



Using Motion Commands and the Command Queue

A tutorial covering the use of Motion Commands and the Command Queue on Motion enabled E-Nodes.

Contents

Companion Documentation.....	3
Overview.....	3
General use of Motion Commands and the Command Queue.....	3
Motion Commands.....	3
am() - absolute move.....	3
rm() - relative move.....	4
vm() - velocity mode.....	4
tm() - torque mode.....	5
pm() - position mode.....	5
gr() - gear lock mode.....	6
cam() - electronic cam.....	8
va() - vector absolute.....	9
vr() - vector relative.....	9
Motion Command Variables.....	10
address.....	10
node_type.....	10
status.....	10
activity.....	10
mode.....	11
demand.....	11
ferr.....	11
vel.....	11
accel.....	11
decel.....	11
ratchet.....	11
offset.....	11
prop_gain.....	11
diff_gain.....	11
integ_gain.....	11
enc0_vel.....	11
torque.....	11
enc0.....	12
enc1.....	12
Move Synchronisation.....	12
Questions.....	12

Companion Documentation

Full technical detail on the Motion Commands and the Motion Command Queue can be found in the document, 'Control-C Language xpxx.pdf'.

Overview

Control-C provides a range of commands for generating and controlling motion on motion enabled E-Nodes. The motion can be on many different types of physical device from simple rotary motors to linear actuators or hydraulics. These commands are detailed in the document 'Control-C Language xpxx.pdf' but in this document they are collected together to deal specifically with the subject of motion control.

General use of Motion Commands and the Command Queue

When a running program encounters a motion command it first determines on which node the command should be run and then sends the command to the 'Motion Command Queue' on that node. Motion commands are queued in this way to prevent the main program from being held up whilst one motion command waits for a previous one to complete its move.

If a motion command is sent to a node that is not capable of controlling a motor the command will be discarded and not placed in the queue.

The size of motion commands varies but generally about 20 commands can be held in the command queue. If the command queue overflows an error will result.

Queued motion commands are executed in the order they are received. When a command is executed from the queue the queue length is reduced by one and reported in the variable 'queue'.

When a motion command reaches the head of the queue and another motion command is currently running the action taken will vary depending on the types of the commands. For example if both commands are absolute move types ('am()') the second command will wait for the first command to fully complete before it executes. However there are types that will take control over the current command, stop it running and then immediately execute itself. Please refer to the detailed descriptions of the individual motion commands for more information.

- ▶ *Motion commands can be prevented from automatically loading from the command queue by clearing the status 'AUTOLOAD' bit. i.e. status[AUTOLOAD]=0.*
- ▶ *Motion commands can be removed from the head of the queue by altering the 'queue' variable. Please refer to the 'queue' variable description for further information.*

Motion Commands

Currently the range of motion commands and their functions are as follows:-

am() - absolute move

Generates one or more absolute position moves on the addressed nodes. When executed the one or more absolute positions are sent to the motion command queues of their respective nodes. They are then performed in the order determined by their queue position. If the programmer wishes for the moves to take place simultaneously they must ensure the queues are empty before issuing the command.

'am' moves are velocity profiled and obey 'accel', 'vel' and 'decel' values on the addressed nodes.

Example:

```
am(1000:1,1000:2); // node 1 and 2 move to absolute 1000
```

- ▶ During execution the 'mode' variable will report 'ABS'
- ▶ When in absolute move mode changing the 'demand' variable will immediately change the target position but changing the 'offset' will have no effect.
- ▶ 'am' is a queued command. When a 'am' command reaches the head of the queue it can take one of several actions. In all cases when the command has loaded it will automatically run if 'status[AUTORUN]' is set. Otherwise it will wait for the start command 'status[START_MOVE]=1'. The cases are:-
 1. If no motion command is currently running it will automatically load.
 2. If a 'am', 'rm', 'va', 'vr' or 'cam' command is currently running it will wait for the command to complete and then load.
 3. if a 'vm', 'tm', 'pm' or 'gr' command is currently running it will stop the command and then load.
- ▶ NOTE: If status[AUTOLOAD] is not set no commands are loaded from the command queue.

rm() - relative move

Generates one or more relative position moves on the addressed nodes. Same as 'am()' in all other respects.

