Servo Motion Control Algorithms and Tuning of PID Parameters

Every servo control project must deal with this task: tuning the parameters of the servo control algorithm, which is almost always a variant of PID control algorithm. Proper tuning of the parameters of the PID servo control algorithm makes the motion control system achieve good performance. Poor tuning under-utilizes its capabilities to say the least.

For every Servo Drive/Motor Supplier, we need to use a Supplier provided drive communication and setup software, and commission the servo drive/motor as follows:

1. Download and Install on PC: Supplier provided software for the Servo Drive Setup
2. Connect PC and Servo Drive with a communication cable (most commonly USB)
3. Run the Communication Software on PC and Establish Communication between the PC and Servo Drive
4. Enable/Disable Drive
5. Configure the Drive: Travel Limits, Torque Limits, etc.
6. Tune the Servo Loop: PID gains
7. Test motions: Jog, Stop, Index, Home

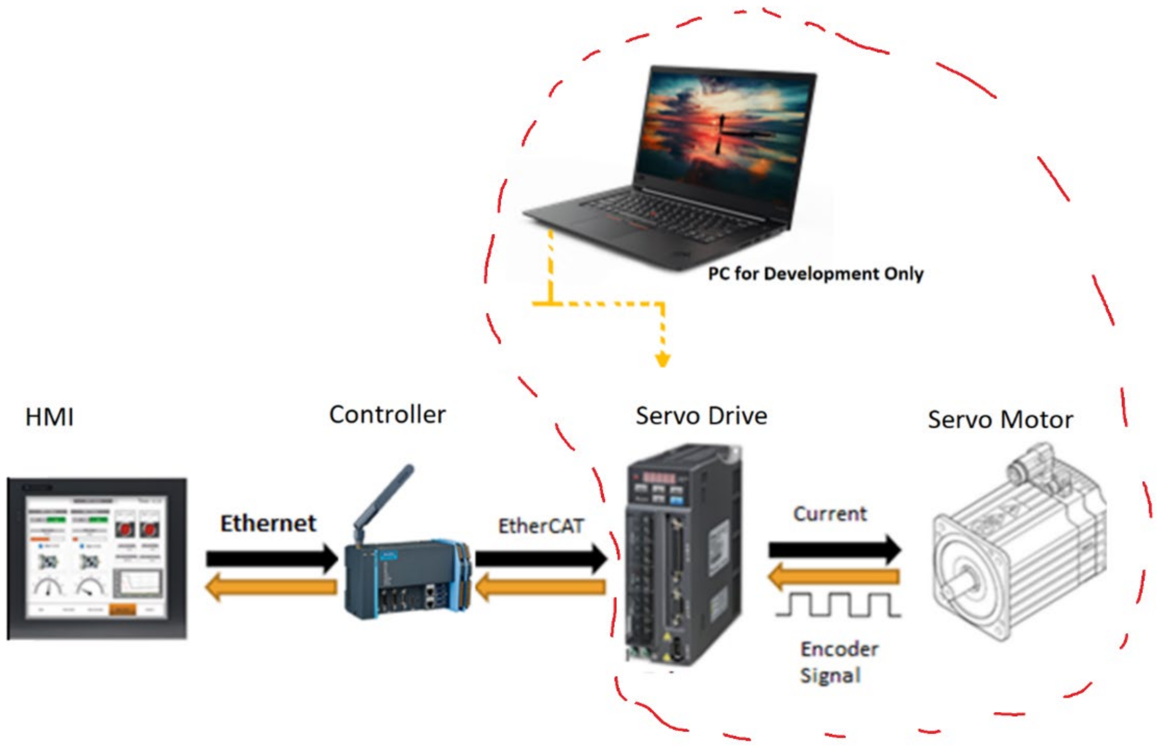

Fig. A-1: A single axis servo motion control system: servo motor, drive, controller, HMI for normal operation. A Laptop PC is used for offline application development purpose.

For EtherCAT interfaced drives, to control the Servo Drive by the Motion Controller, the EtherCAT Master, install ESI file to CODESYS IDE

1. Download the ESI file for the Servo Drive
2. Install the ESI file to the CODESYS IDE, Device Depository.
3. Use CODESYS application software for motion control, addressing the drive in the application code via the ESI file interface driver software.

Figure A-1 below shows the system we deal with: HMI, controller, drive, motor, load and the PC for the development purpose. For servo tuning, we use the components in dotted red circle: Laptop PC, Servo Drive and Servo Motor.

Tuning of servo control algorithm is very important for the performance of the motion control. Figure A-2 shows conceptual illustration of different cases of tuning and the resulting motion performance. Too low gains results in slow and sluggish motion. Too large gains results in high vibrations and even instability. Our goal is to find tuning parameter values that result in “good” smooth and fast motion.

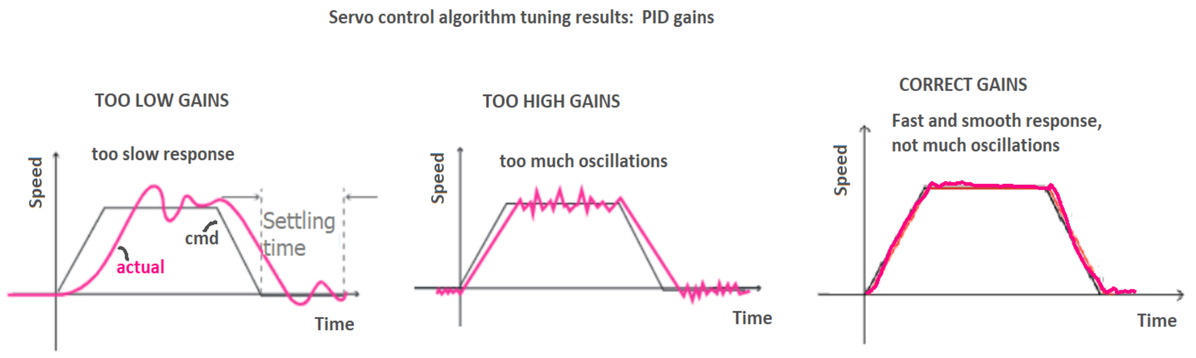

Fig. A-2: Commanded speed versus actual speed in a servo control system. Depending on how well the servo algorithm parameters are tuned, the tracking performance can be significantly different.

Servo control loop tuning can be explained using frequency response analysis. There are three main control loops in servo motor control (Fig. A-3):

1. Current (torque) control loop,
2. Speed control loop,
3. Position control loop.

These control loops can be implemented either in the motion controller or in the servo drive. In EtherCAT slave servo drives, the default mode of the drive configuration is that all three loops are implemented in the Servo Drive and the motion controller simply sends the commanded position signal. However, an EtherCAT slave Servo Drive can be configured to implement current mode, speed mode, or position mode. If current mode only is implemented in the drive, the motion controller implements the position loop and speed mode, providing current command to the Servo Drive. If the Servo Drive is configured to implement speed loop and current loop, the motion controller implements just the position loop and provides the commanded speed to the Servo Drive.

