RELAZIONE TECNICA SUL NUOVO CONTROLLO DEL ROTATORE *BFOSC* E DELLA SLITTA A 2 ASSI PER LA CAMERA DI GUIDA DEL TELESCOPIO CASSINI

RAPPORTO TECNICO 10-2010-1

I.Bruni[1], R.Gualandi[1], G.Cosentino[2], G. Innocenti[1], T. Trombetti[3]

[1]*INAF – Osservatorio Astronomico di Bologna*
[2]*Dipartimento di Astronomia – Università di Bologna*
[3]*INAF-IASF Bologna, via P. Gobetti 101, 40129, Bologna, Italy*

1 Introduzione

La camera di guida del Telescopio di Loiano, tramite la quale si effettua un accurato inseguimento del Telescopio durante l' acquisizione dati, è meccanicamente sostenuta da una slitta mobile avente due assi tra loro ortogonali (XY), interfacciati a loro volta a due servomotori che muovono la camera alternativamente nelle due direzioni per una più performante ricerca delle stelle di guida. Inoltre, un terzo servomotore (Z) è stato collegato alla flangia affinchè il *BFOSC* sia libero di ruotare attorno all'asse ottico del Telescopio.
La gestione dei tre assi fa capo ad un controllore d'uso commerciale. In questa relazione viene descritto il funzionamento dell'elettronica e del software, sviluppato dagli autori in $VB6$, che permette all'utente un controllo globale sulla strumentazione accessoria di piano focale. Questa nuova configurazione consente di ottimizzare i tempi tecnici necessari alla ricerca di una stella di guida, alla rotazione del *BFOSC* durante le osservazioni polarimetriche e, generalmente, in tutti quei casi in cui si necessita il posizionamento accurato in fenditura di due sorgenti vicine (10´ al massimo).

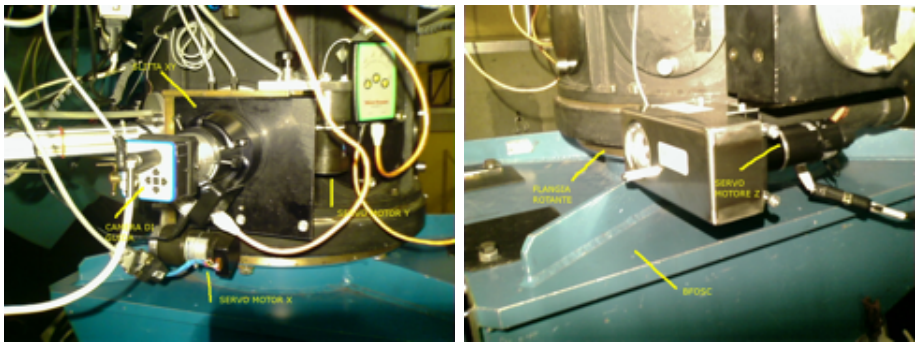


Figure 1: Foto a sinistra: slitta XY e camera di guida. Foto a destra: servomotore della rotazione del BFOSC (asse Z).

2 LA MECCANICA

La meccanica relativa alla guida automatica XY (foto a sinistra, Fig. 1) monta due servomotori *Portescap MGM9532* con motoriduttore ed encoder incrementale sull'asse, mentre sulla flangia di rotazione del *BFOSC*, alloggiato su un supporto appositamente progettato dalla ditta *T.S.M* di Loiano, è installato un servomotore *Maxon RE*40 con motoriduttore *GP*42*C* (in APPENDICE 1 sono allegate le specifiche tecniche) ed encoder incrementale sull'asse. Nella tabella sottostante sono riportate alcune tra le principali caratteristiche del suddetto materiale:

Servomotore	**Portescap MGM9532** (non più in produzione)
Tensione d'esercizio	24 V DC
Potenza Nominale	20 W
Coppia d'uscita	24.5 Ncm
Rapporto d'uscita	19.7:1

Servomotore	**Maxon RE40** (Order number 148867)
Tensione d'esercizio	24 V DC
Potenza Nominale	150 W
Velocità massima	8200 rpm
Corrente massima	6 A
Giunto di sicurezza	Mayr, EAS-SMARTIC diam.=12mm a morsetto
Puleggia per cinghia	Chiaravalli, T5, Z=42
Cinghia	Chiaravalli, Poliuretano filo e acciaio, T5 larghezza=16mm, Sv=390mm
Encoder	Maxon Held 5540 A11 0652A
Sensore di Prossimità	BDC, induttivo M5 N.C. PNP uscita a cavo

Il movimento degli assi XY avviene mediante due viti senza fine a cui sono interfacciati i servomotori Portescap. Il moto viene trasferito con coppie di pulegge e cinghie dentate (codice $RS474-4984$) in rapporto $1:1$. Inoltre sono montati 4 microinterruttori (codice $RS301-2270$, 2 per ogni asse) per impedire rotture meccaniche dovute ai problemi dei fine corsa che, se raggiunti, interrompono immediatamente le alimentazioni. Non esistono invece sensori di fine corsa sull'asse Z (foto a destra, Fig. 1). Lo "zero" della scala angolare è un parametro libero del software. Tuttavia il mozzo della puleggia solidale con il rotatore, attraverso un sistema di frizionamento, avverte gli eventuali sforzi meccanici dovuti essenzialmente a contatti accidentali del *BFOSC* con ostacoli perimetrali. In queste eventualità la frizione entra in gioco evitando danni strutturali al motore ed alla meccanica di rotazione. L'accoppiamento di pulegge tra il motore e la vite senza fine che ruota il *BFOSC* è in rapporto $2:1$ ed è garantito da una cinghia dentata. Con questa demoltiplica, la rotazione del *BFOSC* è funzionale anche nelle posizioni meccanicamente più svantaggiate come quelle a declinazioni piuttosto basse ($< 0°$).

3 L'ELETTRONICA

Il controllo dei tre servomotori è gestito dal dispositivo *PIC-SERVO SC* motion control board della ditta *JrKerr* (www.jrkerr.com). Lo schema sottostante (Fig. 2) descrive la struttura dei collegamenti tra i servomotori ed il PC di controllo.

