Annex G: Interface Control Document BSM to MCU

Distribution List

SPIRE-Project	Ken J. King	
	Bruce M. Swinyard	
	Matt Griffin	
	Doug Griffin	
	John Delderfield	
UK ATC	Colin Cunningham	
	Gillian Wright	
	Ian Pain	
	Brian Stobie	
	Brenda Graham	
LAM	Dominique Pouliquen	
	Patrick Levacher	
	Didier Ferrand	

Record of Issue

Date	Index	Remarks
20. Jul.01	4.0	First Issue in this form (previously released as two separate documents)

CONTENTS

1 Introduction

This document outlines interfaces between the BSM and the MCU, in two sections: warm electronics and software.

2 Warm Electronics Interfaces

This document outlines the electrical interface between the Beam Steering Mirror (BSM) assembly and the warm electronics.

The BSM assembly comprises two controllable axes with position sensors and torque motors. It operates at 4 deg. K, remotely (approx. 5m) from the warm electronics, which operates at ~300 deg K.

The BSM electronics has 3 types of analogue sub-circuits per axis :
a) Position sensor current source
b) Position sensor output instrumentation amplifier
c) Motor power amplifier.

In addition, there is provision for a deployable end-stop (DES) to ensure any mechanical failure on launch would leave the mirror in a useable attitude. A launch damper may be instigated instead of (or in parallel with) the DES, which will work by shorting the motor coils by means of switches in the warm electronics. The PCAL mechanism and thermometry wiring are also attached to the BSM structure and harness.

All BSM sub-circuits and associated wiring are duplicated to provide full redundancy. In the event of a detected error, the faulty component will be switched out and the backup component used.

2.1 BSM Assembly to Warm Electronics

Each position sensor has 5 connections to a magnetoresistive element designed to form half of a Wheatstone bridge. Two connections supply current and three sense the bridge voltage. The warm electronics has a constant current source and a set of differential receivers for each axis sensor. It also contains voltage-to-current power amplifiers for each axis motor. This ensures that the motor is driven by the correct current independent of its resistance.
To enable back-EMF sensing for the estimation of rate if a position sensor fails, 4-wire connections are used for the motors.
The BSM wiring diagram is shown in Figure 1, and Table 1 lists the wiring requirements for screening and pairing of signals.

Note that the prime and redundant wiring is identical and has separate connectors, so Figure 1 shows the wiring for either the Prime or Redundant circuits. Therefore, the complete BSM wiring will comprise twice the wiring shown in Figure 1.
Dashed lines around an assembly indicate a common assembly to both Prime and Redundant circuits, e.g. the ‘Deployable End-stop’ assembly is one assembly with a total of 4 wires running to it, 2 Prime and 2 Redundant.

The instrument wiring harness wiring for the motor is defined to have a maximum resistance of 20 ohms.

2.2 BSM to Warm Electronics Interface

The position sensor, power amplifier and Deployable Endstop connect to the warm electronics power amplifiers, differential amplifiers and A-D and D-A converters.

2.3 Position Sensor Interface

2.3.1 Current Source

Each position sensor requires a constant current from a suitable precision source, typically Analog Devices AD584.

Value	1.0 mA per sensor
Tolerance on value	+/- 5%
Variation over 4 hours	+/- 80 ppm (assumes temperature variation of 1 deg/hr in electronics)
Noise	< 1.0 μA rms, 0 to 25 Hz (TBD)
Load Voltage capability	> 1V

2.3.2 Position sensor amplifier

The position sensors require a high-impedance differential amplifier, typically Analog Devices AD524.

2.3.3 Motor Interface

Each axis motor is driven by a voltage-to-current amplifier.

Transfer function gain	10 mA/V +/- 5%
Current sensing resistor	1Ω
Load impedance	350 Ω +/- 25% @20 deg.C
< 5 Ω	@ 4 deg.K
Load Current	50 mA peak (TBD)

-3 dB bandwid _____ *> 5 kHz*

2.3.4 Deployable Endstop Switch

Peak Current	50 mA (TBD)

2.3.5 Deployable Endstop Sensor Interface

Microswitch to + 5V supply (TBD), convert to logic level.

2.4 Electronics to SMEC Processor Interface

2.4.1 A-D and D-A Signals

2.4.2 Sample Rate

The sample rate for chop and jiggle position sensor A-D and motor drive D-A interfaces is 100 μS.