The logic of the servo control algorithm is set by each supplier and is not modified by the end user. Furthermore, Suppliers never provide the exact details of their servo control algorithm. Within the algorithm’s logic, there are parameters that must be tuned to best fit the algorithm for different applications.

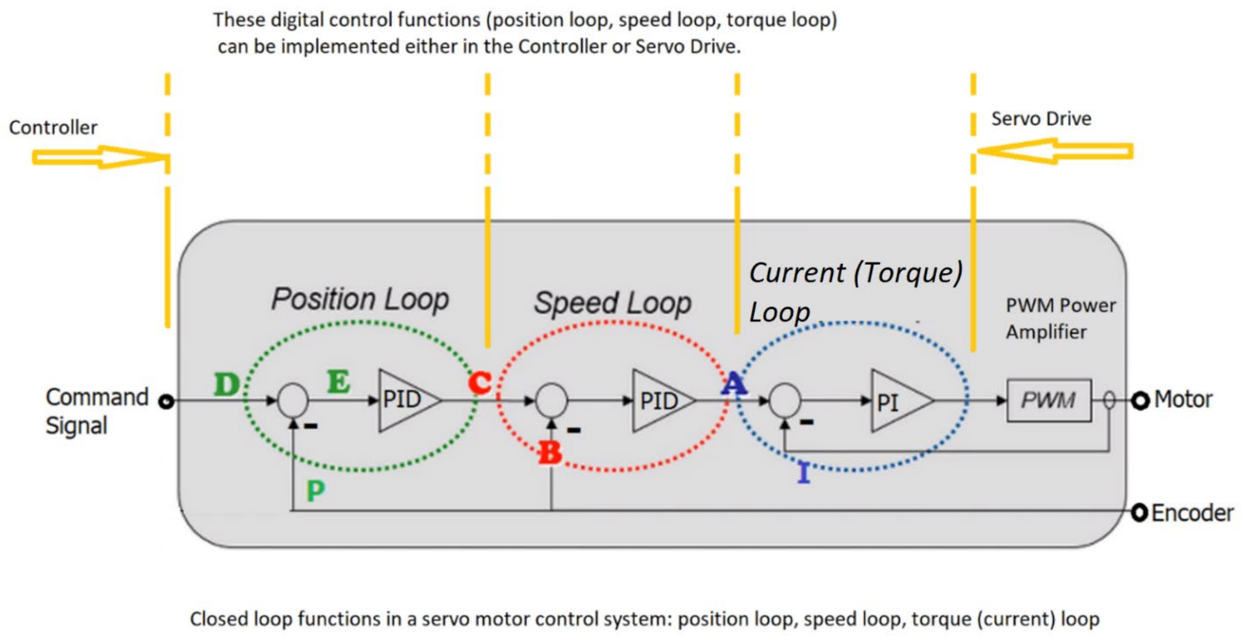
These digital control functions (position loop, speed loop, torque loop) can be implemented either in the Controller or Servo Drive.
Controller
Servo Drive
Position Loop
Speed Loop
Current (Torque) Loop
PWM Power Amplifier
Command Signal
D
E
PID
C
PID
A
PI
PWM
Motor
B
I
P
Encoder
Closed loop functions in a servo motor control system: position loop, speed loop, torque (current) loop

Fig. A-3: Servo Main Control Loops.

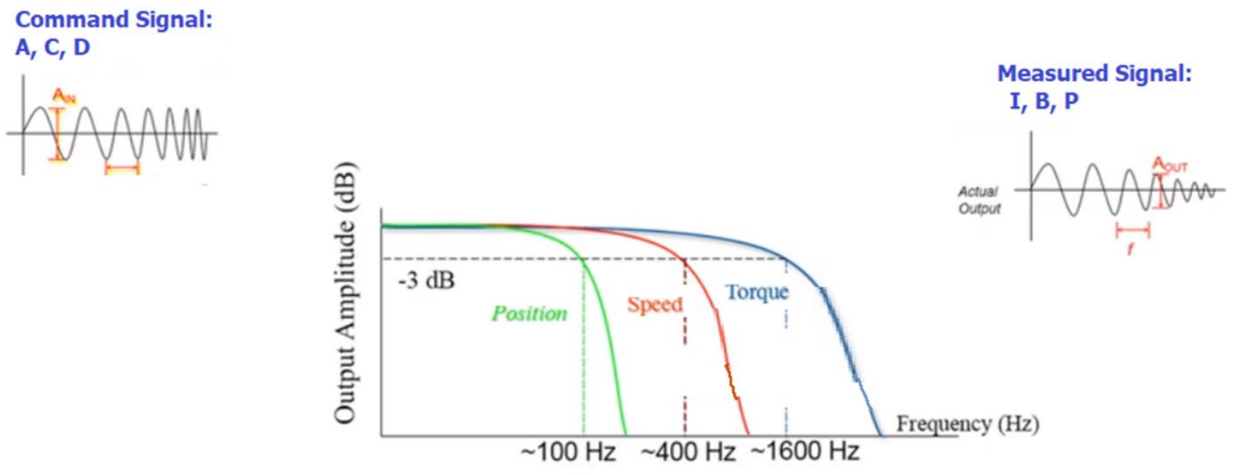

Fig. A-4: Frequency response relationship between input (command signal) and output (measured feedback signal): relative bandwidth of current (torque) loop, speed loop and position loop.

The data for figure A-4 can conceptually be obtained as follows. To obtain the "Torque" (current) loop frequency response, configure the drive in "current mode". Command a small magnitude (small relative to maximum current command, i.e., 10% of maximum current) sinusoidal signal, that is the commanded current signal. Start with a slow sinusoid frequency. Then record the actual current magnitude and phase relationship in steady state between the command signal and actual signal. Steady state means that wait for long enough for the input and output relationship is steady. Repeat that for many other frequencies. Then plot the ratio of output signal magnitude to input signal magnitude versus frequency in logarithmic scale. We can also plot the phase angle versus the frequency in the logarithmic scale. The resulting two plots is called "Bode plots". Here we show only the magnitude ratio plot.

Repeat the same measurement for speed and position loops. Configure the drive for "speed mode". Generate same command signal (that is commanded speed in this case), measure actual speed and record steady state data and plot.

Similarly for position loop, configure the drive for "position mode", generate same command signal (that is commanded position in this case), measure actual position and record steady state data and plot.

Each inner loop acts as a slave (follower) to the command of outer loop (Fig. A-4). Each inner loop must be at least 3 to 5 times faster than its immediate outer loop. Velocity loop and position loop quite often are implemented as one combined loop.