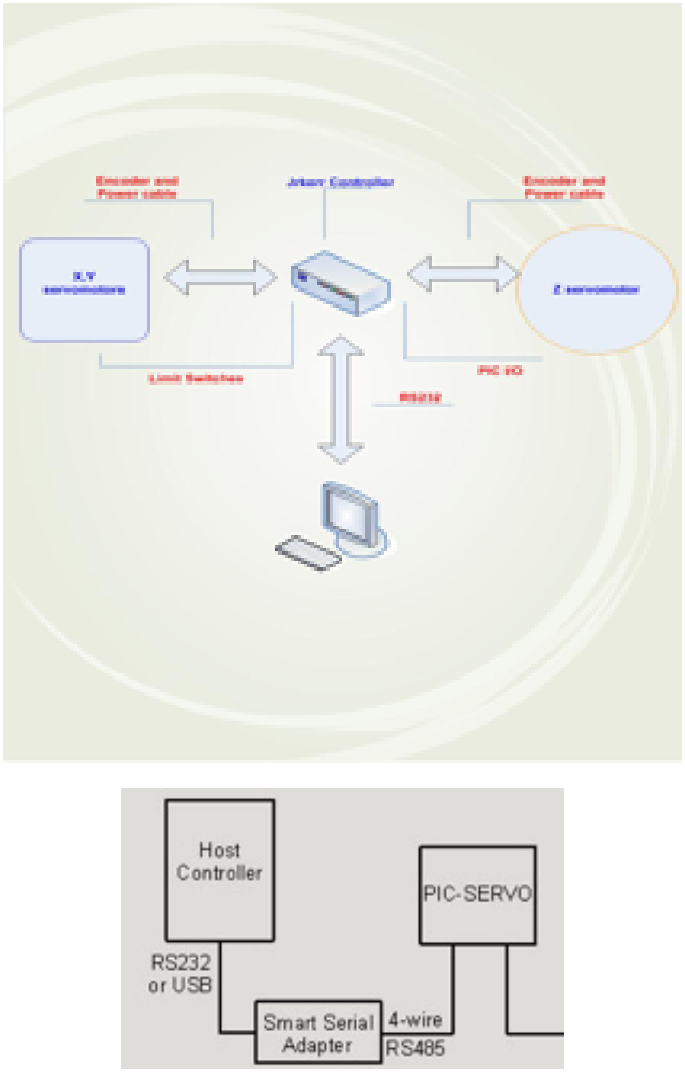


Figure 2: Schematizzazione dei collegamenti tra i servomotori ed il PC di controllo.

Il sistema è costituito modularmente da 5 elementi:
- 3 servo controllori *PIC-SERVO SC*,
- 1 adattatore di comunicazione seriale $Z323 - 485(SSA - 485)$,
- 1 interfaccia di *I/O* digitale *PIC-IO*.

Il PC (Host controller), su cui è installato il software di controllo, comunica attraverso una porta seriale *RS*232 (USB).
I moduli *PIC-SERVO* sono tra loro connessi in cascata (fino ad un massimo di 32 elementi) mediante una linea *RS*485 costituita da 4 fili (full duplex). Il protocollo di trasmissione è impostato a 19200*bps*, 8bit di dati, 1bit di start, 1bit di stop e nessuna parità. La comunicazione tra il PC ed i moduli è strettamente di tipo master/slave (l'host trasmette un comando e riceve lo "status" dai moduli) . Come mostrato in Fig. 3, la trasmissione dei pac-

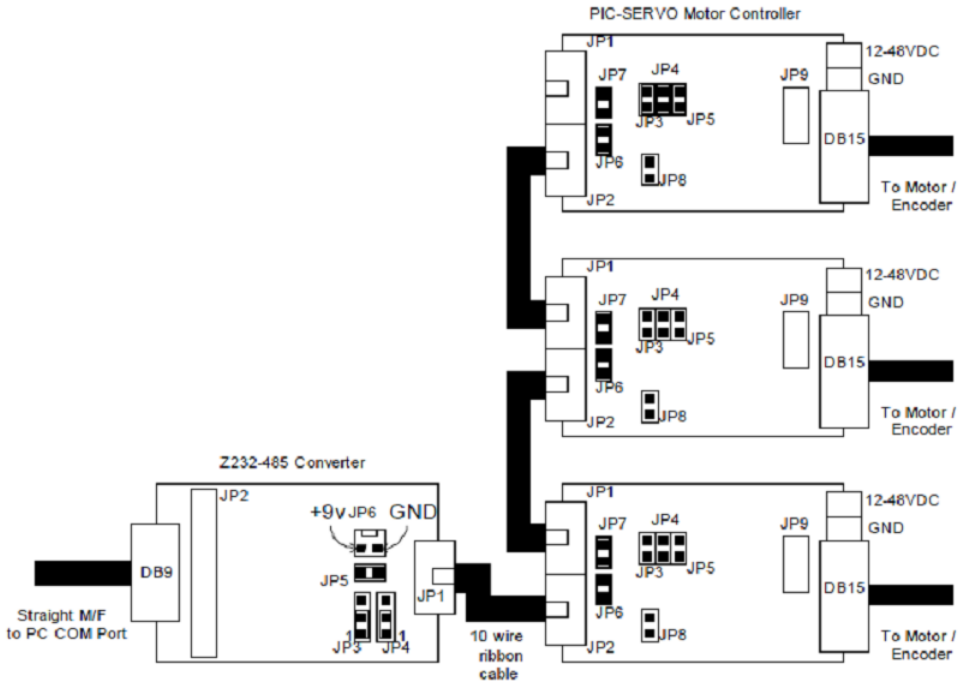


Figure 3: Schematizzazione dei collegamenti tra i singoli moduli e l'dattatore seriale $RS232-RS485$.

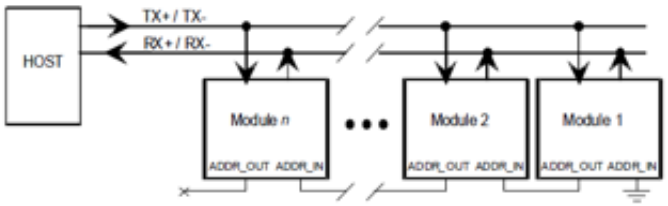


Figure 4: Schematizzazione del bus di trasmissione e ricezione dati.

chetti dati avviene su una linea dedicata, diversa quindi da quella di ricezione. Allo scopo di evitare sovrapposizioni di informazioni sulla comune linea di ricezione, i moduli sono pilotati da segnali che ordinatamente disabilitano ed abilitano le singole trasmissioni *(TX_ENABLE output)*.
L'host spedisce pacchetti dati così strutturati:

• Header byte: è l'inizio della trasmissione dei dati. I moduli della catena non eseguiranno nessun comando se non dopo aver ricevuto il byte di Header.
• Module address byte: è l'indirizzo del modulo.
• Commad byte: è il byte del comando.
• Additional data byte: contiene dati specifici che potrebbero essere richiesti da un particolare comando.
• Chechsum byte: è la somma del Module address e del Additional data byte.