2.4.3 Chop and Jiggle A-D

Resolution 12 bits (TBD) minimum
Full Scale i/p +/- 10V
Conversion time < TBD μS

2.4.4 Chop and Jiggle D-A

Resolution 8 bits (TBD) minimum
Full Scale o/p +/- 10V
Conversion time < TBD μS

2.5 DEPLOYABLE END-STOP SIGNALS

2.5.1 End-stop Command

Value 5V Logic
HIGH = Deploy End-stop
LOW = Withdraw End-stop

2.5.2 End-stop sensor

Value 5V Logic
HIGH = End-stop engaged
LOW = End-stop disengaged

2.6 PCAL TO WARM ELECTRONICS INTERFACE

Refer to the Design Description Document,Annex B for the PCAL INTERFACE.

To minimise noise coupling, the PCAL wiring is kept physically separate from the BSM wiring, and the connector spare pins are allocated between these groups also.

2.7 ISOLATION

The BSM electronics and wiring will have a minimum resistance to chassis of 10 Mohm (TBD) at 100V DC (TBD).

2.8 CONNECTOR TYPE

The Connector used will be a 37-way MDM micro-D type, one for prime wiring and one for redundant wiring.

TABLE 1 BSM Prime and Redundant Wiring

TP = Twisted pair
TT = Twisted triple
TQ = Twisted quad

Pin	Title	Max Voltage	Max Current	Wire Type /Comment
15	Chop motor supply	+/- 15 V	40 mA	STP(15,34)
34	Chop motor return	0V	40 mA	STP(15,34)
16	Chop motor supply sense	+/- 15 V	10 µA	STP(16,35)
35	Chop motor return sense	0V	10 µA	STP(16,35)
36	Jiggle motor supply	+/- 15 V	40 mA	STP(36,18)
18	Jiggle motor return	0V	40 mA	STP(36,18)
37	Jiggle motor supply sense	+/- 15 V	10 µA	STP(37,19)
19	Jiggle motor return sense	0V	10 µA	STP(37,19)
17	Motor screen	0V	0	Screen (commoned)
1	Chop sensor supply	0.4 V	1 mA	STP(1,20)
20	Chop sensor return	0V	1 mA	STP(1,20)
2	Chop sensor supply sense	0.4 V	10 µA	STT(2,3,21)
3	Chop sensor return sense	0V	10 µA	STT(2,3,21)
21	Chop sensor o/p	0.4V	10 µA	STT(2,3,21)
4	Jiggle sensor supply	0.4 V	1 mA	STP(4,23)
23	Jiggle sensor return	0V	1 mA	STP(4,23)
5	Jiggle sensor supply sense	0.4 V	10 µA	STT(5,6,24)
6	Jiggle sensor return sense	0V	10 µA	STT(5,6,24)
24	Jiggle sensor o/p	0.4V	10 µA	STT(5,6,24)
22	Sensor screen	0V	0	Screen (commoned)
7	Mechanism Thermometer 1	0V	2.5 nA	STQ(7,26,8,27)
26	Mechanism Thermometer 2	0V	2.5 nA	STQ(7,26,8,27)
8	Mechanism Thermometer 3	0V	2.5 nA	STQ(7,26,8,27)
27	Mechanism Thermometer 4	0V	2.5 nA	STQ(7,26,8,27)
25	Thermometer Screen	0V	0	Screen(7,26,8,27)
13	Deployable Endstop Engage	+28V	100 mA	STT(13,22,14)
22	Deployable Endstop Retract	+28V	100 mA	STT(13,22,14)
14	Deployable Endstop Common	0V	100 mA	STT(13,22,14)
33	Dep. Endstop Actuator Screen			Screen(13,22,14)
12	Deployable Endstop Sensor	5 V	10 mA	STP(12,30)
30	Deployable Endstop Sensor Return	0 V	10 mA	STP(12,30)
31	Deployable Endstop Sensor Screen	0 V	0	Screen(12,30)
28	PCAL 1	TBD	TBD	STQ(28,29,10,11)
29	PCAL 2	TBD	TBD	STQ(28,29,10,11)
10	PCAL 3	TBD	TBD	STQ(28,29,10,11)
11	PCAL 4	TBD	TBD	STQ(28,29,10,11)
33	PCAL Screen	0V	0	Screen(28,29,10,11)

