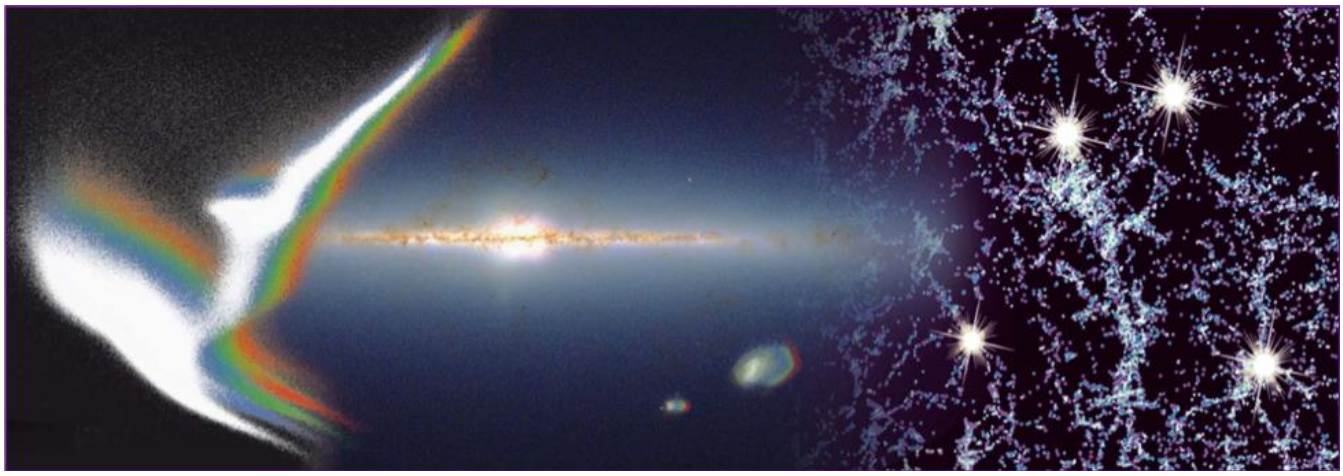


SDSS-V Positioner Interface Control Document

SDSS-V_0087_FPS_Positioner_interfaces-v1.0

1.2

3/30/21

Luzius Kronig

SDSS-V

Lead Author / Owner	SDSS Job Title	Date	Signature
Luzius Kronig (EPFL)	Doctoral Assistant		
Contributors			
Loïc Grossen (EPFL) Ricardo Araujo (EPFL) Dan Pappalardo Richard Pogge	Scientific Assistant ETS/HES Engineer		

REVISION LOG

Version	Date	Affected Section(s)	Changes / Comments
0.1	10/23/19	All	Initial Draft
1.0	12/06/19	All	Released
1.1	06/08/20	Changes in: 5.3 new: 6,7,8,9	New Sections: Syncline, Operation, Motor calibration, Troubleshooting
1.2	02/15/21	New 5.3.58-5.3.64,7.4, 7.5, 1.3	Explanations on control methods

Table of Contents

List of Figures

List of Tables

Introduction

This document provides a description of the interfaces of the Robotic Fiber Positioners for the SDSS-V FPS project.

1.1 System overview

The goal is to develop and deploy by end 2020 a robotic fiber positioner system (with 500 robots each) on both the 2.5m Sloan telescope at Apache Point Observatory (APO) in New-Mexico and on the 2.5m Irenee du Pont telescope at the Las Campanas Observatory in Chile. Each Telescope has 500 zonal fiber positioners arranged in a hexagonal array, each carrying a science fiber for the BOSS spectrographs, and a subset of 300 carrying a second fiber for the APOGEE spectrograph. The distribution of the 300 APOGEE fibers among the 500 positioners is optimized to maximize the common field of view. The positioner array is interspersed with a set of field-illuminated fiducials to establish a fixed reference frame against which fiber positions are measured with a fiber viewing camera. The basic layout of the FPS focal plane is shown in Figure 1.

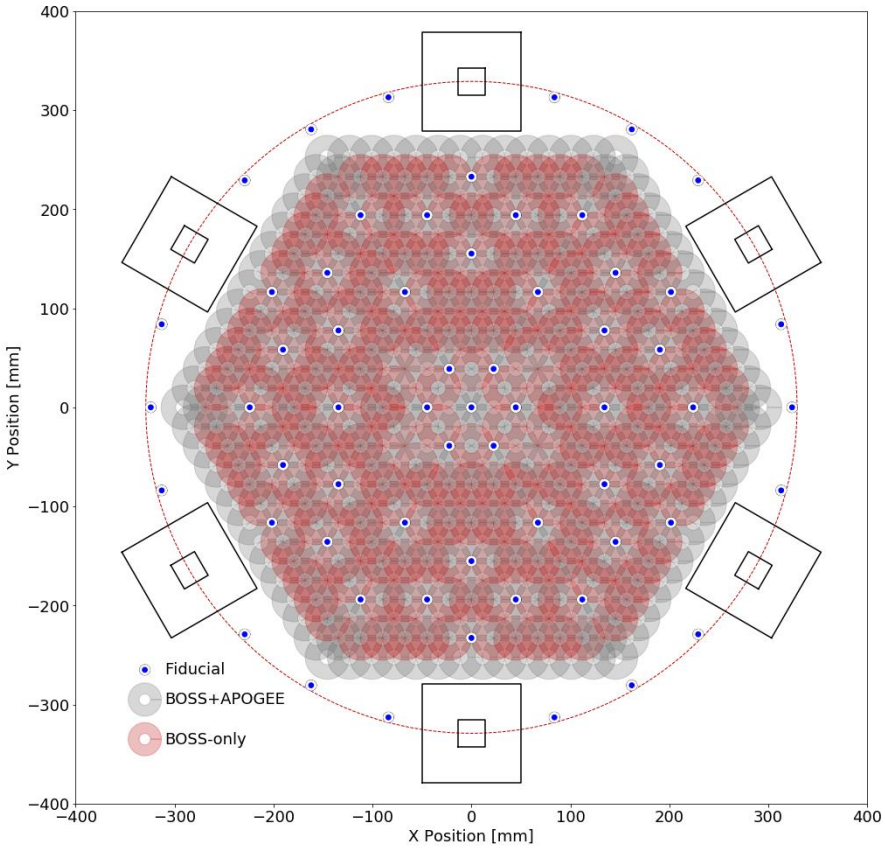
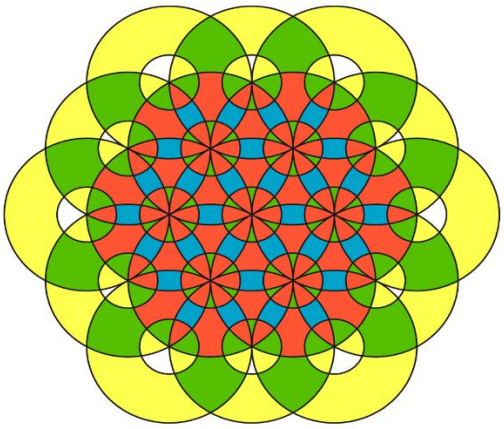


Figure 1: FPS focal plane layout showing positioners (colored + and annuli), fiducials (blue & white circles), and guide camera locations (black squares). Gray and red transparent circles show the patrol

fields of each positioner. The large red dashed circle is the field of view of the current generation plug plates

The workspace of each positioner can overlap with the workspaces of the neighbouring positioners such as shown on
Figure 2.



Color	Number of positioner reaching the zone
Yellow	1
Green	2
Red	3
Blue	4

Figure 2: Overlapping workspaces

1.2 Positioner overview

The positioner is composed of two actuators (alpha and beta as shown in

Figure 4) and an electronic board to control the actuators and communicate with the Instrument Electronics Box (IEB).

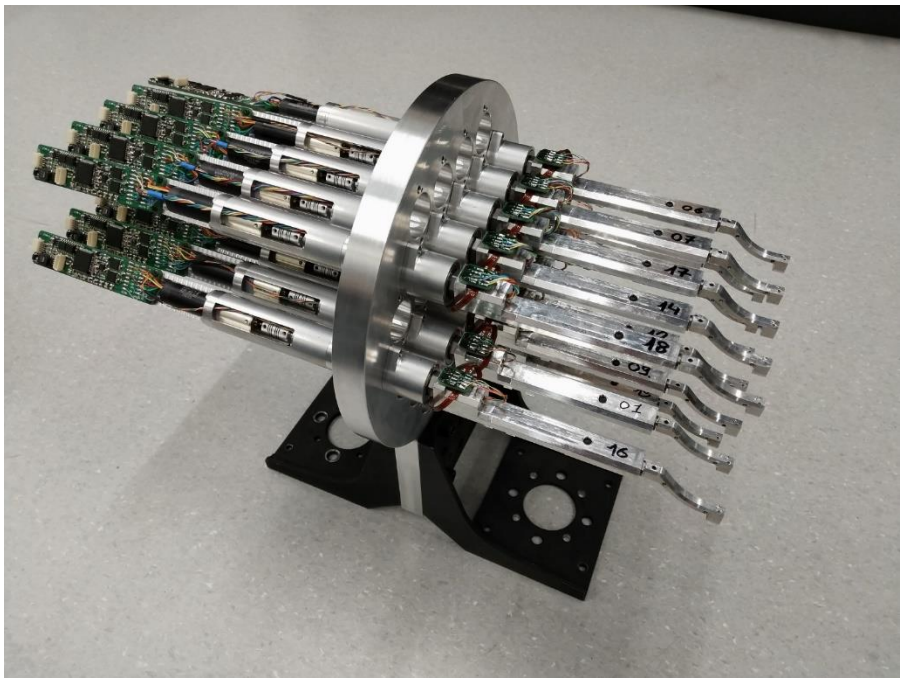


Figure 3: SDSS-V Fiber Positioners P1 prototypes

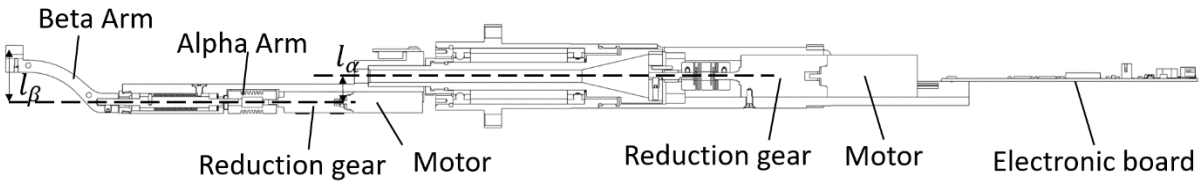


Figure 4: Positioner actuators cut view (P1 model)

1.3 Handling Instructions

- Avoid electrostatic discharges.
- Use the default modes described under section 7.
- In case of problems follow the troubleshooting guide under section 9.
- When handling the positioner never pull, push or exert any force on the alpha or beta arm (black parts with numbers).
- When installing positioners make sure that the motor cables for the beta arm stay always well within the allowed envelop. Cables which reach into the envelop of neighboring robots may tear due to repeated impacts from other robots.

2 Mechanical Interface

2.1 Positioner housing

The positioner is mounted on the focal plate through an 18.5mm G6 hole for precise positioning with minimal play. The positioner is then fixed onto the plate using two M3 screws 25 mm apart. A pin is placed on the support plate, and a corresponding hole is machined in the positioner for precise orientation.

Due to the size and shape of the top part of the positioner (beta arm), it must be mounted in the plate through the front side, the fixation is then done with the two screws inserted from the front side of the plate and screwed into the threaded holes of the plate.

