Akribis-Agito Controllers

User's Manual

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Table of contents

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Additional Information

Additional information can be found within one of these additional manuals:

- Product's specific Hardware Manual.
- Akribis-Agito Controller Keywords Reference.
- PC Suite Manual.
- User Program Language Manual.

In case you fail to locate some required information within these manuals, or if given information do not answer your needs, please do not hesitate to contact Akribis and we will be glad to help (and to properly update the relevant manual).

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

How to Begin

Connecting power

Connect the power supply to the controller according to the detailed description at the relevant product's Hardware Manual.

Connecting communication

Connect the desired communication cable between the controller and the PC, according to the detailed description at the relevant product's Hardware Manual.

Installing the PC suite and connecting

Install Akribis's PC Suite according to the installation instructions that appear within the Akribis PC Suite Manual.

Use ThePC Suite Manual to properly define the communication channel and to open the communication channel (connect to the controller).

Now you shall be ready to work with the product. Of course, you will later need to connect the motor, encoder, safety (STO) and other connections to be able to create motions.

It is deeply recommended that you will read the PC Suite Manual, so you will be able to easily access all the parameters and functions that are referred to within this manual.

This manual does not necessarily describe each parameter keyword in details. You may find the detailed information (pear each keyword) at the Keywords and Communication Reference Manual.

For example, the MotorType parameter is described at the next chapter. Its possible values are provided, such as: DC Brush, Rotary DC Brushless and so on. But the relevant value to set for each option is not provided. Such detailed information can be found under the MotorType keyword page, within the Keywords and Communication Reference Manual.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Setup and Configuration

Motor information

The motor information must be set up correctly for the controller to function safely and correctly.

The main motor parameters are:

Motor Type (MotorType parameter keyword):

The motor type determines the way that the voltage is applied to the motor and how the feedback information is used.
The available motor types are:

- **Unknown**

  This is the default value of a new controller. No voltage will be applied to the power stage outputs before a different, valid motor type is set by the user.

- **DC Brush/ Voice Coil**

  If a DC brush motor type is set the full voltage is applied between two power terminals. See the hardware manual to find out which terminals to use. This type of motor is assumed to be rotary.

- **Linear/Rotary DC Brushless**

  The voltage is applied to three motor phases and commutation is used to determine the voltage that is applied to each motor power terminal. The difference between the types is actually in the configuration of the feedback and how motor poles are treated. The separate types are there to help with configuration in the PC suite.

  If the motor type is linear it is regarded as a single pole pairs motor. During feedback configuration the feedback information must be entered in a way that enables the controller to determine the ration between pole pairs and feedback reading.

  See the hardware manual for correct phase connection.

- **Simulation**

  This motor type is used for simulation during development. It allows generating simulated profiler, input and output behaviors without actually connecting a motor to the controller.

  With this motor type there is no voltage output to the power terminals. However, it is possible to set a motion mode and start simulated motion. The profiler related variables will change as if there is a motor. No faults will be generated. This is useful for development and demonstration.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Number of Pole Pairs (PolePrs parameter keyword):

This parameter is relevant for rotary brushless motors. This value, along with the feedback resolution, determine how the commutation of the phases is done. If the ratio between the motor pole pairs and the feedback resolution is wrong the commutation may not function correctly. The result of this can be anything from noisy or jumpy motion to severe damage to the motor.

Feedback setup

Akribis's controller supports a variety of position feedback types. Some of these feedback options may be supported only for some of the products and some require special production options. Please refer to the hardware manual to verify which feedback options are supported by your controller.

Basic encoder setup

The feedback setup for the main and auxiliary encoders are almost identical so there is no need for separate descriptions. In general, the same parameters are used for auxiliary encoder, with the prefix "aux" added (for example AuxEncDir instead of EncDir).

The supported feedback types (EncType parameter keyword) are:

Incremental Encoder

This is the "standard" encoder with A/B inputs and an optional index (Z) input. To configure this type of encoder the following parameters should be set:

- EncRes – The encoder resolution. For a rotary motor: The number of encoder counts per revolution. This value, together with the number of pole pairs per revolution is used for commutation.
- EncDir – The direction in which the encoder counts can be inverted for convenience.

Nikon Encoder

There are several types of encoders provided by Nikon. Use the hardware manual together with you encoder's data sheet to make sure your encoder is compatible with the controller. Using an incompatible encoder may result in unexpected behavior.
For 17 bit Nikon encoder set the following parameters:

- EncRes – For Nikon 17bit encoder, this should be set to 131072.
- EncDir – The direction in which the encoder counts can be inverted for convenience.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Advanced encoder setup

Modulus (or Modulo) Operation
Modulus mode is useful for rotary motors that move in the same direction for a long time as well as for cases where the same desired behavior depends on the position within a revolution.

ModRev is used for the modulus mode of the controller. In the modulus mode the main position (Pos) is between 0 and (ModRev-1). When the actual position exceeds (ModRev-1) the value of Pos is calculated to remain within the range of 0 to (ModRev-1). This allows a rotary axis to move in the same direction indefinitely without exceeding any numerical limits.
ModRev = 0 turns off the Modulo mode.

Notes:

- Do not use Modulo together with input shaping.
- If you manually set the position using SetPosition to a value that exceeds the range of ModRev unexpected behavior may occur.
- When using Modulo together with smoothing (Jerk not 0), the values of ModRev and Jerk should be set carefully. Refer to the Keywords and Communication Reference Manual, the relevant pages for Jerk or ModRev keywords, for detailed description of the limitations.

Encoder Emulation

A digital output can be assigned as encoder emulation. The encoder emulation output will generate pulses according to the main encoder position. The ratio between the number of pulses on the emulation output and the main encoder position is determined by EmulRat.

EmulIndexType determines the form of the pulse on the emulated encoder index output and how it is related to the main encoder index and pulses.

User Units

User units can be used for the user's convenience, to convert the values of the position and other values derived from it to units other than encoder counts. For example: if the user prefers the reading in mm, and there are 100 encoder counts per mm then UsrUnits = 100.

All the internal calculations and the control will use encoder pulses, so no actual resolution is lost. Only the user interface will be in user units for all the relevant parameters.

For a more detailed explanation about user units and how they are calculated please refer to the communication and keyword reference.

Auxiliary encoder

The auxiliary encoder setup is almost identical to the main encoder setup. See the main encoder setup description above.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Pulse/direction encoder

The pulse/direction input is another position input to the controller (can be used as a command).

Most of the setup is the same as the main encoder setup.
Sometimes it is necessary to multiply this input by a ratio (for example: if it is used for gearing).

PDFact and PDFactDen are used to determine this ratio.

PDFact multiplies the pulse/direction input counting before it is used.

PDFactDen divides the pulse/direction input counting before it is used. PDFact and PDFactDen together determine the resulting generated reference as follows:

Reference = $[Number\ of\ input\ pulses] * \frac{PDFact}{PDFactDen}$

PDFact can be positive or negative to enable a change in the direction of the signal, while PDFactDen is always positive.

Other configurations

You will need to configure also the following controller characteristics:

- Digitla I/O's.
- Analog I/O's.
- Safety.

This can be done using the PC Suite (Config menu). Please refer to later chapters within this manual for detailed explanation regarding the relevant parameters for each of these functions.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Motion Modes

The various motion modes determine what type of reference will be generated for the controlled motor. The reference can be generated internally to arrive to a desired location (point to point), or as a continuous constant speed (jog). The reference can also be generated from an external signal, using a dedicated pulse/direction input as reference or manipulating an input reference using an ECAM table or gearing.

The motion mode is determined at the beginning of the motion and remains the same until the motion ends. The beginning of the motion is when “Begin” is entered. The end of the motion can be reached when the motor arrives to its target (point to point), or by the user (“Stop”, “StopRep” etc.).

Some of the motion parameters can be changed during motion. The actual position and velocity can be queried in communication or recorded for later viewing. Other status reports are available to the user to help determine whether motion is in one of its normal states or an error occurred.

Note:

The descriptions of parameters in this document are only to provide good understanding of the functionality. For full descriptions of each parameter, its units and other attributes please refer to the communication and keywords document.

Multi-axis synchronization and Begin-On-Input

Using the parameter BeginDInOn, the user can define that a motion (at any of the Motion Modes described below) will not actually start when the Begin command is sent to the controller but only after a rising edge is detected at a specified (user-defined) discrete input (any of the optically isolated, or differential or Sync inputs).

Using this feature, a motion can be synchronized with an external event. By using the same signal as an input to multiple number of controllers, a multi-axes synchronized motion can be created.

This parameter and feature are supported only by some of Akribis’s controllers.

Please refer to the Keywords and Communication Reference manual for detailed description of the BeginDInOn keyword and how to use it to synchronize a multi-axes motion.

Point to Point

Point to point motion mode (MotionMode = 1) will generate a smoothed trapezoidal motion from the current location of the motor to the target position. The parameters that determine the shape of the profile are:

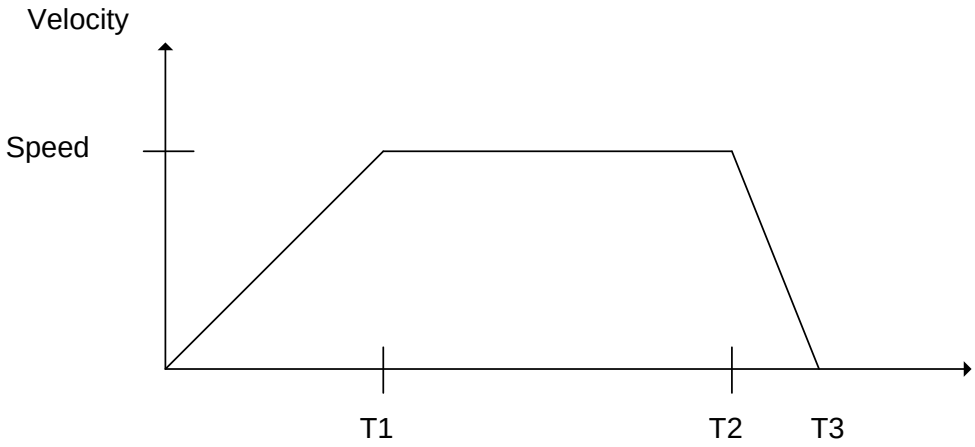
Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Parameter	Description
AbsTrgt	The absolute position target
RelTrgt	The position target relative to the current position
Speed	The desired speed in the constant speed part of the motion
Accel	The desired acceleration
Decel	The desired deceleration
Jerk	Determines the smoothing of the profile. 2^{Jerk} is the number of control samples that will be needed to reach the specified Accel or Decel (time to build the acceleration … which actually defines the profile's jerk).

If we temporarily ignore the jerk, for simplicity, it is easy to see the trapezoidal shape of the velocity:

The velocity increases from 0 to "Speed" at a constant rate determined by "Accel". The velocity then remains constant, until it reaches the distance from the target where deceleration should begin. Then, the velocity is decreased at a constant rate determined by "Decel" until it reaches 0 (stops).

The resulting speed curve is thus a trapeze:



T1 = Speed / Accel
T2 = T3 – Speed/Decel

The value of T3 is determined according to the traveled distance.
If Jerk is added, the shape of the speed curve will change. The acceleration will now change gradually, according to jerk. This will add parabolic parts in the beginning and end of the acceleration and deceleration.

Adding Jerk protects the system against stress that can be caused by a sudden change in acceleration. The price of this protection is that the motion time is longer.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

AbsTrgt is the absolute target for a point to point motion. AbsTrgt is the desired target position in the end of the motion regardless of the current position. For point to point motion this will be the location in which the motion will end.

If RelTrgt is not 0 then AbsTrgt is ignored and RelTrgt is used to determine the actual target of the next motion.

RelTrgt is the relative target for the motion. It determines the distance to travel from the current position.

AbsTrgt can be changed during motion and the target of the current motion will change immediately.

How to start a point to point motion

The following example of beginning a point to point motion can be used as a template (an axis letter should be added at the beginning of each keyword. It is omitted here.) :

```
MotionMode = 1
RelTrgt = 0          // We will use AbsTrgt to determine the end point of the motion
AbsTrgt = 200000     // Move to position 200000 user units
Speed = 20000        // The constant speed should be 200000 user units/sec
Accel = 100000       // Accelerate at 100000 user units/sec^2
Decel = 400000       // Decelerate at 400000 user units/sec^2
Jerk = 4             // Use jerk
MotorOn = 1          // Enable the motor
Begin                // Begin the motion
```

The motion will now begin.

On the fly changes

All the motor parameters above (Begin is not a parameter) can be changed during the motion and will take effect if relevant. Changing “Speed” during the constant velocity phase will cause the motor to accelerate or decelerate to the new speed, using “Accel” or “Decel”. Changing “Accel” after the acceleration phase ended will, obviously, has no effect on the current motion. The target position can be changed at any time until the motion ends.

Stopping the motion

The point to point motion will normally end when the target is reached.
To stop the motion before the target is reached use the “Stop” command. The motion will stop using “Decel” to decelerate and applying “Jerk”.
For emergency stop use “Abort”. This will force an immediate end of motion.
The motion can also end due to a fault condition or limit. If a limit is reached (either software or hardware limit), the motion will stop using “EmrgDec” to decelerate. Typically, EmrgDec is set as high as the system can tolerate.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Checking the motion status

During motion the position, velocity, current and other motion related parameters can be queried or recorded.
To know at what stage the motion is at the moment use “MotionStat”. See the keyword reference for details about the various values possible. MotionStat can also be used in a user program to wait for the motion to reach one of its stages. For example, the program can wait for the motor to begin deceleration before setting an output.

If the motion ends with an error use MotionReason to find out what is the reason for the end of motion. Use ConFlt to check for protections that were activated during the motion.

Jog

Jog motion mode (MotionMode = 0) will generate a constant speed motion that will continue indefinitely until stopped. The parameters that determine the shape of the profile are:

Parameter	Description
Speed	The desired speed in the constant speed part of the motion
Accel	The desired acceleration
Decel	The desired deceleration (for “Stop” command and on the fly changes)
Jerk	Determines the smoothing of the profile. 2^{Jerk} is the number of control samples that will be needed to reach the specified Accel or Decel (time to build the acceleration … which actually defines the profile’s jerk).

The motion will begin by accelerating according to the values of Accel and Jerk, the same way described in “Point to Point” motion.
After reaching the constant speed the motion will continue until something happens to cause a change.

How to start a jog motion

The following example of beginning a jog motion can be used as a template (an axis letter should be added at the beginning of each keyword. It is omitted here.) :

```
MotionMode = 0
Speed = 20000          // The constant speed should be 200000 user units/sec
Accel = 100000         // Accelerate at 100000 user units/sec^2
Decel = 400000         // Decelerate at 400000 user units/sec^2
Jerk = 4               // Use jerk
MotorOn = 1            // Enable the motor
Begin                  // Begin the motion
```

The motion will now begin.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

On the fly changes

All the motor parameters above (Begin is not a parameter) can be changed during the motion and will take effect if relevant. Changing “Speed” during the constant velocity phase will cause the motor to accelerate or decelerate to the new speed, using “Accel” or “Decel”. Changing “Accel” after the acceleration phase ended will, obviously, has no effect on the current motion.

Stopping the motion

To stop the motion use the “Stop” command. The motion will stop using “Decel” to decelerate and applying “Jerk”.
For emergency stop use “Abort”. This will force an immediate end of motion.
The motion can also end due to a fault condition or limit. If a limit is reached (either software or hardware limit), the motion will stop using “EmrgDec” to decelerate. Typically, EmrgDec is set as high as the system can tolerate.

Checking the motion status

During motion the position, velocity, current and other motion related parameters can be queried or recorded.
To know at what stage the motion is at the moment use “MotionStat”. See the keyword reference for details about the various values possible. MotionStat can also be used in a user program to wait for the motion to reach one of its stages. For example, the program can wait for the motor to begin deceleration before setting an output.

If the motion ends with an error use MotionReason to find out what is the reason for the end of motion. Use ConFlt to check for protections that were activated during the motion.

Point to Point Repetitive

Point to point repetitive motion mode (MotionMode = 2) is most useful during system development. In this mode the motor moves back and forth between two points. It waits for a determined time in each end point. During this motion the developer can change parameters and use the auto recording feature of the PC suite to see how the changes affect the motion.
The parameters that determine the shape of the profile are:

Parameter	Description
AbsTrgt	The absolute position target
RelTrgt	The position target relative to the current position
Speed	The desired speed in the constant speed part of the motion
Accel	The desired acceleration
Decel	The desired deceleration
Jerk	Determines the smoothing of the profile. 2^{Jerk} is the number of control samples that will be needed to reach the specified Accel or Decel (time to build the acceleration … which actually defines the profile’s jerk).
RptWait	The duration of the wait between motions in milliseconds

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

How to start a point to point repetitive motion

The following example of beginning a point to point motion can be used as a template (an axis letter should be added at the beginning of each keyword. It is omitted here.) :

```
MotionMode = 2
RelTrgt = 0              // We will use AbsTrgt to determine the end point of the motion
AbsTrgt = 200000  // Move to position 200000 user units
Speed = 20000            // The constant speed should be 200000 user units/sec
Accel = 100000           // Accelerate at 100000 user units/sec²
Decel = 400000           // Decelerate at 400000 user units/sec²
Jerk = 4                 // Use jerk
RptWait = 500            // Wait 0.5 second after each motion
MotorOn = 1              // Enable the motor
Begin                    // Begin the motion
```

The motion will now begin. The motor will move from its current location to location 200,000, wait for 0.5 seconds, return to the current location, wait for 0.5 second and repeat the same motion.

On the fly changes

All the motor parameters above (Begin is not a parameter) can be changed during the motion and will take effect if relevant. Changing “Speed” during the constant velocity phase will cause the motor to accelerate or decelerate to the new speed, using “Accel” or “Decel”. Changing “Accel” after the acceleration phase ended will, obviously, have no effect on the current motion. The target can be changed at any time until the motion ends. The location from which the motion begins is saved at the beginning of the repetitive motion and cannot be changed without beginning a new motion.

