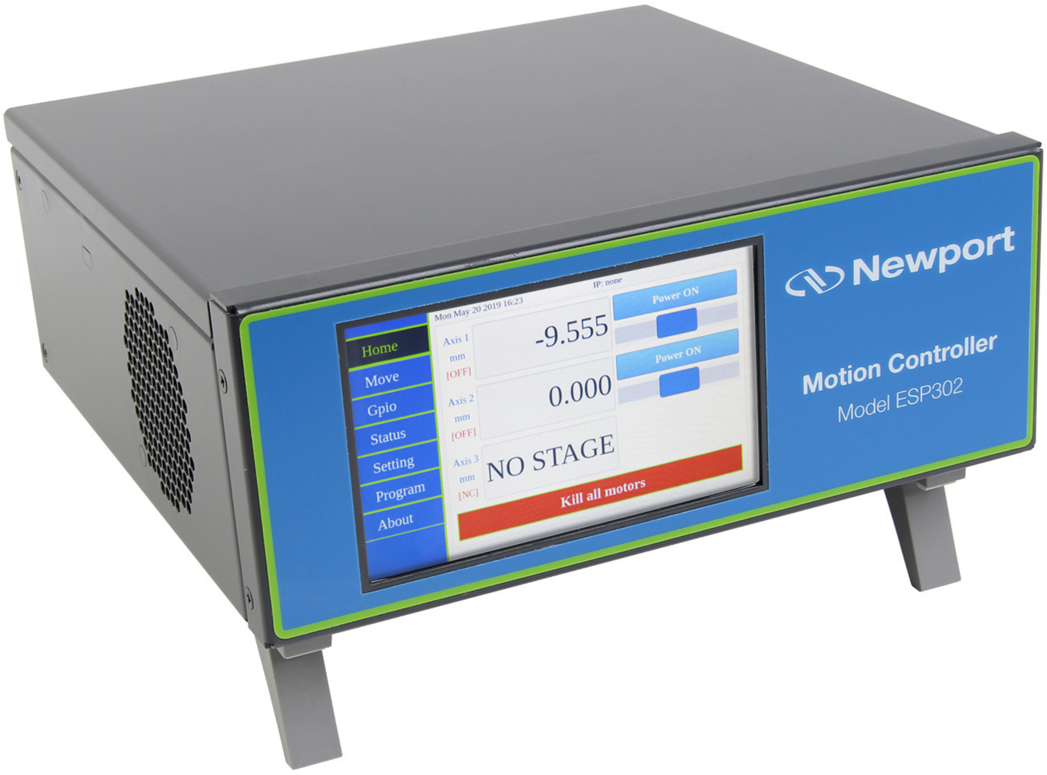
ESP302

Friendly Motion Controller/Driver



ESP COMPATIBLE

Newport®

Features Manual

©2019 by Newport Corporation, Irvine, CA. All rights reserved.

Original instructions.

No part of this document may be reproduced or copied without the prior written approval of Newport Corporation. This document is provided for information only, and product specifications are subject to change without notice. Any change will be reflected in future publishings.

Table of Contents

Friendly Motion Controller/Driver ESP 302 Controller

1.0 Introduction

1.1 Scope of the Manual

The present **ESP302 Features Manual** describes all the functions implemented in the standard controller. Additional **Feature sheets** are available for special customer functions.

2.0 Advanced Capabilities

2.1 Grouping

2.1.1 Introduction – Advanced Capabilities

Coordinated motion of multiple axes is required to produce a desired contour in a multi-dimensional space. For instance, if we want to move from one point to another along a line or along a circle, or a combination of both line and circle, we require coordinated motion of multiple axes. One way to facilitate such coordinated motion is ***"grouping"*** the axes involved in producing the desired motion. This is akin to defining the coordinate system in which the desired contour is being made.

Coordinated motion on a 2-D plane, therefore, requires a group comprised of any two axes. ESP302 trajectories are limited to a 2-D plane.

The procedure for defining a group and all the group parameters required for making coordinated motion is described in Section 2.1.3 the commands that actually make the coordinated motion. The procedure for making "long" moves or contours that involve a combination of circular and linear moves is described in Section 2.1.4. Miscellaneous grouping commands are discussed in Section 2.1.5.

2.1.2 Defining a Group and Group Parameters

This subsection discusses the method for defining a group and all the group parameters.

2.1.2.1 Creating a Group

The ASCII command used to create a new group is **HN.** For instance, the command **1HN2, 3** assigns axis numbers 2 and 3 to group number 1. One such group must be defined first before those axes can be moved in a coordinated fashion. A group can comprise of axes anywhere from one to three.

If a group has only one axis assigned to it, a linear motion of the group is similar to moving that axis from one point to another. Circular motion of a group with only one axis cannot be made.

If a group has more than two axes assigned to it, circular motion of the group is made using the first two axes in the group.

The order in which axes are assigned to a group is very important. This is because it specifies the frame of reference in which coordinated motion of axes takes place. For instance, the command **1HN2, 3** assigns axis numbers 2 and 3 to group number 1, where axis #2 is equivalent to X-axis and axis #3 is equivalent to Y-axis in a traditional Cartesian coordinate system. Reversing the order of axes (E.G., **1HN3, 2**) reverses the axis assignment.

A few rules that are in place for easy management of group are as follows:

- An axis cannot be a member of different groups at the same time.
- An axis cannot be assigned more than once in a group.
- A group has to be deleted before axes assigned to it can be changed.
- An axis assigned to a group cannot be moved individually using commands such as **PA** and **PR.** Use group linear move commands instead.

Refer to the description of this command in the *commands section* (See the Remote Mode chapter in the Programmer’s Manual) for correct syntax, parameter ranges, etc.

2.1.2.2 Defining Group Parameters

Group parameters such as velocity, acceleration, deceleration, jerk, and e-stop deceleration must be defined for every group following the creation of that group. These parameters are used to produce the desired coordinated motion of the group. They override any original values specified for individual axes. The axes' original values are restored when the group to which they have been assigned is deleted. Refer to the description of **HV, HA, HD, HJ,** and **HE** commands in the *commands section* (See the Remote Mode chapter in the Programmer's Manual) for correct syntax, parameter ranges, etc.

2.1.3 Making Linear and Circular Moves

This subsection discusses the method for making linear and circular moves of groups. While coordinated motion of axes with different motor types and different encoder resolutions is supported, it is assumed that all axes have the same units of measure.

2.1.3.1 Making a Linear Move

Once a group has been defined and all group parameters have been specified, the ASCII command **HL** is used to move the group from an initial position to a final position along the line. The current position of axes is the initial position of linear move. The desired final position is specified along with this command.

This command makes all axes assigned to the group move with predefined group (tangential) velocity, acceleration and deceleration along a line.

The linear move is a true linear interpolation, meaning:

$$(y - y_0) = m(x - x_0)$$
$$m = \frac{(y_f - y_0)}{(x_f - x_0)}$$

where: X_0 and Y_0 represent initial position of the group.

X_f and Y_f represent desired final position of the group.

2.1.3.2 Making a Circular Move

Once a group has been defined and all group parameters have been specified, the ASCII command **HC** can be used to move the group from an initial position to a final position along a circle. The current position of axes is the initial position of circular move. The final position of move is calculated based on the desired center of circle and sweep angle specified along with this command. All sweep angles are measured in degrees. The sign of angles follow the trigonometric convention: positive angles are measured counterclockwise.

This command makes all axes assigned to the group move with predefined group (tangential) velocity, acceleration and deceleration along a circle.

The circular move is a true arc of a circle, meaning:

$$r = \sqrt{(x_0 - x_c)^2 + (y_0 - y_c)^2}$$
$$\theta = a\tan 2\left(\frac{(y_0 - y_c)}{(x_0 - x_c)}\right)$$
$$\theta_0[axis\#2] = \theta$$
$$\theta_0[axis\#3] = \theta - \frac{\pi}{2}$$
$$\theta_f[axis\#2] = \theta_0 + \theta_{cmd}$$
$$\theta_f[axis\#3] = \theta_0 + \theta_{cmd}$$
$$x_f = r * \cos(\theta_f[axis\#2]) + x_c$$
$$y_f = r * \cos(\theta_f[axis\#3]) + y_c$$

where: X_0 and Y_0 represent initial position of the group.

X_c and Y_c represent desired center position of the circular move.

X_f and Y_f represent calculated final position of the group.

R is radius of the circle.

θ_0 is the base initial angle of an axis.

θ_f is the final angle of an axis, which is dependant on the sweep angle, θ_{cmd}

Both **HL** and **HC** can initiate the desired motion if they are received while the group is holding position. On the other hand, if they are received while a group move is in progress, the new commands get queued into a "via-point" buffer. The queued commands are executed on a FIFO basis when the move already in progress has reached its destination. The group does not come to a stop at the end of last move. Instead, there will be s smooth transition to the new move command, just as if it were one compound move (combination of multiple moves). The next section details the procedure for making contours or "long" moves using "via-point" buffers.

Refer to the description of **HL** and **HC** commands in the *commands section* (See the Remote Mode chapter in the Programmer's Manual) for correct syntax, parameter ranges, etc.

2.1.4 Making Contours

This subsection discusses the method for making contours. Contouring is the process of making complex trajectories or "long" moves that may involve linear and circular move segments.

These move segments can be sequenced in any order. Arcs can be followed by arcs or lines, and lines by arcs or other lines as shown in the following figures. Since there is no pre-processing of move segments involved in making a contour, the user must ensure that there is no change in tangential velocity at the transition from one move to another. If this constraint is not satisfied, the transition from one move segment to another may cause excessive accelerations and shocks that could damage the stages.

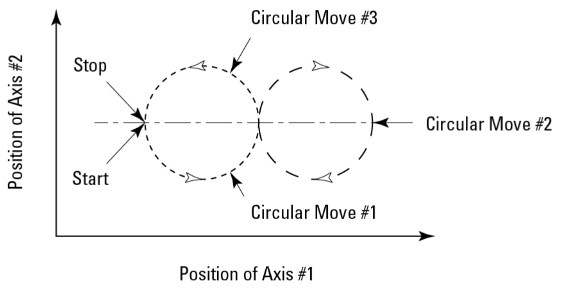


Figure 1: A contour with multiple circular moves.

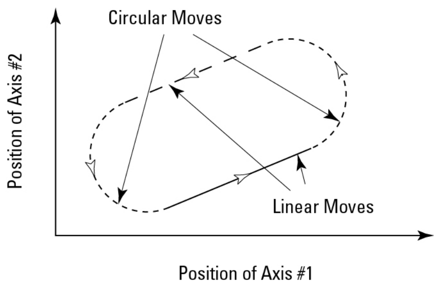


Figure 2: A contour with multiple linear and circular moves.