- ▶ During execution the 'mode' variable will report 'REL'.
- ▶ When in relative move mode changing the 'demand' variable will immediately change the target position and change the 'offset' to the correct value relative to the start position. Changing the 'offset' will change the 'demand' position to reflect the new offset value.
- ▶ 'rm' is a queued command. When a 'rm' command reaches the head of the queue it can take one of several actions. In all cases when the command has loaded it will automatically run if 'status[AUTORUN]' is set. Otherwise it will wait for the start command 'status[START_MOVE]=1'. The cases are:-
 1. If no motion command is currently running it will automatically load.
 2. If a 'am', 'rm', 'va', 'vr' or 'cam' command is currently running it will wait for the command to complete and then load.
 3. if a 'vm', 'tm', 'pm' or 'gr' command is currently running it will stop the command and then load.
- ▶ NOTE: If status[AUTOLOAD] is not set no commands are loaded from the command queue.

vm() - velocity mode

Enters velocity controlled move mode on one or more addressed nodes. When executed the one or more velocity commands are sent to the motion command queues of their respective nodes. They are then performed in the order determined by their queue position. If the programmer wishes for the moves to take place simultaneously they must ensure the queues are empty before issuing the command.

'vm' moves are velocity profiled and obey 'accel', 'vel' and 'decel' values.

Example:

```
vm(20:1,30:2) // set velmode on node 1 & 2
```

- ▶ During execution the 'mode' variable will report 'VEL'.
- ▶ When in velocity mode the maximum velocity may be varied directly by changing the 'vel' variable and the 'demand' variable is updated with the latest absolute position.

► 'vm' is a queued command. When a 'vm' command reaches the head of the queue it can take one of several actions. In all cases when the command has loaded it will automatically run if 'status[AUTORUN]' is set. Otherwise it will wait for the start command 'status[START_MOVE]=1'. The cases are:-

1. If no motion command is currently running it will automatically load.
2. If a 'am', 'rm', 'va', 'vr' or 'cam' command is currently running it will wait for the command to complete and then load.
3. if a 'tm', 'pm' or 'gr' command is currently running it will stop the command and then load.

► NOTE: If status[AUTOLOAD] is not set no commands are loaded from the command queue.

tm() - torque mode

Enters torque controlled move mode on one or more addressed nodes. When executed the one or more torque commands are sent to the motion command queues of their respective nodes. They are then performed in the order determined by their queue position. If the programmer wishes for the moves to take place simultaneously they must ensure the queues are empty before issuing the command.

Torque mode assumes the node is controlling a motor/amplifier set up for torque mode operation. In torque mode the torque value given is translated into a voltage and output on the DAC0 analogue drive output.

torque value	DAC0 output
100	10v
0	0v
-100	-10v

Example:

```
tm(20:1,80:2) // set torque mode on node 1 & 2. 2 and 8 volts
               // respectively
```

► During execution the 'mode' variable will report 'TOR'.

► When in torque mode the demand position is updated with the latest positional value from 'enc0'.

► 'tm' is a queued command. When a 'tm' command reaches the head of the queue it can take one of several actions. In all cases when the command has loaded it will automatically run if 'status[AUTORUN]' is set. Otherwise it will wait for the start command 'status[START_MOVE]=1'. The cases are:-

1. If no motion command is currently running it will automatically load.
2. If a 'am', 'rm', 'va', 'vr' or 'cam' command is currently running it will wait for the command to complete and then load.
3. if a 'vm', 'pm' or 'gr' command is currently running it will stop the command and then load.

► NOTE: If status[AUTOLOAD] is not set no commands are loaded from the command queue.

pm() - position mode

Enters position controlled move mode on one or more addressed nodes. When executed the one or more position commands are sent to the motion command queues of their respective nodes. They are then performed in the order determined by their queue position. If the programmer wishes for the moves to take place simultaneously they must ensure the queues are empty before issuing the command.

Position controlled moves are simple point to point moves. There is no velocity profiling and within the limits of the system, maximum power will be used for acceleration and deceleration whilst moving to the new absolute position. The maximum velocity set by the 'vel' value is observed and the move may also be limited by ratcheting if enabled.

Example:

```
pm(2000:1,8000:2) // immediate move to 2000 on node 1
                  // and 8000 on node 2
```

▶ During execution the 'mode' variable will report 'POSIT'.

▶ 'pm' is a queued command. When a 'pm' command reaches the head of the queue it can take one of several actions. In all cases when the command has loaded it will automatically run if 'status[AUTORUN]' is set. Otherwise it will wait for the start command 'status[START_MOVE]=1'. The cases are:-

1. If no motion command is currently running it will automatically load.
2. If 'pm' or 'gr' command is currently running it will immediately load.
3. If a 'am', 'rm', 'va', 'vr' or 'cam' command is currently running it will wait for the command to complete and then load.
4. If a 'vm' or 'tm', command is currently running it will stop the command and then load.