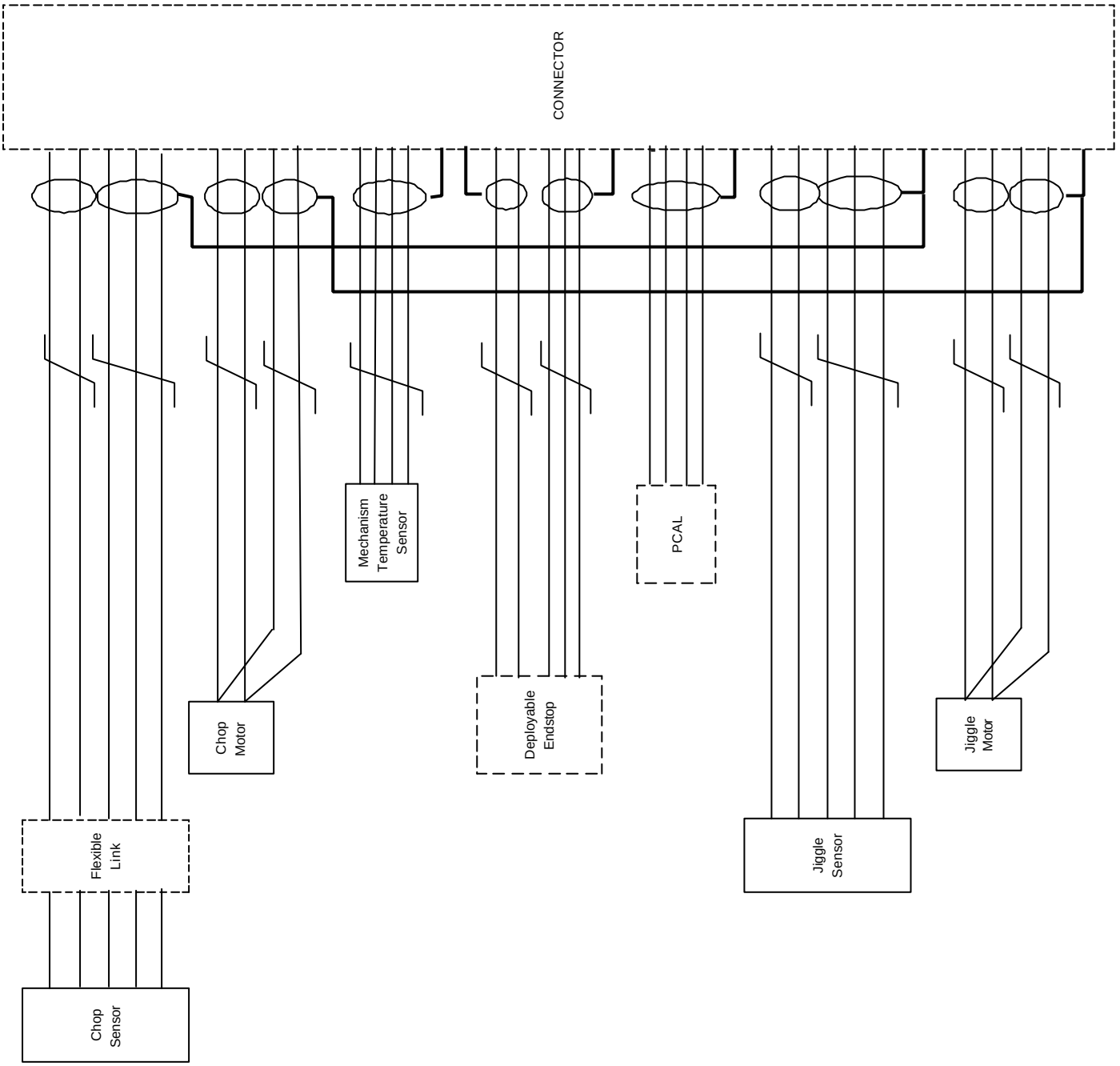

FIGURE 1 The BSM Prime (and Redundant) Wiring.

BEAM STEERING MIRROR CONTROL

3 SOFTWARE REQUIREMENTS

3.1 CHANGE RECORD

1.0	First Issue
1.1	Remove 'Control Structure Command Requirements' section – include necessary flags, etc in control parameters table. Correct observer coefficients used in code for variable 'cha_h'. Add note to indicate that any suitable code for sinusoidal shaping would be OK (maybe some easier assembler method ?)
1.2	Fix some variable name errors. Correct integral gain term in expression for 'ch_pe'. Include 'Kt' in control parameter list.
1.3	Add the Jiggle code – copy of Chop code with name changes
1.4	Incorporated into the BSM Design Description annex G

3.2 INTRODUCTION

This document describes the BSM software operation in 'pseudo-code' – however, the automatically produced code by the Matlab/Simulink Real-Time Workshop build process that links to the hardware-in-the-loop dSPACE development system used with the BSM should be used in preference, as no manual translation is involved.