Dopo che un modulo riceve il pacchetto dei comandi, viene verificato che l'indirizzo corrisponde a quello del modulo richiesto e, se il Chechsum byte non dà errori, il comando viene eseguito insieme alla trasmissione verso l'host del pacchetto di "status".
Quest'ultimo è così costituito:

• Status byte: contiene informazioni sullo stato del modulo, incluso lo stato del byte di chechsum del pacchetto dati arrivati dall'host.
• Additional status byte: contiene informazioni addizionali e programmabili dall'utente, come ad esempio il tipo di modulo e la velocità di un motore ad esso collegato.
• Chechsum byte: è la somma di tutti i byte.

Il microcontrollore, cuore del sistema, è il $PIC18F2331$ a 28 pin (www.microchip.com) (Fig. 5) temporizzato con un quarzo da $10MHz$.

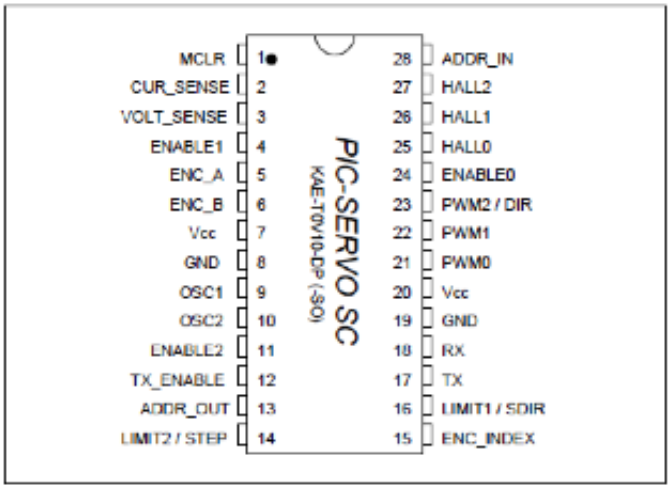


Figure 5: Microcontrollore PIC-SERVO SC.

Questo controlla i servo-motori in DC (48V max , 3A max) con encoder incrementali, include ingressi per i segnali di fine corsa (limit switch) ed ingressi per la protezione da assorbimenti eccessivi di corrente.
I contatori per velocità ed accelerazioni sono a 32 bit. Tutti i parametri operativi definiti dall'utente sono memorizzati in una *EPROM* interna. Di seguito sono riportate alcune caratteristiche:

Servo rate:	1953.125 Hz
Serial baud rate:	9,600 - 230,400 baud
	(faster rates may be possible but are untested)
PWM frequency:	19,531.25 Hz (fixed)
PWM resolution:	10 bit
Encoder counting rate:	2.5 MHz (max.)
Max step pulse rate:	100 KHz (see Section 4.4.6)
Max Command rate:	1000 Hz max. (approx.)

Gli encoder incrementali a cui può interfacciarsi il PIC-SERVO SC sono standard a due canali AB (quadratura) come mostrato in Fig. 7.

L'encoder emette 2 onde quadre con duty-cycle pari al 50% e sfasate di $\pm 90°$ dipendentemente dalla direzione di rotazione dell'asse. Un contatore a 32 bit misura la posizione corrente del motore, parametro che viene trasmesso al pacchetto di "status" proveniente dai

Pin	Symbol	Description
1	MCLR	Reset pin, active low. Connects directly to Vcc for automatic reset on power-up. When the ***PIC-SERVO SC*** is operated in stand-alone mode for Step & Direction systems, this pin can be used to enable/disable the servo system.
2	CUR_SENSE	Analog input for current sensing or for use as a general analog input. (0 - +5v). (See Section 4.7)
3	VOLT_SENSE	Analog input for sensing the motor supply voltage (connected through a voltage divider resistor network). It is used to detect undervoltage and overvoltage conditions. The input voltage must be in the range of 0.9v to 4.5v for normal operation. (See Section 4.7)
4	ENABLE1	Active HIGH output for enabling half-bridge #1 when in 3-Phase mode
5	ENC_A	Input pin connects to Channel A of your encoder. (Input has digital noise filter)
6	ENC_B	Input pin connects to Channel B of your encoder. (Input has digital noise filter)
7	Vcc	Supply voltage - connect to +5v DC.
8	GND	Ground connection.
9	OSC1	Connects directly to a 10 MHz clock source or to one side of a 10 MHz crystal. An internal phase-lock loop boosts the actual operating frequency to 40 MHz. See Microchip documentation for details of crystal connections.
10	OSC2	Connects to the other side of a 10 MHz crystal. N.C. if a clock source is used.
11	ENABLE2	Active HIGH output for enabling half-bridge #2 when in 3-Phase mode
12	TX_ENABLE	This output can be connected to the enable pin of an RS485 driver to enable output over a shared response line. Not used if a point-to-point serial connection (like RS232) is used.
13	ADDR_OUT	Output pin which powers up in the HIGH state, and is lowered when the ***PIC-SERVO***'s address is programmed. Normally connected to ADDR_IN of an adjacent controller. When in Step & Direction mode, this pin is raised on a servo fault condition and can be used as a fault detection flag.
14	LIMIT2 / STEP	REV limit switch input, except when in Step & Direction mode, it is the Step pulse input.
15	ENC_INDEX	Input pin connects to the Index pulse output of your encoder.
16	LIMIT1 / SDIR	FWD limit switch input, except when in Step & Direction mode, it is the Direction input. (0 = FWD, 1 = REV).
17	TX	Serial transmit output. Connects to the transmit input of an RS485 or an RS232 driver chip.
18	RX	Serial receive input. Connects to the receive output of an RS485 or an RS232 driver chip.
19	GND	Ground connection.
20	Vcc	Supply voltage - connect to +5v D.C.
21	PWM0	PWM output pin for PWM & Direction and antiphase PWM modes, and also used for driving half-bridge #0 when in 3-phase mode.
22	PWM1	PWM output for driving half-bridge #1 when in 3-Phase mode.
23	PWM2 / DIR	PWM output for driving half-bridge #2 when in 3-Phase mode. In PWM & Direction mode, it is used as the Direction bit (0 = FWD, 1 = REV)
24	ENABLE0	Active HIGH output for enabling the amplifier when in PWM & Direction and antiphase PWM modes, and enables half-bridge #0 when in 3-Phase mode.
25,26, 27	HALL0, HALL1, HALL2	Hall effect sensor input pins used for commutation when in 3-Phase mode. These pins have internal 20K pull-up resistors.
28	ADDR_IN	This pin must be pulled low to enable communications. Normally tied to the ADDR_OUT pin of the previous ***PIC-SERVO*** on the same RS485 network. This pin has an internal 20K pull-up resistor.

