             motorSpeedB=0 'azzera velocità motore B
             motorRotSenseA=0 'resetta il verso di rotazione motore A
             motorRotSenseB=0 'resetta il verso di rotazione del motore B



             Const portNumber As Byte = 3       'numero di porta per la seriale software
             Const baudRate As Long =9600       'baudRate per comunicazione con i controller; se i controller non rispondono verificare che
                                                'accidentalmente non abbiano cambiato la velocità di comunicazione a 2400 baud.

            Call OpenQueue(inputBuffer,   inputBufferSize)                                  'apertura della coda di ricezione dai controller
            Call OpenQueue(outputBuffer, outputBufferSize)                                  'apertura della coda di trasmissione ai controller
            Call DefineCom3(MOTOR_A_Rx_PIN  , MOTOR_A_Tx_PIN  , bx0000_1000) 'setta la seriale come inverted, no parity, 8 bits.
            Call OpenCom(portNumber, baudRate, inputBuffer, outputBuffer)                   'apre la porta sul BUS del motore A

End Sub



'/*-------------------------------------------------------------------------------------
' PROCEDURA PER SETTAGGIO PARAMETRI MOTORI "SetEngine"
'-------------------------------------------------------------------------------------
'motorSpA     : velocità di rotazione del motore A
'motorSpB     : velocità di rotazione del motore B
'motorRotSens : verso di rotazione dei motori
'motorRotSens =0000_0000 ruota a sinistra
'motorRotSens =0000_0001 avanza
'motorRotSens =0000_0010 retrocedi
'motorRotSens =0000_0011 ruota a destra
'-------------------------------------------------------------------------------------*/

Public Sub SetEngine( _
   ByVal motorSpA As Byte, _
   ByVal motorSpB As Byte, _
   ByVal motorRotSens As Byte)

            motorSpeedA=motorSpA
            motorSpeedB=motorSpB
            motorRotSense=motorRotSens

            motorRotSenseA=motorRotSens And bx0000_0001         'primo bit di "motorRotSens" è il verso del motore A
            motorRotSenseB=((motorRotSens\2) And bx0000_0001) 'secondo bit di "motorRotSens" è il verso di B

End Sub



'/*-----------------------------------------------------------------------------------------------------------------------------------------
' PROCEDURA PER LA LETTURA DELLA DIREZIONE ATTUALE DI ROTAZIONE DEI MOTORI "GetMotorRotSense"
'-----------------------------------------------------------------------------------------------------------------------------------------*/

Public Function GetMotorRotSense() As Byte
            GetMotorRotSense=motorRotSense
End Function



'/*-------------------------------------------------------------------------------------------------------------
' PROCEDURA PER IL SETTAGGIO DEI PARAMETRI DI ARRESTO MOTORI "EngineStop"
'-------------------------------------------------------------------------------------------------------------*/

Public Sub SetEngineStop()
            motorSpeedA=0
            motorSpeedB=0
End Sub

'/*-----------------------------------------------------------------
' PROCEDURA PER ARRESTO MOTORI "EngineStop"
'-----------------------------------------------------------------*/

Public Sub StopEngine()
```

```
            if (semaphorePositionRefresh) then
              motorData(2)=0              'comando di arresto rapido motori
              Call DefineCom3(MOTOR_B_Tx_PIN, MOTOR_B_Rx_PIN, bx0000_1000) 'Apre la seriale software sul canale B
              Call PutQueue(outputBuffer,motorData, 2)                     'Prova a fermare il motore B

              Call Delay(0.015)
              Call DefineCom3(MOTOR_A_Tx_PIN, MOTOR_A_Rx_PIN, bx0000_1000) 'Apre la seriale software sul canale A
              Call PutQueue(outputBuffer,motorData, 2)                     'Prova a fermare il motore A
              motorData(2)=8                                               'comando SPDCON.SETDIR
            End If
End Sub


Attribute VB_Name = "Servo_Motor"
'-----------------------------------------------
Option Explicit

'/*---------------------------------------
'PARAMETRI DELLA TORRETTA
'---------------------------------------*/


Public  servoAngle   As Single    'posizione della torretta


'/*-------------
'COSTANTI
'-------------*/

Private Const nSteps                 As Integer = 200        'numero di passi di scansione della torretta
Private Const servoPin               As Byte    = 12         'pin di connessione al servo
Private Const refreshPeriod          As Single  = 0.03       'frequenza 33 HZ
Private Const PWM_START              As Single  = 0.00069  'valore iniziale a 0°   (1.0 ms)
Private Const PWM_MULTIPLIER    As Single  = 0.00157  'valore finale   a 180° (2.0 ms)



