

NAME

move_info() – returns what would happen on a move

DESCRIPTION

The `move_info()` function returns information about what would happen on a subsequent `move_all` command given the current motor positions and current values in the `A[]` array. Such a function might be called, for example, within the `user_premove` macro to determine which motors will be moved to allow extra limit checking can be performed involving the relative position of motors.

If called with no arguments, returns a two-dimensional associative array containing the move information. The array is indexed by motor number and information keyword. The keywords are

- "to_move" – nonzero if motor will move
- "error" – reason the motor will not move
- "commanded" – the commanded position
- "magnitude" – magnitude of the move in user units
- "current" – current position in user units
- "current_dial" – current position in dial units
- "target" – target position in user units
- "target_dial" – target position in dial units
- "backlash" – backlash for this move in user units
- "leftover" – reminder due to motor resolution

If called with a single argument that is one of the above keywords, the function returns a one-dimensional associative array indexed by motor number containing values for that keyword for each motor. If called with a motor number or mnemonic as a single argument, the function returns a one-dimensional associative array, indexed by the above keywords containing values for the one motor. If called with two arguments, motor number and keyword, the function returns the corresponding single value.

No matter how the function is called, the internal code will calculate values for all motors. Thus, if multiple values are needed, it is most efficient and recommended to call the function once selecting arguments that will return all the needed values.

The "to_move" element will be 1 or 0, indicating whether the motor would move or not. If there is condition that prevents the move, the "error" element will contain one of these strings:

- "low limit" – move exceeds low limit
- "high limit" – move exceeds high limit
- "busy" – motor is busy
- "read only" – motor is configured as read only
- "protected" – motor configuration does not allow moving
- "disabled" – motor has been disabled
- "externally disabled" – shared motor has been disabled
- "unusable" – motor did not respond to presence test

Otherwise, there will be no "error" array element.

The "target" and "target_dial" values are the final position after backlash. The "magnitude" value contains the distance to the target position and does not include the magnitude of the backlash.

The "leftover" value is the fractional value that is the difference between the requested position and the achievable position given the finite resolution of the motor. For example, if a motor has 1000 steps per degree, each step corresponds to 0.001 degrees. If one asks to move

the motor to a position of 1.0004 degrees, the motor will move to 1 degree and the leftover value will be 0.0004 degrees.

The "commanded" value is the target position in user units to the full precision requested. The other position-related values are rounded to the motor resolution. The "commanded" position is saved after a move and is available using special arguments to the built-in `read_motors()` functions.

As mentioned above, if multiple values are needed, it is better to make a single call of `move_info()` saving the return values in an array, rather than making multiple calls, as each call involves calculations for all the motor positions and values, even if only selected values are returned. For example,

```
{
  local i, m[]
  m = move_info()
  for (i = 0; i < MOTORS; i++)
    if (m[i]["to_move"]) {
      ...
    }
}
```

For the most part, the `move_info()` function will reflect what will happen on the next `move_all` command. However, for shared motors that can be moved by other processes or for motors that have positions that drift or have jitter, the status and position may change between the `move_info()` call and the `move_all` call.