

Add-ons to Adept SmartController Systems.

Documentation



*Last update:
March 2005.*

Table Of Content

1.Introduction.....	3
a.Adept Certified Developer.....	3
b.Cerebellum Automation.....	3
c.Add-ons.....	3
2.Licensing.....	4
3.Common interface.....	5
4.Specific interface.....	8
a.SSI.....	8
i.Wiring.....	8
ii.Software Setup.....	8
iii.Calibration.....	8
iv.Calibration setup.....	9
v.Diagnostic.....	10
b.S-Curve / E-Gearing.....	11
i.Common routines.....	11
ii.S-Curve.....	15
iii.E-Gearing.....	17
c.Serial port.....	18
i.Wiring.....	18
ii.configuration.....	19
iii.Operation.....	21

1. Introduction

a. Adept Certified Developer

The Adept Certified Developer (ACD) program allows you to access third-party system developers who specialize in Adept product integration. These developers have partnered with Adept because they possess in-depth knowledge and experience with Adept products.

b. Cerebellum Automation

Cerebellum Automation is a European leader in high quality custom servo solutions and embedded applications for high-end motion control in the robotics, medical, robotics, food, electronics and automotive industries.

Cerebellum Automation has been one of the first ACD to be certified and is allowed and capable of working on software layers as deep as embedded servo code, which means that solutions requiring high precision and very short reaction time can be developed to solve your needs. For mid- and high-volume customers, Cerebellum Automation offers Adept servo boards as standalone (i.e. without V+ programming) for a better software integration of the complete cell, thus lowering total cost of ownership.

c. Add-ons

Beside custom development, Cerebellum Automation has implemented standard features for Adept SmartController systems that are sold as “add-ons” to the Adept V+ environment. Currently Cerebellum Automation offers the following add-ons:

- SSI absolute encoder on sMI6
- **E-Gearing** and **S-Curve** for sMI6 and MotionBlox
- **Serial port** support for sMI6 and MotionBlox

Don't hesitate to contact a Cerebellum sales representative for the development of feature specific to your application.

2. Licensing

All Cerebellum Automation add-ons are included in these 2 files:

- CER_AUT.V2
- CERxxxxx.SRV (where xxxxxx represent the version number)

When you get these files, you can use them for free for a limited time to evaluate our software. More precisely, the number of boots is limited to 50 times. The key for the evaluation license is **0**.

To purchase a license, contact your Cerebellum sales representative with the security ID of the Adept SmartControllers along with the add-ons that you wish to use.

To get the security ID of the SmartController simply type, at the V+ prompt
ID

To activate the license, you need to pass it to the **cer.main()** routine described below. There is one license number for all features. Its value depends on the controller security ID and which Add-ons are activated.

3. Common interface

Because all Cerebellum Automation features need to be set up together, they all share the same entry point contained in CER_AUT.V2:

cer.main(check.crc, status, \$path, license, ssi[,], egear[,], rs[,])

Where:

- **check.crc** (optional) flag indicating whether to check the CRC of the accompanying servo code file (CERxxxxx.SRV). If it is set to TRUE and if a CRC error is detected it means that the servo file is corrupted and the setup is then aborted. However because the CRC checking takes about a minute and because a file corruption cannot permanently damage the system, it is acceptable to set **check.crc** = FALSE once the system is in production. Default value is FALSE.
- **status** (optional) standard V+ error code returned by the routine. Strictly negative means that an error occurred
- **\$path** (optional) path to the servo code file. Default is current directory.
- **license** (optional) a DOUBLE equal to the license number you received from Cerebellum Automation. This parameter is optional, if it is not supplied, the software is in evaluation mode
- **ssi[robot, motor]** (optional) 2-dimensional array indicating which motors of which robots use the SSI feature. For Belt axis use 63 for **robot** and the belt number for **motor**.
- **egear[robot, motor]** (optional) similar to ssi[robot, motor] except that it is used to indicate which motors are using the E-gearing or S-Curve feature.
- **rs[robot, motor]** (optional) similar to ssi[robot, motor] except that it is used to indicate which boards are using the RS232 serial ports. Robot and motor must designate any axis on the board where the serial line is connected.
- Similar arrays are added at the end of the routine as optional parameters as new features become available from Cerebellum Automation.

cer.main() is meant to be called at the very beginning of the startup sequence: either from the AUTOBOOT file or from the calibration file.