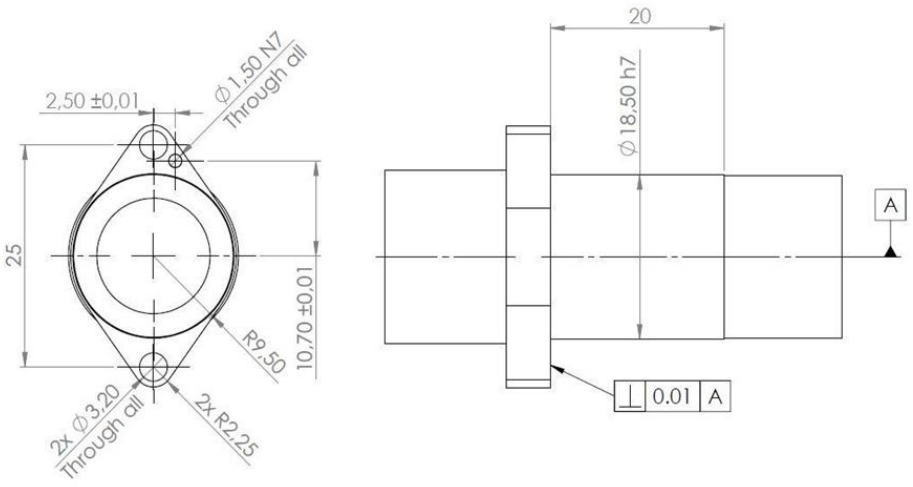


Figure 5: Housing fixation view

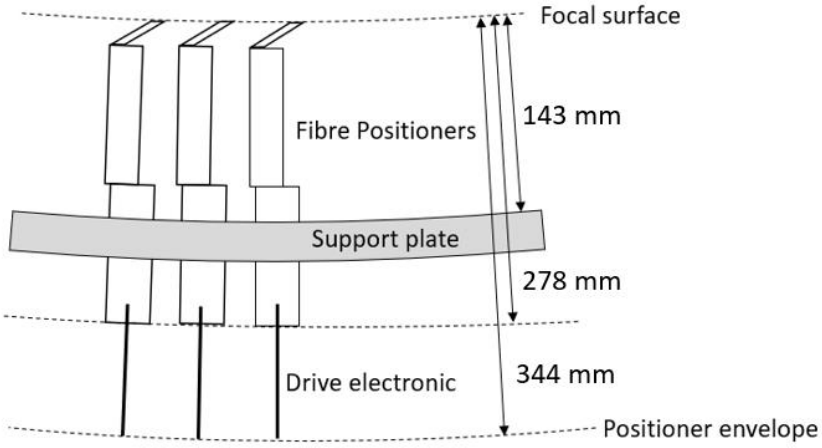


Figure 6: Positioner on support plate schema

2.2 Hard stops

The range of the α and β motors are shown on Figure 7 and are defined by:

$$-\alpha_{HSL} < \alpha < 2\pi + \alpha_{HSU}$$
$$-\beta_{HSL} < \beta < 2\pi + \beta_{HSU}$$

Both motors are capable to make at least one full rotation covering the initial configuration twice at 0° and 360°. Hardstops are used for datum initialization and to limit the overall range. The lower bound hardstop angles are defined as α_{HSL} and β_{HSL} and the upper bound hardstops

are α_{HSU} and β_{HSU} respectively. The exact hardstop values are defined by the mechanical design and manufacturing tolerances.

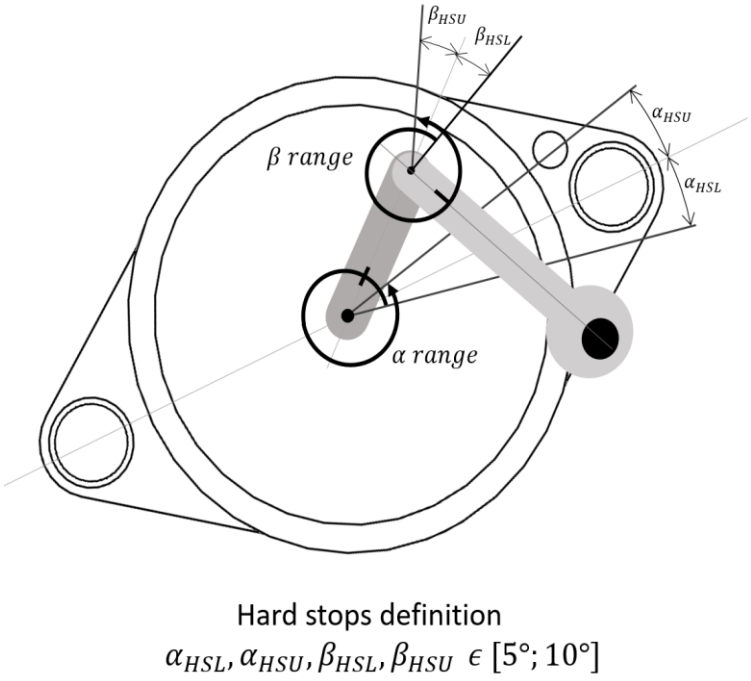


Figure 7: Hardstop definition

3 Ferule and fiber Interface

The ferrule fiber interface is shown in appendix 7.1

4 Electrical Interface

Three signals must be provided to the positioners: the power supply, the communication signals and a synchronization signal. The electrical interfaces are thus subdivided in two main categories, power and communication. Figure 8 shows how the positioners are grouped into different daisy chains. Each daisy chain has its own CAN bus. Figure 9 shows how the daisy chains are distributed in the focal plane.

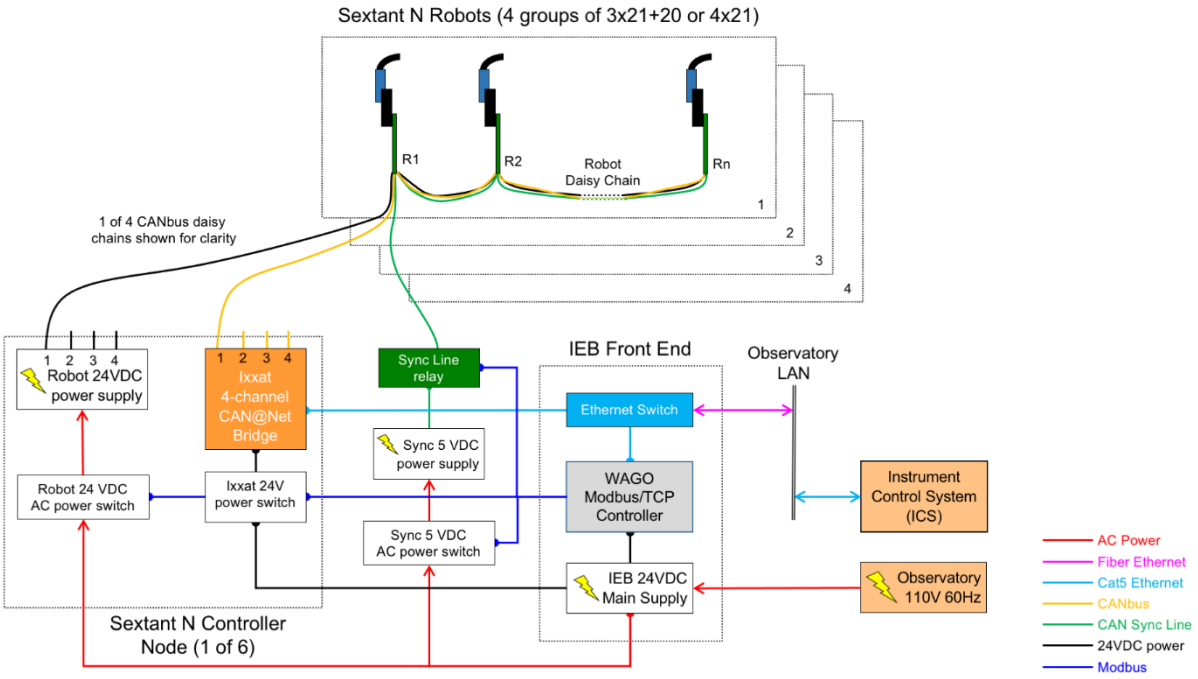


Figure 8: SDSS-V Fiber Positioners Electrical overview

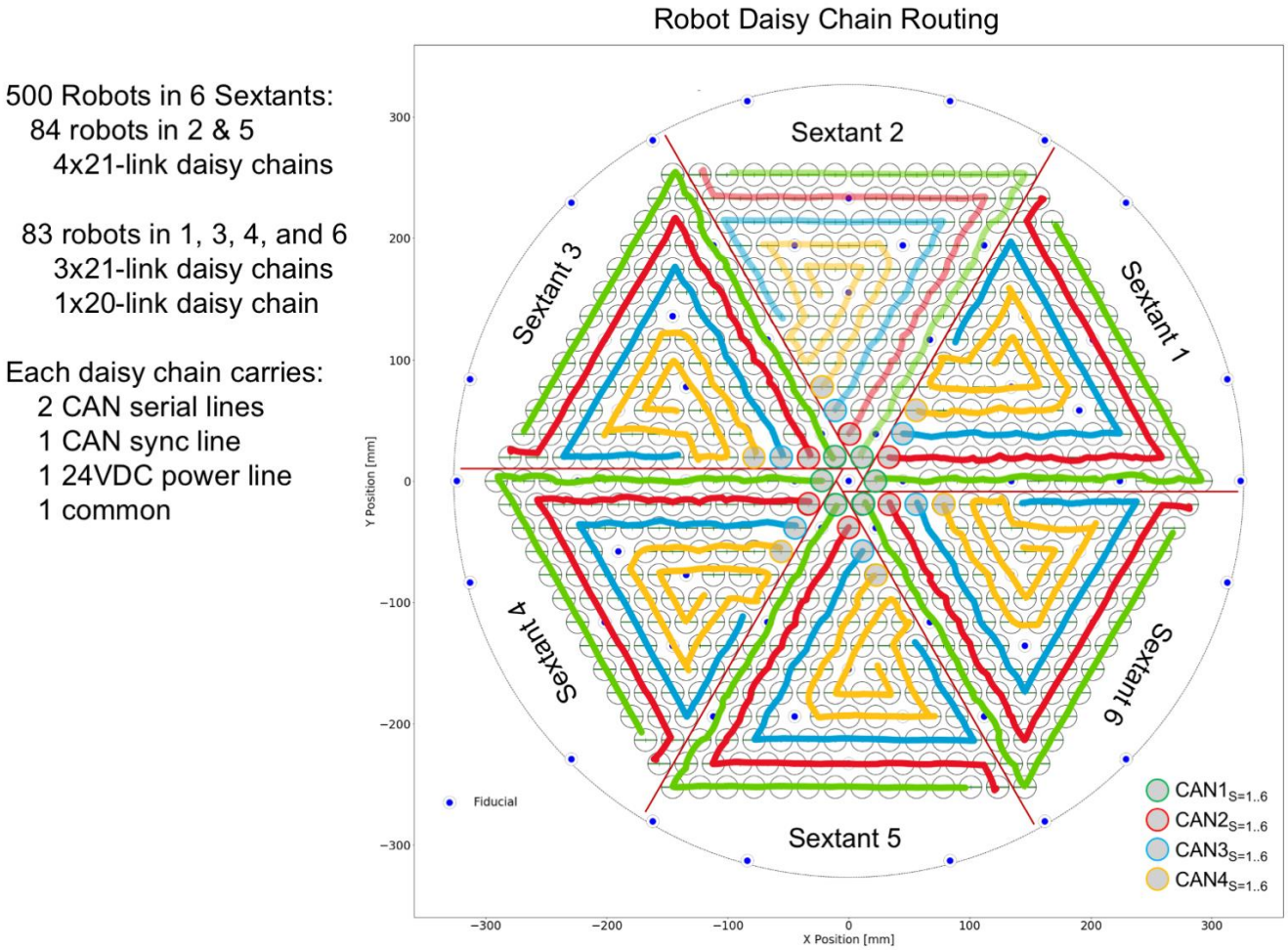


Figure 9: Distribution of the daisy chains in the focal plane

4.1 Sextant Controller

Figure 8 and 10 show how all positioners are provided with 24V power, CAN communication and the sync line. The focal plane is divided into six parts with either 84 robots or 83 three robots per part. Every part has its own sextant controller which provides the 84/83 positioners with power, CAN and the sync line. Every sextant controller drives four daisy chains with either 20 or 21 robots per chain. All 6 sextant controllers are located in the IEB (1 enclosure). Each sextant controller has its own 24VDC power supply and 4 channel CAN module. They are controlled by the ICS via an Ethernet switch and PLC.