Stopping the motion

The point to point repetitive motion will continue indefinitely until it is stopped.
To stop the motion before the target is reached use the “Stop” command. The motion will stop using “Decel” to decelerate and applying “Jerk”.
To stop at one of the end points use “StopRep”. The current point to point motion will continue until the end point is reached and then the repetition will stop.
For emergency stop use “Abort”. This will force an immediate end of motion.
The motion can also end due to a fault condition or limit. If a limit is reached (either software or hardware limit), the motion will stop using “EmrgDec” to decelerate. Typically, EmrgDec is set as high as the system can tolerate.

Checking the motion status

During motion the position, velocity, current and other motion related parameters can be queried or recorded.
To know at what stage the motion is at the moment use “MotionStat”. See the keyword reference for details about the various values possible. MotionStat can also be used in a user

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

program to wait for the motion to reach one of its stages. For example, the program can wait for the motor to begin deceleration before setting an output.

If the motion ends with an error use MotionReason to find out what is the reason for the end of motion. Use ConFlt to check for protections that were activated during the motion.

Direct and indirect modes

In some of the motion modes the reference signal is not generated internally by the profiler. The reference signal may be pulses that are entered through a designated input, an analog command, a master motor that should be followed or multiplied, or any other signal.

In some of the cases, the input reference signal is not smooth. For example, consider a pulse/direction input that is used as position reference. If the pulses in the input are entered in bursts, it is possible that many pulses will be received in one sample time. If 1000 pulses are received in one sample time, when the motor is at rest, this is equivalent to a step response of 1000 pulses.

If the user does not want high accelerations and decelerations, they can use an indirect mode.

In an indirect mode the 1000 pulses become the relative target for the motion, and the profile is generated by the profiler. The reference now will be slower and smoother, using the user set parameters.

If the motor is already in motion the input pulses are added to the current target.

Indirect modes can also be used with other reference inputs and not only pulse/direction.

Pulse/Direction

Pulse/direction mode (MotionMode = 3 for direct, MotionMode = 4 for indirect), the position reference is not entered by using a target command. The reference is entered from the dedicated pulse/direction input (see hardware manual). Whenever pulses are received from the pulse input the position reference is updated and the motor will move accordingly.
The calculation of the trajectory is different in direct and indirect modes. See explanation above.

It is also possible to determine a ratio between the number of input pulses and the value of the position reference. This is done using “PDFact” (pulse/direction factor) and “PDFactDen” (pulse/direction factor denominator). The number of pulses is multiplied by PDFAct and divided by PDFactDen.

The parameters that affect the profile if indirect mode is applied:

Parameter	Description
Speed	The desired speed in the constant speed part of the motion
Accel	The desired acceleration
Decel	The desired deceleration (for “Stop” command and on the fly changes)

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Jerk	Determines the smoothing of the profile. 2^{Jerk} is the number of control samples that will be needed to reach the specified Accel or Decel (time to build the acceleration … which actually defines the profile's jerk).

The parameters that affect both pulse/direction modes:

Parameter	Description
PDFact	Multiplies the number of input pulses before entering it to position reference
PDFactDen	Multiplies the number of input pulses before entering it to position reference

How to start a pulse/direction motion

The following example of beginning an indirect pulse/direction motion can be used as a template (an axis letter should be added at the beginning of each keyword. It is omitted here.) :

```
MotionMode = 4
Speed = 20000          // The constant speed should be 200000 user units/sec
Accel = 100000         // Accelerate at 100000 user units/sec^2
Decel = 400000         // Decelerate at 400000 user units/sec^2
Jerk = 4               // Use jerk
PDFact = 4        // Multiply the number of input pulses by 4
PDFactDen = 5          // Divide the result by 5 so the overall ratio is 4/5
MotorOn = 1            // Enable the motor
Begin                  // Begin the motion
```

The motion will now begin.

To start a direct pulse/direction motion:

```
MotionMode = 3
PDFact = 4        // Multiply the number of input pulses by 4
PDFactDen = 5          // Divide the result by 5 so the overall ratio is 4/5
MotorOn = 1
Begin
```

On the fly changes

All the motor parameters above (Begin is not a parameter) can be changed during the motion and will take effect if relevant.

Stopping the motion

To stop the motion use the "Stop" command. Any pulses that are entered after the Stop command is received are ignored. If it is a direct mode the motion will stop immediately. In indirect mode the motion will stop using Decel.
For emergency stop use "Abort". This will force an immediate end of motion.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

The motion can also end due to a fault condition or limit. If a limit is reached (either software or hardware limit), the motion will stop using “EmrgDec” to decelerate. Typically, EmrgDec is set as high as the system can tolerate.

Checking the motion status

During motion the position, velocity, current and other motion related parameters can be queried or recorded.
To know at what stage the motion is at the moment use “MotionStat”. See the keyword reference for details about the various values possible. MotionStat can also be used in a user program to wait for the motion to reach one of its stages. For example, the program can wait for the motor to begin deceleration before setting an output.

If the motion ends with an error use MotionReason to find out what is the reason for the end of motion. Use ConFlt to check for protections that were activated during the motion.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

ECAM

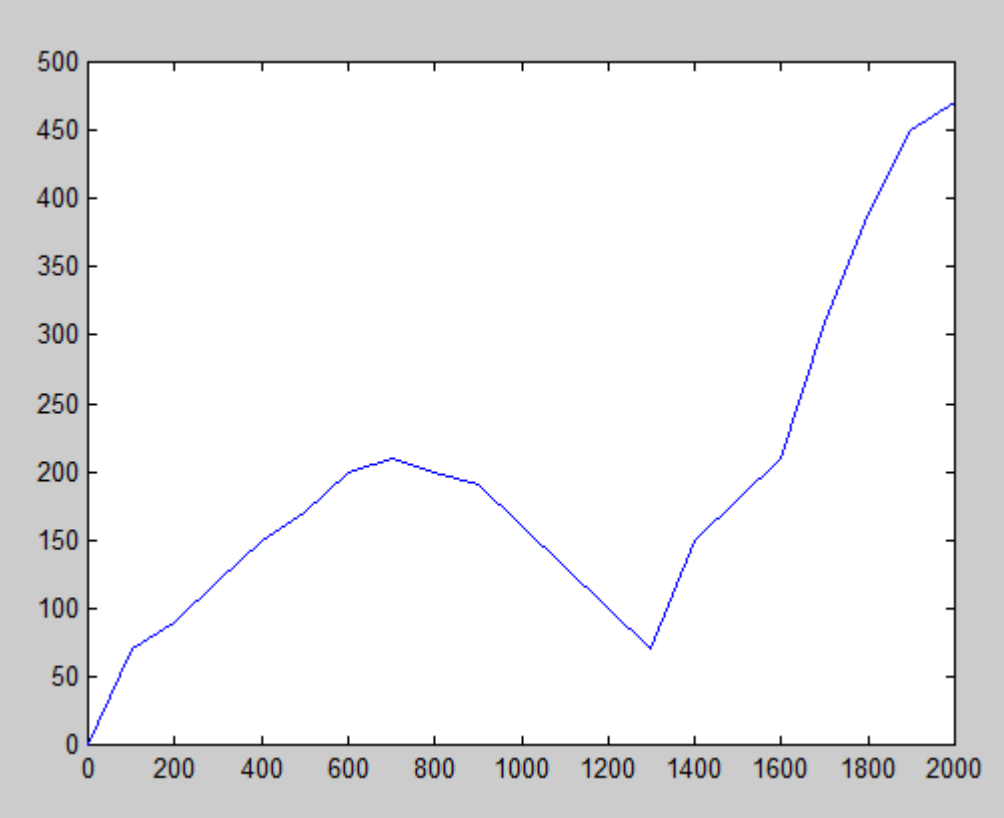
ECAM motion (MotionMode = 7 for direct and MotionMode = 8 for indirect) is used in applications where the motor needs to move in a complex motion that is none linearly dependent on an input value.

When using ECAM the controlled axis will follow a defined input according to a table we call ECAM table. The ECAM table defines the relation between the “master” (= input) value and the “slave” (= controlled axis position).

For example, the ECAM table below is represented by the following graph (using linear interpolation between table entries):

Master value	Slave position
0	0
100	70
200	90
300	120
400	150
500	170
600	200
700	210
800	200
900	190
1000	160
1100	130
1200	100
1300	70
1400	150
1500	180
1600	210
1700	310
1800	390
1900	450
2000	470

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945



The X-axis of the graph represents the position of the master. The Y–axis values represent the position of the slave.

When the master input changes, the position reference of the slave will change according to the required position.

If the master position increases and then decreases, the slave will go forward and backwards on the graph. If the master position is to the left of the table the slave will remain in the position defined by the left edge of the table. If the master moves to the right of the table, the slave will remain in the last position defined by the table.

Note that the master position is calculated relative to the initial position. If the ECAM motion begins when the master value is, for example, 1500, then the value of entry to the ECAM table is calculated as: *actual master value – 1500*.

So, for this example using the above table, when the master reaches value 1700 the table entry for value 200 is used and the slave position reference will be 90.

Storing a basic ECAM table in GenData array

The values of the required position are stored in GenData array. The values must be stored in consecutive locations of the array. The actual spacing between master positions can be set using ECAMGap.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

To store the above table in GenData starting from array index 100 (any array index can be selected), simply enter the values to consecutive locations in GenData:

GenData[100] = 0
GenData[101] =70
GenData[102] = 90
…
…
…
GenData[120] = 470

Note:

The ECAM table data (into the GenData[] array) as well as all other related ECAM parameters can be typed in manually (over the communication of from a user program) or, alternatively, you can use the PC Suite dedicated tool: Tool/Draw that provides the means to define a desired ECAM contour, using spline tools, as well all other related ECAM parameters, and to download them to the controller.

This User's Manual shows the manual way. You can refer to the PC Suite User's Manual for guidelines regarding how to use the Tools/Draw tool.

To link this list of values to an ECAM motion we need to assign them the properties of an ECAM table. There can be up to 10 tables ready for use at the same time. Every property is saved for every table, and switching between tables is done using a single keyword.

This will be table no. 1:

ECAMStart[1] = 100
ECAMStartCyc[1] = 100

The above parameters are equal for this specific examples, and both point to the beginning of the table. Later their separate meanings will be explained.

ECAMEndCyc[1] = 120
ECAMEnd[1] = 120

Same as the beginning, both parameters now point to the end of the table.

The master values of the original table were 0,100,200… so we set the gap parameter:

ECAMGap[1] = 100

The process of generating a table and entering the values is made easy by using PCSuite. The Draw window in the tools menu automates the procedure.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Starting a simple ECAM motion

Now we should select the master for the motion. Common master selections are the position of the auxiliary encoder or the value of an analog input. Another common way of using ECAM is defining time as the master axis.

Any parameter can be selected as the master value for ECAM motion, including array members. The master selection is done using "Complex CAN code", the same way the recording parameters are selected. When using Akribis PC suite the complex CAN code is generated automatically. Refer to Appendix I for detailed description of how to calculate the Complex CAQN Code of a given parameter.

For example, to use AuxPos as master, the complex CAN code is 3. Enter:

ECAMMaster[1] = 3

For this first example we will ignore the following parameters and set them to defaults:

ECAMCycles[1] = 1
ECAMMasterIni[1] = 0
ECAMInterp[1] = 0

To begin the ECAM motion we need to state that we are using table no. 1:

ECAMTableNum = 1

The motion mode for ECAM is 7:

MotionMode = 7
Begin

Now, when the master moves between its current position POS_0 and $POS_0 + 2000$ the slave will follow the graph above using linear interpolation between the table entries. If the master moves to a position lower than POS_0 the slave will remain in the position assigned to the first table entry. If the master moves to a position higher than $POS_0 + 2000$ the slave will remain in the position defined by the last point of the table.

Stopping an ECAM motion

There are two ways to stop the ECAM motion:

Stop: the motion will stop immediately, the same as in pulse/direction

StopECAM: Use StopECAM to make the current cycle become the last cycle. After the master value exceeds the end of the current cycle, the deceleration part (Part III) of the table will be followed. When the master value passes the end of part III the motion will stop and MotionStat will be 0.

More general ECAM motions

The following sectins will describe more general ECAM motions and aspects.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Cyclic motion with acceleration and deceleration sections

In general, the ECAM table can be more complicated and include three parts:

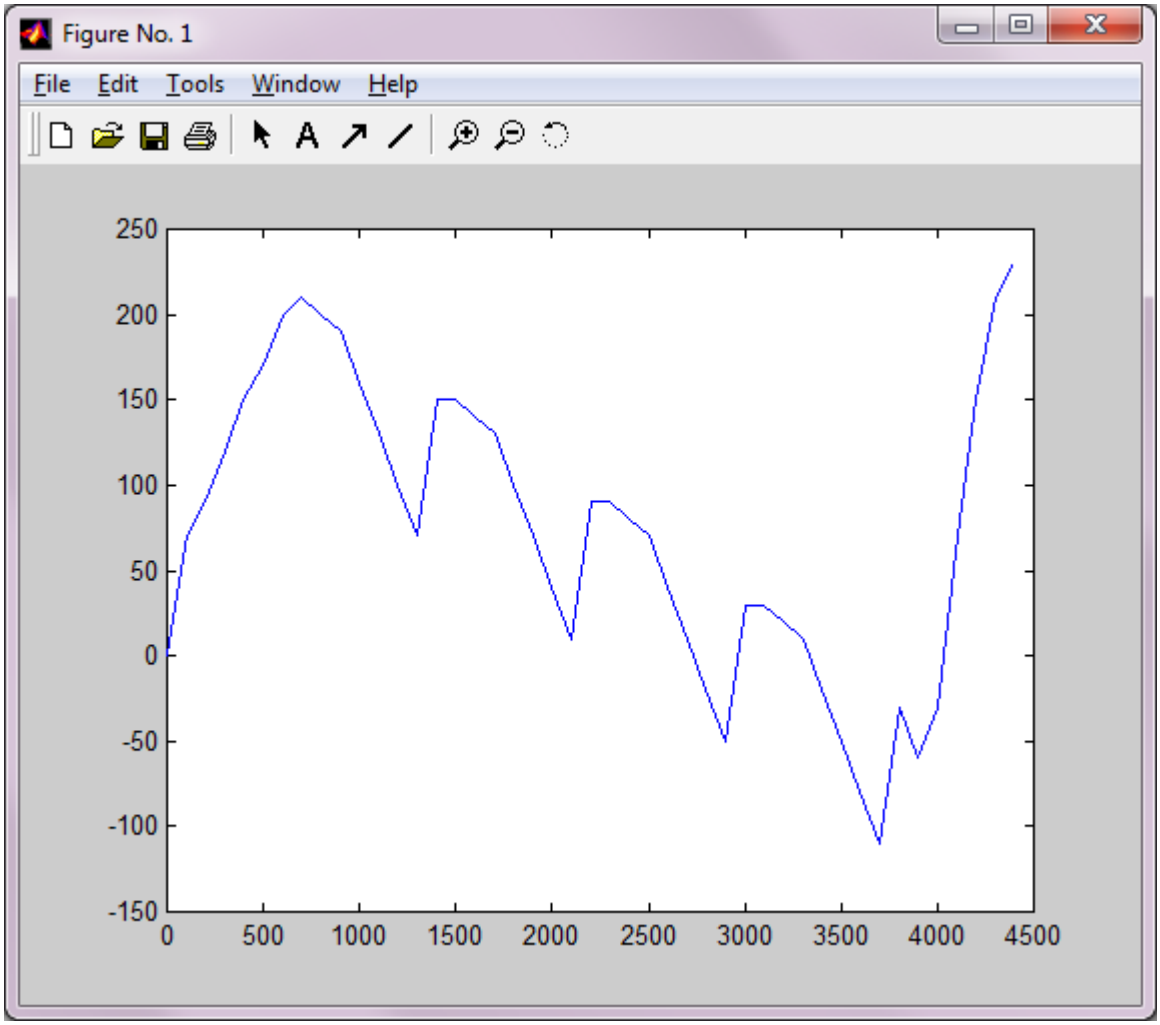
Part I: From ECAMStart[] to ECAMStartCyc[]. This part of the table is followed once, in the beginning of the motion, and is usually used to accelerate the slave to the velocity that is needed in the beginning of the repetitive motion.
Part II: From ECAMStartCyc[] to ECAMEndCyc[]. This part is the cyclical part of the motion. It is repeated ECAMCycles[] times assuming the master motion is in a monotonous direction.
Part III: From ECAMEndCyc[] to ECAMEnd[].This part of the table is followed once, in the end of the motion, and is usually used to decelerate the slave from the velocity that is needed in the end of the repetitive motion.

We can use the same table as above, but decide that the values that were entered in locations 7-14 will be repeated 4 times. The table stays the same. We only need to change the following parameters:

ECAMStartCyc[1] = 107
ECAMEndCyc[1] = 114
ECAMCycles[1] = 4

Now the motion will look like this:

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945



The first points, when the master value is between 0 and 1400 are the same as before. After that the part between 700 and 1400 is repeated 3 more times for a total of 4 times. Note that each repetition compensates for the position difference between the slave position at the beginning and end of the table. Each cycle has the same shape, and it begins at the last point of the previous cycle.

Endless ECAM:

If ECAMCycles[x] is set to 2147483647 (exactly), the controller will perform infinite number of ECAM cycles, until some external event will stop the motion (such as: Stop or StopECAM commands are received, or hitting limit switches …) or disable the motor (MotorOn=0 message or some fault).

For negative values of ECAMCycles[x], use a value of -2147483648 (exactly) to create endless ECAM.