Let us assume we have three separate loops, and each is controlled by a form of PID control algorithm. The task of tuning is to find the best combination of PID algorithm parameters for a given application. In order to measure the frequency response of each loop, we can do the following:

1. Current loop only: position loop and speed loop are disabled

- Command a sinusoidal current signal at signal point A, slowly varying frequency from zero to a few kHz ranges, i.e., 10kHz.
- Record the actual current signal at point I.
- Calculate the magnitude ratio of I / A, and phase difference between I(t) and A(t) at selected frequencies: get a table of frequency, I/A, Phase (I, A).

The amplitude ratio data I/A as function of frequency should look like "blue" plot on the above diagram. Phase data, Phase (I, A) versus frequency, can be plotted in a second figure, but not shown in the figure.

2. Speed loop only: disable position loop, current loop is enabled and tuned.

- Command a sinusoidal speed command signal at signal point C, slowly varying frequency from zero to a few kHz ranges, i.e., 2kHz.
- Record the actual signal (velocity feedback) at point B.
- Calculate the magnitude ratio of B/C, and phase difference between B(t) and C(t) at selected frequencies: get a table of frequency, B/C, Phase (B, C).

The amplitude curve C/B as function of frequency should look like the "red" plot in the figure above. Phase data, Phase (C, B) versus frequency, can be plotted in a second figure, but not shown in the figure.

3. Position loop:

- Command a sinusoidal position command signal at signal point D, slowly varying frequency from zero to a few 100 Hz range, i.e., 500Hz.
- Record the actual signal at point P.
- Calculate the magnitude ratio of P/D, and phase difference between P(t) and D(t) at selected frequencies: get a table of frequency, P/D, Phase (P, D).

The amplitude curve P/D as function of frequency should look like the "green" plot in the figure above. Phase data, Phase (P, D) versus frequency, can be plotted in a second figure, but not shown in the figure.

Consider the plots of recorded amplitude ratios I/A, B/C, P/D (and phase functions Phase (I, A), Phase (C, B), Phase (P, D)) as function of frequency for current loop, speed loop, and position loop response. These responses will change depending on the tuning parameter values chosen for the control algorithm of each loop. The speed of response, also called bandwidth, of a dynamic system, hence each of the control loops, is defined as the frequency at which output response magnitude to input response magnitude ratio is 0.707 (- 3.0 dB). Notice that an inner loop essentially acts as a "server" or "slave" to its outer loop. Hence, the inner loop should be faster than the outer loop in order to be able track its commands from the outer loop.

The rule of thumb is that the tuning parameter values should be chosen such that each outer loop is 3 to 5 times faster than its inner loop. For example, if the desired position loop bandwidth is 100Hz, the speed loop bandwidth should be at least 300 to 500 Hz (i.e., 400 Hz), and the current loop bandwidth at least 3 to 5 times that, i.e., 1600Hz.

Notice that, since each outer loop must be 3 to 5 times slower than its inner loop, the ultimate bandwidth of the closed loop control is determined by the current loop. For example, if we have a current loop with bandwidth of 500Hz only, and we want to maintain a ratio of 5 between each inner and outer loop, then the bandwidth of speed loop can only be 100 Hz and position loop can only be 20 Hz. If the current loop bandwidth was 2500Hz, then speed loop bandwidth can be 500 Hz, and position loop bandwidth can be 100 Hz.

The processes of achieving the desired bandwidth for each control loop is the task of tuning the control loops (servo loops). Tuning is finding a specific numerical value for parameters of the PID like algorithms, along with other improvements such as notch filters for vibration suppression, friction compensation parameters, backlash compensation parts of the servo control algorithm.

Current Control Loop:

Most inner closed loop control is the current control loop (Fig.A-5). Everything in terms of control accuracy depends on how accurately the current is controlled. In addition to the current loop

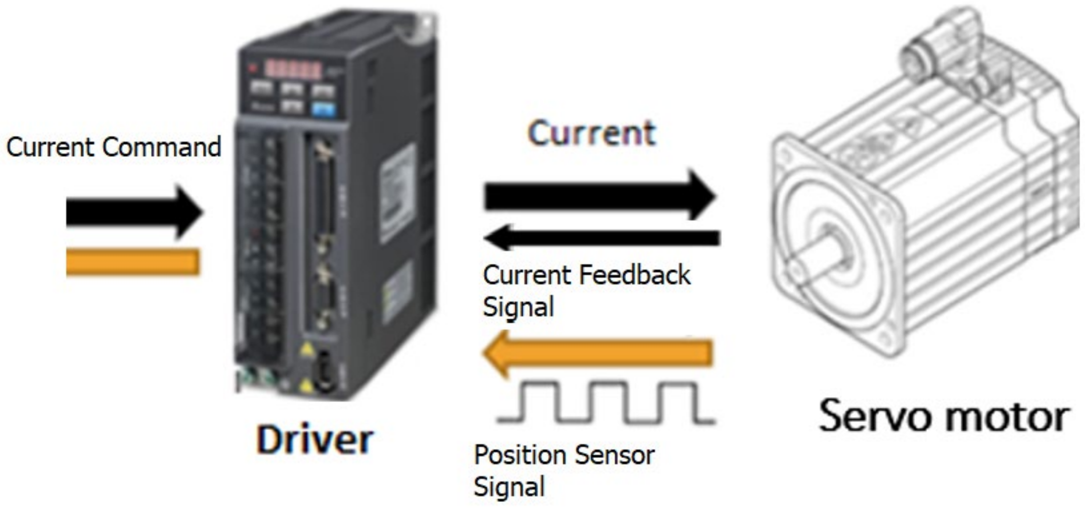

Fig. A-5: Servo drive and motor: servo drive in current mode.

Control algorithm between commanded current and actual measured current, which is typically a PI algorithm. There is also some inaccuracy in current control due to commutation error. By commutation error, we mean the change in generated torque for a given constant current as rotor position changes.