▶ NOTE: If status[AUTOLOAD] is not set no commands are loaded from the command queue.

gr() - gear lock mode

Enters electronic gear locked move mode on the addressed node. When executed the gear lock command is sent to the motion command queues of the addressed node. It will then be performed in the order determined by its queue position.

When the gear lock command is executed from the queue the node will control the position of its associated motor and encoder input (enc0) to that of encoder input 1 (enc1) multiplied by the given gear ration where:

```
gear_ratio      =   master_teeth / slave_teeth
demand position =   enc1 * gear_ratio
```

It may be described as a slave shaft (enc0), geared to a master shaft (enc1).

For gear locking to work the node must be set up to receive regular updates for the value of the master encoder (enc1). This is accomplished by sharing the master encoder value over the E-net using 'broadcast variables'. The node reading the master encoder issues the following command:

```
tx_bcv0 = varnum(enc0); // issued by the master node
```

This causes the master encoder value to be broadcast every 'bcv_interval'. The controlling node issues the following command to receive the broadcasts:

```
rx_bcv0 = varnum(enc1); // issued by the slave node
```

This causes the controlling node to place the broadcast variable value 0 into the variable 'enc1'.

Gear locking can be engaged in several different modes as specified by the 'Engagement mode' parameter. Variables 'mgpos', master gear position and 'sgpos', slave gear position are used for some of these. 'mgpos' and 'sgpos' are simple variables that must be initialised (if required) by the programmer prior to the execution of gear lock mode .

Mode	Operation
0	Normal gear locking takes place immediately using the current 'enc0' and 'enc1' positions.

- 1 Normal gear locking takes place when 'enc1' becomes GREATER THAN OR EQUAL TO 'mgpos'.
- 2 Normal gear locking takes place when 'enc1' becomes LESS THAN OR EQUAL TO 'mgpos'.
- 3 Registered gear locking takes place immediately. 'mgpos' and 'sgpos' are used for the initial engagement positions providing exact registration.
- 4 Registered gear locking takes place immediately after 'enc1 becomes GREATER THAN OR EQUAL TO 'mgpos'. 'mgpos and 'sgpos are then used for the initial engagement positions to provide exact registration
- 5 Registered gear locking takes place immediately after 'enc1 becomes LESS THAN OR EQUAL TO 'mgpos'. 'mgpos and 'sgpos are then used for the initial engagement positions to provide exact registration

During gear locked operation both 'master_teeth' and 'slave_teeth' can be varied at will to change the active gear ratio. The slave direction may also be changed by making either 'master_teeth' or 'slave_teeth' negative. Finally if either 'master_teeth' or 'slave_teeth' are set to zero the slave axis will halt (dis-engage) at the position the change was made. Thus once in gear lock mode the programmer can vary the gear ratio, change direction, dis-engage and re-engage the drive at will.

► *Note: For smooth operation of REGISTERED locking it is recommended that the programmer ensures that both the master and slave are very close to the positions in 'mgpos' and 'sgpos' before engagement. If this is not the case the slave will jump to the correct position, having calculated that position from 'mgpos', 'sgpos', 'enc0', 'enc1' and the gear ratio.*

► *Note: Gear locking uses position mode for controlling the motor position. Position controlled moves are simple point to point moves. There is no velocity profiling and within the limits of the system, maximum power will be used for acceleration and deceleration whilst moving to the new absolute position. A maximum velocity set by the 'vel' variable is observed and the move may also be limited by ratcheting if enabled.*

► *Note: If a 'pm' command is issued during a gear locked move gear locking will be exited immediately and the slave will move directly to the demanded position using position mode. This feature may be used to exit gear lock mode and stop the motor at an exact position.*

► *During execution the 'mode' variable will report 'GEAR' unless dis-engaged when it will report 'BRK' for 'braked'.*

► *'gr' is a queued command. When a 'gr' command reaches the head of the queue it can take one of several actions. In all cases when the command has loaded it will automatically run if 'status[AUTORUN]' is set. Otherwise it will wait for the start command 'status[START_MOVE]=1'. The cases are:-*

1. *If no motion command is currently running it will automatically load.*
2. *If 'pm' or 'gr' command is currently running it will immediately load.*
3. *If a 'am', 'rm', 'va', 'vr' or 'cam' command is currently running it will wait for the command to complete and then load.*
4. *if a 'vm' or 'tm', command is currently running it will stop the command and then load.*

► *NOTE: If status[AUTOLOAD] is not set no commands are loaded from the command queue.*

Example:

Registered gear locking when 'enc1' >= 'mgpos'

```
mgpos = 1000;
sgpos = 2000;
gr(100,200,4); // ratio = 0.5, reg mode 4, default node address
```

cam() - electronic cam

This command provides an electronic cam profile motion command. It is used in conjunction with a pre-defined global array of demand positions. When the command is executed it defines the start and finish positions in the array and the number of control loop periods between each increment through the array.