In order to store the multiple move segment commands needed to make a contour, we make use of a "via point" buffer. This "via point" buffer contains group move commands essential to make a new move segment upon completion of the move segment currently in progress. The new move commands are pulled out of the buffer on a FIFO basis. The "via point" buffer can hold a maximum of 100 group move commands. If more than 100 group move commands are issued by a user, the excess commands are flow-controlled by the firmware.

This mechanism will block the portal through which the commands were issued until all the commands issued have been executed.

It is, therefore, recommended that the user take advantage of ASCII command, **HQ** which tells the number of commands that can be put in the "via point" buffer at any given time. This allows a user to control the flow of commands manually, while ensuring the availability of that portal for other commands such as **HP, TP,** etc.

The trajectory generator checks if the "via point" buffer has a new target position (i.e., any new move segments pending?) while the current move is in progress. If "via point" buffer is empty, the group comes to a stop upon completion of current move segment. Otherwise, it begins a new move segment without stopping after completing the current move. The group transitions from current move segment to a new move segment smoothly if the tangential velocity at the transition is ensured to be constant.

The ASCII command **HQ** is used to query the available "via point" buffer space. The commands **HL** and **HC** are used to queue linear move or circular move commands into the "via point" buffer. Refer to the description of these commands in the *commands section* (See the Remote Mode chapter in the Programmer's Manual) for correct syntax, parameter ranges, etc.

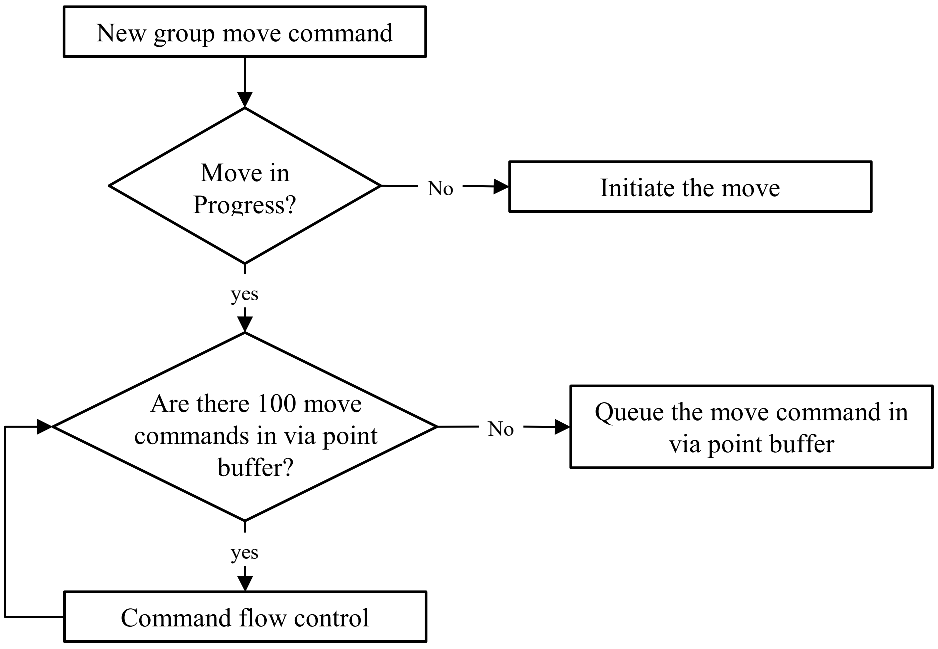

New group move command
Move in Progress?
No
Initiate the move
yes
Are there 100 move commands in via point buffer?
No
Queue the move command in via point buffer
yes
Command flow control

Figure 3: Block Diagram of Via Point Data Handling by Command Processor.

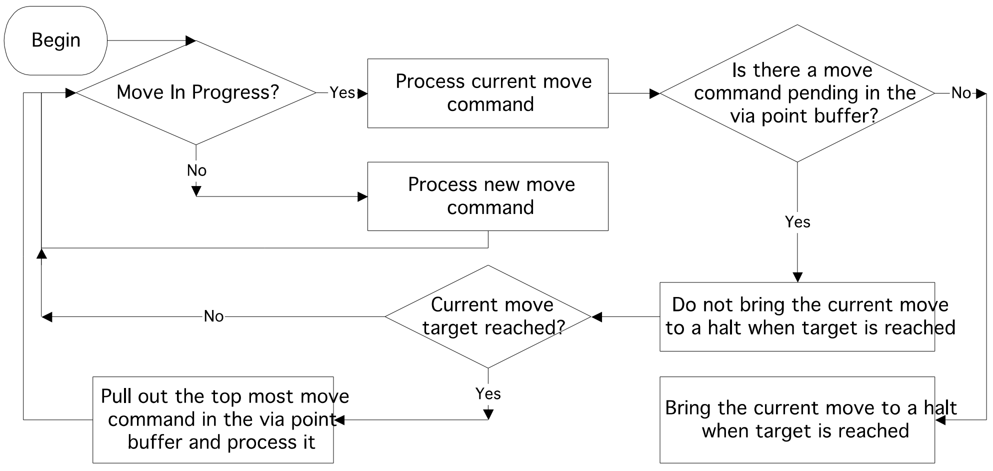

Begin
Move In Progress?
Yes
Process current move command
Is there a move command pending in the via point buffer?
No
No
Process new move command
Yes
Current move target reached?
No
Do not bring the current move to a halt when target is reached
Yes
Pull out the top most move command in the via point buffer and process it
Bring the current move to a halt when target is reached

Figure 4: Block Diagram of Via Point Data Handling by Trajectory Generator.

2.1.5 Miscellaneous Commands

The following commands are available to operate on a group of axes simultaneously:

- **HO** and **HF:** These commands are used to turn ON and turn OFF the power to all axes in a group respectively. The axes assigned to a group can be powered ON or OFF individually using **MO** and **MF** commands also. A group is considered to be ON if all axes assigned to that group are ON.
- **HP:** This command is used to read the actual position of all axes in a group.
- **HS:** This command is used to stop the group motion.
- **HW:** This command is used to wait for the group motion to stop and a user settable delay period thereafter.
- **HX:** This command is used to delete a group.
- **HZ:** This command is used to read the size or the number of axes assigned to group.

2.2 Slaving a Stage to Joystick or a Different Stage

2.2.1 Introduction – Slaving a Stage

The ESP302 allows to define a master slave mode of 2 axis. The trajectory of the slave is the desired trajectory or actual position of the master scaled by reduction ratio.

Refer to the SS and GR command to configure the master slave mode.

2.2.2 Slave to a Different Stage

The following steps may be taken for a slave axis to follow master's position. This mode may be chosen exclusively when absolute or relative move commands can be issued to the master.

Steps	Move Command	Action by Move Command
1. Define master-slave relationship.	**2SS1**	Axis #2 is the slave of axis #1.
2. Defines master-slave reduction ratio.	**2GR0.5**	Master's position is scaled by 0.5 to obtain slave's position.
3. Issue move commands to master axis.	**1PA10** **1PR10**	Move master to absolute 10 units. Move master by relative 10 units.

Table 1: Slave to a Different Stage Steps.

2.2.3 Slave to a Joystick

If the slave axis is required to jog based on a DIO bit status (such as through joystick), follow these steps:

Steps	Move Command	Action by Move Command
1. Assign DIO bits for jogging slave axis.	**2BP0, 1**	Jog axis #2 in negative direction if DIO bit #0 is low. Jog axis #2 in positive direction if DIO bit #1 is low.
2. Enable DIO bits for jog mode.	**2BQ1**	
3. Change DIO bit value physically.		

Table 2: Slave to a Joystick Steps.

Refer to the description of the ASCII commands (See the Remote Mode chapter in the Programmer's Manual), for additional description, correct syntax, parameter ranges, etc.

2.3 Synchronize Motion to External and Internal Events

2.3.1 Introduction – Synchronize Motion

Certain applications require the use of inputs from an external source to command the motion controller to perform certain tasks. These tasks can be to either initiate motion of desired axes (written in a user's stored program) or to inhibit motion of desired axes or, more simply, to just monitor the motion status of these axes. The ESP302 motion controller addresses these issues by taking advantage of the digital I/O interface available on the controller.

The 16 digital I/O bits are divided into two (2) ports: A and B.

Port A covers DIO bits 0 – 7, port B covers bits 8 – 15.

The direction of each port can be setup to be either input or output. If a port is configured to be an input, the DIO bits that belong to that port can only report the state – HIGH or LOW logic level – of the corresponding DIO hardware. On the other hand, if the port is configured to be an output, the DIO bits in that port can be used to either set or clear the state of the corresponding hardware. Each DIO bit has a pull-up resistor to +5V. As a result, all bits will be at HIGH logic if not connected to external circuit and configured as input. Furthermore, the direction of all the ports is set to input by default following a controller reset.

The next section details the way in which these DIO bits can be used to initiate the motion of desired axes through stored programs. The subsequent sections outline the way to inhibit the motion of desired axes and to monitor the motion status of these axes using DIO bits.

2.3.2 Using DIO to Execute Stored Programs

ESP 302 controller can synchronize the initiation of any motion profile to external events. In order to accomplish this task, users must write their desired motion profile as a stored program and assign this stored program to a desired DIO bit.

The direction of the DIO port bit belongs to must then be set to "input" in order for the controller to detect the external event. Once these preliminaries are completed, the controller will execute the user specified stored program whenever it detects a change in the state – HIGH to LOW logic level – of the corresponding DIO hardware. Please review the examples below for further clarifications.

Example 1:

```
1EP                     | Define stored program 1
1MO; 2MO                | Turn axes 1,2 ON
1PA0; 2PA0              | Move axes 1,2 to absolute 0 units
1WS100; 2WS100          | Wait for axes 1,2 motion to complete
QP                      | End of program
0BG1                    | Assign DIO #0 to run stored program 1
BO 02H                  | 02H = (0010)Binary
                        | Set DIO ports A to input and port B to output
                        | i.e., set bits 0 – 7 to input and 8 – 15 to output
```

After the above commands are sent to the controller, the controller will execute the stored program 1 when DIO bit #0 changes its state from HIGH to LOW logic level.