'/*-------------------------------------------------------------------------------
TASK PER GESTIONE TORRETTA ServoTask()
'-------------------------------------------------------------------------------
'Il seguente Task si occupa di generare ad una frequenza di 33 HZ
'un'onda quadra di ampiezza opportuna sul pin 12 del BX24P
'-------------------------------------------------------------------------------*/

Public Sub ServoTask()

  Do
            Call PulseOut(servoPin, PWM_START + _
                          (PWM_MULTIPLIER * _
                           servoAngle), 1)                  'generazione del segnale modulato in ampiezza
            Call Sleep(refreshPeriod)                       'attesa di 0.03 secondi (33 cicli al secondo)
  Loop

End Sub


Attribute VB_Name = "Sharp_GP2D12_Ranger"
'-----------------------------------------------------
Option Explicit
'-----------------------------------------------------
' Sensore IR Sharp GP2D12
'-----------------------------------------------------

'/*-------------------------------------------------------------------------------------------------------
'PROCEDURA PER L'AGGIORNAMENTO DELLO STATO DEI SENSORI  "IrRefresh"
'-------------------------------------------------------------------------------------------------------
'distanceVector : array contenente le letture dei sensori
'successVector(): array contenente l'esito delle letture
'sensorNumber  : numero di sensori collegati
'-----------------------------------------------------------------*/

Public Sub IrRefresh( _
   ByRef distanceVector() As Byte, _
   ByRef successVector() As Boolean, _
   ByVal sensorNumber As Byte)
```

```
Dim i As Byte

            For i=1 To 3'sensorNumber
              Call GetRange(distanceVector(i), successVector(i),20)
            Next

            For i=4 To 6'sensorNumber
              Call GetRange(distanceVector(i), successVector(i),19)
            Next

End Sub



'/*-------------------------------------------------------------------------------
' PROCEDURA PER GESTIRE I SENSORI Sharp GP2D12   "GetRange"
'-------------------------------------------------------------------------------
'distance: misurazione effettuata
'success:  esito della misurazione
'inputPin: pin con funzione ADC a cui è connesso il sensore
'-------------------------------------------------------------------------------*/

Private Sub GetRange( _
   ByRef distance As Byte, _
   ByRef success As Boolean, _
   ByVal inputPin As Byte)

Dim voltage      As Single
Dim range        As Single
Const minVolt    As Single = 0.399
Const maxVolt    As Single = 2.60

   Call GetADC(inputPin, voltage)

   voltage = voltage * 5.0                          'passa da mV (la risoluzione degli Adc è di 5mv) a Volt

   range = voltageToRange(voltage)                  'inverte la funzione distanza(volt)

   distance=FixB(range*100.0) And bx0011_1111  'converti in byte


   If (voltage >= minVolt) And (voltage <= maxVolt) Then   'controlla se la misurazione è andata a buon fine
     Call PutBit(distance,6,0)                             'inserisci il flag di esito positivo nel byte da inviare via radio
     success = True
   Else
     Call PutBit(distance,6,1)                             'inserisci il flag di esito negativo nel byte da inviare via radio
     success = False
   End If

End Sub



'/*-------------------------------------------------------------------------------------------------------------------
' PROCEDURA PER INTERPRETARE LE MISURAZIONI DEI SENSORI Sharp GP2D12 "voltageToRange"
'-------------------------------------------------------------------------------------------------------------------
'voltage: tensione generata dal sensore
'-------------------------------------------------------------------------------------------------------------------*/

Private Function voltageToRange( _
   ByVal voltage As Single) As Single

  'META' INFERIORE DELL'INTERVALLO DI TENSIONE GENERATO DAL SENSORE_____________

   If (voltage < 0.8181650) Then

      If (voltage < 0.5860420) Then

         voltageToRange = -1.431295831 * voltage + 1.371201638    'segmento 1

      Else

         voltageToRange = -0.751946517 * voltage + 0.973074437'segmento 2
```

```
      End If

 'META' SUPERIORE DELL'INTERVALLO DI TENSIONE GENERATO DAL SENSORE_______

   Else
      If (voltage < 1.1492463) Then

         voltageToRange = -0.349412673 * voltage + 0.643735317   'segmento 3

      ElseIf (voltage < 1.6363301) Then

         voltageToRange = -0.177086187 * voltage + 0.445689738   'segmento 4

      Else

         voltageToRange = -0.076191604 * voltage + 0.280592896   'segmento 5

      End If
   End If

End Function



Attribute VB_Name = "I2C"
'-----------------------------------
Option Explicit

'/*-------------------------------------------------------------
'Dichiarazione dei parametri del BUS I2C
'E' possibile scegliere qualsiasi pin di I/O
'disponibile sul BX24P come linee SCA ed SDL
'-------------------------------------------------------------*/

