Motion Control Topics

Optimizing the PID Controller

The **Tune Oscilloscope** can be used as a tool to optimize the controller settings, thereby optimizing your system performance. To do that you only need to know a few things about the control scheme of APOSS:

How the Control Process Works

The position controller has two parts:

1. The Set point Generator interprets the various positioning commands in APOSS and generates a series of set point positions that eventually ends with the desired position.
   Normally, all positioning processes have a trapezoidal shaped velocity curve. That means that after a phase of constant acceleration there is a phase with constant velocity and finally a phase with constant deceleration, which ends at the desired target position.
2. The PID controller receives the set point positions from the Set point generator and in turn calculates the speed reference needed for the motor to follow the current set point position.
   By setting the PID control parameters you can directly influence to what degree and how quickly a deviation from a theoretical set path (as specified by the set point series) should be counteracted.

The following behavior indicates that the control parameters are not optimally adjusted:

- Drive vibrates
- Drive is loud
- Frequent occurrence of position errors
- Poor control accuracy

!!! The control parameters are load-dependent. Thus the drive must be optimized under the actual conditions of use.

In exceptional cases it may be necessary to determine various sets of control parameters while working with heavily changing load conditions and to re-program them in subsequent application programs depending on the motion process.

Control Loop Design

The PID control unit of the positioning system transfers the necessary output frequency via an internal speed reference to the amplifier. This set value is periodically re-calculated with an interval of one millisecond (interval is programmable by the TIMER parameter). APOSS is by default set with soft 'fit for all' controller parameters.

The PID filter works according to the following formula:

1 = (FFVEL standardized) * (Setpoint velocity)

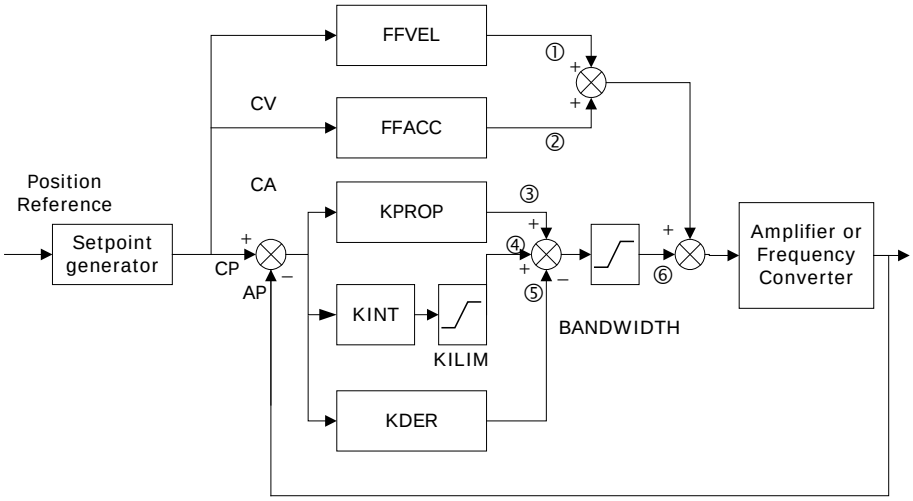
2 = (FFACC standardized) * (Setpoint acceleration)

3 = KPROP * (Position deviation CP – AP)

4 = KINT * (Sum of all previous position deviations) (limited by KILIM)

5 = KDER * (Difference of position deviation)

6 = 3 + 4 + 5 (limited by BANDWIDTH)



AV actual velocity, calculated as difference of the actual velocity minus the last velocity

AP actual position (calculated from encoder feedback) in qc.

CP current position set point in qc.

CV velocity set point in qc/ms; it is calculated as difference of the actual set point position minus the last set point position.

CA acceleration set point

Position deviation	=	is calculated by CP-AP.
Difference of the position deviation	=	position deviation – old position deviation
KILIM standardized	=	a KILIM value of 1000 corresponds with a command value of 100 %.
FFVEL standardized	=	a FFVEL value of 1000 and a command velocity = = VELMAX corresponds with a command value of 100 %
FFACC standardized	=	a FFACC value of 1000 and a command acceleration = = VELMAX/ms corresponds a command value of 100 %.

!!! In SYNCV mode the PID controller is working with speed deviation instead of position deviation. Speed deviation is calculated by CV-AV.

The controller utilizes two control strategies at the same time:

1. An open-loop feed-forward control. Since an asynchronous motor inherently has a good open loop performance, the feed-forward control is a very important part of the controller in most applications. Benefits from using feed-forward control include a very fast and accurate response to changes in the Set point reference.
2. A closed-loop PID control. The PID controller monitors the difference between the actual position and the Set point position. Based on this information, it calculates a control signal to minimize the position deviance. Thus the controller is able to compensate for changes in load or friction. The PID controller is also necessary to compensate for any position deviance caused by inaccurate setting of the open-loop feed-forward control.

In short: The feed-forward control is used to handle changes in the Set point reference (especially important in synchronization applications), while the PID control is used to handle changes in load conditions or inaccuracies of the feed-forward control.

Please see the section on how to optimize your controller settings.

PID Factors

Proportional Factor KPROP (11)

The ***Proportional Factor*** is multiplied with the position deviance and the result is added to the control signal. Since the calculated control signal is proportional to the position deviance (or error) this kind of control is called proportional control. The behavior of the proportional control is similar to that of a spring - the further the spring is extended the stronger the counter-force it produces.

Influence of the ***Proportional Factor***:

KPROP too small	large position deviation since the controller cannot adequately compensate for load and friction changes
KPROP larger	quicker reaction, smaller steady-state deviation, larger overshoot, less damping
KPROP too great	heavy vibrations, instability

Derivative Factor KDER (12)

The ***Derivative Factor*** is multiplied with the derivative of the position deviance (the 'velocity' of the position deviance) and the result is added to the control signal. The behavior of the derivative control is similar to that of an absorber - the faster the absorber is extended the stronger the counter-force it produces. Thus using the ***Derivative Factor*** increases damping in your system.

Influence of the ***Derivative Factor***:

KDER small	no effect
KDER larger	better dampening, less overshoot; if KPROP is increased simultaneously: faster reaction to control deviation at the same level of vibration.
KDER too large	heavy vibrations, instability

If you are working with a normal PID algorithm, then FFVEL (36) must always be just as high as the KDER share in order to achieve this typical KDER-damping.

However, if you are controlling large, slow-acting motors which could vibrate, then the control will be limited via the bandwidth. For this set the BANDWIDTH (35) very low, e.g. 10 % and work with FFVEL (36) and FFACC (37), which are both set relatively high.

Integral Factor KINT (13)

The sum of all error is calculated every time the control signal is updated. The ***Integral Factor*** is then multiplied with the sum of all positioning errors and added to the overall control signal. Thus in case that steady-state position errors occurs in your application, make sure you use the integral part of the controller. Steady-state errors will be leveled out as the summed error increases over time until the control signal eventually matches the load.

It is possible to set a limit for the control signal generated by the integral part of the controller (anti-windup).

Influence of the ***Integral Factor***:

KINT very small	steady-state position deviance is very slowly regulated to zero
KINT larger	faster regulation towards zero steady-state position deviance, larger overshoot
KINT too large	heavy vibrations, instability

Integration Limit KILIM (21)

The ***Integration limit*** sets a limit for the control signal generated by the integral part of the controller. This helps to prevent the so called “wind-up” problem which typically occurs in applications where the overall control signal (the internal speed-reference) is at a maximum for long periods of time.

Velocity Feed-forward: FFVEL (36)

The ***Velocity Feed-forward*** factor is a scaling factor that is multiplied with the derivative of the setpoint position (the velocity of the setpoint). The result of this operation is added to the

overall control signal. This feature is especially useful in applications where there is a good correlation between the control signal and the speed of the motor. This is indeed the case with most applications.

With a setting of 1000 (recommended in most applications) an output of 100 % is generated when the setpoint velocity is equal to its maximum setting (specified by the parameters VELMAX (1) and ENCODER (2).

The scaling of the FFVEL parameter is dependent on the correct setting of the maximum reference as well as VELMAX (1) and ENCODER (2).

Acceleration Feed-forward: FFACC (37)

The ***Acceleration Feed-forward*** factor is multiplied with the 2nd derivative of the setpoint position (the acceleration of the setpoint) and the result is added to the control signal. This feature should be used to compensate for the torque used to accelerate/decelerate the system inertia.

!!!

Scaling of the FFACC factor is dependent on the setting of the ***Shortest Ramp Time*** RAMPMIN (31). You should therefore increase the FFACC accordingly when decreasing ***Shortest ramp time*** RAMPMIN (31) and vice versa.

Sampling time for PID Control: TIMER (14)

For particularly slow systems you can slow down the entire control system by entering multiples of 1 ms as the sampling time. However, it is important to remember that such a change influences all the controller parameters!

Therefore, normally you should not deviate from the value of 1 ms.

BANDWIDTH (35)

A ***Bandwidth*** of 1000 means that the set value is being executed 100 %, thus Derivative, Proportional and Integral Factors are effective as defined.

But if you are operating a system which could be jeopardized by vibrations, for example, a crane with a heavy load then you can limit the bandwidth in which the PID controller should function. BANDWIDTH of 300 makes a limitation of 30 % possible. The build-up of a vibration is thus prevented since the control is only moved to with 30 % of the calculated set value.

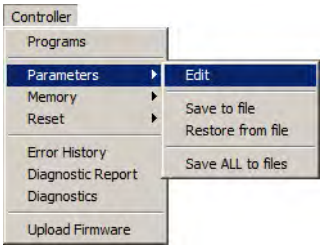
However, then it is necessary also to use the feed-forward part of the controller in order to achieve the corresponding control.

