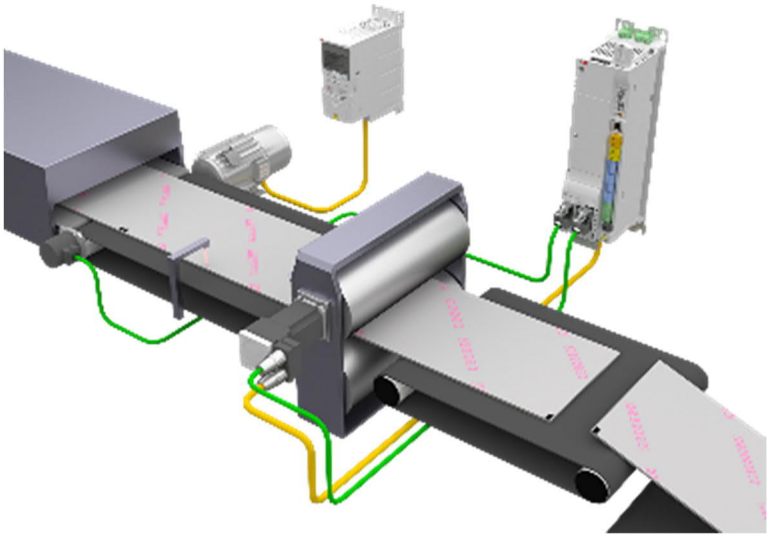
Motion Control Products

Application note
Rotary Cutter

AN00226

Rev D (EN)

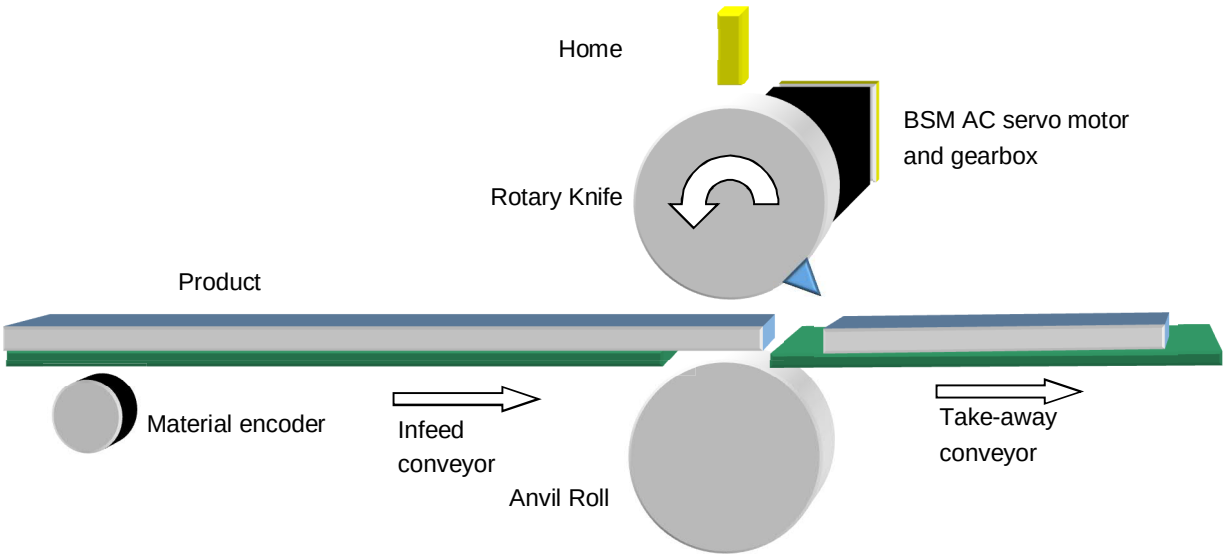
The term “flying shear” usually brings to mind an application where a linear cutting mechanism is synchronised to a moving product to cut it to length. But the same techniques can be used in Mint to achieve outstanding results with rotary cutting mechanisms too



Introduction

Synchronization between two axes may be required for all sorts of applications. AN00116 illustrates one example of this, a linear flying shear application which requires precise synchronization of axis speeds over defined product and axis travels. AN00116 showed how the Mint FLY and MASTERDISTANCE keywords can be used to solve this type of application with ease. AN00122 then expanded on these concepts to show how the same keywords may be used to synchronize a rotating axis with a moving product, using an intermittent embossing wheel as an example application.