Public Const SCL As Byte = 6        ' I2C linea clock
Public Const SDA As Byte = 5        ' I2C linea dati
Private I2cAck    As Boolean



'/*-----------------------------------------------------------------------------------------------
' PROCEDURA PER L'INVIO DI 1 BYTE SUL BUS I2C       "I2cByteWrite"
'-----------------------------------------------------------------------------------------------
'I2cAddr: indirizzo della periferica a cui inviare i dati
'I2cReg : registro interno alla periferica su cui scrivere
'I2cData: dato da inviare
'-----------------------------------------------------------------------------------------------*/

Public Sub I2cByteWrite(ByVal I2cAddr As Byte, _
                        ByVal I2cReg As Byte,   _
                        ByVal I2cData As Byte)
            Call I2cStart
            Call I2cOutByte(I2cAddr)                'invia l'indirizzo del dispositivo
            Call I2cOutByte(I2cReg)                 'invia l'indirizzo del registro nel dispositivo
            Call I2cOutByte(I2cData)                'invia i dati
            Call I2cStop
End Sub



'/*-----------------------------------------------------------------------------------------------
' FUNZIONE PER LA RICEZIONE DI 1 BYTE DAL BUS I2C     "I2CByteRead"
'-----------------------------------------------------------------------------------------------
'I2cAddr: indirizzo della periferica da cui effettuare la lettura
'I2cReg : registro interno alla periferica da esportare
'-----------------------------------------------------------------------------------------------*/

Public Function I2CByteRead(ByVal I2cAddr As Byte, _
                            ByVal  I2cReg As Byte) As Byte

            Call I2cStart
            Call I2cOutByte(I2cAddr)                'invia l'indirizzo del dispositivo
            Call I2cOutByte(I2cReg)                 'invia l'indirizzo del registro nel dispositivo
            Call I2cStart                           'rigenera la condizione di inizio
```

```
        I2cAddr = I2cAddr + 1
        Call I2cOutByte(I2cAddr)                'invia l'indirizzo del device su cui effettuare la lettura
        I2cAck = False                          'setta il protocollo per l'invio del Nak
        I2CByteRead = I2cInByte()               'acquisisci il dato e invia il Nak
        Call I2cStop                            'genera la condizione di arresto
End Function



'/*-------------------------------------------------------------------------------------------
' FUNZIONE PER LA RICEZIONE DI 2 BYTE SUL BUS I2C    "I2CWordRead"
'-------------------------------------------------------------------------------------------
'I2cAddr: indirizzo della periferica da cui effettuare la lettura
'I2cReg : registro interno alla periferica da esportare
'-------------------------------------------------------------------------------------------*/

Public Function I2CWordRead(ByVal I2cAddr As Byte, _
                            ByVal I2cReg As Byte) As UnsignedInteger

        Set I2CWordRead = New UnsignedInteger
        Call I2cStart
        Call I2cOutByte(I2cAddr)                'invia l'indirizzo del dispositivo
        Call I2cOutByte(I2cReg)                 'invia l'indirizzo del registro nel dispositivo
        Call I2cStart                           'rigenera la condizione di inizio
        I2cAddr = I2cAddr + 1
        Call I2cOutByte(I2cAddr)                'invia l'indirizzo del device su  cui effettuare la lettura
        I2cAck = True                           'setta il protocollo per l'invio dell' Ack
        I2CWordRead = CuInt(I2cInByte() * 256)
        I2cAck = False                          'setta il protocollo per l'invio del Nak
        I2CWordRead = I2CWordRead + CuInt(I2cInByte())
        Call I2cStop
End Function



'/*-------------------------------------------------------------------
' PROCEDURA PER L'INVIO DI UN BYTE  "I2cOutByte"
'-------------------------------------------------------------------
'I2cData : dato da inviare
'-------------------------------------------------------------------*/

Private Sub I2cOutByte(I2cData As Byte)
        Call ShiftOut(SDA, SCL, 8, I2cData)     ' invia i bit in sequenza
        Call PutPin(SDA, bxInputTristate)       ' cambia lo stato della linea dati
        Call PutPin(SCL, bxOutputHigh)          ' sincronizza il bite di Ack
        Call PutPin(SCL, bxOutputLow)
End Sub



'/*-------------------------------------------------------------------------------------------
' FUNZIONE PER LA RICEZIONE DI UN BYTE DAL BUS I2C    "I2cInByte"
'-------------------------------------------------------------------------------------------*/