Optimizing your Controller Settings Step-by-step

For best results use the functions in the **Tune Oscilloscope** for this purpose; these help you to evaluate and optimize the PID-Controller on the basis of graphs of the set and actual curves.

However, we recommend only changing one value at a time and subsequently determining the improvement using tests from within the **Tune Oscilloscope**.

Click on **Controller** → **Parameters** → Edit and select the controller for which you are currently adjusting the settings.



Setting Controller Behavior

Before you adjust the controller parameters it is important to determine which controller behavior should be achieved.

!!! The drive elements must never be operated outside of the technical specifications. Thus the maximum acceleration is determined by the 'weakest' drive element.

- 'Stiff' axis: the fastest reaction possible is mainly influenced by the Proportional Factor. You can judge the results on the basis of the velocity graph.
- Damping of vibrations is mainly influenced by the Derivative Factor. The results can best be assessed in the velocity graph.
- Temporary (static) deviations in position are mainly reduced by the Integral factor and can best be judged on the basis of the position graph.

The quickest route to optimal control

The following simplified procedure usually leads to acceptable results in most cases:

1. Set the ***Integration Factor*** KINT (13) to zero.
2. Increase the ***Proportional Factor*** KPROP (11) in small increments until a slight vibration is visible in the velocity graph.
3. Now increase the ***Derivative Factor*** KDER (12) thus decreasing the vibrations.
4. Repeat steps 2 and 3 until there the machine no longer tends to vibrate and the velocity curve has been adjusted to the desired controller behavior.

Ten steps for optimum control for M1 and newer controllers

The following procedure will optimize your controller settings in most applications:

1. Make sure that you have specified correct values for VELMAX (1), ENCODER (2), and RAMPMIN (31). If you change these settings again at a later point you may need to optimize the controller again.
2. Set the parameter POSERR (15) to a very high value (e.g. 1000000) to avoid getting position errors (Error 8) during the following tests.

   !!! To avoid damaging the system, make sure that you set the POSERR within the limits of the system since position error monitoring is not active with extremely high values.
3. Optimize the ***Velocity Feed-forward*** control:
   → **Execute** a **Testrun** with KPROP=0, KDER=0, KINT=0, FFACC=0, and FFVEL =100.
   View the velocity profiles: If the Actual Velocity profile is lower than the Commanded Velocity profile, increase FFVEL. Of course if the Actual Velocity profile is higher than the specified Commanded Velocity you should decrease FFVEL.
   Run successive test runs until the two velocity profiles shown in the **Testrun** graph have the same maximum value.
   FFVEL is now optimized, save the current value.

4. In systems with large inertia and/or rapid changes in the reference velocity it is a good idea to use and optimize the ***Acceleration Feed-forward*** control (make sure the inertial load is connected when optimizing this parameter):
   Execute a **Testrun** with KPROP=0, KDER=0, KINT=0, FFACC=0, and FFVEL at the optimized value found above. Use the highest possible acceleration setting. If RAMPMIN (31) is adjusted properly an acceleration value of 100 and a deceleration value of 100 should be sufficient. Start out with a low setting of FFACC approx. 10.
   View the velocity profiles: If during acceleration the actual velocity is constantly lower than the reference velocity profile, then set a higher value of FFACC and → **Start** the **Testrun** again.
   Run successive test runs until the two velocity profiles shown in the **Testrun** graph have similar ramp-up and ramp-down curves.
   FFACC is now optimized, save the current value.
5. Next step is finding the maximum stable value of the ***Proportional Factor*** in the PID controller:
   Execute a **Testrun** with KPROP=0, KDER=0, KINT=0. Set FFVEL, and FFACC at the optimized values found above.
   View the velocity profile. If the velocity profile is not oscillating then increase KPROP and repeat the test run.
   Run successive test runs until the actual velocity profile is oscillating mildly.
   Decrease this "mildly" unstable KPROP value to about 70 %. Save this new value.
6. In order to dampen the oscillations created by the KPROP-part of the controller, the ***Derivative Factor*** should now be optimized.
   → **Start** a **Testrun** with KINT=0 and KDER=200. Set FFVEL, FFACC and KPROP at the optimized values found above.
   Run successive test runs with increasing values of the KDER factor. At first the oscillations will gradually reduce. Stop increasing KDER when the oscillations begin to increase.
   Save the last value of KDER.
7. In any system that requires a zero steady-state error, the ***Integral*** part of the controller must be used. Setting this parameter though is a trade-off between achieving zero steady-state error fast (which is good) and increasing overshoot and oscillations in the system (which is bad).
8. If you are using the ***Integral*** part of the PID controller, remember to reduce the KILIM as much as possible (without losing the ***Integral Factor*** effect of course) in order to reduce oscillations and overshoot as much as possible.
9. Reduce the BANDWIDTH as much as possible. With a properly optimized open-loop control BANDWIDTH could be reduced to as little as 6 or 12 % (60 – 120).
10. Set the POSERR (15) parameter back to normal e.g. 20000.

Once you have concluded the test run → **Save** the new parameters as the user parameters. Thus, these parameters are saved in the controller and in the future will be used for all programs.

What to do, if ...?	
… there is a tendency towards instability?	In the event of a strong tendency towards instability reduce the parameters ***Proportional*** KPROP and ***Derivative Factor*** KDER again, or reset the parameter ***Integral Factor*** KINT.
… stationary precision is required?	If stationary precision is required then the parameter ***Integral Factor*** KINT must be increased.
… the tolerated position error is exceeded?	If the test run is constantly interrupted with the message "position error" set the parameter for the ***Maximum Tolerated Position Error*** POSERR (15) – within the tolerable limits – as large as possible. If the position error occurs during the acceleration phase, then that suggests that the set acceleration can not be achieved under the existing load conditions. Increase the ***Max. Tolerated Position Error*** or determine a maximum acceleration suitable for the entire system. If position errors do not occur until after the acceleration phase and they can be delayed but not eliminated by increasing the ***Max. Tolerated Position Error***, this suggests that the maximum velocity (RPM) chosen is too high. Determine a maximum velocity suitable for the entire system.
… the maximum acceleration is not achieved?	In general, the technical data for a drive are only valid for a freely rotating axle end. If the drive is carrying a load the maximum acceleration is reduced. The theoretical maximum acceleration will also not be achieved if, for example, the PID controller output is too small or the amplifier / motor is not sized correctly and therefore does not provide enough energy for peak consumption during acceleration.

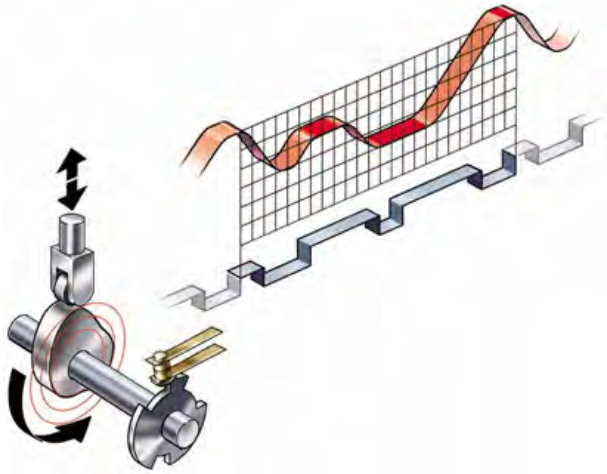
CAM Control

How a CAM control with APOSS operates

In order to realize CAM control, you need – depending on the application – at least one curve which describes the slave position in relation to the master position, as well as the engaging and disengaging behavior. Of course, many additional parameters are required for a CAM control which, together with the Fix points of the curve, produces a curve profile.

The synchronization in CAM-Mode (SYNCC command) can also be performed with marker correction (SYNCCMM and SYNCCMS). This would be required, for example, if the products are transported irregularly on a conveyor belt, or if adding errors need to be compensated for.

Diagram of the principle: The mechanical cam disc and the mechanical camshaft are shown on the left; the curves for the electronic CAM control and the electronic CAM box are shown on the right:



In order to create the curve profile, use the → **CAM-Editor**, into which you first load the controller parameters that have already been set. Then you set the Fix points of the curve and define the parameters required for your application. You can enter all values in physical or user-defined units under a Windows interface. You can constantly control the curve profile graphically; in this way, you can check velocity and acceleration of the slave axis.

Interpolation

The **CAM-Editor** calculates the curve from Fix points with the help of a spline interpolation. This is optimized to a minimum torque. In order to prevent speed leaps in the case of repeated curve cycles, the velocity at the beginning and the end is equated. You can choose between three types of curve for this calculation. In either type, the interpolation takes account of the gradient of the curve at the beginning and the end. Either the gradient at the beginning and at the end is averaged; or the gradient at the beginning of the curve is also used for the end of the curve, or the gradient of the curve at the beginning and the end is set to zero.

Tangent Points for Straight Sections

For areas where the velocity must be constant and the acceleration = 0, you need to use tangent points. A straight line will be drawn instead of a spline between these points.

Accuracy

The Fix points are used directly as interpolation points provided that this is permitted by the interval distance. The **CAM-Editor** performs a linear interpolation between the interpolation points. If a fixpoint is not hit by the selected interval distance, then the corresponding slave reference value does not exist in the interpolation table. You can avoid such deviations if you have activated → ☑ **Snap on Grid.**

Internal Realization as Array

The curve profiles are realized internally as arrays which you can call up by means of a DIM instruction and the SETCURVE command.

