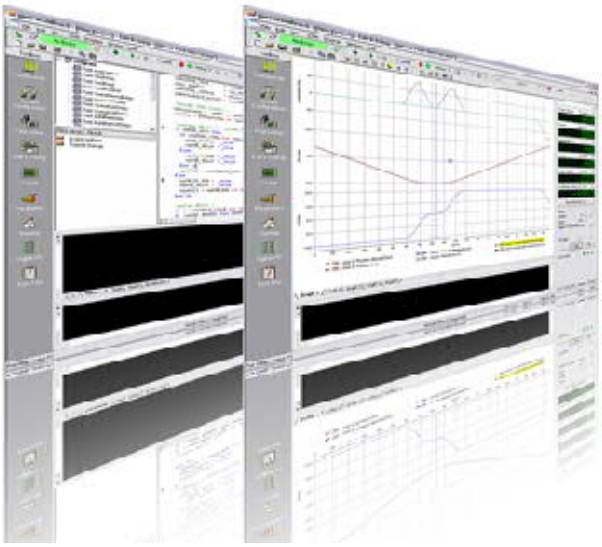
Motion Control Products

Application note
Using the move buffer

AN00117

Rev C (EN)

One of the powerful features of Mint is the move buffer. The move buffer allows multiple moves to be loaded in advance and triggered on demand. It allows many axes to be synchronised together and complex interpolated move types to be performed. Moves can be contoured together to make smooth paths, and feedrate control allows these moves to be performed with differing speeds and accelerations. Moves can also be blended together in order to reduce motion time. The move buffer can even inform a Mint program when it needs restocking



Loading and triggering moves

When a move command is issued, the move is stored in, and executed from, the move buffer. To start the move off, called triggering, the GO keyword is one method that can be used.
For example:

```
MOVER(0) = 10          ' Load a relative move of 10 units
GO(0)                  ' Trigger the motion
```

For some move types, it is possible to load more than one move at once.
For example:

```
MOVER(0) = 10          ' Load a relative move of 10 units
MOVER(0) = 20          ' Load a relative move of 20 units
GO(0)                  ' Trigger the motion
```

Here, two moves are loaded into the buffer and then triggered. The GO keyword will trigger all moves currently loaded in the move buffer so it only needs to be called once. The axis will move forward 10 units and stop. The axis will then move forward another 20 units.
The GO keyword can also be used to synchronise motion on multiple axes.
For example, the following code will start motion on axes 0 and 1 simultaneously:

```
MOVER(0) = 10          ' Load a relative move of 10 units on axis 0
MOVER(1) = 20          ' Load a relative move of 20 units on axis 1
GO(0,1)                ' Trigger the motion on axes 0 and 1
```

Sometimes it is inconvenient to trigger every move as it is loaded into the move buffer. The TRIGGERMODE keyword can be used to enable automatic triggering. This means that a move begins to execute as soon as it is loaded into the move buffer.

```
TRIGGERMODE(0) = _trAuto       ' Enable automatic triggering
```

```
MOVER(0) = 20                ' Load and start a relative move of 20 units
```

There are many other ways to trigger moves that have been loaded into the move buffer (e.g. it is possible to trigger motion on an axis when an encoder passes a particular value). Please refer to the TRIGGERMODE keyword in the Mint help system for further details.

Sizing the move buffer

Each axis has its own move buffer which can be set to a user defined size. For some move types, many moves can be loaded in advance, for others, only one move can be loaded. Mint will only halt program execution when it reaches a command that affects the move buffer, such as trying to load a move when the move buffer is full. It is important to make sure that your code will always trigger motion before the move buffer becomes full otherwise program execution (in that particular task) will halt. The following code segment shows this behaviour:

Insufficient move buffer size

```
MOVEBUFFERSIZE(0) = 2
MOVEA(0) = 10
MOVEA(0) = 20
MOVEA(0) = 30
GO(0)
```

Execution would halt here

Sufficient move buffer size

```
MOVEBUFFERSIZE(0) = 3
MOVEA(0) = 10
MOVEA(0) = 20
MOVEA(0) = 30
GO(0)
```