No Other task shall be running during cer.main execution.

Example:

To configure motors 2 and 3 of robot 1 with SSI encoders and be able to use the E-gearing feature on motors 1, 2 and 3 of robot 2 use the following code:

```
.PROGRAM cer.example(status)
; ABSTRACT: Example routine to configure Cerebellum Add-ons.
;
; INPUT PARM: none.
;
; OUTPUT PARM: status standard V+ error code
;
;* Copyright (c) 2004 by Cerebellum Automation, SAS.

    AUTO $path
    AUTO DOUBLE license
    LOCAL ssi[,], egear[,], rs[,]

    $path = "NFS>XD:\CEREB\"           ;Where all the Add-on files reside
    MCS "DELETES @ ssi[,], egear[,]" ;Make sure it's undefined

    MCS "LOAD " + $path + "CER_AUT.V2" ;Attempt to load Cerebellum file
    status = ERROR(-1, 2)              ;Check for file loading errors
    IF status < 0 GOTO 100

    MCS "LOAD " + $path + "LICENSE.V2" ;Attempt to load file containing the
license
    status = ERROR(-1, 2)              ;Check for file loading errors
    IF status < 0 GOTO 100

    CALL get.license (license)

    ; Which motor use an SSI encoder on an sMI6
    ssi[1, 2] = TRUE                   ;ssi[robot, motor]
    ssi[1, 3] = TRUE

    ; Which motors will be using electronic gearing or S-Curve trajectory
    egear[2, 1] = TRUE                 ;egear[robot, motor]
    egear[2, 2] = TRUE
    egear[2, 3] = TRUE

    ; Which boards use the serial ports
    rs[1, 1] = TRUE                   ;The board to which robot 1 (motor 1) is connected
    rs[2, 1] = TRUE                   ;The board to which robot 2 (motor 1) is connected

    CALL cer.main(FALSE, status, $path, license, ssi[,], egear[,], rs[,])

    IF status < 0 THEN
        TYPE "an error occurred during Cerebellum Automation's add-on configuration:"
        TYPE status, $ERROR(status)
        GOTO 100
    END

100 MCS "DELETER @ ssi[,], egear[,]"
    MCS "DELETES a.cer_aut"
    MCS "DELETES get.license"

    RETURN
.END
```

in license.v2:

```
.PROGRAM get.license(license)
  license = ^H12345678
.END
```

4. Specific interface

a. SSI

The SSI (Synchronous Serial Interface) feature allows connecting SSI encoders or drives with SSI emulation to the Adept sMI6.

As of today, this feature has been tested with the following drives:

- Bosch-Rexroth Indramat, Ecodrive serie.
- Kollmorgen ServoStar serie.

i. Wiring

The SSI encoders use the same pins on the sMI6 as the standard incremental encoders. The Data+ and Data- signals from the drive or encoder connect to the A+ and A- signals. The Clock+ and Clock- signals from the drive or encoder connect to the index lines I+ and I-.

Refer to the sMI6 manual for detailed information about the location of these pins.

ii. Software Setup

There is no special setup beside the initial configuration through `cer.main()` as described above and the calibration, as described below.

The SSI input is by default set to receive Gray code, on 25 bits, with no parity bit and a clock frequency of 384 kHz (other configurations available on demand).

iii. Calibration

Because SSI encoders are absolute, the calibration is straight forward, it only consists in off-setting the zero of the encoder so that it corresponds to the kinematical zero of your robot. This is done automatically during `cer.main` for the encoders that are configured to be SSI (with the `ssi[,]` array).

However, if you need to re-calibrate the robot or do your own specific calibration routine, here is a sample calibration routine:

```

.PROGRAM cer.ssi.cal(mtr, status)

    AUTO abs_homepos, encsgn, abs_pos, rbt_pos

    abs_homepos = 0
    status = 0

    CALIBRATE 1 ;Load and initialize calibration utility

; Read absolute location at zero position (calibration offset)

    CALL ca.ams(, mtr, -ca.hompos, , abs_homepos, status)
    IF status < 0 GOTO 100

; Read absolute current position.

    CALL ca.ams(, mtr, -ca.absencpos, , abs_pos, status)
    IF status < 0 GOTO 100

; Compute the current location of the joint.

    CALL ca.ams(, mtr, -ca.encsgn, , encsgn, status) ;Encoder sign
    IF status < 0 GOTO 100
    rbt_pos = SIGN(encsgn) * abs_pos + abs_homepos

; Set the position of the robot to its actual current location.

    CALL ca.ams(, mtr, ca.cmd_here, rbt_pos, , status)
    IF status < 0 GOTO 100

; wait for the servo status packet that has the new position.

    WAIT
    WAIT

; Restore V+ control of motors BEFORE checking for errors or clearing NOT.CAL bits.

    SETDEVICE (3, mtr, status, 2) 1 ;Restore V+ control
    IF status < 0 GOTO 100

100 RETURN

.END