Example 2:

```
2EP                     | Define stored program 2
1MO; 2MO                | Turn axes 1,2 ON
1PA0; 2PA0              | Move axes 1,2 to absolute 0 units
1WS100; 2WS100          | Wait for axes 1,2 motion to complete
1DL                     | Define a label 1
1PR2; 2PR2              | Move axes 1,2 by relative 2 units
1WS100; 2WS100          | Wait for axes 1,2 motion to complete
1PR-2; 2PR-2            | Move axes 1,2 by relative –2 units
1WS100; 2WS100          | Wait for axes 1,2 motion to complete
1JL10                   | Jump to label 1 10 times
QP                      | End of program
1BG2                    | Assign DIO #1 to run stored program 2
BO 02H                  | 02H = (0010)Binary
                        | Set DIO ports A to input and port B to output
                        | i.e., set bits 0 – 7 to input and 8 – 15 to output
```

After the above commands are sent to the controller, the controller will execute stored program 2 when DIO bit #1 changes its state from HIGH to LOW logic level.

2.3.3 Using DIO to Inhibit Motion

The ESP302 motion controller can inhibit the motion of any axis in response to external events. In order to accomplish this task, users must define the DIO bit to be employed to inhibit the motion of a desired axis and the logic state in which that bit should be in order to inhibit motion. Once this done, the feature has to be enabled. Furthermore, the direction of the DIO port this DIO bit belongs to must be set to "input" in order for the controller to detect the external event.

At this point, if the selected axis is already in motion, and DIO bit is asserted, E-stop is executed per E-stop configuration (Refer "**ZE**" command for further details). If the axis is not moving, any new move commands are refused as long as the DIO bit is asserted. In either case, "AXIS-XX DIGITAL I/O INTERLOCK DETECTED" error is generated, where XX is the axis whose motion is inhibited through DIO. Please review the example below for further clarifications.

Example 3:

```
2BK1,1                  | Use DIO bit #1 to inhibit motion of axis #2. This DIO bit
                        | should be HIGH when axis #2 motion is inhibited
2BL1                    | Enable inhibition of motion using DIO bits for axis #2
BO 02H                  | 02H = (0010)Binary
                        | Set DIO ports A to input and port B to output
```

```
                                  | i.e., set bits 0 – 7 to input and 8 – 15 to output
```

After the above commands are sent to the controller, the controller will inhibit the motion of axis #2 when DIO bit is at a HIGH logical level, and generate appropriate error message.

2.3.4 Using DIO to Monitor Motion Status

User's applications can monitor motion status – desired axis is in motion or standstill – through ESP motion controller's DIO. This status bit can in turn be used to drive external processes such as turning on/off a mechanical brake, for instance. In order to accomplish this task, users must define the DIO bit to be employed to monitor the motion status of a desired axis and the logic state in which that bit should be in when the axis is not in motion. Once this is done, the feature has to be enabled. Furthermore, the direction of the DIO port this DIO bit belongs to must be set to "output" in order for the controller to report the motion status.

At this point, if the selected axis is not in motion, the DIO bit changes its state to the level specified as described earlier. Please review the example below for further clarifications.

Example 4:

```
2BM9,1          | Use DIO bit #9 to indicate motion status of axis #2. This
                DIO
                | bit will be set to HIGH when axis #2 is not in motion
2BN1            | Enable notification of motion status using DIO for axis #2
BO 02H          | 02H = (0010)Binary
                | Set DIO ports A to input and port B to output
                | i.e., set bits 0 – 7 to input and 8 – 15 to output
```

After the above commands are sent to the controller, the controller will set DIO bit #9 to a HIGH logical level when axis #2 is not in motion.

Commands related to utilizing DIO for initiating/inhibiting motion of desired axis and notifying motion status of these axes are listed in the table below (See the Remote Mode chapter in the Programmer’s Manual, for additional details):

Command	Description
BG	Assign DIO bits to execute stored programs
BK	Assign DIO bits to inhibit motion
BL	Enable DIO bits to inhibit motion
BM	Assign DIO bits to notify motion status
BN	Enable DIO bits to notify motion status
BO	Set DIO port A, B direction

Table 3: Commands to Synchronize Motion to External Events.

3.0 Motion Control Tutorial

3.1 Motion Systems

A schematic of a typical motion control system is shown in **Figure 5.1**

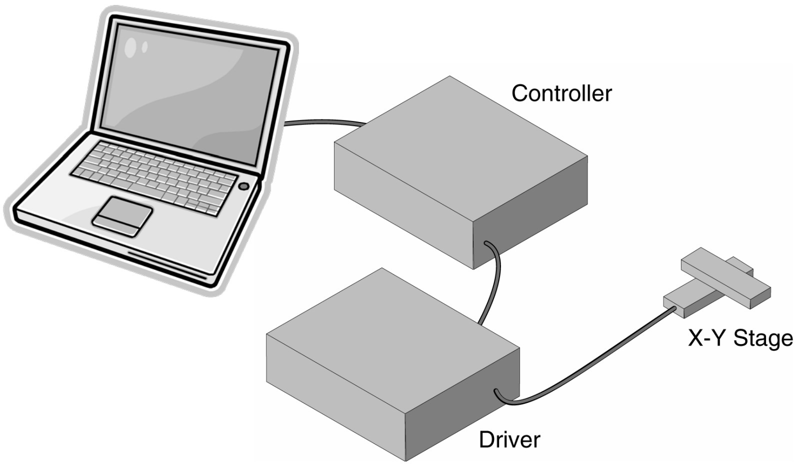


Figure 5: Typical Motion Control Systems

Its major components are:

Controller

An electronic device that receives motion commands from an operator directly or via a computer, verifies the real motion device position and generates the necessary control signals.

Driver

An electronic device that converts the control signals to the correct format and power needed to drive the motors.

Motion Device

An electro-mechanical device that can move a load with the necessary specifications.

Cables

Needed to interconnect the other motion control components.

If the user is like most motion control users, they started by selecting a motion device that matches certain specifications needed for an application. Next, the user should choose a controller that can satisfy the motion characteristics required. The changes are that the user is less interested in how the components look or what their individual specs are, but want to be sure that together they perform reliably according to their needs.

We mentioned this to make a point. A component is only as good as the system lets (or helps) it to be.

For this reason, when discussing a particular system performance specification, we will also mention which components affect performance the most and, if appropriate, which components improve it.

3.2 Specification Definitions

People mean different things when referring to the same parameter name. To establish some common ground for motion control terminology, here are some general guidelines for the interpretation of motion control terms and specifications.

- As mentioned earlier, most motion control performance specifications should be considered system specifications.
- When not otherwise specified, all error-related specifications refer to the position error.

- The servo loop feedback is position-based. All other velocity, acceleration, error, etc. parameters are derived from the position feedback and the internal clock.
- To measure the absolute position, we need a reference, a measuring device, which is significantly more accurate than the device tested. In our case, dealing with fractions of microns (0.1 µm and less), even a standard laser interferometer becomes unsatisfactory.
- For this reason, all factory measurements are made using a number of high precision interferometers, connected to a computerized test station.
- To avoid unnecessary confusion and to easily understand and troubleshoot a problem, special attention must be paid to avoid bundling discrete errors in one general term. Depending on the application, some discrete errors are not significant. Grouping them in one general parameter will only complicate the understanding of the system performance in certain applications.

3.2.1 Following Error

The Following Error is not a specifications parameter but, because it is at the heart of the servo algorithm calculations and of other parameter definitions, it deserves our attention.

As will be described later in Section 3.3: Control Loops, a major part of the servo controller's task is to make sure that the actual motion device follows as close as possible an ideal trajectory in time. The user can imagine having an imaginary (ideal) motion device that executes exactly the motion profile they are requesting. In reality, the real motion device will find itself deviating from this ideal trajectory. Since most of the time the real motion device is trailing the ideal one, the instantaneous error is called Following Error.

To summarize, the Following Error is the instantaneous difference between the actual position as reported by the position feedback device and the ideal position, as seen by the controller. A negative following error means that the load is trailing the ideal motion device.

3.2.2 Error

Error has the same definition as the Following Error with the exception that the ideal trajectory is not compared to the position feedback device (encoder) but to an external precision-measuring device.

In other words, the Following Error is the instantaneous error perceived by the controller while the Error is the one perceived by the user.

3.2.3 Accuracy

The accuracy of a system is probably the most common parameter users want to know. Unfortunately, due to its perceived simplicity, it is also the easiest to misinterpret.

The Accuracy is a static measure of a point-to-point positioning error. Starting from a reference point, the user should command the controller to move a certain distance. When the motion is completed, the user should measure the actual distance traveled with an external precision-measuring device. The difference (the Error) represents the positioning Accuracy for that particular motion.

Because every application is different, the user needs to know the errors for all possible motions. Since this is practically impossible, an acceptable compromise is to perform the following test.

Starting from one end of travel, the user can make small incremental moves and at every stop, the user should record the position Error. The user performs this operation for the entire nominal travel range. When finished, the Error data is plotted on a graph similar to Figure 6**.**

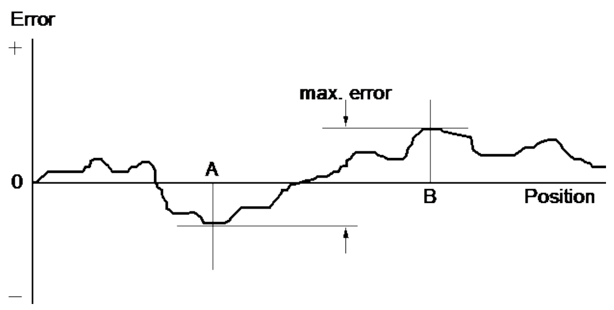


Figure 6: Position Error Test.

The difference between the highest and the lowest points on the graph is the maximum possible Error that the motion device can have. This worst-case number is reported as the positioning Accuracy. It guaranties the user that for any application, the positioning error will not be greater than this value.

3.2.4 Local Accuracy

For some applications, it is important to know not just the positioning Accuracy over the entire travel but also over a small distance. To illustrate this case, Figure 7 and Figure 8 shows two extreme cases.

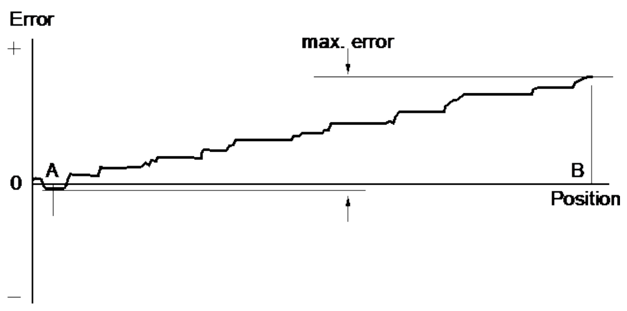


Figure 7: High Accuracy for Small Motions.