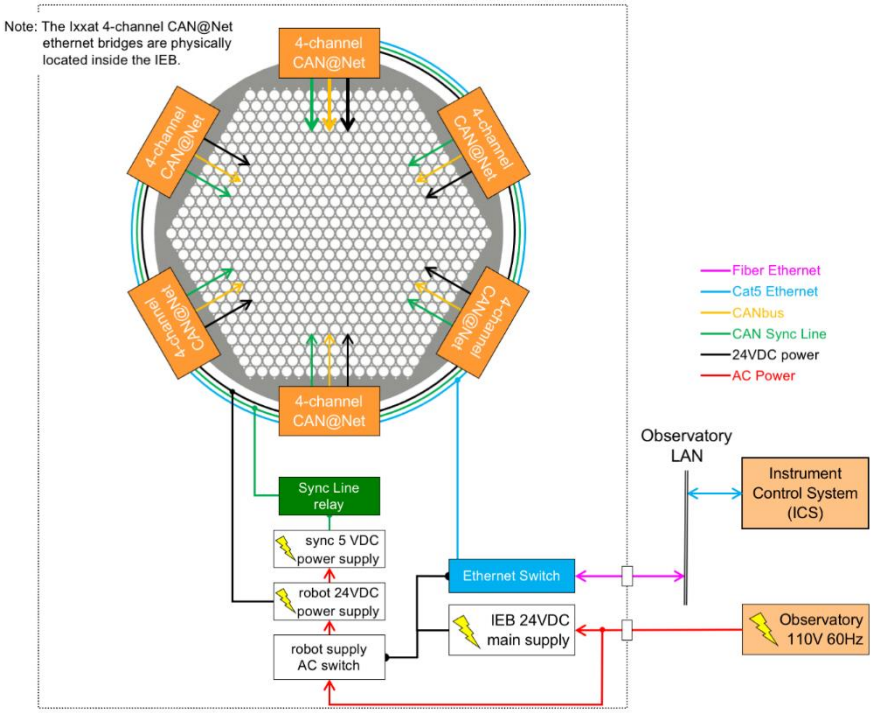


Figure 10: SDSS-V Fiber Positioners Electrical overview

4.2 Power distribution

The power in the IEB includes an “always on” subset that includes the Ethernet switch and the mini PLC. The PLC reads all process and environmental data (temperatures, relative humidity, pressure, and coolant flow) for the FPS. The power and the Ethernet-CAN module to each sextant of positioners can be remotely powered down via the PLC. Therefore the 24VDC positioner power and/or the CAN module can be powered down independently for every sextant. The 24V tension for the positioners minimizes the cable losses and is widely used in the industrial field. The system will be operated for low power consumption.

4.3 Positioner connections

The positioner is connected to the main control box using a daisy chain of 5 cables as shown in Figure 11. The sync line input range goes from -30V to 30V. It is inactive below 1.9 V and active above 2.6 V.

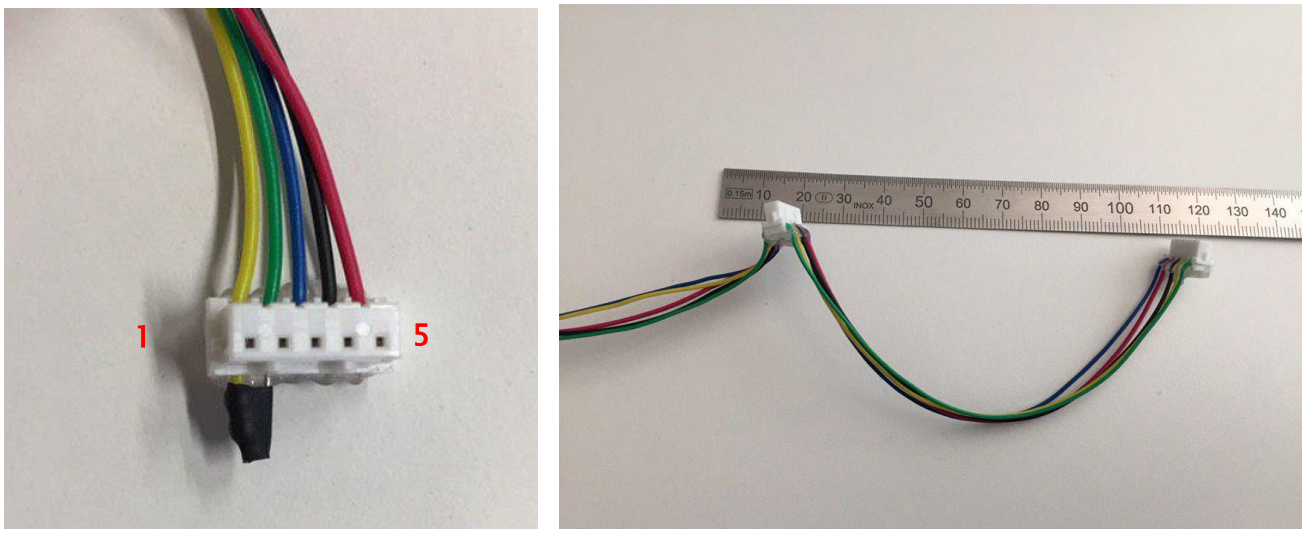


Figure 11: Electronic connector (left) and daisy-chained cable (right)

The connector on the board is S5B-PH-SM4-TB-LF-SN
The daisy chainable connector is 05KR-D6S-P

The pin out of the connector is as per the following table:

Connector pin	Signal	Description
1	CANH	CAN High signal for communication
2	CANL	CAN Low signal for communication
3	SYNC	Synchronization line
4	GND	Ground
5	Power	Power input (9-24 V DC)

Table 1: Positioner bus connector pin out

4.4 Electronics Board

Each positioner is equipped with an electronic board for commanding the brushless DC motors. A sketch of the board with corresponding functionalities is shown in Figure 12.

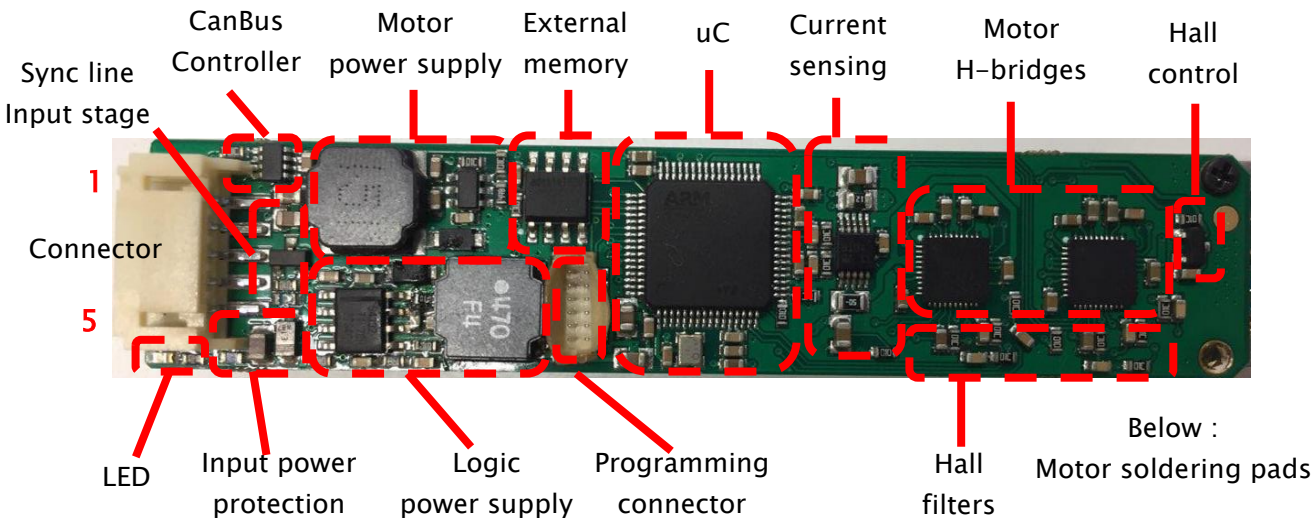


Figure 12: Electronics board

5 Communication Interface

5.1 General communication protocol for P1 devices

The communication with the bootloader and the main application is done using a CAN bus at a speed of 1 Mbit/s. It uses the extended frame format with 29 identifier bits. The 29 identifier bits are divided in four parts: the ID of the positioner, the command to send, the command unique identifier and a response code as it can be seen in Table 2. This way every response frame from the positioner can be associated with the corresponding command number and every command instance can also be identified.

Table 2: 29 CAN bits splitting for P1 devices

CAN 29 bits identifier																													
11 Higher bits											8 Upper middle bits								6 Lower middle bits						4 Lower bits			
28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Positioner ID											Command number								Command uid						Response code			

Every positioner has a unique ID but can also receive and respond to some broadcast commands (broadcast commands are sent to ID 0).

The positioner will respond to every received command with a frame containing its ID, command number responded to, the command uid that was sent with the command, a response code corresponding to the result of the executed command and extra data if needed.

The data on the bus is sent using little-endian format. The possible response codes are listed in Table 3.

Table 3: Possible Response Codes for P1 devices

Response code	Description	Remark
0	Command accepted	Command was processed without any errors.
1	Value out of range	Value that was sent is out of range (speed, current, etc.)

Response code	Description	Remark
2	Invalid trajectory	Trajectory point is not possible (out of bounds…)
3	Already in motion	Cannot start new move, because already in motion.
4	Datum not initialized	Initiation of actuators has not been performed
5	Incorrect amount of data	Amount of data with command is invalid.
6	Calibration mode active	A calibration mode is active (e.g. motor, datum, cogging)
7	Motor not calibrated	The motors are not calibrated and therefore the hall sensors can't be used
8	Alpha collision detected	a collision is detected on alpha, the flag has to be first cleared with stop trajectory
9	Beta collision detected	a collision is detected on alpha, the flag has to be first cleared with stop trajectory
10	Invalid broadcast command	Command no possible as broadcast.
11	Invalid bootloader command	Command not valid for bootloader mode
12	Invalid command	Command not valid for main application mode
13	Unknow command	Command unknown
14	Datum not calibrated	Calibrate datums first
15	Hall sensors disabled	hall sensors are disabled and can therefore not be used, switch on hall sensors and/or change from open loop control without collision detection

5.2 Bootloader V03.80.01 communication commands (P1 devices)

At power on, the device will enter the bootloader and directly load the main application. In order to enter the bootloader mode and perform maintenance operations (like firmware upgrade), the sync line must be driven to GND. If the bootloader mode is entered, the positioner

will wait for commands until a 10 seconds timeout is reached. If no valid command is received for 10 seconds, the device will then boot the main application. If a valid command is received the 10 seconds timer will be reset.

Table 4 lists the most useful commands available during bootloader mode.

Table 4: Useful bootloader commands for P1 devices

Cmd	Description	Broadcast	Command extra data	Answer extra data
1	Get ID	Yes	None	Positioner ID
2	Get firmware version	Yes	None	Bootloader firmware version
3	Get status	Yes	None	Bootloader status register
200	Start firmware upgrade	No	Size and checksum	None
201	Firmware data	No	Firmware data	None

5.2.1 Get ID command (1)

This command will return a frame containing 4 bytes of data representing the positioner ID as an unsigned integer of 32 bits.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	None	None

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	ID of the positioner	None

5.2.2 Get firmware version (2)

This command will return a frame containing 4 bytes of data representing the actual version of the firmware running on the device. The firmware versions are identified using semantic versioning number style with the format XX.YY.ZZ. The first received byte will be zero, the second one will correspond to XX, the third one to YY and the last one to ZZ. The bootloader firmware has YY has 80 to be distinguished from the main application firmware.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	None	None

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	Version number	None

5.2.3 Get status (3)

This command will return a frame containing 4 bytes of data as an unsigned integer of 32 bits representing the status register of the bootloader. Table 5 describes the meaning of each bit, if a bit is not represented it means it is not used.

Bit	Name	Description	Mask (hex)
0	BOOTLOADER_INIT	Bootloader has been initialized	0x00000001
1	BOOTLOADER_TIMEOUT	Bootloader timeout reached	0x00000002
9	BSETTINGS_CHANGED	Bootloader flags have changed (new firmware or firmware validation was done)	0x00000200
16	RECEIVING_NEW_FIRMWARE	New firmware is being received	0x00010000
24	NEW_FIRMWARE_RECEIVED	New firmware reception was completed	0x01000000

Bit	Name	Description	Mask (hex)
25	NEW_FIRMWARE_CHECK_OK	New firmware check was ok	0x02000000
26	NEW_FIRMWARE_CHECK_BAD	New firmware check was not ok	0x04000000

Table 5: Bootloader status register for P1 devices

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	None	None

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	Status of the positioner	None

5.2.4 Start firmware upgrade (200)

This command will initiate the new firmware upgrade process. It must be sent with 8 bytes of data. The first 4 bytes correspond to the size of the of the image to be sent, and the second 4 bytes correspond to the checksum of the image to be sent. The checksum is a CRC32 checksum compatible with zlib checksum calculation. The positioner will respond with a command accepted code (0), except if the firmware size sent is too large, in which case it will return an out of range code (1).

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	Size of firmware	Checksum of firmware

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	None	None

5.2.5 Firmware data (201)

This command should contain 8 bytes of data, except if it is the last chunk of the image, in which case it should only contain the appropriate number of bytes (from 1 to 8). The positioner will respond with a command accepted code. Once the full firmware image has been transmitted the positioner will calculate the checksum and if the results correspond to the previously sent checksum it will use the new firmware has main application.

