Coordinated Multi-Axis Motion Control via CAN bus.

Jan Bosteels, Advanced Motion Controls

Abstract: this paper describes a mechanism for multi-axis coordinated motion control via the CAN bus. The trajectory generation function is shared between the CAN host and CAN servo nodes. The position loop is closed in the CAN servo node. The trajectories of each axis are de-coupled, and all axes are synchronized via the CANopen Time Stamp message.

Introduction

Multi-axis coordinated motion control has traditionally been solved using dedicated multi-axis motion controllers, either PC based or standalone. Coordinated motion encompasses motion involving multiple axes, where there exists a relationship between the axes (for example to describe a circle, two axes must move in a sine-cosine relationship). Trajectory generation and position loop algorithms are implemented on a single hardware platform, due to the real-time requirements. Figure 1 shows a block diagram of this approach in case of a 2-axis system. The command signal is typically a +/-10V signal, and the feedback is a position signal, which can be generated by an encoder, resolver, potentiometer, etc.

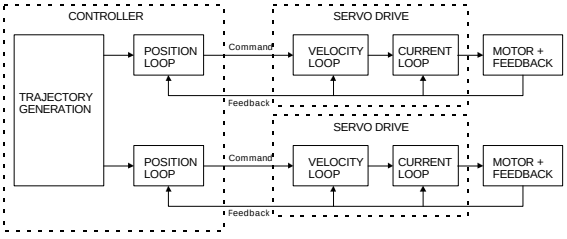

Figure 1

The servo drive can be of an analog or digital nature, and can operate in velocity or torque mode. The motor velocity or torque is proportional to the command signal. The servomotor can be a permanent magnet DC brushed motor or a permanent magnet DC or AC 3-phase brushless motor. The position loop algorithm is commonly a PID type servo algorithm. Although this approach works well in many applications, it exhibits some significant drawbacks:

- Wiring: all feedback wiring needs to be brought back to the controller. Since motors are physically mounted on the machine, and the controller and drives are typically mounted in an enclosure, low-level signals need to be wired over relatively long distances. This can create signal integrity problems, which can be catastrophic in a closed loop system as it may cause motor run-away. Cost of such wiring also tends to be fairly significant. The analog +/-10V command signal can have potential noise problems, resulting in servo loop jitter, which in turn affects accuracy.
- Flexibility: the addition of axes is difficult or impossible. With a fixed central hardware platform, scaling (changing the number of axes under control) is difficult or quasi impossible.
- Diagnostics: the controller-to-drive interface does not provide detailed drive diagnostics. Typically, a simple drive fault output is connected to the controller. The controller does not have access to drive parameters such as phase currents or voltages, or more detailed information about drive failures. Such extensive

diagnostics could support preventive maintenance or help troubleshoot problems more efficiently.

Networking

As digital networking was introduced into the domain of motion control, different approaches have been investigated to achieve coordinated motion. One approach has been to keep the trajectory generation and position loop algorithms in a central host controller, which sends digital torque or velocity commands via a high-speed serial link to a digital servo drive. The servo drive updates the controller with the actual motor position via that same serial interface. This approach basically replaces the +/-10V and position feedback interface with a high-speed, digital interface. Since the drive can be mounted in the direct vicinity of the motor, and since the drive typically needs the motor position for proper commutation (in case of 3-phase brushless motors), this provides an elegant wiring solution. Figure 2 shows the block diagram of such a system.