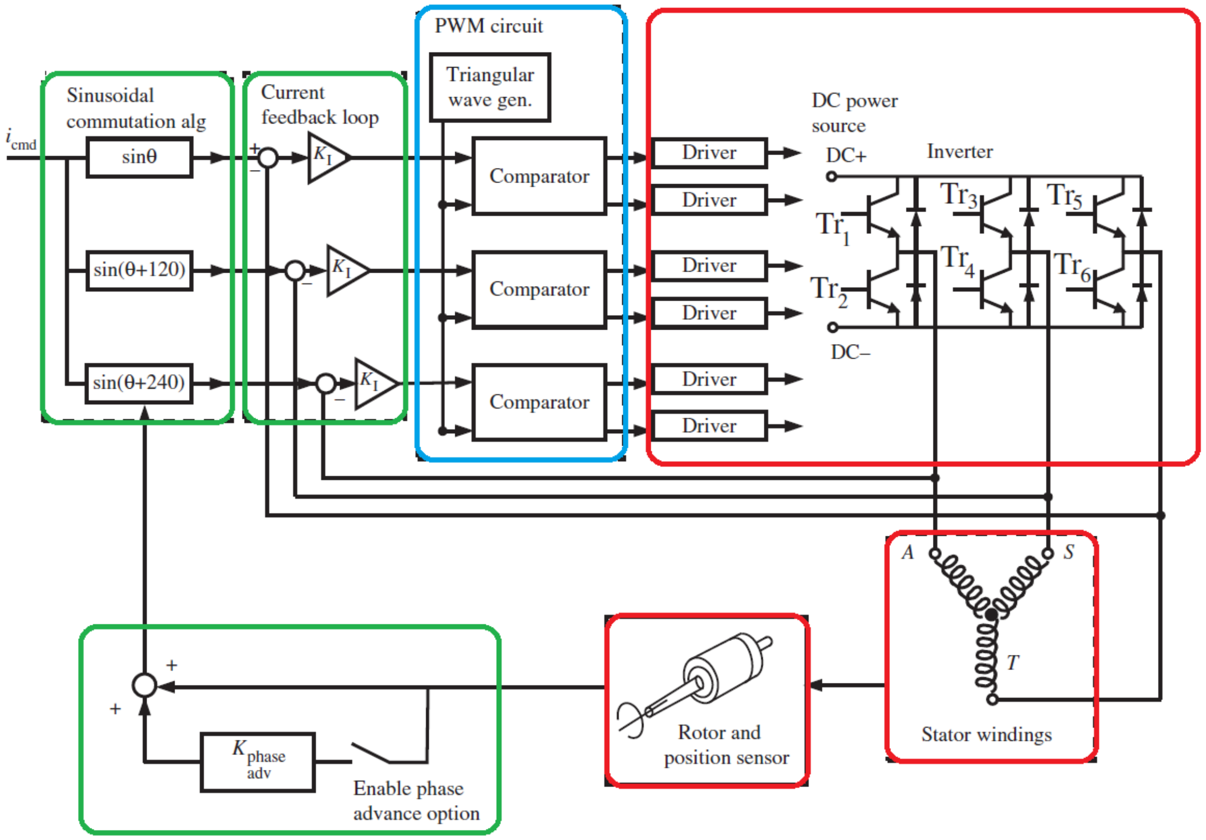

Fig. A-6: Block diagram of a servo drive and brushless servo motor: servo drive has power stage transistors (six), PWM circuit, current feedback loop for each phase, and current commutation algorithm that determines the commanded current to each phase.

Figure above (Fig. A-6) shows the block diagram of a current controlled drive for a three-phase brushless DC motor. The switch set is based on the familiar H-bridge but uses three bridge legs instead of two legs as is the case for an H-bridge drive for brush-type DC motors. Since each leg of the H-bridge has two power transistors, the brushless motor drive has six power transistors. The stator windings are connected between the three bridge legs. The so-called Y-connection shown is the most common type of phase winding connection, while Δ-connection is used in rare cases. At any given time, three of the transistors are ON and three are OFF. Furthermore, two of the windings are connected between the DC bus voltage potential and have current passing through them in a positive or negative direction, whereas the third winding acts as the balance circuit. The combination of the ON/OFF transistors determines the current pattern on the stator, hence the flux field vector generated by the stator. In order to generate the maximum torque per unit current, the objective is to keep the stator’s magnetic field perpendicular to that of the rotor. By controlling the phase currents in the stator phase windings, we control the stator’s magnetic field magnitude and direction, a vector quantity). Therefore, the torque direction and magnitude can be controlled by controlling the stator’s magnetic field relative to that of the rotor.

As rotor position changes, there is a ripple effect in the torque generated for a given current as function of rotor position due to the finite number of stator electromagnets and rotor electromagnets in brushless servo motors.

It is easier to understand the current ripple due to commutation errors for brush-type DC servo motors (Fig. A-7). Since there are a finite number of brushes, as rotor position changes, the torque gain (hence torque) changes for a constant current.

The analogous condition exists in brushless servo motors: in this case, current commutation is done in software, i.e., a sinusoidal function or a similarly shaped look-up table, for each phase of the stator current loop. The objective is to maintain a constant gain between current and torque, independent of rotor position. However, it is impossible to achieve this perfectly. The sources of error are due to

1. the finite sampling of commutation angle in the algorithm,
2. the mismatch between the actual magnetic field of the motor (back EMF function) as rotor angle varies for one revolution versus the functions implemented in the commutation algorithm. This is what meant in industry by “matched drive and motor” – that is custom tuning the commutation algorithm of a specific servo drive to the measured back EMF of a specific motor.

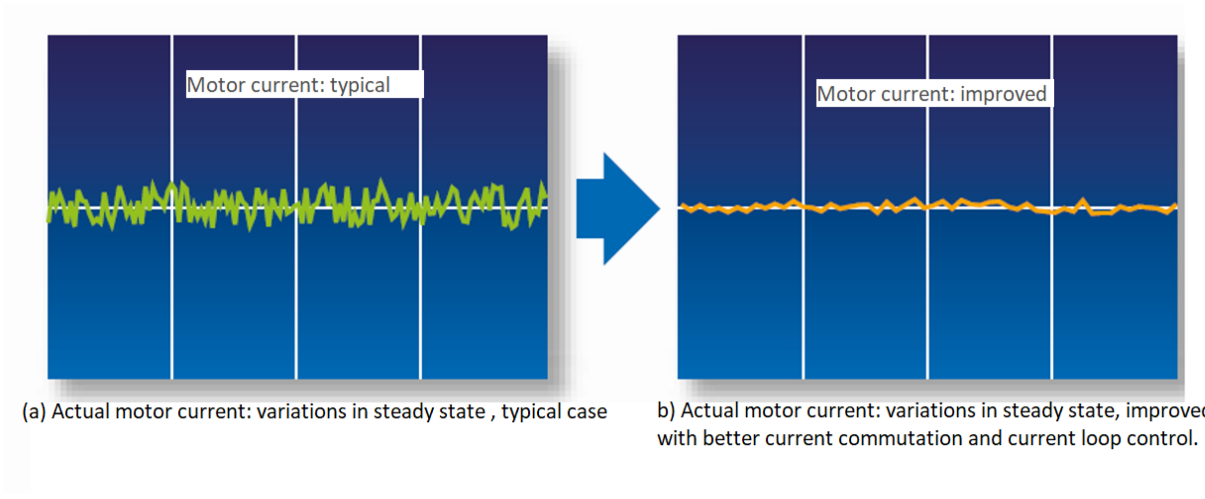

Fig. A-7: An example of current control loop performance.

Phase advance option in software is the fact that there always some time delay, in the order of inverse of the current loop bandwidth, for the actual power currents to develop in the motor stator windings. By the time these currents develop to form the stator’s magnetic field, the rotor (and its magnetic field) is at a little different position, resulting in non-optimal (non-perpendicular) magnetic field vectors. Anticipating that delay effect, and that difference gets larger as the rotor speed increases, we can in software change the feedback signal to anticipate that error so that we target the stator current vector distribution (hence the stator magnetic field) to occur a little different orientation to achieve the optimal perpendicular relationship between the stator and rotor magnetic fields.