Figure 6: Pinout del Microcontrollore PIC-SERVO SC.

moduli. Il PIC-SERVO SC viene utilizzato in "modalità posizione" attraverso un loop chiuso, con periodo pari a $1953.125Hz$ (detto anche servo-tick), che controlla la posizione corrente dei motori e la confronta con quella desiderata. Successivamente un "filtro di controllo" minimizza l'errore di posizione. Il filtro usato dal PIC-SERVO SC è il PID (proporzionale-integrale-derivativo) che consiste nel seguente calcolo:

$$Output = K_p * PosError - K_d * (PosError - prevPosError) + K_i * IntegralError$$

K_p, K_d, K_i sono parametri (16 bit) che il programmatore, nel codice, deve ottimizzare in funzione del motore utilizzato. In APPENDICE 2 ed APPENDICE 3 sono elencate le piedinature di controllo dei motori, degli encoder e le configurazioni dei ponticelli a bordo delle schede.

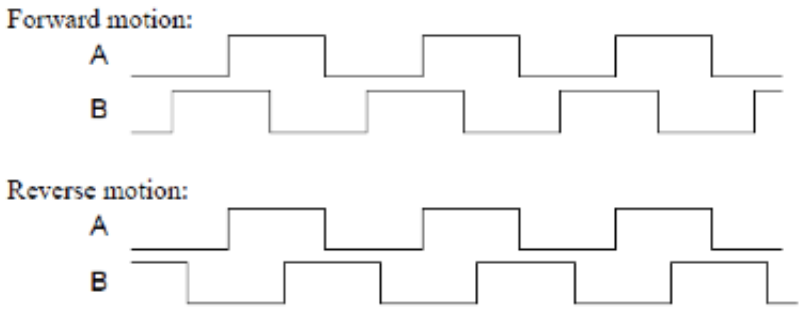


Figure 7: Diagramma degli impulsi provenienti dagli encoder relativi.

4 IL SOFTWARE

Il software "XY&Z *BFOSC*" è stato scritto in VB6 (in ambiente Windows), una delle piattaforme di sviluppo compatibili con i drivers rilasciati dalla casa. Per comunicare con il $PIC-SERVOSC$ si deve accedere alle funzioni contenute nella libreria $NMCLIB04.DLL$. Prima di ogni comando, è necessaria la connessione al dispositivo e l'inizializzazione della rete di moduli collegati (allo stesso modo è doveroso eseguire lo "shutdown" della rete prima di disconnettersi dall'hardware). In Fig. 9 è mostrata l'interfaccia utente. Tramite un messaggio, all'avvio il programma notifica all'utente di verificare che l'angolo di posizionamento del *BFOSC* sia pari a 270° (Fig. 8).

Figure 8: Messaggio di notifica iniziale del software che gestisce la rotazione del *BFOSC*.

Infatti, dal momento che nel sistema di rotazione non sono attualmente installati interruttori di fine corsa che avvertano dell'avvenuto raggiungimento dello zero, la posizione deve essere impostata manualmente ad ogni avvio.

L'interfaccia è divisa in 3 sezioni:

• il "Guide Star Finder Control", che contiene i comandi per il controllo manuale dello spostamento lungo gli assi a diverse velocità, oppure avvia la scansione automatica del campo di vista.

• il "Rotator *BFOSC* Control", dove sono incluse le operazioni manuali ed automatiche di rotazione del *BFOSC*.

• "M.P. Track" che fa muovere i 2 assi in modo coordinato ed a velocità programmate, per inseguire oggetti che seguono traiettorie con moti non siderali (pianetini e comete) e di cui si conoscono i moti propri. Essendo ancora in fase sperimentale non verrà trattato.

Tramite "COM Link" (punto 1 della Fig. 9) si avvia la connessione dell'adattatore di comunicazione seriale. Nel "Link Log" viene visualizzato il messaggio di avvenuta connessione con la porta COM selezionata. La successiva inizializzazione dei 2 motori X e Y della slitta si può effettuare attraverso "Initialize X,Y,Z Motors" (punto 2), fin quando intervengono gli switch di fine corsa a fermarne il moto ed il cui stato è monitorato nel log "Limit Sw. X,Y". L'azionamento dei 2 fine corsa azzera i contatori a 32bit collegati agli encoder.

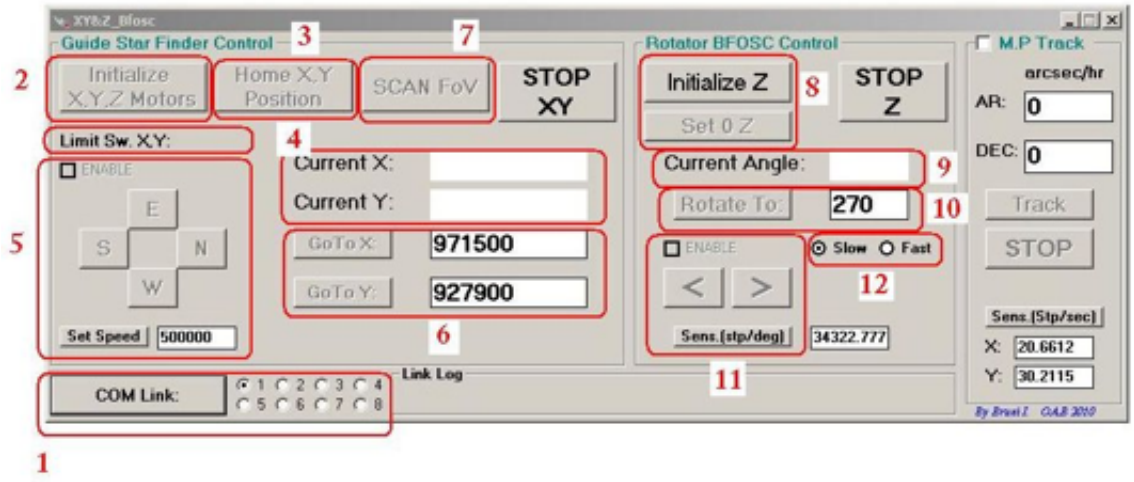


Figure 9: Intrerfaccia di guida, rotazione e tracking.