Note that endless ECAM is possible only if selecting a master that can "work" forever. Useful examples are AuxPos and CounterUp[]. Both of them roll over at the maximal value of 32 bits.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

The endless ECAM is designed to provide smooth and transparent operation during this rolling over.

Reversing the master direction

All that is needed to reverse the master position is a negative value for ECAMGap. If, for example, we make ECAMGap[1] = -100 in the previous example, then when the master will move in the negative direction the slave will behave the same as we see above. This is useful if the master value cannot be reversed.

Bidirectional master motion

The cyclic ECAM motion can continue on both sides of the beginning point. To do that, set ECAMCycles[] to a negative value. If ECAMCycles[] = -N:

When the master moves in the positive direction the slave will do N cycles and then use part III of the ECAM table to decelerate.
When the master moves in the negative direction the slave will do N cycles and then use part I of the ECAM table to decelerate.

Note: In this mode, immediately when the Begin command is received the slave will receive a command to move to the position indicated by the value in ECAMStartCyc. If this value is not 0 it will be a step command.

This mode can also be combined with a reversing of the master direction.

Initial master position

ECAMMasterIni[] is subtracted from the beginning position if needed to compensate for master motion.

Example:

If ECAMMasterIni[1] = 0 and the Begin command is received when the master position is 3000 then all master positions are relative to 3000. The ECAM motion will start immediately, giving position command to the position pointed by ECAMStart[1]

If ECAMMasterIni[1] = 2000 and the Begin command is received when the master position is 3000 then all master positions are relative to 1000. The motion will start immediately as if the master already moved to value 1000. The user must design the table so that this motion is reasonable.

Example: Contour (time based ECAM)

For a time based ECAM we can use the table in the example above, and only change the value of ECAMMaster[1].

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

We can use one of the two internal counters. CounterUp[1] counts up every sample time. We can set a starting value to CounerUp[1] to make sure is will not overflow during our motion, or read it to make sure there is enough time.

If ECAMGap = 100 the motion will advance from one table point to the next in 100 samples. The complex CAN code for CounterUp[1] is 65576 (this is automatically calculated by the PC suite in the Draw tool). Refer to Appendix I for detailed description of how to calculate the Complex CAQN Code of a given parameter.

Enter ECAMMaster[1] = 65576 and begin an ECAM motion. The motion will be performed according to the advance of the counter. You can change the gap to make the motion slower or faster.

Warning: Do not manually change the value of the counter during motion. This may result with big jumps of the motor. Note that until the motion is stopped by a stop or stopECAM command the motion is still active.

List of relevant keywords

ECAMStart[n]: The GenData index that holds the first value of ECAM table n.
ECAMStartCyc[n]: The GenData index that holds the first value of the cyclic part of ECAM table n.
ECAMEndCyc[n]: The GenData index that holds the last value of the cyclic part of ECAM table n.

ECAMEnd[n]: The GenData index that holds the last value of ECAM table n.

ECAMGap[n]: The difference in master values that is represented by the gap between each two table points. Each table can have a different gap.

ECAMCycles[n]: The number of times to repeat the cyclic part of the motion for table n.

ECAMMasterIni[n]: This number is subtracted from the initial master position to compensate for the master motion.

ECAMCycCount[n]: The number of the current cycle (read only).

ECAMInterp[n]: The interpolation to be used between data points. At the moment only linear interpolation is available and ECAMInterp is always 0.

ECAMMaster[n]: The parameter to be used as the master value for ECAM table n. ECAMMaster is in complex CAN code. Refer to Appendix I for detailed description of how to calculate the Complex CAQN Code of a given parameter.

ECAMTableNum: The number of the ECAM table to be used.

StopECAM: This command stops the ECAM after the current cycle using the deceleration part of the table.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Gearing

The gearing mode (MotionMode = 5 for direct and MotionMode = 6 for indirect) is used in applications where the controlled motor motion is determined as a constant ratio to the master motor (auxiliary feedback).
The ratio between the master position and the position reference is determined by MasterFact. The value of MasterFact is internally divided by 65536 to allow fractions.

The parameters that affect the profile if indirect mode is applied:

Parameter	Description
Speed	The desired speed in the constant speed part of the motion
Accel	The desired acceleration
Decel	The desired deceleration (for “Stop” command and on the fly changes)
Jerk	Determines the smoothing of the profile. 2^{Jerk} is the number of control samples that will be needed to reach the specified Accel or Decel (time to build the acceleration … which actually defines the profile’s jerk).

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

The parameters that affect both gearing modes:

Parameter	Description
MasterFact	Multiplies the number of input pulses before entering it to position reference. Internally divided by 65536. Use 65536 for 1:1 ratio.

How to start a gearing motion

The following example of beginning an indirect gearing motion can be used as a template (an axis letter should be added at the beginning of each keyword. It is omitted here.) :

```
MotionMode = 6
Speed = 20000          // The constant speed should be 200000 user units/sec
Accel = 100000         // Accelerate at 100000 user units/sec2
Decel = 400000         // Decelerate at 400000 user units/sec2
Jerk = 4               // Use jerk
MaterFact = 131072     // (=65536 * 2) The controlled motor will move twice as fast
MotorOn = 1            // Enable the motor
Begin                  // Begin the motion
```

The motion will now begin.

To start a direct pulse/direction motion:

```
MotionMode = 3
MaterFact = 131072     // (=65536 * 2) The controlled motor will move twice as fast
MotorOn = 1
Begin
```

On the fly changes

All the motor parameters above (Begin is not a parameter) can be changed during the motion and will take effect if relevant.

Stopping the motion

To stop the motion use the “Stop” command. Any changes in the master position after the Stop command is received are ignored. If it is a direct mode the motion will stop immediately. In indirect mode the motion will stop using Decel.
For emergency stop use “Abort”. This will force an immediate end of motion.
The motion can also end due to a fault condition or limit. If a limit is reached (either software or hardware limit), the motion will stop using “EmrgDec” to decelerate. Typically, EmrgDec is set as high as the system can tolerate.

Checking the motion status

During motion the position, velocity, current and other motion related parameters can be queried or recorded.
To know at what stage the motion is at the moment use “MotionStat”. See the keyword reference for details about the various values possible. MotionStat can also be used in a user

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

program to wait for the motion to reach one of its stages. For example, the program can wait for the motor to begin deceleration before setting an output.

If the motion ends with an error use MotionReason to find out what is the reason for the end of motion. Use ConFlt to check for protections that were activated during the motion.

FIFO

FIFO is a special motion mode in which the controller performs a sequence of linear and parabolic segments, as defined by the user before the motion and optionally also during the motion.

The motion is created according to motion segments which are stored in a FIFO memory.

The motion segments definitions can be pushed to the FIFO at any time (before or during the motion), providing that the FIFO is not full. If the FIFO is full, the push operation is rejected with a suitable error.

If, during a motion in this mode, the controller reaches the last element in the FIFO, and completing this motion segment, and yet no new element was pushed, the motion is automatically ended.

The motion can be also stopped using the Stop or the StopFIFO functions. The first decelerate to zero speed and the second makes the currently executed motion segment to be the last segment.

Each motion segment can be of type Velocity or Acceleration. Velocity type segment is a segment in which the velocity reference is constant and Acceleration type segment is a segment in which the acceleration reference is constant.

The time length of each segment (FIFOCycleTime) is a fixed value of number of control samples. However, it can be modified at any time that the controller is ending a given segment and starting a new one.

The FIFO is of size 512 entries. Each entry has type and value. See below for possible FIFO entry types. If all entries are motion entries, the FIFO can hold up to 512 motion segments. Of course, it can be re-filled over the communication, during the motion sequence.

The controller provides variety of parameters and functions to handle the FIFO and the motion behavior, as described in details below.

Possible segment motions

1. Velocity type motion segment:

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

a. Segment duration is known in number of samples (sample time of the control loop). It is stored in the parameter FIFOCycleTime.
b. The segment starts naturally from the last position reference.
c. Motion definition can be given by one of:

i. Move linearly with given delta for position reference.

1. Final target position is calculated using the given delta. The delta is given with scaling of the controller SAMPLING_FREQUENCY (typically 16384 Hz). This means that a delta of 1 [count] is entered as 16384. And a delta of 2.5 [counts] is entered as 2.5*16384 = 40960.

   This means that fractional values are supported to ensure trajectory smoothness between the segments.
   As a result, the delta is limited to ±131071.

   If larger delta is required, the segment shall be divided into few segments, or another motion definition shall be used (see below).

2. Segment velocity is calculated, in [counts/sec].
3. The velocity is used to increment the position reference at each sample.
4. At last sample, the position reference is assigned to the desired value (to compensate for possible increments inaccuracies).

ii. Move linearly with given velocity.

1. Segment velocity is provided in [counts/sec].
2. The velocity is used to increment the position reference at each sample.
3. At last sample, there is the resulted position reference, as calculated by the increments.
4. Note that in this definition type, the segment is not limited in length as the previous one (having limited value of the position delta).

2. Acceleration type motion segment:
(Details explanations are not repeated as they are similar to the above described cases)

a. Segment duration is known in number of samples.
b. The segment starts naturally from the last position reference, as well as from the end velocity of the previous segment.
c. Motion definition can be given by one of:

i. Move parabolic with given delta for position reference.
1. Final target position is calculated using the given delta.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

2. Segment acceleration (or deceleration) is calculated, in [counts/sec^2].
3. The acceleration is used to increment the desired velocity and position at each sample, based on:

   VelRefProfiler = VelRefProfiler + Acceleration * Ts
   PosRef = PosRef + VelRef
   Ts is the controller sampling time, typically 1/16384.

4. At last sample, the position reference is assigned to the desired value (to compensate for increments inaccuracies).

ii. Move parabolic with given acceleration.
   1. Segment acceleration (or deceleration, if negative) is provided in [counts/sec^2].
   2. The acceleration is used to increment the desired velocity and position at each sample, based on:

      VelRefProfiler = VelRefProfiler + Acceleration * Ts
      PosRef = PosRef + VelRef
      Ts is the controller sampling time, typically 1/16384.

   3. At last sample, there is the resulted velocity reference and position reference, as calculated by the increments.

FIFO entry types

1. FIFO Motion Mode parameters:

   a. The FIFO motion mode performs segment motions as a function of the segment definition itself, and as a function of some parameters (like the FIFOCycleTime – the time duration of a single segment motion).
   b. The values of these parameters can be modified by pushing a new value into the FIFO. This value is used when the controller reaches this location in the FIFO. It is always between two motion segments. As a result, it is a way to assign values to the relevant parameters in a synchronized way.
   c. The new value is effective immediately and will be used till a new value is reached in the FIFO.
   d. Such FIFO entries do not consume motion time.
   e. A given FIFO motion sequence can use new parameters between each segment, or can avoid using new parameters at all. Pushing new motion mode parameters into the FIFO is optional and shall be used according to the desired overall motion sequence/contour.
   f. It is assumed that there will be limited number of such entries between motion segments (to avoid CPU load within the control interrupt when reaching this location in the FIFO).

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

g. FIFO entry type to assign a value to a parameter is given in one of:

i. FIFO_CYCLE_TIME:
It assigns a value to the parameter FIFOCycleTime. This is the FIFO cycle time, which is the duration, in control samples, of each FIFO motion segment.

Use the function FIFOPushCycle to make such a push:

For example:

FIFOPushCycle, 1000

pushes a value of 1000 into the FIFO, with type of FIFO_CYCLE_TIME.

It will assign the value of 1000 to FIFOCycleTime, when the FIFO motion will reach this entry in the FIFO. It then will be used for all coming segments, till it will be changed again.

ii. More to be added in the future.

2. FIFO Motion segments:

a. The FIFO motion mode performs segment motions that are located in the FIFO array.
b. A motion segment is created by pushing a motion segment into the FIFO.
c. This segment will be executed when the FIFO motion will reach this entry.
d. Each motion segment “consumes” a time period which is equal to the value of FIFOCycleTime (its value when this segment is executed).
e. Various type of motion segments are supported, as defined above in details.
f. FIFO entry type to define a motion segment is given in one of:

1. LINEAR_BY_DELTAPOS:
It is a FIFO entry that initiates a segment of velocity motion, defined by the delta position reference from the start of the segment to its end.

The value is scaled by the sampling frequency of the control loop. Naturally with Akribis controllers it is 16384 [Hz].

Use the function FIFOPushLinP to make such a push:

For example:

FIFOPushLinP, 163024896

Pushes a value of 163024896 into the FIFO, with a type of LINEAR_BY_DELTAPOS.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

It will initiate a motion segment of linear motion, with delta position of 9950.25 (=163024896/16384).

2. LINEAR_BY_VELOCITY:
   Similarly to the above, but the relevant pushing function is FIFOPushLinV, the pushed type is LINEAR_BY_VELOCITY and the pushed value is the velocity for this segment, in [counts/sec] (no scaling is required).

3. PARABOLIC_BY_DELTAPOS:
   It is a FIFO entry that initiates a segment of acceleration motion, defined by the delta position reference from the start of the segment to its end.

   The value is scaled by the sampling frequency of the control loop. Naturally with Akribis controllers it is 16384 [Hz].

   Use the function FIFOPushParP to make such a push:

   For example:

   FIFOPushParP, 163024896

   Pushes a value of 163024896 into the FIFO, with a type of PARABOLIC_BY_DELTAPOS.

   It will initiate a motion segment of acceleration (parabolic) motion, with delta position of 9950.25 (=163024896/16384).

4. PARABOLIC_BY_ACCLERATION:
   Similarly to the above, but the relevant pushing function is FIFOPushParA, the pushed type is PARABOLIC_BY_ACCLERATION and the pushed value is the acceleration for this segment, in [counts/sec^2] (no scaling is required).

FIFO parameters and functions

The following are the keywords relevant for FIFO motions:

1. FIFOValue[]
   This keyword shows the FIFO as an array. It shows the “value” entries. Each element in the array is a part of a FIFO entry. The complete FIFO entry consists of this value and of the type of this value (see next item).

2. FIFOType[]

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

This keyword shows the FIFO as an array. It shows the "type" entries. Each element in the array is a part of a FIFO entry. The complete FIFO entry consists of a value (see previous item) and of the type of this value.

3. FIFOStatus[]:

   a. FIFOStatus[1] → FIFO_INDEX
   Holds the current value of the pointer into the FIFO.

   b. FIFOStatus[2] → FIFO_FREE
   Holds the number of free entries in the FIFO.

   c. FIFOStatus[3] → FIFO_COUNT_DOWN
   Holds the number of [samples] remained to complete the current motion segment.

   d. FIFOStatus[4] → FIFO_SEGMENT_VELOCITY
   Holds the velocity that is used in this segment, in [counts/sec].

   e. FIFOStatus[5] → FIFO_SEGMENT_ACCELERATION
   Holds the acceleration that is used in this segment, in [counts/sec^2].

4. Push functions:

   a. FIFOPushCycle
   Push a value into the FIFO that will be used as a new value for FIFOCycleTime.

   b. FIFOPushLinP
   Push a value into the FIFO that will be used as a motion segment instruction. The motion segment is of type Velocity (linear position) and the value defines the delta position (in [counts] and scaled by the control sampling frequency) for this segment (from its start till its end).

   c. FIFOPushLinV
   Push a value into the FIFO that will be used as a motion segment instruction. The motion segment is of type Velocity (linear position) and the value defines the velocity (in [counts/sec]) during this segment.

   d. FIFOPushParP
   Push a value into the FIFO that will be used as a motion segment instruction. The motion segment is of type Acceleration (parabolic position) and the value defines the delta position (in [counts] and scaled by the control sampling frequency) for this segment (from its start till its end).

   e. FIFOPushParA
   Push a value into the FIFO that will be used as a motion segment instruction. The motion segment is of type Acceleration (parabolic position) and the value defines the acceleration (in [counts/sec^2]) during this segment.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

5. FIFORemove
   Removes the last entry (recently pushed) from the FIFO.

6. FIFOClear
   Clears the entire FIFO.

   Note:

   If a FIFO motion halts due to some fault or limit, its FIFO may be not empty (it will remain as it was when the motion was halted). You may need to perform FIFOClear before starting a new FIFO motion.

7. FIFOCycleTime
   Defines the period of time, in [samples], for a motion segment.

8. MotionMode
   Needs to have the relevant value to define FIFO motion (MotionMode = 9).

9. Begin
   Starts a FIFO motion sequence if the MotionMode is set to this mode (9).

10. Stop
    Stops the motion using deceleration.

11. StopFIFO
    Stops the motion by defining the currently executed motion segment as the last segment.

Miscellaneous

1. When the motion is ended (by StopFIFO or by ending the last segment), the remaining position reference's fraction (if any) is cleared to zero.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Data Recording

The data recording feature allows the user to record the value of any controller parameter, including arrays. The recorded data is saved in the controller and can be uploaded for viewing.

The easiest way to use the recording is with the PC suite. The PC suite helps in configuring the data to record and displays the uploaded data in a very useful graphic interface.

Up to 4 different variables can be recorded with a total recording length of 8000 points. The user can record up to 8000 points of a single variable or up to 2000 points each of 4 variables.

Selecting the data to record

The parameters that should be recorded are written into RecParam[1..4]. All the non-zero values are recorded.

The parameters are represented by their complex CAN code. This allows the controller maximum flexibility: any parameter, including arrays, can be recorded. The price of this flexibility is some difficulty in assigning the parameters.

The PC suite has a very convenient user interface that eliminates this difficulty completely. If, for some reason it is necessary to enter these numbers manually, refer to Appendix I for detailed description of how to calculate the Complex CAN Code of a given parameter.

Recording Length and Gap

The user can determine the length of the recording (how many points are recorded), and the gap between sampled points. The minimum possible gap is 1 sample time. For a controller with 16KHz control loops this means a sample every ~60 microseconds.

The user can record with a small gap for a short period of time, or increase the gap to see longer processes.