This application note, AN00226, provides a further example of the use of the Mint FLY and MASTERDISTANCE keywords, using a very common application type – a rotary knife. The code provided with this application note will execute on both the MicroFlex e190 and MotiFlex e180 intelligent servo drives and an example CP600 series HMI project is also included for reference.



The illustration above shows a simplified version of the typical features for a rotary cutter application. A conveyor moves the product / material to be cut to length at a constant linear velocity through the machine. The conveyor may be driven by an induction motor and variable frequency drive (e.g. ACS355). An encoder detects the position and speed of the conveyor as it

runs. This is usually termed the ‘master’ or ‘material’ encoder. A second conveyor usually runs slightly faster than the first conveyor to take away the cut lengths of material (and to therefore create a gap between finished products). The rotary cutter itself is driven by an AC servo motor (via a suitable gearbox to ensure good inertia matching between the load and the motor) which in turn is connected to either a MicroFlex e190 or MotiFlex e180 intelligent servo drive.

The drive monitors the material encoder (wired to encoder channel 2 on the drive) and uses it to determine the velocity / position profile for the cutter motor. Both the material encoder and the rotary cutter are scaled into units of linear travel (so for the material this is the number of counts per unit of conveyor travel and for the rotary cutter this is the number of counts per unit of travel around the cutter circumference).

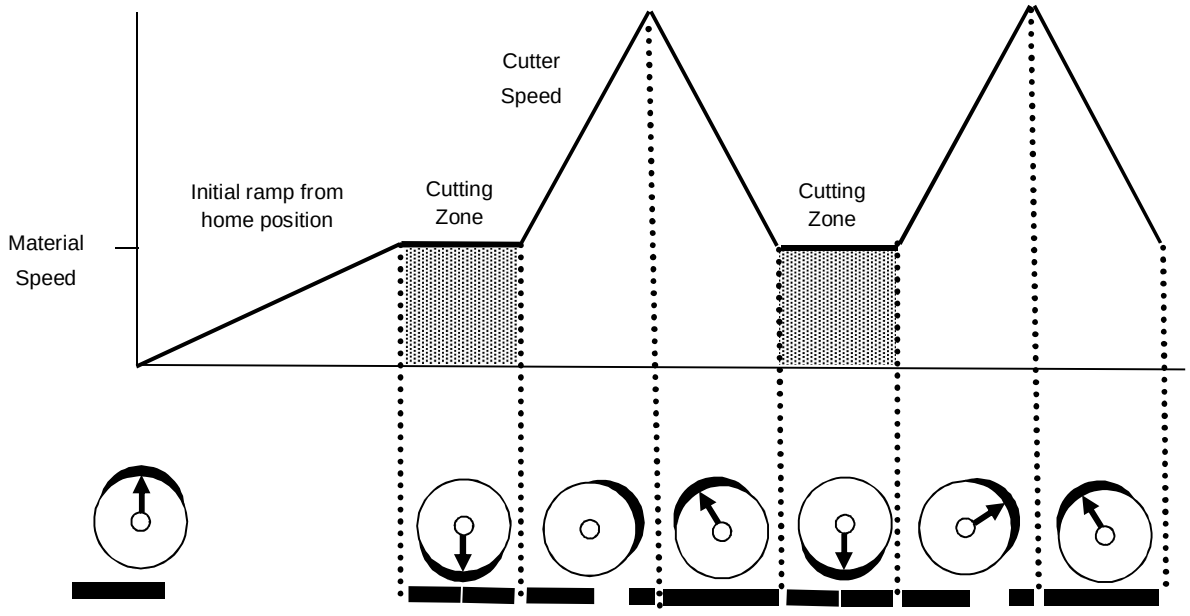
Motion profile

It is typical for the surface speed of the rotary cutter to be synchronized to the material over a specified angle of the cutter itself (e.g. 30 degrees, 15 degrees either side of the cut point at the bottom dead centre of the cutter’s rotation).
As the cutter is scaled into units of travel around the circumference this angle is converted into linear travel (e.g. if the cutter circumference was 300 mm then 30 degrees would equate to 300 * 30 / 360 = 25 mm of travel around the circumference).