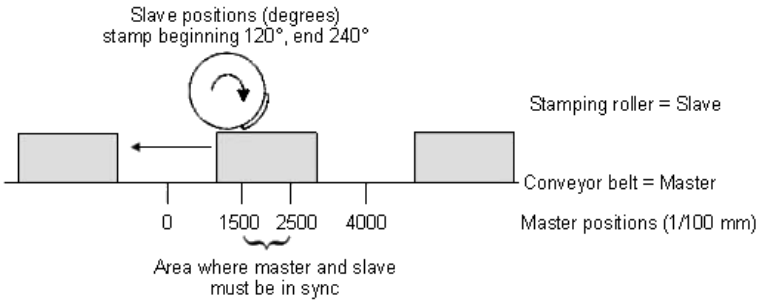
Quick Start Tutorial for Impatient Users

1. Start → CAM-Editor and load controller parameters using the → Read from controller toolbar button. Alternatively, opening an existing configuration zbc file using File → Open will automatically start the CAM-Editor and load the parameters.
2. Enter gearing factors or set user units MU and UU.
3. Use the → Add CAM array toolbar button to add a new CAM curve array.
   Note that Interpolation is not necessary except for controllers with very old firmware versions (older than 6.4.44). If interpolation is necessary, then define the → Number of Intervals for a master cycle length. The → Interval Time in the index card Curve Info should not be less than 20 ms.
4. Add at least three → Fix Points for master and slave.
5. Add → Start Stop Points for the engaging and disengaging.
6. Define in the index card Curve Info the → Slave Stop Position.
7. Enter in the index card Curve Info the number of → Cycles/min Master.
8. Check the velocity and acceleration of the slave with the help of the diagram.
9. Save the parameters in the controller by using the → Write to controller toolbar button. Alternatively, File → Save can be used to save the parameters in a configuration zbc file which can later be downloaded to the controller.

Example: Stamping of Boxes with Use-by date

The following example explains step by step how to edit the curve for this application of the CAM control and then how to incorporate it into your control program.

A roller is supposed to stamp an inscription with a length of 10 cm on cardboard boxes. The stamp corresponds to a roller section of 120 degrees. 60 cardboard boxes are transported on the conveyor belt per minute. The cardboard boxes are always transported on the conveyor belt at exactly the same distance from each other (e.g. by means of a mechanical set pattern). During the printing process, the stamping roller and the cardboard box must run in sync:



How to Edit the Curve Step by Step

1. Set the controller with the required parameters and save these user parameters with Parameters → Save to file with the extension "zbc".
2. Open this zbc file with File → Open. This will automatically start the CAM-Editor.
3. Determine the gearing factor of the master in MU units.

   The input should be possible in 1/10 mm resolution.

   The drive is connected with the conveyor belt by means of a gearing of 25:11; i.e. the motor makes 25, the drive pulley 11 revolutions.
   Gearing factor = 25/11

   Incremental encoder directly on the master drive; encoder resolution = 4096.

   The drive pulley has 20 teeth/revolution, 2 teeth correspond to 10 mm, thus 1 revolution corresponds to = 100 mm conveyor belt feed or 1000/10 mm.
   Thus, the scaling factor is 1000.

$$\frac{\text{Gearing Factor} * \text{Encoder Resolution} * 4}{\text{Scaling Factor}} \text{ qc} = 1 \text{ MU}$$

$$\frac{25/11 * 4096 * 4}{1000} \text{ qc} = \frac{25 * 4096 * 4}{1000 * 11} \text{ qc} = 1 \text{ MU}$$

$$\frac{2048}{55} \text{ qc} = 1 \text{ MU} = \frac{\text{Syncfactor Master SYNCFACTM (49)}}{\text{Syncfactor Slave SYNCFACTS (50)}}$$

Enter these values in the index card → **Synchronization** (the selected units should always be whole numbers):

Sync factor Master (49) = 2048
Sync factor Slave (50) = 55

4. Enter the gearing factor of the slave in UU units:

   Gearing factor = 5/1

   Encoder resolution (incremental encoder) = 500

   One revolution of the roller is 360 degrees. We are going to work with a resolution of 1/10 degrees. This means that we are dividing one revolution of the roller into 3600 work units:
   Scaling factor = 3600

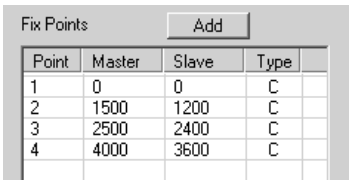
$$\frac{\text{Gearing Factor} * \text{Encoder Resolution} * 4}{\text{Scaling Factor}} \text{ qc} = 1 \text{ UU}$$

$$\frac{5/1 * 500 * 4}{3600} \text{ qc} = \frac{5 * 500 * 4}{3600} \text{ qc} = \frac{25}{9} \text{ qc} = 1 \text{ UU} = \frac{\text{POSFACT\_Z (23)}}{\text{POSFACT\_N (26)}}$$

   Enter these whole number values in the index card → **Encoder**:

   User factor numerator (23) = 25
   User factor denominator (26) = 9

5. Define → **Fix Points** for the conveyor belt (master) and the roller (slave). The function → **Snap on Grid** should be activated.

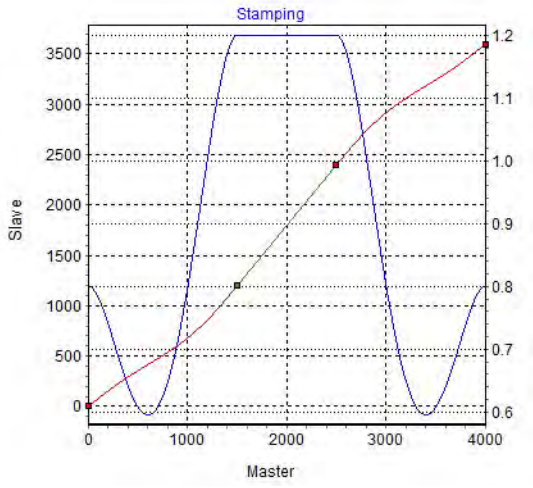


6. Master and slave must run synchronously with the same velocity between the position 1500 and 2500. This requires a straight line connecting these two points.

   Double-click in the column → **Type** for the fixpoint on the left at position 1500. Select → **Tangent** in the subsequent dialog.

   Alternatively, you can move the cursor to the fixpoint 1500, click on the right mouse button and select → **Tangent** in the subsequent pop-up menu. A straight line (colored green) will now connect this point and the following point.

7. Activate the diagram of the → ☑ **Velocity** to see the corresponding velocity curve:



8. Enter the → **Cycles/min Master** = 60 in the index card → **Curve Info**. This is the (maximum) number of cardboard boxes that are processed per minute.

9. Verify whether the acceleration of the slave is within the limit. For this purpose, you must activate the illustration of the → ☑ **Acceleration** and of the → ☑ **Acc. Limit.**

10. In order to load the curve into your control system, you must first save the file as a zbc file using **File → Save**.
You will see the name of the curve and the number of array elements in the title bar. You will need the latter for the DIM instruction during programming.

11. Load the zbc file with the modified parameters and the – automatically generated – curve arrays into the controller by means of **Parameters →** Restore from file**.**

Program Sample: Stamping of Boxes with Use-by date

Since the curve is stored internally as an array, the DIM instruction must appear first in your program:

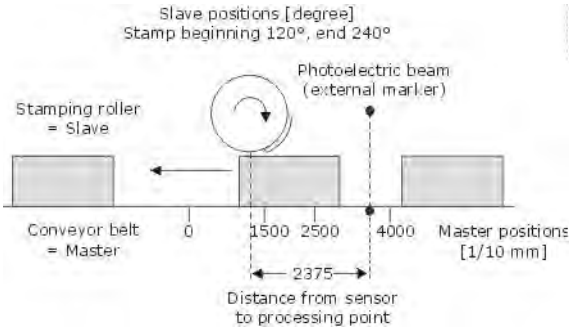
```
DIM stamping[92]        // Number of elements from the title bar of the CAM-Editor
HOME                    // Slave axis performs a home run (switch for zero position on top)
                        // afterwards, the slave will be in the zero position (0 degrees)
                        // (Omitted if an absolute encoder is used)
SETCURVE stamping       // Set stamping curve
                        // Assumption: A box is positioned at the processing point
                        // with its front edge and the master stands idle
DEFMCPOS 1000           // 1000 corresponds to this position (front edge of box)
POSA CURVEPOS           // Bring slave to the curve position that corresponds to the master position
SYNCC 0                 // Change into and remain in CAM-Mode
SYNCCSTART 0            // Engage roller immediately with the set maximum velocity
        // This does not cause any movement since the master is idle and in the correct position
        // Now the master can be started
start                   // Empty main loop so that program will not be terminated
                        // Additional processing could take place here
GOTO start
```

Example: Printing of Cardboard boxes with Marker Correction

If the boxes are not always transported at the exact same distance from each other, markers will be required that can recognize a box and correct the synchronization.

The following section describes how you can adapt the curve of the previous example to this application.

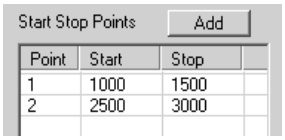
Again, a roller is supposed to stamp an inscription with a length of 10 cm on cardboard boxes. A maximum of 60 boxes are transported on the conveyor belt per minute. During the printing operation, both the stamping roller and the box must run in sync.



How to Edit a Curve for Synchronization with the Marker

1. Steps 1 to 8 are the same as in the previous example.

9. [Define the point pairs for the engaging and disengaging in list → **Start-Stop Points**. We make the assumption that engaging takes place at the beginning of the box and disengaging is to take place until the end of the box.



10. In the index card → **Curve Data,** determine the position in which the roller should stop if no other **Slave Stop Position** is defined in the program.