Private Function I2cInByte() As Byte
        I2cInByte = ShiftIn(SDA, SCL, 8)
        If I2cAck = True Then
          Call PutPin(SDA, bxOutputLow)
        Else
          Call PutPin(SDA, bxOutputHigh)
        End If
        Call PutPin(SCL, bxOutputHigh)      'generazione del bit di ack per il byte ricevuto
        Call PutPin(SCL, bxOutputLow)
End Function



'/*-------------------------------------------------------------------------------------------
' PROCEDURA PER GENERAZIONE CONDIZIONE DI INIZIO  "I2cStart"
'-------------------------------------------------------------------------------------------
'Procedura per la generazione della condizione di inizio
'trasmissione  sul bus I2C.
'-------------------------------------------------------------------------------------------*/

Private Sub I2cStart()
```

```
            Call PutPin(SDA, bxOutputHigh)  'inizio sequenza di bit per start
            Call PutPin(SCL, bxOutputHigh)
            Call PutPin(SDA, bxOutputLow)
            Call PutPin(SCL, bxOutputLow)   'fine sequenza di bit per start
End Sub



'/*-------------------------------------------------------------------------------------------
' PROCEDURA PER GENERAZIONE CONDIZIONE DI STOP    "I2cStop"
'-------------------------------------------------------------------------------------------
'Procedura per la generazione della condizione di fine
'trasmissione  sul bus I2C.
'-------------------------------------------------------------------------------------------*/

Private Sub I2cStop()
            Call PutPin(SDA, bxOutputLow)    'Inizio sequenza di bit per stop
            Call PutPin(SCL, bxOutputHigh)
            Call PutPin(SDA, bxOutputHigh)   'fine sequenza di bit per stop
End Sub
```

6.2 Codice sorgente per il software di controllo remoto

// RobotLinkDlg.h : header file

```
//
//{{AFX_INCLUDES()
#include "mscomm.h"
#include "slider.h"
//}}AFX_INCLUDES

#if !defined(AFX_ROBOTLINKDLG_H__56843C18_542E_40DE_AA56_C4BE3456FBB4__INCLUDED_)
#define AFX_ROBOTLINKDLG_H__56843C18_542E_40DE_AA56_C4BE3456FBB4__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

/*////////////////////////////////////////////////////////////////////////////////////////
DEFINIZIONE DELLA CLASSE ASTRATTA GenericRobot{}
////////////////////////////////////////////////////////////////////////////////////////*/
class GenericRobot {

public:

            virtual     ~GenericRobot()=0{};
            GenericRobot(unsigned char robotID);

            int   GetXPos();
            int   GetYPos();
            float GetRobotAngle();
            float GetTurretAngle();

            long  GetNackNumber();
            bool  GetDataReady();
            bool  GetParitySuccess();
            float GetChannelEfficiency();

            unsigned char GetRobotID();
            unsigned char GetSensorRange(unsigned char sensorID);

            COleVariant GetFrame();
            void UpdateState(COleVariant input);
            void SetAction(unsigned int pulse,
                        unsigned char action,
```

```
                        unsigned char servoAngle);


private:

        unsigned char Parity(unsigned char *data,
                             unsigned short m_startByte,
                             unsigned short m_endByte );


        unsigned char m_inBuffer [15];
        unsigned char m_outBuffer[5];
        CByteArray m_cOlevariantAppo;

        int m_xPos;
        int m_yPos;
        float m_anglePos;
        float m_servoPos;
        float m_channelEfficiency;
        bool  m_ack;
        bool  m_dataReady;
        bool  m_paritySuccess;
        long int m_totalFrameNumber;
        long int m_transmissionErrorNumber;
        long int m_nackNumber;
        unsigned char m_robotID;
};


/*////////////////////////////////////////////////////////////////////////
DERIVAZIONE DELLA CLASSE GENERIC ROBOT
////////////////////////////////////////////////////////////////////////*/

class RemoteControlledRobot: public GenericRobot
{
public:
        ~RemoteControlledRobot(){};
         RemoteControlledRobot(unsigned char robotID):GenericRobot(robotID){};
};




/////////////////////////////////////////////////////////////////////////////
// CRobotLinkDlg dialog

class CRobotLinkDlg : public CDialog
{
// Construction
public:
        CRobotLinkDlg(CWnd* pParent = NULL);          // costruttore standard

// Dialog Data
        //{{AFX_DATA(CRobotLinkDlg)
        enum { IDD = IDD_ROBOTLINK_DIALOG };
        CProgressCtrl       m_Progress_F_R;
        CProgressCtrl       m_Progress_F_L;
        CProgressCtrl       m_Progress_F_C;
        CProgressCtrl       m_Progress_B_R;
        CProgressCtrl       m_Progress_B_L;
        CProgressCtrl       m_Progress_B_C;
        COleVariant myVar;
        CMSComm             m_seriale;
        BOOL     m_flag1;
```