```

This routine is available in CER_AUT.V2 and applies to the currently SELECT'ed robot. It's source is also available in SSI.CAL, which can be used like standard.cal as a robot calibration file. To do this, you simply need to copy SSI.CAL in the \CALIB\ directory and change the calibration file name in SPEC to "ssi.cal". This way, the V+ CALIBRATE command will calibrate the SSI encoder transparently.

iv. Calibration setup

Before the calibration can be used as above, one needs to set the absolute position of the encoder at the kinematical zero of the robot (i.e. the calibration offset).

To do this, you need to activate the SSI add-on by calling `cer.main()` and manually place the robot at what you want to be the zero location or any other known position (`current.pos`). Then execute the routine `cer.ssi.setup(motor, robot, status, current.pos)` included in `CER_AUT.V2` for each motor that needs the SSI calibration setup. `current.pos` is optional and assumed to 0 if not supplied.

Note that if 2 motors are **coupled** to the same joint, you will need to call `cer.ssi.setup()` twice for the motor that influence the position of only one joint: once before the other motor, so that the correct position is used and once after, because changing the position of the other motor also change the position of this motor.

v. Diagnostic

You can use the standard SPEC diagnostic menu to see if you get a valid position. The index information, doesn't apply.

Note that some drives update their position at a lower rate than the 8khz servo loop rate of the sMI6. Therefore the velocity signal can be quite noisy if you log it. However because of the internal filter in the smi6 servo code, this has been verified to have no influence on the overall performance of the system.

b. S-Curve / E-Gearing

The Electronic Gearing add-on, allows defining the motion relative to another motor or external encoder. The same license also enables the use of S-Curve trajectory outside of V+ control.

These 2 features use the same parameters for their trajectory:

i. Common routines

• CALL cer.enable (mot.num, enable, error)

- **Function**

This program enables/disables ASYNC mode for the specified motor.

- **Input Parameters**

mot.num Motor number; applies to the currently SELECT'ed ROBOT
enable If TRUE, enable ASYNC mode for the specified motor. If FALSE, disable ASYNC mode for the specified motor.

- **Output Parameter**

error Standard success/error code

- **Details**

ASYNC mode must be enabled before any Cerebellum routine that requires ASYNC mode can be run. ASYNC mode must be disabled for normal V+ motion operations to continue. ASYNC mode is enabled/disabled on a motor-by-motor basis.

While a motor or motors are in ASYNC mode, V+ will continue to plan a motion for the motor(s). However, motion control is left to the application program using the Cerebellum routines. The remaining motors on the robot will continue to function normally. That is, the robot can be commanded to move using the standard V+ motion commands.

ASYNC mode is typically required for Cerebellum routines that attempt to move a motor: cer.sdrive (for S-Curve), cer.egear (for electronic-gearing)

It is acceptable to put a motor in ASYNC mode during a motion, as long as a new Cerebellum routine (cer.sdrive or cer.egear) is called right after to take over the trajectory generation. However V+ should not be currently executing a trajectory for a motor that is switch out of ASYNC mode with cer.enable(mtr, FALSE, error)

NOTE: ASYNC mode can also be disabled by issuing the V+ instruction DISABLE POWER. Subsequent power-up of the robot will enable SYNC mode.

- **CALL cer.max.vel (mot.num, max.vel, error)**

- **Function**

This program sets the maximum velocity for electronic-gearing and S-Curve for the specified motor. This program can be called at any time, even when the motor is moving, the parameter will be taken into account immediately while still respecting the constraints of a 3rd order trajectory.