In case of failure of the new firmware to boot, a watchdog will reset the positioner after 30 seconds and the positioner will revert back to the previous version of the firmware. If the main firmware boots correctly, next time the bootloader is entered, it will create a backup copy of the actual firmware to be able to revert back to the latest functional version in case of failure during the next upgrade.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	Firmware data bytes 3 to 0	Firmware data bytes 7 to 4

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	None	None

5.3 Main application V04.01.13 communication commands (P1 devices)

Table 6 summarizes the available commands for positioners with firmware V04.01.13.

Cmd	Description	Broadcast	Command extra data	Answer extra data
1	Get ID	Yes	None	Positioner ID
2	Get firmware version	Yes	None	Main firmware version
3	Get status	Yes	None	Status register
10	Send new trajectory	No	Number of trajectory points	None
11	Send trajectory data	No	Position and time	None
12	Trajectory data end	No	None	None
13	Trajectory abort	Yes	None	None
14	Start trajectory	Yes	None	None
15	Stop trajectory	Yes	None	None
18	Fatal error collision	No	Send by robot	Alpha or Beta
20	Go to datums	No	None	None
21	Go to datum alpha	No	None	None
22	Go to datum beta	No	None	None
23	Calibrate datum	No	None	None
24	Calibrate datum alpha	No	None	None
25	Calibrate datum beta	No	None	None
26	Calibrate motor	No	None	None
27	Calibrate motor alpha	No	None	None
28	Calibrate motor beta	No	None	None
29	Get datum calibration error	No	None	Alpha and Beta error
30	Go to absolute position	No	Positions	Time to complete move
31	Go to relative position	No	Positions	Time to complete move
32	Get current position	No	None	Current positions

Cmd	Description	Broadcast	Command extra data	Answer extra data
33	Set current position	No	Positions to set	None
34	Get offsets	No	None	Alpha and beta offset
35	Set offsets	No	Alpha and beta offset	None
40	Set speed	No	Speeds	None
41	Set current	No	Current	None
44	Get hall current position	No	None	Hall positions
45	Get motor calibration error	No	None	Alpha and beta error
47	Calibrate cogging torque	No	None	None
48	Calibrate cogging alpha	No	None	None
49	Calibrate cogging beta	No	None	None
53	Save calibration data	No	None	None
54	Get position and current alpha	No	None	Alpha position and current
55	Get position and current beta	No	None	Beta position and current
56	Get currents	No	None	Alpha and beta currents
57	Get position and torque alpha	No	None	Alpha position and torque
58	Get position and torque beta	No	None	Beta position and torque
59	Get torques	No	None	Alpha and beta torque
106	Get cogging vector length	No	None	Cogging vector length
107	Get cogging value positive	No	Cogging indices	Cogging values at indices
108	Get cogging value negative	No	Cogging indices	Cogging values at indices

Cmd	Description	Broadcast	Command extra data	Answer extra data
110	Get cogging angles	No	Cogging indices	Cogging angles at indices
111	Set collision margin	No	Alpha and beta collision margin	None
112	Set holding currents	No	Holding currents	None
113	Get holding currents	No	None	Holding currents
116	Switch ON hall for moving	Yes	None	None
117	Switch OFF hall for idle	Yes	None	None
118	Set Alpha closed loop and collision detection	No	None	None
119	Set Alpha closed loop and no collision detection	No	None	None
120	Set Alpha open loop and collision detection	No	None	None
121	Set Alpha open loop and no collision detection	No	None	None
122	Set Beta closed loop and collision detection	No	None	None
123	Set Beta closed loop and no collision detection	No	None	None
124	Set Beta open loop and collision detection	No	None	None
125	Set Beta open loop and no collision detection	No	None	None
126	Switch ON LED	No	None	None
127	Switch OFF LED	Yes	None	None

Cmd	Description	Broadcast	Command extra data	Answer extra data
128	Switch ON precise move alpha	No	None	None
129	Switch OFF precise move alpha	No	None	None
130	Switch ON precise move beta	No	None	None
131	Switch OFF precise move beta	No	None	None
132	Get raw temperature	No	None	Temperature

Table 6: Main application commands for P1 devices

5.3.1 Get ID command (1)

This command will return a frame containing 4 bytes of data representing the positioner ID as an unsigned integer of 32 bits.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	None	None

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	ID of the positioner	None

5.3.2 Get firmware version (2)

This command will return a frame containing 4 bytes of data representing the actual version of the firmware running on the device. The firmware versions are identified using semantic versioning number style with the format XX.YY.ZZ. The first received byte will be zero, the second one will correspond to XX, the third one to YY and the last one to ZZ. The bootloader firmware has YY has 80 to be distinguished from the main application firmware.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	None	None

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	Version number	None

5.3.3 Get status (3)

This command will return a frame containing 8 bytes of data as an unsigned integer of 64 bits representing the status register of the positioner. Table 7 describes the meaning of each bit, if a bit is not represented it means it is not used.

Name	Description	Mask (hex)
SYSTEM_INITIALIZATION	Internal system initialized	0x0000000000000001
RECEIVING_TRAJECTORY	Receiving a trajectory	0x0000000000000010
TRAJECTORY_ALPHA_RECEIVED	All coordinates for alpha trajectory received	0x0000000000000020
TRAJECTORY_BETA_RECEIVED	All coordinates for beta trajectory received	0x0000000000000040
LOW_POWER_AFTER_MOVE	Sets 0 current on alpha and 30 current on beta after move	0x0000000000000080
DISPLACEMENT_COMPLETED	Displacement completed	0x0000000000000100
DISPLACEMENT_COMPLETED_ALPHA	Alpha datum initialized	0x0000000000000200
DISPLACEMENT_COMPLETED_BETA	Beta datum initialized	0x0000000000000400
COLLISION_ALPHA	alpha arm collision detected	0x0000000000000800
COLLISION_BETA	beta arm collision detected	0x0000000000001000
CLOSED_LOOP_ALPHA	Positioner alpha arm in closed loop control	0x0000000000002000
CLOSED_LOOP_BETA	Positioner beta arm in closed loop control	0x0000000000004000
COLLISION_DETECT_ALPHA_DISABLE	detection of alpha arm collisions disabled	0x0000000000020000
COLLISION_DETECT_BETA_DISABLE	detection of beta arm collisions disabled	0x0000000000040000
MOTOR_CALIBRATION	Motor calibration mode active	0x0000000000080000

Name	Description	Mask (hex)
MOTOR_ALPHA_CALIBRATED	Alpha motor calibrated	0x0000000000100000
MOTOR_BETA_CALIBRATED	Beta motor calibrated	0x0000000000200000
DATUM_CALIBRATION	Datum calibration mode active	0x0000000000400000
DATUM_ALPHA_CALIBRATED	Alpha datum calibrated	0x0000000000800000
DATUM_BETA_CALIBRATED	Beta datum calibrated	0x0000000001000000
DATUM_INITIALIZATION	Datum initialization mode active	0x0000000002000000
DATUM_ALPHA_INITIALIZED	Alpha datum initialized	0x0000000004000000
DATUM_BETA_INITIALIZED	Beta datum initialized	0x0000000008000000
HALL_ALPHA_DISABLE	Alpha hall sensor disabled (is toggled on when in open loop without collision detection)	0x0000000010000000
HALL_BETA_DISABLE	Beta hall sensor disabled (is toggled on when in open loop without collision detection)	0x0000000020000000
COGGING_CALIBRATION	Cogging torque calibration mode active	0x0000000040000000
COGGING_ALPHA_CALIBRATED	Alpha cogging calibrated	0x0000000080000000
COGGING_BETA_CALIBRATED	Beta cogging calibrated	0x0000000100000000
ESTIMATED_POSITION	Position stored in non-volatile memory is estimated (there was a motion during sudden shutdown)	0x0000000200000000
POSITION_RESTORED	Position was restored from non-volatile memory	0x0000000400000000
SWITCH_OFF_AFTER_MOVE	switch off H-bridge after move, this overwrites LOW_POWER_AFTER_MOVE	0x0000000800000000
PRECISE_MOVE_ALPHA	Precise move alpha on/off	0x0000002000000000
PRECISE_MOVE_BETA	Precise move beta on/off	0x0000004000000000
SWITCH_OFF_HALL_AFTER_MOVE	Hall sensors are temporally switched off during idle	0x0000008000000000

Table 7: Positioner status register for P1 devices

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	None	None

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	Status of the positioner	Status of the positioner

5.3.4 Send new trajectory (10)

This command should contain 8 bytes of data, each 4 bytes correspond to an unsigned integer corresponding to the number of coordinates to be expected for alpha and beta trajectories respectively. If this command is sent a second time before the full number of coordinates have been sent, the positioner will restart all coordinates counters to zero and expect the full trajectories again. The maximum amount of coordinates for a trajectory is 1024, but the first coordinate will be calculated internally to the actual position and timestamp of 0, so that 1023 extra coordinates points can be sent.

This command cannot be issued if the positioner is in motion or if it has not been initialized. It will respond with a corresponding code depending on the status (already in motion or not initialized).

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	Alpha trajectory size	Beta trajectory size

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	None	None

5.3.5 Send trajectory data (11)

This command should contain 8 bytes of data, the 4 first bytes correspond to a position and the 4 last bytes to a timestamp. The positions are given in a signed 32-bit number. The LSB

corresponds to 1 / 2^30 of a complete rotation (360°), for example 90° is 90 / 360 * 2^30 = 268435456.

The time stamps are given as an unsigned 32-bit number corresponding to the time stamp at which the corresponding position should be achieved, each time step is 0.5 ms, for example 10 seconds timestamp is 10 / 0.0005 = 20000.

This command should be sent after command Send new trajectory. It should be sent as many times as the number of coordinates required for alpha and beta trajectories defined by command Send new trajectory. The first coordinates will be treated as alpha coordinates until the number of alpha coordinates is attained, and the following ones will be treated as beta coordinates.

The first coordinate of the trajectory that will be internally used is the actual position of the actuator and a time stamp of 0. This first coordinate is not considered in the number of coordinates transmitted with command Send new trajectory.

The response code of every coordinate sent should be checked as the positioner will perform a check of the position to make sure it is within motion bounds and of the speed to make sure it is not trying to go faster than the maximum speed allowed by the actuators. If any of the limits are exceed a response code of 1 (Value out of range) will be return, otherwise a 0 (Command accepted) will be returned.

Table 8 lists an example of the steps to follow to send a trajectory and initiate the motion.

Step	Command	Data	Description
1	10	3 and 4	Start a new trajectory with 3 coordinates for alpha and 4 for beta
2	11	134217728 and 10000	First alpha coordinate: angle 45° timestamp 5 seconds
3	11	268435456 and 20000	Second alpha coordinate: angle 90° timestamp 10 seconds
4	11	134217728 and 30000	Third alpha coordinate: angle 45° timestamp 15 seconds

Step	Command	Data	Description
5	11	268435456 and 20000	First beta coordinate: angle 90° timestamp 10 seconds
6	11	134217728 and 30000	Second beta coordinate: angle 45° timestamp 15 seconds
7	11	268435456 and 40000	Third beta coordinate: angle 90° timestamp 20 seconds
8	11	134217728 and 50000	Fourth beta coordinate: angle 45° timestamp 25 seconds
9	12	None	Validate the trajectory, check for response code to make sure trajectory has been fully received and is ok
10	14	None	Start trajectory

Table 8: Example of trajectory transmission for P1 devices

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	Position	Timestamp

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	None	None

5.3.6 Trajectory data end (12)

This command must be issued after transmitting a full trajectory to confirm that all the coordinates have been accepted and are valid. A return code will be sent back depending on the validity of the trajectories, either code 0 (command accepted) or code 2 (invalid trajectory).

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	None	None

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	None	None

5.3.7 Trajectory abort (13)

This command is used to stop the motion of the actuators. It will also reset the trajectories and stop any movement or calibration mode.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	None	None

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	None	None

5.3.8 Start trajectory (14)

This command is used to initiate the motion after sending and validating a trajectory.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	None	None

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	None	None

5.3.9 Stop trajectory (15)

This command is used to stop the motion of the actuators. It will also reset the trajectories, stop any calibration modes if active and clear collision flags. This command is equivalent to "5.3.7 Trajectory abort" except that "stop trajectory" also clears the collision flags.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	None	None

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	None	None

5.3.10 Fatal error collision (18)

This is not a command which can be send to the positioner. It is only a response send by the positioner in case a collision is detected. The positioner which detects a collision stops to move. All the other positioners continue to move unless a "stop trajectory (15)" or "trajectory abort (13)" is sent.