The BSM is controlled via software running on a DSP. The BSM software controls the position of the two BSM axes in response to external commands from the host software. Each axis can move independently.
The movement with respect to time is profiled via stored parameters to give a minimum energy, minimum noise position change, particularly for step commands.
In general the movements are repetitions of the same position/time profile.

In addition, in the event of measured behaviour resulting in a fault diagnosis, some system backup procedures are available.
Diagnosis of excessive position errors and analysis of recorded transient behaviour during operation can result in modifications to the control system by uploading different parameters into electrically-eraseable memory.

3.3 WAVEFORM COMMAND REQUIREMENTS

The BSM is slaved to the input demands at all times, so to perform a repetitive chop pattern the host processor has to issue a succession of position demands at the relevant times.

For example, to perform a 2 Hz chop between BSM positions p1 and p2, the following chop axis demand sequence and timing is required.
The command update rate (Ts) must always be >= 0.1 mS .

3.3.1.1.1 Time Demand

0.0 → (0.5-Ts) p1
0.5 → (1.0-Ts) p2
1.0 → (1.5-Ts) p1
1.5 → (2.0-Ts) p2
2.0 → (2.5-Ts) p1
etc.

A step command waveform is assumed, and above a certain amplitude (10% of peak), it is profiled to produce a sinusoidal acceleration demand.
Other waveforms, such as triangular, can be approximated by a succession of incremental step demands, however the resolution will always be dependant on the update rate.

Table 1 : Waveform Commands

No.	Parameter	Value or Range
1	Chop Position	+/- 2.4 degrees (BSM axes)
2	Jiggle Position	+/- 0.6 degrees (BSM axes)

3.4 CONTROL SYSTEM DIAGNOSTIC DATA

Some control system data may be used by the host system to determine detected failure correction activities.

Table 2 : Control System Diagnostic Data

No.	Parameter	Range
1	Chop position	+/- TBD
2	Chop current	+/- TBD
3	Chop motor voltage	+/- TBD
4	Jiggle position	+/- TBD
5	Jiggle current	+/- TBD
6	Jiggle motor voltage	+/- TBD

3.5 CONTROL ALGORITHMS

Both axes have position control loops, running at a fixed sample rate of 100 μS.

3.6 CHOP CONTROL

3.6.1 Description

The algorithm comprises two main parts – the control loop itself, and the trajectory sequencer. On reception of a new position, the transition will be profiled, and the profile applied to the control loop. Smaller changes than 10% of maximum will be applied to the control loop directly

The control loop processes the position demand using a digital filter to produce a torque demand to the mirror motor. The algorithm implements rate feedback with low-error position integration.

3.6.2 Trajectory Sequencer

(NOTE : Any other suitable code that is more efficient may be used instead of the following algorithm, as long as the maximum error between the resultant sinusoidal sequence and a perfect sinusoidal sequence is no more than 7.5% of full scale at any point, and the harmonic content is less than 5% of full scale)

This is a pseudo-code description of the trajectory sequencer. A sample rate of 'Ts' seconds is assumed, where Ts >= 0.1 mS.

The trajectory shape is stored as a vector of normalised positions.
If the input to the sequencer (i.e. the demand from the MCU) changes, and the change is large enough to require profiling, any further changes that occur within the nominal step time (15 mS for chop, 50 mS for jiggle) are ignored to ensure completion of the step.

3.6.3 CHOP CONSTANT LIST

Position change threshold at which profiling is assumed necessary:
Pcth = 0.24 deg

3.6.3.1 CHOP VARIABLE LIST

Pc1	Present position command from host
Pc0	Previous position command from host
Pc	Position command to chop position loop
sequencer	Flag = TRUE when sending a sequence to the mirror control loop
Psc	Discrete value of sinusoidal sequence, obtained by multiplying the normalised sine table by the difference between the new position and the last position
m,n	counter variables
Pic	Interpolated values between discrete sine levels obtained from 'Psc'

CHOP PROFILE TABLE

n	Prc(n)
1	0.0
2	0.013
3	0.05
4	0.109
5	0.188
6	0.283
7	0.389
8	0.5
9	0.611
10	0.717
11	0.812
12	0.891
13	0.95
14	0.987
15	1.0