The recording gap in the controller is set in sample times (See RecGap) and the recording length (RecLength) is set as a number of recording points. When using the PC suite you can set the length and gap in time units. These units are automatically translated to controller units and rounded if needed. If the recording length resulting from the settings exceeds the allowed recording length and error message is displayed. Try to decrease the overall recording time, increase the gap between points or record less vectors.

Recording Trigger

The recording process can be started immediately when the command “RecStart” is sent. However, in many cases it is more useful to start the recording when a certain “trigger” event happens.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

The trigger source can be any parameter that is accessible through communication including array members.

The trigger source does not have to be one of the recorded values. For example, it is possible to record the values of position, position reference, velocity and current and the trigger will be a change in MotionStat value. This way, the recording can be started when the motion begins. The trigger source is compared to a set value (RecTrigVal), using a comparison operation that is determined by RecTrigTyp. All the options for comparisons are listed in the communication and keywords documentation. Some examples are: Greater than, Less than, Equal, Rising edge, Falling edge.

If only 1 bit of the trigger source should be used (or several bits), use RecTrigMask to mask the required bits.

Trigger location

Many times it is useful to see the value of the recorded vector for some time before the trigger. For example, the user may want to see the recorded values in the last second before motion started, and the motion takes 4 seconds. The entire duration of the recording will be 5 seconds. The trigger should be located 20% after the recording begins. To achieve this use RecTrigPos = 20.

When using a trigger that is not in the beginning position of the recording, data must be accumulated before the trigger event happens. To achieve this, the data is recorded all the time since RecStart command initiates the recording. The data is recorded cyclically until the trigger event happens.

The trigger event is not expected before there is enough data to fill the requested pre-trigger area. If we return to the example above, the pre-trigger time is 1 second. If MotionStat meets the trigger condition after 0.4 second, it will not be considered as a trigger. The recording will start waiting for a trigger only after 1 second.

How it continues depends on the type of trigger (RecTrigType). If the trigger condition requires the trigger source value to fill a condition, (eg. Greater then, equal etc.) The recording will “detect” a trigger immediately when 1 second passes. Otherwise, if the trigger condition requires a change in the trigger source value (such as rising edge or falling edge), the trigger will be detected only the next time it happens after more than 1 second.

Forcing trigger

In some cases the user may want to use a manual trigger. Use “RecTrigForce” command to force a trigger, or use the PC Suite button. When this command is received the recording mechanism will behave as if a trigger was detected regardless of whether an actual parameter was selected as trigger and what its value is.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

The recording parameters

A full explanation of each parameter can be found in the communication and keywords documentation. The recording parameters and functions are listed below for convenience with a short explanation of each:

Parameter/ command	Functionality
RecData	This is the array that holds the recorded data values and a header that describes the recorded information.
RecParam	An array of the parameters that will be recorded
RecLength	The number of data points that will be collected for each recorded vector
RecGap	The gap in sample times between each two data points
RecTrigSrc	The parameter that will be used as trigger source
RecUpload	The command that will cause the controller to send all the values in RecData through communication
RecTrigTyp	The type of event on the trigger source that will be captured as trigger
RecTrigVal	The value to which the trigger source is compared
RecTrigPos	The position of the trigger event in the recording (%)
RecStart	The command that starts the recording process
RecStat	Reports the status of the recording
RecStop	Stop the recording
RecTrigMask	A mask can be applied to use bits or bit fields as trigger source
RecTrigForce	Force a trigger manually

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Commutation

Commutation is the process of alternating the current between the different motor phases to generate motion (for DC Brushless motors only).

To commutate the current correctly the controller must have information about the electrical angle of the motor. This information can be received from Hall sensors, for example.

In the absence of Hall sensors, the electrical angle of the motor must be derived from the encoder readings, which are generally not aligned with the motor electrical angle. As a result, an initialization process is required in order to align the encoder reading and the motor's electrical phase.

With Akribis's controllers, this can be done in one of several methods.

ComtMode[] is an array that controls the electrical angle detection process.

ComtMode[1] selects which method will be used to detect the electrical angle:

- 0 – Jump to zero. The motor will jump from its present location to the electrical zero angle of one of the phases. This method usually involves a large, sudden movement.
- 1 – Minimal motion. In this method the motor will only perform a very small movement, and detect the electrical angle in its present location.
- 2 – Absolute encoder. For absolute encoder, perform angle detection once per each motor + controller set using one of the methods above. After the angle is detected use the Save command to save all the parameters. The absolute location of angle 0 will be saved in the controller's flash memory. In all the following uses of the same motor – controller set, use ComtMode[1]=2 to calculate the current electrical angle using the number saved in memory without any motion.

When using the "Jump to zero" method the user can determine the amount of voltage that will be applied to the motor during the process. If the voltage (and therefore the resulted current) is too low, mechanical factors in the system (such as friction) may prevent the motor from moving and cause failure of the process. High currents that are applied to the same phase for a long time can cause damage. The user should apply a current that is enough to move but not too much.

The voltage (and thus the current) to the motor is increased gradually over a period of time so the maximum current is only applied for a short time. Every voltage (current) is applied for a step duration of 10mSec.

ComtMode[2] : Used only in "jump to zero" mode. Determines by how much the voltage to the motor is increased after each step. The resulting current depends on the bus voltage and the motor resistance. To prevent damage it is recommended to set this number low during the first experiments and raise it only if necessary. If needed change ComtMode[3].

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

ComtMode[3] determines how many voltage steps will be applied during the angle detection process. Each voltage value is applied to the motor for 10 msec and then the voltage is increased. It is recommended to start with a 1 second detection process (ComtMode[3] = 100). After the detection is done, with motor disabled, move the motor manually to a different position and repeat the process. The motor is expected to move. If you don't see any motion, increase ComtMode[3] and move the motor again until you see a detection process that causes movement. As long as you don't see motion it is possible that the current is not enough to overcome friction or the motor was in the 0 position in the first place.

About the "Minimal motion" method:

With incremental encoder (or first using a motor with absolute encoder), there is no information about the electrical angle of the motor after power on. The "Jump to zero" method overcomes this difficulty by forcing the motor to jump into a known electrical angle (0 degrees). But this involves a relatively large motor's jump. The "Minimal motion" method performs a different process, to minimize this motor movement.

The idea in general is to temporarily, during the process (about 2-3 seconds), to close a control loop that tries to maintain the motor at its current position, while commanding the current control loop with a fixed current command (set be the user) and an electrical angle that is the output of the control loop. As a result, the control loop (trying to keep the motor at its position when the process was started) will find the suitable electrical angle that will keep the motor in this location although current is applied to the motor.

This is, by definition, the electrical angle of the motor position, minus 90 degrees.

So, the control loop finds the electrical angle of the motor, at its current position.

Since this angle is not known initially (a zero angle is assumed), the motor will slightly move, till the control loop will be stabilized into steady state, where the motor will be back in its original position.

In order to optimally overcome frictions and to ensure the accuracy of the process, after this phase (in closed loop) is completed, a second phase is performed. In this phase, a user defined current (higher than the one used in the closed loop) is applied in open loop to the motor, to force it into the position that is related to the detected angle.

Knowing the detected angle, and its related position, the drive can initiate the commutation variables (where is the "zero" of the electrical cycle) and is ready for motions.

When using the "Minimal motion" method the user shall determine the amount of current that will be used during the process, as follows:

ComtMode[6]: Is the current to use, in [mA], during the process of looking for the electrical angle in closed loop, as explained above. Typically, it is around 10% to 20% of the motor's peak current. It must be above the current that is required to overcome the system friction.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

ComtMode[7]: Is the current to use, in [mA], during the last phase of the minimal motion commutation process. In this phase, the current is applied in open loop, so that the motor is forced to the position that is related to the detected electrical angle. Typically, it is around 15% to 30% of the motor's peak current. It must be above the current that is required to overcome the system friction and for optimal performance, it shall be higher than ComtMode[6].

Note that while ComtMode[6] and [7] are used to set the current reference (CurrRef) during the process, the actual value of CurrRef is subjected to the drive current limitations, as defined by the user.

As the "Minimal motion" method involves a closed loop control to look for the electrical angle at the current position, it involves gain and integral terms parameters for the closed loop, as explained above. These parameters shall be set by the user as follows:

ComtMode[8]: Is the gain of the PI control filter. Typical value is 5000. Changes may be required depending on the system, the encoder resolution and the motor's number of poles.

ComtMode[9]: Is the integral gain of the PI control filter. Typical value is 20. Changes may be required depending on the system, the encoder resolution and the motor's number of poles.

ComtMode[4] holds the position of the electrical zero of a motor with absolute encoder. If the motor or controller need to be replaced, the electrical angle of the motor must be detected again and saved. The value of ComtMode[4] is automatically assigned following a commutation process with ComtMode[1]=0 or 1. This means that for a motor with absolute encoder, you shall first perform the detection process using ComtMode[1] = 0 (or 1) and then, after the process is successfully completed and ComtMode[4] is automatically updated by the controller, you can change to ComtMode[1]=2 ("Absolute encoder" mode) and save to the Flash, so that there will be no need for any motion in the next times.

ComtMode[5] is used to request a new commutation process. To repeat the commutation process enter ComtMode[5] = 1282. A new commutation process will begin and the value of ComtMode[5] will be cleared.

Commutation process is automatically performed following power on or reset.

Specific products notes:

1. Product 1:

    a. Currently, only the "Jump to zero" method is supported.
    b. Repeating the commutation process is done by ComtMode[1]=1282 (and not ComtMode[5]).

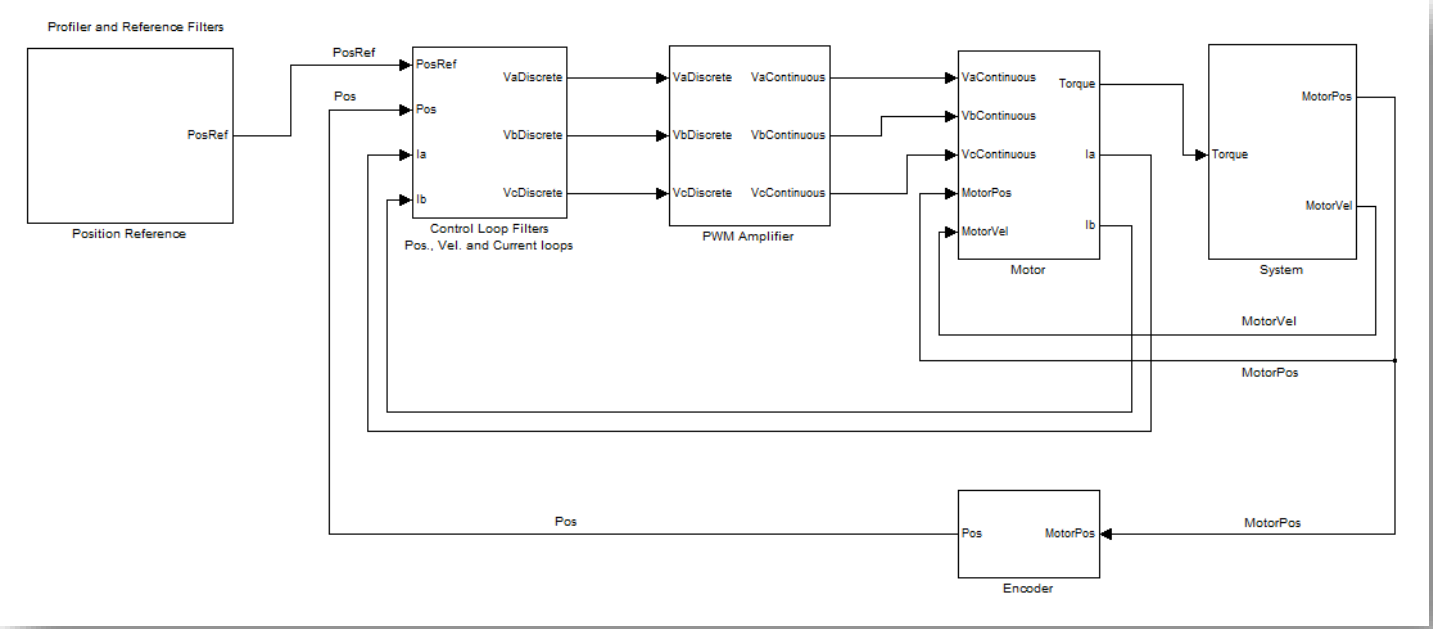
Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Control Filters

This chapter describes the structure of the control filters (Position, Velocity and Current control filters).

Overall structure

The overall structure of a controlled system, consisting of: servo drive (controller + amplifier), motor and load, is presented in the following figure:



The system consists of the following sub-systems:

- Controller:

  Includes the profiler, position reference filters and the Position, Velocity and Current closed loop control filters.

- Amplifier:

  Includes the PWM commands and PWM Bridge.

- Motor:

  The servo motor.

- Load:

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

The system load.

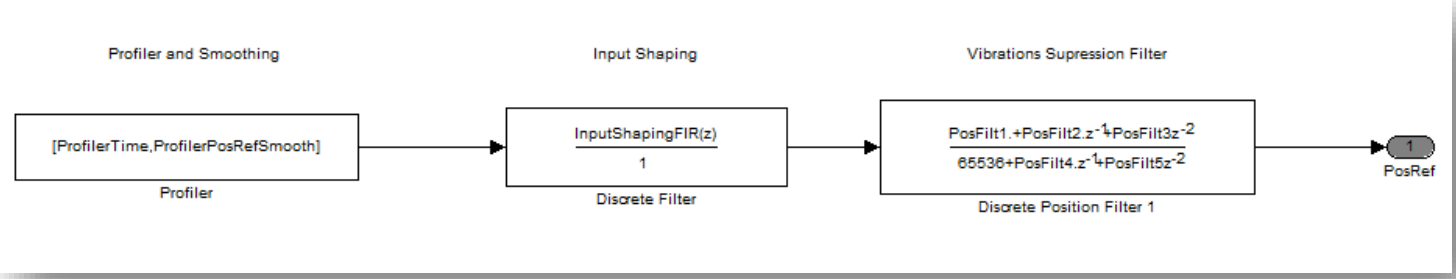
- Encoder:

  The position sensor.

The following chapters present the details of the position reference filters, as well as the control filters of the position, velocity and current control loops.

Position Reference

The following figure presents the structure of the position reference block:



PosRef, which is the output of the position reference block, is the position command for the position control loop, as described in the next chapter.

PosRef is the position reference, as generated by the profiler block (see below) and filtered using the Input Shaping and Vibrations Suppression Filters blocks.

The Profiler

Generates position trajectory, each control sample time, according to the currently active Motion Mode and motion parameters (Speed, Acceleration, PDFact …).

Refer to the chapter about Motions for detailed description of each motion mode and its parameters.

Input Shaping filter

Input Shaping algorithm is an algorithm that is applied on the position command (PosRef) in order to modify it so that oscillations at the load will be canceled.

Important notes:

- The Input Shaping algorithm can cancel (or reduce) oscillations only if they are created as a result of the motion itself.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

- The Input Shaping algorithm is effective to cancel oscillations that are not observed at the encoder feedback, just as it is effective to cancel observed oscillations.
- The algorithm is performed on the position reference, so it is not a part of the closed loop and has no effect on its stability of performance (for example: disturbance rejection).

The Input Shaping algorithm, with its implementation at Akribis's controllers, can reject (or reduce) the oscillations of 1 or 2 frequencies.

The user needs to measure the frequency of the oscillation and its damping factor. These values are entered to the controller as parameters (see below). Once the Input Shaping is activated, the relevant oscillations shall be cancelled (or at least significantly reduced).

The algorithm is quite tolerant to in accuracies of the specified frequency and damping. However, the more accurate the input data, the better will be the rejection of this oscillation.

Important note:

- The usage of the Input Shaping algorithm modifies the shape of the position reference profile. However, not only the shape is modified, but also its timing. With Input Shaping activated (with a single frequency), the profile time is extended by 1 cycle time of the frequency to reject. This means that the Input Shaping is effective for systems with oscillations of more than one cycle (low damping factor values).

Relevant parameters:

- ShapingOn:

  A value of "0" disables the Input Shaping algorithm.
  A value of "1" enables it.

- ShapingFreq[]:

  Defines the 1st (ShapingFreq[1]) frequency to reject and the 2nd (ShapingFreq[2]) frequency to reject.

  Use a value of "0" at ShapingFreq[2] for rejection of a single frequency.

  The value of ShapingFreq[] is the frequency value multiplied by 65536.

  For example, to reject a frequency of 10.25Hz, use: ShapingFreq[1]= 671744.
  This enables fractional frequencies.

- ShapingDamp[]:

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Defines the 1st (ShapingDamp[1]) damping factor and the 2nd (ShapingDamp[2]) damping factor.

The value of ShapingDamp[] is the damping factor value (a value between 0 to 1) multiplied by 65536.

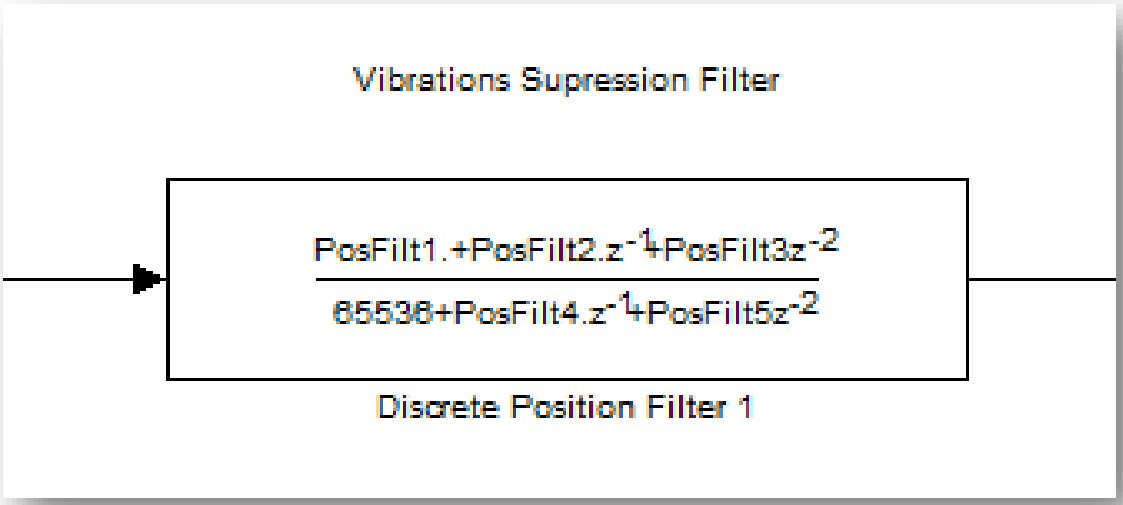
For example, to define a damping factor of 0.1, use: ShapingDamp[1]= 6554.