    The roller always needs to return to the position 0 degrees: → **Slave Stop Position** = 0

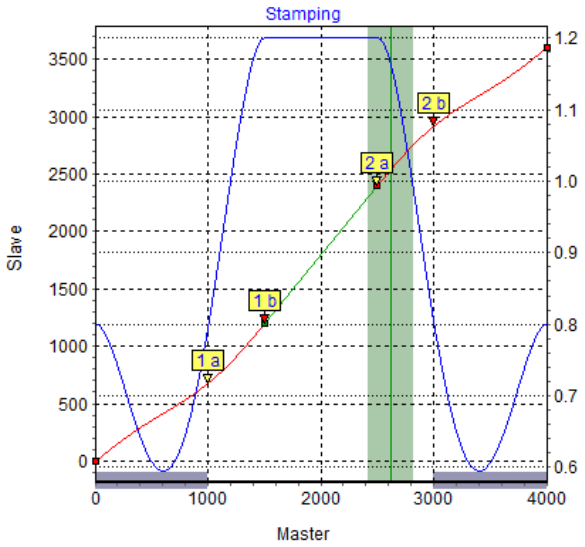
11. The photoelectric beam (external marker) has a distance of 237.5 mm from the processing point (= stamp touches the cardboard box) and detects the beginning of the box (corresponding to master position 1000). The marker distance is therefore 2375. Enter this value in the index card → Sync Marker and define the permitted tolerance for the appearance of the markers and the external marker type = 2 for the master

Master Marker Distance	SYNCMPULSM (58)	= 2375
Master Marker Tolerance	SYNCMWINM (68)	= 200
Master Marker Type	SYNCMTYPM (60)	= 2

    Enter the master position in the index card → **Curve Data**:

    Master Marker Position = 1000

12. Take a look at the curve profile in order to determine when the correction of the synchronization may begin at the earliest and when it must be finished. The vertical green line indicates the master position where the marker is recognized, the light green area shows the tolerance window for the appearance of the master marker.



At the earliest, the correction may begin when the printing of a box has been completed since any change of velocity during the printing process would damage the box. The correction must be finished completely when the next cardboard box reaches the processing point. In this example, the master positions at the end and beginning of a box are quite suitable:

Correction Start = 3000
Correction End = 1000

Enter the values in the index card → **Curve Data**; the depiction of the area is shaded in blue in the curve profile.

13. Verify whether the velocity and acceleration of the slave remain within the limit. For this purpose, you must activate the illustration of the → ☑ **Velocity** and of the → ☑ **Vel.-Limit** and then the illustration of the → ☑ **Acceleration** and of the → ☑ **Acc.-Limit.**
14. Use **File → Save as** to save the zbc file, for example "marker".
15. Load the zbc file with the modified parameters and the – automatically generated – curve arrays into the controller by means of **Parameters → Restore from file.**

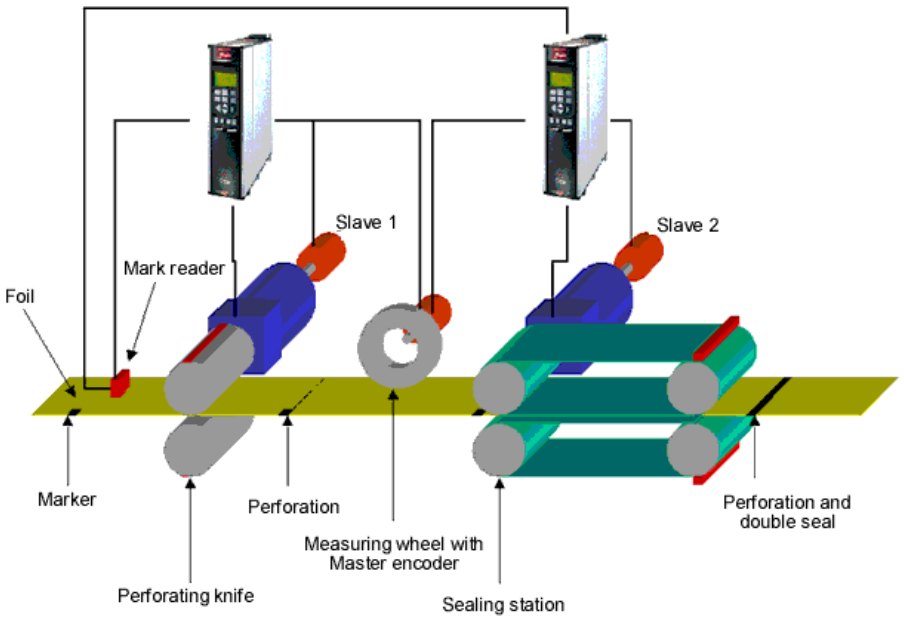
Program Sample: Printing of Cardboard boxes with Marker Correction

The DIM instruction must appear first in your program:

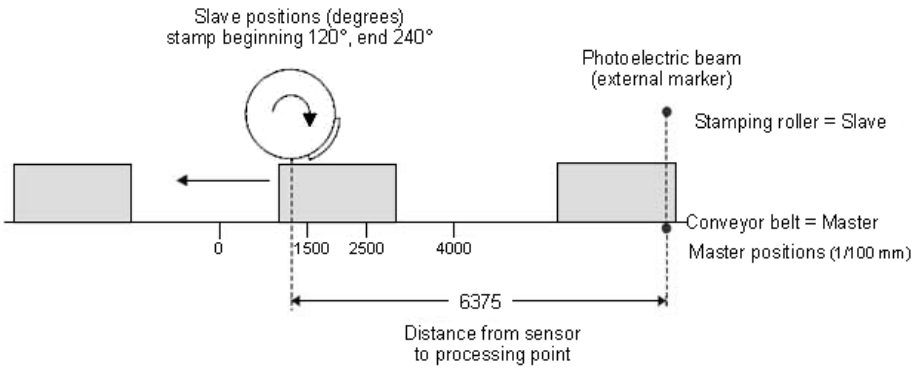
```
DIM marker[112]          // Number of elements from the title bar of the CAM-Editor
HOME                     // Slave axis performs a home run (switch for zero position on top)
                         // afterwards, the slave will be in the zero position (0 degrees)
                         // (Omitted if an absolute encoder is used)
SETCURVE marker          // Set stamp curve with marker
dist = GET SYNCMPULSM    // Distance to sensor
DEFMCPOS (1000-dist)     // This is the location that corresponds to the sensor signal
SET SYNCMSTART 2000
        // Counting of the master pulse does not begin until the next edge comes from the sensor.
SYNCCMM 0                // Synchronize in CAM-Mode until motor stop
SYNCCSTART 1             // Engage roller with start point pair 1
        // Synchronous operation
WAITI 4 ON               // Wait for input signal when conveyor belt is being switched off
SYNCCSTOP 2 0            // Disengage roller with stop point pair 1 and stop at position 0 degrees
```

If the Sensor Distance is larger than one Master Cycle Length

In many applications, the marker cannot be placed within one master cycle length, for example in the case of the following equipment for the production of plastic bags:



Since no markers can be installed between the slaves here, there is only a marker reader in this application; the welding station is much farther away than one master cycle length. Since the distance of the sensor is larger than one master cycle length, a buffer is provided for the marker deviation. When the marker appears, the value is entered in the buffer and read out upon the appearance of the next marker:

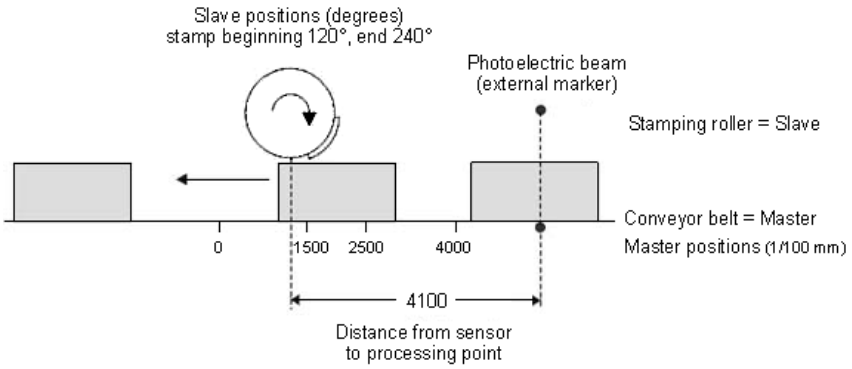


In order to assess in what area corrections may be made, you should subtract the master cycle length as often as necessary until the value is < 1 master cycle length. This is the maximum permitted distance for making corrections. In this example, it is 6375 – 4000 = 2375 and thus the same correction range as in the previous example.

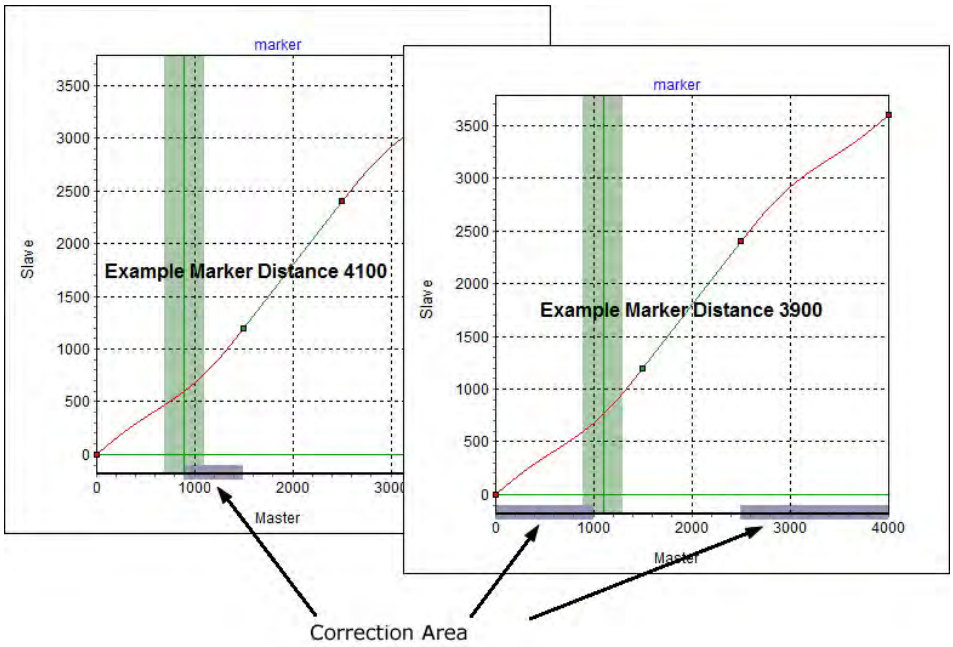
Problematic Situations in the Determination of the Marker Distance

If the marker is mounted so close to the processing point that there is no time left to correct the synchronization after the marker has been detected, then you can remedy the problem only through mechanical modification of the marker.