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid=0 and response code	None	None

5.3.11 Go to datums (20)

This command is used to find the zero reference at the hard stop, in case the position is lost (datum not initialized). The command sends both axes in a negative (clockwise as seen from the top) direction till the hard stops are detected. Note that the datum is found by detecting a collision at the harstops, **therefore it must be made sure that the positioner cannot collide with anything on its way to the hardstop**!

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	None	None

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	None	None

5.3.12 Go to datum alpha (21)

This command is used to find the zero reference at the hard stop, in case the position is lost (datum not initialized). The command sends the alpha axis in a negative (clockwise as seen from the top) direction till the hard stop is detected. Note that the datum is found by detecting a collision at the harstops, **therefore it must be made sure that the positioner cannot collide with anything on its way to the hardstop**!

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	None	None

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	None	None

5.3.13 Go to datum beta (22)

This command is used to find the zero reference at the hard stop, in case the position is lost (datum not initialized). The command sends the beta axis in a negative (clockwise as seen from the top) direction till the hard stop is detected. Note that the datum is found by detecting a collision at the harstops, **therefore it must be made sure that the positioner cannot collide with anything on its way to the hardstop**!

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	None	None

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	None	None

5.3.14 Calibrate datum (23)

This command is ideally only executed once of the positioner's lifetime. It calibrates the angle between stator windings and the hardstop. This ensures better repeatability of datum initializations. The command sends both axes in a negative (clockwise as seen from the top) direction till the hard stops are detected. Note that the datum is found by detecting a collision at the harstops, **therefore it must be made sure that the positioner cannot collide with anything on its way to the hardstop**! The calibration changes of this command will be lost after a power cycle if a "save calibration data (53)" command is not sent before switching off the positioner.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**

ID and command and command uid	None	None

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	None	None

5.3.15 Calibrate datum alpha (24)

This command is ideally only executed once of the positioner's lifetime. Same as "Calibrate datum" but only for the alpha axis. The calibration changes of this command will be lost after a power cycle if a "save calibration data (53)" command is not sent before switching off the positioner.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	None	None

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	None	None

5.3.16 Calibrate datum beta (25)

This command is ideally only executed once of the positioner's lifetime. Same as "Calibrate datum" but only for the beta axis. The calibration changes of this command will be lost after a power cycle if a "save calibration data (53)" command is not sent before switching off the positioner.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	None	None

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	None	None

5.3.17 Calibrate motors (26)

This command calibrates the hall sensors amplitudes and the angular offset between the magnetic coils and the hall sensors and is needed for correct operation of the closed loop control and collision detection. For that it moves both motors a few turns. The calibration changes of this command will be lost after a power cycle if a “save calibration data (53)” command is not sent before switching off the positioner.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	None	None

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	None	None

5.3.18 Calibrate motor alpha (27)

Same as “Calibrate motors” but only for the alpha axis. The calibration changes of this command will be lost after a power cycle if a “save calibration data (53)” command is not sent before switching off the positioner.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	None	None

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	None	None

5.3.19 Calibrate motor beta (28)

Same as “Calibrate motors” but only for the beta axis. The calibration changes of this command will be lost after a power cycle if a “save calibration data (53)” command is not sent before switching off the positioner.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	None	None

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	None	None

5.3.20 Get datum calibration error (29)

Get datum calibration error for both axes. Response is sent as int32 in degrees. When the calibrate datum (23) command is executed, the positioner drives into the hardstop three times. At each collision the motor position is recorded, and at the end the new zero is defined as the average of the recorded values + four motor turns. This command returns the difference between maximum and minimum recorded motor position in degrees.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	None	None

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	Alpha error	Beta error

5.3.21 Go to absolute position (30)

This command must contain 8 bytes of data, the first 4 bytes correspond to the absolute position for alpha arm and the second 4 bytes the absolute position for the beta arm. Refer to Send trajectory data (11) for details on position encoding. The positioner will move at the speed set by command Set speed (40). It will return a response containing 8 bytes of data. The first 4 bytes correspond to the time required to complete the alpha motion and the second 4 bytes to the time required to complete the beta motion, for details on time encoding, please refer to Send trajectory data (11).

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	Alpha target position	Beta target position

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**

ID, command, command uid and response code	Alpha motion completion ETA	Beta motion completion ETA

5.3.22 Go to relative position (31)

This command must contain 8 bytes of data, the first 4 bytes correspond to the relative position for alpha arm and the second 4 bytes the relative position for the beta arm. Refer to Send trajectory data (11) for details on position encoding. The positioner will move at the speed set by command Set speed (40). It will return a response containing 8 bytes of data. The first 4 bytes correspond to the time required to complete the alpha motion and the second 4 bytes to the time required to complete the beta motion, for details on time encoding, please refer to Send trajectory data (11).

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	Alpha target position	Beta target position

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	Alpha motion completion ETA	Beta motion completion ETA

5.3.23 Get current position (32)

This command will return 8 bytes of data, the first 4 bytes corresponding to the current alpha position and the last 4 bytes to the current beta position. Refer to Send trajectory data (11) for details on position encoding.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	None	None

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	Alpha position	Beta position

5.3.24 Set current position (33)

This command must contain 8 bytes of data. The 4 first bytes corresponding to a alpha position and the 4 last bytes to an beta position. The current mechanical position will be set as the corresponding sent positions. Refer to Send trajectory data (11) for details on position encoding. This command is an alternative to go to datums (20) in order to reinitialize the position after the position is lost. **Note that this command should in general not be used and if it is used, it should be followed up by a go to datum (20), when the positioner is close to its datums and no collision with neighbors can occur.** The risk of this command is to set the datum reference wrongly. For example, if the positioner is at position (10,10) deg and we set the position to (180,180) deg. If we now send the positioner to (0,0) it will crash into its hardstops. For a simple position shift (e.g. when the positioner has been turned in its mount) the set offset (35) command should be used.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	Alpha position	Beta position

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	None	None

5.3.25 Get offsets (34)

This command will return 8 bytes of data, the first 4 bytes corresponding to the current alpha offset and the last 4 bytes to the current beta offset. Refer to Send trajectory data (11) for details on position encoding.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	None	None

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	Alpha offset	Beta offset

5.3.26 Set offsets (35)

This command sets the offsets of both rotation axes. The offsets are in the same format as the send positions. Refer to Send trajectory data (11) for details on position encoding. An offset shifts the zero position with respect to the hardstop. This command allows to define the positioner (0,0) with respect to the FVC for the alpha axis and with respect to an arm-stretched-out configuration for the beta axis. (The offset should be set to α_{off} and β_{init} in Figure 1 in the SDSS-V Positioner Calibration and Validation document). The calibration changes of this command will be lost after a power cycle if a “save calibration data (53)” command is not sent before switching off the positioner.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	Alpha offset	Beta offset

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1

ID, command, command uid and response code	None	None

5.3.27 Set speed (40)

This command must contain 8 bytes of data. The first 4 bytes correspond to the alpha actuator speed in RPM at the motor level (not at the arm level), and the last 4 bytes to the beta actuator speed. This speed is used to move the positioner with Go to absolute position (30) and Go to relative position (31) commands. The positioner will check that the speed is not out of the limits set at factory.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	Alpha RPM speed (actuator)	Beta RPM speed (actuator)

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	None	None

5.3.28 Set current (41)

This command must contain 8 bytes of data representing two unsigned integers. The two integers correspond respectively to the alpha current and beta current in percentage. This command is mainly useful for testing new actuators at factory and should not be needed for normal use. The positioner will check that the current is not out of the limits set at factory. The current limit set at factory will depend on the assembled model of actuator.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	Alpha current in percentage	Beta current in percentage

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	None	None

5.3.29 Get hall current position (44)

This command will return 8 bytes of data, the first 4 bytes corresponding to the raw hall value of the alpha position and the last 4 bytes to the raw hall value of the beta position. Refer to Send trajectory data (11) for details on position encoding.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	None	None

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	Alpha position hall	Beta position hall

5.3.30 Get motor calibration error (45)

Get motor calibration error for both axes. Response is sent as int32 in degrees.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	None	None

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1

ID, command, command uid and response code	Alpha error	Beta error

5.3.31 Calibrate cogging torques (47)

This command calibrates the cogging torques of the motor. For that it moves both motors a few turns. The calibration can take up to 40min. Note that the cogging torque calibration is not essential for the use of the positioner but is allows for a better collision detection. The calibration changes of this command will be lost after a power cycle if a "save calibration data (53)" command is not sent before switching off the positioner. Note that the cogging torque is calibrated close at the hardstops. The positioner will automatically drive to the hardstop first, before calibration starts.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	None	None

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	None	None

5.3.32 Calibrate cogging alpha (48)

Same as "Calibrate cogging torques" but only for the alpha axis. The calibration changes of this command will be lost after a power cycle if a "save calibration data (53)" command is not sent before switching off the positioner. Note that the cogging torque is calibrated close at the hardstops. The positioner will automatically drive to the hardstop first, before calibration starts.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	None	None

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	None	None

5.3.33 Calibrate cogging beta (49)

Same as “Calibrate cogging torques” but only for the beta axis. The calibration changes of this command will be lost after a power cycle if a “save calibration data (53)” command is not sent before switching off the positioner. Note that the cogging torque is calibrated close at the hardstops. The positioner will automatically drive to the hardstop first, before calibration starts.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	None	None

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	None	None

5.3.34 Save calibration data (53)

This command saves the calibration data to the memory. After calibrating the positioner the save command should be sent, otherwise the calibration is lost after a power cycle. The following calibration parameters are saved: offsets, hall amplitudes, angle between hall sensor and coils, cogging torque data, low power current, angle between coils and harstop. Saving the

calibration data is necessary after sending on of the following commands at least before a power cycle:

- Calibrate datum (23)
- Calibrate datum alpha (24)
- Calibrate datum beta (25)
- Calibrate motor (26)
- Calibrate motor alpha (27)
- Calibrate motor beta (28)
- Calibrate cogging torque (47)
- Calibrate cogging alpha (48)
- Calibrate cogging beta (49)
- Set offsets (35)
- Set holding current (112)

Any other changes (deactivating collision detection, change in collision margin etc.) are not saved and are hence always lost after a power cycle.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	None	None

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	None	None

5.3.35 Get position and current alpha (54)

This command sends back the position and measured current (in percent) of the alpha axis. It can be used to probe position and current during movement. Refer to Send trajectory data (11) for details on position encoding.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**

ID and command and command uid	None	None

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	Position	Current

5.3.36 Get position and current beta (55)

This command sends back the position and measured current (in percent) of the beta axis. It can be used to probe position and current during movement. Refer to Send trajectory data (11) for details on position encoding.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	None	None

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	Position	Current

5.3.37 Get currents (56)

This command sends back the measured currents of both axes. It can be used to get measured currents during movement.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	None	None

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	Current alpha	Current beta

5.3.38 Get position and torque alpha (57)

This command sends back the position and commanded torque (in percent) of the alpha axis. It can be used to prove position and current during movement. Refer to Send trajectory data (11) for details on position encoding.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	None	None

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	Position	Torque

5.3.39 Get position and torque beta (58)

This command sends back the position and commanded torque (in percent) of the alpha beta. It can be used to prove position and current during movement. Refer to Send trajectory data (11) for details on position encoding.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	None	None

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1

ID, command, command uid and response code	Position	Torque

5.3.40 Get torques (59)

This command sends back the commanded torques (in percent) of both axes. It can be used to get the currents during movement.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	None	None

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	Torque alpha	Torque beta

5.3.41 Get cogging vector length (106)

This command sends back the length of the cogging vector.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	None	None

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	Cogging vector length	None

5.3.42 Get cogging value positive (107)

With this command we obtain the positive cogging value depending on the sent cogging vector index of both axes.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	Cogging vector index alpha	Cogging vector index beta

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	Cogging value positive alpha	Cogging value positive beta

5.3.43 Get cogging value negative (108)

With this command we obtain the negative cogging value depending on the sent cogging vector index of both axes.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	Cogging vector index alpha	Cogging vector index beta

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	Cogging value negative alpha	Cogging value negative beta

5.3.44 Get cogging angles (110)

With this command we obtain the angle of the cogging vector index of both axes. The angles are in degrees for one single turn of the electric motor (input of the gear).