Successivamente, "Home X,Y Position" (punto 3), aziona nuovamente i motori portandoli a 100000 step dalla posizione di fine corsa. Quest'ultima sarà la posizione di partenza per le scansioni del campo.
La posizione corrente è visualizzata in "Current X" e "Current Y" (punto 4) . Nel punto 6, con "GoTo.." è possibile spostare la slitta in una posizione qualsiasi. I numeri di posizione visualizzati non corrispondono a grandezze calibrate, sono valori relativi. Il pulsante "SCAN FoV" (punto 7) ha la funzione di attivare una scansione a velocità controllata del campo di guida per la ricerca di stelle sufficientemente brillanti. Inoltre è possibile avere il controllo manuale degli spostamenti nelle 4 direzioni (punto 5). Per quanto riguarda il controllo della flangia rotante, con "Initalize Z" e "Set 0 Z" (punto 8) si inizializza il controllore del terzo asse e si azzera (via sofware) il contatore dell'angolo di partenza. L'angolo corrente viene costantemente visualizzato (punto 9) ed inoltre è possibile gestire le rotazioni in modo manuale ed automatico impostando semplicemente i gradi desiderati. Per ragioni di sicurezza, gli interruttori di fine corsa non vengono sollecitati se non nella prima operazione di azzeramento dei valori degli encoder. Nominalmente le escursioni di movimento sono comprese tra $99800 \div 2230000$ step in X e $99800 \div 2020000$ step in Y. I valori digitali che ritornano quando gli interruttori di fine corsa si trovano nei possibili stati sono: 57128 (fc 1), 25128 (nc), 89128 (fc 2). Il campo inquadrato uscente dalla faccia laterale dell'ottagono del Telescopio è notevolmente più grande del campo in cielo visto dal chip del CCD di guida. Per questa ragione, col pulsante "SCAN FoV" vengono azionati a bassa velocità (cosi da veder la traccia delle stelle) alternativamente gli assi X ed Y per scansionare (come schematizzato nella figura sottostante) l'intero campo a disposizione per la ricerca di stelle guida.

Con questo lavoro, seppur preliminare per certi aspetti, si è fortemente ottimizzata una delle attività più penalizzanti in termini di tempo sottratto alle misure scientifiche: la ricerca delle stelle di guida. Il software e l'hardware si son dimostrati, in più di 6 mesi di test, effettivamente stabili. Il passo successivo sarà quello di implementare una ricerca di stelle brillanti tramite query sul database del catalogo GSC2, portando ad un ulteriore e sensibile riduzione dei tempi morti. Per quanto riguarda la rotazione del *BFOSC*, c'è da segnalare che il Telescopio è limitato nei movimenti da alcune posizioni critiche perché può urtare contro elementi strutturali quali i piloni della montatura. La possibilità di urti aumenta quando il *BFOSC* non si trova in posizione di lavoro a $270^{/circ}$ e quindi si renderà necessario creare

Figure 10: Schema della modalità di scansione del campo di vista.

una mappatura analitica delle posizioni proibite installando una serie di sensori affinché il software possa lavorare in sicurezza.

A APPENDICE 1

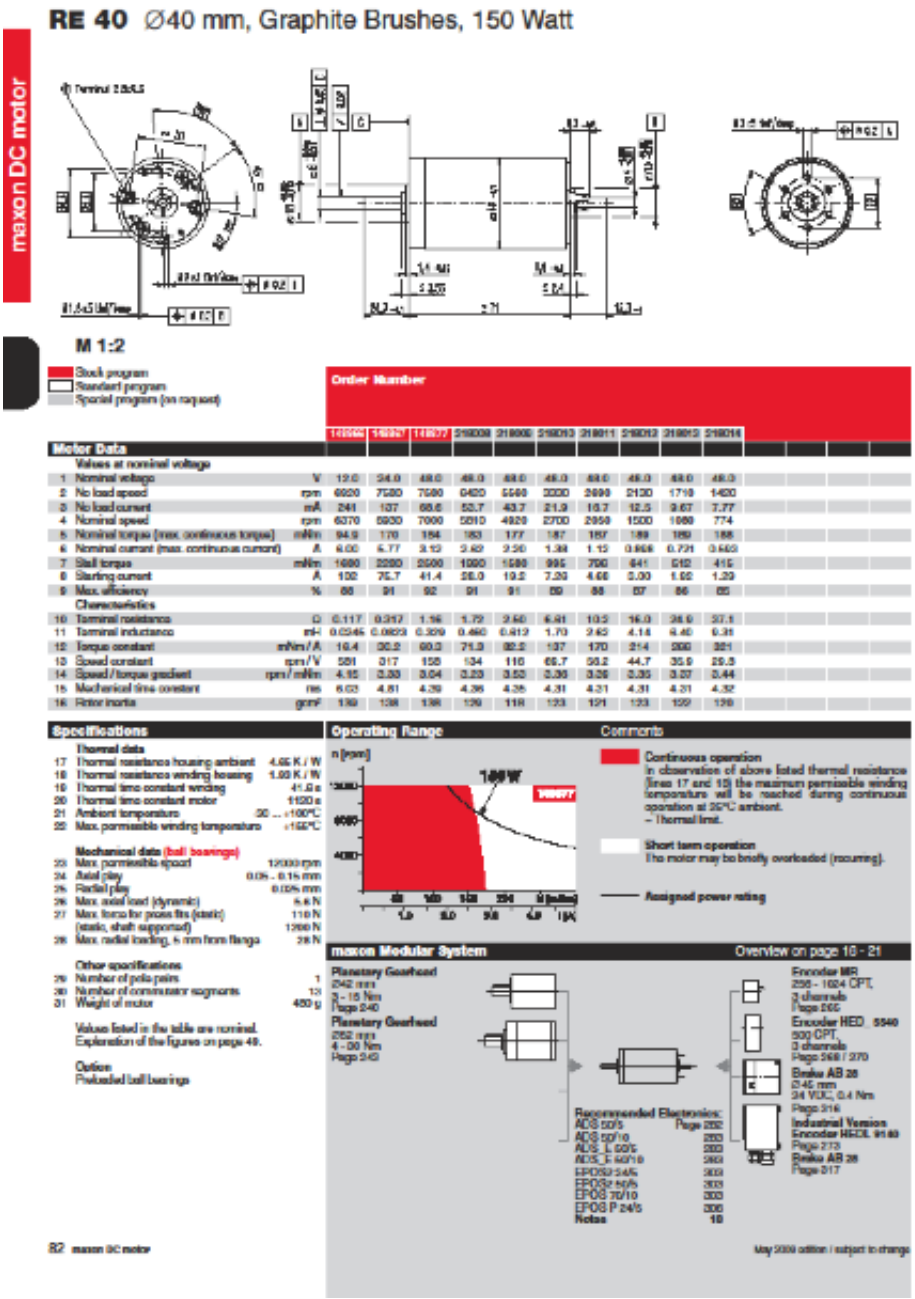
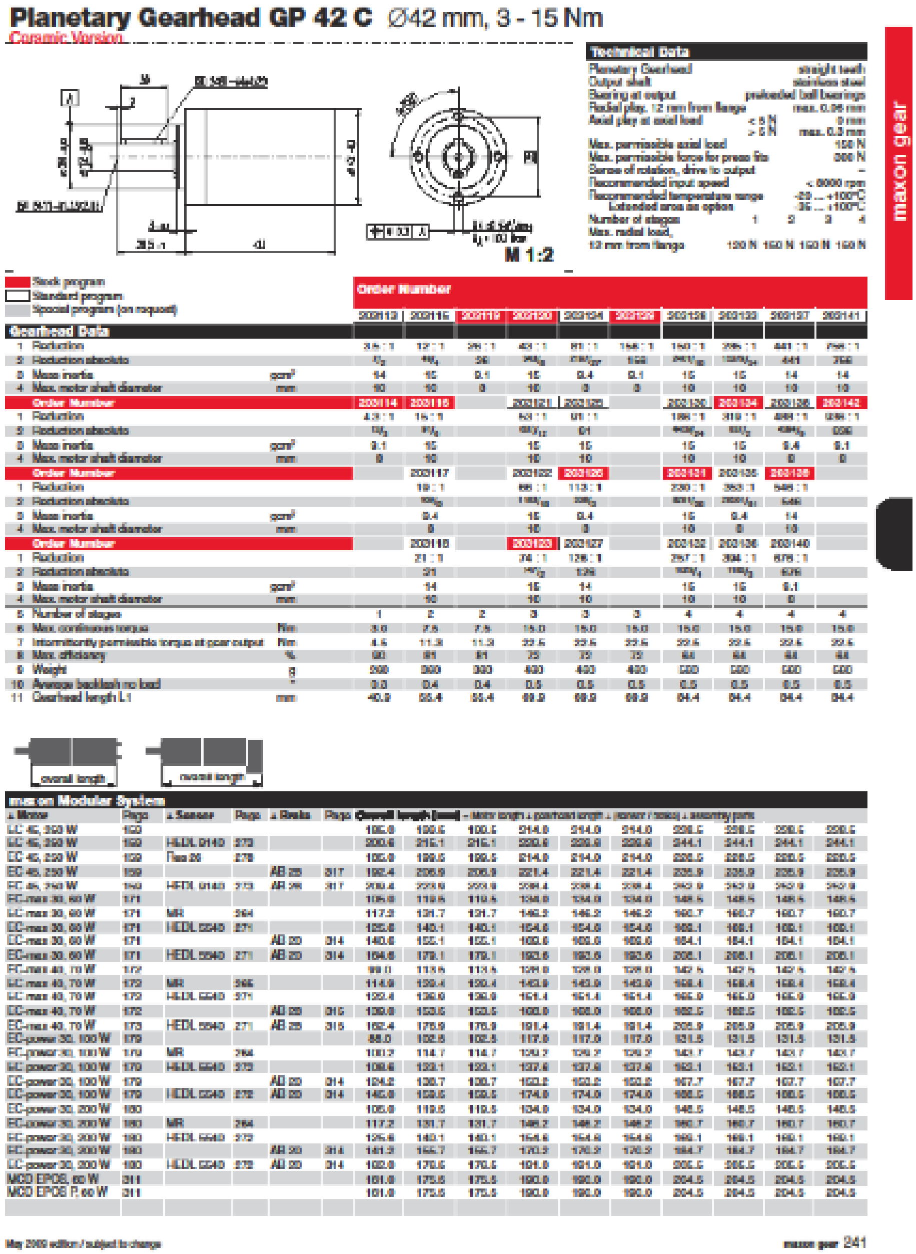