3.6.3.2 CHOP SEQUENCER PSEUDO-CODE

The trajectory time is chosen to be 15 mS for Chop.
Code is expected to be running at a sample time of Ts = 0.1 mS.
The symbol % indicates a comment following the symbol.

```
%  ( The new value of demanded position, Pc1, is obtained from the MCU. )
%
%    Set a flag true if a change has occurred, and we are not in the middle of a previous sequence :
IF  ( (abs(Pc1 – Pc0)  >  Pcth) AND sequencer = FALSE) THEN sequencer = TRUE
```

```
%   If flag is true, then calculate the next position value in the sequence :
IF (sequencer = TRUE )
THEN
        FOR n = 1 to 15
        BEGIN
                %   Generate discrete sine steps :
                Psc = Pc0 + (Pc1-Pc0)*Prc(n)
                FOR m = 1 to 10
                BEGIN
                        %   Interpolate between steps :
                        IF (n<15)
                        THEN
                                %   Generate interpolations ( Prc(x) is from table ) :
                                Pic = (Pc1-Pc0)*(Prc(n+1)-Prc(n))*m/10
                        ELSE
                                %   Don't interpolate last discrete step :
                                Pic = 0
                        END IF
                        %   Calculate complete profile value for this step :
                        Pdc = Psc + Pic
                        %   ( Insert code to output 'Pdc' to mirror axis D-A converter )
                        %   Re-set flag to allow another sequence if this is the last sequence value :
                        IF ((m+n) = 25) THEN sequencer = FALSE
                END
        END
ELSE
%   Simply pass on the latest position value as no sequence is required :
        Pdc = Pc1
        %   ( Insert code to output 'Pdc' to mirror axis D-A converter )
END IF

%   Store previous value of position
Pc0 = Pc1
```

3.6.4 Chop Control Code

3.6.4.1 Description

This pseudo-code description can be used as an alternative to the code automatically produced by dSPACE, which is the hardware-in-the-loop simulation system used in BSM development. However, the dSPACE code should be used in preference, as it does not involve 'manual' translation from the Simulink block diagrams.

Code is expected to be running at a sample time of Ts = 0.1 mS.

The Control loop is based on a state-variable feedback scheme, using an observer to estimate the rate and acceleration of the mirror. The acceleration feedback is used to limit actual mirror acceleration to allow a limited slew rate in the power amplifier at the loop bandwidth required.

At low errors, an integrator is used to reduce position errors further.

Two backup control schemes are used – for the case of a broken flex, the control code is the same, but the control constants are different. For the case of no position sensor being available, the axis rate and accelerations are estimated from the motor voltage.

No useful operation is likely with complete flex joint failure AND no position sensor data – however, 'parking' to a useful fixed position may be possible.

3.6.4.2 Chop Control Parameters

Parameter	Description	Value
prime	prime control scheme	0
broken_flex	control scheme for broken flex joint	1
no_sensor	control scheme for no position sensor	2
chop_control	Select control scheme	prime, broken_flex, no_sensor

Parameter	Description			
		Control Scheme Value		
		Prime	Broken Flex	No Pos. Sensor
Kt	motor torque const.	TBD	TBD	not used
ch_perr_gain	position loop gain	TBD	TBD	not used
ch_rat_gain	rate loop s.f.	TBD	TBD	TBD
ch_acc_gain	acceleration loop s.f.	TBD	TBD	not used
ch_acc_lim	acceleration limit	TBD	TBD	not used
ch_int_gain	integral gain	TBD	TBD	not used
ch_rat_lim	rate limit	TBD	TBD	not used
ch_int_th	integration threshold	TBD	TBD	not used
ch_obs_a1	state coefficient 1	TBD	TBD	not used
ch_obs_a2	state coefficient 2	TBD	TBD	not used
ch_obs_a3	state coefficient 3	TBD	TBD	not used
ch_obs_a4	state coefficient 4	TBD	TBD	not used
ch_obs_b1	input coefficient 1	TBD	TBD	not used
ch_obs_b2	input coefficient 2	TBD	TBD	not used
ch_obs_b3	input coefficient 3	TBD	TBD	not used
ch_obs_b4	input coefficient 4	TBD	TBD	not used
ch_obs_c1	output coefficient 1	TBD	TBD	not used
ch_obs_c2	output coefficient 2	TBD	TBD	not used
ch_obs_d3	output coefficient 3	TBD	TBD	not used
ch_obs_d4	output coefficient 4	TBD	TBD	not used
ch_pos_sf	position s.f.	not used	not used	TBD
ch_cur_sf	current s.f.	not used	not used	TBD
ch_curd_sf	current_dot s.f.	not used	not used	TBD
ch_ddif1	diff. filter coeff1	not used	not used	TBD
ch_ddif2	diff. filter coeff2	not used	not used	TBD