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	Cogging vector index alpha	Cogging vector index beta

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	Angle alpha	Angle beta

5.3.45 Set collision margin (111)

The positioner can detect collisions. The sensitivity of the collision detection can be adjusted. For that a collision margin value can be sent for both axes. The value for the collision value can range between -30 and 30. Default settings are a margin of -5 for the alpha axis and 20 for the beta axis. This command can be useful if a positioner has more viscosity than others and produces often false positive collision errors. Note that a change in the collision margin is not saved after shutting down and has to be resend after every power cycle.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	Alpha collision margin	Beta collision margin

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**

ID, command, command uid and response code	none	none

5.3.46 Set holding current (112)

This command sets the holding current which is used during idle phase. Default values are 0 for the alpha axis and 30 for the beta axis. Higher holding currents lead to a higher power consumption during the idle phase. Changes of the holding current can be saved to the flash with the save command.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	Alpha holding current	Beta holding current

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	none	none

5.3.47 Get holding current (112)

Get the current holding current values.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	None	None

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	Alpha holding current	Beta holding current

5.3.48 Switch ON hall for moving (116)

Hall sensors must not be in sleeping mode for movements. This command has to be sent before positioners are used. It's used as a wake up call after the observation phase in idle. It takes about 5 seconds to wake up the hall sensors from sleep.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	none	none

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	none	none

5.3.49 Switch OFF hall for idle (117)

Once all positioners are in place, observation can start. This command switches off the hall sensor to save some power and reduce the heat load into the focal plane. Note that positioners cannot be switched off completely because of a remaining cogging torque in the beta motor the position is not guaranteed to micrometer accuracy if the positioners are without power.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	none	none

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	none	none

5.3.50 Set alpha closed loop with collision detection (118)

This command sets the alpha motor into closed loop with collision detection enabled. This is the default mode and enabled automatically after a power cycle.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	none	none

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	none	none

5.3.51 Set alpha closed loop without collision detection (119)

This command sets the alpha motor into closed loop with collision detection off. After a power cycle the default mode defined under command 118 (closed loop with collision detection) is loaded.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	none	none

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	none	none

5.3.52 Set alpha open loop with collision detection (120)

This command sets the alpha motor into open loop with collision detection enabled. After a power cycle the default mode defined under command 118 (closed loop with collision detection) is loaded.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	none	none

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	none	none

5.3.53 Set alpha open loop without collision detection (121)

This command sets the alpha motor into open loop with collision detection off. After a power cycle the default mode defined under command 118 (closed loop with collision detection) is loaded.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	none	none

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	none	none

5.3.54 Set beta closed loop with collision detection (122)

This command sets the beta motor into closed loop with collision detection enabled. This is the default mode and enabled automatically after a power cycle.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	none	none

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	none	none

5.3.55 Set beta closed loop without collision detection (123)

This command sets the beta motor into closed loop with collision detection off. After a power cycle the default mode defined under command 122 (closed loop with collision detection) is loaded.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	none	none

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	none	none

5.3.56 Set beta open loop with collision detection (124)

This command sets the beta motor into open loop with collision detection enabled. After a power cycle the default mode defined under command 122 (closed loop with collision detection) is loaded.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	none	none

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	none	none

5.3.57 Set beta open loop without collision detection (125)

This command sets the beta motor into open loop with collision detection off. After a power cycle the default mode defined under command 122 (closed loop with collision detection) is loaded.

Command sent:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID and command and command uid	none	none

Response received:

Header	**Integer/Unsigned integer 0**	**Integer/Unsigned integer 1**
ID, command, command uid and response code	none	none

5.3.58 Switch ON LED (126)

Switch ON led. The led blinks at start up and is then automatically switched off. This command allows to switch it on again.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	none	none

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	none	none

5.3.59 Switch OFF LED (127)

This command switches off the led. It is a broadcast command.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	none	none

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	none	none

5.3.60 Switch ON precise move alpha (128)

Switch ON precise move for alpha. See 7.4 for more details.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	none	none

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	none	none

5.3.61 Switch OFF precise move alpha (129)

Disables the precise move for alpha. The precise move is ON by default.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	none	none

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	none	none

5.3.62 Switch ON precise move beta (130)

Switch ON precise move for beta. See 7.4 for more details.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	none	none

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	none	none

5.3.63 Switch OFF precise move beta (131)

Disables the precise move for beta. The precise move is ON by default.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	none	none

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	none	none

5.3.64 Get raw temperature (132)

The microcontroller has a temperature sensor. This command will give the raw temperature measurement unsigned integer of 32 bits.

Command sent:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID and command and command uid	none	none

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid and response code	Raw uC temperature	none

6 Syncline

The syncline can be used to start the trajectories on all positioners. This command is equivalent to starting the trajectory by CAN bus with “start trajectory (14)”. The advantage of the syncline is that the timing of all trajectory is better controlled over different CAN buses. In order to start a trajectory, the syncline is changed from ground to high. The syncline has to be held for a few milliseconds at high because it detects a rising edge on the syncline but then waits a short amount of time and checks again the syncline and only starts the trajectory if the syncline is still high. This is very important cause it filters out false positives when the syncline has some jitter on it. Then it starts the trajectory. The CAN response is the same as when starting a trajectory over the CAN line except with uid zero.

Response received:

Header	Integer/Unsigned integer 0	Integer/Unsigned integer 1
ID, command, command uid=0 and response code	None	None

7 Normal operation

During normal operation the positioners are used to reposition the optical fibers between observations.

7.1 Send Trajectory

Before repositioning every positioner has to receive its trajectory which ensures no collision between neighbors during the move. The trajectory is sent to each positioner individually with the above explained commands:

-Send new trajectory (10)
-Send trajectory data (11)
-Trajectory data end (12)

The trajectory is checked in the microcontroller so that it does not exceed the positioner performance. Note that a position at time zero does not have to be provided since the positioner will start the trajectory at its current position. The current position of the axes can be obtained with command:

-Get actual position (30)

7.2 Start Trajectory

Once all positioners are loaded with a valid trajectory, they are all started with either the broadcast command “Start trajectory (14)” or the syncline. Not that if any of the positioner does not acknowledge the start trajectory command or sends a collision detected response, it is not going to move and therefore all positioner should be immediately stopped with the broadcast command “stop trajectory (15)”. The end of the trajectories can be verified with the DISPLACEMENT_COMPLETED flag by using “Get status (3)”. Correct end positions can be checked with “Get actual position (30)”.

7.3 Correction iterations

In order to get micrometer precision, correction iterations are needed. First, we need to make sure that the output of the alpha and beta axis are not within the backlash of the commanded position. Every “Go to absolute position (30)” and “Go to relative position (31)” command performs a back and for movement which assures that the output position is not within the backlash. For this every approach ends with a 0.9 degrees movement in negative direction.

After the trajectory move a “Go to relative position (31)” with (0,0) actuation is commanded. This performs the zig zag move of 0.9 degrees and brings the positioner within 10-30 micrometer to the commanded target. Then the FVC measures all fibers and calculates the error Δr between commanded and measured positions ($\Delta r = r_{cmd} - r_{meas}$). For all Δr bigger than the required precision a correction move is needed. The correction move is obtained by calculating a new commanded position ($r_{cmd-new} = r_{cmd} + \Delta r$) and using the backwards kinematic to calculate new absolute (α, β) positions, which are then executed with a “Go to absolute position (30)” or “Go to relative position (31)” command. One to maximum three correction iterations should be enough to obtain micrometer (<5 micron) precision.

For further more detailed information: “Precision control of miniature SCARA robots for multi-object spectrographs, Journal of Optomechatronics“

7.4 Open vs Closed loop control vs Precise Movements

Both axes of the positioner are driven by three-phase brushless DC motors with a permanent magnet on the rotor. The motors are equipped with analog Hall sensors with which the orientation of the rotor can be measured over a full turn of the motor. The simplest control is to apply three synchronized PWM signals (GND-6V) on the phases of the motor in such a way

that when averaged over a short time a directional magnetic field is created. The permanent magnet naturally aligns itself with the applied field.

Open loop control:

With the open loop control a strong magnetic field is produced with a fixed strong intensity. In order to move the motor, the magnetic field is rotated. The rotor will follow the rotating magnetic field closely. This control has the advantage that the hall sensor is not needed due to the strong intensity of the field which ensures that the rotor doesn't slip. The disadvantage of this control is that it is very inefficient and consumes a lot of power whereas most of it is released as resistance heating. If the hall sensors are not used (open loop control with collision detection off), possible slippage of the rotor cannot be detected which leads to an error between actual and commanded output position (Not to mention the disabled collision detection). Therefore, the open loop control is intended to be used only as backup (e.g. broken hall sensors).

Closed loop control:

In the closed loop control not only the direction of the magnetic field, but also its intensity is controlled. The hall sensor provides position information of the rotor. The magnetic field is then set either plus or minus 90 degrees relative to the permanent magnet depending which direction the motor should move. The intensity is adjusted with a closed loop controller such that the optimal torque is generated. This control reduces the power consumption significantly. It also ensures that the rotor follows the commanded path much more closely. The disadvantage of this control is its dependency of the hall sensor.

Note that collision detection is available for both control methods and can be either enabled or disabled. The different modes can easily be changed with the commands 118-125 for both motors. Note that after a power cycle the default control is loaded (closed loop with collision detection enabled). The current control mode can be checked with "Get status (3)" (bits: COLLISION_DETECT_ALPHA_DISABLE, CLOSED_LOOP_ALPHA, COLLISION_DETECT_BETA_DISABLE, CLOSED_LOOP_BETA).

Precise movements:

Every "Go to absolute position (30)" and "Go to relative position (31)" command performs a back and for movement which assures that the output position is not within the backlash. For this every approach ends with a 0.9 degrees movement in negative direction. This means that for e.g. an absolute move (30) the positioner is first driven with a constant speed (defined with Set speed (40)) to the specified target +0.9 degrees with the default control (either closed or open

loop control). Then the control automatically switches to open loop control and the positioner is commanded to drive the 0.9 remaining degrees in negative direction at a constant speed. A relative position command (31) will do the same, but the end position will be shifted by the provided value. As an illustrative example the positioner is at 30 degrees and if the motor is commanded to 20 degrees with command (30) (which is equivalent to a -10 deg relative command (31)), the positioner will move to 20.9 degrees in closed loop first, stops and switches then to open loop to drives automatically from 20.9 to position 20 degrees. If the positioner is at 30 degrees and is commanded to absolute 40 degrees (equivalent to relative 10deg), then the positioner moves first to 40.9 degrees, switches to open loop and drives back to 40 degrees, in order to satisfy the negative approach. The precise approach move is executed in open loop to better satisfy repeatability for the correction iterations and to provide a better transition to the static holding torque for the idle phase. After the move the positioner transitions straight to the holding torque which is 0% for alpha and 30% for beta by default.

The precise movements can be switched off with commands (130) and (131) in case other homing strategies are used. However, the default homing strategy described under 7.3 is advised to be used for fast convergence.

For further more detailed information on the implementation of the control: “Precision control of miniature SCARA robots for multi-object spectrographs, Journal of Optomechatronics“

8 Calibration of motor parameters

There is a section in the flash reserved for the calibration data of the positioner. At start-up the firmware checks this section and if the data is available loads the parameters and initializes the correct flags. Without the calibration data, the flags indicate that the calibration has not been done yet, and therefore some functions which rely on this data are not available. The motor needs to calibrate the following things:

- Amplitudes of hall sensors
- Offset between hall sensor and magnetic coils
- Offset between hard stop and magnetic coils
- Cogging torque (improves performance but not essential for operation)

In order to calibrate the motor the following commands are send in following order.

- Calibrate motors (26)

- Calibrate datum (23)
- Calibrate cogging torques (47) (can take up to 40 min!!)
- Save calibration data (53) (saves the just calibrated data to flash):

After every command, you need to wait till it finishes with the calibration step. This can be done by verifying that the displacement completed and the corresponding calibration flag is set with command “Get status (3)”.

Note that this calibration should, in theory, only be done once at the beginning of the positioner's life. If you cut the power the calibration data is not lost if it has been saved before.

Difference between Calibrate datum (23) and Go to datum (20)
When the calibrate datum (23) command is executed, the positioner drives into the hardstop three times. At each collision the motor position is recorded, and at the end the new zero is defined as the average of the recorded values + four motor turns. The “get datum calibration error” gives you the difference between maximum and minimum recorded motor position in degrees. When the positioner loses its position, then it can be rezeroed by using the “go to datum” command, since it only hits into the hardstop once and it doesn't redefine the zero. It will only set the motor turns to the correct number. This ensures to find the exact previously defined zero as long as the datum is detected accurately within one motor turn.

