AST/RO Technical Memorandum #31
10 April 1998

New chopper interface documentation

Alberto D. Bolatto
Boston University Department of Astronomy
725 Commonwealth Ave., Boston, MA 02215

1 Overview

The new chopper interface consists of

- one Compumotor Dynaserv DM1045B brushless motor with 230VAC power input
- one Compumotor Dynaserv DM1045B paired 230VAC driver
- one Galil DMC730 3-axis controller

The driver operates in the torque mode, receiving a ± 10 V command signal from the third axis of the Galil controller directly proportional to the current applyed to the motor. Therefore all the motion control is done through the DMC730, the driver acting only as a transconductance amplifier.

The DMC730 comunicates with the host computer through the serial port (for the start/stop commands) and the digital input/output ports (while the chopper is running). It also reads the status from the driver and stops the motion in case of trouble (driver overload, driver off, etc) reporting a trouble byte using the serial line.

2 Driver setup

The driver is shipped in position mode. Setting up the driver in torque mode involves changing the jumper states into the driver case. Note that there is an error in the manual about how to set torque mode in the controller. The jumpers MODE and CALIB in the section JP1 should be respectively open and shorted (not open and open, as the manual says). Therefore the correct configuration is

Section JP1
MODE = open
CALIB = shorted
RATE1 = shorted
RATE2 = shorted

this results in torque mode with full encoder resolution (655,360 pulses per revolution). Section JP2 should be configured as detailed in the manual for torque mode, shorting the jumper 200.

Section JP2
I = open
P = open
100 = open
200 = shorted
PV = open
VE = open
TPRQ = shorted

3 Driver-controller interface

The following pins on the CN1 HONDA connector in the Dynaserv driver have been interfaced to the Z axis of the DMC730 controller as detailed:

Driver pin	Driver name	DMC730 pin	Cable color	Comments
13	A+	A+	Brown	encoder output
14	A-	A-	BBrown	encoder output
15	RDY+	IS5V	BBlue	DMC internal 5V isolated source
16	RDY-	IN2	BGrey	current flows when driver on
19	DIR-			short to Dynaserv pin 20 (DIR+)
23	SRVON-			short to Dynarserv pin 50 (AGND)
24	SRVON+	AEN	Orange	servo on signal
29	B-	B-	Green	encoder output
30	B+	B+	BGreen	encoder output
47	OVL	IN1	Grey	current flows when in overload
49	VIN	ACMD	BOrange	analog ± 10 command line
50	AGND	GND	Blue	analog grounding
		IDX+		short to GND
		IDX-		short to GND
		VIN		short to IS5V
		VRET		short to ICOM

Therefore the DMC730 is able to sense through two of its digital inputs if the driver is experiencing problems. More on this in the I/O Bus section.

The colors correspond to the conductors inside the calbox and in the cable between the back of the calbox and the CN1 connector in the Dynaserv driver. A "B" before the color indicates the conductor is mostly white with bands of that color.

4 Optical Switches

The motor has 3 NAIZ fiber optics reflective sensors attached to the motor encasing, which are activated by the tertiary mirror at certain positions: right (or reverse) limit switch (RLS), left (or forward) limit switch (FLS), and the home switch (HMS). Three UFZ1301 red led amplifiers, attached to the inside of the telescope, produce the light signal that is sent through three UFZRG1 coax fiber-optics sensors, each equiped with a UZFXMR1 pin-point focusing lens. The light is reflected by an extension of the primary mirror mount and captured back by the sensor, which sends it to the amplifiers that convert it into a digital voltage. The amplifiers are powered by the 24 V regulated power supply located insite the calbox controller box. The SCOM pin in the Z axis of the GALIL controller is also connected to the +24V pin, since the sensing output of the red led amplifiers is measured against their power supply. These limit switches are "direct", i.e., the voltage of the sensing output is high when the light beam is being reflected, and low otherwise. According to that, the limit switch logic inside the GALIL should not be reversed using the IL ,,1 command. The status of the limit switches can be interrogated using the command MG in the GALIL combined with the variables _RLZ, _FLZ and _HMZ (e.g., MG _RLZ). A result of 1 denotes an active switch, while 0 is for unactive. The connector cabling in the back of the calbox controller box is as follows

Switch	Motor conn.
Power Home	A
Sensing Home	B
Power RLS	C
Sensing RLS	D
Power FLS	E
Sensing FLS	F
Ground	G

The home reacquisition precision of the system has been measured to be better than 2 asec in the sky.