Where the move buffer size is set to two, after two moves have been loaded, program execution will pause (within the task) until a space in the buffer becomes available to load the third move. Where the move buffer size is set to three, the program flow is not halted.
A large move buffer size is required when moves are to be contoured together, so that the axes do not start and stop between each move. Increasing the move buffer size does increase memory usage and will slow Mint slightly. The move buffer size should be set in the Startup block of the Mint program before any axes are enabled. As well as there being a limit on the move buffer size per axis, there is also a limit for the total move buffer size for the controller as a whole (e.g. a NextMove e100 should be limited to 400 move buffer spaces in total).

Restocking the move buffer

Where moves are repeatedly loaded, it may not be practical to allocate the move buffer large enough to contain all the moves. The number of free spaces in the buffer can be read with the MOVEBUFFERFREE keyword. In the following example, moves are loaded using the Comms array. Position is written out to a keypad so program execution cannot be allowed to stop.

```
Loop
  ' Wait until data valid flag (Comms location 1) and there is space in the buffer
  If (COMMS(1) = 1) And (MOVEBUFFERFREE(0) > 0) Then
    MOVER(0) = COMMS(2)          ' Load move with data (Comms location 2)
    GO(0)                        ' Trigger motion
    COMMS(1) = 0                 ' Indicate to host that move is loaded
  Endlf
  Print #3, "Pos = ", POS(0)     ' Keypad is assigned to terminal 3
End Loop
```

Automatically restocking the move buffer (NextMove, MotiFlex e180 and MicroFlex e190)

To lessen the work involved in checking and keeping the move buffer stocked with moves, Mint can generate an event when the number of free spaces in the move buffer reaches a threshold. Moves can be loaded from the event handler and then program execution will resume as normal.

```
'Main code block
```

```
Loop
  …
End Loop
Event MOVEBUFFERLOW
  ‘ Wait until data valid flag (Comms location 1)
  If (COMMS(1) = 1) Then
    MOVER(0) = COMMS(2)          ‘ Load move with data (Comms location 2)
    GO(0)            ‘ Trigger motion
    COMMS(1) = 0             ‘ Indicate to host that move is loaded
  EndIf
End Event

‘ Startup code – executed when the controller is powered up
Startup
  ‘ Setup the move buffer
  MOVEBUFFERSIZE(0) = 20       ‘ Create 10 spaces in the move buffer
  MOVEBUFFERLOW(0) = 5         ‘ Create event when there are more than 5 spaces
End Startup
```

Keeping track of moves (NextMove, MotiFlex e180 and MicroFlex e190)

To keep track of moves in the buffer, it is possible to attach an identifier to each move as it is loaded and to read the identifier of the currently executing move. For example:

```
MOVEBUFFERSIZE(0) = 20          ‘ Create 20 spaces in the move buffer
MOVEBUFFERID(0) = 1             ‘ Set the ID to be stored in the buffer to 1
MOVER(0) = 20                   ‘ Load and start a relative move of 20 units
MOVEBUFFERID(0) = 2             ‘ Set the ID to be stored in the buffer to 2
MOVER(0) = 10                   ‘ Load and start a relative move of 10 units
GO(0)                           ‘ Trigger motion
Pause MOVEBUFFERID(0) = 2       ‘ Wait until the second move starts to execute
OUTX(0) = _ON                   ‘ Turn on digital output 0
```

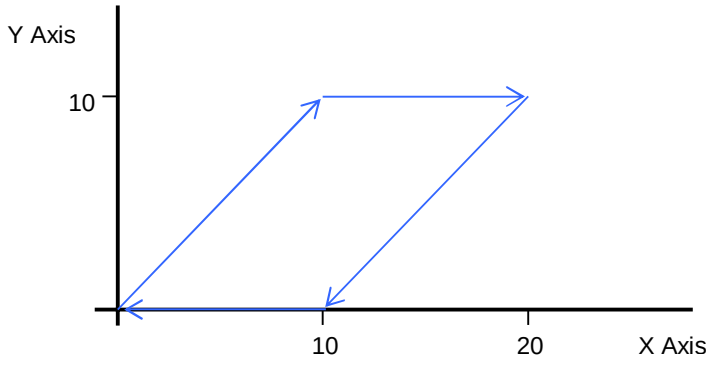
Once motion has finished, it is possible to read the ID of the last move executed using the MOVEBUFFERIDLAST keyword.