Position and Speed Control Loop:

The typical tuning parameters are numbers for:

1. proportional gain,
2. derivative gain,
3. integral gain: enable/disable integral action based on certain conditions of the commanded signal, including integrator anti-windup logic and integral control limit
4. velocity feedforward gain,
5. acceleration feedforward gain,
6. dead band and limit on error passed on to the servo loop,

Depending on the algorithm logic, there may be other parameters that may be tuned such as

7. gravity compensation,
8. friction compensation,
9. nonlinear compensation,
10. backlash compensation,
11. vibration suppression:
    a. vibrations within (below) position loop bandwidth can be actively suppressed as part of feedforward filters (command-shaping) of the position loop
    b. vibrations above position loop bandwidth can be actively suppressed only by anti-resonant **notch filters** specifically targeting selected frequencies so that they are specifically not excited. Notch filters to cancel resonance at specific frequency (or frequencies); parameters for each notch frequency: 1. notch frequency, and 2. width of the notch around that frequency,

c. Low pass filter; parameters: 1. bandwidth and 2. damping ratio

12. control signal limit (saturation imposed at software)
13. gain scheduling: multiple sets of the gains can be determined in advance, and in real time, gains can be switched from one set to another set based on a logic that is dependent of measurements. For example, there can be two sets of tuning parameters; Gain Set 1 and Gain Set 2. We can define a "Condition" to switch between Gain Set 1 and Gain Set 2. The "Condition" can be any condition, i.e., when the commanded velocity is zero. Furthermore, the switching condition can be defined to have a delay time period after the "Condition" is TRUE, and that the gain values are linearly interpolated between two values (for each gain) for within a transition time period. An example is shown below (Fig. A-8)

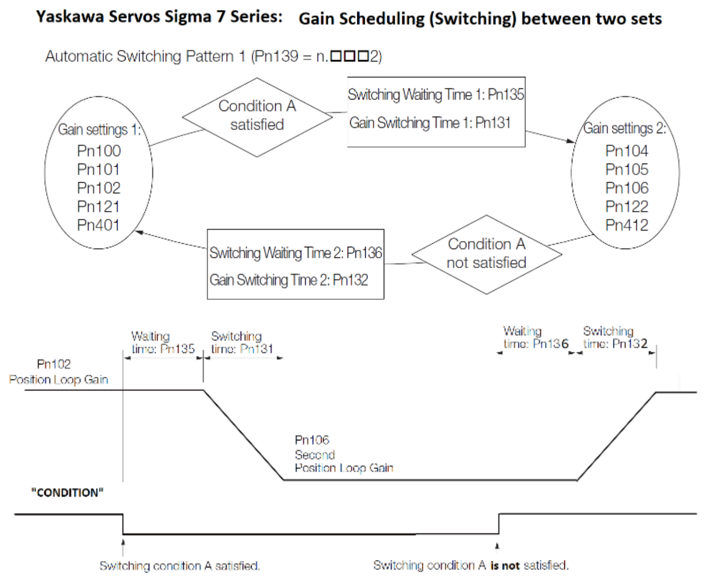
Yaskawa Servos Sigma 7 Series: Gain Scheduling (Switching) between two sets
Automatic Switching Pattern 1 (Pn139 = n.□□□2)
Condition A satisfied
Switching Waiting Time 1: Pn135
Gain Switching Time 1: Pn131
Gain settings 1: Pn100 Pn101 Pn102 Pn121 Pn401
Gain settings 2: Pn104 Pn105 Pn106 Pn122 Pn412
Switching Waiting Time 2: Pn136
Gain Switching Time 2: Pn132
Condition A not satisfied
Waiting time: Pn135
Switching time: Pn131
Waiting time: Pn136
Switching time: Pn132
Pn102 Position Loop Gain
Pn106 Second Position Loop Gain
"CONDITION"
Switching condition A satisfied.
Switching condition A is not satisfied.

Fig. A-8: Gain scheduling example in servo tuning: Yaskawa Sigma 7 series servo drive case.

Here we will address the practical tuning of the first five parameters. Also in our discussions, it is our objective to provide this procedure and insightful understanding without complicated mathematics.

Textbook digital PID servo control algorithm is shown in time-domain and z-domain block diagram forms below (Fig. A-9).

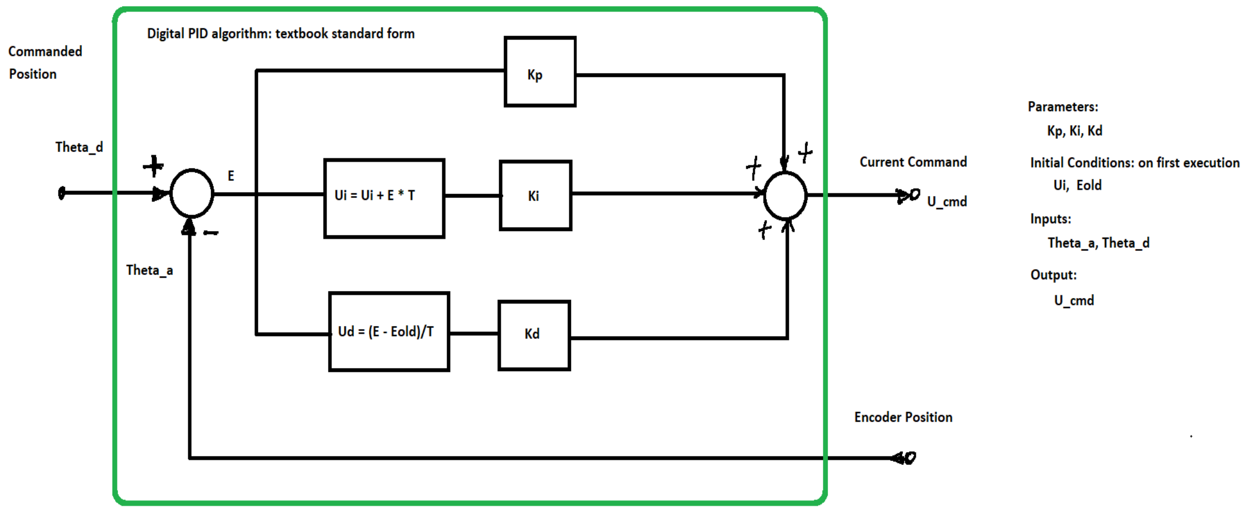

Fig. A-9a: Digital PID control algorithm block diagram, where block relations are expressed with time domain notation.