```
	int		m_X_Pos;
	int		m_Y_Pos;
	BYTE	m_IR_Back_Center;
	BYTE	m_IR_Back_Left;
	BYTE	m_IR_Back_Right;
	BYTE	m_IR_Front_Left;
	BYTE	m_IR_Front_Right;
	float	m_Robot_Angle;
	float	m_Servo_Angle;
	CSlider	m_Servo_Control;
	int		m_IR_Front_Center;
	CSlider	m_Pulse_Slider;
	long	m_nack;
	float	m_Efficienza_Canale;
	CString	m_Stato_Canale;
	//}}AFX_DATA

	// Overriding della funzione virtuale
	//{{AFX_VIRTUAL(CRobotLinkDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// supporto DDX/DDV
	//}}AFX_VIRTUAL

// Implementation
protected:
	HICON m_hIcon;

	// Funzioni di gestione dei messaggi
	//{{AFX_MSG(CRobotLinkDlg)
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	afx_msg void OnOnCommMscomm1();
	afx_msg void OnCheck1();
	virtual void OnCancel();
	afx_msg void OnGoForward();
	afx_msg void OnRotateLeft();
	afx_msg void OnRotateRight();
	afx_msg void OnGoBackward();
	afx_msg void OnClickServoControl();
	afx_msg void OnStopengine();
	afx_msg void OnComportCom1();
	afx_msg void OnComportCom2();
	afx_msg void OnComportCom3();
	afx_msg void OnComportCom4();
	afx_msg void OnComportCom5();
	afx_msg void OnComportCom6();
	afx_msg void OnComportCom7();
	afx_msg void OnComportCom8();
	afx_msg void OnComportCom9();
	afx_msg void OnComportExit();
	afx_msg void OnHelp();
	DECLARE_EVENTSINK_MAP()
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
private:
	void SetCom(unsigned char port);
	RemoteControlledRobot m_Octagon(0);
};


//{{AFX_INSERT_LOCATION}}
```

```
#endif // !defined(AFX_ROBOTLINKDLG_H__56843C18_542E_40DE_AA56_C4BE3456FBB4__INCLUDED_)


// RobotLinkDlg.cpp : file di implementazione
//

#include "stdafx.h"
#include "RobotLink.h"
#include "RobotLinkDlg.h"
#include "afxcoll.h"
#include "stdlib.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif




/*////////////////////////////////////////////////////////
Implementazione della classe GenericRobot
////////////////////////////////////////////////////////*/


GenericRobot::GenericRobot(unsigned char robotID)
                :m_robotID(m_robotID)
{

        m_ack=true;
        m_nackNumber=0;
        m_dataReady=true;
        m_paritySuccess=false;
        m_totalFrameNumber=1;
        m_channelEfficiency=1;
        m_transmissionErrorNumber=0;
        m_cOlevariantAppo.SetSize(5);

        for (int i=0;i<=4;i++)
        {
          m_cOlevariantAppo[i]=0;
          m_inBuffer[i]=0;
        }
}



void GenericRobot::UpdateState(COleVariant input)
{

        ++m_totalFrameNumber;

        for (int i=0;i<=14;i++)
        {
          m_inBuffer[i]=unsigned char(input.bstrVal[i]);
        }


        //controllo della parità
        m_paritySuccess=((Parity(m_inBuffer,1,7 )==m_inBuffer[0]) &&
                         (Parity(m_inBuffer,8,13)==m_inBuffer[14]));
```

```
        if (m_paritySuccess)
        {

         m_xPos=int(((m_inBuffer[1]&63)*128+ m_inBuffer[2])* //decodifica della coordinata X
                    (((m_inBuffer[1] & 64)>>6)*(-2)+1));          //decodifica del segno


         m_yPos=int(((m_inBuffer[3]&63)*128+ m_inBuffer[4])*   //decodifica della coordinata Y
                     (((m_inBuffer[3] & 64)>>6)*(-2)+1));          //decodifica del segno


         m_anglePos=float((((m_inBuffer[5]&63)*128+ m_inBuffer[6])* //decodifica della coordinata
                           (((m_inBuffer[5] & 64)>>6)*(-2)+1))%      //decodifica del segno
                           360);                                     //adattamento al range[0°-360°]


         m_ack=(m_inBuffer[7]==0); //richiesta di ritrasmissione
        }
        else
        {
         ++m_transmissionErrorNumber;
        }