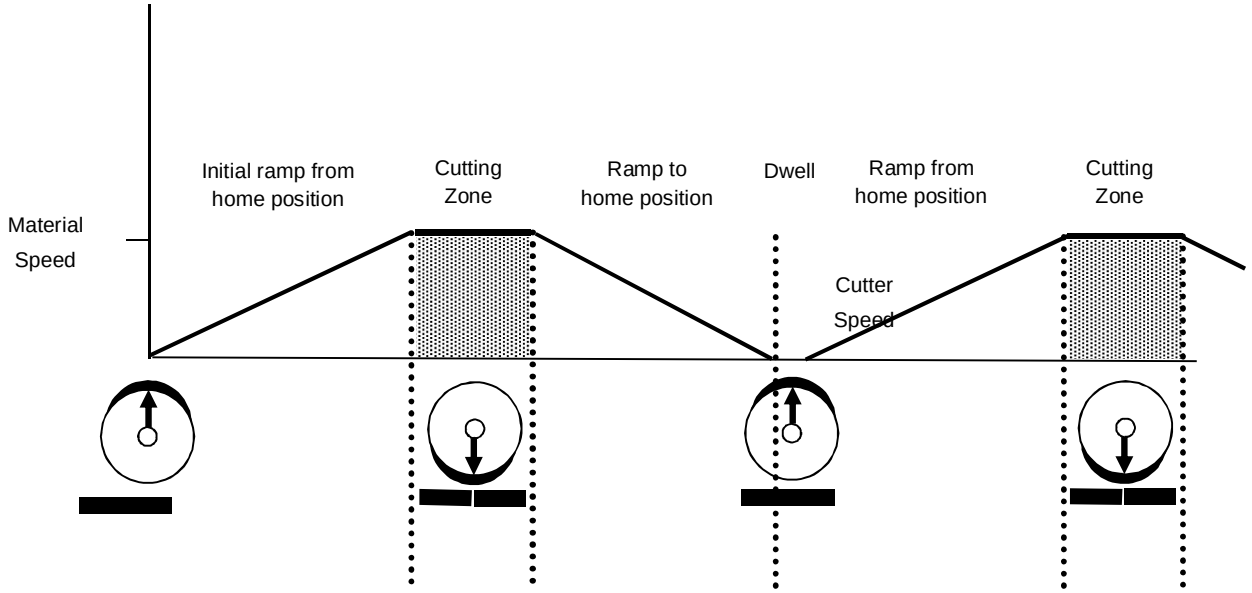
Usually the synchronization between the two axes is 1:1, so if the cutter travels 25 mm around its circumference during the cutting region the material itself will also travel 25 mm. As a result, to achieve the overall cut length the cutter must complete the remainder of one revolution over the remaining material travel (e.g. if the required cut length is 150 mm the cutter must rotate the remaining 275 mm of its circumference over the next 125 mm of material travel).

The figure below illustrates the relationship between the speed ratios of the two axes. For our example we’ll assume the cutter start position is aligned with the home sensor (i.e. at the “12 o-clock” position as shown in our original simplified application diagram).

The shaded area indicates when the surface speed of the rotary axis exactly matches the linear speed of the material:



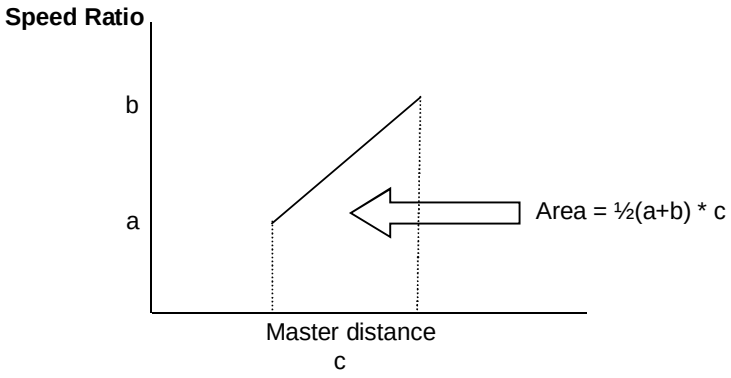
The motion profile above shows how the cutter axis velocity might appear for cut lengths shorter than the circumference of the cutter itself (i.e. the cutter has to speed up in between successive cuts). As the required cut length increases the speed ratio between cuts decreases until eventually the cut length is so long that the cutter has to stop / dwell for some time before restarting…



Each element of the required speed profile can be split into what are termed ‘segments’ (this has also been illustrated on the figures above). The distance travelled by the material for each segment can be specified in Mint by using the MASTERDISTANCE keyword.