Figure 11: .



Planetary Gearhead GP 42 C Ø42 mm, 3 - 15 Nm
Technical Data
maxon gear
M 1:2
Order Number
maxon gear 241

Figure 12: .

B APPENDICE 2

Motor Connector P1 - DB15 Male

Pin	*Definition*
1	Motor Output (M+)
2	Motor Output (M+)
3	LED power - pulled up to 5v with a 330 ohm resistor (for use with opto-interrupt type switches)
4	Limit Switch 1 (pulled up to 5v with a 4.7k resistor)
5	Encoder Channel A (pulled up to 5v with a 4.7k resistor)
6	Encoder Channel B (pulled up to 5v with a 4.7k resistor)
7	Limit Switch 2 (pulled up to 5v with a 4.7k resistor)
8	Encoder Index (pulled up to 5v with a 4.7k resistor)
9	Motor Output (M-)
10	Motor Output (M-)
11	GND
12	GND
13	GND (supplied to encoder)
14	+5v (supplied to encoder)
15	GND

Network Connectors JP1, JP2 - 10 Pin Flat Ribbon IDC Sockets

Pin	*Definition*
1	***PIC-SERVO*** RCV+
2	***PIC-SERVO*** RCV-
3	***PIC-SERVO*** XMT+
4	***PIC-SERVO*** XMT-
5	***PIC-SERVO*** ADDR_IN on JP1, ADDR_OUT on JP2
6	GND
7	Logic power (7.5 - 12vdc)
8	GND
9	Logic power (7.5 - 12vdc)
10	GND

External Amplifier Connector JP9 - 8 Pin Single Row Locking Header

Pin	*Definition*
1	PWM output - 20 KHz square wave magnitude signal or Antiphase signal
2	Direction output (not needed if in Antiphase mode)
3	Amplifier Enable output (active high)
4	Limit2 input (same as P1, pin 7) *or* Step input if in Step & Direction mode
5	Limit1 input (same as P1, pin 4) *or* Direction input if in Step & Direction mode
6	MCLR input - can be used to enable/disable controller if in Step & Direction mode - pull low to disable controller (pulled up to 5v with a 4.7k resistor)
7	ADDR_OUT output - can be used as a servo fault indicator if in Step & Direction mode (signal goes high when servo is disabled)
8	GND

Motor Power Connector P2 - Screw Terminals

Pin	*Definition*
1	Motor Power 12 - 48vdc (at edge of board)
2	Motor Power Ground (connected internally to logic ground)

JEFFREY KERR, LLC • *www.jrkerr.com*

Figure 13: .

Logic Power Connector JP8 - 2 Pin Locking Header
(Use only if logic power is **not** supplied via the network communications cable.)

Pin	*Definition*
1	7.5 - 12vdc (pin towards the lower edge of the board)
2	Ground (pin towards the center of the board)

2.2 Jumpers

PIC-SERVO Motor Control Board Jumpers:

Jumper	*Description*
JP3	Connects ADDR_IN to GND. Insert jumper for the last ***PIC-SERVO*** on the network (or if only 1 ***PIC-SERVO*** is used)
JP4, JP5	Enables termination resistors on RX and TX. Insert these jumpers for the last ***PIC-SERVO*** on the network (or if only 1 ***PIC-SERVO*** is used).
JP6,JP7	Logic power interconnection. Inserting JP6 connects logic power to network connector JP2. Inserting JP7 connects logic power to JP1. These are used to control the distribution of logic power over the network cables. Normally both these jumpers are installed.

Figure 14: .