3.6.4.3 Chop Control Pseudo-Code

```
%       ( Accept position demand 'Pdc' from Trajectory Sequencer )

IF (chop_control = prime) OR (chop_control = broken_flex)
THEN
        %       Observer to estimate rate and acceleration :
        % ------------------------------------------------------------------------------------------------------------
        %       ( Read the scaled Position Sensor output 'pc', and scaled current 'ci' )
        cob_u1  =       pc*Kt           %  torque = current measurement*torque constant
        cob_u2  =       ci
        %       state update :
        cx1     =       ch_obs_a1*cx1_p + ch_obs_a2*cx2_p + ch_obs_b1*ob_u1 +       ch_obs_b2*cob_u2
        cx2     =       ch_obs_a3*cx1_p + ch_obs_a4*cx2_p + ch_obs_b3*ob_u1 + ch_obs_b4*cob_u2
        %       output update :
        chr_h   =       ch_obs_c1*cx1_p                                                 % rate estimate
        cha_h   =       ch_obs_c2*cx2_p + ch_obs_d3*cob_u1 + ch_obs_d4*cob_u2 % accel. estimate
        %       store states for next cycle :
        cx1_p   =       cx1
        cx2_p   =       cx2
END IF

IF (chop_control = no_sensor)
THEN
        %       read motor voltage 'ch_volts' and motor currrent 'ch_cur'
        %       calculate filtered differential of current
        ch_curd =       ch_ddif1*ch_curd_p + ch_ddif2*(ch_cur – ch_cur_p)
        %       now estimate rate :
        chr_h   =       ch_volts –  ch_cur_sf*ch_cur – ch_curd_sf*ch_curd
        %
        %       store variables for next cycle
        ch_cur_p        =       ch_cur
        ch_curd_p       =       ch_curd
END IF

%       Axis control code using previous estimates :
% ------------------------------------------------------------------------------------------------------------

IF (chop_control = prime) OR (chop_control = broken_flex)
THEN
        ch_err  =       Pdc – pc                                        % position summing junction

        %       windowed integrator :
        IF ( abs(ch_err) > ch_int_th ) THEN
                chio    =       0.5e-4*ch_err*( 1.0 + che_p + chio_p)       % assumes Ts = 1e-4
        ELSE
                chio    =       0
        END IF
        che_p   =       ch_err                                          % store for next cycle
        chio_p  =       chio


        ch_pe   =       ch_perr_gain*(ch_err + ch_int_gain*chio)

        IF ( abs(ch_pe) > ch_rat_lim ) THEN                             % rate limit
                ch_pel  =       ch_rat_lim*sgn(ch_pe)
        END IF
```

```
ch_ad    =        ch_pel – ch_rat_gain*chr_h          % rate summing junction

IF ( abs(ch_ad) > ch_acc_lim ) THEN                   % acceleration limit
        ch_adl   =        ch_acc_lim*sgn(ch_ad)
END IF

ch_id    =        ch_adl – ch_acc_gain*cha_h          % chop current demand

END IF

IF (chop_control = no_sensor)
THEN
        ch_id    =        ch_pos_sf*P_dc – ch_rat_gain*chr_h
END IF
```

See File SPIRE-ATC-MDL-XXX for the Matlab/Simulink model representing the control loop and mechanism.

See File SPIRE-ATC-SW-XXX for the 'C' code produced by the dSPACE prototyping system from a Matlab/Simulink model, performance verified by hardware-in-the-loop tests in report SPIRE-ATC-XXX-000xxx.

3.7 JIGGLE CONTROL

3.7.1 Description

The algorithm is essentially similar to the Chop axis, apart from some parameter values to suit the different inertia, motor torque and flex joint characteristics.

3.7.2 Trajectory Sequencer

This is a pseudo-code description of the trajectory sequencer. A sample rate of 'Ts' seconds is assumed, where Ts >= 0.1 mS.
The trajectory shape is stored as a vector of normalised positions.