For each segment the area under the curve corresponds to the distance travelled by the cutter axis. In Mint programs this distance/area can be specified using the FLY keyword.

If we examine any segment we can see that the segment area (i.e. the value for FLY) can be derived from:



Area = Master distance * (Initial Speed Ratio + Final Speed Ratio) /2

Translating this into Mint code we get…

FLY(0) = MASTERDISTANCE(0) * (Initial Speed Ratio + Final Speed Ratio) / 2

This generic formula can be used to calculate FLY if only MASTERDISTANCE is known and vice-versa.

Simple application example

Consider our rotary cutting machine to have the following characteristics:

Conveyor information:
Material encoder driven via lay on measuring wheel with 100 mm diameter (100 x pi mm circumference)
Material encoder has 4096 line (16384 quadrature) resolution
Encoder connected to encoder channel 2 on MicroFlex e190 or MotiFlex e180 servo drive

Cutter axis information:
Cutter diameter = 200 mm (200 x pi mm circumference)
Motor encoder resolution = 131072 (quadrature) counts per revolution
Gearbox ratio = 3:1
Total synchronized angle = 40 degrees
Cutting speed ratio = 1:1

Note that for this example we are “hard coding” values for the synchronization angle, cutter diameter and cutting speed ratio but in practice these values are very often adjustable from an HMI so it is typical to use variables in the calculations to make the code more general and suitable for systems with varying dimensions/properties.

The first thing we should do is scale our material encoder and cutter axis into linear mm of travel:

ENCODERSCALE(2) = 16384 / (100 * pi) ‘counts per rev of encoder / measuring wheel circumference
SCALEFACTOR(0) = (131072 * 3) / (200 * pi) ‘counts per gearbox output rev / cutter circumference

We can now start to calculate our MASTERDISTANCE and FLY segment values for each segment of our cutter profile using the generic formulae for flying shears that we created earlier…

FLY = MASTERDISTANCE * (Initial Speed Ratio + Final Speed Ratio) / 2
⇨ MASTERDISTANCE = (2 * FLY) / (Initial Speed Ratio + Final Speed Ratio)

Start segment:
For this example we will assume the cutter starts at the home position (top dead centre) and must ramp up to the start of the synchronized cut region (20 degrees before bottom dead centre; i.e. 160 degrees of travel from the home position) so that the speed becomes matched 1:1 with the material.

Initial ratio = 0 ; Final Ratio = 1 ; FLY = (200 * pi) * 160 / 360
⇨ MASTERDISTANCE = ((2 * 200 * pi) * 160 / 360) / (0 + 1) = (400 * pi) * 160 / 360

In our Mint program we would therefore include these two lines of code to start the cutter:

MASTERDISTANCE(0) = (400 * _pi) * 160 / 360
FLY(0) = (200 * _pi) * 160 / 360

If we calculate this value of MASTERDISTANCE we will find that the result is approximately 558.5 mm. It can therefore be seen that the very first cut length will be at least this distance plus the material travel to the cut point (i.e. half of the synchronized MASTERDISTANCE that we will calculate next). In a real application we might therefore consider alternative ways to start the cutter, depending on whether the first cut needs to produce a correct length product or not. We will examine this in detail later in this application note.

Synchronized cut segment:
Initial Ratio = 1 ; Final Ratio = 1 ; FLY = (200 * pi) * 40 / 360
⇨ MASTERDISTANCE = ((2 * 200 * pi) * 40 / 360) / (1 + 1) = (200 * pi) * 40 / 360

In our Mint program we would therefore include these two lines of code to carry out the synchronized cut:

MASTERDISTANCE(0) = (200 * _pi) * 40 / 360
FLY(0) = (200 * _pi) * 40 / 360

(As our speed matching ratio is 1:1 and both axes are scaled into linear mm of travel it should be no surprise that the MASTERDISTANCE and FLY values are identical!)

Remaining cutter segments:
To calculate these remaining segments we must first determine whether we need to load only two segments (splitting the remaining cutter travel and material travel in half for each segment) or whether we need to load three segments (including a dwell segment to ensure our overall cut length is correct).

To make this decision we need to calculate the total master distance (material travel) achieved during:

- the synchronized cut region
- bringing the cutter to a stop from the cut ratio
- restarting the cutter to the required cut ratio

If the sum of these master distances is less than the required cut length then we need to include a dwell segment to extend the cutting cycle. If the sum of these master distances is greater than (or equal to) the required cut length then it is only necessary to load the synchronized cutting segment and then split the remaining distances in half.