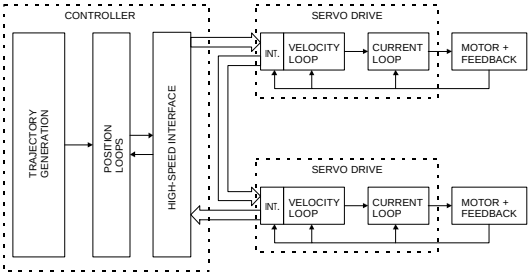

Figure 2

The drawback of this approach is that the serial connection between the controller and the drive needs to be relatively fast, which adds cost and complexity to both drive and controller. Some of the existing solutions are:

- SERCOS: optical fiber solution, running at 2, 4, or 16Mbit/s
- IEEE1394 (a.k.a. FireWire): copper wire or optical fiber solution running at 200, or 400 Mbit/s
- 100BaseT, 10BaseT (a.k.a. Ethernet): copper wire solution running at 10 or 100 Mbit/s

The actual communication protocol used with the above physical layers varies. Some protocols are open standard; some are manufacturer specific and thus proprietary. Position loop update rates are in the 5 to 10 kHz range, which means torque or velocity commands need to be sent, and actual positions received, at 200 to 100 μsec intervals. It is also important to maintain synchronization between all axes, which means transmission latencies need to be minimized. Also, since the drive is already of a digital nature, its resources are not fully used if it is only to control motor torque or velocity.

Distributed Approach

An alternative approach, suitable for many applications, is to move the position loop to the digital drive. In this new scenario, the controller sends position commands to the drives, which can be done at a lower rate (at least lower compared to position loop update rates). A more cost effective, lower speed serial link can be used. To further reduce the position command update rate, the drive could perform higher order interpolation. This means now that the servo drive has both a trajectory generation and position loop algorithm implementation. With this mechanism, the host controller would split the overall position trajectory in segments, and would send the segment information to the servo drive. The servo drives perform a higher order interpolation on the segment end point information. In essence, this means that the trajectory generation is shared between the host controller and the servo drive. Figure 3 shows a block diagram for a 2-axis system.

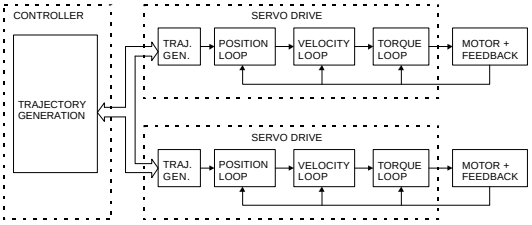

Figure 3

Segmentation

In a multi-axis system, the overall, multi-dimensional, trajectory can be decomposed into a position vs. time profile for each axis. The position profile for each axis can be split into segments, as Figure 4 shows.

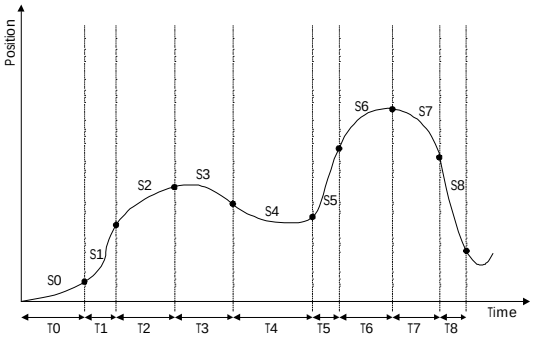

Figure 4

This segmentation is carried out in the controller. Each segment Si has an associated segment time Ti (i=0, 1, 2,…), which can vary between segments, but also between axes. Selection of the segment time is somewhat controller and application dependent. In addition, each segment has a segment end point position and velocity Pi and Vi (i=0, 1, 2, …). Note that the segment end point velocity is the slope of the position profile in that point (velocity is the first time derivative of position). Also note that the end point of one segment is the start point of the next segment. The position and velocity of the end point of one segment is the same as the position and velocity of the starting point of the next segment.