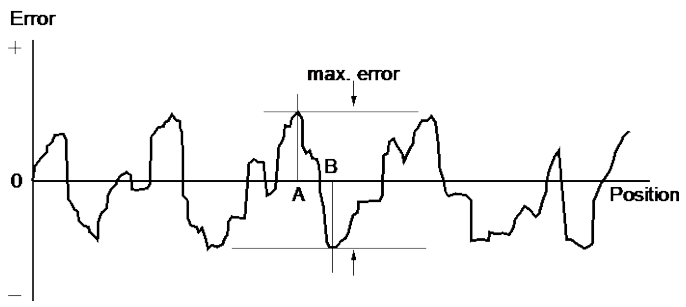


Figure 8: Low Accuracy for Small Motions.

Both error plots from Figure 7 and Figure 8 have a similar maximum Error. But, if the user compares the maximum Error for small distances, the system in Figure 8 shows significantly larger values.

For applications requiring high accuracy for small motions, the system in Figure 7 is definitely preferred.

"Local Error" is a relative term that depends on the application; usually no Local Error value is given with the system specifications. The user should study the error plot supplied with the motion device and determine the approximate maximum Local Error for the specific application.

3.2.5 Resolution

Resolution is the smallest motion that the controller attempts to make. For all DC motor and most all standard stepper motor driven stages supported by the ESP302, this is also the resolution of the encoder.

Keeping in mind that the servo loop is a digital loop, the Resolution can be also viewed as the smallest position increment that the controller can handle.

3.2.6 Minimum Incremental Motion

The Minimum Incremental Motion is the smallest motion that a device can reliably make, measured with an external precision-measuring device. The controller can, for instance, execute a motion equal to the Resolution (one encoder count) but in reality, the load may not move at all. The cause for this is in the mechanics.

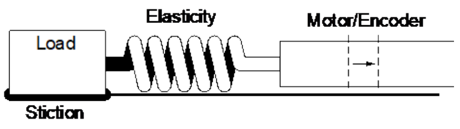


Figure 9: Effect of Stiction and Elasticity on Small Motions.

Figure 9 shows how excessive stiction and elasticity between the encoder and the load can cause the motion device to deviate from ideal motion when executing small motions.

The effect of these two factors has a random nature. Sometimes, for a small motion step of the motor, the load may not move at all. Other times, the accumulated energy in the spring will cause the load to jump a larger distance. The error plot will be similar to Figure 10.

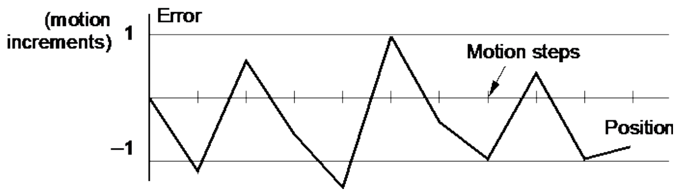


Figure 10: Error Plot.

Once the Maximum Incremental Motion is defined, the next task is to quantify it. This more difficult for two reasons: one is its random nature and the other is in defining what a *completed motion* represents.

Assume that the user has a motion device with a 1 µm resolution. If every time the user commands a 1 µm motion the measured error is never greater than 2%, the user will probably be very satisfied and declare that the Minimum Incremental Motion is better than 1 µm.

If, on the other hand, the measured motion is sometimes as small as 0.1 µm (a 90% error), the user could not say that 1 µm is a reliable motion step. The difficulty is in drawing the line between acceptable and unacceptable errors when performing a small motion step. The most common value for the maximum acceptable error for small motions is 20%, but each application ultimately has its own standard.

One way to solve the problem is to take a large number of measurements (a few hundred at minimum) for each motion step size and present them in a format that an operator can use to determine the Minimum Incremental Motion by its own standard.

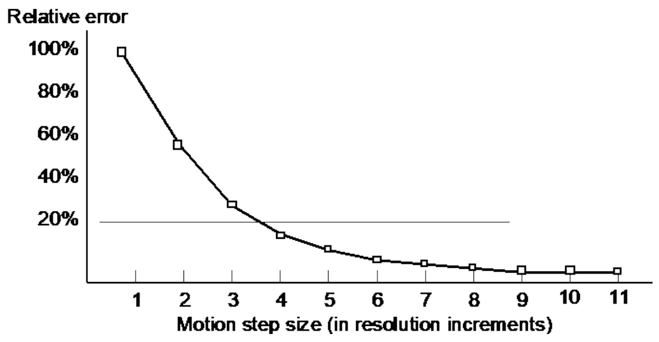


Figure 11: Error vs. Motion Step Size.

Figure 11 shows an example of such a plot. The graph represents the maximum relative error for different motion step sizes. In this example, the Minimum Incremental Motion that can be reliably performed with a maximum of 20% error is one equivalent to 4 resolution (encoder) increments.

3.2.7 Repeatability

Repeatability is the positioning variation when executing the same motion profile. Assuming that the user has a motion sequence that stops at a number of different locations, the Repeatability is the maximum position variation of all targets when the same motion sequence is repeated a large number of times. It is a relative, not absolute, error between identical motions.

3.2.8 Backlash (Hysteresis)

For all practical purposes, Hysteresis and Backlash have the same meaning for typical motion control systems; the error caused by approaching a point from a different direction. The difference is that Hysteresis refers to the compliance of the mechanical components, while Backlash represents the "play", or looseness, in the mechanical drive train.

All parameters discussed up to now that involve the positioning Error assumed that all motions were performed in the same direction. If the user tries to measure the positioning error of a certain target (destination), approaching the destination from different directions could make a significant difference.

In generating the plot in Figure 6 we said that the motion device will make a large number of incremental moves, from one end of travel to the other. If the user commands the motion device to move back and stop at the same locations to take a position error measurement, the user would expect to get an identical plot, superimposed on the first one. In reality, the result could be similar to Figure 12.

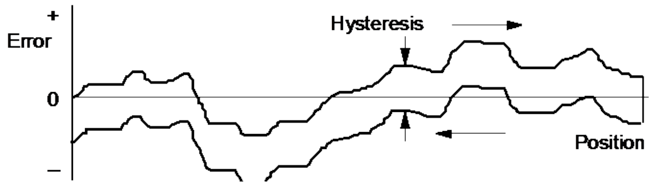


Figure 12: Hysteresis Plot.

The error plot in reverse direction is identical with the first one but seems to be shifted down by a constant error. This constant error is the Hysteresis of the system.

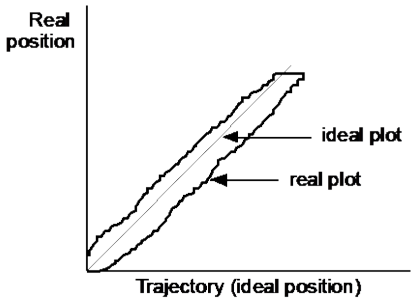


Figure 13: Real vs. Ideal Position.

To justify a little more why we call this Hysteresis, lets do the same graph in a different format (Figure 13). Plotting the real versus the ideal position will give the user a familiar hysteresis shape.

3.2.9 Pitch, Roll and Yaw

These are the most common angular error parameters for linear translation stages. They are pure mechanical errors and represent the rotational error of a stage carriage around the three axes. A perfect stage should not rotate around any of the axes, thus the Pitch, Roll and Yaw should be zero.

The commonly used representation of the three errors is shown in Figure 14. Pitch is rotation around the Y axis, Roll is rotation around the X axis, and Yaw is rotation around the Z axis.

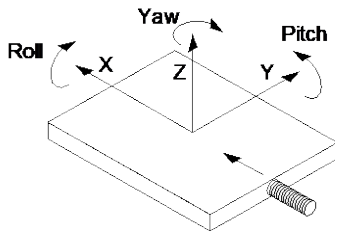


Figure 14: Pitch, Roll and Yaw Motion Axes.

The problem with this definition is that, though correct, it is difficult to remember. A more graphical representation is presented in Figure 15. Imagine a tiny carriage driven by a giant lead screw. When the carriage rolls sideways on the lead screw pitch, we call that Pitch. And, when the carriage deviates left or right from the straight direction (on an imaginary Y trajectory), we call it Yaw.

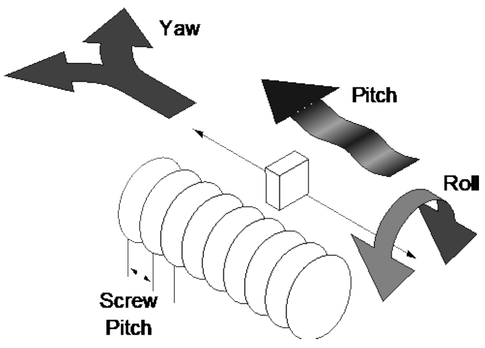


Figure 15: Pitch, Yaw and Roll Motion Axes.

3.2.10 Wobble

This parameter applies only to rotary stages. It represents the deviation of the axis of rotation during motion. A simple form of Wobble is a constant one, where the rotating axis generates a circle (Figure 16).

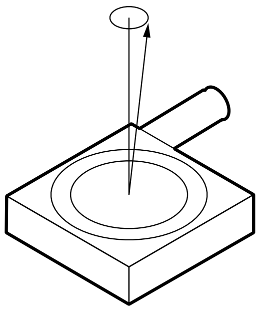


Figure 16: Wobble Generates a Circle.

A real rotary stage may have a more complex Wobble, where the axis of rotation follows a complicated trajectory. This type of error is caused by the imperfections of stage’s bearing way machining and/or ball bearings.

3.2.11 Load Capacity

There are two types of loads that are of interest for motion control applications: static and dynamic loads.

The static Load Capacity represents the amount of load that can be placed on a stage without damaging or excessively deforming it. Determining the Load Capacity of a stage for a particular application is more complicated than it may first appear. The stage orientation and the distance from the load to the carriage play a significant role. For a detailed description on how to calculate the static Load Capacity, please consult the motion control catalog tutorial section.

The dynamic Load Capacity refers to the motor's effort to move the load. The first parameter to determine is how much load the stage can push or pull. In some cases the two values could be different due to internal mechanical construction.

The second type of dynamic Load Capacity refers to the maximum load that the stage could move with the nominal acceleration. This parameter is more difficult to specify because it involves defining an acceptable following error during acceleration.