3.7.2.1 JIGGLE CONSTANT LIST

Position change threshold at which profiling is assumed necessary:
Pjth = 0.06 deg

3.7.2.2 JIGGLE VARIABLE LIST

Pj1	Present position command from host
Pj0	Previous position command from host
Pj	Position command to chop position loop

Pj1	Present position command from host
Pj0	Previous position command from host
Pj	Position command to chop position loop

sequencer	Flag = TRUE when sending a sequence to the mirror control loop

Psj	Discrete value of sinusoidal sequence, obtained by multiplying the normalised sine
	table by the difference between the new position and the last position
m,n	counter variables
Pij	Interpolated values between discrete sine levels obtained from 'Psj'

JIGGLE PROFILE TABLE

n	Prj(n)
1	0.0
2	0.004
3	0.017
4	0.038
5	0.067
6	0.103
7	0.146
8	0.196
9	0.25
10	0.309
11	0.371
12	0.435
13	0.5
14	0.565
15	0.629
16	0.691
17	0.75
18	0.804
19	0.854
20	0.897
21	0.933
22	0.962
23	0.983
24	0.996
25	1.0

3.7.2.3 JIGGLE SEQUENCER PSEUDO-CODE

The trajectory time is chosen to be 50 mS for Jiggle.
Code is expected to be running at a sample time of Ts = 0.1 mS.
The symbol % indicates a comment following the symbol.

```
%  ( The new value of demanded position, Pj1, is obtained from the MCU. )
%
%    Set a flag true if a change has occurred, and we are not in the middle of a previous sequence :
IF  ( (abs(Pj1 – Pj0)  >  Pcth) AND sequencer = FALSE) THEN sequencer = TRUE

%   If flag is true, then calculate the next position value in the sequence :
IF (sequencer = TRUE )
THEN
        FOR n = 1 to 25
        BEGIN
                %    Generate discrete sine steps :
```

```
                Psj = Pj0 + (Pj1-Pj0)*Prj(n)
                FOR m = 1 to 20
                BEGIN
                        %   Interpolate between steps :
                        IF (n<25)
                        THEN
                                %   Generate interpolations ( Prj(x) is from table ) :
                                Pij = (Pj1-Jc0)*(Prj(n+1)-Prj(n))*m/10
                        ELSE
                                %   Don't interpolate last discrete step :
                                Pij = 0
                        END IF
                        %   Calculate complete profile value for this step :
                        Pdj = Psj + Pij
                        %   ( Insert code to output 'Pdj' to mirror axis D-A converter )
                        %   Re-set flag to allow another sequence if this is the last sequence value :
                        IF ((m+n) = 45) THEN sequencer = FALSE
                END
        END
ELSE
%   Simply pass on the latest position value as no sequence is required :
        Pdj = Pj1
        %   ( Insert code to output 'Pdj' to mirror axis D-A converter )
END IF

%   Store previous value of position
Pj0 = Pj1
```

3.7.3 Jiggle Control Code

3.7.3.1 Description

Code is expected to be running at a sample time of Ts = 0.1 mS.
The Control loop is based on a state-variable feedback scheme, using an observer to estimate the rate and acceleration of the mirror. The acceleration feedback is used to limit actual mirror acceleration to allow a limited slew rate in the power amplifier at the loop bandwidth required.
At low errors, an integrator is used to reduce position errors further.

3.7.3.2 Jiggle Control Parameters

Parameter	Description	Value
prime	prime control scheme	0
broken_flex	control scheme for broken flex joint	1
no_sensor	control scheme for no position sensor	2
jig_control	Select control scheme	prime, broken_flex, no_sensor

Parameter	Description			
		Control Scheme Value		
		Prime	Broken Flex	No Pos. Sensor
Kt	motor torque const.	TBD	TBD	not used
jg_perr_gain	position loop gain	TBD	TBD	not used
jg_rat_gain	rate loop s.f.	TBD	TBD	TBD
jg_acc_gain	acceleration loop s.f.	TBD	TBD	not used
jg_acc_lim	acceleration limit	TBD	TBD	not used
jg_int_gain	integral gain	TBD	TBD	not used

Parameter	Description			
		Control Scheme Value		
		Prime	Broken Flex	No Pos. Sensor
jg_rat_lim	rate limit	TBD	TBD	not used
jg_int_th	integration threshold	TBD	TBD	not used
jg_obs_a1	state coefficient 1	TBD	TBD	not used
jg_obs_a2	state coefficient 2	TBD	TBD	not used
jg_obs_a3	state coefficient 3	TBD	TBD	not used
jg_obs_a4	state coefficient 4	TBD	TBD	not used
jg_obs_b1	input coefficient 1	TBD	TBD	not used
jg_obs_b2	input coefficient 2	TBD	TBD	not used
jg_obs_b3	input coefficient 3	TBD	TBD	not used
jg_obs_b4	input coefficient 4	TBD	TBD	not used
jg_obs_c1	output coefficient 1	TBD	TBD	not used
jg_obs_c2	output coefficient 2	TBD	TBD	not used
jg_obs_d3	output coefficient 3	TBD	TBD	not used
jg_obs_d4	output coefficient 4	TBD	TBD	not used
jg_pos_sf	position s.f.	not used	not used	TBD
jg_cur_sf	current s.f.	not used	not used	TBD
jg_curd_sf	current_dot s.f.	not used	not used	TBD
jg_ddif1	diff. filter coeff1	not used	not used	TBD
jg_ddif2	diff. filter coeff2	not used	not used	TBD