In case of a third order interpolation between segment start and end points, the position x(t) has the following format:

$$x(t) = a \cdot t^3 + b \cdot t^2 + c \cdot t + d$$

Given the start time T_s and end time T_e = T_s + Ti (where i=0, 1, 2, … is the segment number), the position and velocity of the segment start and end time,

$$x(T_s) = x_s$$

$$v(T_s) = \dot{x}(T_s) = \dot{x}_s$$

$$x(T_e) = x_e$$

$$v(T_e) = \dot{x}(T_e) = \dot{x}_e$$

, one can determine the 4 coefficients of the third order curve between the segment end points:

$$x_s = a \cdot T_s^3 + b \cdot T_s^2 + c \cdot T_s + d$$

$$\dot{x}_s = 3 \cdot a \cdot T_s^2 + 2 \cdot b \cdot T_s + c$$

$$x_e = a \cdot T_e^3 + b \cdot T_e^2 + c \cdot T_e + d$$

$$\dot{x}_e = 3 \cdot a \cdot T_e^2 + 2 \cdot b \cdot T_e + c$$

The 4 equations above allow solving for the four coefficients a, b, c, and d. This interpolation is performed in the servo drive, for each segment.

Through the segment time and end point position and velocity, the original position profile can be recreated with some degree of accuracy. The inaccuracies introduced by the third order interpolation can be established by comparing the original position profile and the profile resulting from the segmentation and third order interpolation of each segment. This inaccuracy can be reduced by careful selection of the segment time. Intuitively one can see that by reducing the segment time, the maximum error between the original profile and the profile generated in the drive can be reduced. However, this requires that segment information needs to be sent more frequently, which in turn presents a design trade-off.

The actual relationship between segment time and the error created by interpolation is mathematically very

complex. The example below will provide some further insight. Assume that the position profile is the following: $x(t) = \cos(wt)$; where $\omega = 2\pi/T$. T is the period of the cosine profile. Since most complex motion trajectories are based on straight lines and arc-segments, this example is quite relevant. Assume that T = 2 seconds, and that the profile is split in 10 segments. This would mean that the segment time is 200 milliseconds. It can be calculated that the worst-case error due to interpolation is approximately 0.04, or about 4%. If the profile is split in 20 segments, with a segment time of 100 milliseconds, the worst-case error is approximately 0.009, or about 0.9%.

From this example it can be seen that the error resulting from third order interpolation can be reduced dramatically by reducing the segment time. Notice however that the segment time in this example is still relatively large, compared to a typical position loop update rate. Final selection of the segment time is application dependent.

CAN Implementation

The above approach has been implemented utilizing the CAN bus and the CANopen communication protocol, see Figure 5. The maximum bit rate of CAN is 1Mbit/s. A CAN message frame can contain up to 8 bytes of data, and 127 nodes can be connected to a single CAN bus. The 8 data bytes of a single CAN message, used to send segment information, are assigned as follows:

- 3 bytes for the segment end point position (in position units)
- 3 bytes for the segment end point velocity (in position per second units)
- 1 byte for the segment time (1-255 milliseconds)
- 1 byte for an integrity counter (0-255, with roll-over)

The messages above are generated in the CAN host. The end point of one segment is obviously the start point of the next segment; therefore only segment end point information is required. It is always assumed that the first segment start point position and velocity are zero. The CAN servo node performs a third order interpolation, since segment start and end point position and velocity are known (to determine the four coefficients of a cubic curve, four equations are necessary). The segment time can vary, which improves accuracy in case of rapid position changes.

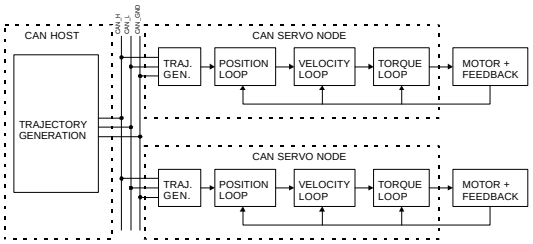

Figure 5

The integrity counter ensures that each node receives consecutive segments in the proper sequence. The above-described CAN messages are allocated a certain COB-ID range, within the CANopen communication protocol (as a matter of fact, they use COB-ID's within the receive PDO COB-ID range).