Interpolated (multi - axis) motion (multi-axis products only)

NextMove can perform the following multi-axis interpolated moves: VECTORA/VECTORR, CIRCLEA/CIRCLER and HELIXA/HELIXR. When a multi-axis move is loaded, the first axis specified is known as the ‘master axis’. The SPEED, ACCEL, DECEL, ACCELJERK and DECELJERK parameters set for the master axis are used for the motion path of the interpolated move. For NextMove ESB-2 it is important to set the move buffer size for **all** of the axes involved in a multi-axis move. If using NextMove e100 then it is only necessary to set the move buffer size for the master axis (thereby reducing the total number of move buffer spaces consumed).

```
ESB: MOVEBUFFERSIZE([0,1]) = 4;          ‘ Create 4 spaces in the move buffer for axis 0 and 1
e100: MOVEBUFFERSIZE(0) = 4   ‘ Create 4 spaces in the move buffer for axis 0
VECTORA(0,1) = 10, 10           ‘ Load an absolute vector move to position 10, 10
VECTORA(0,1) = 20, 10           ‘ Load an absolute vector move to position 20, 10
VECTORA(0,1) = 10, 0            ‘ Load an absolute vector move to position 10, 0
VECTORA(0,1) = 0, 0             ‘ Load an absolute vector move to position 0, 0
GO(0)                           ‘ Trigger motion
```

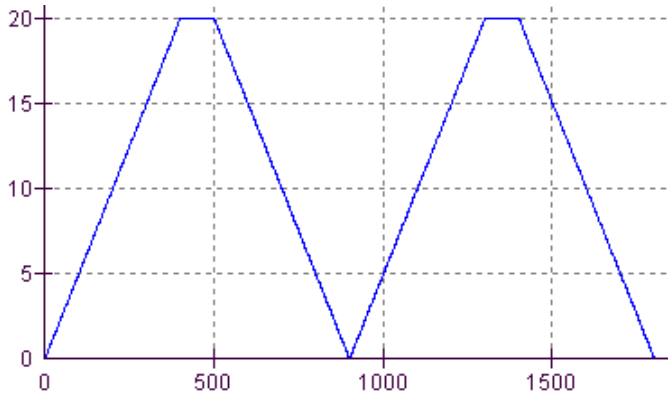
Note that to trigger motion on multi-axis moves it is only necessary to trigger the master axis.



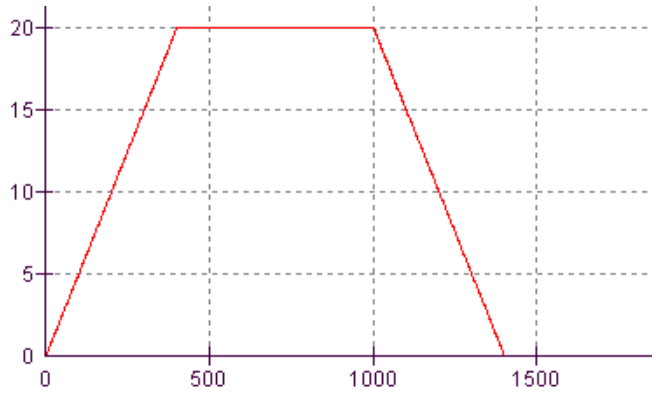
Creating contoured paths (NextMove, MotiFlex e180 and MicroFlex e190)

In a contoured move a series of moves are joined together (i.e. merged) so that they are performed with a constant path velocity. For example, the following plots show the effect of contouring:

```
CONTOURMODE(0) = _ctmCONTOUR_OFF
MOVER(0) = 10
MOVER(0) = 10
GO(0)
```