On the other hand, the same effect might also occur if the marker distance is larger than the master cycle length and if an insufficient distance likewise remains after the subtraction of this value, for example:



The value is entered into the buffer when the marker appears. The buffer is read out only when the next marker is recognized. This means that the marker is only "recognized" at the master position 900 and that there is little time left in our example to correct the error. It is the same effect as if the sensor had been mounted at the value (distance – master cycle length) or (4100 – 4000), respectively, i.e. only 10 mm in front of the processing point.



Thus, it would be better to install the sensor in such a way that the distance to the processing point is either smaller or substantially larger than one master cycle length; here, for example, at a distance of 3900. Then it is possible to correct from 2500 to 1000.

Alternatively, the sensor could be installed further away, for example at a distance of 7900; this has the same effect as if the sensor had been installed at distance – master cycle length (7900 – 4000), i.e. 3900 in front of the processing point. This allows enough time to correct the synchronization.

If this cannot be done mechanically, then the values must be manipulated somewhat in order to avoid the solution with the buffer. Please proceed as follows:

Subtract a value x from the actual distance so that the distance becomes < than the master cycle length, for example 4100 – 200 = 3900. You also subtract the value x from the master position, i.e. 1000 – 200 = 800.

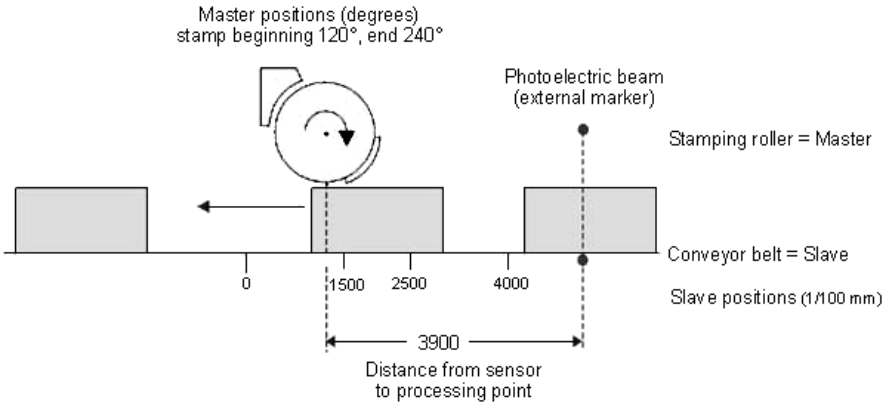
Enter both values in the index cards → Sync Marker and → Curve Data:

Master Marker Distance SYNCMPULSM (58) = 3900
Master Marker Position = 800

Since no buffer is generated now, it would be possible to correct from 2500 to 800, for example.

Example: Slave Synchronization with Marker

In the following example, the conveyor belt is the slave and the stamp roller the master, since the take-up and delivery of the dye must be continuous for a uniform printing process. A maximum of 20 cardboard boxes are transported on the conveyor belt per minute. The distance of the boxes is not larger than one master cycle length. Stamp roller and box must run in sync during the printing operation:



In contrast to the synchronization with master marker correction, here the slave position is being corrected instead of the curve.

How to Edit a Curve for the Slave Synchronization

1. Set the controller with the required parameters and save these user parameters with **Parameters → Save to file** with the extension "zbc".
2. Open this zbc file with **File → Open**. This will automatically start the CAM-Editor.
3. Determine the gearing factor of the master in MU units.

   Gearing factor = 5/1

   Encoder resolution (Incremental encoder) = 500

   One revolution of the roller is 360 degrees. We are going to work with a resolution of 1/10 degrees. This means that we are dividing one revolution of the roller into 3600 work units:
   Scaling factor = 3600

   $$\frac{\text{Gearing factor} * \text{Encoder Resolution} * 4}{\text{Scaling Factor}} \text{ qc} = 1 \text{ MU}$$

   Enter these whole number values in the index card → **Synchronization**:

Sync factor Master (49)	= 25
Sync factor Slave (50)	= 9

4. Enter the gearing factor of the slave in UU units:

   The input should be possible in 1/10 mm resolution.

   The drive is connected with the conveyor belt by means of a gearing of 25:11; i.e. the motor makes 25, the drive pulley 11 revolutions.
   Gearing factor = 25/11

   Incremental encoder directly on the master drive; encoder resolution = 4096

   The drive pulley has 20 teeth/revolution, 2 teeth correspond to 10 mm, thus 1 revolution corresponds to = 100 mm conveyor belt feed or 1000/10 mm.
   Thus, the scaling factor is 1000

   $$\frac{\text{Gearing factor} * \text{Encoder Resolution} * 4}{\text{Scaling Factor}} \text{ qc} = 1 \text{ UU}$$

   Enter these values in the index card **Encoder Data** using Parameters → Edit

User factor numerator (23) = 2048
User factor denominator (26) = 55

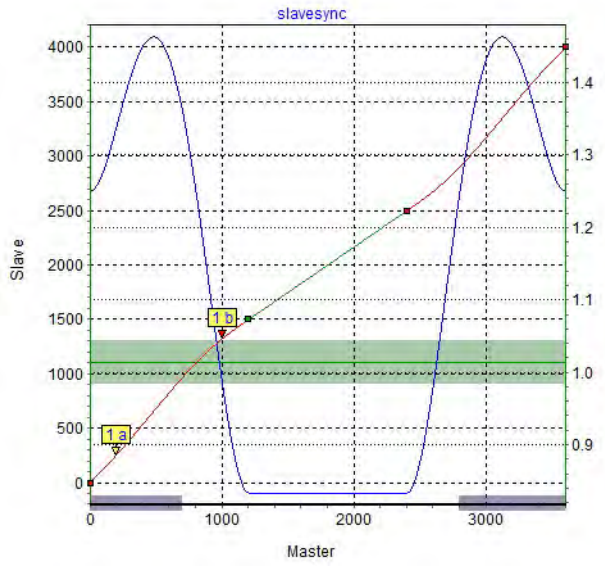
5. Define → **Fixpoints** for the roller (slave) and the conveyor belt (master). The function → **Snap on Grid** should be activated.

Point	Master	Slave	Type
1	0	0	C
2	1200	1500	C
3	2400	2500	C
4	3600	4000	C

6. Master and slave must run synchronously with the same velocity between the master positions 1200 to 2400. For this, you need to have a straight line that is determined with two tangent points. Double-click in the column → **Type** for the fixpoint on the left at position 1200.

Point	Master	Slave	Type
1	0	0	C
2	1200	1500	T
3	2400	2500	C
4	3600	4000	C

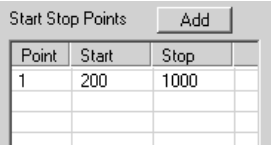
Activate the diagram of the → ☑ **Velocity** to see the corresponding velocity curve as it is shown in this figure:



8. Enter the → **Cycles/min Master** = 20 in the index card → **Curve Info.** This is the (maximum) number of cardboard boxes that can be processed per minute.

9. Verify whether the acceleration of the slave is within the limit. For this purpose, you must activate the illustration of the → **Acceleration** and of the → **Acc. Limit.**

10. Define in the list → **Start-Stop Points** with some safe distance in order to start the synchronization at the beginning. Engaging should take place between 20 and 100 degrees because it must be completed at 120 degrees.



11. In the index card → **Curve Data**, define the position where the conveyor belt should stop if no other **Slave Stop Position** is being defined in the program:

The conveyor belt should always stop in position 0: → **Slave Stop Position** = 0

12. The photoelectric beam (external marker) has a distance of 390 mm from the processing point (= stamp touches the box) and detects the beginning of the box (corresponds to slave position 1000). Thus, the marker distance is 3900. Enter this value in the index card → **Synchronization** and define the permitted tolerance for the appearance of the markers and the external marker type = 2 for the slave:

Slave Marker		
Distance	SYNCMPULSS (59)	= 3900
Tolerance	SYNCMWINS (69)	= 200
Type	SYNCMTYPS (61)	= 2

    Enter the value in the index card → **Curve Data**:
    Slave-Marker-Position = 1000

13. Take a look at the curve profile in order to determine when the correction of the synchronization may begin at the earliest and when it must be finished. The green horizontal line indicates the master position where the marker is recognized, the light green area shows the tolerance window for the appearance of the master marker. (See PC Software for color.)

    At the earliest, the correction may begin when the printing of a cardboard box has been completed, since any change of velocity during the printing process would damage the printing stamp and/or the box. Also, the correction must have been completed in its entirety when the next box arrives at the processing point. In this example, the slave positions at the end and beginning of a box are quite suitable. Enter the values in the index card → **Curve Data**:

Correction Start	= 2800
Correction End	= 750

14. Verify whether the velocity and acceleration of the slave remain within the limit. For this purpose, you must activate the illustration of the → **Velocity** and of the → **Vel.-Limit** and then the illustration of the → **Acceleration** and of the → **Acc.-Limit.**

15. **File** → **Save** the zbc file.