C APPENDICE 3

Encoder Line Drivers

Technical Data

HEDL-550X/554X
HEDL-556X/557X
HEDL-560X/564X
HEDL-9000/9100/9200
HEDL-9040/9140
HEDL-9060/9160/9260
HEDL-9061/9161

Features

- **Available on Both Encoder Modules (HEDS-9000 Series) and Encoder Kit Housings (HEDS-5500 Series)**
- **Complementary Outputs**
- **Industry Standard Line Driver IC**
- **Single 5 V Supply**
- **Onboard Bypass Capacitor**
- **70°C and 100°C Versions Available**

Description

Line Drivers are available for the HEDS-55XX/56XX series and the HEDS-9000/9100/9200/9040/9140 series encoders. The line driver offers enhanced performance when the encoder is used in noisy environments, or when it is required to drive long distances.

The 70°C version utilizes an industry standard line driver IC (26LS31) which provides complementary outputs for each encoder channel. The 100°C version utilizes an industry standard line driver IC, 9GC31, which provides complementary outputs for each encoder channel. Thus, the output of the line driver encoder is A, $\overline{A}$, B, $\overline{B}$ and I, $\overline{I}$ for three channel versions. Suggested line receivers are 26LS32 and 9GL32I.

For additional information, please refer to:
HEDS-5500/5540/5600/5640 data sheet,
HEDS-9000/9100/9200 data sheets,
HEDS-9000 series extended resolution data sheet, and
9GL32I data sheet.

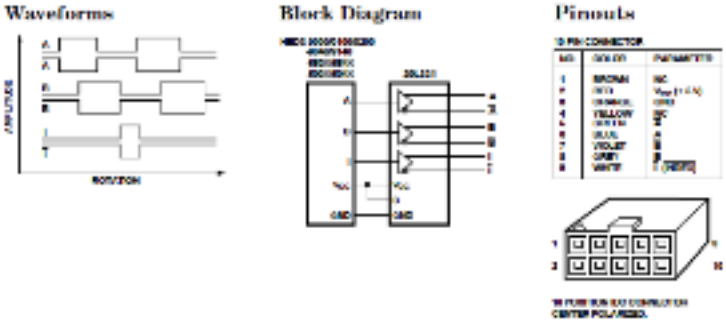
Device Characteristics

Parameter	Characteristic	Notes
Termination	10 conductor ribbon cable with 10 position IDC Berg connector	See pinout
Electrical Outputs	Complementary outputs: A, $\overline{A}$, B, $\overline{B}$, I, $\overline{I}$	I and $\overline{I}$ available only on three channel encoders
Line Driver Components	26LS31 line driver IC, decoupling capacitor on PC board.	
Operating Temperature Range:	0°C to 70°C	70°C Series
	0°C to 100°C	100°C Series
Storage Temperature:	-40°C to 70°C	70°C Series
	-40°C to 100°C	100°C Series

ESD WARNING: NORMAL HANDLING PRECAUTIONS SHOULD BE TAKEN TO AVOID STATIC DISCHARGE.

2-72 5965-5878E

Figure 15: .



Note: I/I only available on three channel encoders.

Line Driver Base Parts Available:

70°C Line Driver Base Part	100°C Line Driver Base Part	Channels	Refer to the following encoder data sheet for additional information and option codes (XXX = resolution and/or shaft size)
HEDL-5600#XXX	HEDL-5668#XXX	A, B	HEDS-5600#XXX
HEDL-5605#XXX	HEDL-5669#XXX	A, B	HEDS-5605#XXX
HEDL-5640#XXX	HEDL-5670#XXX	A, B, I	HEDS-5640#XXX
HEDL-5645#XXX	HEDL-5671#XXX	A, B, I	HEDS-5645#XXX
HEDL-9400#XXX	HEDL-9072#XXX	A, B	HEDS-9400#XXX
HEDL-9405#XXX	HEDL-9073#XXX	A, B	HEDS-9405#XXX
HEDL-9440#XXX	HEDL-9074#XXX	A, B, I	HEDS-9440#XXX
HEDL-9445#XXX	HEDL-9075#XXX	A, B, I	HEDS-9445#XXX
HEDL-9100#XXX	HEDL-9060#XXX	A, B	HEDS-9100#XXX
HEDL-9040#XXX	HEDL-9061#XXX	A, B, I	HEDS-9040#XXX
HEDL-9140#XXX	HEDL-9161#XXX	A, B, I	HEDS-9140#XXX
HEDL-9100#XXX	HEDL-9060#XXX	A, B	HEDS-9100#XXX

Ordering Information:

For option code selection, refer to the data sheet for the corresponding "HEDS" part number (see right column).

Figure 16: .

D APPENDICE 4

Pinout DB15 (controller)	Pinout DB9 (motor)
1	
2	1 (+mot)
3	
4	
5	4 (A)
6	8 (B)
7	
8	3 (index)
9	
10	6 (-mot)
11	
12	7 (gnd)
13	9 (gnd)
14	5 (+5V)
15	

Figure 17: .

E APPENDICE 5

Microsoft Visual Basic .net Users

a. Add the file NMCDEFINES.VB to your project

b. Copy the file NMCDEFINES.VB into the same folder as your other source code files.

For all three programming environments, you will have to place a copy of the NMCLIB04.DLL in the Windows system folder or else in the same folder as the executable you are creating. Copies of the .H or .VB header files needed for your environment can be found in the example programs on our web page *www.jrkerr.com/software.html*.

2. Network Initialization and Shutdown Functions

a. Use the following code segment to initialize the network of controllers:

```
int nummod;

ErrorPrinting(0);    //suppress printing of internal error messages

//Initialize NMC controllers on COM1: using 19200 Baud
nummod = NmcInit("COM1:", 19200);
```

`nummod` should be equal to the number of controller modules connected to the network.

b. Use the following code to read the type of module at a given address:

```
unsigned char modtype;
unsigned char addr;        //module address (1 - 32)
unsigned char stat_items;  //specifies which data should be returned

addr = 1;
stat_items = SEND_ID;

NmcReadStatus(addr, stat_items); //Use NmcReadStatus to read ID from
                                 //  controller
modtype = NmcGetModType(addr);   //retrieve the module type
```

`modtype` should equal 0 for a ***PIC-SERVO*** controller, 2 for a ***PIC-I/O*** and 3 for a ***PIC-STEP***.

c. Reset controllers to power-up state and shutdown the network:

```
//Reset the network of controllers - 0xFF resets all controllers
NmcHardReset(0xFF);

//Clean-up and close the COM port
NmcShutdown();
```

