AN00103-003 - Product Labeling

Related Applications or Terminology

- *Modifying final / target position ‘on the fly’*
- *Cut to length with registration control*
- *Press feeder component alignment*
- *Product sleeving*
- *Emergency target change*
- *Label feeding - label position correction*

Overview

The illustration below shows the layout of a typical labeling machine.

- Moveable roller which takes up slack while dispensing a label
- Take-up reel
- Motor driven pinch roller assembly
- Supply reel
- Optical sensor for detecting the edge of a label
- Peel Bar
- Optical sensor for detecting the edge of a product
- Maste

Supported Controllers

- NextMovePCI
- NextMoveBX
- NextMoveST
- NextMoveES
- NextMoveESB
- NextMove100
- MintDrive
- Flex+Drive

Relevant Keywords

- MASTERDISTANCE
- FLY
- EVENT FASTIN
- FASTLATCHMODE
- FASTPOS
- FASTSELECT
Randomly spaced products pass along a conveyor belt and are detected by a photoelectric sensor. The signal from this sensor is connected to the Mint motion controller input assigned for use as the 'Fast Position Latch' input. This latches the position of the pinch roll motor's encoder within 1µS of its occurrence.

A servomotor is used to drive the pinch rolls, which in turn pull the label stock off of the supply reel. A peel bar ensures that the labels are removed from their backing at a fixed point within the process.

The master encoder (or conveyor) signal is fed into the master/auxiliary encoder input on the Mint motion controller. This encoder signal is used to track movement of the product along the conveyor belt. The counts per revolution will be selected for the accuracy needed in the system. The higher the number in counts per rev (and relationship of the distance the conveyor travels for that number of encoder counts) will determine the final accuracy attainable by the system. A web break sensor (not shown) can be placed somewhere in the path of the label stock to give an immediate signal (interrupt) to the controller to allow it to react accordingly. This input could be assigned as a dedicated STOPINPUT for example.

The label stock must ramp up to a speed that matches the speed of the product passing by. Once at speed it must remain locked to the same speed as the product while the label is being applied. Once the label has been applied the label stock must then ramp down to a stop and position the next label on the peel bar ready for the next apply cycle. The requirement to match speeds over known distances makes this application ideal for the MintMT FLY command.

The cycle starts as soon as the 'product present' sensor detects an object. Because the sensor is located ahead of the peel bar the program will command that the product travel a specified distance (in master encoder counts) before the label stock is accelerated up to synchronous speed.

It is common to make this initial delay adjustable by the operator as this provides a method of setting the label position on the product. A Baldor HMI panel can be connected to provide this user interface.

**Worked Example**

A labeling machine similar to that illustrated earlier has the following characteristics:

The master encoder is mounted on the product conveyor with 1000 pulses per rev (4000 quadrature encoder count)

- The conveyor travels 0.075 mm for each encoder count
- A Baldor BSM resolver-based servo motor, driven via a 3:1 speed reduction gearbox by a MintDrive™, is mounted to drive a set of pinch rollers (nips) with a 60mm circumference (the drive will resolve 4096 counts per motor revolution yielding label travel resolution of 0.0048 mm per count)
- Length of label to be applied is 75mm
- The product detection sensor is located 30mm away from the edge of the peel bar

After waiting the initial delay distance the BSM motor will ramp up to speed as the product travels a further 6mm.
The number of encoder counts produced by the conveyor encoder over this 6mm period can be calculated from:

\[ \text{Encoder count} = \frac{\text{Distance traveled}}{\text{Resolution}} \]

Substituting into this formula for the conveyor results in:

\[ \text{Encoder count} = \frac{6}{0.075} = 80 \text{ counts} \]

In a flying shear application, the relationship between the distance traveled by the master axis (\text{MASTERDISTANCE}) and the distance traveled by the slave axis (\text{FLY}) is as shown in the figure below.

This figure illustrates that for the slave axis to accelerate up to synchronous speed with the master axis it must travel the distance denoted by area B, whilst at the same time the master axis travels the distance denoted by A+B. It can also be clearly seen that B = A (or put another way, the slave travels exactly half the distance traveled by the master).

Whilst at synchronous speed the master and slave axes travel the same distance (C).

To decelerate to a stop the slave again travels half the distance (D) traveled by the master axis (D + E).

We have already stated that our slave axis will accelerate over a period of 6mm travel on the master axis, hence we can now see that the slave must travel 3mm to achieve synchronous speed.