}



void GenericRobot::SetAction(unsigned int pulse,
                             unsigned char action,
                             unsigned char servoAngle)
{
        m_outBuffer[1]=action;
        m_outBuffer[2]=(pulse>>6)& 127;  //parte alta di pulse
        m_outBuffer[3]=pulse & 127;      //parte bassa di pulse
        m_outBuffer[4]=servoAngle;
        m_dataReady=true;
}



COleVariant GenericRobot::GetFrame()
{
        m_outBuffer[0]=Parity(m_outBuffer,1,4);

        if(m_ack)
        {
         m_dataReady=false;
        }
        else
        {
         ++m_nackNumber;
        }

        for (int i=0;i<=4;i++)
          m_cOlevariantAppo[i]=m_outBuffer[i];

        return (COleVariant(m_cOlevariantAppo));
}




unsigned char GenericRobot::Parity(unsigned char *data,
```

```
                                        unsigned short m_startByte,
                                        unsigned short m_endByte )
{
unsigned char parity;

        parity=unsigned char(data[m_startByte]);

        for (int i=(m_startByte+1);i<=m_endByte;i++)
          parity=(parity ^ data[i]);

        return(parity);
}



float GenericRobot::GetChannelEfficiency()
{
        m_channelEfficiency=(float(m_totalFrameNumber-
                                m_transmissionErrorNumber)/
                                float(m_totalFrameNumber));


        if (m_totalFrameNumber>=25)  //sensibilità delle statistiche agli errori
        {
          m_totalFrameNumber/=2;
          m_transmissionErrorNumber/=2;
        }

        if (m_channelEfficiency>=0)
        {
          return (m_channelEfficiency);
        }
        else
        {
          return (0);
        }

}



int GenericRobot::GetXPos()
{
        return (m_xPos);
}



int GenericRobot::GetYPos()
{
        return (m_yPos);
}



float GenericRobot::GetRobotAngle()
{
        return (m_anglePos);
}



bool GenericRobot::GetDataReady()
{
```

```
	return(m_dataReady);
}


bool GenericRobot::GetParitySuccess()
{
	return(m_paritySuccess);
}


float GenericRobot::GetTurretAngle()
{
	return (m_servoPos);
}


long GenericRobot::GetNackNumber()
{
	return (m_nackNumber);
}

unsigned char GenericRobot::GetRobotID()
{
	return (m_robotID);
}


unsigned char GenericRobot::GetSensorRange(unsigned char sensorID)
{
	if (sensorID<=5)
	{
	  return (m_inBuffer[8+sensorID]);
	}
	else
	{
	  return (0);
	}
}


/*////////////////////////////////////////////////////////////////////////
Termine implementazione della classe GenericRobot
////////////////////////////////////////////////////////////////////////*/

/*////////////////////////////////////////////////////////////////
Metodo per la gestione degli eventi generati
dal controllo Mscomm1.
////////////////////////////////////////////////////////////////*/

void CRobotLinkDlg::OnOnCommMscomm1()
{

  m_Octagon.UpdateState(m_seriale.GetInput());

  UpdateData(true);

	if (m_Octagon.GetParitySuccess())
	{
	   //Edit Box sensori frontali
```

```
            m_IR_Front_Left  =m_Octagon.GetSensorRange(0);
            m_IR_Front_Center=m_Octagon.GetSensorRange(1);
            m_IR_Front_Right =m_Octagon.GetSensorRange(2);

            //Edit Box sensori posteriori

             m_IR_Back_Left  =m_Octagon.GetSensorRange(3);
            m_IR_Back_Center=m_Octagon.GetSensorRange(4);
            m_IR_Back_Right =m_Octagon.GetSensorRange(5);

             //Progress Bar sensori frontali

             m_Progress_F_L.SetPos(m_Octagon.GetSensorRange(0));
             m_Progress_F_C.SetPos(m_Octagon.GetSensorRange(1));
             m_Progress_F_R.SetPos(m_Octagon.GetSensorRange(2));

            //Progress Bar sensori posteriori

            m_Progress_B_L.SetPos(m_Octagon.GetSensorRange(3));
            m_Progress_B_C.SetPos(m_Octagon.GetSensorRange(4));
            m_Progress_B_R.SetPos(m_Octagon.GetSensorRange(5));

            //Visualizzazione delle variabili posizionali

             m_X_Pos=m_Octagon.GetXPos();
             m_Y_Pos=m_Octagon.GetYPos();
             m_Robot_Angle=m_Octagon.GetRobotAngle();
             m_Servo_Angle=float(m_Servo_Control.GetValue());
             m_nack=m_Octagon.GetNackNumber();
           }

           m_Efficienza_Canale=m_Octagon.GetChannelEfficiency()*100;

           UpdateData(false);

           if (m_Octagon.GetDataReady())
             m_seriale.SetOutput(m_Octagon.GetFrame());