- **Input Parameters**

mot.num Motor number; applies to the currently SELECT'ed ROBOT
Max.vel Maximum velocity in mm/s or deg/s

- **Output Parameter**

error Standard success/error code

- **CALL cer.max.accel (mot.num, max.accel, error)**

- **Function**

This program sets the maximum acceleration for electronic-gearing and S-Curve for the specified motor. This program can be called at any time, even when the motor is moving, the parameter will be taken into account immediately while still respecting the constraints of a 3rd order trajectory.

- **Input Parameters**

mot.num Motor number; applies to the currently SELECT'ed ROBOT
Max.accel Maximum acceleration in mm/s² or deg/s²

- **Output Parameter**

error Standard success/error code

- **CALL cer.max.decel (mot.num, max.decel, error)**

- **Function**

This program sets the maximum deceleration for electronic-gearing and S-Curve for the specified motor. This program can be called at any time, even when the motor is moving, the parameter will be taken into account immediately while still respecting the constraints of a 3rd order trajectory.

- **Input Parameters**

mot.num Motor number; applies to the currently SELECT'ed ROBOT
Max.decel Maximum deceleration in mm/s² or deg/s²

- **Output Parameter**

error Standard success/error code

- **CALL cer.ramp.time (mot.num, ramp.time, error)**

- **Function**

This program sets the acceleration ramp time for electronic-gearing and S-Curve for the specified motor. This program can be called at any time, even when the motor is moving, the parameter will be taken into account immediately while still respecting the constraints of a 3rd order trajectory.

- **Input Parameters**

mot.num Motor number; applies to the currently SELECT'ed ROBOT
Max.accel Acceleration ramp time in seconds

- **Output Parameter**

error Standard success/error code

- **Details**

This parameter controls the maximum jerk (the derivative of the acceleration or the change rate of the acceleration). It controls the slope of the acceleration and deceleration so that the trajectory takes this amount of time to reach the maximum acceleration. If the trajectory is too short for the maximum acceleration to be reached, the actual ramp time will be shorter than the one specified here, however the slope will remain the same.

Note also that if you change the maximum acceleration without changing the ramp time, the actual max jerk will change as well and it is the jerk that represents the hardness on the mechanic. However, we preferred to use the ramp time as the parameter to control the jerk as it is easier to get a representation of what it means: it is easier to have a notion of time than of mm/s^3 .

ii. S-Curve

S-Curve trajectory typically replaces and enhance the Advanced Servo Library routine `sl.drive()`. By using a **3rd order trajectory** (no step in acceleration, which means that the jerk is always limited), the motion is much smoother and the life of the mechanic is greatly improved.

`Cer.sdrive()` or `sl.drive()` are usually used to control a motor independently from its kinematic module.

However, another big advantage of this add-on is that it can be started during a motion. So if during a motion from point A to point B, a sensor indicates you that you should actually mode to point C first, your program can immediately call `cer.sdrive()` to **alter the trajectory** so that its target position is point C. This new trajectory will be computed on the fly and comply to the max velocity, max acceleration, max deceleration and max jerk constraints.

• **CALL `cer.sdrive (mot.num, position, absolute, error)`**

• **Function**

This program sets the acceleration ramp time for electronic-gearing and S-Curve for the specified motor.

The indicated motor must be in ASYNC mode (see `cer.enable`).

• **Input Parameters**

<code>mot.num</code>	Motor number; applies to the currently SELECT'ed ROBOT
<code>position</code>	Position or distance in mm or deg to move to or by.
<code>absolute</code>	If TRUE, then the previous argument is an absolute position to move to. If FALSE, then the previous argument is a distance relative to the current position to move.

• **Output Parameter**

<code>error</code>	Standard success/error code
--------------------	-----------------------------

• **Details**

This routine starts to move a motor out of V+ control following an S-Curve trajectory. It can be called in any state of the motor (moving with a V+ MOVE, moving with another `cer.sdrive` or still), a smooth trajectory to the new destination will be computed, respecting the max vel, max accel, max decel and max jerk constraints.

- **CALL cer.break (mot.num, error)**

- **Function**

This routine blocks program execution until the trajectory for the specified motor has completed and the actual position is within tolerance of the commanded position.

- **Input Parameters**

mot.num Motor number; applies to the currently SELECT'ed ROBOT

- **Output Parameter**

error Standard success/error code

- **Details**

This routine is almost equivalent to sl.break except it also checks that the axis is IN TOLERANCE.