3.7.3.3 Jiggle Control Pseudo-Code

(Except for the change in variable names, the jiggle control code is the same as the chop code)

```
%       ( Accept position demand 'Pdj' from Trajectory Sequencer )

IF (jig_control = prime) OR (jig_control = broken_flex)
THEN
        %       Observer to estimate rate and acceleration :
        % ------------------------------------------------------------------------------------------------------------
        %       ( Read the scaled Position Sensor output 'pj', and scaled current 'ji' )
        job_u1  =       pj*Kt           %  torque = current measurement*torque constant
        job_u2  =       ji
        %       state update :
        jx1     =       jg_obs_a1*jx1_p + jg_obs_a2*jx2_p + jg_obs_b1*ob_u1 +   jg_obs_b2*job_u2
        jx2     =       jg_obs_a3*jx1_p     +   jg_obs_a4*jx2_p     +   jg_obs_b3*job_u1    +
        jg_obs_b4*job_u2
        %       output update :
        jgr_h   =       jg_obs_c1*jx1_p                                         % rate estimate
        jga_h   =       jg_obs_c2*jx2_p + jg_obs_d3*job_u1 + jg_obs_d4*job_u2   % accel. estimate
        %       store states for next cycle :
        jx1_p   =       jx1
        jx2_p   =       jx2
END IF

IF (jig_control = no_sensor)
THEN
        %       read motor voltage 'jg_volts' and motor currrent 'jg_cur'
        %       calculate filtered differential of current
        jg_curd =       jg_ddif1*jg_curd_p + jg_ddif2*(jg_cur – jg_cur_p)
        %       now estimate rate :
        jgr_h   =       jg_volts –  jg_cur_sf*jg_cur – jg_curd_sf*jg_curd
```

```
        %
        %       store variables for next cycle
        jg_cur_p        =       jg_cur
        jg_curd_p       =       jg_curd
END IF

%       Axis control code using previous estimates :
% ------------------------------------------------------------------------------------------------------------------

IF (jig_control = prime) OR (jig_control = broken_flex)
THEN
        jg_err  =       Pdj – pj                                % position summing junction

        %       windowed integrator :
        IF ( abs(jg_err) > jg_int_th ) THEN
                jgio    =       0.5e-4*jg_err*( 1.0 + jge_p + jgio_p)       % assumes Ts = 1e-4
        ELSE
                jgio    =       0
        END IF
        jge_p   =       jg_err                                  % store for next cycle
        jgio_p  =       jgio


        jg_pe   =       jg_perr_gain*(jg_err + jg_int_gain*jgio)

        IF ( abs(jg_pe) > jg_rat_lim ) THEN                     % rate limit
                jg_pel  =       jg_rat_lim*sgn(jg_pe)
        END IF

        jg_ad   =       jg_pel – jg_rat_gain*jgr_h          % rate summing junction

        IF ( abs(jg_ad) > jg_acc_lim ) THEN                     % acceleration limit
                jg_adl  =       jg_acc_lim*sgn(jg_ad)
        END IF

        jg_id   =       jg_adl – jg_acc_gain*jga_h              % jgop current demand

END IF

IF (jig_control = no_sensor)
THEN
        jg_id   =       jg_pos_sf*P_dc – jg_rat_gain*jgr_h
END IF
```

See File SPIRE-ATC-MDL-XXX for the Matlab/Simulink model representing the control loop and mechanism.

See File SPIRE-ATC-SW-XXX for the ‘C’ code produced by the dSPACE prototyping system from a Matlab/Simulink model, performance verified by hardware-in-the-loop tests in report SPIRE-ATC-XXX-000xxx.

3.8 LAUNCH CONTROLS

During launch, a deployable end-stop and switchable damping by short-circuiting the motors prevents flex joint damage.

3.8.1 Deploy End-stop and Dampers

Variable Name	***Value***	***Meaning***
deploy_endstop	1	Deploy endstop and dampers
	0	Remove endstop and dampers

3.8.2 End-stop Status

Variable Name	**Value**	**Meaning**
endstop_status	1	End-stop and dampers on
	0	End_stop and dampers off