We have already calculated the master distance for the synchronized cut region: (200 * _pi) * 40 / 360
We have also already calculated the master distance required to take the cutter from zero speed to the cut ratio, when we calculated the starting segment values: (400 * _pi) * 160 / 360
If we assume that the cut ratio is fixed at 1:1 and never changes then we can also see that the master distance to stop the cutter is equal to the master distance needed to start the cutter : (400 * _pi) * 160 / 360

We can calculate this “total master distance” for our application and store this in a Mint variable named fTotalMSD…

fTotalMSD = ((200 * _pi) * 40 / 360) + (2 * (400 * _pi) * 160 / 360)

If we assume the cut length is entered into the system via an HMI and eventually stored somewhere in a Mint variable named fCutLength we therefore have all the information needed to write the required application logic…

```
If fTotalMSD < fCutLength Then

    ‘Three segments needed…..Stop, Dwell and Restart…
    MASTERDISTANCE(0) = (400 * _pi) * 160 / 360
    FLY(0) = (200 * _pi) * 160 / 360

    MASTERDISTANCE(0) = fCutLength – fTotalMSD
    FLY(0) = 0

    MASTERDISTANCE(0) = (400 * _pi) * 160 / 360
    FLY(0) = (200 * _pi) * 160 / 360

Else

    ‘Two segments needed…..each is half the remaining distance
      MASTERDISTANCE(0) = (fCutLength – ((200 * _pi) * 40 / 360)) / 2
      FLY(0) = ((200 * _pi) * (360 – 40) / 360) / 2

      MASTERDISTANCE(0) = (fCutLength – ((200 * _pi) * 40 / 360)) / 2
      FLY(0) = ((200 * _pi) * (360 – 40) / 360) / 2

End If
```

Having loaded a complete cycle our application can now keep repeatedly loading the synchronized cutting segment and the required “remaining segments” continually to keep cutting the material to the required length. Some applications require that the cutter is stopped at the home position before the material stops, others leave the cutter permanently “cycling” once started so the stop position is completely random, depending on when the material is stopped. For the purposes of our simple example we will assume the cutter remains “in cycle” and stops and starts with the stopping and starting of the material once initially started.

The Mint MOVEBUFFERSIZE must be set to ensure there is adequate space to load the required segments and an appropriate motion trigger command (typically GO) must be used to action the motion, refer back to AN00116 or the Mint help file for further details about sizing the move buffer and triggering motion.

Compensating for axis drift

As can be seen from the preceding example code, most of the logic for loading MASTERDISTANCE and FLY segments involves floating point mathematical calculations. As a result there will inevitably be some rounding errors over time, the net effect being the cutter axis will “drift” out of position over time (so eventually the speed will be synchronized with the material when the knife is somewhere other than the 40 degree region near the bottom dead centre of the cutter.

We therefore need a mechanism in our code to detect the position of the cutter within its cycle and compare this with where the controller believes the cutter should be. This is most easily achieved using a combination of the MOVEPULSEOUTX instruction (which allows the controller to pulse a digital output with a precise position with respect to the motion profiler) and the Mint Latch Event (which can be used to capture the cutter axis encoder value). By configuring the Latch event to trigger from the digital output we have a simple method by which we can compare the actual cutter position within one cycle with its ideal value. We can then use the Mint OFFSET command to “phase shift” the axis to compensate for the measured drift.

Steps to configure this behaviour are:

(a) Configure an ENCODERWRAP for the cutter axis. In our example the cutter motor is fitted with a 3:1 gearbox and the motor’s encoder resolution is 131072 counts per rev resulting in an ENCODERWRAP(0) of 393216. If the application properties are such that there are a fractional number of encoder counts per cutter cycle it will be necessary to use the home sensor (via an additional Latch event) to reset the cutter ENCODER value to the expected value for the given home sensor position every cycle

(b) Include a MOVEPULSEOUTX command in the middle of the synchronized cutting segment – by placing the MOVEPULSEOUTX here we ensure that the subsequent OFFSET command to correct for the measured drift does not take place until the cut has taken place. To insert the MOVEPULSEOUTX into the middle of the synchronous region requires a split of the previous segment into two halves (this instruction requires an additional space in the move buffer)

```
MASTERDISTANCE(0) = ((200 * _pi) * 40 / 360) / 2
FLY(0) = ((200 * _pi) * 40 / 360) / 2
MOVEPULSEOUTX(0, 1) = 10   ‘Pulse output 1 for 10ms
MASTERDISTANCE(0) = ((200 * _pi) * 40 / 360) / 2
FLY(0) = ((200 * _pi) * 40 / 360) / 2
```