Important note for tuning:

- Typical tuning of a system shall start with this filter disabled. It can be applied and tuned during the more advanced phases of the system tuning.

Vibrations Suppression filter

The Vibrations Suppression filter is also a filter that is applied on the position reference. It is used to reduce load vibrations. The following figure shows the structure of the filter.



PosFilt[1] to PosFilt[5] are the filter parameters and the filter equation is:

$Y_K = (X_K * PosFilt[1] + X_{K-1} * PosFilt[2] + X_{K-2} * PosFilt[3] – Y_{K-1} * PosFilt[4] – Y_{K-2} * PosFilt[5]) / 65536$

Where X is the input to the filter and Y is its output.

Important notes:

- The user does not directly enter values into the PosFilt[] array parameter. Instead, the user uses the PosFiltDef[] and PosFiltOn[] parameters to define the filter characteristics (type, parameters and enable/disable).
- Many filter's types are supported. The typical filter to use for the Vibrations Suppression filter is a Notch filter. In some cases, a second order low pass filter may be useful as well.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

- Please refer to the Keywords and Communication Reference Manual, for the "PosFiltOn" keyword page, for detailed description of the available filter types and how to properly define the desired filter (or to disable this filter if it is not needed).
- Typical tuning of a system shall start with this filter disabled. It can be applied and tuned during the more advanced phases of the system tuning.

Accel/Decel Shaping

For some of Akribis's controllers, an additional input shaping algorithm is supported (although it does not appear in the above schemes).

With this algorithm, the shaping of the position trajectory (profile) is achieved by real-time modification of the acceleration and the deceleration of the motion, as a function of the distance between the current position reference to the final target position.

The idea of this algorithm is to enable fast small-distance point to point motions (this requires high acceleration and deceleration values) while enabling reduced deceleration at the end of the motion (the distance to the target becomes smaller and smaller), to minimize overshoot and to provide minimal settling time.

Careful tuning of this algorithm can provide significantly improved motion time for short motions.

Relevant parameters:

- AccShapeOn:

  Turns on (=1) and off (=0) the acceleration shaping feature. In acceleration shaping the user can select different factors to multiply the acceleration (and deceleration) depending on the distance from the target. This way, the acceleration is not constant and the user can choose, for example, to begin the motion with a large acceleration, then decrease it and also end the motion with gradual deceleration.

- AccShapeDist[]:

  It is an array that holds distances from the absolute target. If acceleration shaping is on and we are in a point to point motion, if the distance to the target is smaller than AccShapeDist[1], the acceleration is multiplied by AccShapeFact[1] (see below). If the distance to the target is between AccShapeDist[1] and AccShapeDist[2], the acceleration(or deceleration) is multiplied by AccShapeFact[2]. And so on.

- AccShapeFact[]:

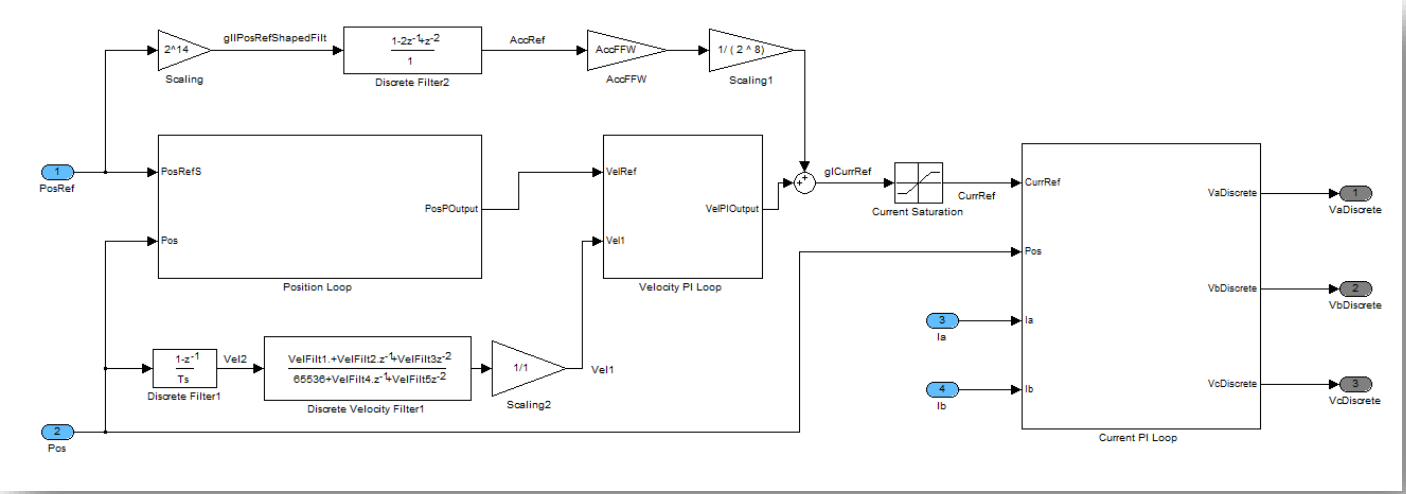
  It is an array that holds factors that multiply the acceleration and the deceleration. If acceleration shaping is on and we are in a point to point motion, if the distance to the target is smaller than AccShapeDist[1], the acceleration is multiplied by AccShapeFact[1]. If the distance to the target is between AccShapeDist[1] and AccShapeDist[2], the acceleration(or deceleration) is multiplied by AccShapeFact[2]. And so on.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Closed control loop filters

This chapter describes the structure of the closed loop control filters (Position, Velocity and Current).

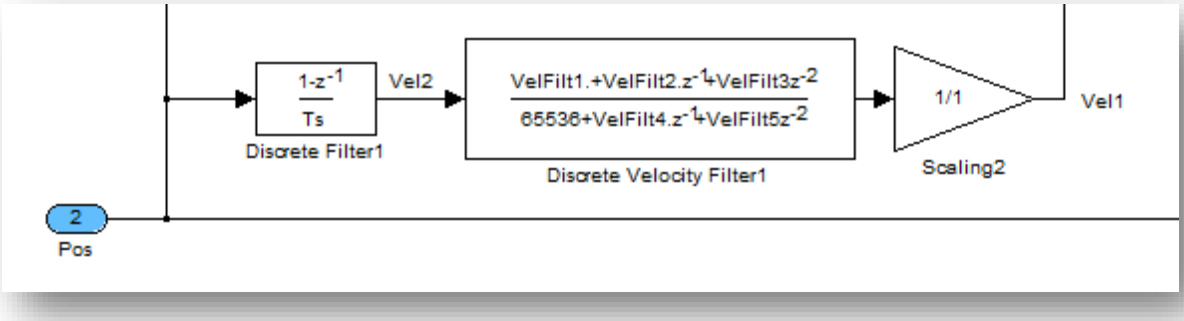
The following figure presents the general structure of the closed loop control filters:



This scheme shows a nested closed control loop (Position over Velocity over Current), as described in details in the following sections.

Velocity feedback

The velocity feedback is derived from the position feedback.



Vel[2] is a direct discrete derivative of the position feedback (Pos).

Then, a bi-quad filter is used to calculate Vel[1], which is the filtered feedback velocity. Vel[1] is used as the feedback for the velocity control.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

VelFilt[1] to VelFilt[5] are the filter parameters and the filter equation is:

$Y_K = (X_K * VelFilt[1] + X_{K-1} * VelFilt[2] + X_{K-2} * VelFilt[3] - Y_{K-1} * VelFilt[4] - Y_{K-2} * VelFilt[5]) / 65536$

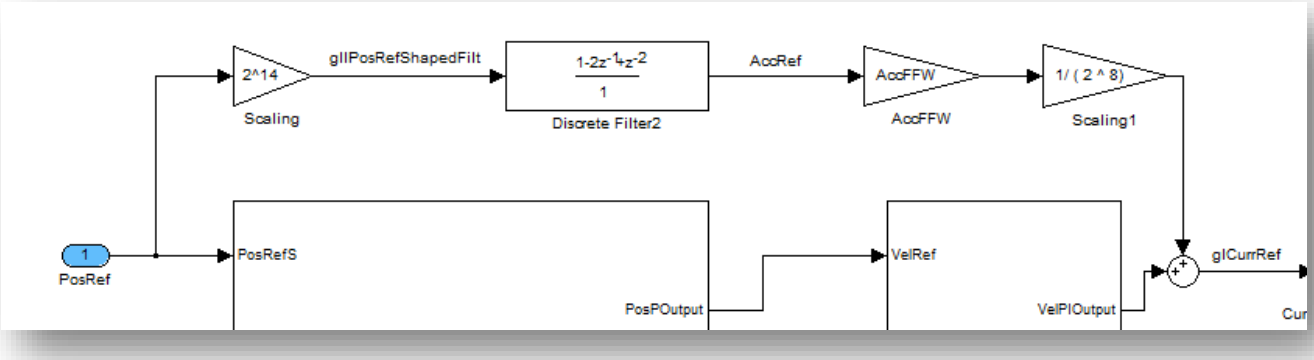
Where X is the input to the filter (vel[2] [1]) and Y is its output (Vel[1]).

Important notes:

- The user does not directly enter values into the VelFilt[] array parameter. Instead, the user uses the VelFiltDef[] and VelFiltOn[] parameters to define the filter characteristics (type, parameters and enable/disable).
- Many filter's types are supported. The typical filter to use for the velocity feedback filter is a second order low pass filter.
- Please refer to the Keywords and Communication Reference Manual, for the "VelFiltOn" keyword page, for detailed description of the available filter types and how to properly define the desired filter (or to disable this filter if it is not needed).
- Typical values for its bandwidth are 150 to 800 [Hz], but this depends on the application and the system. Typical damping factor is 0.5.

Acceleration Feed Forward (FFW)

The acceleration feed forward (FFW) enables the injection of current command that is



proportional to the second derivative of the position reference (actually: the acceleration reference).

Generally speaking, a proper setting of this AccFFW parameter will improve the control loop tracking, minimizing position errors and overshoots during motions.

Note the built-in scaling inside the acceleration feed forward equations.

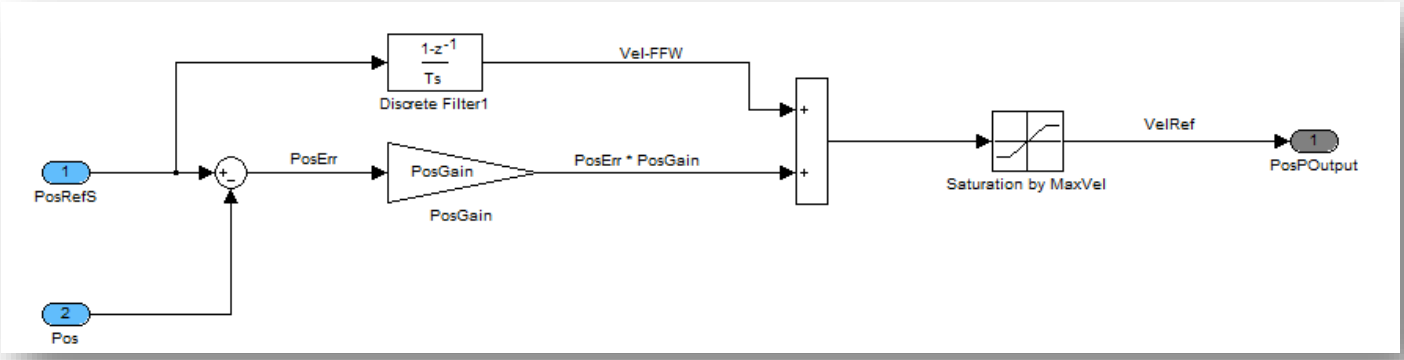
[1] Refer to the chapter regarding the Dual Loop mode. In this mode, Vel[1] is derived from the auxiliary encoder velocity, and not from Vel[2].

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

It is recommended to start the system tuning with AccFFW = 0, and to later gradually increate it to improve the system performance and reduce overshoots.

Position control

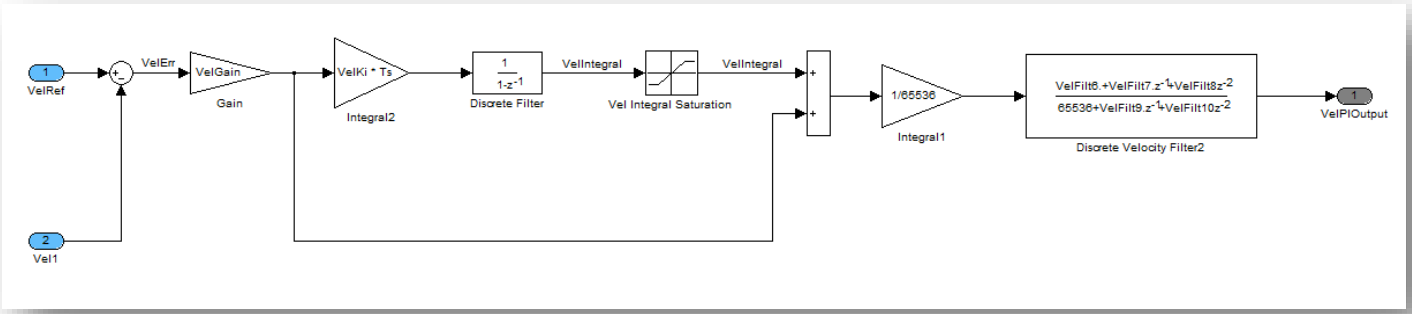
The following figure presents the position control equations:



It consists of a gain (PosGain parameter) and a built-in velocity FFW (required to provide the derivative term of the position reference). The output (VelRef) is the reference for the velocity control loop and is limited by the MaxVel parameter.

Velocity control

The following figure presents the velocity control equations:



It consists of a gain (VelGain parameter) and an integral term (VelKi parameter), cascaded with a bi-quad filter (Velocity Bi-Quad Filter 2).

The integral saturation is internally set to limit the integral term to PeakCL*65536, with PeakCL being a user defined parameter that defines the maximal current loop command (see below).

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Note the scaling of the VelKi parameter: Ts is the controller sampling time, typically 1/16384 [sec] (~61μsec).

VelFilt[6] to VelFilt[10] are the bi-quad filter parameters and the filter equation is:

$Y_K = (X_K * VelFilt[6] + X_{K-1} * VelFilt[7] + X_{K-2} * VelFilt[8] – Y_{K-1} * VelFilt[9] – Y_{K-2} * VelFilt[10]) / 65536$

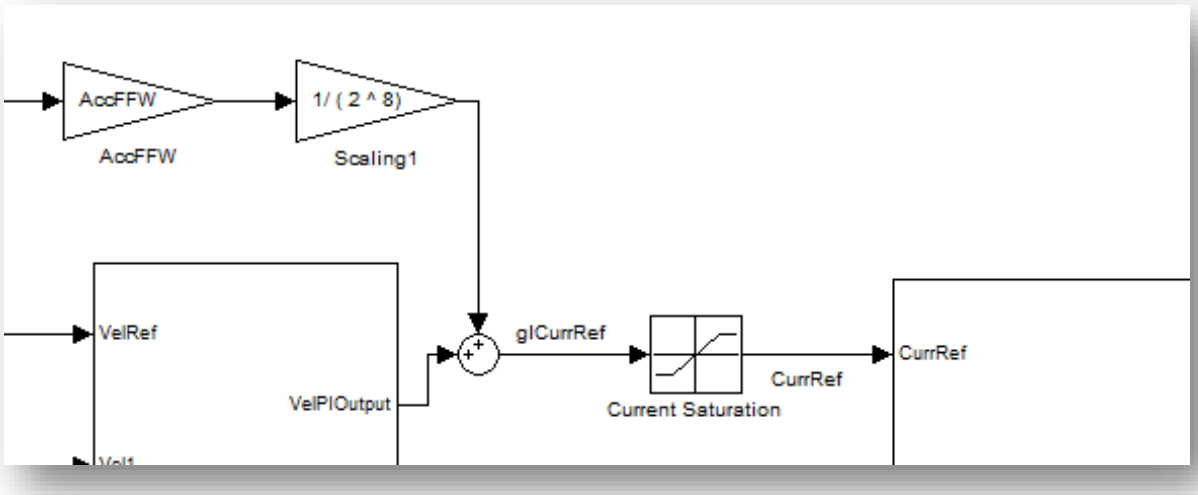
Where X is the input to the filter (output of the velocity PI) and Y is its output (actually the command to the current control loop, see below).

Important notes:

- The user does not directly enter values into the VelFilt[] array parameter. Instead, the user uses the VelFiltDef[] and VelFiltOn[] parameters to define the filter characteristics (type, parameters and enable/disable).
- Many filter's types are supported. The typical filter to use for the velocity filter 2 is either a second order low pass filter or a Notch filter (if needed). However, other filters can be used, such as lead-lag.
- Please refer to the Keywords and Communication Reference Manual, for the "VelFiltOn" keyword page, for detailed description of the available filter types and how to properly define the desired filter (or to disable this filter if it is not needed).
- It is recommended to start the tuning process of a new system with this filter disabled and later on to consider a second order low pass filter or a Notch filter (values strongly depends on the application and the system).