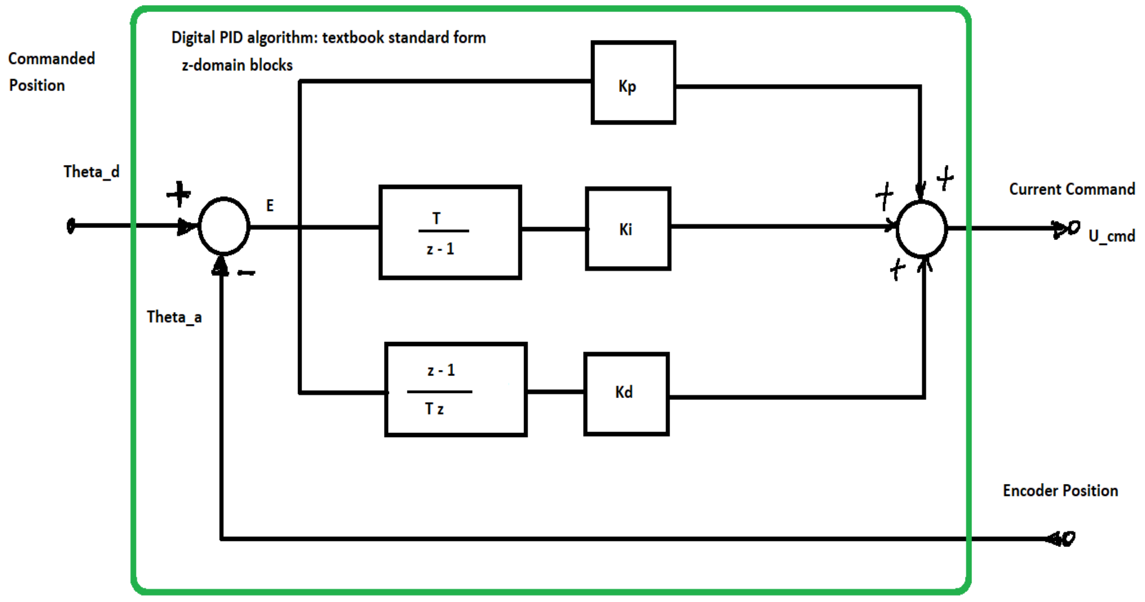

Fig. A-9b: Digital PID control algorithm block diagram, where block relations are expressed in z-domain notation.

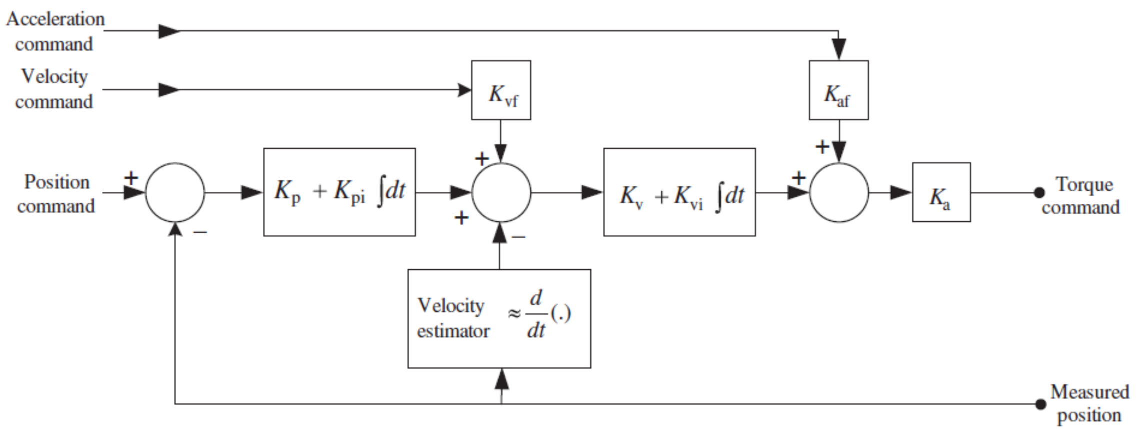

Fig. A-10a: A commercial example of a PID control algorithm used in servo motor control.

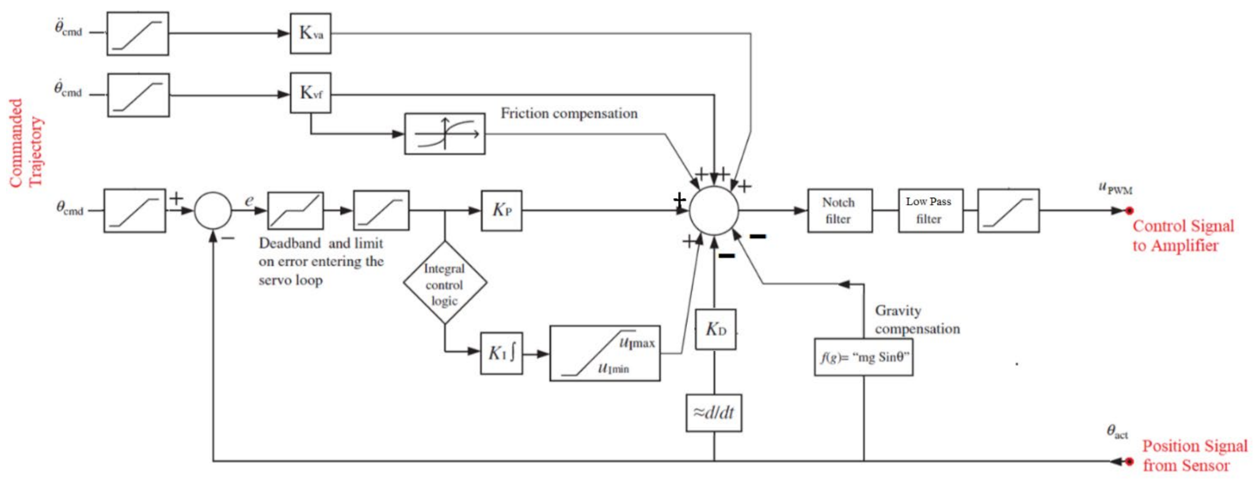

Fig. A-10b: Another commercial example of a PID control algorithm used in servo motor control.

Command motion is always generated in the form of commanded position, commanded velocity and commanded acceleration by the trajectory generator (Fig. A-10). Whether to use all of them or some of them in a closed loop PID algorithm is up to the PID algorithm type. Some PID position control algorithms may use all three, some may use just the commanded position. If the PID algorithm implements only a closed loop velocity (speed) control mode, then commanded position would not be used. On the feedback side, in position mode, we always use position and velocity feedback, but not acceleration feedback due to the level of noise involved in measuring/estimating acceleration.