9 Troubleshooting

Here we investigate possible reasons that a response code other than “command accepted” is received. Refer to table 3 for a complete list of response codes.

9.1 Value out of range or Invalid Trajectory

Values from commands such as “send trajectory”, “set current”, “go to position” and “set speed” are checked and if they exceed specified boundaries the response is a “value out of range” or “invalid trajectory” message. For a “send trajectory” and “start trajectory” command, the first point at time zero is the current position of the positioner. The first commanded position should lie within the allowed speed limits.

9.2 Already in motion

The positioner is already executing a move program. A move is completed when the "DISPLACEMENT_COMPLETED" flag is set, which can be checked with the "Get status (3)" command. Any active mode can be immediately stopped with the "Stop trajectory (15)" command.

9.3 Datum not initialized

The absolute position of both motors is being kept track of during operation in a volatile memory. To prevent position loss the positioner has the capability to detect when power is cut and to use the fading energy in the capacitors to save the current position in a non-volatile memory. At start up the position is retrieved from the memory. In rare cases it can happen that the position couldn't be stored at shut down or with an uncalibrated positioner. In that case the positioner loses its zero reference regarding the hard stop.

The command "Go to datums (20)" will send the positioner to both hardstops. As soon as the motor reaches the datum a collision is detected, and the position initialized. Note that with this command it has to be made sure that the positioner cannot collide with any objects on its way to the datums.

In case a straight-forward collision free path to the datums are not guaranteed. Two options exist.

- The "Go to relative position (30)" command can be used to "manually" bring the motors close to the hardstops by making sure that no collisions occur. Once close to the hardstops the "Go to datums (20)" is executed.
- If we approximately know the lost motor position, the positioner can be initialized with the command "Set current position (33)" The risk of this command is to set the datum reference wrongly. For example, if the positioner is at position (10,10) deg and we set the position to (180,180) deg. If we now send the positioner to (0,0) it will crash into its hardstops. A normal trajectory can now be used to bring the positioner back close to the hardstops and then send a "Go to datums (20)" to initialize the positioner correctly.

9.4 Collision detected

Here we describe how to resolve a collision detection.

9.4.1 Collision with obstacle

When a collision is detected both axes of the positioner stop. A visual check should be performed to verify if the positioner collided with an obstacle or its hardstops. In case of a collision, the collision flags are cleared with a "Stop trajectory (15)" command. The positioner should then be moved away from the obstacle by at least 2 degrees. This can be done with a "Go to relative position (31)", "Go to absolute position (30)" or "Start trajectory (14)". Note that the trajectory data should only be sent after the flags have been cleared, since the "Stop trajectory (15)" deletes the stored trajectory.

9.4.2 False collision detected due to increased friction

Several reasons are to be considered in case a collision is detected, when no actual collision (with obstacle or hardstops) happened. It might be the result, that the positioner has an increased viscosity within the gear. The torque needed to move a positioner can be modelled with a simple stiction and linear viscous friction model. We can visualize the motor current depending on the speed and see how this curve changed in the past, to give us an indication of change and increase in friction. Note that in temperatures below zero degrees C the viscosity of the positioner increases significantly and therefore the maximum motor speed should be limited to 2000 to 3000 rpm. For positioners with increased internal friction we can use the "Set collision margin (111)" command to increase the collision detection threshold. Default settings are a margin of -5 for the alpha axis and 20 for the beta axis. The maximum threshold is 30 for both axes. Note that a change in the collision margin is not saved after shutting down and must be resend after every power cycle. After setting an increased collision margin, the collision flags should be cleared and then positioner moved as explained above.

9.4.3 False collision detected due to broken hall sensor or motor phase

If increasing the collision margin doesn't resolve the problem and the positioner still detects collisions after clearing the flags with "Stop trajectory (15)", the problem might be a hardware problem. In order to investigate a hardware problem, we can set the malfunctioning axis in open loop control with collisions deactivated with command "Set alpha open loop without collision detection (121)" or "Set beta open loop without collision detection (125)". In open loop control the hall sensors are not necessary to move the positioners. We can now probe the value of the hall sensors with "Get hall current position (44)", then command a relative move (31) with low speed (1000rpm) of a few degrees and check the hall sensors again (44). We also must check visually if the malfunctioning axis moves after sending the relative move command.

- If the rotation axis moved but the hall sensor value did not change, most likely the hall sensors are faulty. The most probable cause is that one of the hall sensor wires detached.
- If the rotation axis did not move, the problem seems to be with the motor. Either a motor cable became lose, the h-bridge failed or there is so much friction that the positioner can't move.
- If the positioner does move and the hall sensor tracks the position accurately within +- 0.04 degrees of the current position (command (32)) everything should be working ok. However, as a last resort it is possible that the holding torque during the idle phase is not enough. The consequence is that the rotor position moves when the positioner is in idle and therefore, there is a big difference between hall sensor and commanded position after the idle phase, which leads to a collision detection at the start of the move. The holding torque can be increased with the command "Set holding current (112)".

Tthat the positioner can still be used with a defect hall sensor. The defect rotation axis can be used in open loop mode with either "Set alpha open loop without collision detection (121)" or "Set beta open loop without collision detection (125)". Note, that in this mode a collision detection is not available, and the power consumption is considerably higher during the moves.

9.5 Hall sensor disabled

This message can have two origins:

- A move ("trajectory start", "go to absolute" or "go to relative") cannot start after the command "Switch OFF hall for idle (116)" which temporally disables the hall sensors to save power. Therefore, the command "Switch ON hall for moving (116)" must be sent to enable the hall sensors again. It takes about 5 seconds to wake the hall sensors from sleep.
- For the calibration modes "motor calibration", "cogging calibration" and "datum calibration" the hall sensors must be working and switched on. For this both axes cannot be in open loop control with collision off. That means that these calibration modes are not available after sending the commands "Set alpha open loop without collision detection (121)" or "Set beta open loop without collision detection (125)".

9.6 Calibration mode active

A calibration mode is active. The command "Get status (3)" gives information on which calibration mode is active. Once the "DISPLACEMENT_COMPLETED" flag is active the positioner is ready to receive new move commands. Any active calibration mode can be immediately stopped with the "Stop trajectory (15)" command.

9.7 Motor not calibrated

Calibration of the cogging torque and datums requires first the calibration of the motors. The motors are calibrated by sending the “Calibrate motors (26)” command.

10 Operation at cold temperatures

Tests showed that the positioners have some issues at cold temperatures. At very cold temperatures many false collision detections occur due to the increased friction and problems with the hall sensors (9.4.2). However, increasing the collision margin is not solving the problem in this case. To guarantee the correct operation of the positioners in a very cold environment we propose two measures:

1) Limit the maximum motor speed to 3000 rpm at negative temperatures (requirement: 5000 rpm)
2) And in addition, heat the motors with an additional power of 0.7 watts (total idle power per heated positioner: 1.2 watts instead of 0.5 watts) at temperatures below -10 deg C.

The motors can be heated by increasing the holding torque during idle phase. The default holding torque during idle is 0% for alpha and 30% for beta motors. We propose to increase the holding torque to 45% for alpha and leave 30% for beta for temperatures below -10 deg C (command “Set holding currents (112)”).

11 Safe zones

This section gives the maximal error we can expect with a positioner fulfilling the specifications of the CFT. The thermal effects do not influence the arm lengths enough to have an impact on the safezone.

11.1.1 Error on arm lengths

Error on arm lengths	Alpha	Beta
Length tolerance [µm]	±100	±100
Circularity deviation [µm]	±15	±15
Thermal effects [µm][1]	-6; +3	-13; +6

Total error [µm]	−121; +118	−128; +121

[1]Assuming parts are made in aluminum at 20°C, the temperature change is [-40; +20] [K]

11.1.2 Error on arm angles

Error on arm angles	Alpha	Beta
Hysteresis [°]	±0.30	±0.30
Non-linearity [°]	±0.60	±0.60
Fiber torque [°]	±0.50	±0.50
Dynamic control error [°][1]	±0.10	±0.08
Total error [°]	±1.50	±1.48
Equivalent lateral error [µm][2]	±197	±848
Thermal effects [µm][3]	±0	−1; +1
Total lateral error	±197	±849

[1]The dynamic control error represents a 90° motor error.
[2]The equivalent length error ε_{lin} is calculated as follows:

$$\varepsilon_{\alpha,lin} = \sin(\varepsilon_\alpha) \cdot (l_\alpha + l_{\alpha,err})$$

And

$$\varepsilon_{\beta,lin} = \sin(\varepsilon_\alpha + \varepsilon_\beta) \cdot (l_\beta + l_{\beta,err})$$

With ε_α and ε_β the angular error on alpha and beta respectively, l_α and l_β the alpha and beta arm lengths (l_α=7.4mm and l_β=16.2mm, the arm end), $l_{\alpha,err}$ and $l_{\beta,err}$ the errors on the alpha and beta arm lengths.
[3]Assuming parts are made in aluminum at 20°C, the temperature change is [-40; +20] [K].

11.1.3 Total safe zone

The total safe zone is the stack up of the alpha error and the beta error.

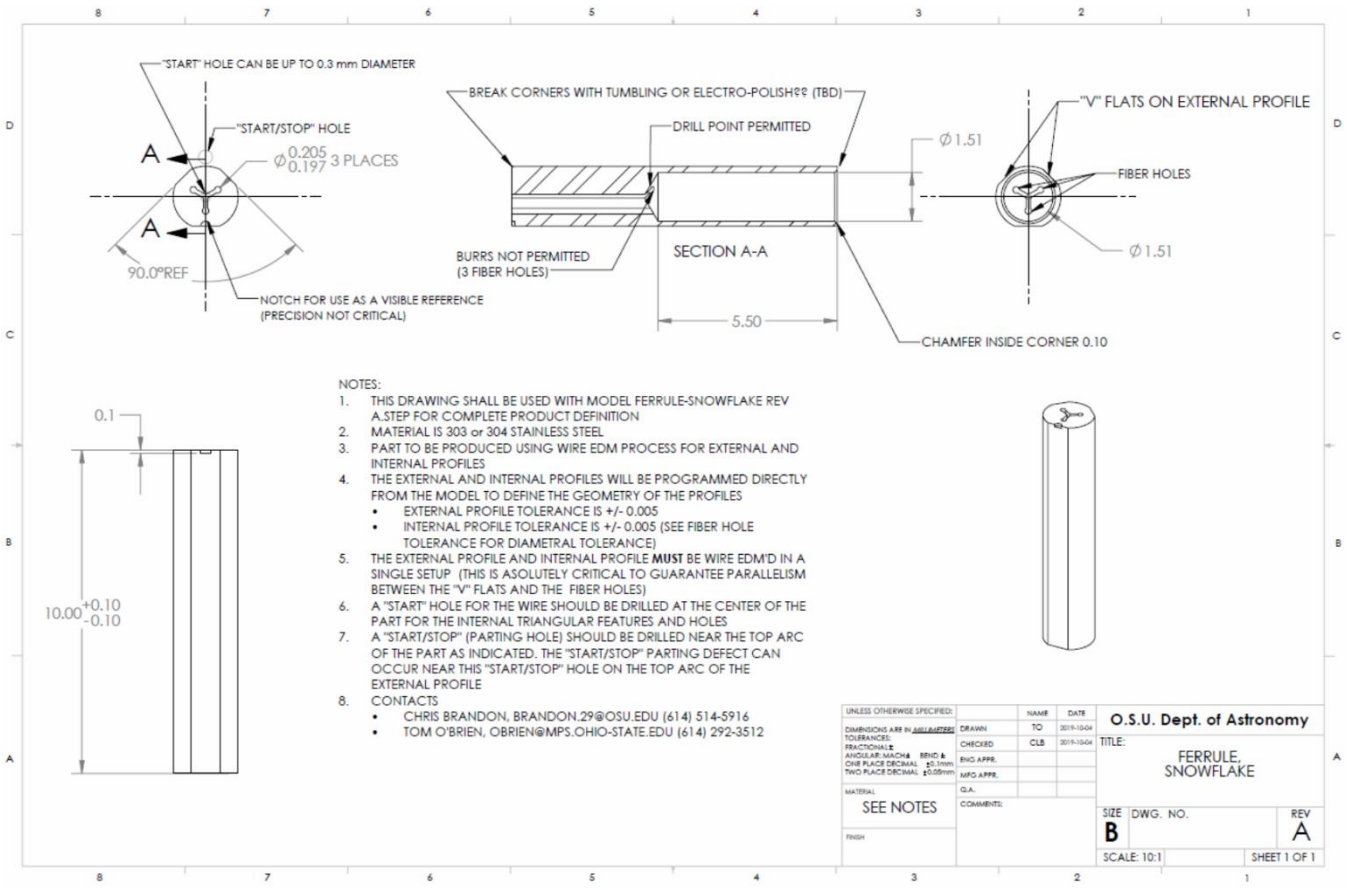
	Endpoint error
Maximal length error [µm]	−249; +239
Maximal lateral error [µm]	±1046

Note that this is the absolute maximal error for a positioner conforming to the specifications. As the values can change during the movement, it is recommended to always take the above specified safe zone, which is the maximal value.