5 Proximity Switches (SUPERSEDED BY THE PREVIOUS SECTION)

The motor has 3 OMROM 2E2-NPN proximity switches attached to the motor encasing and activated by the tertiary mirror when it is at certain positions: two of them are limit switches and the third is a home switch. An excitation voltage of 24V is applyed between the red (positive) and black (negative) wires. The switch is then read between the red (positive) and white (negative) wires. The 24V are taken from the unregulated power supply inside the calbox controller box. The +24V pin is connected also to the SCOM pin on the limit switches. It is necessary to run IL command to invert the logical state of the limit switches inside the DMC, otherwise they appear naturally as active. The connector cabling is as follows

Switch cable	Motor conn. pin	Calbox control. cable
Red Home	A	Red
White Home	B	Green
Red RLS	C	Brown
White RLS	D	Orange
Red FLS	E	Yellow
White FLS	F	White
Ground	G	Black

6 I/O Bus

Some pins in the controller input and output ports are connected to one of the digital input/output cards on Royal. During the "running" phase of the chopper TRACKER controls two input bits (called END and CON) in port B of the I/O card, that are connected to the input port of the controller (currently bits 3 and 4, bits 1 and 2 are inputs from the driver). TRACKER also reads two bits of status (called REF and SRC) coming from the controller (currently bits 0 and 1 on the output port) in port A of the I/O card. These bits are respectively named ENDBIT, CONBIT, REFBIT and SRCBIT by the software running inside the GALIL.

While the chopper program is running in the DMC controller, a positive transition of the CON bit (from 0 to 1) sends the chopper to the source position. Correspondingly a negative transition (from 1 to 0) sends the chopper to the reference position. A zero in the END input while the chopper is at the source ends the chopping program and releases the CPU of the DMC. The END input is ignored if the tertiary is at the reference position.

The DMC controller signals the position of the tertiary mirror using the REF and SRC bits. While the tertiary is at the reference position, REF is 1 and SRC is 0. Correspondingly at the source position SRC is 1 and REF is 0. During movement, both SRC and REF are zero. If a problem arises in the driver and is detected by the DMC through the overload or driver ready bits both SRC and REF will be set to one and the result of the TI command (tell inputs) will be send by the DMC trough the serial port.

Problems with the driver are sensed by the GALIL through two input bits called RDY and OVL, connected to the READY and OVERLOAD outputs of the Dynaserv. These bits are monitored inmediatly after any movement command by the GALIL routine #CHKTRB. They allow the GALIL to sense trouble conditions such as the Dynaserv is not powered or the motor is being driven against one of the hard stops and consequently is overloaded.

The interface can be better appreciated by following the logic of the following C code (called move.c). This code emulates what TRACKER does relative to the chopper movement.

```
#include "../adelie/iodef.h"
#include "chop.h"
#include <stdlib.h>

main(ac,av)
int ac;
char *av[];
{
    int nrpt,ninter,curint,curpt,st,oldpos;
    float intime;
```

```
    intime=ac>1?atof(av[1]):1.0;
    nrpt=ac>2?atoi(av[2]):10;
    ninter=intime*100;
    curpt=0;
    curint=0;
/* The first position is ATREF */
    ChpCommand(GOREF); /* --- NEW --- PIO has to be initialized to GOREF!! */
                       /* Alberto Bolatto 12/1996 */
    oldpos=ATSRC;

    for(;;) {
st=ChpStatus;
switch (st) {
case MOVING:
    printf("Moving\n");
    break;
case ATSRC:
    if (oldpos==ATSRC) {
/* This is executed when the controller doesn't respond fast enough to
   a request to change position. Normally this wouldn't happen, but
   if the controller program is changed this will alert if something
   goes wrong */
printf("Controller not responding\n");
exit(2);
    }
    curint++;
    if (curint>ninter) {
curint=0;
curpt++;
if (curpt>nrpt) {
/* The controller only accepts an END code when it's at the source */
    printf("Integration finnished!\n");
    ChpCommand(ENDCHOP);
/* We have to set a flag so OBS reads the end code in the serial line
   just to be sure everything went as planned */
    exit(0);
} else {
```

```
    printf("Time on source completed\n");
    oldpos=ATSRC;
    ChpCommand(GOREF);
}
    }
    break;
case ATREF:
    if (oldpos==ATREF) {
/* This is executed when the controller doesn't respond fast enough to
   a request to change position. Normally this wouldn't happen, but
   if the controller program is changed this will alert if something
   goes wrong */
printf("Controller not responding\n");
exit(3);
    }
    curint++;
    if (curint>ninter) {
curint=0;
printf("Time on ref completed\n");
oldpos=ATREF;
ChpCommand(GOSRC);
    }
    break;
case TROUBLE:
default:
    printf("Shit!!\n");
/* We have to set some flag for OBS to go to the serial line, read
   the error code and print a diagnostic */
    exit(1);
}
usleep(10000);
    }

}
```

