Motion Control Products

Application note
Modifying target position

AN00105

Rev E (EN)

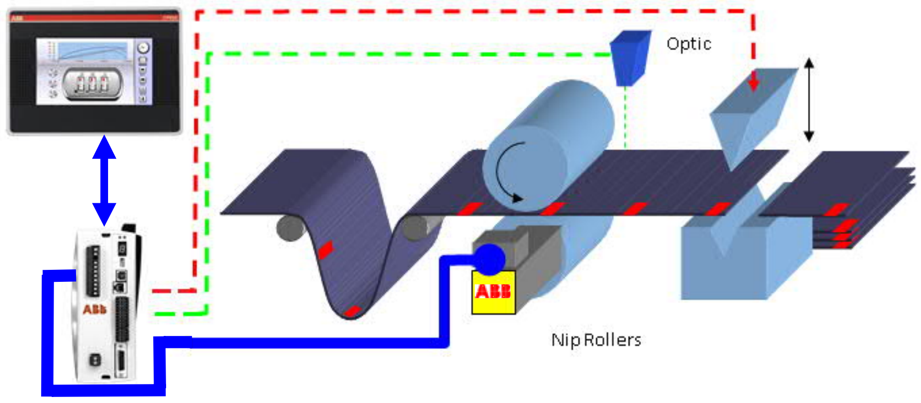
In some applications it is useful to change the target position of an axis during a move, without stopping motion until the new demand position has been reached.

Introduction

There are many applications that may require the ability to change the current move target during motion without affecting the current velocity of the axis. Examples of these include cutting pre-printed material to length, detecting the current position of a label or product and making a correction to this 'on the fly', press feeder systems where it is crucial to align a component with the press tool and indexing conveyor systems where it is necessary to set a new target position for every flight to avoid a drift in position over time due to fractional scale factors.

The Mint language provides a simple solution to this requirement with the INCR (increment relative position) and INCA (increment absolute position) commands. These commands work in a similar way to MOVEA and MOVER but allow the end point of the move to be changed at any time, even whilst the move is in progress.

Application example - Cut to registration



This application note comes complete with an example program to control a setup similar to the one pictured above. The program will work with a MicroFlex e190 or MotiFlex e180 AC servo drive with Mint memory module (+N8020 option).

Let us firstly take a brief look at the mechanical implementation of the example machine. In this example, nip rollers are driven by a BSM servo motor with SmartAbs feedback (131072 counts per revolution). The servo motor is controlled by an ABB MicroFlex e190 (or MotiFlex e180) servo drive. The nip rollers draw printed material from an unwind system, into the jaws of a cutter. A fast acting photoelectric sensor is wired to one of the two fast inputs available on the drive (input 1 in our example) and detects the preprinted registration marks on the material to be cut. An ABB CP600 HMI panel could easily be integrated in to the system to allow the operator to enter a nominal ‘default length’ of material to be pulled by the nip rollers.

In the main program loop the cycle starts by setting the value of POS of the nip roller axis to zero and setting a program flag “bRegMarkSeen” to false.

```
POS(_axNipRoller) = 0
bRegMarkSeen = _false
```

We then issue a move of the programmed default index distance (if we were to use a HMI, this distance could be easily modified by the user) and trigger it with a GO command. A program flag (bMoveInProgress) is then set to form an interlock with the latch event to prevent false triggering of motion. The program then waits for the move to be completed before turning this flag back off.

```
INCA(_axNipRoller) = fDefaultDistance
GO(_axNipRoller)
bMoveInProgress = _true
Pause IDLE(_axNipRoller)
bMoveInProgress = _false
```

Once the move is complete an output is turned on which controls the cutter solenoid. A dwell is set in the form of a wait command as a simple method of ensuring that the cut cycle has time to complete before the material starts moving again.

```
OUTX(_nCutter) = _on
Wait(nCutterDwell)
OUTX(_nCutter)  = _off
Wait(nCutterDwell)
```

In a real application the cutting mechanism would more than likely have sensors to check cutter actuation has taken place and the cutter is clear. However for the purposes of this application note it is not necessary. Likewise if no registration marks are seen by the registration sensor the sample program will carry on cutting material at the default length continuously. Some form of sensor failure detection could be easily implemented here if required, such as the method described in application note AN00230 that details a typical indexing conveyor application.

Ideally the registration sensor will pick up every registration mark. The signal from this sensor will then be used to trigger a latch event within the Mint program. This latch event is configured in the startup block of the program to capture the axis position upon the rising edge of the input signal (a template latch configuration is contained within the Mint Library in Mint Workbench). The first line of the event is an interlock to ensure latch events have no effect unless an index is in progress and to ensure only the first latch received during an index is processed.

```
If (bMoveInProgress = _true) And (bRegMarkSeen = _false) Then
```

Within the event the new target position is calculated. This is done by summing the LATCHVALUE with an offset value. The new move target is then issued using the INCA command and triggered with the GO command.

```
fNewDistance = LATCHVALUE(_axNipRoller) + fSensorOffset
INCA(_axNipRoller) = fNewDistance
GO(_axNipRoller)
```

The offset value is the distance of the registration sensor from the cut position. Some consideration should be given to how far away the sensor is located from the cutter. When a new INCA command is issued the target position changes immediately. If the new target position is just in front of the demanded position and the axis is moving at such a rate that there is insufficient distance to decelerate and achieve the new target position the axis will decelerate to a stop and then reverse to the target position. To avoid this happening we position the sensor a greater distance away from the cut position than the required deceleration distance.

The example code also contains a simple ONERROR event. If an error occurs whilst the program is running this will result in the event being called. The code contained within this event will print information such as error code, error description and line number (if applicable) to the terminal window. The motion task is ended and an error active flag is set. This flag forms an interlock with other parts of the program preventing the motion task from being started until the flag is cleared. Another task called “ErrorHandler” is then ran. The error handler will pause the program waiting for any key to be pressed in the Workbench terminal window as a method of acknowledging and clearing the error active flag (you will need to click on the terminal window first so that the cursor is present). In a real application a reset button on an HMI would be used for this purpose.

The sample program also includes a stop handler framework (a template of which is present in the Mint library). A stop input and stop input mode are configured in the startup block. When the input activates the axis stops according to the programmed stop input mode and the program flow will jump to the Mint stop event. This in turn process a subroutine called “doHandleStop”. The subroutine sets the stop active flag which interlocks with other parts of the program to prevent motion. The routine then runs a task called “StopHandler” which waits for the stop input to clear before resetting the stop active flag.

Should you require further help support with an application such as this please do not hesitate to contact your local ABB technical support office.

Contact us

For more information please contact your
local ABB representative or one of the following:

new.abb.com/motion
new.abb.com/drives
new.abb.com/drivespartners
new.abb.com/PLC