Current command saturation

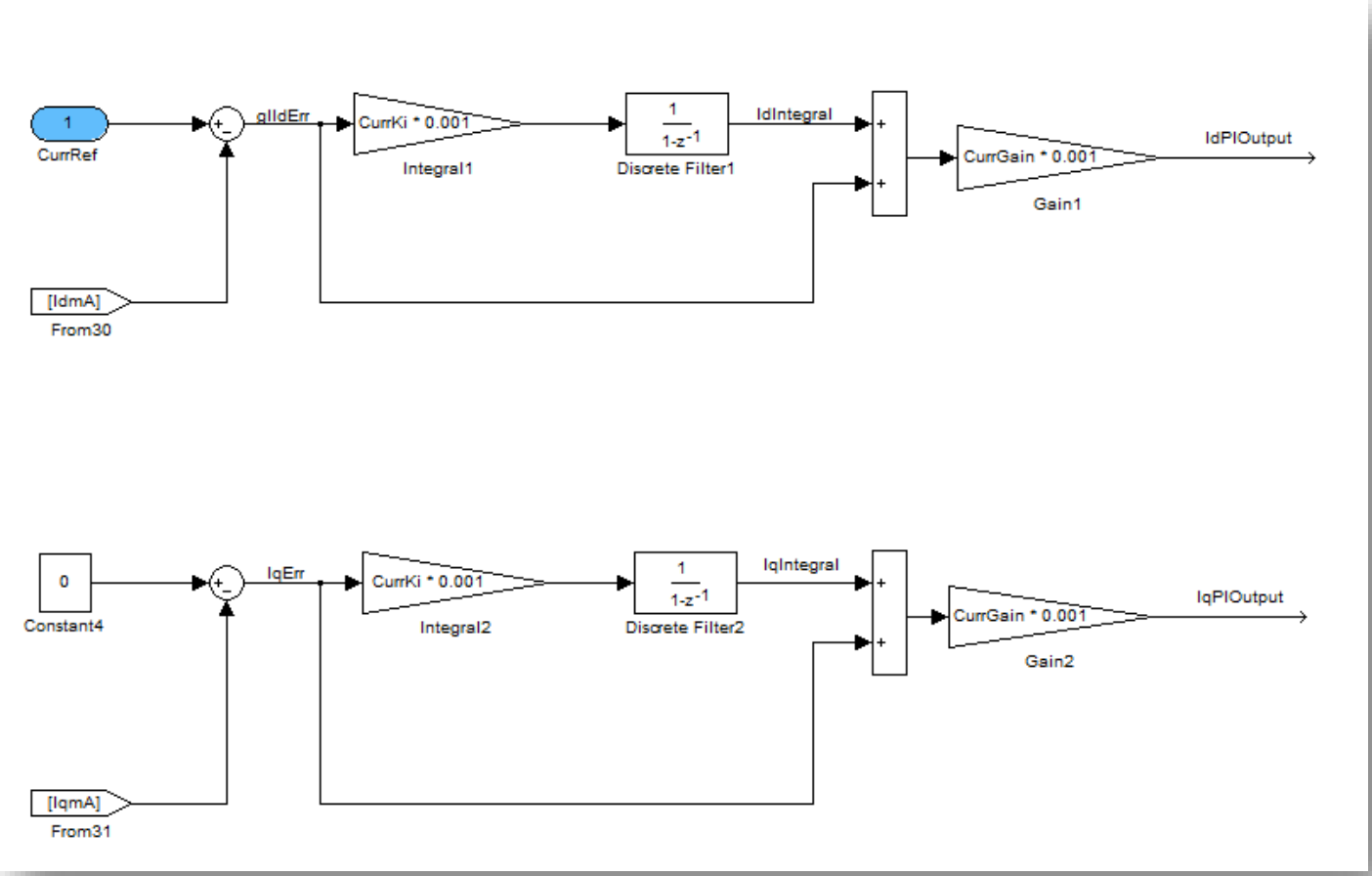
The output of the velocity control filter is summed with the acceleration feed forward and then limited by the PeakCL (Peak Current Limit) parameter, so that the command to the current control loop (and thus the actual motor's current) will not exceed PeakCL (in [mA]).



Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Current Control

The current control is implemented using a Vector Control scheme, as (partially) shown in the following figure:



Notes about the current control scheme:

- The current control filter is PI. Note the built-in scaling of the CurrKi and CurrGain parameters.
- Two control loops are implemented.
  - The first is on Id (Direct Current) that uses CurrRef (output of the velocity control loop and the acceleration feed forward, see above) as its reference. This control loop is responsible to control the effective motor's current and to actually generate the desired torque.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

- The second is the quadrature current control loop. Its reference is zero as it is desired to have only the effective direct current, for optimal and efficient motor operation (field weakening and phase advance are currently not supported).

- The above figure does not show it, but of course:
  - Id and Iq are calculated from the measured Ia and Ib (motor's phases currents), using the Park transformation.
  - Va and Vb (PWM commands to the amplifier) are calculated from Vd, Vq (in the figure: IdPIOutput, IqPIoutput, respectively) using inversed transformation.
  - Vc is calculated as: Vc = -(Va + Vb).

Important notes:

- Some of Akribis's products (especially those for low voltage and VCM motors) do not implement current control loop and the output of the velocity control loop is directly used as the PWM command.
- While it is not useful and not needed in most of the cases, Akribis's controllers supports also a simplified, none-vector, current control scheme, where the control loops of the current independently controls Ia and Ib.
  - This mode is enabled using the ControlMode parameter. Please refer to the ControlMode keyword page at the Keywords and Communication Reference manual.
  - When using this mode, the current control tracking has reduced performance at high speeds and it may result with inefficient operation of the motor and extra heading (as Iq is not zero).

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Special Control Features

This chapter describes the special control algorithms that are supported by Akribis's controllers.

Gain Scheduling

Gain Scheduling is the process in which the controller automatically switches between different sets of control filter parameters as a function of some pre-defined criteria. It may be as a function of the motor speed (very low velocities do require a different control filter), or motion/settling/no motion, or distance to target and so on.

Akribis controllers, at the moment, provide gain scheduling for the following control filter parameters: PosGain, VelGain, VelKi, VelFFW (not supported by all controllers) and AccFFW.

All these parameters are arrays, with a typical (product dependent) size of 5 elements. Therefore, up to 5 sets of control filters can be defined.

Set number [1] (PosGain[1], VelGain[1], VelKi[1], AccFFW[1]) is the default set that is used after power on or reset and if the gain scheduling is not active.

Relevant parameters:

- ScheduleSet:

  Reports the number of the currently activated control set. It is "1" upon power on or reset and it is then controlled by the Gain Scheduling algorithm, according to its operation mode (see below: ScheduleMode).

  ScheduleSet can be also set manually by the user, of the ScheduleMode is set to manual mode.

- ScheduleMode:

  Defines the Gain Scheduling operation mode. Few modes are supported:

  - SCHEDULE_MODE_NONE 0

    Gain scheduling is disabled.

  - SCHEDULE_MODE_MANUAL_DINPORT 1

    The active control set can be controlled manually (by writing to ScheduleSet), or it can be controlled using one of the digital inputs by proper setting of DInMode[].

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

- SCHEDULE_MODE_OPTIMAL_SETTLING_BY_TIME 2

  Set 1 is used during motion, Set 2 is used during settling and Set 3 is used at all other times.

  Settling is a user defined time, in msec, that starts from end-of-motion, using the ScheduleTime parameter.

- SCHEDULE_MODE_OPTIMAL_SETTLING_BY_IN_TARGET 3

  Set 1 is used during motion, Set 2 is used during settling and Set 3 is used at all other times.

  Settling is the time from end-of-motion till the axis is settled into the target position (refer to InTargetStat, InTargetTime, InTargetTol).

- SCHEDULE_MODE_BY_VELOCITY_RANGE 4

  Set 1 is used while velocity is below ScheduleVel[1], Set 2 is used while velocity is below ScheduleVel[2] (and above ScheduleVel[1]) and so on …

  Set 5 is used while velocity is above ScheduleVel[4].

- SCHEDULE_MODE_BY_POSITION_RANGE 5

  Set 1 is used while position is below SchedulePos[1], Set 2 is used while position is below SchedulePos[2] (and above SchedulePos[1]) and so on …

  Set 5 is used while position is above SchedulePos[4].

- SCHEDULE_MODE_BY_QUIET_STANDING 6

  Set 1 is used after a user defined time (using ScheduleTime parameter) of no-motion. Set 2 is used at all other times (during motions and for this defined time after the motion).

- SCHEDULE_MODE_BY_PD_PULSES 7

  Set 1 is used after a user defined time (using ScheduleTime parameter) of no input pulses at the pulse/direction input port. Set 2 is used at all other times (during incoming pulses and for this defined time after the last pulse).

- SCHEDULE_MODE_BY_TEMPERATURE_RANGE 8

  Supported only by some of Akribis's controllers. Please check with Akribis.

  Set 1 is used while the temperature (PwrTemp) is below ScheduleTemp[1], Set 2 is used while the temperature is below ScheduleTemp[2] (and above ScheduleTemp[1]) and so on … Set 5 is used while the temperature is above ScheduleTemp[4].

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

- ScheduleTime:

  Defines a period of time, in [msec], that is used by some of the gain scheduling modes (see above).

- SchedulePos[]:

  Defines a set of position values that is used by the gain scheduling mode SCHEDULE_MODE_BY_POSITION_RANGE, as explained above.

- ScheduleVel[]:

  Defines a set of velocity values that is used by the gain scheduling mode SCHEDULE_MODE_BY_VELOCITY_RANGE, as explained above.

- ScheduleTemp[]:

  Supported only by some of Akribis's controllers.

  Defines a set of temperature values that is used by the gain scheduling mode SCHEDULE_MODE_BY_TEMPERATURE_RANGE, as explained above.

Dual loop

DualLoopOn is used to enable dual loop control structure.

If DualLoopOn == 0 (the default value), the dual loop control structure is disabled and the default control loop structure is used. In this mode, only the main encoder is used for both the position feedback (Pos) and for the velocity feedback (Vel[1] = filter(Vel[1]).

If DualLoopOn == 1, the dual loop control structure is enabled. In this mode, the main encoder reading (Pos) is used for the position feedback and the auxiliary encoder reading (AuxPos) is used to derive the velocity feedback (Vel[1] = filter(AuxVel).

Note that although the auxiliary encoder speed is reported at AuxVel and the main encoder velocity is reported normally at Vel[1], Vel[2] and Vel[3], when the dual loop structure is used, Vel[1] (used as the feedback for the velocity control loop) is the filtered velocity of the auxiliary encoder and not of the main encoder.

This means that while typically Vel[2] is used as the input to the 1st velocity bi-quad filter (whose output is Vel[1]), when dual loop is enabled, the AuxVel is used as an input to this filter.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Due to the built-in structure of the closed loop control filter (specifically, due to the built-in velocity feed-forward) the main encoder (used for position feedback) and the auxiliary encoder (used for the velocity feedback) must be properly scaled when dual loop is enabled.

Proper scaling means that if the motor is moving at a given velocity, the reading of Vel[2] and AuxVel will be identical. This is a function of the main and the auxiliary encoder resolutions and the machine structure.

Scaling is supported using the DualLoopFact parameter.
The AuxVel is multiplied by DualLoopFact/65536 before it is used as the velocity feedback.

Enhanced speed range

The Enhanced speed range operational mode is activated using the ControlMode parameter.

To activate a given special control algorithm, the bit that corresponds to this algorithm should be set at ControlMode.

The default value of ControlMode is 0.

To activate the enhanced speed range algorithm set bit 0 (0x0001).

What is the enhanced speed range mode?

Basically, the maximal speed that a given system can reach is defined by the BEMF of the motor and the DC Bus voltage of the power supply[2], so that the motor BEMF (a direct function of the speed) can't be higher than VBus.

Or:

Maximal Speed = VBus / Ke

Ke: motor BEMF constant

The enhanced speed range operation mode is a special algorithm that enabling motor speeds (without any degradation of torque or motor efficiency!) to be up to 15% higher comparing to this limitation.

This is done by real-time management of the voltage at the motor neutral point.

Auto-Gain, Auto-Inertia, Auto-PID

Please refer to the "Inertia Auto Tuning Manual for detailed description of these algorithms.

[2] Indeed, the maximal speed is also a function of the required motor's current and motor's resistance, but this is ignored here for the simplicity of the explknations.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Advanced Feedback Features

Lock and Event

Lock and event (also known sometimes as capture and compare) are ways of attaching a signal to a certain feedback position.

The “Lock” feature saves the feedback location when a hardware signal change happens. For incremental encoder this is done by hardware, so the exact position is read even if the input signal changes between controller samples.

The “Event” controls a controller output that is changed when a selected position is reached by the motor.

Due to the internal implementation of the controller, “Lock” and “Event” use the same hardware, so only one of these features can be active at any time. When “LockEn” is set to “1”, “EventOn” is automatically reset to “0” and the other way.

Lock

If the lock feature is enabled (LockEn = 1), when the value of the lock input changes, the position reading is saved in LockVal and LockCntr is increased by 1.

The number of the digital input that is used as the lock source is determined by LockSrc.

Note:

Maximum one lock per sample time is handled. If more than one lock occurs only the 1st is handled.

When using absolute encoders the position cannot be captured at the hardware level, so the position on the next interrupt is locked.

The same set of parameters also exists for the auxiliary feedback, using the prefix “aux” before each parameter.

Event

Event will change the value of an output when the position reaches a certain predetermined value.

There are three possible types of events (determined according to EventType):

0 – Single event
1 – Events by gap
2 – Events by table

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

The output that is used for event is a default output. See the hardware manual for details.

Single Event
A single pulse appears on the output when the position reaches the value in EventBegPos.

Events By Gap
There is a pulse on the output starting from EventBegPos and whenever the position increases by EventGap until EventEndPos is passed.

Events By Table
The events don't have to be equally spaced. A table of positions for the events can be stored in GenData array. The index to the first position should be entered into ETStart. The index to the last position should be entered into ETEnd.
The positions in the table must be arranged in increasing order.

Pulse Width
The output pulse is generated by the FPGA that is running on an 80MHz clock. The pulse width (duration) is set to the desired number of clocks.

Error Mapping

The feedback encoder is assumed to provide the accurate position of the motor (or the load, depending on the machine structure).

However, in some application, the encoder reading does not accurately measure the motor or (more relevant) the load position.

Why? Many reasons can be considered. For example:

1. The encoder is attached to the motor, while the mechanical transmission between the motor and the load (whose position is of interest) is not constant.
2. The mechanical structure of the system is not linear (decentralized wheel transmission).
3. The system is an XY system, and the XY are not exactly 90 degrees from each other.

For such cases, Akribis's controllers support 1D (1 dimensional) and 2D (2 dimensional) encoder error corrections mechanism.

The user can define a map (table) of the position reading errors, and the controller will automatically (each sample time) correct the encoder reading so that the final position reading (at the Pos keyword) will have a corrected value, reflecting, as much as possible, the actual position of the motor/load.

Note that since the position reading correction is done at each sample time, and is done on the feedback reading, it does not only correct the final motion position, but also the velocity along the motion.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

The error mapping is a relatively complex feature. It is strongly recommended to use the PC Suite (Feedback/Error Mapping window) to access and to use this functionality.
Two operational modes are supported:

- 1D Error mapping:

  The main encoder reading is corrected as a function of a single encoder input (typically the main encoder itself) and a list of corresponding errors.

- 2D error mapping:

  The main encoder reading is corrected as a function of a two encoder inputs and a 2D table of corresponding errors.

The relevant parameters are:

- MapType:

  If MapType == 0 Error correction is disabled.
  If MapType == 1 1D error mapping is activated.
  If MapType == 2 2D error mapping is activated.
  Use this parameter to enable/disable the error correction/mapping and to define its mode (1D or 2D).

- MapEncoder[]:

  MapEncoder[] defines which encoder feedback is used as an entry value into the error correction/mapping table.

  MapEncoder[] is an array with two entries. MapEncoder[1] is used for 1D error mapping and also as the first encoder for 2D error mapping. MapEncoder[2] is only used for 2D error mapping, used as the second encoder definition.

  If MapEncoder[1/2] == 1 The main encoder of the A axis is used as the entry to the table.

  If MapEncoder[1/2] == 2 The auxiliary encoder of the A axis is used as the entry to the table.

  For multi-axes controllers only:

  If MapEncoder[1/2] == 3 The main encoder of the **B** axis is used as the entry to the table.

  If MapEncoder[1/2] == 4 The auxiliary encoder of the **B** axis is used as the entry to the table.

  And so on …

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

- MapTable[]:

  MapTable[] is a large array that holds the table of errors to be used for the error correction.