7 A Word About Tuning

The PID (proportional-integral-differential) servo has been tuned to optimal performace. This involves setting the proportionality constants for the proportinal, integral and differential terms inside the GALIL, as well as setting the maximum speed and acceleration of the mirror. Since we need a stiff servo, i.e., a servo whose asymptotic servo becomes zero pretty fast, an integral constant has been included in the design. Unfortunatly, including an intergral term in the servo equation makes the system prune to instability. Instability is manifest as oscillations of exponentially increasing amplitude in the position of the tertiary mirror. Since our motor is pretty powerful, these oscillations risk smashing the tertiary around against the hard stops and either destroying the drive system or the mirror itself, with danger for the secondary mirror too. The proportional gain and the damping of the servo (or differential gain) are consequently set to the maximum values that garantee that instability will never occur. Please, **DO NOT CHANGE THESE PARAMETERS** unless you understand what you are doing, and when testing new tuning paramenters ALWAYS have someone watching the mirror and someone close to the controller (or the driver) power switch.

8 Software

The relevant software consists of three parts: a general tuning/testing software written by GALIL that runs under DOS called SDK_700, the software that runs inside the controller, and C software that constitutes part of OBS and TRACKER.

SDK_700 is the Servo Design Kit, a software bought from GALIL specially writen for the DMC-700 series that eases the tasks of communication with the controller, tuning, and performance measurement. This software can be found in dumb.spole.gov running under DOS. It is extremely easy to use and self-documenting. A word of warning. Do not wander into the tuning menues if you don't know what you're doing. Some of the tuning tasks activate without asking the user first, and can damage the tertiary motor. Auto-tuning is specially dangerous in this respect. The Absolute Stability Test is one of the tasks that activates as soon as you enter the menu option. Probably the most useful part of the software are the terminal (accessed by pressing F4) and the digital scopes that allow the user to monitor the position, speed, outputs, etc of the GALIL.

The software inside the DMC controller is the core of the interface. It consists of two main parts: the initialization procedure (#INITCH) and the chopping procedure (#CHOP). The first one takes care of the proper initialization of all the relevant parameters of the GALIL and homes the chopper whenever the GALIL is reset (for example, on power on), or when the "bs init:" is executed inside OBS. The #CHOP part is active during chopping and is the part that exchanges information with TRACKER. The following is a commented listing of the software inside the DMC controller.

```
NO
NO Tertiary mirror motor control
NO Alberto Bolatto 12/95
NO
#CHOPPER
JS #INITCH ;NO Initialization
JS #ECHOPR ;NO Echo parms. in serial
JP #FINCH,TYCH=0 ;NO If TYCH=0 end
JS #CHOP   ;NO Otherwise start chopping
#FINCH
EN
#CHOP
SB SRCBIT
PR ,,TRCH
BGZ ;AMZ   ;NO Wait gets to ref.
JS #CHKTRB ;NO Check for trouble
WT GDCH    ;NO Wait to settle
CB REFBIT  ;NO Signal at reference
AI CONBIT  ;NO Wait for CON transition
SB REFBIT  ;NO Signal moving
PR ,,-TRCH ;NO Move to next pos. (src)
BGZ ;AMZ
JS #CHKTRB
WT GDCH
CB SRCBIT  ;NO Signal at src.
AI -CONBIT ;NO Wait for trans. in CON
```

```
JP #ENDCH,@IN[ENDBIT] ;NO End if END
SB SRCBIT  ;NO Signal moving
JP #CHOP,TYCH=1       ;NO If ss chop
TRCH=-TRCH            ;NO If ds chop
JP #CHOP              ;NO Loop
#ENDCH
EN
#INITCH
GN ,,3
KI ,,1
ZR ,,244    ;NO Tuning parameters
IL ,,0     ;NO LS are direct
PS ,,5.0568 ;NO Preliminary scale factor
SP ,,50000
AC ,,262144 ;NO Slow for homming
OVLBIT=1   ;NO from Dynaserv
RDYBIT=2   ;NO from Dynaserv
CONBIT=3   ;NO from computer
ENDBIT=4   ;NO from computer
SRCBIT=0   ;NO to computer
REFBIT=1   ;NO to computer
SB SRCBIT
SB REFBIT  ;NO signal moving
JS #FINDOR,_MOZ ;NO If off find origin
JS #CHKTRB
SP ,,1370000  ;NO Full speed
AC ,,40000000 ;NO Strong acceleration
PA ,,DICH  ;NO Move to abs. offset
BGZ ;AMZ   ;NO Wait till movement done
JS #CHKTRB ;NO Check for trouble bits
WT GDCH    ;NO Wait a guard time
CB SRCBIT
SB REFBIT  ;NO At source
EN
#FINDOR
SHZ        ;NO Start the servo
JS #LRZCH,_LRZ=0 ;NO At R limit?
```

```
JS #LFZCH,_LFZ=0 ;NO At F limit?
JP #HITRLS ;NO Force to hit R limit
#LRZCH     ;NO If at LRZ won't move
PR ,,800   ;NO Move off the R limit
BGZ ;AMZ
EN
#LFZCH
PR ,,-800  ;NO Move off the F limit
BGZ ;AMZ
#HITRLS    ;NO Look for R limit hit
HOMEZ=1    ;NO We are homing
PR ,,-100000  ;NO Hit R limit
BGZ ;AMZ   ;NO Never completes
JS #CHKTRB
EN         ;NO LIMSWI takes over
#LIMSWI
STZ
AMZ        ;NO Stop moving
JP #HOMING,HOMEZ ;NO Are we homing?
JP #TROUBLE      ;NO Signal trouble
RE
#HOMING    ;NO Move fwd
PR ,,7500  ;NO near home swtch
VELZ=_SPZ  ;NO Store act. speed
BGZ ;AMZ
SP ,,500   ;NO Slow speed f accuracy
FEZ
BGZ ;AMZ   ;NO Find home transition
SP ,,VELZ  ;NO Restore old speed
PR ,,-1520 ;NO Offset to center wrt
BGZ ;AMZ   ;NO limit switches
DP ,,0     ;NO Define zero
HOMEZ=0    ;NO Homing finished
RE         ;NO Return fr interrupt
#CHKTRB
JP #TROUBLE,@IN[RDYBIT]
JP #TROUBLE,@IN[OVLBIT]=0
```

```
EN
#TROUBLE
TI
MOZ        ;NO Tell inputs & stop
CB SRCBIT
CB REFBIT  ;NO Signal trouble
ZS         ;NO Zero stacks
EN
#ECHOPR
MG TYCH {F1}
MG DICH {F4}
MG TRCH {F4}
MG GDCH {F2}
EN
```