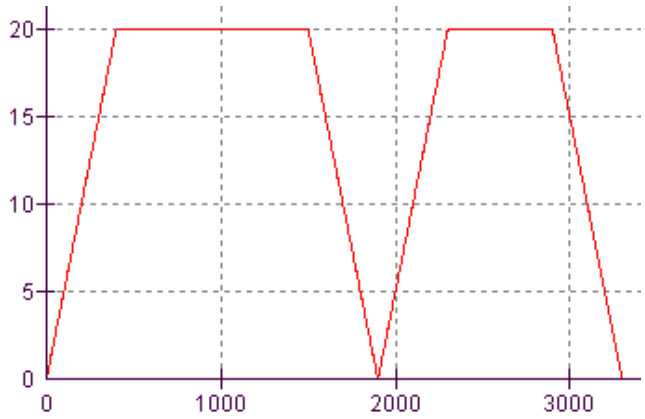



```
CONTOURMODE(0) = _ctmCONTOUR_ON
MOVER(0) = 10
MOVER(0) = 10
GO(0)
```



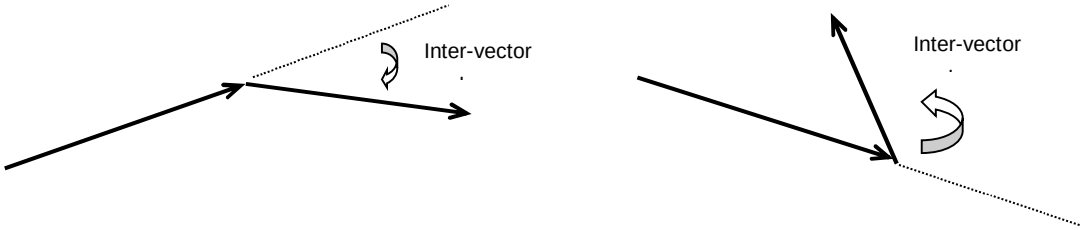
When a move is loaded into the buffer, the contour mode is stored as well, allowing some moves to be loaded with contouring turned on and others with it turned off. For example:

```
MOVEBUFFERSIZE(0) = 5                    ' Create 5 spaces in the move buffer
CONTOURMODE(0) = _ctmCONTOUR_ON ' Turn contouring on
MOVER(0) = 10                            ' Load a relative move of 10 units
MOVER(0) = 10                            ' Load a relative move of 10 units
CONTOURMODE(0) = _ctmCONTOUR_OFF         ' Turn contouring off
MOVER(0) = 10                            ' Load a relative move of 10 units
CONTOURMODE(0) = _ctmCONTOUR_ON ' Turn contouring on
MOVER(0) = 10                            ' Load a relative move of 10 units
MOVER(0) = 10                            ' Load a relative move of 10 units
GO(0)                                    ' Trigger motion
```



When a move is loaded, the state of contouring controls whether the end of that move will be a stop point. In the example above, the third move was loaded with contouring off, making the end of that move a stop point.

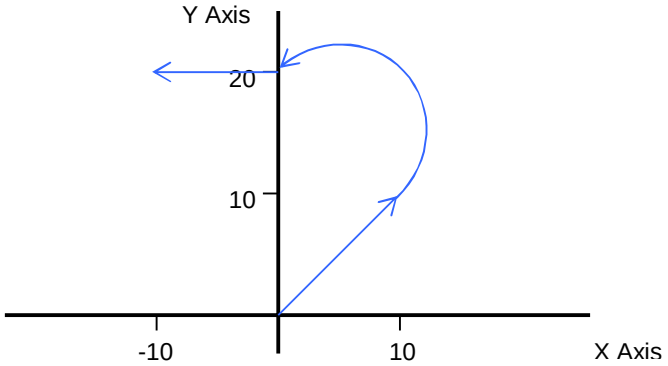
When interpolated moves are loaded with contouring turned on, the path speed is kept constant across multiple vectors. This means that the axis components of a move may not be continuous in velocity. There will be a 'clunk' at vector boundaries. To avoid jarring the machine, the angle between vectors should be kept small. Mint can automatically look out for sharp angle changes using inter-vector angle control.
The inter-vector angle is defined as the angle between two vectors or arcs.