16. Load the zbc file with the modified parameters and the – automatically generated – curve arrays into the controller by means of **Parameters** → **Restore from File.**

Program Sample: Slave Synchronization with Marker

In order to determine the master position, a switch on the master is required that indicates the zero position. In order to put the slave into the correct position, it will be moved forward to the photoelectric beam. This corresponds to the beginning of the box = 1000. Then the slave will be moved further by 2900 (= marker distance 3900–1000); thus, the slave is exactly in front of the processing point with the beginning of the cardboard box 1000 i.e. at slave position 0.

```
DIM slavesync[108]          // Number of elements from the title bar of the CAM-Editor
HOME                        // Slave axis performs a home run (switch for zero position on top)
        // Afterwards, the slave will be in the zero position (0 degrees)
        // (Omitted if an absolute encoder is used)
DEFMCPOS 0                  // Curve starts at master position 0
SET SYNCMSTART 2000         // Counting of the master pulse does not begin
                            // until the next edge comes from the sensor
SETCURVE slavesync          // Set curve for the slave synchronization
        // Go to start
CSTART
CVEL 10                     // Go forward slowly until photoelectric beam appears
oldi = IPOS                 // oldi = last marker position of the slave
WHILE (oldi == IPOS) DO     // Wait until box is detected
ENDWHILE
POSA (IPOS + 2900)          // Move box forward by 2900
SYNCCMS 0                   // Synchronize in CAM-Mode
SYNCCSTART 1                // Engage with start-stop point pair
```

CAM Box

How a CAM Box with APOSS operates:

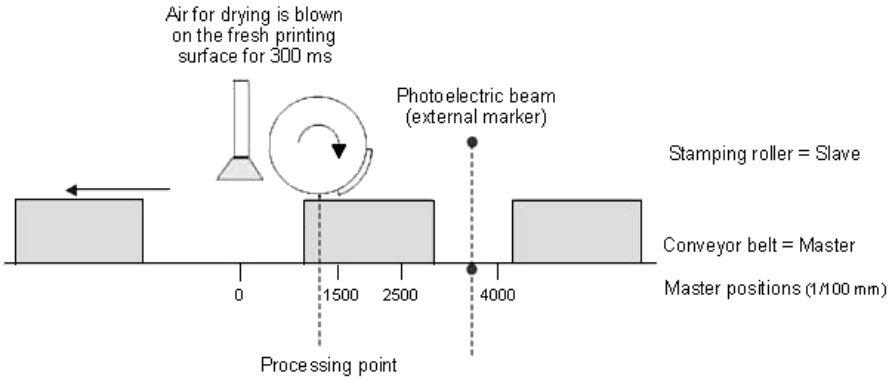
The mechanical camshaft is also reproduced by one (or more) curves. In order to realize CAM box, it must be possible to engage and disengage the slave at specific master positions over and over again.

This is possible with APOSS with the position interrupt commands ON MAPOS .. GOSUB and ON APOS .. GOSUB. It is possible to call up a subprogram whenever a defined master position has been passed (both in the positive or negative direction).

It is possible to realize many applications that are typical for the packaging industry in connection with a curve profile in which several curves have been defined for engaging and disengaging.

Example of a CAM box

After a cardboard box has been printed, the fresh print is to be dried immediately in the air stream:




```
ON MCPOS 2500 GOSUB drier
        / Call up a subprogram when the master position 2500 is passed in positive direction
SUBMAINPROG
    SUBPROG drier
    OUT 1 1             // Turn on drier
    DELAY 300           // Dry for 300 ms
    OUT 1 0             // Turn off drier
RETURN
ENDPROG
```

Limited Jerk

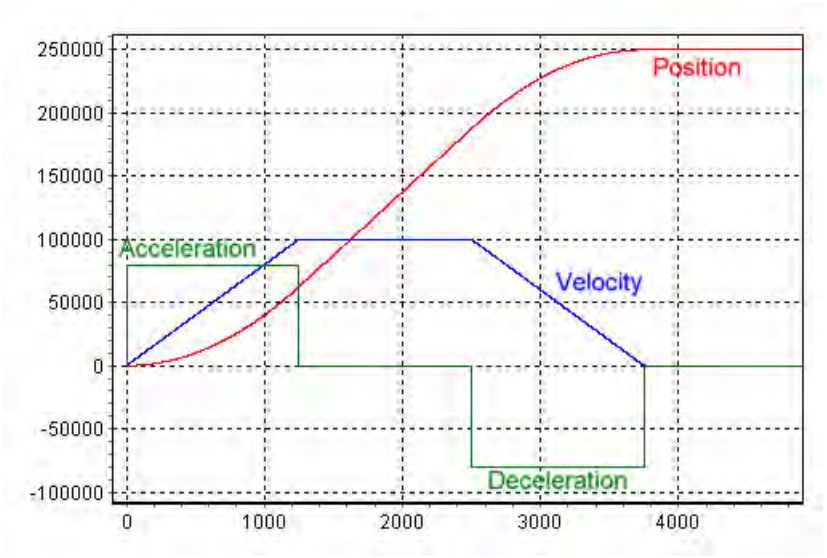
Understanding Limited-Jerk Movements

Limited-jerk movements are similar to normal Trapezoidal movements except that the user may control the "gentleness" of the acceleration and deceleration ramps. This allows the user to limit the "jerk" caused by the "instantaneous" acceleration of a Trapezoidal movement.

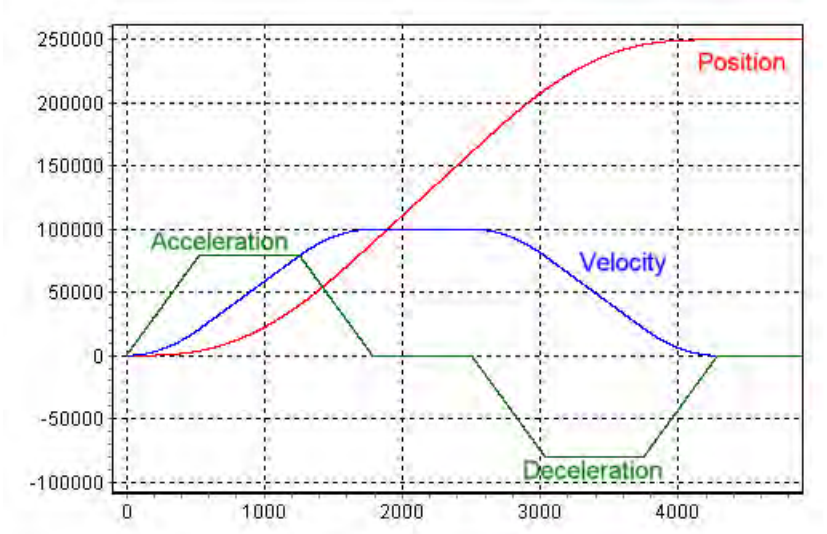
Typical applications where limited jerk is required:

- lift
- movement of heavy loads

As an example, the following chart shows the acceleration, velocity, and position curves for a Trapezoidal movement from one position to another position. The sharp changes in acceleration cause the motor to experience a "jerk" at the beginning and end of each velocity ramp.



The following chart shows the same movement done using a Limited-jerk movement. Note that the acceleration changes are no longer "instantaneous" and that the "corners" of the velocity curve have become rounded. This will result in a smoother motor movement. It will also require slightly longer to get to the desired target position because the motor takes longer to accelerate to maximum acceleration.



The user can control the "gentleness" of the acceleration ramp using 4 parameters:

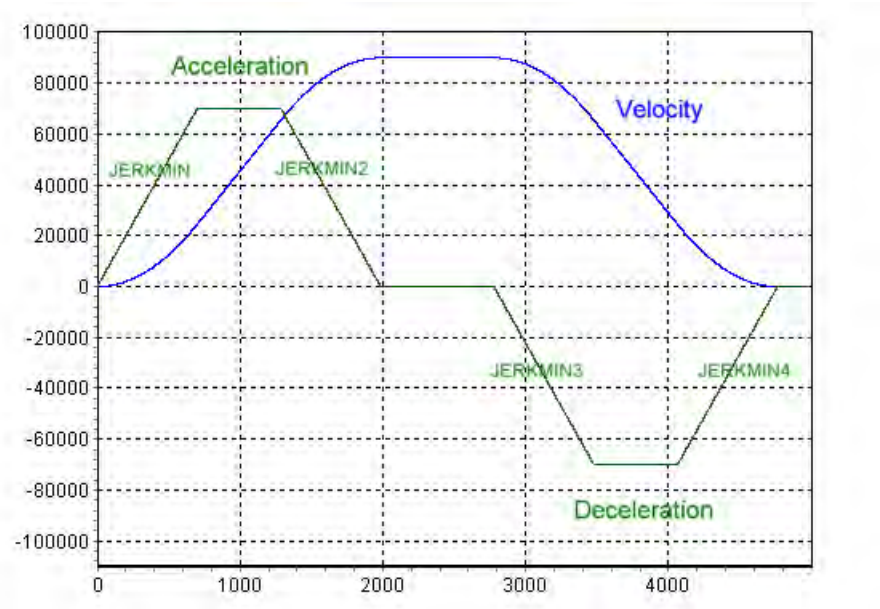
JERKMIN — Acceleration ramp-up constant. This specifies the number of milliseconds required to ramp the acceleration up from 0 to maximum acceleration.

JERKMIN2 — Acceleration ramp-down constant. This specifies the number of milliseconds required to ramp the acceleration *down* from maximum acceleration to 0 (i.e. normally to constant maximum velocity). If set to 0, this will default to the same value as JERKMIN.

JERKMIN3 — Deceleration ramp-up constant. This specifies the number of milliseconds required to ramp the deceleration *up* from 0 to maximum deceleration. If set to 0, this will default to the same value as JERKMIN.

JERKMIN4 — Deceleration ramp-down constant. This specifies the number of milliseconds required to ramp the deceleration down from maximum deceleration to 0 (i.e. normally to 0 velocity). If set to 0, this will default to the same value as JERMIN.