9 The bs Command

The chopping routine, as now implemented, allows two types of chop: one side chop (type 1) and symmetric or double-sided chopping (type 2). It also allows for an initial displacement (offset) to be introduced before the chopping begins. In this way it's easy to produce small strip-maps by just changing the displacement of the chopper. The central position of the chopper can be altered without chopping using one of the parameters in the bs command together with chop type 0, which only effect is to change the mirror "rest" position.

To define completely the chopping action, three more paramenters are needed: the throw (or extent of the chop), the integration time (actual time spent integrating in each position), and the wait time or guard time. The latter is the time that the controller waits before sending the ATSRC or ATREF code to the computer, after the movement has been completed. This time is spent settling the mirror, and the longer the time the better the accuracy to which the mirror is settled. The tradeoff is of course the duty cycle or efficiency of the chop. For small chops (10 arcmin in the sky) a settling time of 20 ms is probably enough (settles to 60 asec accuracy). Better settling can be attained if we wait for 55 ms (30 asec in the sky). You should set GDCH to the highest value that doesn't compromise too much our duty cycle.

10 Chopper Performace and Some Peculiarities You Should Be Aware Of

The displacement in the sky, beam shape and performace have been measured for the current setup in a series of tests. The result of the displacement tests is that the mirror displacement angle is a factor of 6 higher than the displacement observed in the sky (Fig. 1). During the software development a factor of 10 was assumed, according to the information available at that moment. Because of that, the software inside the GALIL will move the beam in the sky exactly a factor of -1.66 more than the number of arcseconds passed as the parameters DICH or TRCH. This problem has been fixed inside OBS, so the C software corrects for the adequate factor the throw and displacement parameters. You only need to be aware of this fact if you are issuing commands directly to the GALIL.

The beam shape becomes very distorted when chopping 40 arcmin and the efficiency drops to the point that the test HII region H32 is almost completely undetectable at 80 arcmin. Probably the maximum usable chop is between 10 and 20 arcmin.