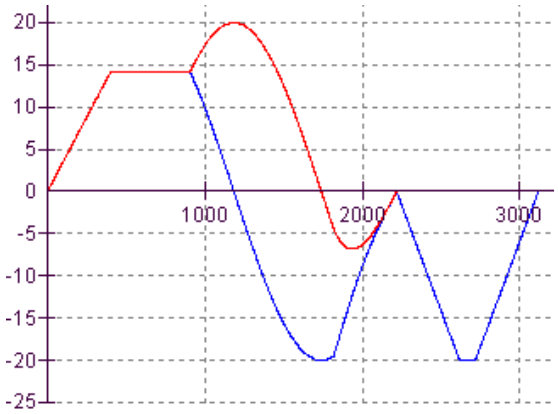
If the angle between two vectors exceeds the threshold set with CONTOURPARAMETER(axis,0), then a stop point will be created, otherwise the axes will continue to give a constant path speed.

Example for NextMove ESB-2:

```
MOVEBUFFERSIZE([0,1]) = 5;                          ' Create 5 spaces in the move buffer
CONTOURMODE([0,1]) = _ctmCONTOUR_ON + _ctmVECTOR_ANGLE_ON;
                                                    ' Turn contouring on with inter-vector angle control
CONTOURPARAMETER([0,1],0) = 10;                     ' Set threshold to 10 degrees.
VECTORR(0,1) = 10, 10                               ' Load a relative move of 10 units
CIRCLER(0,1) = -5, 5, 180                                   ' Load a relative move of 10 units
VECTORR(0,1) = -10, 0                               ' Load a relative move of 10 units
GO(0)                                               ' Trigger motion
```



The axes will automatically stop between the arc and the second vector since the angle between these two vectors exceeds the threshold of 10 degrees. The velocity traces of X and Y are shown below:



When performing a series of contoured moves, it is important to maintain as much distance as possible in the move buffer. If the move distance remaining in the move buffer ever drops to less than the distance required to decelerate the axis to a stop, then the axes will start to decelerate. Once the deceleration has started, the axes will come to a stop before starting again, even if more moves are loaded into the buffer.

The SUSPEND keyword can be used to pause motion in the move buffer as well as to single step through individual moves in the move buffer.

Feedrate control (NextMove, MotiFlex e180 and MicroFlex e190)

As moves are being executed from the move buffer, if any of the motion profile parameters are changed (SPEED/FEEDRATE, ACCEL, DECEL, ACCELJERK and DECELJERK), the change will have an immediate effect on the move in progress. Feedrate control allows these profile parameters to be loaded into the move buffer with each move. For example, in a routing application, some moves may be required to move at a different speed. The following example for NextMove ESB-2 moves around a square with rounded corners. The corner sections are performed at a slower speed than the straights:

```
MOVEBUFFERSIZE([0,1]) = 10;                    ' Create 10 spaces in the move buffer
CONTOURMODE([0,1]) = _ctmCONTOUR_ON + _ctmVECTOR_ANGLE_ON;
                                               ' Turn contouring on with inter-vector angle control
CONTOURPARAMETER([0,1],0) = 10;                ' Set threshold to 10 degrees.
FEEDRATEMODE([0,1]) = _frFEEDRATE;             ' Store feedrate in the move buffer
FEEDRATE([0,1]) = 20;                          ' Set the feedrate to 20 units/s
VECTORA(0,1) = 20, 0
FEEDRATE([0,1]) = 10;                          ' Set the feedrate to 10 units/s
CIRCLEA(0,1) = 20, 5, 90
FEEDRATE([0,1]) = 20;                          ' Set the feedrate to 20 units/s
VECTORA(0,1) = 25, 25
FEEDRATE([0,1]) = 10;                          ' Set the feedrate to 10 units/s
CIRCLEA[0,1] = 20, 25, 90
FEEDRATE([0,1]) = 20;                          ' Set the feedrate to 20 units/s
VECTORA(0,1) = 5, 30
FEEDRATE([0,1]) = 10;                          ' Set the feedrate to 10 units/s
CIRCLEA(0,1) = 5, 25, 90
FEEDRATE([0,1]) = 20;                          ' Set the feedrate to 20 units/s
VECTORA(0,1) = 0, 5
FEEDRATE([0,1]) = 10;                          ' Set the feedrate to 10 units/s
CIRCLEA(0,1) = 5, 5, 90
GO(0,1)                                        ' Trigger motion
```