(c) Configure a Latch event (in the Mint startup module) that is captures the cutter latched encoder value whenever output 1 operates…

```
LATCHENABLE(1) = 0
LATCHSOURCE(1) = _lsENCODER
LATCHSOURCECHANNEL(1) = 0
LATCHTRIGGERMODE(1) = _ltmOUTPUT
LATCHTRIGGERCHANNEL(1) = 1
LATCHTRIGGEREDGE(1) = _ltePOSITIVE_EDGE
LATCHMODE(1) = _lmAUTO_ENABLE + _lmINHIBIT_VALUE
LATCHINHIBITVALUE(1) = 0.8 * 200 * _pi
```

For this example we have used latch channel 1. Note that by using an inhibit value mode we can use 80% of the cutter circumference a filter distance to prevent unintended triggering of the fast latch.

(d) Enable the “anti-drift” latch once the axis has successfully homed using…

```
LATCHENABLE(1) = 1
```

(e) Configure the OFFSETMODE for the cutter axis to apply the drift corrections over a distance that means the corrections are not applied quickly, but equally allows them to complete before the next synchronized cut segment occurs…

```
OFFSETMODE(0) = _ofTWO_SEGMENT
OFFSETDISTANCE(0) = fCutLength * 300 / 360  '300 degrees out of the next 360 degrees
```

(f) Include the Latch event that calculates and applies the drift correction…

```
Event LATCH1
  'Knife should be at bottom dead centre (the cut point) every time this latch occurs (i.e. at 180 degrees of travel)...
  fLatch = LATCHVALUE(1)
  fDrift = WrapOffset(fLatch, (180 / 360) * (_fKnifeDia* _pi), 0, (_fKnifeDia* _pi))
  If (AXISMODE(0) And _mdOFFSET) <> _mdOFFSET Then
    OFFSET(0) = fDrift
    GO(0)
  End If
End Event
```

Modifying the start segment

Again, for this topic we will use hard coded values from our application example to hopefully make it easier to understand the principles of operation, but in practice it is better to use variables and generically calculate relevant data for the program to make program flow decisions about (see the Mint application code included with this application note for examples of how to achieve this).

As discussed earlier, by ramping the cutter axis up to a 1:1 ratio from the home position in a single segment the material will travel 558.5mm plus half the synchronized segment (34.906 mm, making approx. 593.4 mm in total) before the first cut occurs. For long product lengths this initial “wrong length” may not prove an inconvenience, but for short lengths (e.g. 200 mm products) this could be a significant wastage and could prove difficult for the outgoing conveyor and any subsequent process to handle. So for product lengths less than 593.4mm it would be advantageous to modify the start segment so that the correct cut length is achieved immediately.

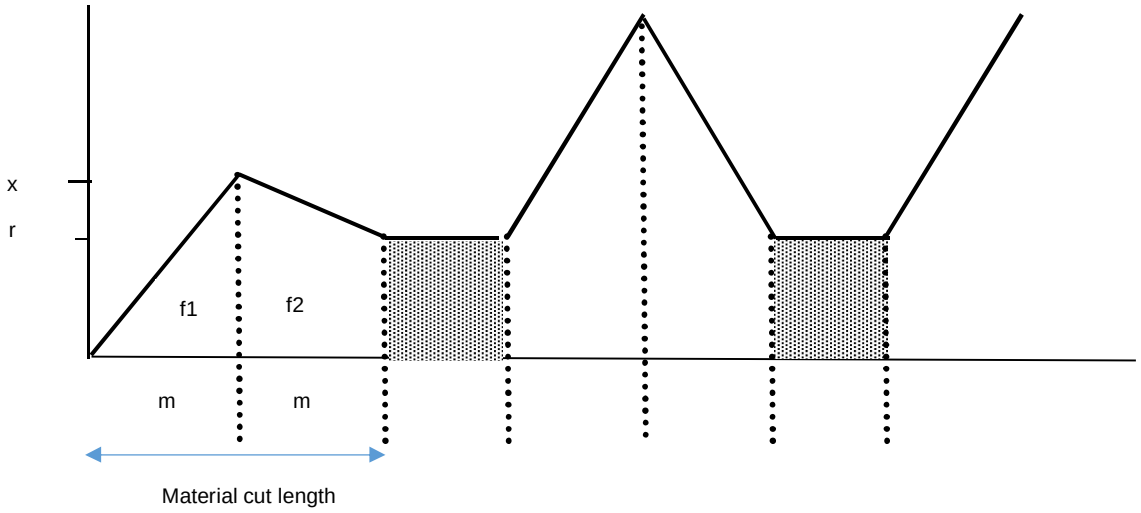
Similarly, for long product lengths it may be preferable to include an initial delay (i.e. MASTERDISTANCE with a FLY of zero) before loading the default starting segment.