These constants correspond to the "slopes" of the different portions of the acceleration curve. This is shown in the chart below. When set to larger and larger numbers, the acceleration and/or deceleration will become gentler and gentler as the ramps become longer and longer. Note that the acceleration slope determined by the JERKMIN acceleration ramp-up constant will be used whenever the acceleration is being ramped-up. It is not used only to ramp-up from 0 to maximum acceleration. Similarly, JERKMIN2 is used whenever the acceleration is being ramped-down, JERKMIN3 is used whenever the deceleration is being ramped-up, and JERKMIN4 is used whenever the deceleration is being ramped-down.



Limited-jerk movements will not normally exceed the velocity and acceleration limits set for the controller (e.g. limits set by the VEL, ACC, DEC, etc., commands). In the above chart, these limits can be recognized by the "plateaus" in the acceleration curve. If the current velocity and/or acceleration are outside these limits when the Limited-jerk movement is started, then the movement is accelerated or decelerated as necessary to bring the movement within the set limits.

It is important to understand that for Limited-jerk movements, "acceleration" is defined as "speeding up" in either direction (i.e. either forwards or backwards) and "deceleration" is defined as "slowing down" in either direction. A result of this is that maximum velocity, maximum acceleration, maximum deceleration, and the four JERKMIN values are all independent of the direction of movement. This can have important consequences when a Limited-jerk movement must "reverse" the direction of the motor, particularly when maximum deceleration differs from maximum acceleration. In this case, the Limited-jerk movement will ensure that the *deceleration* ramp flows smoothly into an *acceleration* ramp at exactly 0 velocity when the direction changes and without exceeding either the deceleration or acceleration limits.

A Limited-jerk movement can be used in three different situations:

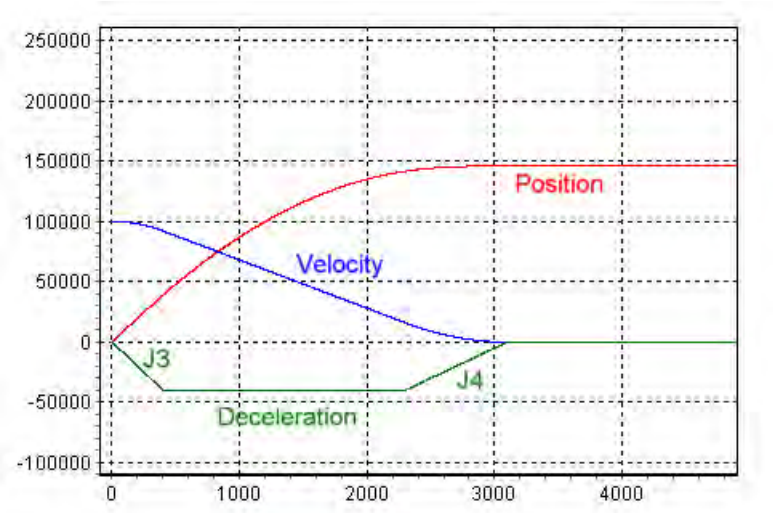
1. Stopping from the current velocity and acceleration (where the final position is not important).
2. Changing from the current velocity and acceleration to a specified constant velocity (where the positions are not important).
3. Moving from the current position (and the current velocity and acceleration) and stopping at a specified position.

Examples of Limited-Jerk Movements

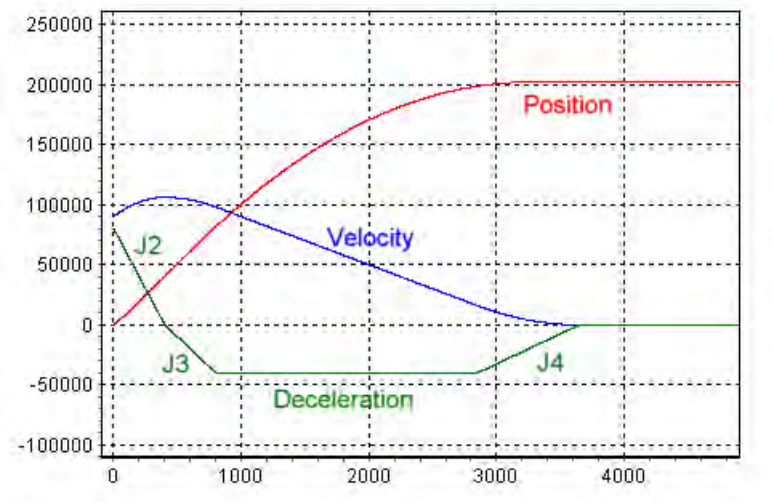
In the following examples, maximum acceleration has been set to a value higher than maximum deceleration so the motor can speed up faster than it can slow down. As well, JERKMIN has been set smaller than JERKMIN2, JERKMIN2 smaller than JERKMIN3, and JERKMIN3 smaller than JERKMIN4. This was done so that the various segments of the curves can be visually distinguished in the charts. The four JERKMIN values have been labeled simply J1, J2, J3, and J4.

Stopping

The chart below shows a stopping movement that begins from a positive constant velocity. The curve consists of a deceleration ramp-up segment (using JERKMIN3), followed by a constant deceleration segment (at maximum deceleration), and finally a deceleration ramp-down segment to 0 velocity (using JERKMIN4).



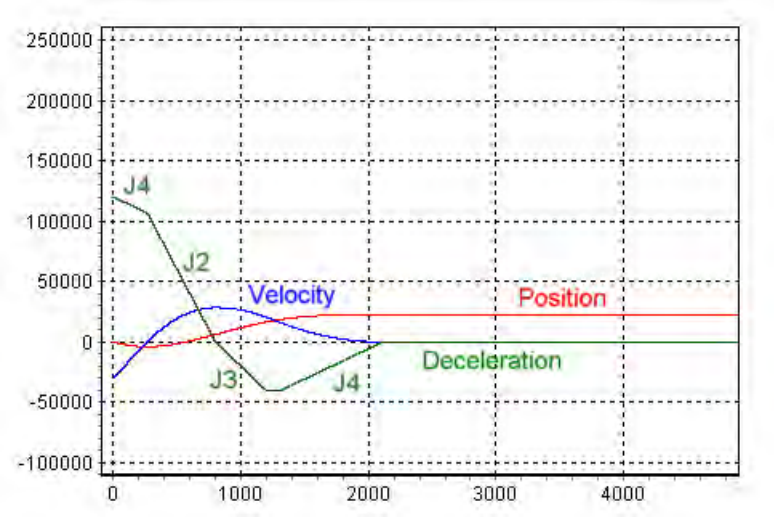
The chart below shows a stopping movement that begins from a positive velocity and a positive acceleration. Since the initial acceleration is positive, the curve must start with an acceleration ramp-down segment to 0 acceleration (using JERKMIN2). This will then be followed by a deceleration ramp-up segment (using JERKMIN3), a constant deceleration segment, and a deceleration ramp-down segment to 0 velocity (using JERKMIN4).



The chart below shows a stopping movement that begins from a *negative* velocity and a very high deceleration. (It is a deceleration because the speed is slowing down.) However, because the initial deceleration is so high, the motor is unable to stop without overshooting 0 velocity and 'coming back'.

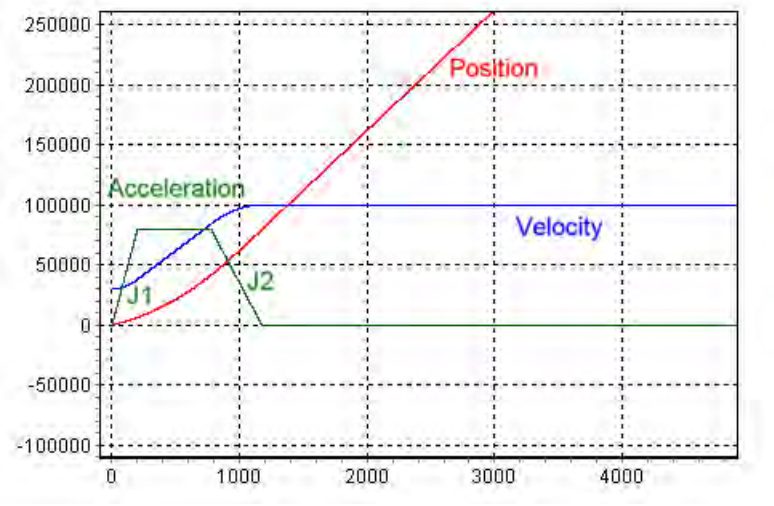
So the curve starts with a deceleration ramp-down segment (using JERKMIN4) to slow the deceleration as much as possible before reaching 0 velocity. At 0 velocity, the 'deceleration'

becomes an 'acceleration' because the direction has changed. Hence, the curve continues by ramping-down the acceleration (using JERKMIN2) until it gets to 0 acceleration. The motor is now at a constant positive velocity and so the curve will finish in the normal way with a deceleration ramp-up segment (using JERKMIN3), a constant deceleration segment (very short in this example), and a deceleration ramp-down segment to 0 velocity (using JERKMIN4).