In this example, FEEDRATEMODE is set to _frFEEDRATE which means 'store feedrate in the move buffer'. Other bits in the FEEDRATEMODE keyword allow ACCEL and DECEL and ACCELJERK and DECELJERK to be stored in the move buffer.

The FEEDRATEMODE keyword also allows some moves to be loaded as 'fast traverse' moves. These moves will respond to the FEEDRATEOVERRIDE keyword which allows the speed of fast traverse moves to be modified, even though the feedrate of that move was specified when the move was loaded.

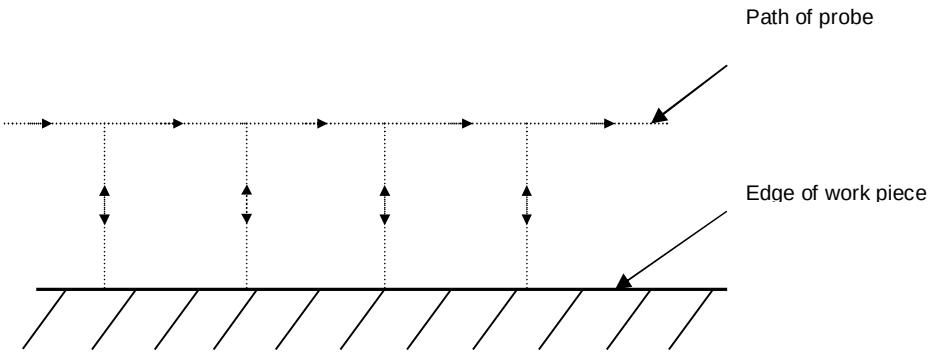
In this example, the first and third moves will be affected by the value of FEEDRATEOVERRIDE but the second move will not.

```
MOVEBUFFERSIZE([0,1]) = 5;                ' Create 5 spaces in the move buffer
CONTOURMODE([0,1]) = _ctmCONTOUR_ON + _ctmVECTOR_ANGLE_ON;
                                          ' Turn contouring on with inter-vector angle control
FEEDRATEMODE([0,1]) = _frFEEDRATE + _frOVERRIDEENABLE;
                                          ' Store feedrate in the move buffer, fast traverse
FEEDRATE([0,1]) = 20;                     ' Set the feedrate to 20 units/s
VECTORR(0,1) = 10, 10
FEEDRATEMODE([0,1]) = _frFEEDRATE;        ' Store feedrate in the move buffer
FEEDRATE([0,1]) = 10;                     ' Set the feedrate to 10 units/s
VECTORA(0,1) = 10, 10
FEEDRATEMODE([0,1]) = _frFEEDRATE + _frOVERRIDEENABLE;
                                          ' Store feedrate in the move buffer, fast traverse
FEEDRATE([0,1]) = 20;                     ' Set the feedrate to 20 units/s
VECTORA(0,1) = 10, 10
FEEDRATEOVERRIDE([0,1]) = 150;                  ' Set override to 150%
GO(0,1)                                   ' Trigger motion
```

The first and third moves will execute at a speed of 30 units/s.

Move blending (NextMove ESB-2 only)