The maximum frequency at which the chopper can operate is set by the settling time of the mirror, after the movement is completed. Unfortunatly, high frequency oscillations (about 80 Hz) prevent the mirror of settling as fast as it should. Wether this oscillations are asociated with the sampling time of the feedback loop (1 ms) or with mechanical resonances in the structure is, at the time of writing this memo, unclear. The end result is that the maximum frequency at which the chopper can be driven with some duty cycle is probably 4 Hz. The maximum usable frequency with a reasonable duty cycle is probably 1 or 2 Hz (the actual duty cycle will depend on the throw and on the maximum position error that you are willing to allow at the begining of the integration).

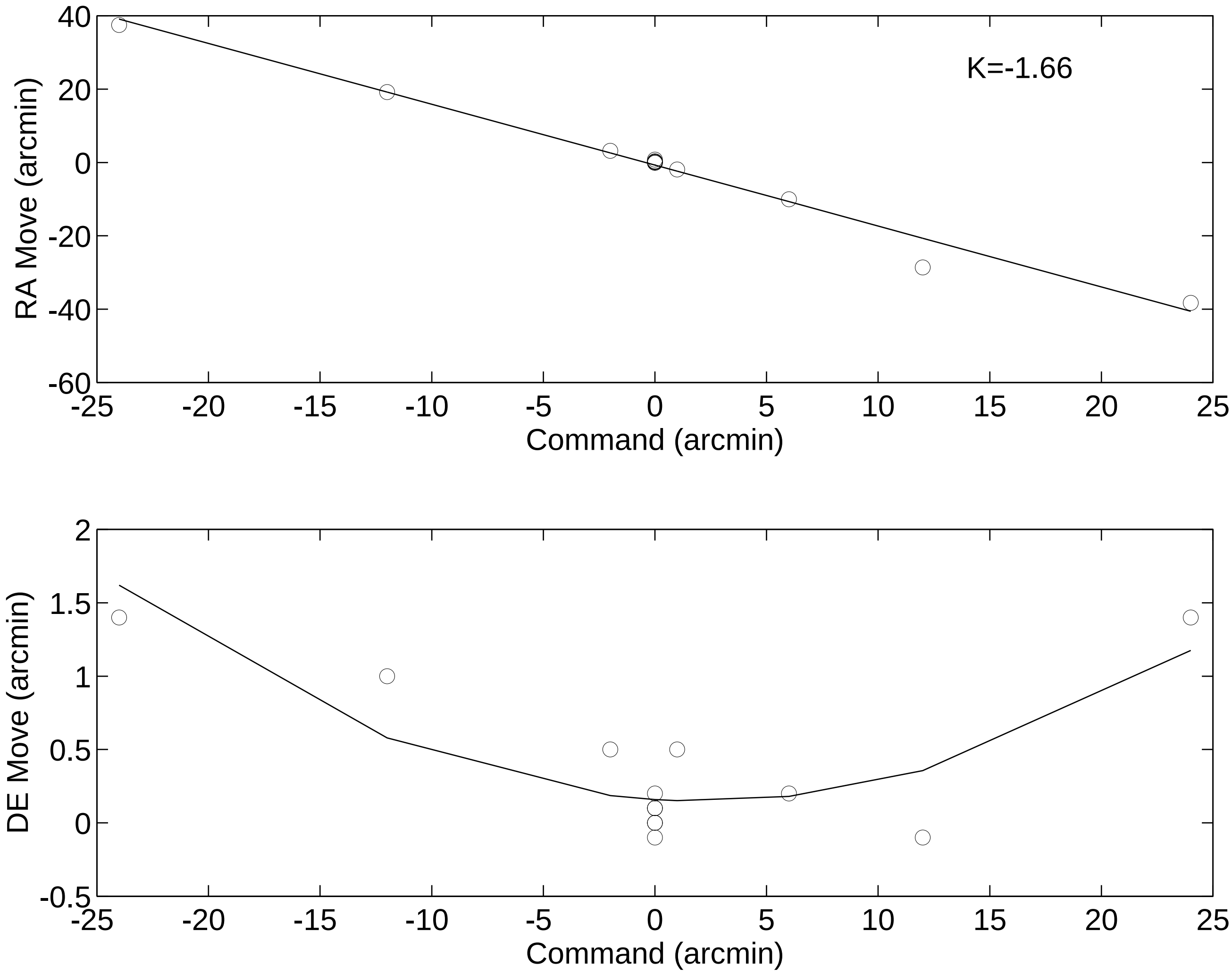

Figure 1: Measurements of the beam displacement in the sky vs. the chopper controller commanded displacement performed during the 1996/1997 summer. The 1.66 factor is incorporated into OBS and it is transparent to the user. You only need to be aware of it if commanding directly the DMC controller. The displacement in elevation is difficult to measure because it is small, even for unpractically large chopper throws.