- **CALL cer.stop (mot.num, error)**

- **Function**

This program commands the specified motor to come to a stop. The stopping trajectory will bring the motor to a null velocity as fast as possible, but still respecting the max deceleration and max jerk constraints.

- **Input Parameters**

mot.num Motor number; applies to the currently SELECT'ed ROBOT

- **Output Parameter**

error Standard success/error code

- **Details**

This routine will return immediately, however no new calls to cer.sdrive() will be accepted until the motor has come to a stop. To check whether the motor has stopped yet, use cer.break()

iii. E-Gearing

To engage an electronic gearing an S-Curve trajectory is computed so that the engaging phase is as smooth as possible while still time optimal.

- **CALL cer.egear (mot.num, master, gear.ratio, offset, absolute, error)**

- **Function**

This program commands the specified motor to follow the motion of the master axis. The engaging trajectory will bring the motor to a velocity equal to that of the master times the gear ratio as fast as possible, but still respecting the max deceleration and max jerk constraints.

In summary, the position of the slave motor is controlled with the following formula:

$$\text{slave.pos}(t) = \text{offset} + \text{gear.ratio} * \text{master.pos}(t)$$

The indicated motor must be in ASYNC mode (see cer.enable).

- **Input Parameters**

mot.num	Motor number; applies to the currently SELECT'ed ROBOT
master	Gadget number of the master axis (currently, it has to be on the same board as the slave motor). The master axis can be an external encoder or a motor. The motor can be under V+ control or controlled by a Cerebellum function.
Gear.ratio	Ration of slave velocity to master velocity. A ration of 0.5 means that the slave moves at half the speed of the master. Note that if you want to use a rational ration like 1/3 you should define this variable as DOUBLE. If it is negative, the 2 axis will move in opposite directions.
Offset	Offset in mm or deg (in slave space) between the position of the master and that of the slave. A positive offset means that the slave is ahead of the master
absolute	If TRUE, the offset is used between the absolute positions of the slave and the master. If FALSE, the offset is used from the current positions of the slave and master.

- **Output Parameter**

error	Standard success/error code
-------	-----------------------------

- **Details**

This routine will return immediately. The gearing link will remain until a new trajectory is commanded to the slave with cer.sdrive(), cer.stop() or and ASL routine.

c. Serial port

The Serial port add-on allows configuring the serial port interface and accessing it for read and write operations.

The serial ports have the same characteristics on all servo boards:

- they use a Dual UART that can be configured for either one port with RTS/CTS signals or 2 ports without RTS/CTS lines.
- They can be configured for any baud rate up to 115200 Baud (contact us if you need higher rate), data length, parity control, number of stop bits and software flow control.
- Full duplex

This add-on is used to provide **more serial ports** without adding any new hardware. They also offer **higher communication rate** than SmartController ports.

See the file example.v2 for an example of how to use the serial ports with the Cerebellum add-on

i. Wiring

- sMI6

The detailed documentation of the sMI6 can be found at :

www.adept.com/main/KE/DATA/Controller/SmartMotion_Installation/SmartMotion_IG.pdf

The pins for the serial communication are specified page 52. They are all on the Xencoder connector:

TX1:	sMI6 pin 29;	MP6-E Panel pin 49
RX1:	sMI6 pin 14;	MP6-E Panel pin 47
TX2 / CTS1:	sMI6 pin 44;	MP6-E Panel pin 50
RX2 / RTS1:	sMI6 pin 15;	MP6-E Panel pin 48

On the sMI6, the serial lines can also be configured as one RS-422 or one RS-485 line, contact us for more information.

- MotionBlox40 / sCobra

The serial ports are located on the RS-232 DSUB9 connector:

TX1:	pin 2
RX1:	pin 3
TX2 / CTS1:	pin 7
RX2 / RTS1:	pin 8

If you want to use the 2 serial ports at the same time, you can simply make a Y cable to separate the 2 serial lines

ii. configuration

In order to communicate (read and write) to a serial port, you first need to obtain a device ID, which will be used subsequently to address the desired serial port. This is done through the `cer.DUARTConfig()` routine, which is also used to define the configuration of the DUART (one port with hardware flow control or 2 ports without flow control).

To designate which port you want to address, you simply pass the axis and robot numbers of a motor that is on the same board as the serial port. This method, as compared to addressing through the serial number of the board as the advantage that the same software can be used on different cells or after a field replacement of an SMI6.