In both cases it is typical to initially home the cutter so that the material starts feeding from a “new” product lead edge.

For the long product length the MASTERDISTANCE for the dwell segment is easily calculated – it is the difference between the required cut length and the total master distance we will travel for our default start segment and half the synchronized segment (in our example fProductLength – 593.4).

For short product lengths (less than 593.4mm in our case) it is typical to split the starting segment into two, dividing the master distance for the start in half, making each master distance for the two segments one half of the required cut length. The sum of the two FLY segments then used to start the cutter must total the linear distance the cutter travels around its circumference to travel from the home position to the start of the synchronized segment. As our synch angle is 40 degrees wide, +/-20 degrees

either side of the bottom dead centre, the sum of the FLY segments must total (200 * _pi) * (180-20) / 360 = 279.253 mm (any slight error in this value will be corrected by the drift compensation code described earlier). The diagram below illustrates this principle:



m = cut length / 2 [for this example let's assume we required a 200 mm cut length]
f1 + f2 = (200 * _pi) * (180 – 20) / 360 [279.253 mm but for clarity of working we'll call this F]
x is our unknown speed ratio which acts as the final speed ratio for our first fly segment and the start ratio for our second fly segment
r is the final speed ratio for the second segment (typically 1:1) which is then used to start the synchronized cut segment

Using the generic formula for flying shears we can derive the following equations…

0.5 * (0 + x) * m = f1
0.5 * (x + r) * m = f2

…and using f1 + f2 = F we can substitute these two equations to give us…

(0.5 * x * m) + (0.5 * (x + r) * m) = F

Expanding these gives…
(0.5 * x * m) + (0.5 * x * m) + (0.5 * r * m) = F

⇨ (x * m) + (0.5 * r * m) = F

Re-arranging to find x results in:

x = (F – (0.5 * r * m)) / m

We can then load our two segments by substituting this value of x back into our original two equations for f1 and f2:

MASTERDISTANCE(0) = m
FLY(0) = 0.5 * (F – (0.5 * r * m)) / m
MASTERDISTANCE(0) = m
FLY(0) = 0.5 * (((F – (0.5 * r * m)) / m) + r) * m

[Where F = 279.253 and m = 100]

Example files

There are many other features that could be added to an application like this, rather than document them all, a sample Mint program (and HMI project using CP600 connected via Modbus TCP) is included with this application note that provides the following functions:

- HMI entry of application properties that are often adjusted at “run-time” (e.g. Synch angle, home speed, cut length, cut ratio)
- HMI display of completed cut counter
- Ability to change cutting parameters (e.g. cut ratio and cut length) whilst cutter is running
- Demonstrates use of program variables to create a generic solution rather than a hard coded specific solution
- Drift compensation logic
- Display of Mint application errors as string data on HMI (e.g. “Fatal following error exceeded”)
- Example program structure for handling Mint stop input activation and recovery
- Example program structure for handling Mint errors and recovery
- Example program structure for storing and retrieving HMI data that needs to be non-volatile
- Example program structure for setting default values when using Mint program for first time
- Example program structure for controlling drive’s enable status and interlocking this with other program logic
- Example home routine that shows how to ensure the home logic always ensures an initial cut is performed during homing
- Logic to ensure the first cut is subsequently the correct length (assumes the cutter is homed initially and material is not started until homing has been completed)

If you should require assistance with any additional features required for a rotary cutter application, please contact your local ABB office for technical support.

Contact us

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

new.abb.com/drives/low-voltage-ac/motion
new.abb.com/drives
new.abb.com/channel-partners
new.abb.com/plc