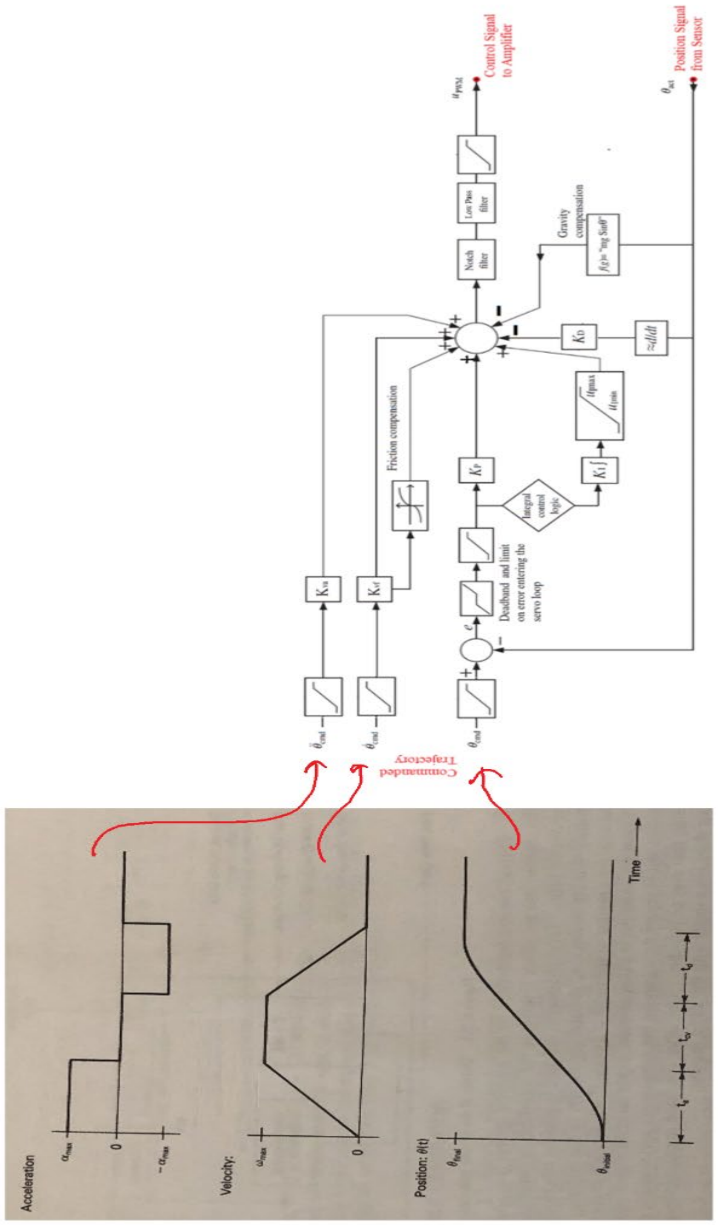

Fig. A-11: Commanded motion signals in servo control: a typical motion command profile uses trapezoidal velocity command.

The commanded motion generator function uses different velocity profile algorithm to generate the commanded position, commanded velocity and commanded acceleration for a given "index" motion (a move from one position to another position).

The trapezoidal profile is trapezoidal in commanded velocity; hence acceleration profile is positive constant (during increasing speed), zero (during constant speed), and negative constant (during deceleration). In trapezoidal profile, acceleration curve is discontinuous. "Jerk", time derivative of acceleration, values are infinite at acceleration change points (Fig. A-11).

S-curve profile is another popular commanded motion generator algorithm. As shown in the figure, the acceleration profile is trapezoidal, hence "Jerk" profile has finite values, resulting in smoother (fewer mechanical vibrations) motions.

Other commonly used motion profiles are sin^2 and quadratic functions for different phases of acceleration functions (Fig. A-12).

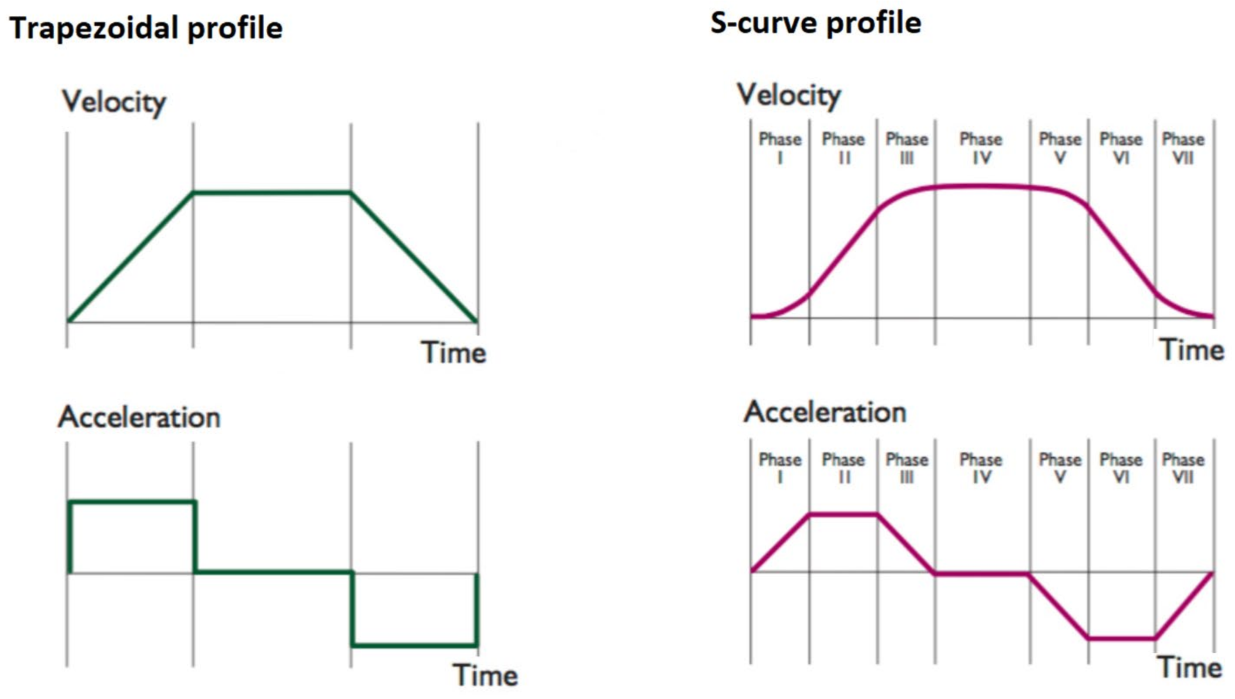

Fig A-12: Two examples of commanded motion profiles: trapezoidal velocity profile and S-curve velocity profile.

Servo Tuning:

The question of "servo tuning" is simply a question of finding the values for the parameters, i.e., the gains, of the servo control algorithm for position, speed and current loops: "what should be the numbers". The logic of the servo control algorithm is usually cannot be changed by the end user, just the parameters can be adjusted.

Assumptions:

1. We assume the servo drive current control loop is properly tuned.
2. We will assume the PID servo control algorithm, implemented either at the drive or at the controller, includes the five gains. As noted earlier, actual implementation differs for different controllers and applications, but every servo control algorithm has these components minimally and these five components largely determine the performance.