}



/*////////////////////////////////////////////////////////////
Metodo per la selezione della porta seriale
////////////////////////////////////////////////////////////*/

void CRobotLinkDlg::SetCom(unsigned char port)
{
           UpdateData(true);

           if (m_seriale.GetPortOpen())
           {
             m_seriale.SetPortOpen(false);
             m_seriale.SetCommPort(port);
             m_seriale.SetPortOpen(true);
           }
           else
           {
             m_seriale.SetCommPort(port);
             m_seriale.SetPortOpen(true);
           }

           m_Stato_Canale="Canale radio attivo: " + CString(port+48);
```

```
	UpdateData (false);
}



class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

 // Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // supporto DDX/DDV
	//}}AFX_VIRTUAL

// Implementazione
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
	  // No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CRobotLinkDlg dialog

CRobotLinkDlg::CRobotLinkDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CRobotLinkDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CRobotLinkDlg)
	m_nack = 0;
	m_X_Pos = 0;
	m_Y_Pos = 0;
	m_flag1 = FALSE;
	m_IR_Back_Left = 0;
	m_IR_Back_Right = 0;
	m_IR_Front_Left = 0;
	m_IR_Back_Center = 0;
	m_IR_Front_Right = 0;
	m_Robot_Angle = 0.0f;
```

```
	m_Servo_Angle = 0.0f;
	m_IR_Front_Center = 0;
	m_Efficienza_Canale = 0.0f;
	m_Stato_Canale = _T("Nessun canale attivo.");
	//}}AFX_DATA_INIT
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CRobotLinkDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CRobotLinkDlg)
	DDX_Control(pDX, IDC_PROGRESS_F_R, m_Progress_F_R);
	DDX_Control(pDX, IDC_PROGRESS_F_L, m_Progress_F_L);
	DDX_Control(pDX, IDC_PROGRESS_F_C, m_Progress_F_C);
	DDX_Control(pDX, IDC_PROGRESS_B_R, m_Progress_B_R);
	DDX_Control(pDX, IDC_PROGRESS_B_L, m_Progress_B_L);
	DDX_Control(pDX, IDC_PROGRESS_B_C, m_Progress_B_C);
	DDX_Control(pDX, IDC_MSCOMM1, m_seriale);
	DDX_Text(pDX, IDC_X_POS, m_X_Pos);
	DDX_Text(pDX, IDC_Y_POS, m_Y_Pos);
	DDX_Text(pDX, IDC_IR_BACK_CENTER, m_IR_Back_Center);
	DDX_Text(pDX, IDC_IR_BACK_LEFT, m_IR_Back_Left);
	DDX_Text(pDX, IDC_IR_BACK_RIGHT, m_IR_Back_Right);
	DDX_Text(pDX, IDC_IR_FRONT_LEFT, m_IR_Front_Left);
	DDX_Text(pDX, IDC_IR_FRONT_RIGHT, m_IR_Front_Right);
	DDX_Text(pDX, IDC_ROBOT_ANGLE, m_Robot_Angle);
	DDX_Text(pDX, IDC_SERVO_ANGLE, m_Servo_Angle);
	DDX_Control(pDX, IDC_Servo_Control, m_Servo_Control);
	DDX_Text(pDX, IDC_IR_FRONT_CENTER, m_IR_Front_Center);
	DDX_Control(pDX, IDC_Pulse_Slider, m_Pulse_Slider);
	DDX_Text(pDX, IDC_NACK_EDIT, m_nack);
	DDX_Text(pDX, IDC_EDIT5, m_Efficienza_Canale);
	DDX_Text(pDX, IDC_EDIT1, m_Stato_Canale);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CRobotLinkDlg, CDialog)
	//{{AFX_MSG_MAP(CRobotLinkDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_Go_Forward, OnGoForward)
	ON_BN_CLICKED(IDC_Rotate_Left, OnRotateLeft)
	ON_BN_CLICKED(IDC_Rotate_Right, OnRotateRight)
	ON_BN_CLICKED(IDC_Go_Backward, OnGoBackward)
	ON_BN_CLICKED(IDC_STOPENGINE, OnStopengine)
	ON_COMMAND(IDM_COMPORT_COM1, OnComportCom1)
	ON_COMMAND(IDM_COMPORT_COM2, OnComportCom2)
	ON_COMMAND(IDM_COMPORT_COM3, OnComportCom3)
	ON_COMMAND(IDM_COMPORT_COM4, OnComportCom4)
	ON_COMMAND(IDM_COMPORT_COM5, OnComportCom5)
	ON_COMMAND(IDM_COMPORT_COM6, OnComportCom6)
	ON_COMMAND(IDM_COMPORT_COM7, OnComportCom7)
	ON_COMMAND(IDM_COMPORT_COM8, OnComportCom8)
	ON_COMMAND(IDM_COMPORT_COM9, OnComportCom9)
	ON_COMMAND(IDM_COMPORT_EXIT, OnComportExit)
	ON_COMMAND(IDM_HELP, OnHelp)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// Gestore dei messaggi CRobotLinkDlg
```