3. *PIC-SERVO* Motor Initialization

a. Use the following code segment to set the servo gains, enable the amplifier, and start the motor servoing to its current position:

```
unsigned char addr;        //module address
unsigned char ol, sr, dc;  //output limit, servo rate, deadband comp.
short int kp, kd, ki, il;  //position gain, derivative gain, intergral
                           //  gain, integration limit
unsigned char sm;          //step rate multiplier
unsigned char mode;        //specifies stopping options

addr = 1;            //set address for module 1
kp = 100;            //following gains are a good starting point
kd = 1000;           // for most motors and encoders
ki = 0;
il = 0;
ol = 255;
sr = 1;
dc = 0
mode = AMP_ENABLE | STOP_ABRUPT;

ServoSetGain2(addr, kp, kd, ki, il, ol, cl, el, sr, dc, sm);
ServoStopMotor(addr, mode);
```

The `AMP_ENABLE` flag should always be included in the `mode` byte when issuing and subsequent stop commands, unless you are actually wishing to disable the amplifier. (Note: If your amplifier requires the `AMP_ENABLE` bit to be low for normal operation, you should NOT include it.) The `STOP_ABRUPT` flag tells the ***PIC-SERVO*** to start servoing to its current position.

b. The following function can be used to reset the motor encoder position counter to zero:

```
ServoResetPos(addr);
```

c. If you are using the ***PIC-SERVO SC***, you have the option of setting the output mode and limit switch options. The following example sets the output mode for 3-phase commutation and enables the limit switch protection to stop the motor abruptly when a limit switch is hit. Note that the `ServoSetIoCtrl()` function should be called before enabling the amplifier as in the example above.

```
unsigned char addr;        //module address
unsigned char mode;        //specifies I/O Control options

addr = 1;                  //set address for module 1
mode = THREE_PHASE | LIMSTOP_ABRUPT;

ServoSetIoCtrl(addr, mode);
```

4. *PIC-SERVO* Motion Commands

a. Move to a goal position with acceleration ramping:

```
unsigned char addr;        //module address
unsigned char mode;        //motion options
long     pos, vel, acc;    //position, velocity & acceleration
unsigned char pwm;         //raw PWM value - unused

addr = 1;
mode = LOAD_POS | LOAD_VEL | LOAD_ACC | ENABLE_SERVO | START_NOW;
pos = 5000;          //move to position 5000 encoder counts
vel = 100000;        //100,000 ~= 3000 encoder counts per second
acc = 100;           //acceleration value
pwm = 0;             //pwm value is not used

ServoLoadTraj(addr, mode, pos, vel, acc, pwm);
```

Note that the position is an absolute (not relative) value referenced to the encoder's zero position. You should always allow one position move to complete before attempting to move to a new position (except for the ***PIC-SERVO SC***).

b. Detect when a motion is complete:

```
unsigned char addr;        //module address
unsigned char current_status_byte;
BOOL move_complete;

NmcNoOp(addr);       //update the status byte data
current_status_byte = NmcGetStat(addr);
move_complete = current_status_byte & MOVE_DONE;
```

Note that the `MOVE_DONE` bit is part of the status byte.

c. Move at a constant velocity with acceleration ramping:

```
unsigned char addr;        //module address
unsigned char mode;        //motion options
long     pos, vel, acc;    //position, velocity & acceleration
unsigned char pwm;         //raw PWM value - unused

addr = 1;
```

```
mode = LOAD_VEL | LOAD_ACC | ENABLE_SERVO | VEL_MODE | START_NOW;
pos = 0;              //position not used
vel = 100000;         //100,000 ~= 3000 encoder counts per second
acc = 100;            //acceleration value
pwm = 0;              //pwm value is not used

ServoLoadTraj(addr, mode, pos, vel, acc, pwm);
```

Note that the velocity value should always be positive. To move in the reverse direction, you should include the mode flag `REVERSE` when setting the `mode` byte.

d. Start a motion and decelerate to a stop automatically when a limit switch is hit:

```
unsigned char addr;         //module address
unsigned char mode;         //motion options
long     pos, vel, acc;    //position, velocity & acceleration
unsigned char pwm;          //raw PWM value - unused
unsigned char homing_mode; //homing command options
BOOL still_homing;          //flag for checking homing

//First, use the ServoSetHoming function to activate the homing function

addr = 1;
homing_mode = ON_LIMIT1 | HOME_STOP_SMOOTH;  //stop smoothly when limit
                                             // switch 1 is activated
ServoSetHoming(addr, homing_mode);


//Next, execute a motion which will move the motor towards the limit switch

mode = LOAD_POS | LOAD_VEL | LOAD_ACC | ENABLE_SERVO | START_NOW;
pos = 100000;         //move to position beyond the limit switch
vel = 100000;         //100,000 ~= 3000 encoder counts per second
acc = 100;            //acceleration value
pwm = 0;              //pwm value is not used

ServoLoadTraj(addr, mode, pos, vel, acc, pwm);

//Lastly, poll the HOME_IN_PROG bit to determine when the
//switch has been hit

do
   NmcNoOp(addr);           //update the status byte
   still_homing = NmcGetStat(addr) & HOME_IN_PROG;
while (still_homing);
```

e. Stop a motor with a deceleration ramp:

```
unsigned char addr;         //module address
unsigned char mode;         //specifies stopping options

mode = AMP_ENABLE | STOP_SMOOTH;
ServoStopMotor(addr, mode);
```

Note that you should use the procedure described in **4b** to determine when the motor has actually come to a stop. To stop abruptly, use `STOP_ABRUPT` instead of `STOP_SMOOTH`.

f. Turn the motor off and disable the amplifier:

```
unsigned char addr;         //module address
unsigned char mode;         //specifies stopping options

mode = MOTOR_OFF;
ServoStopMotor(addr, mode);
```

5. *PIC-SERVO* Reading Status Data

a. Read limit switches:

```
unsigned char addr;              //module address
unsigned char current_status;    //status byte value
BOOL switch1, switch2;           //boolean switch values

addr = 1;

NmcNoOp(addr);        //Limit switch values are in the status byte, so the
                      //  NoOp command can be used to update the status

current_status = NmcGetStat(addr);     //fetch the value of the status byte
switch1 = current_status & LIMIT1;
switch2= current_status & LIMIT2;
```

b. Read the position and velocity at the same time:

```
unsigned char addr;         //module address
long pos;                   //motor position in encoder counts
short int vel;              //motor vel. In encoder counts/servo tick
unsigned char stat_items;   //specify which data should be returned

stat_items = SEND_POS | SEND_VEL;  //specify both position and velocity
                                   //  should be read from contorller
NmcReadStatus(addr, stat_items);   //Read data from controllers

pos = ServoGetPos(addr);    //retrieve the position and velocity data
vel = ServoGetVel(addr);    //from the local internal data structure
```

Note that `NmcReadStatus()` does something very different from `NmcGetStat()`! You should also note that the velocity returned is only a 16 bit quantity, whereas the commanded velocity is 32 bits. The velocity returned should be approximately equal to the commanded velocity divided by 65,536 (2^{16}).