- ▶ During execution the 'mode' variable will report 'CAM'.
- ▶ 'cam' will only work on nodes with motion control capability and is ignored by those that don't.
- ▶ 'cam' is a motion command and will be stored in the motion command queue to await execution in the correct order.
- ▶ The array used for 'cam' MUST be global and MUST reside on the node that executes the command. The declared address for the array will automatically determine the address to which the command will be sent for execution.
- ▶ 'cam' is a queued command. When a 'cam' command reaches the head of the queue it can take one of several actions. In all cases when the command has loaded it will automatically run if 'status[AUTORUN]' is set. Otherwise it will wait for the start command 'status[START_MOVE]=1'. The cases are:-
 5. If no motion command is currently running it will automatically load.
 6. If an 'am', 'rm', 'va', 'vr' or 'cam' command is currently running it will wait for the command to complete and then load.
 7. If an 'vm', 'tm', 'pm' or 'gr' command is currently running it will stop the command and then load.
- ▶ NOTE: If status[AUTOLOAD] is not set no commands are loaded from the command queue.

When 'cam' is first executed the node will position the motor at the position given by '[start]' and then increment through the array until it reaches '[finish]'. For each increment it will wait the given number of control loop cycles. If this number is greater than 1 it will perform linear interpolation between the current and next position in the array.

'cam' uses 'position mode' to control the motor. This is a point to point control method with no velocity profiling. Individual increments are velocity limited to the value given in 'vel' and subject to ratcheting if enabled. Please refer to the description of 'pm' mode for more detail.

The following example shows how to generate a sinusoidal cam profile in two ways. The first does not use linear interpolation. The second uses linear interpolation to reduce the array size. With the second method the sinusoid will actually be made up from a series of sort straight moves and the programmer must determine if this degradation in smoothness is worth the reduction in array size. In many cases the inertia of the motor and load effectively smooth the profile to an acceptable level.

Example:

```
var a[1000];

main()
{
  var i;
  for(i=0;i<1000;i++) a[i] = 200 * sin(i*0.36);
  cam(a[0],a[999],1); // without linear interpolation
  for(i=0;i<100;i++) a[i] = 200 * sin(i*3.6);
  cam(a[0],a[99],10); // with linear interpolation
}
```

va() - vector absolute

Generates a vector absolute move along 'x' and 'y' axes. A vector move is one where the two associated axes start and finish the move at the same time and preserve the 'x' axis maximum velocity and accelerations along the vector path.

Example:

```
va(1000:1,1000:2); // node 1 and 2 move to absolute 1000
```

- ▶ During execution the 'mode' variable will report 'VA'.
- .
- ▶ Vector moves can be 'linked'. If linking is on, only the first and last vectors in a series of vectors use acceleration and deceleration respectively. The inner vectors remain at maximum velocity and thus are seamlessly linked.
- ▶ Motion commands are ignored on nodes without motion capability. For vector commands both the 'x' and 'y' addresses MUST be motion capable. If both are not the command is simply ignored. If only one is motion capable then the move sequence will stop whilst one waits infinitely for the other.
- ▶ 'va' is a queued command. When issued, the 'va' command will enter the motion command queues on the nominated node addresses. When the 'x' axis command reaches the head of its queue it will automatically wait for the 'y' axis command to reach the head of its queue. At this point the 'x' axis command can take one of several actions. In all cases when the command has loaded it will automatically run if 'status[AUTORUN]' is set. Otherwise it will wait for the start command 'status[START_MOVE]=1'. The cases are:-
 1. If no motion command is currently running it will automatically load.
 2. If a 'am', 'rm', 'va', 'vr' or 'cam' command is currently running it will wait for the command to complete and then load.
 3. if a 'vm', 'tm', 'pm' or 'gr' command is currently running it will stop the command and then load.
- ▶ NOTE: If status[AUTOLOAD] is not set no commands are loaded from the command queue.

vr() - vector relative

Generates a vector relative move along 'x' and 'y' axes.

- ▶ During execution the 'mode' variable will report 'VR'.

Same as 'va()' in all other respects.