Each CAN servo node has a 15-level FIFO buffer, which receives the segment messages. This buffer can be further managed through standard CANopen objects. The CAN servo node automatically sends an error message in case:

- a CAN message with incorrect length was received
- the buffer is full
- the buffer is empty
- a non-consecutive counter value is detected.

Actual motion is started on all CAN servo nodes at the same time via a broadcast message, from the CAN host. Each CAN servo node removes segment messages from the buffer as needed (one at a time). The trajectory

generation of each segment is accomplished dynamically, on-the-fly.

CAN servo node synchronization is maintained through the CANopen TIME STAMP message, which is broadcast regularly by the CAN host, and is received by each CAN servo node at the exact same time. Each CAN servo node compares the time difference between consecutive TIME STAMP messages with its own time measurement. Internal time base changes are made dynamically, and thus synchronization between all axes is assured.

Selection of the segment time is based on:

- Number of axes
- CAN bus bit rate
- CAN bus load
- CAN host capability
- Application performance requirements

Under ideal circumstances, the CAN host would change the segment time dynamically, based on the position profile shape and curvature. Practically, it suffices to select a fixed segment time, which can be applied throughout the whole position profile.

Example:

At 1Mbit/s, an 8 data byte message, takes worst-case approximately 130 microseconds to transmit (130 bits total message length which is comprised of 64 data bits, CAN node address, CRC…). In a 4-axis system, the total time required to send a segment message to all 4 axes is approximately 4*130=520 microseconds. Additional time must be allotted for other messages on the CAN bus, such as: status messages, emergency messages, etc… Note that it is not necessary to send segment messages at rates typically found in centralized controllers (in the order of 100 – 500 microseconds). Since the CAN servo node performs cubic interpolation, the position update rates can be in the order of 10 – 200 milliseconds, depending on the application. In case of 10 millisecond segment times, the CAN busload, created by the segment message transmission is 520 microsecond/10 millisecond = 5%. One can see immediately that the relatively slow CAN bus can easily accommodate such rates and provide sufficient bandwidth for other messages.

Conclusion

As networking is introduced into the field of motion control, real “distributed” control is possible. It is important to allocate machine control functionality properly across “intelligent” devices on the network. By shifting the position loop to the servo node, and by “distributing” trajectory generation over host and servo node, a simpler, lower cost network can be utilized. Some further advantages of the above approach are:

- “Soft Motion”: the motion trajectory is generated in the PC or controller CPU. No dedicated motion control hardware is required, except for a network interface (in this case a CAN interface).
- Flexibility: the axis count can be changed more easily; no additional hardware is required on the host controller. Other devices such as I/O modules, sensors, etc, can also be connected to the CAN bus.
- Network Requirements: the network speed requirements are reduced through this “partial trajectory generation”. This allows selection of a more robust, less complex, and more cost-effective network (such as CAN).
- De-coupling: the position loop in the servo drive is de-coupled from the host controller. In a system with a centralized controller, the trajectory generation and position loop

algorithms share the same hardware. Increasing the number of axes typically requires reduction of the position loop update rate for each axis under control.

- Wiring: devices can be mounted on the machine, which dramatically reduces wiring cost and complexity, and enhances machine reliability.
- Diagnostics: the CAN bus interface, in conjunction with the CANopen protocol, provides full access to the servo node internal variables and state machine. This becomes increasingly important for remote diagnostics and field repair.

About the author:
Jan Bosteels has an electro-technical engineering degree from the University of Gent, Belgium. He is at present the digital product manager at Advanced Motion Controls, and has worked in the motion control industry in various positions for the last 10 years. He is also an active member of the IEEE.

Jan Bosteels
Advanced Motion Controls
3805 Calle Tecate
Camarillo, CA 93012
USA
Tel: (805) 389-1935
Fax: (805) 389-1165
E-mail: jbosteels@a-m-c.com
http://www.a-m-c.com