Again we can convert this to slave encoder counts from the formula

\[ \text{Encoder count} = \frac{\text{Distance traveled}}{\text{Resolution}} \]

Substituting into this formula for the slave (BSM) axis results in:

\[ \text{Encoder count} = \frac{3}{0.0048} = 625 \text{ counts} \]
We can now construct our first two flying shear segments....

\[
\text{MASTERDISTANCE}(0) = n\text{DelayPeriod} \quad \text{Variable initial delay}
\]
\[
\text{FLY}(0) = 0 \quad \text{Slave distance traveled} = 0
\]
\[
\text{MASTERDISTANCE}(0) = 80 \quad \text{Accelerate up to synchronous}
\]
\[
\text{FLY}(0) = 625 \quad \text{speed over 6mm of master}
\]

The remainder of the profile is made up from two further FLY segments.

Once at synchronous speed (where the distance traveled by the slave equals the distance traveled by the master) and one to bring the slave axis to a halt (where the distance traveled by the slave is half the distance traveled by the master).

The synchronous distance traveled by the slave must be the total label length – the distance traveled to accelerate up to speed (i.e. 75mm – 3mm). This equates to 15000 counts.

Hence the \text{MASTERDISTANCE} value must correspond to 72mm of product travel (960 counts).

We will bring the slave axis to a halt over 6mm of travel of the master. This allows us to use the values previously calculated for \text{MASTERDISTANCE} and FLY during the acceleration phase.

We can now construct our final two flying shear segments....

\[
\text{MASTERDISTANCE}(0) = 960 \quad \text{Product travels 72mm}
\]
\[
\text{FLY}(0) = 15000 \quad \text{Slave distance traveled} = 72mm
\]
\[
\text{MASTERDISTANCE}(0) = 80 \quad \text{Decelerate to zero speed}
\]
\[
\text{FLY}(0) = 625 \quad \text{over 6mm of master travel}
\]

We have now completed all the code necessary for applying the labels. The next stage is to ensure that subsequent labels on the label stock are consistently positioned at the peel bar.

The labels will, to some extent, be positioned irregularly on their backing material so it is not possible to implement a fixed distance move. Instead the position of the ‘next’ label must be detected as the previous label is being applied.

The optical sensor detecting edges of labels is connected to one of the fast inputs on the Mintdrive\textsuperscript{11} so that a leading edge signal can be used to capture (within 1µS) the position of the label following the one being applied. This position is automatically latched to the Mint keyword FASTPOS (if using a Nextmove e100 the FASTxxxxx keyword group is replaced by the LATCHxxxxx keyword group – refer to the MintMT Help File for further details).

As the sensor is a known fixed distance from the peel bar (30mm or 6250 counts in our case) then, once the flying shear apply cycle has been completed, an absolute positional move can be implemented to locate the next label correctly. This is usually such that the lead edge of the label is just hanging off of the peel bar.

This absolute position will correspond to the captured label position + the fixed distance. By assigning a variable (nPeelPos for example) to the fixed distance the HMI can be used to make fine adjustments if necessary.
The Mint code to implement this label positioning is therefore...

```plaintext
PAUSE IDLE(0) ' Wait for the slave axis to halt motion
MOVEA(0) = FASTPOS(0) + nPeelPos ' Calculate target position
GO(0) ' Implement move
```