3.2.12 Maximum Velocity

The Maximum Velocity that could be used in a motion control system is determined by both motion device and driver. Usually it represents a lower value than the motor or driver is capable of. In most cases, including the ESP302, the default Maximum Velocity may be increased. The hardware and firmware are tuned for a particular maximum velocity that cannot be exceeded.

3.2.13 Minimum Velocity

The Minimum Velocity usable with a motion device depends on the motion control system but also on the acceptable velocity regulation. First, the controller sets the slowest rate of motion increments it can make. The encoder resolution determines the motion increment size and then, the application sets a limit on the velocity ripple.

To illustrate this, take the example of a linear stage with a resolution of 0.1 µm. If the user sets the velocity to 0.5 µm/sec, the stage will move 5 encoder counts on one second.

But, a properly tuned servo loop could move the stage 0.1 µm in about 20 ms. The position and velocity plots are illustrated in Figure 17.

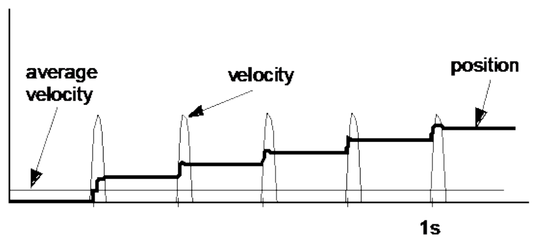


Figure 17: Position, Velocity and Average Velocity.

The average velocity is low but the velocity ripple is very high. Depending on the application, this may be acceptable or not. With increasing velocity, the ripple decreases and the velocity becomes smoother.

This example is truer in the case of a stepper motor driven stage. The typical noise comes from a very fast transition from one step position to another. The velocity ripple in that case is significantly higher.

In the case of a DC motor, adjusting the PID parameters to get a softer response will reduce the velocity ripple but care must be taken not to negatively affect other desirable motion characteristics.

3.2.14 Velocity Regulation

In some applications, for example scanning, it is important for the velocity to be very constant. In reality, there are a number of factors besides the controller that affect velocity.

As described in the Minimum Velocity definition, the speed plays a significant role in the amount of ripple generated, especially at low values. Even if the controller does a perfect job by running with zero following error, imperfections in the mechanics (friction, variation, transmission ripple, etc.) will generate some velocity ripple that can be translated to Velocity Regulation problems.

Depending on the specific application, one motor technology can be preferred over the other.

As far as the controller is concerned, the stepper motor version is the ideal case for a good average Velocity Regulation because the motor inherently follows precisely the desired trajectory. The only problem is the ripple caused by the actual stepping process.

The best a DC motor controller can do is to approach the stepper motor's performance in average Velocity regulation, but it has the advantage of significantly reduced velocity ripple, inherently and through PID tuning. If the DC motor implements a velocity closed loop through the use of a tachometer, the overall servo performance increases and one of the biggest beneficiary is the Velocity Regulation.

3.2.15 Maximum Acceleration

The maximum Acceleration is a complex parameter that depends as much on the motion control system as it does on application requirements. For stepper motors, the main concern is not to lose steps (or synchronization) during the acceleration. Besides the motor and driver performance, the load inertia plays a significant role.

For DC motor systems the situation is different. If the size of the following error is of no concern during the acceleration, high Maximum Acceleration values can be entered. The motion device will move with the highest natural acceleration it can (determined by the motor, driver, load inertia, etc) and the errors will consist of just a temporary larger following error and a velocity overshoot.

In any case, special consideration should be given when setting the acceleration. Through in most cases no harm will be done in setting a high acceleration value, avoid doing so if the application does not require it. The driver, motor, motion device and load undergo maximum stress during high acceleration.

3.2.16 Combined Parameters

Very often a user looks at an application and concludes that they need a certain overall accuracy. This usually means that the user is combining a number of individual terms (error parameters) into a single one. Some of these combined parameters even have their own name, even though not all people mean the same thing by them: Absolute Accuracy, Bi-directional Repeatability, etc. The problem with these generalizations is that, unless the term is well defined and the testing closely simulates the application, the numbers could be of little value.

The best approach is to carefully study the application, extract from the specification sheet the applicable discrete error parameters and combine them (usually add them) to get the worst-case general error applicable to the specific case. This method not only offers a more accurate value but also gives a better understanding of the motion control system performance and helps pinpoint problems.

Also, due to integrated nature of the ESP302 motion controller, many basic errors can be significantly corrected by another component of the loop. Backlash, Accuracy and Velocity Regulation are just a few examples where the controller can improve motion device performance.

3.3 Control Loops

When talking about motion control systems, one of the most important questions is the type of servo loop implemented. The first major distinction is between open and closed loop. Of course, this is of particular interest when driving stepper motors. As far as the DC servo loop, the PID type is by far the most widely used.

The ESP302 implements a PID servo loop with velocity and acceleration feed-forward.

The basic diagram of a servo loop is shown in Figure 18. Besides the command interpreter, the main two parts of a motion controller are the trajectory generator and the servo controller. The first generates the desired trajectory and the second one controls the motor to follow it as closely as possible.

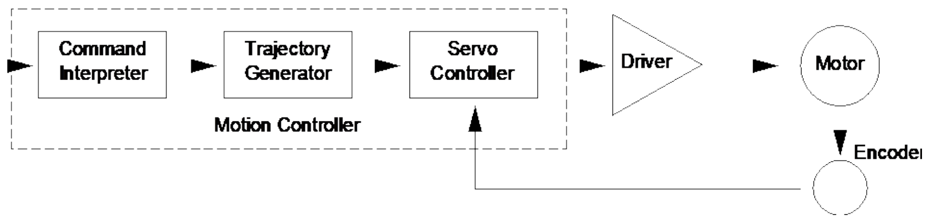


Figure 18: Servo Loop.

3.3.1 PID Servo Loops

The PID term comes from the proportional, integral and derivative gain factors that are at the basis of the control loop calculation. The common equation given for it is:

$$K_p \bullet e + K_i \int e\,dt + K_d \bullet \frac{de}{dt}$$

where: **K_p** = Proportional gain factor

K_I = integral gain factor

K_d = derivative gain factor

e = instantaneous following factor

The program for most users is to get a feeling for this formula, especially when trying to tune the PID loop. Tuning the PID means changing its three gain factors to obtain a certain system response, task quite difficult to achieve without some understanding of its behavior of servo loops.

The following paragraphs explain the PID components and their operation.

P Loop

Let’s start with the simplest type of closed loop, the P (proportional) loop. The diagram in Figure 19 shows its configuration.

Every servo cycle (100 µs), the actual position, as reported by the encoder, is compared to the desired position generated by the trajectory generator. The difference **e** is the positioning error (the following error). Amplifying it (multiplying it by $\mathbf{K_p}$) generates a control signal that, converted to an analog signal, is sent to the motor driver.

There are a few conclusions that could be drawn from studying this circuit:

- The motor control signal, thus the motor voltage, is proportional to the following error.
- There must be a following error in order to drive the motor.
- Higher velocities need higher motor voltages and thus higher following errors.
- At stop, small errors cannot be corrected if they don't generate enough voltage for the motor to overcome friction and stiction.
- Increasing the $\mathbf{K_p}$ gain reduces the necessary following error but too much of it will generate instabilities and oscillations.

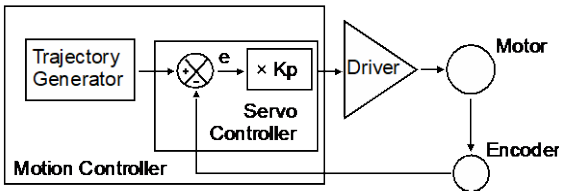


Figure 19: P Loop.

PI Loop

To eliminate the error at stop and during long constant velocity motions (usually called steady-state error), an integral term can be added to the loop. This term integrates (adds) the error every servo cycle (100 µs) and the value, multiplied by the $\mathbf{K_i}$ gain factor, is added to the control signal (Figure 20).

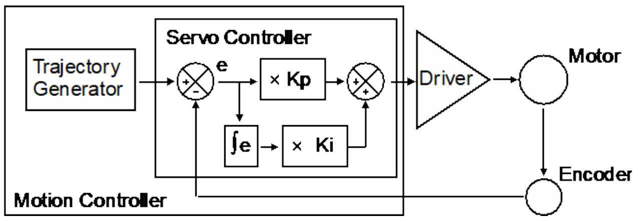


Figure 20: PI Loop.

The result is that the integral term will increase until it drives the motor by itself, reducing the following error to zero. At stop, this has the very desirable effect of driving the positioning error to zero. During a long constant velocity motion it also brings the following error to zero, an important feature for some applications.

Unfortunately, the integral term also has a negative side, a severe de-stabilizing effect on the servo loop. In the real world, a simple PI Loop is usually undesirable.

PID Loop

The third term of the PID Loop is the derivative term. It is defined as the difference between the following error of the current servo cycle (100 µs) and of the previous one. If the following error does not change, the derivative term is zero.

Figure 21 sows the PID servo loop diagram. The derivative term is added to the proportional and integral one. All three process the following error in their own way and, added together, form the control signal.

The derivative term adds a damping effect that prevents oscillations and position overshoot.

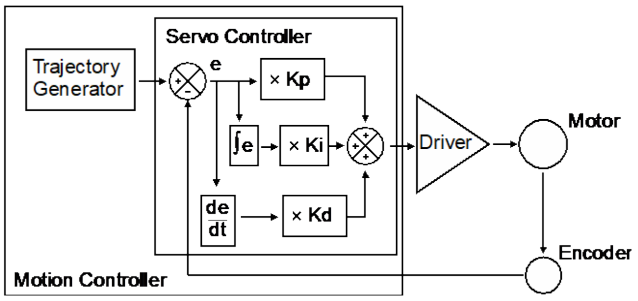


Figure 21: PID Loop.

3.3.2 Feed-Forward Loops

As described in the previous paragraph, the main driving force in a PID loop is the proportional term. The other two correct static and dynamic errors associated with the closed loop.

Taking a closer look at the desired and actual motion parameters and at the characteristics of the DC motors, some interesting observations can be made. For a constant load, the velocity of a DC motor is approximately proportional with the voltage. This means that for a trapezoidal velocity profile, for instance, the motor voltage will have also a trapezoidal shape (Figure 22).

The second observation is that the desired velocity is calculated by the trajectory generator and is known ahead of time. The obvious conclusion is that we could take this velocity information, scale it by $\mathbf{K}_{vff}$ factor and feed it to the motor driver. If the scaling is done properly, the right amount of voltage is sent to the motor to get the desired velocities, without the need for a closed loop.