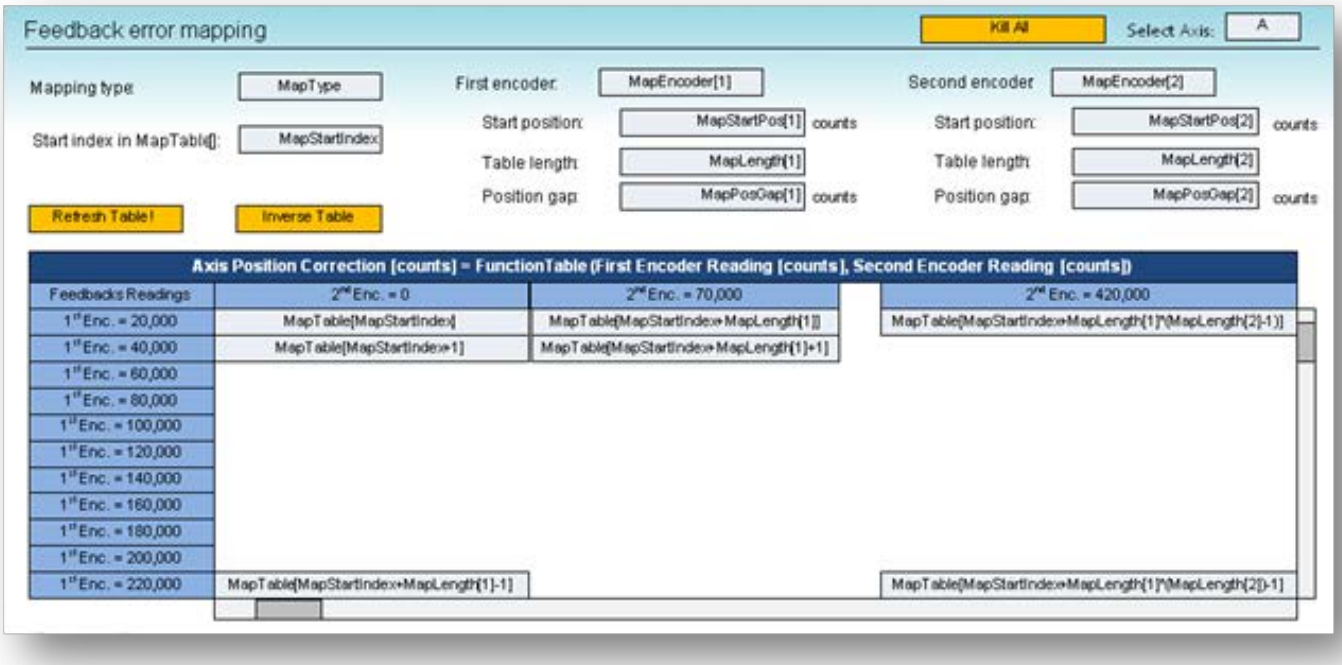
  For 1D error correction, the user should place a list of values (error values) at any location within the MapTable[]. The MapStartIndex shall point to the index at MapTable[] where the first element in this list is stored.

  Note that MapTable[] starts at MapTable[1] (like all arrays at Akribis's controllers).

  Each error in the list corresponds to a given input value at the (selected) encoder input. It is assumed that the first point (MapTable[MapStartIndex[1]]) corresponds to MapStartPos[1] and the following points correspond to equally spaced input positions, using MapPosGap[1]. The number of table entries is defined by MapLength[1].

  For 2D error correction, the MapTable is built from a list of errors as defined above, per each value of the second encoder input (which starts at MapStartPos[2] and increments by MapPosGap[2], for a MapLength[2] number of points). The 2D table is organized within MapTable[] (which is a 1D array) as a series of lists. First is the list for the first value of the second encoder, and then comes the list of the second value and so on.

  Refer to:



Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

- MapStartIndex:

  The index of the first element at MapTable[] that holds the beginning of the errors table. Typically use a value of 1, unless you would like to store multiple corrections tables at MapTable[] (this is possible and limited only by the size of the MapTable[] array).

- MapStartPos[]:

  MapStartPos[1] defines the input encoder reading that corresponds to the beginning of the table for 1D error correction and the input value of the 1st encoder at the first point of the table in 2D error correction.

  MapStartPos[2] is used only for 2D error mapping and defines input value of the 2nd encoder at the first point of the table in 2D error correction.

- MapLength[]:

  MapLength[1] defines the number of entries at the errors table, for 1D error mapping. MapLength[1], for 2D error mapping, defines the number of entries that relates to the 1st encoder input (the number of "rows" at the 2D error mapping table).

  MapLength[2], is used only for 2D error mapping, and defines the number of entries that relates to the 2nd encoder input (the number of "columns" at the 2D error mapping table).

- MapPosGap[]:

  MapPosGap[1], for 1D error mapping, defines the input encoder gap (in [counts]) between two consecutive entries to the table.

  For 2D error mapping, MapPosGap[1] defines the gaps of the 1st encoder, while MapPosGap[2] (used only in 2D error mapping) defines the gap for the 2nd encoder.

How the corrected position reading value is calculated?

For 1D error mapping:

- The relevant encoder input is read (it can be the main encoder or the auxiliary encoder, but typically it is the main encoder of this axis).
- If the encoder input value is smaller than MapStartPos[1], the first error value is used (MapTable[MapStartIndex]).
- Else if the encoder input value is higher than MapStartPos[1]+(MapLength[1]-1)*MapPosGap[1] the last error is used (MapTable[MapStartIndex + MapLength[1] - 1]).
- Otherwise, the related pointer into the table is calculated and a linear interpolation is used between the relevant two table entries, to calculate the resulted error value.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

- The resulted error value is **added** to the main encoder reading, resulting with the feedback position at the Pos parameter. Note that the error is added to the encoder reading (this affects the way the user shall measure and calculate the table of errors).

For 2D error mapping:

- The process is very similar, but:
- The process is first applied on the 1st encoder.
- The process is then applied on the 2nd encoder.
- Now that we have the rectangle within the error is located, a process of 3 linear interpolations is executed in order to find the resulted 2D error result.
- This error is now used as described above to correct the main encoder reading.

Advanced Low Speed Measurement ("1/T" or "One over T")

1/T (OneOverT) is a process in which the motor velocity (using the main encoder feedback) is measured accurately, especially for low speeds.

The motor velocity, as reflected at the Vel[] parameter is derived from the encoder feedback by a simple derivative, as follows:

$Vel[2] = (Pos_K - Pos_{K-1}) / Ts$

(Ts is the controller sampling time, typically with Akribis's controllers: 1/16384=61μs).
This creates a reading with poor resolution of 1/Ts (typically 16384).

As a result, if the motor velocity is, for example, 10000 [counts/sec], Vel[2] will show values of 0 and 16384.

This is, of course, not enough for accurate velocity measurement.

Indeed, Vel[1] is a filtered value of Vel[2], and Vel[3] is a moving average filter of Vel[2], but still, they do not provide an accurate and immediate velocity sensing, especially at low speeds.

At this point, the 1/T mechanism provides and alternative method to measure the motor velocity from the encoder feedback.

Generally speaking, the idea is to measure the time duration of the encoder input pulses. This is done between changes at the encoder input, using a timer that is fed by a very fast frequency clock. Now, if we have the time duration (let's call it T) of, for example, 1 encoder count, we can calculate the accurate velocity by:

Vel = 1/T.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

To enable accurate and valid measurement both at high and low speeds, the user can control the number of encoder input pulses (whose time period is measured) and the frequency that is used to measure this time period.

Higher frequencies will provide better accuracy for the measurement, but the time can overflow at low speeds (in case of overflow, the velocity reading is set to 0).
Higher number of encoder pulses will provide better accuracy, but the reading will be less immediate and the time can also reach overflow.

The velocity that is calculated using the 1/T mechanism is reported at Vel[4].

Relevant parameters:

- OneOverTOn:

  Set to 0 to disable the 1/T measurement (Vel[4] reports 0).
  Set to 1 to enable the 1/T measurement.

- OneOverTGap:

  Defines the number of encoder pulses that are used to measure the time period.
  The actual number of pulses is $2^{OneOverTGap}$.

  Note that a value of 4 (or multiplication of 4) for the actual number of pulses will provides a more accurate velocity reading because it will not be affected by the shift between the A and B encoder signals (which are not always shifted exactly by 90°).

- OneOverTFreq:

  Defines the frequency that is used to measure the time duration.

  The actual frequency, in Hz, is given by:

  1/T frequency = SYSTEM_CLOCK / $2^{OneOverTFreq}$

As a result, the equation that is used to calculate the 1/T reading is:

Vel[4] = Number of pulse [counts] / Measured time [sec]
Vel[4] = $2^{OneOverTGap}$ / (TimerValue * TimerPeriod)
Vel[4] = $2^{OneOverTGap}$ * Timer Frequency / TimerValue
Vel[4] = ($2^{OneOverTGap}$ * SYSTEM_CLOCK / $2^{OneOverTFreq}$) / TimerValue
Vel[4] = SYSTEM_CLOCK / TimerValue * ($2^{OneOverTGap}$ / $2^{OneOverTFreq}$)

Note that the 1/T velocity reading is valid only for incremental encoder input.

Note that the 1/T velocity reading (reported at Vel[4]) is used only for reporting (data recording) and is not used as part of the closed control loop filters.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Digital I/O's Special Functions

Digital Input Port

The digital input port is a series of input ports.

Each input can be individually configured to customize its functionality. The configuration will determine the input response to the level of the input signal or a change in the level.

Input configuration

The input configuration is done by using the following parameters:

Parameter	Meaning
DInLog	Determines whether the logic used in any individual input is inverted or not
DInPort	Read the value of the digital input port. Each input is represented by a bit.
DInMode	An array the size of the input port that assigns the required functionality to each input.
DInFilt	A filter that is applied to the pulse/direction input

The digital inputs can be configured using DInMode to any of the following functions:

Value	Input Mode
0	User input (general purpose input, frequently used in user programs)
1	Not in use
2	Motor on (the motor is enabled if this input is high)
3	Begin motion
4	Stop motion
5	Clear input pulse: When the motor is on and this input is set (rising edge) stop the motion and disable the motor. On a falling edge begin a new motion.
6	Abort / resume motion
7	Alarm reset: To function, the alarm reset signal must be on for at least 20 mSec. After that time, ConFlt is cleared. If an output was assigned the functionality of alarm this output is reset.
8	Abort
9	Reverse limit switch: If the motor is moving in the negative direction it will stop when this input goes high.
10	Forward limit switch: If the motor is moving in the positive direction it will stop when this input goes high.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

11	Torque limit on
12	Activate dynamic brake (not supported by all types of drives, please consult Akribis)
13	Activate motor brake (not supported by all types of drives, please consult Akribis)
14	Change velocity gain: see ScheduleSet. A change in the input will select one of two possible values for the velocity gain.
15	Add filter
16	Switch between position control (high) and velocity control (low). The changes will apply only when disabling the motor.
17	Switch between velocity control (high) and current control (low). The changes will apply only when disabling the motor.
18	Switch between position control (high) and current control (low). The changes will apply only when disabling the motor.
19	Clear absolute encoder

Digital Output Port

The digital output port is a series of digital outputs that can be used for various functions.

The values of the bits in **DOutPort** affect the discrete output port of the controller as follows:

Bit 0 of DOutPort corresponds to output 1, bit 1 of DOutPort corresponds to output 2 etc.

The actual state of the output also depends on other related parameters:

The value of bit 0 of DOutPort is XORed with bit 0 of DOutLog.
The value of bit 1 of DOutPort is XORed with bit 1 of DOutLog.
And so on…

If DOutMode[x] is not 0, the output state of the relevant output will not be effected by the value of the relevant bit in DOutMode. The output will reflect only the special functionality.

Example:

If DOutPort = 0x03
And DOutLog = 0x01
And all the values in DOutMode are 0 then:
The value of output 1 is inverted, since bit 0 of DOutLog is 1. Bit 0 of DOutPort is 1. The XOR of the bits is 0 (or you can think of it as 1 inverted) so output 1 will be off.
The value of output 2 is not inverted, since bit 1 of DOutLog is 0. Bit 1 of DOutPort is 1 and output 2 will be on.
If we now change DOutMode[1] to reflect the motor on state, the value of output 1 will be on when the motor is disabled and off when the motor is enabled, since the logic of this output is inverted. The value of the bit in DOutPort has no effect on the output state.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

DOutMode is an array that determines the functionality of every digital output. To assign a functionality to digital output no. 1 enter the appropriate number into DOutMode[1]. To assign a functionality to digital output 2 enter the appropriate number to DOutMode[2] and so on.
The possible values for DOutMode and the corresponding functions are listed in the table below:

Number	Functionality
0	User output
1	Alarm – the assigned output will be on when ConFlt is not 0
2	End of motion
3	Motor on

Example:

To see the motor on status on output 4:
DOutMode[4] = 3
If the logic of output 4 is not inverted, output 4 will be on when the motor is enabled and off when it is disabled.

DOutLog can be used to invert the logic of the digital outputs. If the bit in DOutLog that corresponds to an output is 0, the logic of that output is unchanged. 1in DOutPort will result in an output in “on” condition, and 0 will be “off”.
If the corresponding bit in DOutLog is 1 the logic of the output is inverted: 1 will result in “off” and 0 will be “on”.
Note that bit 0 (LSB) of DOutLog corresponds to input 1.

DOutType: Some of the digital outputs of the controller can be configured as either sink or source. See the hardware user guide to find out which outputs and connection details.
To use an output as sink set the corresponding bit in DOutType to 0 (default). To use an output as source set the corresponding bit in DOutType to 1.

Note that bit 0 (LSB) corresponds to output 1.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Analog Input Port

The analog input to the controller allows an analog signal or signals to be used. The input analog signal may be used for many different purposes. Some common examples are: analog command (as when using a joy stick to control a motor, or for velocity or current command in Velocity Control Mode and Current Control Mode), changing current or torque limit, monitoring the status of a system component (temperature for example).

The values read from the analog input port are entered into AInPort[n*2] array. The size of the array depends on the number of analog inputs available in the controller. Please refer to the hardware manual to find out the number of inputs in your product.

AInPort[] is a read only array that contains the processed and original reading of the analog input port. The first half of the array holds the readings after they were filtered and after offset, dead band and gain were applied.

The second half of the array holds the original values of the input.

AInFilt[] is an array. Each location in the array corresponds with an analog input. The value that is entered to AInFilt[1], for example, will determine the coefficients of the filter applied to input1.

The coefficients are C_1 = AInFIlt[1]/65536 and C_2 = (1- C_1)

The filter is:

Filtered Analog Input = C_1 * Current input + C_2 * Previous filtered value

To have an unfiltered reading C_1 should be 1, so AInFilt[1] = 65536.

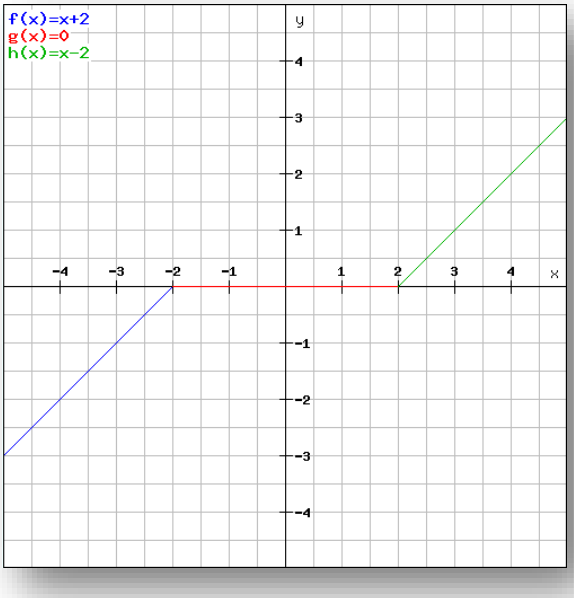
AinGain[] is an array. Each location in the array corresponds with an analog. It is the gain of the analog input multiplied by 65536. The multiplication is needed to allow representation of fractions using only integers. For gain = 1, AInGain = 65536.

AInOffset[] is an array. Each location in the array corresponds with an analog input. It determines the offset for the analog input in millivolt. The value of AInOffset is added to the actual voltage reading on the input.

AInDB[] is an array. Each location in the array corresponds with an analog input. It is the analog input dead band in milliVolts. If the actual input voltage is between (- AInDB) and AInDB the input reading will remain 0. Voltages above AInDB will be read as (actual voltage – AinDB). Values below (-AInDB) will be read as (actual voltage + AinDB).

The filter is calculated first. The offset is added to the input reading next. After adding offset the result is compared to the dead band (AInDB), the gain (AInGain) is applied to the result.
For example, if the dead band is 2 (no offset or gain), the voltage reading as a function of the input voltage will look like this:

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945



Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Analog Output Port

Please refer to your product's Hardware Manual for the number of available analog outputs. Each of the outputs behaves as described below.

The analog output can be directly controlled by the user or can automatically output the value of any another parameter (for monitoring).

The value entered into AOutPort[1] will appear on Analog Out 1 if AOutOode[1] = 0. The value entered into AOutPort[2] will appear on Analog Out 2 if AOutMode[2] = 0 and so on, depending on the number of analog outputs at your product.

The value of AOutPort[] is in [mV]. So, of AOutPort[2] = 120 and AOutMode[2] = 0, the analog output port 2 will have a value of 120mV.

If AoutMode[n] is not equal to 0, it is considered to be a Complex CAN Code (refer to Appendix I) that points to the parameter whose value shall be sent to the analog output port n.

So, if AoutMode[1] = 18 (Complex CAN Code of PosErr), than analog output 1 will reflect the value of PosErr. The Value of PosErr is considered to be in [mV], so a value of 50, for example, at PosErr will create an output of 50mV at the analog output.

Of course, both positive and negative values are supported.

When monitoring the value of a parameter by the analog output, the monitored parameter can have too small or too large value, so it may not optimally fit the range of the analog output (in [mV]). To solve this difficulty, an additional parameter was defined: AOutShifts[n]. This parameter is used when AoutMode[n] is not zero (meaning, in monitoring mode), so that the value that is written to the analog output port is:

Analog Output Port (n) = Monitored Parameter (n) * $2^{AOutShifts[n]}$

With the Monitored Parameter (n) selected by AoutMode[n].

Negative and positive values are supported for AOutShifts[].

Note that if the resulted value to send to the analog output is larger than the analog output range, the analog output will use only the relevant lower bits of the value (no saturation).

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Limits and Protections/Faults

The controller has a variety of limits and protections to prevent damage to the system, motor and the controller itself.

Some of the protections are done by hardware or hard coded in the controller so they cannot be changed or bypassed by the user. Other protections and limits can and should be set by the user to further protect the system that is used and any persons using it.