We have now completed the key sections of Mint code relating to motion of the label stock. The following page shows how these sections of code may be combined to form a complete MintMT application and how, by using variables, the code can be made flexible to suit almost any label length and machine configuration.

```plaintext
Dim nLabelLength As Integer ' Define a variable for theoretical cut length
Dim nDelayPeriod As Integer ' Define a variable for setting label position
Dim nPeelPos As Integer ' Variable distance from label sensor to peel bar
Dim fSlaveRes As Float = 0.0048 ' Travel (mm) per count on the slave axis (label)
Dim fMasterRes As Float = 0.075 ' Travel (mm) per count on the master axis (product)
Dim nAccelDist As Integer ' Label accels up to speed over this master distance
Dim nDecelDist As Integer ' Label decels to a halt over this master distance
Dim nSlewDist As Integer ' Label runs at slew speed over this master distance
Dim nSlaveAccel As Integer ' Label accels up to speed over this slave distance
Dim nSlaveDecel As Integer ' Label decels to a halt over this slave distance
Dim nSlaveSlew As Integer ' Label runs at slew speed over this slave distance
Dim bProductSeen As Integer = _false ' Flag to indicate change of m/c parameters
Const _nHMI = 2 ' HMI is CANopen node 2 – drive must be Node 1

' *** TIP *** Use Define to give remote Coms locations application names
Define cmHMI_ACCEL = Comms(_nHMI, 2) ' HMI panel sets master accel distance
Define cmHMI_DECEL = Comms(_nHMI, 3) ' HMI panel sets master decel distance
Define cmHMI_PEEL = Comms(_nHMI, 4) ' HMI panel sets distance from sensor to peel bar
Define cmHMI_DELAY = Comms(_nHMI, 5) ' HMI panel sets initial delay (label position)
Define cmHMI_LENGTH = Comms(_nHMI, 6) ' HMI panel sets label length

Run HMI ' Multi-tasking allows the HMI task to run independently
Pause Driveenableswitch ' Wait for external drive enable signal
Reset(0) ' Clear errors and enable drive

' The parent task executes continuously
Loop
    Pause bProductSeen And IDLE(0) 'Wait for product detection
    MASTERDISTANCE(0) = nDelayPeriod 'Variable initial delay
    FLY(0) = 0 'Slave distance travelled = 0
    MASTERDISTANCE(0) = nAccelDist 'Accelerate up to synchronous speed
    FLY(0) = nSlaveAccel
    MASTERDISTANCE(0) = nSlewDist 'Slew at synchronous speed
    FLY(0) = nSlaveSlew
    MASTERDISTANCE(0) = nDecelDist 'Decelerate to a stop
    FLY(0) = nSlaveDecel
    GO(0)
    bProductSeen = _false 'Get ready to wait for next product
End Loop
```
' Task to handle HMI interface
Task HMI
  ' Initialise the HMI
  BUSBAUD(1) = 125  ' 125 kBaud
  BUSRESET(1)
  NODETYPE(1, nHMI) = 0
  Repeat
     NODESCAN(1, nHMI)
  Until NODELIVE(1, nHMI)
  CONNECT(1, nHMI) = 1
  ' Once found make a CANopen connection
  Task loops forever waiting for new data once initialized
  Loop
     ' Call subroutine to calculate fly segments
     subCalcSegments
     bDataChanged = _false
  End Loop
End Task

' Subroutine to calculate flying shear segments
Sub subCalcSegments()

  ' Operator enters all system parameters in mm
  nDelayPeriod = Int (cmHMI_DELAY / fMasterRes)
  nAccelDist = Int (cmHMI_ACCEL / fMasterRes)
  nDecelDist = Int (cmHMI_DECEL / fMasterRes)
  nSlewDist = Int ((cmHMI_LENGTH – cmHMI_ACCEL) / fMasterRes)
  nSlaveAccel = Int (cmHMI_ACCEL / (2 * fSlaveRes))
  nSlaveDecel = Int (cmHMI_DECEL / (2 * fSlaveRes))
  nSlaveSlew = Int ((cmHMI_LENGTH – cmHMI_ACCEL) / fSlaveRes)
  nPeelPos = Int (cmHMI_PEEL / fSlaveRes)

End Sub

' Interrupt triggers the EVENT INx (where x denote the input number)
Event In0

  bProductSeen = _true
  ' Set flag to indicate lead edge of product found
End Event

' Fast Interrupt triggers the EVENT FASTINx (where x denote the input number)
Event FastIn4

  PAUSE IDLE(0)
  MOVEA(0) = FASTPOS(0) + nPeelPos
  ' Wait for the slave axis to halt motion
  ' Calculate target position
  GO(0)
  ' Implement move
End Event
'This event will be generated automatically if the operator changes HMI Data
Event BUS1

Dim nEvent As Integer ' Store CANopen event locally
Dim nEventInfo As Integer ' Store additional event info locally

nEvent = CANEVENT(1)
nEventInfo = CANEVENTINFO(1)

If nEvent = _betUNIOP_COMMS_UPDATE Then ' If any HMI data has changed

' Comms 1 is used to indicate new parameters have been entered
Select Case nEventInfo
    Case 1
        bDataChanged = _true
    Case Else
        bDataChanged = _false
End Select

End If

End Event

' Startup code – executed when the controller is powered up
Startup

' Digital Input Configuration – Inputs 0 & 4 active high and +ve edge triggered
INPUTMODE(0) = 000010001 ' *** TIP *** Bit-mapped keywords are easier to understand if you specify
INPUTACTIVELEVEL(0) = 011111111 ' the setting using binary number
INPUTPOSTRIGGER(0) = 000010001 ' format
INPUTNEGTRIGGER(0) = 000000000 ' Input 4 captures Fast position
FASTSELECT(0) = 4
FASTLATCHMODE(0) = 1 ' Allow recapture once Fastin event finished
MOVEBUFFERSIZE(0) = 5 ' Allow up to five moves to be loaded into buffer

End Startup