Because the signal is derived from the velocity profile and it is being sent directly to motor driver, the procedure is called velocity feed-forward.

Of course, this looks like an open loop, and it is (Figure 23). But, adding this signal to the closed loop has the effect of significantly reducing the "work" the PID has to do, thus reducing the overall following error. The PID now has to correct only for the residual error left over by the feed-forward signal.

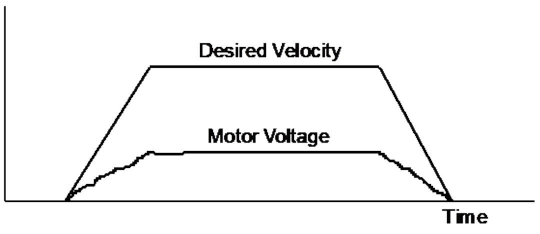


Figure 22: Trapezoidal Velocity Profile.

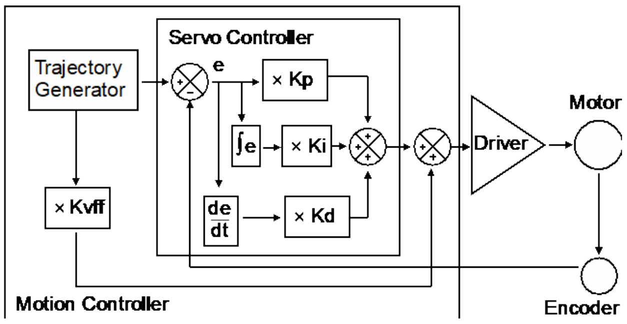


Figure 23: PID Loop with Feed-Forward.

There is another special note that has to be made about the feed-forward method. The velocity is approximately proportional to the voltage and only for constant loads, but this true only if the driver is a simple voltage amplifier or current (torque) driver.

3.4 Motion Profiles

When talking about motion commands we refer to certain strings sent to a motion controller that will initiate a certain action, usually a motion. There are a number of common motion commands that are identified by name. The following paragraphs describe a few of them.

3.4.1 Move

A move is a point-to-point motion. On execution of a move motion command, the motion device moves from the current position to a desired destination. The destination can be specified either as an absolute position or as a relative distance from the current position.

When executing a move command, the motion device will accelerate until the velocity reaches a pre-defined value. Then at the proper time, it will start decelerating so that when the motor stops, the device is at the correct position. The velocity plot has a trapezoidal shape (Figure 24).

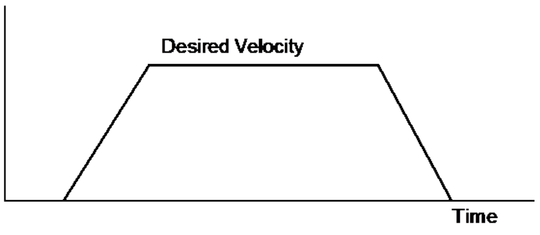


Figure 24: Trapezoidal Motion Profile.

The position and acceleration profiles relative to the velocity are shown in Figure 25.

The position plot of this type of motion will have a S-Curve shape. For this reason, this type of motion is called a S-Curve motion.

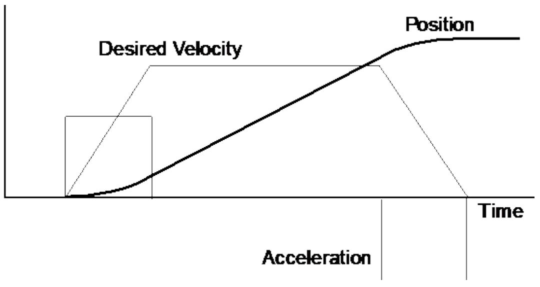


Figure 25: Position and Acceleration Profiles.

Besides the destination, the acceleration and the velocity of the motion (the constant portion of it) can be set by the user before every move command. Advanced controllers like the ESP302 allow the user to change them even during the motion.

3.4.2 Jog

When setting up an application, it is often necessary to move stages manually while observing motion. The easy way to do this without resorting to specialized input devices such as joysticks or track-wheels is to use simple push-button switches. This type of motion is called a jog. When a jog button is pressed the selected axis starts moving with a pre-defined velocity. The motion continues only while the button is pressed and stops immediately after its release.

The ESP302 offers two jog speeds. Both high and low jog speeds are user programmable. The jog acceleration is also the same than the programmed maximum acceleration values.

The jog speed selection is available only through the Front Panel or Website interfaces. The jog through DIO have only one speed.

3.4.3 5.4.3 Home Search

Home search is a specific motion routine that is useful for most types of applications. Its goal is to find a specific point in travel relative to the mounting base of the motion device very accurately and repeatable. The need for this absolute reference point is twofold. First, in many applications it is important to know the exact position in space, even after a power-off cycle. Secondly, to protect the motion device from hitting a travel obstruction set by the application (or its own travel limits), the controller uses programmable software limits. To be efficient though, the software limits must be placed accurately in space before running the application.

To achieve this precise position referencing, the ESP302 motion controller executes a unique sequence of moves.

First, let’s look at the hardware required to determine the position of a motion device. The most common (and the one supported by the ESP302) are incremental encoders. By definition, these are encoders that can tell only relative moves, not absolute position. The controller keeps track of position by incrementing or decrementing a dedicated counter according to the information received from the encoder. Since there is no absolute position information, position "zero" is where the controller was powered on (and the position counter reset).

To determine an absolute position, the controller must find a "switch" that is unique to the entire travel, called a home switch or origin switch. An important requisition is that this switch must be located with the same accuracy as the encoder pulses.

If the motion device is using a linear scale as position encoder, the home switch is usually placed on the same scale and read with the same accuracy.

If, on the other hand, a rotary encoder is used, the problem becomes more complicated. To have the same accuracy, a mark on the encoder disk could be used (called index pulse) but because it repeats itself every revolution, it does not define a unique point over the entire travel.

An origin switch, on the other hand, placed in the travel of the motion device is unique but not accurate (repeatable) enough. The solution is to use both, following a search algorithm.

A home switch (Figure) separates the entire travel in two areas: one for which it has a high level and one for which is low. The most important part of it is the transition between the two areas. Also, looking at the origin switch level, the controller knows on which side of the transition it currently is and which way to move to find it.

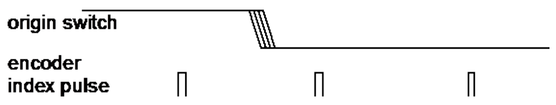


Figure 26: Home (Origin) Switch and Encoder Index Pulse.

The task of the home search routine is to identify one unique index pulse as the absolute position reference. This is done by the first finding the home switch transition and then the very first index pulse (Figure).

So far, we can label the two motion segments D and E. During D the controller is looking for the origin switch transition and during E for the index pulse. To guarantee the best accuracy possible, both D and E segments are performed at a very low speed and without a stop in-between. Also, during E the display update is suppressed to eliminate any unnecessary overhead.

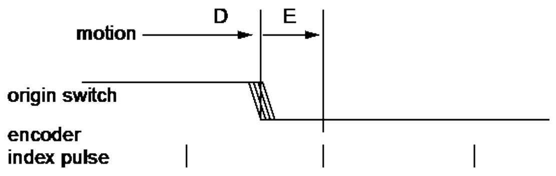


Figure 27: Slow-Speed Home (Origin) Switch Search.

The routine described above could work but has one problem. Using the low speeds, it could take a very long time if the motion device happens to start from the opposite end of travel. To speed things up, we can have the motion device move fast in the vicinity of the home switch and then perform the two slow motions, D and E. The new sequence is shown in Figure 28.

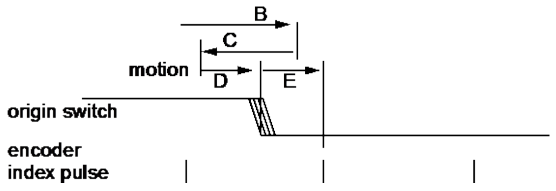


Figure 28: High/Low-Speed Home (Origin) Switch Search.

Motion segment B is performed at high speed, with the pre-programmed home search speed. When the home switch transition is encountered, the motion device stops (with an overshoot), reverses direction and looks for it again, this time with half the velocity (segment C).

Once found, it stops again with an overshoot, reverses direction and executes D and E with one tenth of the programmed home search speed.

In the case when the motion device starts from the other side of the home switch transition, the routine will look like Figure 29.

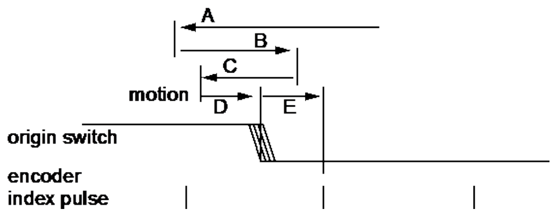


Figure 29: Home (Origin) Search from Opposite Direction.

The ESP302 moves at high speed up to the home switch transition (segment A), and then executes B, C, D and E.

All home search routines are run so that the last segment, E, is performed in the position direction of travel.

CAUTION

The home search routine is very important for the positioning accuracy of the entire system and it requires full attention from the controller. Do not interrupt or send other commands during its execution, unless it is for emergency purposes.

3.5 Encoder

PID closed-loop motion control requires a position sensor. The most widely used technology by far, are incremental encoders.

The main characteristic of an incremental encoder is that it has a 2-bit gray code output, more commonly known as quadrature output (Figure 30).

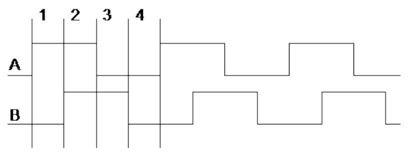


Figure 30: Encoder Quadrature Output.

The output has two signals, commonly known as channel A and channel B. Some encoders have analog outputs (sine – cosine signals), but the digital type are more widely used. Both channels have a 50% duty cycle and are out of phase by 90°. Using both channels and an appropriate decoder, a motion controller can identify four different points within one encoder cycle. This type of decoding is called X4 (or quadrature decoding), meaning that the encoder resolution is multiplied by 4. For example, and encoder with 10 µm phase period can offer a 2.5 µm resolution when used with a X4 type decoder.

Physically, an encoder has two parts: a *scale* and a *read head.* The scale is an array of precision placed marks that are read by the head. The most commonly used encoders, optical encoders, have a scale made out of a series of transparent and opaque lines placed on a glass substrate or etched in a thin metal sheet (Figure 31).