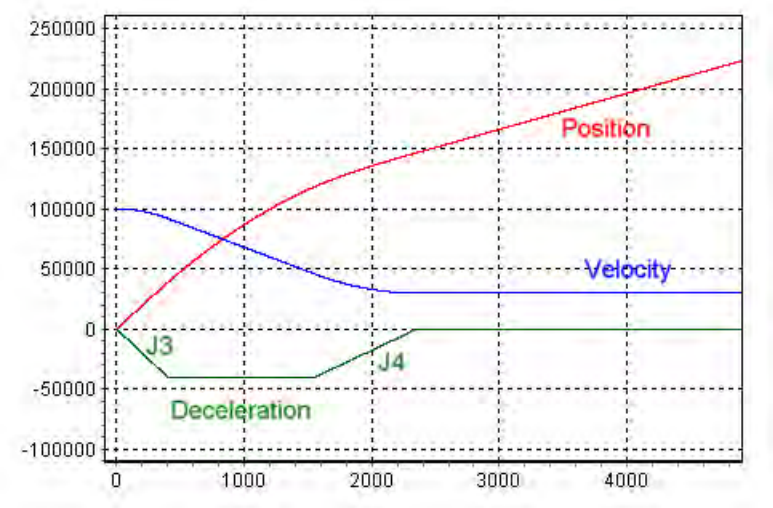


Changing to a Constant Velocity

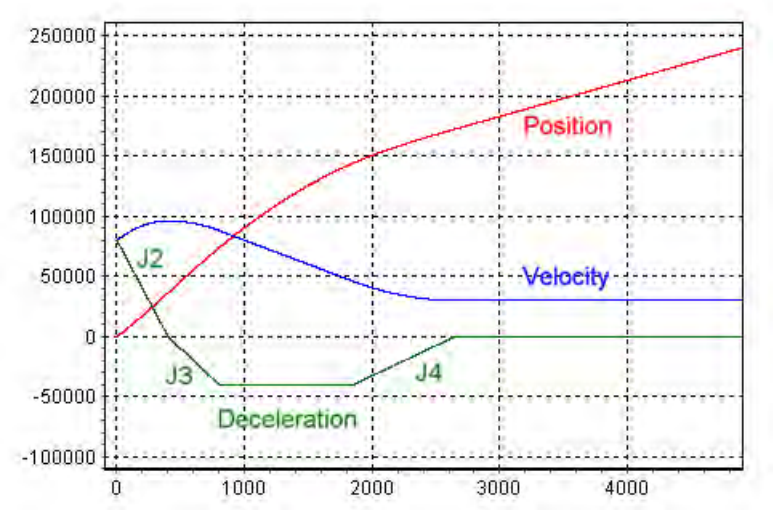
The chart below shows a movement that begins from a positive constant velocity and increases its speed to a higher positive constant velocity. This curve consists of an acceleration ramp-up segment (using JERKMIN), followed by a constant acceleration segment (at maximum acceleration), and finally an acceleration ramp-down segment to constant velocity (using JERKMIN2). Note that the deceleration JERKMIN3 and JERKMIN4 values are not used because there is never any deceleration.



The chart below shows a movement that begins from a high positive constant velocity and decreases its speed to a lower positive constant velocity. This curve consists of a deceleration ramp-up segment (using JERKMIN3), followed by a constant deceleration segment (at maximum deceleration), and finally a deceleration ramp-down segment to constant velocity (using JERKMIN4). Note that the acceleration JERKMIN and JERKMIN2 values are not used because there is never any acceleration.

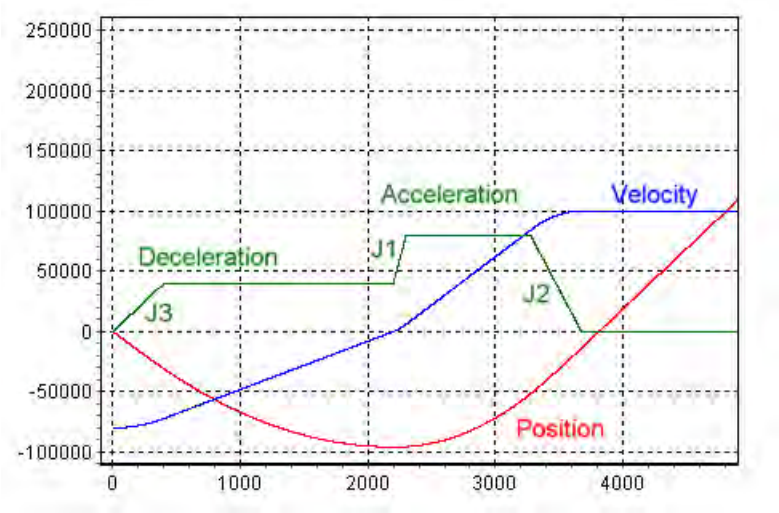


The chart below is similar to the previous example except that it begins with a positive acceleration. In this case, the curve must start with an acceleration ramp-down segment (using JERKMIN2). Once the acceleration reaches 0, then the curve can continue as before.



The chart below shows a movement that begins with a *negative* constant velocity and changes direction to a positive constant velocity. The curve must start by slowing down the speed so that it can 'turn around'. Hence the curve begins with a deceleration ramp-up segment (using JERKMIN3) until it reaches maximum deceleration.

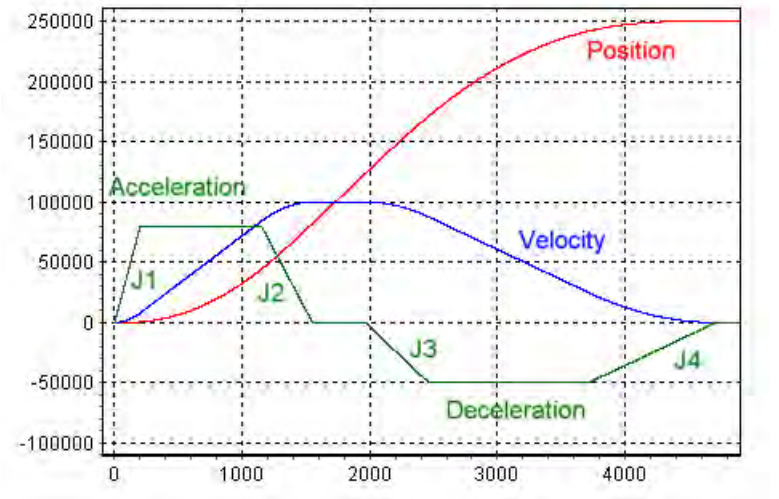
The deceleration continues at maximum deceleration until it reaches 0 velocity. Note that there is no deceleration ramp-down segment because the movement is not stopping. At exactly 0 velocity, the direction reverses and the movement is now accelerating in the other direction. But since maximum acceleration is higher than maximum deceleration for this example, the curve is now able to include an acceleration ramp-up segment (using JERKMIN). The curve finishes in the normal way with a constant acceleration segment and an acceleration ramp-down segment to constant velocity (using JERKMIN2).



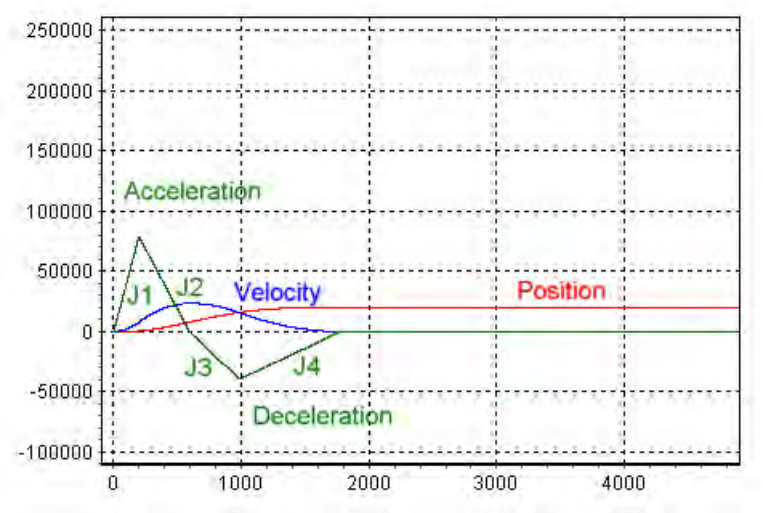
Moving to a Specified Position

The chart below shows a 'normal' movement from being stopped at one position and then moving forward to stop at another position. The curve starts with an acceleration ramp to maximum velocity. This portion of the curve will be similar to the first of the examples shown in "*Changing to a Constant Velocity*". The curve is simply changing to a constant velocity where the constant velocity is the maximum velocity.

Hence, the curve will consist of an acceleration ramp-up segment (using JERKMIN), a constant acceleration segment at maximum acceleration, and then an acceleration ramp-down segment to maximum velocity (using JERKMIN2). The movement then proceeds at maximum velocity until it needs to start the deceleration ramp that will stop the movement at the desired position. The deceleration ramp is identical to the first of the examples shown in "Stopping". The curve will consist of a deceleration ramp-up segment (JERKMIN3), followed by a constant deceleration segment (at maximum deceleration), and finally a deceleration ramp-down segment to 0 velocity (using JERKMIN4) stopping at the desired position.



The chart below shows a typical 'short' movement where maximum velocity cannot be reached. In this case, the curve ramps-up the acceleration (using JERKMIN) for as long as possible. This may or may not reach maximum acceleration, depending on how far away the target position is. There will then be an acceleration ramp-down (using JERKMIN2), followed immediately by a deceleration ramp-up (using JERKMIN3). Again, depending on the target position, there may or may not be a constant deceleration segment. The curve ends with a deceleration ramp-down to 0 velocity at the target position.



The chart below shows an example where the motor is initially moving in the 'wrong' direction and must turn around and 'go back' to the target position. Since it must "turn around", the curve starts with a deceleration ramp-up segment (using JERKMIN3) to maximum deceleration.

This will slow the speed down until it turns around. Deceleration continues at maximum deceleration until 0 velocity is reached and the direction reverses. At exactly this point, the motor is now 'speeding up' but in the other direction. From this point, the curve is similar to a normal movement to a target position, except that the whole curve is inverted because the direction has changed. The curve will have an acceleration ramp-up segment (going backwards), it may or may not have a constant acceleration segment, it will have an acceleration ramp-down segment, it may or may not have a constant velocity segment, it will have a deceleration ramp-up segment, it may or may not have a constant deceleration segment, and finally it will have a deceleration ramp-down to the target position.