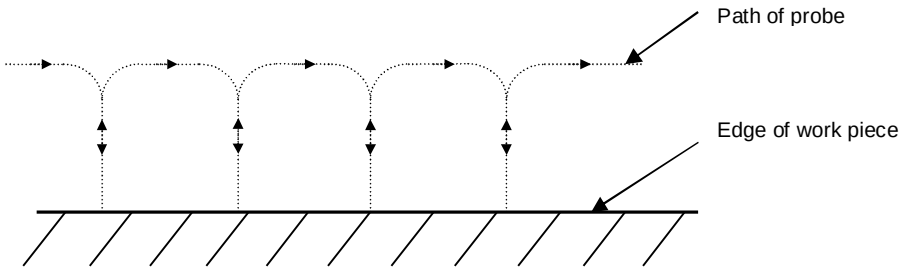
Move blending allows the execution of two moves to be overlapped, so reducing the time needed to execute the moves and produce a smoother tool path. For example, in a tapping application, the drill must be lowered into the work piece, then raised and moved along to the next drill location. The motion would look like:
Slow probe path:




```
VECTORR(0,1) = 0, 10     ' Move drill down
GO(0,1)                  ' Trigger motion
Pause IDLE([0,1])        ' Wait for axes to be idle
OUTX(0) = 1              ' Turn on Output 0 to start the tap
Wait = 250               ' Wait for 250ms
OUTX(0) = 0              ' Turn off Output 0
VECTORR(0,1) = 0, -10    ' Move drill up
VECTORR(0,1) = 10, 0     ' Move drill along
VECTORR(0,1) = 0, 10     ' Move drill down
GO(0,1)                  ' Trigger drill movement
Pause IDLE([0,1])        ' Wait for axes to be idle
OUTX(0) = 1              ' Turn on Output 0 to start the tap
Wait = 250               ' Wait for 250ms
```

```
OUTX(0) = 0              ' Turn off Output 0
VECTORR(0,1) = 0, -10    ' Move drill up
GO(0,1)                  ' Trigger motion
```

Ideal probe path:



To save motion time, the drill could start moving to the next point as it completes the up stroke. This can be achieved by enabling blending. In this example, blending is set to take place automatically at a fixed distance from the end of the move. This is done with the BLENDMODE and BLENDDISTANCE keywords. Blending can be started at any time, causing an immediate overlap of two moves with the BLEND keyword.

```
BLENDMODE([0,1]) = _bmAUTOMATIC;   ' Enable automatic blending
BLENDDISTANCE([0,1])= 2;                ' Start blending 2 units from end of vector
VECTORR(0,1) = 0, 10               ' Move drill down
GO(0,1)                            ' Trigger motion
Pause IDLE([0,1])                  ' Wait for axes to be idle
OUTX(0) = 1                        ' Turn on Output 0 to start the tap
Wait = 250                         ' Wait for 250ms
OUTX(0) = 0                        ' Turn off Output 0
VECTORR(0,1) = 0, -10              ' Move drill up
VECTORR(0,1) = 10, 0               ' Move drill along
VECTORR(0,1) = 0, 10               ' Move drill down
GO(0,1)                            ' Trigger drill movement
Pause IDLE([0,1])                  ' Wait for axes to be idle
OUTX(0) = 1                        ' Turn on Output 0 to start the tap
Wait = 250                         ' Wait for 250ms
OUTX(0) = 0                        ' Turn off Output 0
VECTORR(0,1) = 0, -10              ' Move drill up
GO(0,1)                            ' Trigger motion
```

Output moves

It is sometimes useful to be able to change the state of digital outputs when a move starts to execute. Mint allows digital output moves to be loaded into the move buffer so that they are executed synchronously with the motion. For example:

```
MOVEBUFFERSIZE(0) = 10          ' Create 10 spaces in the move buffer
MOVER(0) = 10                   ' Relative move of 10 units
MOVEOUT(0,0) = 7                ' Turn on output channels 0, 1 and 2 on bank 0
MOVER(0) = 10                   ' Relative move of 10 units
MOVEOUTX(0,1) = 0               ' Turn off output channel 1
MOVER(0) = 10                   ' Relative move of 10 units
MOVEPULSEOUTX(0,3) = 100        ' Turn on output channel 3 for 100ms
MOVER(0) = 10                   ' Relative move of 10 units
GO(0)                           ' Trigger motion
```

At the end of the first move, digital outputs 0, 1 and 2 on output bank 0 are activated and all other outputs on bank 0 are deactivated. At the end of the second move, digital output channel 1 is deactivated. At the end of the third move, digital output channel 3 is activated and a timer started. The fourth move is then started. After 100ms, digital output channel 3 is deactivated.

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