```
BOOL CRobotLinkDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
	 CString strAboutMenu;
	 strAboutMenu.LoadString(IDS_ABOUTBOX);
	 if (!strAboutMenu.IsEmpty())
	 {
		pSysMenu->AppendMenu(MF_SEPARATOR);
		pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
	 }
	}

	SetIcon(m_hIcon, TRUE);
	SetIcon(m_hIcon, FALSE);


	m_Progress_B_C.SetRange(0,80);
	m_Progress_B_L.SetRange(0,80);
	m_Progress_B_R.SetRange(0,80);
	m_Progress_F_C.SetRange(0,80);
	m_Progress_F_L.SetRange(0,80);
	m_Progress_F_R.SetRange(0,80);

	m_Progress_B_C.SetPos(40);
	m_Progress_B_L.SetPos(40);
	m_Progress_B_R.SetPos(40);
	m_Progress_F_C.SetPos(40);
	m_Progress_F_L.SetPos(40);
	m_Progress_F_R.SetPos(40);

	return TRUE;
}

void CRobotLinkDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
	 CAboutDlg dlgAbout;
	 dlgAbout.DoModal();
	}
	else
	{
	 CDialog::OnSysCommand(nID, lParam);
	}
}



void CRobotLinkDlg::OnPaint()
{
	if (IsIconic())
	{
	 CPaintDC dc(this); // device context per il disegno

	 SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

	 // Centra l'icona nel rettangolo dell'utente
```

```
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Disegno dell'icona
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}


HCURSOR CRobotLinkDlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}


BEGIN_EVENTSINK_MAP(CRobotLinkDlg, CDialog)
    //{{AFX_EVENTSINK_MAP(CRobotLinkDlg)
	ON_EVENT(CRobotLinkDlg, IDC_MSCOMM1, 1 /* OnComm */, OnOnCommMscomm1, VTS_NONE)
	ON_EVENT(CRobotLinkDlg, IDC_Servo_Control, -600 /* Click */, OnClickServoControl, VTS_NONE)
	//}}AFX_EVENTSINK_MAP
END_EVENTSINK_MAP()



void CRobotLinkDlg::OnCancel()
{
	if (m_seriale.GetPortOpen())
	  m_seriale.SetPortOpen(false);

	CDialog::OnCancel();
}

void CRobotLinkDlg::OnGoForward()
{
	m_Octagon.SetAction(m_Pulse_Slider.GetValue(),1,90);
}

void CRobotLinkDlg::OnRotateLeft()
{
	m_Octagon.SetAction(m_Pulse_Slider.GetValue(),0,90);
}

void CRobotLinkDlg::OnRotateRight()
{
	m_Octagon.SetAction(m_Pulse_Slider.GetValue(),3,90);
}

void CRobotLinkDlg::OnGoBackward()
{
	m_Octagon.SetAction(m_Pulse_Slider.GetValue(),2,90);
}

void CRobotLinkDlg::OnStopengine()
{
	m_Octagon.SetAction(0,1,90);
}
```

```
void CRobotLinkDlg::OnClickServoControl()
{
	m_Octagon.SetAction(m_Pulse_Slider.GetValue(),5,unsigned char((m_Servo_Control.GetValue()/180.0)*255));
}


/*//////////////////////////////////////
Inizio gestione voci di menu
//////////////////////////////////////*/


void CRobotLinkDlg::OnComportCom1()
{
	SetCom(1);
}

void CRobotLinkDlg::OnComportCom2()
{
	SetCom(2);
}

void CRobotLinkDlg::OnComportCom3()
{
	SetCom(3);
}

void CRobotLinkDlg::OnComportCom4()
{
	SetCom(4);
}

void CRobotLinkDlg::OnComportCom5()
{
	SetCom(5);
}

void CRobotLinkDlg::OnComportCom6()
{
	SetCom(6);
}

void CRobotLinkDlg::OnComportCom7()
{
	SetCom(7);
}

void CRobotLinkDlg::OnComportCom8()
{
	SetCom(8);
}

void CRobotLinkDlg::OnComportCom9()
{
	SetCom(9);
}

void CRobotLinkDlg::OnComportExit()
{
	if (m_seriale.GetPortOpen())
	  m_seriale.SetPortOpen(false);

	CDialog::OnCancel();
}
```