Once you have obtained a device ID, you can setup the port with `cer.comConfig()`

- **CALL `cer.DUARTConfig(robot, motor, comport, config, deviceID, error)`**

- **Function**

This program does the initial configuration of the DUART on the specified board and returns a device ID used to access other commands to this serial port.

- **Input Parameters**

robot	Robot number of a axis located on the board on which the serial port we want to access is located
motor	Motor number of a axis located on the board on which the serial port we want to access is located
comport	1 or 2 to designate which port on the servo board we want to address.
config	1 to configure the DUART for 1 serial port with RTS/CTS hardware flow control 2 to configure the DUART for 2 serial ports without hardware flow control Specifying a <i>comport</i> of 2 with a <i>config</i> of 1 is illegal.

- **Output Parameter**

DeviceID	A handle that will be needed to access the serial port through other routines
error	Standard success/error code

- **CALL cer.comConfig(deviceID, rate, data, parity, stop, xonxoff, error)**

- **Function**

This program does the configuration of the given serial port.

- **Input Parameters**

DeviceID A handle that is used to designate which serial port to configure.

rate Baud rate. Acceptable values are:

 2400

 9600

 19200

 28800

 38400

 57600

 115200(default)

data Data bits. Acceptable values are:

 5 : 5 data bits

 6 : 6 data bits

 7 : 7 data bits

 8 : 8 data bits (default)

parity Parity bit. Acceptable values are:

 0 : no parity (default)

 1 : odd

 2 : even

stop Stop bits. Acceptable values are:

 1 : 1 stop bit (default)

 2 : 2 stop bits for 6, 7 and 8 data bits ; 1.5 stop bit for 5 data bits

XonXoff Flow control. Acceptable values are:

 0 : no flow control (default)

 1 : XON / XOFF software flow control is turned on.

- **Output Parameter**

error Standard success/error code

iii. Operation

Operations on serial ports are simplified to writes with `cer.comwrite()` and reads with `cer.comread()` to read one buffer at a time or `cer.comGetC()` to read character by character

- **CALL `cer.comWrite(deviceID, $output, error)`**

- **Function**

This program writes a string of characters to the given serial port. This routine writes the string to the output buffer and returns to caller right away. This means that the transmission may not be completed by the time this routine returns.

- **Input Parameters**

DeviceID A handle that is used to designate which serial port to write to.

\$output The string to output on the serial port. No end of line character is added, so the user must explicitly write them in the string if his protocol requires it. (It can be useful to use the `V+ $ENCODE()` program instruction for this.

Keep in mind that in V+, a string is limited to 128 characters

Note that NULL characters (ASCII value of 0) will be discarded.

- **Output Parameter**

error Standard success/error code

- **CALL cer.comGetC(deviceID, c, error)**

- **Function**

This program writes a string of characters to the given serial port. This routine writes the string to the output buffer and returns to caller right away. This means that the transmission may not be completed by the time this routine returns.

cer.comGetC() cannot be used at the same time as Cer.comRead(), otherwise characters may be returned out of order.

- **Input Parameters**

DeviceID A handle that is used to designate which serial port to write to.

- **Output Parameter**

c The ASCII value of the next character available in the receiving buffer.
Characters are returned in the order they are received.
If no character has been received, c is equal to -1
If *parity* is not set to *none* and a parity error is detected for this character,
c is set to 255

error Standard success/error code

- **CALL cer.comRead(deviceID, \$input, error)**

- **Function**

This program writes a string of characters to the given serial port. This routine writes the string to the output buffer and returns to caller right away. This means that the transmission may not be completed by the time this routine returns.

Cer.comRead() cannot be used at the same time as cer.comGetC(), otherwise characters may be returned out of order.

- **Input Parameters**

DeviceID A handle that is used to designate which serial port to write to.

- **Output Parameter**

\$input A string representing up to 108 characters available in the receiving buffer.

If no character has been received, \$input is set to "" , the empty string.

If *parity* is not set to *none*, characters that have been received with a parity error are replaced by the ASCII value of 255

error Standard success/error code

Cerebellum Automation, SAS

Galiléo - Parc Altaïs
178, route de Cran-Gevrier
74650 Chavanod - FRANCE

Email: contact@cerebellum-automation.com

Web: www.cerebellum-automation.com

Tél.: +33 (0) 450 02 81 16

Fax: +33 (0) 450 52 07 42