Monitoring faults

When a protection is activated the motor is immediately disabled (the voltage to the motor becomes 0). When this happens the error code is also registered in the variable ConFlt and logged in the error log.

The user can use ConFlt to find which error caused the motor to be disconnected. ErrLog can be used to see if the actual disabling of the motor was preceded by other events that may have caused it to be disabled, and to see also historical events.

For example, it is possible that the motor reached a limit that caused it to start decelerating using the emergency deceleration. This high deceleration may have caused the current to be too high and disable the motor due to overcurrent. This information can be used to make decisions about changes in the design (Is a lower emergency deceleration enough? Maybe the limit should be changed? Maybe a current limit was set lower than necessary?).

The fact that a fault happened can also be monitored by using a digital output that will be set as “alarm”. This output can be monitored to find out that a fault that disconnected the motor happened.

Recovery from a fault

Once a fault is detected (whether by hardware or software, see below), the relevant axis is disabled (MotorOn → 0) and the relevant fault code is stored at ConFlt and pushed to ErrLog (with time tag).

The user may resolve the source of the fault and then the axis can be re-enabled by MotorOn=1.

Assuming that the fault source disappeared, the axis will be enabled again by this command. If the fault still exists, it will be immediately disabled again (again, ConFlt will show the code, as well as ErrLog).

Once the axis is enabled by MotorOn=1, ConFlt is cleared to 0. However, the ErrLog still holds the last fault code, as well as all other historical errors/faults.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Response time of limits and protections/faults

The hardware detected faults (see below) are monitored continuously and the resulted hardware response for each fault will be triggered with tens of Nano-seconds (note that the fault detection mechanism itself may have some low pass filter, depending on the fault type).

The software detected faults (see below) and the limits are checked each control interrupt (typically ~61μs) and respond to the fault within the same (or the next, for less critical faults) control interrupt.

Hardware detected faults/protections

Akribis controllers/drives include some built in protections that are implemented by the hardware to ensure safety and/or fast response. When one of these protections happens the hardware immediately disables the power unit and clears all PWM command to zero.

In addition, the fault event is reported to the software, which will respond to it on the next control interrupt.

From the point of view of the user, the hardware detected faults and the software detected faults behaves the same. Both disable the axis (MotorOn -> 0), set ConFlt to the relevant protection code and push this code also to the ErrLog, with a time tag.

However, as explained above, the hardware detected protections do not depend on the software! They will respond immediately and will disable the axis even if the software is, from some unexpected reason, not responding to the fault as reported by the hardware.

These (hardware detected and responded) protections include:

- STO1

  Safety Torque Off input 1 is disconnected or not enough current is applied at this input.

- STO2

  Safety Torque Off input 2 is disconnected or not enough current is applied at this input.

  This input is not only responded as described above but also, independently, disconnect the power from the PWM signals drivers, as required by the safety regulations.

- Watch dog

  The software fails to periodically access the hardware in a timely manner.

- IPM Fault

  The power unit reported a general fault of the power bridge.

- Over Current

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

The hardware detected an extremely high current at one of the motor's phases (threshold is product dependent). This protection may be generally referred to short between one of the motor phases and GND.

- Disconnected Main Encoder:

  The main encoder is disconnected. The detection is based on identical logic level at the encoder inputs (A+ and A-, or B+ and B-). In some of the products, the index signal is also monitored.

  This protection is valid only for differential incremental encoders. Single ended encoder may result with false detection of this fault. See below for masking this protection.

- Disconnected Auxiliary encoder

  As above, but for the auxiliary encoder input.

Masking hardware detected faults/protections

Generally speaking, the hardware detected faults are non-mask-able.

However, two faults can be masked: the "Disconnected Main Encoder" and the "Disconnected Auxiliary Encoder". Why? Because these faults are reliable only with a differential input encoder.

In case another type of encoder is used, or in case a single ended encoder is used, each one of these faults can be masked using the ProtectMask parameter.

To mask the "Disconnected Main Encoder" protection:

ProtectMask = ProtectMask | 0x0004

To mask the "Disconnected Auxiliary Encoder" protection:

ProtectMask = ProtectMask | 0x0008

The easiest way to mask protections is to use the PC suite's Configuration -> Protections window and check the required mask.

Software detected faults/protections

As explained above, the software detected faults/protections are implemented by the software and are checked every control sample (typically ~61μs).

So, these protections can be considered (from the point of view of the safety regulations) as less safe. However, please note that it is assumed that the software is continuously working and if it is not (from some unexpected reason), the hardware will immediately trigger the watch dog protection…

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Most of the protections that are listed below can be programmed by the user so that they are compatible with the system used. Some of the protections can be completely disabled (by proper setup of their parameters), others can only receive different values for their activation.

The following sections describe the various software detected (and responded) protections.

Unknown Encoder Type

The type of encoder set in EncType or AuxEncType is not one of the allowed types, the motor cannot be enabled.

Motor Stuck

This protection detects if the motor is stuck and in such case disables the motor.

The user must define the parameters that indicate that the motor is stuck.

If the current is above StuckCurr and the velocity does not exceed StuckVel for a continuous duration of StuckTime the motor is considered to be stuck and it will be disabled.

Motor Over Current

If the overall motor current exceeds MaxMotorCurr for more than 4 control samples (typically ~61μs per sample) the motor is disabled.

This value is set by default to the highest value that is suitable for the hardware. The user can only lower the limit in order to protect a motor that cannot handle the full current that the drive can supply.

Phase Over Current

Apart from protecting against too much overall current to the motor, there is a protection for each phase current.

This is done for cases that a single phase is carrying much of the current. If a single phase current exceeds MaxPhaseCurr for more than 4 control samples (typically ~61μs per sample) the motor is disabled.

This value is set by default to the highest value that is suitable for the hardware. The user can only lower the limit in order to protect a motor that cannot handle the full current that the controller can supply.

Maximum Position Error

The position error is the difference between the position reference and the actual motor position that is read from the feedback.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

When the motor is on and using position control, the control loops always try to bring the position error to zero. However, this is often not possible. A very small position error (±1 encoder pulse) is very common by definition since the control only corrects the error when it is indicated by the feedback.

Larger values of error are usually caused during acceleration or deceleration, depending on the mechanical properties of the system, the currents and the setting of the control loops.

There are many other reasons that may cause the position error to be momentarily higher.

However, very high position errors may indicate a problem in the system. The “right” value for MaxPosErr depends on the system and can only be determined by the user.

When the position error (its absolute value) exceeds MaxPosErr the motor is immediately disabled.

Maximum Velocity Error

The velocity error is the difference between the velocity reference and the actual motor velocity that is read from the feedback (actually Vel[1] is used here, which is the feedback velocity after the velocity feedback filter).

When the motor is on and using position or velocity control, the control loops always try to bring the velocity error to zero. However, this is often not possible.

Larger values of error are usually caused during acceleration or deceleration, depending on the mechanical properties of the system, the currents and the setting of the control loops.

There are many other reasons that may cause the velocity error to be momentarily higher.

However, very high velocity errors may indicate a problem in the system. The “right” value for MaxVelErr depends on the system and can only be determined by the user.

When the velocity error (its absolute value) exceeds MaxVelErr the motor is immediately disabled.

Over and Under Bus Voltage Protections

The bus voltage must not exceed a certain limit, otherwise the voltage to the power unit and to the motor is too high and may cause damage.

On the other hand, it is also not allowed to be too low, to allow the hardware to function properly and the motor to be able to reach the desired maximal speed.

A too low bus voltage may also indicate some form of incompatibility between the power supply and the current requirements.

The high bus voltage limit has a double protection:

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

MaxVBusAbs is the absolute maximum allowed value for the bus voltage. If the bus voltage exceeds this value, even for a single sample, the motor is immediately disabled.

MaxVBus should be lower than MaxVBusAbs. This bus voltage value is allowed to be maintained for some time.

If MaxVBus is exceeded for more than the time determined by MaxVbusTime the motor is disabled.

Note that in fact, all the values between MaxVBus and MaxVBusAbs can be present during the time of MaxVBusTime.

MinVBus is the minimum allowed bus voltage. If the bus voltage drops below this value the motor is disabled immediately.

Power Stage Over Temperature

The parameter MaxPwrTemp can be used to decrease the temperature limit for the power stage. By default this value is set to the maximum allowed for this product. The user cannot set this parameter to a higher value.

If the power stage temperature (PwrTemp) is raised above MaxPwrTemp, the axis is immediately disabled.

Software detected limits

Limits, in contrast to faults, do not disable the axis. Instead, they result with a specific limit response.

The following sections describe the various software detected (and responded) limits.

Hardware Position Limits

The user can configure one of the digital inputs to be a reverse position limit (RLS) and another to be the forward position limit (FLS). These inputs are usually connected to hardware switches or other sensors that indicate that the motor passed the allowed travel limit of the motion.

When “on” state in the limit switch is detected the motor will stop if it is moving into the limit area.

The motor will stop using the emergency deceleration (EmrgDec). The motor is not disabled.

If, for example, the forward limit switch is on: If motion is started in the positive direction (further into the limit) the motion will not begin. Only motion in the negative direction, out of the limit, is allowed.

The same thing happens when the reverse position limit is exceeded, only in opposite directions.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

When a motion is stopped due to a hardware limit, this reason is recorded at the MotionReason status parameter.

Software Position Limits

Forward and reverse software position limits can be set by using FwdPLim and RevPLim respectively.

When the motor exceeds the value set in FwdPLim, and if it is moving to the positive direction, it will stop using the emergency deceleration (EmrgDec). The motor is not disabled.

If motion is started in the positive direction (further into the limit) the motion will not begin. Only motion in the negative direction, out of the limit, is allowed.

The same thing happens when the reverse position limit is exceeded, only in opposite directions.

Velocity command limit

The velocity reference (the command to the velocity control loop) is limited by the MaxVel parameter (limitation to ±MaxVel).

Besides limiting the velocity reference, this limitation has no other effect. However, it is expected that if the limitation is in effect, the position error will gradually grow and the Maximum Position Error fault can be triggered as a result.

Peak Current Limit

The current reference (the command to the current control) is limited by the PeakCL parameter (limitation to ±PeakCL).

Besides limiting the current command, this limitation has no other effect. However, it is expected that if the limitation is in effect, the position or the velocity errors will gradually grow and the Maximum Position Error or the Maximum Velocity Error faults can be triggered as a result.

Amplifier/Motor I^2t power limitation

Basically, the PeakCL is the peak current limit (in [mA]), as explained above.

The maximal allowed value for PeakCL is the maximal allowed current for this product.

Note that PeakCL, with sinusoidal commutation, refers to the peak value of the current sinusoidal signal, not to its RMS value.

However, the PeakCL parameter, together with the ContCL and PeakTime parameters, is used by the controller to implement the I^2t amplifier/motor power limitation scheme.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

The maximal values of PeakCL, ContCL and PeakTime define the I^2t limitation scheme as required not to exceed the amplifier capabilities. However, the user can modify these parameters to define a new limitation scheme that will match, for example, the motor power limitations.

For example, a given product can have maximal values as follows:

PeakCL = 16000
ContCL = 8000
PeakTime = 1000

Which means that this product can drive up to 16000mA (16A) for a period of 1000ms (1 sec), and that its continuous current limitation is 8000mA (8A). This is the limitation of the product itself.

However, for a given motor/application, the user can set:

PeakCL = 5000
ContCL = 3000
PeakTime = 500

So not to exceed the power capabilities of the motor, or even of the amplifier if the rated cooling conditions are not provided by the application.

How the amplifier/motor I^2t power limitation scheme works?

This limitation uses the following parameters: ContCL (continuous current limit), PeakCL (peak current limit) and PeakTime (maximal allowed peak current time).

The limitation code always calculates I^2 (using the overall motor current: MotorCurr) over the time ("integration" of I^2). If it becomes higher than $ContCL^2$, the limitation is activated.

Once the limitation is activated, the CurrRef, typically limited (saturated) by ±PeakCL, is now limited by ±ContCL, so that actually the amplifier is now limited to provide current that is no more than ContCL.

At all times, I^2 is calculated over the time, using a first order filter that emulates the heat dissipation at the motor (or better to say, the temperature of the motor). In this way, I^2 is "integrated" over the time to estimate the temperature that is created within the amplifier/motor. The first order filter's coefficient is calculated as a function of PeakTime, PeakCL and ContCL, so that:

Following a long time of no current to the motor, if a PeakCL current is now driven ($PeakCL^2$ is the input to the filter), for a period of PeakTime, the output of the filter will reach the $ContCL^2$ value, which will trigger the limitation and will limit the current to ContCL.

Of course, if a lower current (lower than PeakCL) is driven to the motor, a longer time will be available without triggering the limitation. Actually, if the current is equal or below ContCL, the limitation will be never activated.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Note that the availability of PeakCL current, for a period of PeakTime, is only after long time of no current to the motor (so the filter output is 0). In other cases (e.g.: a long time of 0.9*ContCL), the actual peak time will be (much) shorter, as the filter output will reach $contCL^2$ much faster (this is equivalent to the motor's temperature behavior, as in this case the motor was already heated due to the non-zero current over the time).

Once I^2 over the time goes below 0.9 x ContCL2, the limitation is released. This is needed to create a hysteresis, so that the limitation will not go on/off/on/off very fast around the continuous current.

Note that all the parameters of this limitation can be modified on-the-fly.

Note that the limitation works only on the CurrRef saturation that is typically PeakCL. It has no effect on other limitations that are supported by the controller: Current limitations using analog inputs of fixed values (see: CurrLimMode).

Note that while the user can set values like PeakCL=100 and ContCL = 200 (i.e. Peak < Continuous or Peak=Continuous), such case is invalid and the controller will internally set the Peak and the Continuous parameters' values to default values. So the user needs to take care, when assigning values to ContCL and PeakCL, not to violate these conditions.
This overwriting of PeakCL and ContCL values is reflected as a warning at the ErrLog, that the user can monitor at any time at the PC Suite, using the terminal window (use Ctrl + E).

For debugging and tuning purposes only, it is possible to monitor (record) the parameter DebugData[9]. It holds the output of the I2t filter (in $[mA^2]$). It is possible to see how the output of this filter is responding to the motor current over the time.

If ContCL is set to 2000mA, for example, the limitation will be activated when the filter output (that can be monitored at DebugData[9]) will reach a value of 2000*2000 = 4000000.

The limitation status is reflected at the StatReg parameter, bits 24, 25. In the PC Suite, you will see the "Other warn." LED going "orange" color when the limitation is activated.

Note:

This I^2t power limitation scheme is working only if the current control loop is activated (refer to the ControlMode parameter) or if an external amplifier is used and CurrRef is used to drive an analog output.

In the latter case, CurrRef is used in the equations of this limitation, instead of MotorCurr (see explanation above).

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Appendix I: Complex CAN Code

Each of Akribis's controllers' parameters is identified by a unique CAN code.

The CAN code for each keyword is provided at the Keywords and Communication Reference Manual.

The Complex CAN Code is a value that not only provides the CAN code (meaning: identifies a specific keyword), but also provide a reference to the axis and to the array index (for array keyword).

The complex CAN code is a 32 bit variable. The 32 bits are divided into the following bit fields:

Bits 0 – 9 (LSB) include the CAN code of the parameter in 10 bits.
Bits 10 – 12 are axis number in 3 bits. 0 is the first axis (axis A).
Bits 16 – 31 are the index into an array for arrays (use 0 otherwise).

Example:

To create a Complex CAN Code that points to the keyword Vel[2]:

The CAN code for Vel is 5 (refer to the Keywords and Communication Reference Manual at the Vel keyword page).

Assuming a single axis controller, so the axis number is 0.

The index into the array is 2.

The hexadecimal word is 0x00020005

In decimal value of the Complex CAN Code is 131077.

Complex CAN Codes are used when a value of a given parameter shall identifies another parameter. For example: a parameter to record (at RecParam[]) or a parameter to monitor at the analog output (at AoutMode[]), or to select a parameter to be used as the master for and ECAM motion (at ECAMMaster).

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

Appendix II: Download Firmware

From time to time it is possible that an updated version of the firmware will be released. The firmware update can be done using any communication channel that is used for the controller's normal operation (RS232 or CAN).

We recommend that you take the following considerations into account before downloading firmware:

- Most important: Is the update necessary for your system? If the update includes new features that you do not need it is best to stay with the known version
- To update firmware you will need to connect communication cables to every controller, even if such a cable is not normally present in the machine
- To update firmware you will need a PC with Akribis's PC suite installed

How to download new firmware

To download new firmware you will need:

1. PC with PC suite installed
2. The hex file of the new firmware version you want to download
3. A communication cable to connect the PC to the controller

Before download it is recommended to do the following:

1. Save all the parameters to flash if they are not saved. The parameter section of the flash is not erased in the download process.
2. If there is a user program in the controller it will be erased. Make sure you have a copy of the project.

To start downloading:

1. Connect the PC to the controller and configure the connection
2. End all motions and disable the motor
3. Go to the Manage -> Download Firmware window
4. Browse to the location of the new firmware hex file
5. Enter the password: 160412
6. Select the communication channel
7. Click "Download Firmware"

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945

8. Wait until the progress bar is full and a message indicating the end of the process appears.
9. If needed, download the user program again

What if the download fails?

The download process can fail for several reasons. For example, loss of power or a cable disconnected in the middle of the download. In this case, there is no complete code in the controller.

It is necessary to repeat the download procedure. If the file was not downloaded correctly, after reboot the boot program will be active. This is indicated by a LED flashing on the board (see hardware manual). If the board is installed so that the LED is not visible, try connecting to the board using a terminal. If CR> is replied with BOOT OK> then the boot loader is operating and the download firmware process can be repeated normally.

Note:

The download process requires the PC Suite.

Cell: +972-52-2564079 Tel: +972-3-9046511 Fax: +972-3-5478533 eyal@Akribis.co.il Drom Hasharon Council Center, POB 391, Neve Yarak, Israel 49945