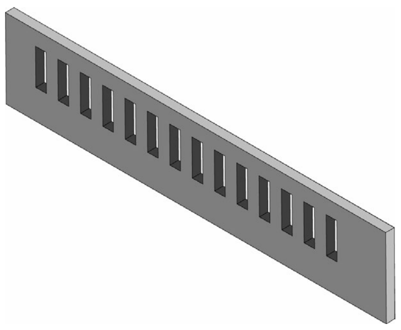


Figure 31: Optical Encoder Scale.

The encoder read head has three major components: a light source, a mask and a detector (Figure 32). The mask is a small scale-like piece, having identically spaced transparent and opaque lines.

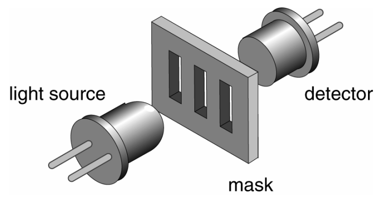


Figure 32: Optical Encoder Read Head.

Combining the scale with the read head, when one moves relative to another, the light will pass through where the transparent areas line up or blocked when they do not line up (Figure 33).

The detector signal is similar to a sine wave. Converting it to a digital waveform, the user will get the desired encoder signal. But, this is only one phase, only half of the signal needed to get position information. The second channel is obtained the same way but from a mask that is placed 90% out of phase relative to the first one (Figure 34).

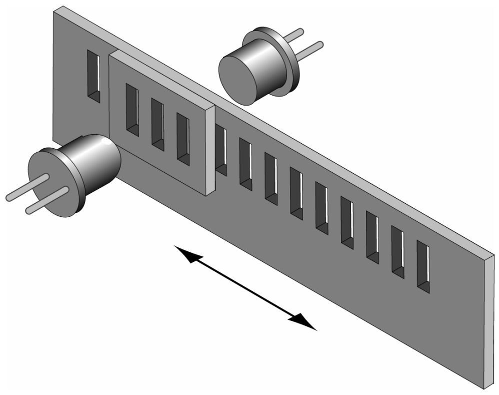


Figure 33: Single-Channel Optical Encoder Scale and Read Head Assembly.

There are two basic types of encoders: *linear* and *rotary.* The linear encoders, also called linear scales, are used to measure linear motion directly. This means that the physical resolution of the scale will be the actual positioning resolution. This is their main drawback since technological limitations prevent them from having better resolutions than a few microns. To get higher resolutions in linear scales, a special delicate circuitry must be added, called scale interpolator.

Other technologies like interferometry or halography can be used but they are significantly more expansive and need more space.

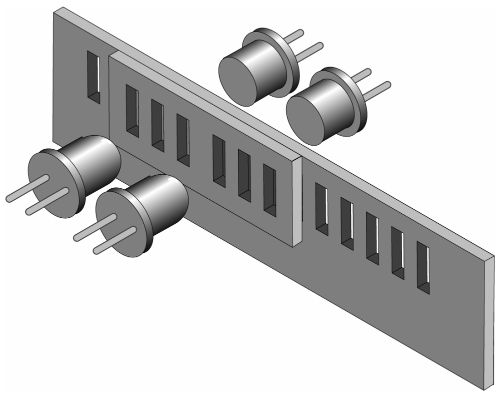


Figure 34: Two-Channel Optical Encoder Scale and Read Head Assembly.

The most popular encoders are rotary. Using gear reduction between the encoder and the load, significant resolution increases can be obtained at low cost. But the price paid for this added resolution is higher backlash.

In some cases, rotary encoders offer high resolution without the backlash penalty. For instance, a linear translation stage with a rotary encoder on the lead screw can easily achieve 1µm resolution with negligible backlash.

NOTE

For rotary stages, a rotary encoder measures the output angle directly. In this case, the encoder placed on the rotating platform has the same advantages and disadvantages of the linear scales.

3.6 Motors

There are many different types of electrical motors, each one being best suitable for certain kind of applications. The ESP302 supports two of the most popular types: *stepper motors* and *DC motors.*

Other technologies like interferometry or halography can be used but they are significantly more expansive and need more space.

Another way to characterize motors is by the type of motion they provide. The most common ones are rotary but, in some applications, linear motors are preferred.

3.6.1 Stepper Motors

The main characteristic of a stepper motor is that each motion cycle has a number of stable positions. This means that, if current is applied to one of its windings (called phases), the rotor will try to find one of these stable points and stay there. In order to make a motion, another phase must be energized which, in turn, will find a new stable point, thus making a small incremental move – a step.

Figure 35 shows the basics of a stepper motor. When the winding is energized, the magnetic flux will turn the rotor until the rotor and stator teeth line up. This true of the rotor core is made out of soft iron. Regardless of the current polarity, the stator will try to pull-in the closest rotor tooth.

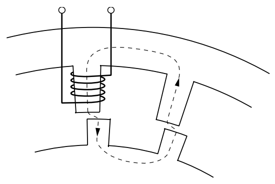


Figure 35: Stepper Motor Operation.

But, if the rotor is a permanent magnet, depending on the current polarity, the stator will pull or push the rotor tooth. This is a major distinction between two different stepper motor technologies: variable reluctance and permanent magnet motors. The variable reluctance motors are usually small, low cost, large step angle stepper motors. The permanent magnet technology is used for larger, high precision motors.

The stepper motor advances to a new stable position by means of several stator phases that have the teeth slightly offset from each other. To illustrate this, Figure 36 shows a stepper motor with four phases and, to make it easier to follow, it is drawn in a linear fashion (as a linear stepper motor).

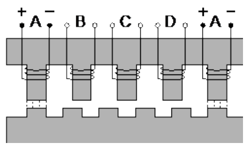


Figure 36: Four-Phase Stepper Motor.

The four phases, from A to D, are energized one at a time (phase A is shown twice). The rotor teeth line up with the first energized phase, A. If the current to phase A is turned off and B is energized next, the closest rotor tooth to phase B will be pulled in and the motor moves one step forward.

If, on the other hand, the next energized phase is D, the closest rotor tooth is in the opposite direction, thus making the motor to move in reverse.

Phase C cannot be energized immediately after A because it is exactly between two teeth, so the direction of movement is indeterminate.

To move in one direction, the current in the four phases must have the following timing diagram (Figure 37).

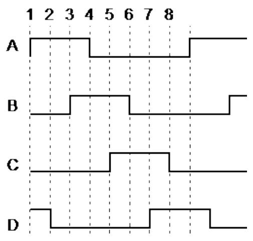


Figure 37: Timing Diagram, Half-Stepping Motor.

Now, what happens if we energize the same two phases simultaneously but with different currents? For example, lets say that phase A has the full current and phase B only half. This means that phase A will pull the rotor tooth twice as strongly as B does. The rotor tooth will stop closer to A, somewhere between the full step and the half step positions (Figure 38).

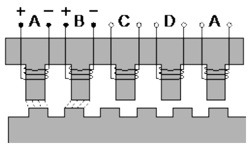


Figure 38: Energizing Two Phases with Different Intensities.

The conclusion is that, varying the ratio between the currents of the two phases, the user can position the rotor anywhere between the two full step locations. To do so, the user needs to drive the motor with analog signals, similar to Figure 39.

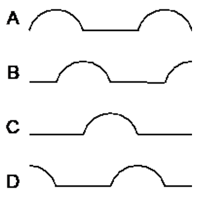


Figure 39: Timing Diagram, Continuous Motion (Ideal).

But a stepper motor should be stepping. The controller needs to move it in certain known increments. The solution is to take the halh-sine waves and digitize them so that for every step command, the currents change to some new pre-defined levels, causing the motor to advance one small step (Figure 40).

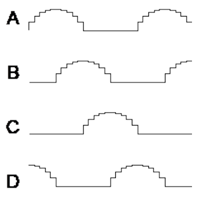


Figure 40: Timing Diagram, Mini-Stepping.

This driving method is called mini-stepping or micro-stepping. For each step command, the motor will move only a fraction of the full-step. Motion steps are smaller so the motion resolution is increased and the motion ripple (noise) is decreased.

However, mini-stepping comes at a price. First, the driver electronics are significantly more complicated. Secondly, the holding torque or one step is reduced by the mini-stepping factor. In other words, for a x10 mini-stepping, it takes only 1/10 of the full-step holding torque to cause the motor to have a positioning error equivalent to one step (a mini-step).

To clarify a little what this means, lets take a look at the torque produced by a stepper motor. For simplicity, lets consider the case of a single phase being energized (Figure 41).

Once the closest rotor tooth has been pulled in, assuming that the user doesn't have any external load, the motor does not develop any torque. This is a stable point.

If external forces try to move the rotor (Figure 42), the magnetic flux will counter this effect. The more teeth misalignment exists, the larger the generated torque.

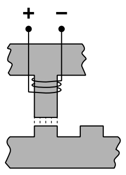


Figure 41: Single Phase Energization.

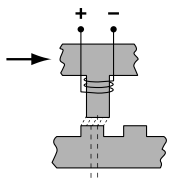


Figure 42: External Force Applied.

If the misalignment keeps increasing, at some point, the torque peaks and then starts diminishing again such that, when the stator is exactly between the rotor teeth, the torque becomes zero again (Figure 43).

This is an unstable point and any misalignment or external force will cause the motor to move one way or another. Jumping from one stable point to another is called missing steps, one of the most critiqued characteristics of stepper motors.

The torque diagram versus teeth misalignment is shown in Figure 44. The maximum torque is obtained at one quarter of the tooth spacing, which is equivalent to one full step.

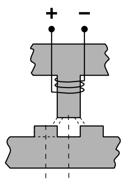


Figure 43: Unstable Point.

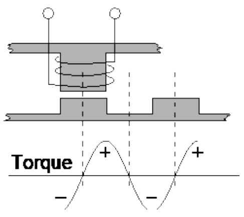


Figure 44: Torque and Tooth Alignment.

This torque diagram is accurate even when the motor is driven with half-, mini-, or micro-steps. The maximum torque is still one full step away from the stable (desired) position.

3.6.1.1 Stepper Motor Types

To simplify the explanation, the examples above are based on a variable reluctance stepper motor. The main characteristic of these motors is that their rotors have no permanent magnets. The variable reluctance motors are easy and inexpensive to make but suffer from higher inefficiency and require a unipolar driver. They are used in low cost, low power applications.

Permanent magnet motors have each "tooth" made out of a permanent magnet, each one having alternate polarity. They are more efficient but the step size is very large due to the physical size of the pole "teeth". They are also being used in low cost and, in particular, miniature applications.