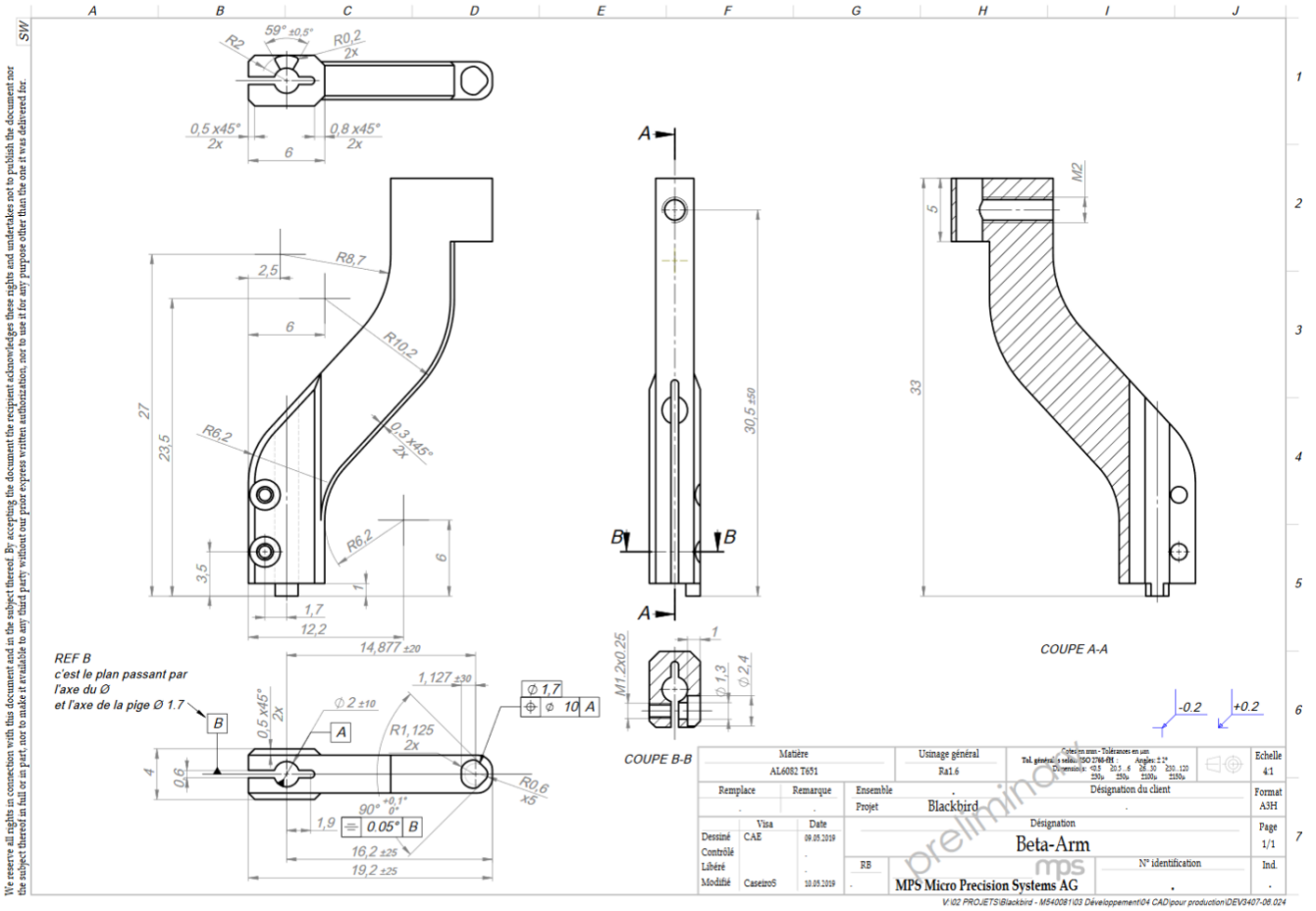
12 Appendix

12.1 Appendix: Beta arm drawing

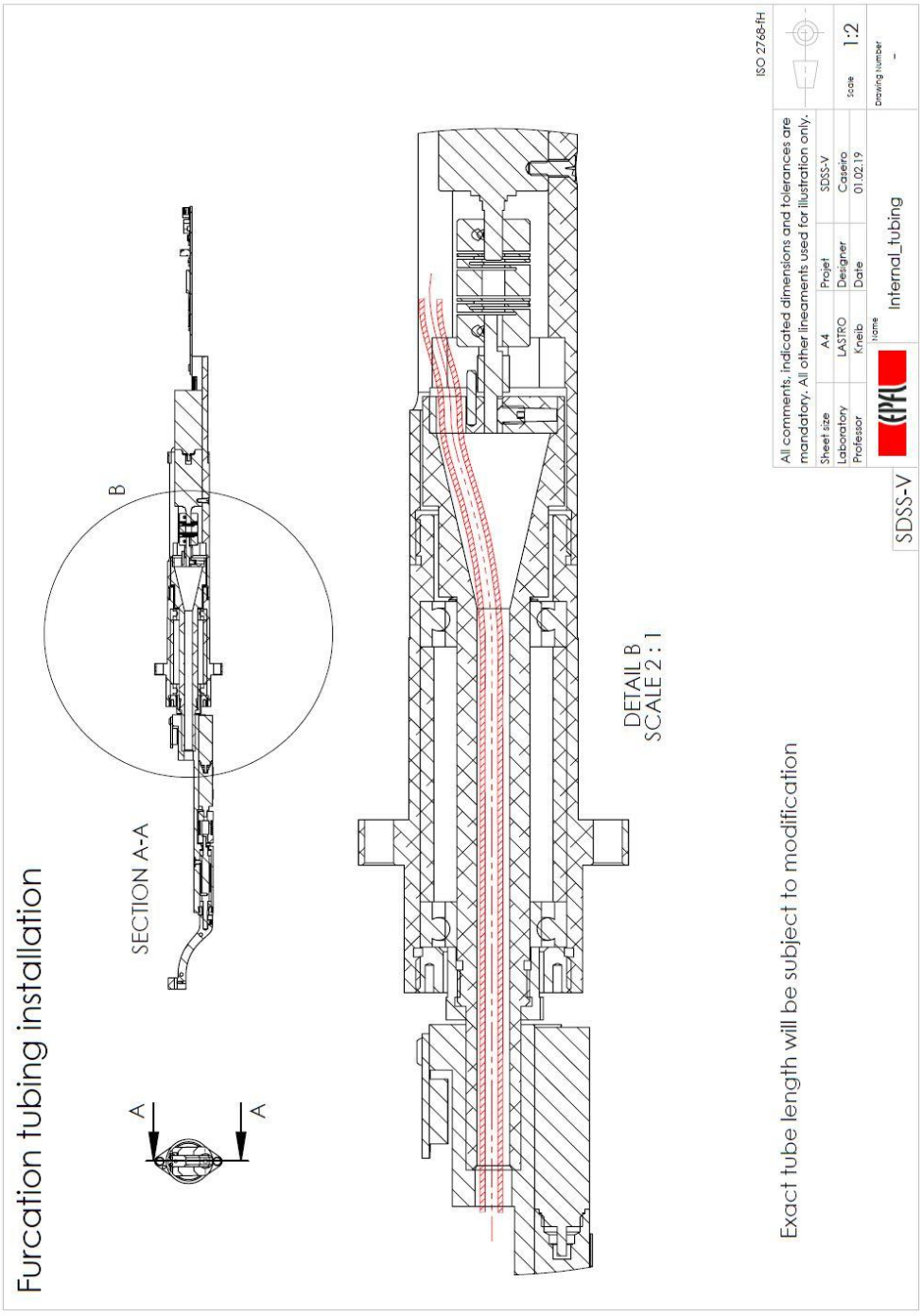
12.1.1 Beta arm ferrule interface



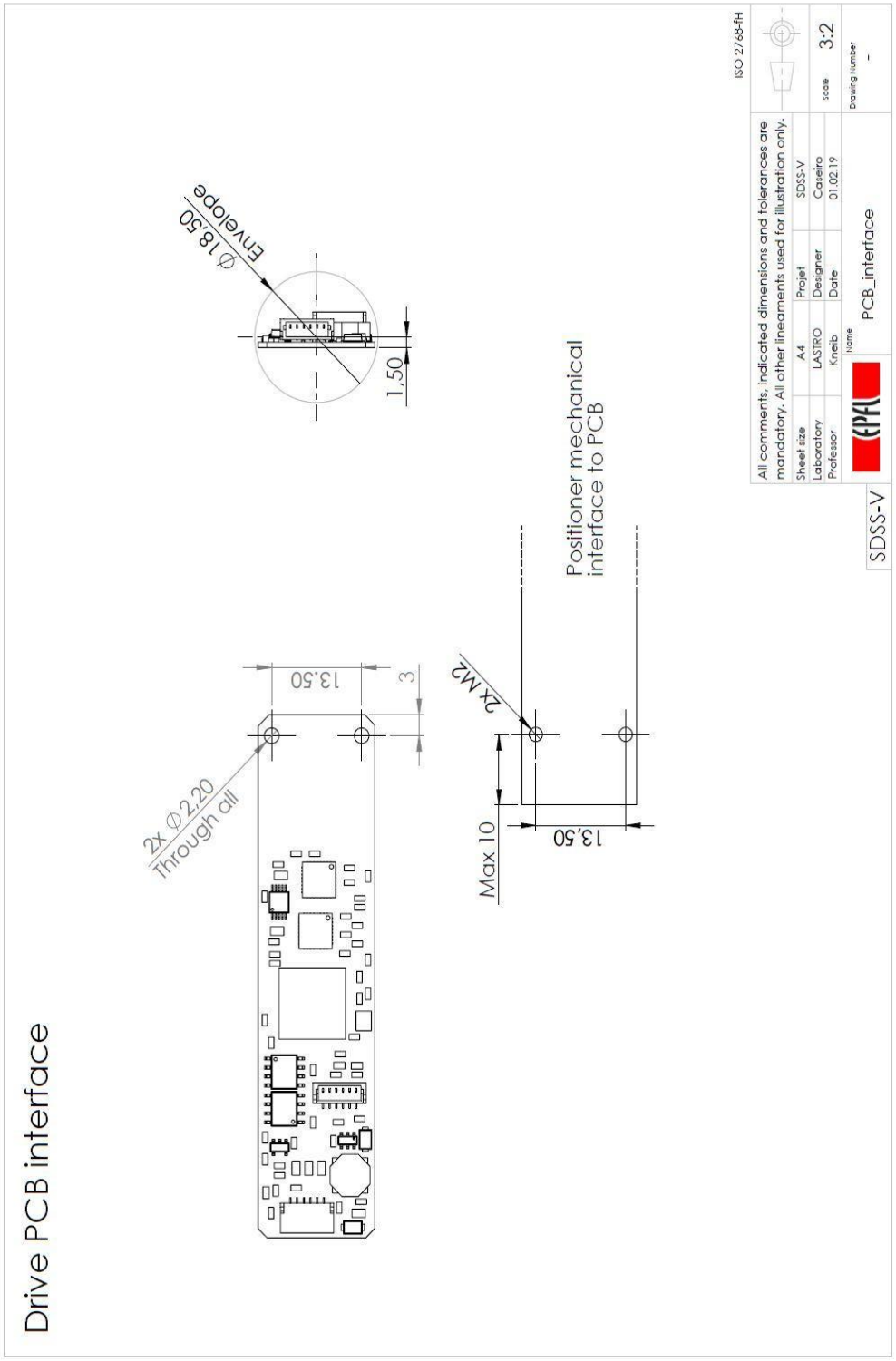
12.1.2 Beta arm envelope



12.2 Appendix: Furcation tubing installation



12.3 Appendix: Drive PCB interface



12.4 Appendix: Mechanical interface