Motion Command Variables

A number of pre-defined variables are associated with motion commands as follows:-

address

All motion commands are referenced to specific node addresses.

node_type

Motion commands are only valid on motion enabled node types.

status

The 'status' variable is an array of bits setting or reporting the status of various conditions. The bit locations with direct relevance to motion commands are:-

<code>status [START_MOVE] =0;</code>	current move is stopped
<code>status [START_MOVE] =1;</code>	current move is started (if autorun == 0)
<code>status [ENABLE_DAC0] =0;</code>	DAC0 output is disabled
<code>status [ENABLE_DAC0] =1;</code>	DAC0 output is enabled
<code>status [MARK] =0;</code>	clear encoder mark pulse seen flag
<code>status [MARK] ==1</code>	true if encoder mark pulse seen
<code>status [CAPTURED] =0;</code>	clear encoder position captured flag
<code>status [CAPTURED] ==1</code>	true if encoder position captured
<code>status [CAP_POL] =0;</code>	encoder capture input polarity = falling
<code>status [CAP_POL] =1;</code>	encoder capture input polarity = rising
<code>status [AUTOLOAD] =0;</code>	motion commands do not auto load from command queue
<code>status [AUTOLOAD] =1;</code>	motion commands auto load from command queue
<code>status [AUTORUN] =0;</code>	motion commands do not automatically run when loaded
<code>status [AUTORUN] =1;</code>	motion commands automatically run when loaded
<code>status [HALT_MOVE] =0;</code>	current motion command is not halted
<code>status [HALT_MOVE] =1;</code>	current motion command is halted
<code>status [LINK] =0;</code>	motion commands are not linked
<code>status [LINK] =1;</code>	motion commands are linked
<code>status [ZERO_ENC] =1;</code>	zero encoder value
<code>status [MARK_POL] =0;</code>	encoder mark polarity = falling
<code>status [MARK_POL] =1;</code>	encoder mark polarity = rising
<code>status [INVERT_ENC0] =0;</code>	encoder count not inverted
<code>status [INVERT_ENC0] =1;</code>	encoder count inverted
<code>status [INVERT_DAC0] =0;</code>	Dac0 output signal not inverted
<code>status [INVERT_DAC0] =1;</code>	Dac0 output signal inverted

activity

Reports the current activity of motion commands. There are a number of defined BUILTIN constants for use with testing the 'activity' variable.

mode

Reports the current mode of motion commands. There are a number of defined BUILTIN constants for use with testing the 'mode' variable.

demand

The target demand position for the internal move profile generator. For some commands this variable can be changed directly (see 'am()' above);

ferr

The reported following error in encoder counts. This is the difference between the output from the profile generator and the actual motor position.

vel

The maximum velocity for any motion command.

accel

The maximum acceleration for any profiled motion command.

decel

The maximum deceleration for any profiled motion command.

ratchet

This variable controls the ratchet operation for 'position', 'cam' and 'gear' mode motion commands

```
ratchet=0;  ratchet disabled  
ratchet=1;  +ve movement allowed  
ratchet=2;  -ve movement allowed
```

offset

The position offset from the demand position for relative moves.

prop_gain

The proportional gain for the PID control loop. Motion commands only.

diff_gain

The differential gain for the PID control loop. Motion commands only.

integ_gain

The intergral gain for the PID control loop. Motion commands only.

enc0_vel

The velocity of encoder0

torque

The torque demand for torque mode commands.

enc0

The reading for encoder0

enc1

The reading for encoder1. This value must be provided. It is usually seeded automatically with a broadcast master encoder value.

The status of a motion command currently being executed is reported in real time

Move Synchronisation

A node can potentially receive motion commands from any programmed node on the network. These are queued and executed in the order of receipt. It is the programmers responsibility to ensure the correct synchronisation of multiple moves across multiple node. This is best achieved by knowing the current states of the nodes concerned and the precise timing of motion command issue.

It is worth noting that motion commands can run automatically when 'status[AUTORUN]==1' but if set to '0' they will not auto run. In this case when a motion command is loaded from the queue it will simply wait for the start command, 'status[START_MOVE]=1'.

Here are two simple ways to ensure move synchronisation:

1. Ensure the queues are empty on all concerned nodes. Issue a multiple move command with 'AUTORUN' set on all nodes. The E-nodes are very fast and will start all the moves within 1 or 2 milliseconds.
2. Ensure the queues are empty on all concerned nodes. Clear 'AUTORUN' on all concerned nodes. Issue the multiple move command. Read all nodes 'modes' to check they have loaded and are ready to run. Issue multiple 'status[START_MOVE]=1' commands.

► *Some motion commands have automatic synchronisation. Please refer to the detailed description of motion commands for further information.*

Questions

If you have any questions regarding either this tutorial or any other aspect of using the E-Node range please contact the staff at Etrol ltd.

Tele: (+44) 01730 816893
email: sales@etrol.co.uk