The most common type of stepper motor is the Hybrid stepper motor. It is the fine "teeth" and stepping angle of a variable reluctance motor and the efficiency of the permanent magnet motor. The rotor is made out of one or more stacks that consist of a pair of magnetically opposite polarized sections. These motors offer the best combination of efficiency and fine stepping angles and can be driven by both unipolar and bipolar drivers.

Advantages

Stepper motors are primarily intended to be used for low cost microprocessor controlled positioning applications. Due to some of their inherent characteristics, they are preferred in many industrial and laboratory applications. Some of their main advantages are:

- Low cost full-step, open loop implementation
- No servo tuning required
- Good position lock-in
- No encoder necessary
- Easy velocity control
- Retains some holding torque even with power off
- No wearing or arcing commutators
- Preferred for vacuum and explosive environments.

Disadvantages

Some of the main disadvantages of the stepper motors are:

- Could loose steps (synchronization) in open loop operation
- Requires current (dissipates energy) even at stop
- Generates higher heat levels than other types of motors
- Moves from one step to another are made with sudden motions
- Large velocity ripples, especially at low speeds, causing noise and possible resonances
- Load torque must be significantly lower than the motor holding torque to prevent stalling and missing steps
- Limited high speed.

3.6.2 DC Motors

A DC motor is similar to a permanent magnet stepper motor with an added internal phase commutator (Figure 45).

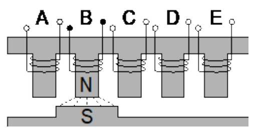


Figure 45: DC Motor.

Applying current to phase B pulls in the rotor pole. If, as soon as the pole gets there, the current is switched to the next phase C, the rotor will not stop but continue moving to the next target.

Repeating the current switching process will keep the motor moving continuously. The only way to stop a DC motor is not to apply any current to its windings. Due to the permanent magnets, reversing the current polarity will cause the motor to move in the opposite direction.

Of course, there is a lot more to the DC motor theory but this description gives the user a general idea on how they work.

A few other characteristics to keep in mind are:

- For a constant load, the velocity is approximately proportional to the voltage applied to the motor.
- For accurate positioning, DC motors need a position feed-back device.
- Constant current generates approximately constant torque.
- If DC motors are tuned externally (manually, etc.) they act as generators.

Advantages

DC motors are preferred in many applications for the following reasons:

- Smooth, ripple-free motion at any speed.
- High torque per volume.
- No risk of loosing position (in a closed loop) .
- Higher power efficiency than stepper motors.
- No current requirement at stop.
- Higher speeds can be obtained than with other types of motors.

Disadvantages

Some of the DC motor's disadvantages are:

- Requires a position feedback encoder and servo loop controller.
- Requires servo loop tuning.
- Commutator may wear out in time.
- Not suitable for high vacuum application due to the commutator arcing.
- Hardware and setup are more costly than for an open loop stepper motor (full stepping).

4.0 Servo Tuning

4.1 Tuning Principles

The ESP302 controller uses a PID servo loop with feed-forward. Servo tuning sets the **Kp, Ki,** and **Kd,** and feed-forward parameters of the digital PID algorithm, also called the PID filter.

Tuning PID parameters requires a reasonable amount of closed-loop system understanding. First review the Control Loops paragraph in the Motion Control Tutorial Section. If needed, consult additional servo control theory books.

Start the tuning process using the default values supplied with the stage. These values are usually very conservative, favoring safe and oscillation-free operation. To achieve the best dynamic performance possible, the system must be tuned for the specific application. Load, acceleration, stage orientation, and performance requirements all affect how the servo loop should be tuned.

4.2 Tuning Procedures

Servo tuning is usually performed to achieve better motion performance (such as reducing the following error statically and/or dynamically) or because the system is malfunctioning (oscillating and/or shutting off due to excessive following error).

Acceleration plays a significant role in the magnitudes of the following error and overshoot, especially at start and stop. Rapid velocity changes represent very high acceleration, causing large following errors and overshoot. Use the smallest acceleration the application can tolerate to reduce overshoot and make tuning the PID filter easier.

NOTE

In the following descriptions, it is assumed that a software utility is being used to capture the response of the servo loop during a motion step command, and to visualize the results.

4.2.1 Hardware and Software Requirements

Hardware Requirements

Tuning is best accomplished when the system response can be measured. This can be done with external monitoring devices but can introduce errors.

The ESP302 controller avoids this problem by providing an internal *tune* capability. When *tune* mode is activated, the controller records a number of different parameters. The parameters can include real instantaneous position, desired position, desired velocity, desired acceleration.

The sample interval can be set to one servo cycle (100 µs) or any multiple of it and the total number of samples can be up to 1000 points.

This is a powerful feature that the user can take advantage of to get maximum performance out of the motion system.

4.2.2 Correcting Axis Oscillation

There are three parameters that can cause oscillation. The most likely to induce oscillation is **Ki,** followed by **Kp** and **Kd.** Start by setting **Ki** to zero and reducing **Kp** and **Kd** by 50%.

If oscillation does not stop, reduce **Kp** again.

When the axis stops oscillating, system response is probably very soft. The following error may be quite large during motion and non-zero at stop. Continue tuning the PID with the procedures described in the next paragraph.

4.2.3 Correcting Following Error

If the system is stable and the user wants to improve performance, start with the current PID parameters. The goal is to reduce following error during motion and to eliminate it at stop.

Guidelines for further tuning (based on performance starting point and desired outcome) are provided in the following paragraphs.

Following Error Too Large

This is the case of a soft PID loop caused by low values for **Kp** and **Kd.**

First increase **Kp** by a factor of 1.5 to 2. Repeat this operation while monitoring the following error until it starts to exhibit excessive ringing characteristics (more than 3 cycles after stop). To reduce ringing, add some damping by increasing the **Kd** parameter.

Increase it by a factor of 2 while monitoring the following error. As **Kd** is increased, overshoot and ringing will decrease almost to zero.

NOTE

Remember that if acceleration is set too high, overshoot cannot be completely eliminated with Kd.

If **Kd** is further increased, at some point oscillation will reappear, usually at a higher frequency. Avoid this by keeping **Kd** at a high enough value, but not so high as to re-introduce oscillation.

Increase **Kp** successively by approximately 20% until signs of excessive ringing appear again.

Alternately increase **Kd** and **Kp** until **Kd** cannot eliminate overshoot and ringing at stop. This indicates **Kp** is larger than its optional value and should be reduced. At this point, the PID loop is very tight.

Ultimately, optimal values for **Kp** and **Kd** depend on the stiffness of the loop and how much ringing the application can tolerate.

NOTE

The tighter the loop, the greater the risk of instability and oscillation when load conditions change.

Errors At Stop (Not In Position)

If you are satisfied with the dynamic response of the PID loop but the stage does not always stop accurately, modify the integral gain factor **Ki.** As described in the Motion Control Tutorial section, the **Ki** factor of the PID works to reduce following error to near zero. Unfortunately it can also contribute to oscillation and overshoot. Change this parameter carefully, and if possible, in conjunction with **Kd.**

Start with the integral saturation (**KS**) set to a high value and **Ki** value at least two orders of magnitude smaller than **Kp.** Increase its value by 50% at a time and monitor overshoot and final position at stop.

If intolerable overshoot develops, increase the **Kd** factor. Continue increasing **Ki, KS** and **Kd** alternatively until an acceptable loop response is obtained. If oscillation develops, immediately reduce **Ki** and **KS.**

Remember that any finite value for **Ki** will eventually reduce the error at stop. It is simply a matter of how much time is acceptable for the application. In most cases it is preferable to wait a few extra milliseconds to get to the stop in position rather than have overshoot or run the risk of oscillations.

Following Error During Motion

This is caused by a **Ki,** and **KS** value that is too low. Follow the procedures in the previous paragraph, keeping in mind that it is desirable to increase the integral gain factor as little as possible.

4.2.4 Points to Remember

- The ESP302 controller uses a servo loop based on the PID with velocity and acceleration feed-forward algorithm.
- Use the lowest acceleration the application can tolerate. Lower acceleration generates less overshoot.
- Use the default values provided with the system for all standard motion devices as a starting point.
- Use the minimum value for **Ki,** and **KS** that gives acceptable performance. The integral gain factor can cause overshoot and oscillations.

A summary of servo parameter functions is listed in Table 4.

Parameter	Function	Value Set Too Low	Value Set Too High
Kp	Determines stiffness of servo loop.	Servo loop too soft with high following errors.	Servo loop too tight and/or causing oscillation.
Kd	Main damping factor, used to eliminate oscillation.	Uncompensated oscillation caused by other parameters being high.	Higher-frequency oscillation and/or audible noise in the motor caused by large ripple in the motor voltage.
Ki	Reduces following error during long motions and at stop.	Stage does not reach or stay at the desired stop position.	Oscillations at lower frequency and higher amplitude
Vff	Reduces following error during the constant velocity phase of a motion.	Negative following error during the constant velocity phase of a motion. Stage lags the desired trajectory.	Positive following error during the constant velocity phase of a motion. Stage is ahead of the desired trajectory.
Aff	Reduces following error during the acceleration and deceleration phases of a motion.	Negative following error during the acceleration phase of a motion. Stage lags the desired trajectory.	Position following error during the acceleration phase of a motion. Stage is ahead of the desired trajectory.

Table 4: Servo Parameter Functions.

Service Form

Your Local Representative

Tel.: ___________________

Fax: ___________________

Name: ____________________	Return authorization #: ____________________ *(Please obtain prior to return of item)*
Company: ____________________	
Address: ____________________	Date: ____________________
Country: ____________________	Phone Number: ____________________
P.O. Number: ____________________	Fax Number: ____________________
Item(s) Being Returned: ____________________	
Model#: ____________________	Serial #: ____________________

Description: ____________________

Reasons of return of goods (please list any specific problems): ____________________

Visit Newport Online at:

www.newport.com

North America & Asia
Newport Corporation
1791 Deere Ave.
Irvine, CA 92606, USA
Sales
Tel.: (800) 222-6440
e-mail: sales@newport.com
Technical Support
Tel.: (800) 222-6440
e-mail: tech@newport.com
Service, RMAs & Returns
Tel.: (800) 222-6440
e-mail: service@newport.com

Europe
MICRO-CONTROLE Spectra-Physics S.A.S
9, rue du Bois Sauvage
91055 Évry CEDEX
France

Sales
Tel.: +33 (0)1.60.91.68.68
e-mail: france@newport.com

Technical Support
e-mail: tech_europe@newport.com

Service & Returns
Tel.: +33 (0)2.38.40.51.55