We now discuss how to determine by trial-and-error experimentation the appropriate values of these five gains. Note that there is not one unique combination of values of these gains. There are many possible combinations that will do the job for us in a reliable manner. Our aim to find the range of values which will work well, then choose a specific value from the acceptable range. In fact, if the servo drive software supports it, we may determine two or more sets of the tuning parameters, and use "Gain Switching" ("Gain Scheduling") to use each set-in different conditions that optimizes the performance.

In addition to the basic five gains discussed, the other components of the servo algorithm that can be tuned for each specific application include filters for vibration suppression, notch-filters, backlash compensation, friction compensation, gravity compensation.

Step 1: Determine K_P

Proportional gain determines the stiffness of the servo control loop. Think of it like a spring which connects a mass to ground. The stiffer the spring is, the smaller the displacement per disturbance force and faster the frequency of oscillations. Likewise, torque command generated by the servo control loop is proportional to the error, and the proportionality constant is the K_P, like a spring constant. The higher the K_P, the stiffer the servo loop behavior. If we make the $K_P$small, the servo loop would be soft. If we make the $K_P$large, the servo loop would be stiff. The larger the K_P, the faster the response, just like a stiff spring. However, there is a limit to how large we can make proportional gain K_P. If it it too large, eventually the servo loop would become unstable. There are two reasons for that:

1. Torque (current) limit: since the drive has a maximum current generation capability, there is saturation. No matter what we command, there is a maximum limit of current we can have. So, if the K_P is very large, then for very small positioning errors, the ($K_P * e$) will be large enough to reach maximum current. Hence, the servo system will almost always operate between $i_{max}$and -i_{max} (minimum and maximum) current. This means servo control algorithm output is like so called bang-bang control switching between two extreme values, not a smooth proportional control output. This nonlinear saturation behavior can lead to unstable closed loop behavior at worst, and very unacceptable dynamic response for the very least.
2. Inherent time delay (filtering effect) of the dynamic relationship between commanded current, actual current, commanded position and actual position. When there is time delay (or its equivalent effect due to dynamic (filter) behavior in closed loop), there is always a finite value of loop gain where the closed loop system goes unstable. Therefore, since a servo control loop has time delay like dynamic components (i.e., dynamic response of the current amplifier (drive)), the closed loop system will always go unstable if the gain is too large.

   Keeping these in mind, we can determine $K_P$to a value between two tests. Ideally, we would like to be able to make the gain $K_P$to be a function of error value, but most servo control algorithms do not accommodate that.

Test:

- Set all other gains to zero or maybe $K_d$to some non-zero value if the closed loop system goes unstable quickly without out some non-zero K_d.
- Command to maintain the position (commanded position is zero or current position, commanded speed and acceleration is zero). Using external torque/force, cause a small position error on the servo axis. Feel/measure the

current command servo loop generates in response. Feel/measure "the stiffness" of the servo loop. Slowly increase K_P. As you increase K_P, you should feel that servo system resists more for the same position error you cause.

- Increase $K_P$until the system starts to buzz and potentially go unstable. That is the maximum value of K_P you can choose, and most likely a value less than that.
- Decide on what is the maximum position error for which you want the servo loop to command maximum current in response.

$$I_{max} = K_P * e_{max}$$

We know I_{max} , we choose e_{max} , then calculate K_P.

Choose a K_P between close to or less than the value determined in the last two steps above.

Ideally, when we are making large position changes, we will use a smaller value of K_P. When we are about the desired steady state position (commanded speed is zero), we like to have large K_P. The large K_P we like to have in steady state position condition is typically too large for large motion tracking for it would cause a lot of saturation, hence nonlinear behavior, and possibly unstable closed loop dynamics. If the servo control algorithm does not accommodate K_P values as function of error or motion command, then one value must be chosen for the worst-case condition. A reasonable solution to that is to have a saturation value on the "position error" signal in the servo loop in order to have a large $K_P$value that would be effective in small errors and steady state, and during large motion the large $K_P$effect would be reduced due to "position error" saturation, hence avoid instability.

Step 2: Determine K_d

Velocity gain is equivalent the damping, like a suspension, in the system. Too little of it will lead to too many oscillations. Too much of it will unnecessarily slow down the dynamic response and if there is noise in the servo loop, it will lead to instability. Derivative gain in its ideal form makes the system more stable.

So, the idea is to start with small value of K_d. Increase it slowly until the servo loop starts to become too oscillatory and unstable. Back off from that value to a lower value that provides enough damping but far from being unstable.

Step 3: Determine K_i

Integral gain, K_i, helps with reducing the steady state error against disturbances. But it generally has negative effect on transient response and stability. Do this test while servo system holding a constant position. Introduce a constant disturbance torque. With K_i= 0, this will lead to a finite steady state position error. With K_i= 0, the steady state error will be made zero over time (assuming we don't have much of friction in the mechanical components of the motion transmission mechanism). The speed by which the steady state error will be made zero under constant disturbance depends on the value of K_i. We want that Ki value to be large enough so that the positioning error is zero as fast as possible, but not too large of K_i such that during transient motion and over time, K_i results in poor transient response and unstable closed loop behavior.

Also, if the servo algorithm logic supports it, limit the maximum integral control term contribution to the final control signal, enable/disable integral control depending on the motion condition, enable/disable integrator anti-windup feature of the algorithm.

Step 4: Determine K_{vf}

Feedforward speed and acceleration gains can improve the transient tracking accuracy. When commanded speed and accelerations are zero, i.e., final position command is reached, these terms have no contribution.

Decide what percentage of maximum current command you want generated for a given maximum expected speed command: i.e.

$$20\% * i_{max} = K_{vf} * V_{cmd\,max}$$

We know i_{max}. We choose percentage, i.e. 20%. We choose $V_{cmd\,max}$.

Then calculate K_{vf}.

Then adjust it by trial error during transient motions especially to improve the tracking accuracy.

Step 5: Determine K_{af}

Decide what percentage of maximum current command you want generated for a given maximum expected acceleraton command: i.e.

$$10\% * i_{max} = K_{af} * A_{cmd\,max}$$

We know i_{max}. We choose percentage, i.e. 10%. We choose $A_{cmd\,max}$.

Then calculate K_{af}.

Then adjust it by trial error during transient motions specifically to improve the tracking accuracy.

References

Cetin, S., Khandekar, F., Motion Control Software using CODESYS and EtherCAT, 2022.

Cetinkunt, S., Mechatronics with Experiments, John Wiley and Sons, 2012, Second Edition.

Tal, J., Motion Control by Microprocessors, 1984

Tal, J., Step-by-Step Design of Motion Control Systems, 1994

Servo Basics by Yaskawa: https://www.youtube.com/watch?v=Gzo9m0tMD0A&feature=emb_rel_pause

Drive Basic by Yaskawa: https://www.youtube.com/watch?v=3-cs4eEiBWo

Understanding PID Control: https://www.youtube.com/watch?v=wkfEZmsQqiA\

Servo Tuning by Yaskawa: https://www.youtube.com/watch?v=X3R_ixtGBZY