

Design Trends

A quarterly publication brought to you by Motion Designs Inc. November 2007

In this issue of Design Trends:

- Technology: stepper Versus brushless servopage 1
- New Product: K-SA Series from Arcus Technologypage 5
- Product Feature: understanding multi-tasking in CTC controllers.....page 6
- Application Solution: distributed motion controlpage 10

The “Stepper Versus Brushless Servo” myth...

There are many discussions and articles regarding the benefits of brushless servo motors over steppers and vice versa. Upon inspection it becomes clear however that this is more an apples to oranges comparison and not a true technology benchmark. Of course both motor types can be utilized as actuators in a motion control application. But if the motor type is properly selected based on the application requirements, there should be no doubt whether one or the other could have been a better choice.

One point that needs to be considered early on is that other than the physical motors themselves, the actual drive (or amplifier) electronics that are used with each motor type play a very large role in how both are compared. But regardless of the actual control mechanism itself (which will be a topic for future prose), it will be clear that the motors themselves are quite distinct and impose quite

different physical limitations (that simply can not be overcome by any kind of drive electronics).

Let us start by pointing out some of the commonalities between both motor types. Both generate torque (or force) based on the interaction between currents driven through stator windings and permanent magnets on the rotor. Also, both are “brush-less” meaning that there are no brushes and commutators inside the motor (in contrast with DC brush type motors).

As you will see, there are many more areas of difference. The first one is the number of poles (i.e. the number of electrical cycles per mechanical revolution). The most common stepper motor (1.8 degree, bifilar wound, 2-phase) has 50 poles, whereas brushless motors are typically 2, 4, 6, or 8 pole (some higher pole motors exist, but are

rare). Why is this important? The frequency of the stator currents is proportional to motor speed and pole count. For example, at 1,000 rpm the above mentioned stepper motor requires 833Hz currents. A typical 4-pole motor would require 33 Hz currents. This has a tremendous impact on the current control bandwidth requirements (which in turn is limited by available bus voltage and motor winding inductance). This automatically leads to a speed limitation, a topic we will get back to later on.

This pole count difference has another repercussion. A stepper motor is inherently a better “open loop” positioning device than a brushless motor. As a matter of fact, it is typically (and practically) impossible to control motor position of a brushless motor without the use of a feedback device (e.g. encoder, resolver...). A stepper motor on the other hand allows position control (with a resolution that is determined by the drive electronics), without the requirement of actually tuning a position control loop. Even if one was to attempt position control with a typical brushless motor without feedback, the positioning would be very coarse and quite unstable. Hence a feedback device is not optional in case of a brushless servo system (we ignore the concept of sensorless feedback for the moment as that would lead us to a control mechanism comparison, not just a motor comparison).

A next step is to take a look at some typical motors and compare them side-

by-side. A very typical size, common to both motor types is a motor with following dimensions:

- Nema23 flange (i.e. a 2.25” x 2.25” square or Ø3.18” round body)
- Motor length around 3 inches (excluding any feedback devices).

A very typical stepper motor, available from several manufacturers, has the following characteristics:

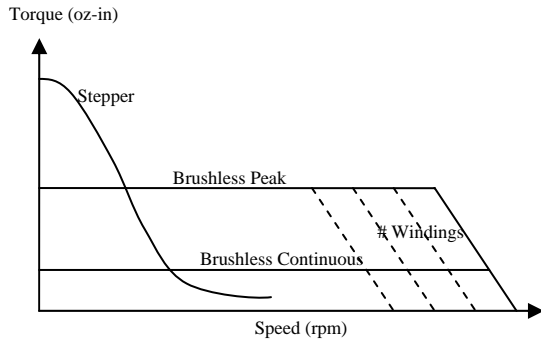
- Peak torque 240 oz-in @ 3Amperes
- 1 Ohm resistance
- 3.3 milliHenry inductance
- 0.006 oz-in-s² inertia
- 2.2 lbs. weight

A very typical 4-pole servo motor, with same overall size, has the following characteristics:

- 120 oz-in peak, 40 oz-in continuous torque
- Resistance range 0.15 – 10 Ohm
- Inductance range 0.10 – 10 milliHenry
- 0.0008 oz-in-s² inertia
- 1.1 lbs. weight

Brushless servo motors typically have a peak and continuous regime, hence the 2 torque figures. Also, they typically have multiple “windings” to choose from, hence the resistance and inductance ranges.

The torque-speed curves for these 2 motors are as follows:

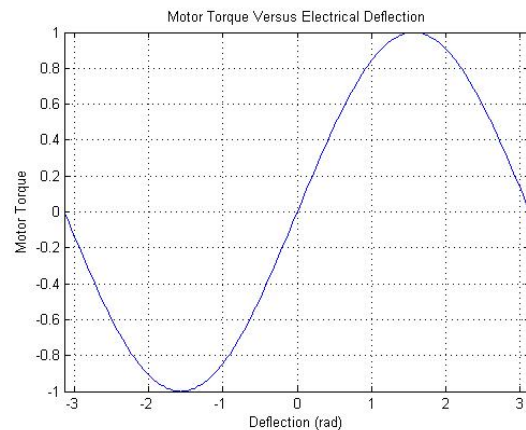


Note that the various brushless motor windings would require different voltage and current levels in order to achieve the higher speed curves (hence is not constant power!).


So one can immediately see that stepper motors pack more peak torque in the same overall volume (this does come at a cost as you will soon see). Actually, if you correlate the stepper motor torque and current specifications to a typical brushless motor torque constant (K_t in oz-in/A), you get 240 oz-in/3 Amperes = 80 oz-in/A. This translates to a K_e (voltage constant) of about 60 V/krpm. This furthermore exemplifies a speed limitation here: at 1,000 rpm, we need to overcome a 60V back-EMF. So clearly this design is more of a low speed and high torque design.

Also notice the difference in inertia between the 2 motors. The stepper motor has 7.5 times the brushless motor inertia. Combined with the peak torque rating this results in about a 3.5 times lower maximum acceleration. Also notice the overall weight difference (2:1). The higher inertia may actually be a benefit for the purpose of inertia matching, but more about that in future articles.

The above discussion of course prompts the question: why are both motors so different if they are both brushless motors. The reason for this is actually design intent. A stepper motor is designed to provide positioning capability without the need for feedback. There is no inherent “torque control”. By this we mean that motor torque is not an explicitly controlled variable, whereas with a brushless servo motor torque control is a fundamental requirement. In order for the stepper motor to be an open loop positioning device it needs to have adequate stiffness. By stiffness we mean the amount of motor torque per position error. A stepper motor creates torque only when there is a misalignment between the stator magnetic field (created by the stator currents) and the rotor permanent magnets as follows:



The deflection is in electrical angle. Since the stepper motor is a 50 pole motor, this torque profile repeats 50 times over one mechanical revolution. Only the section between $-\pi/2$ and $\pi/2$ is stable. Hence maximum torque is achieved at $-\pi/2$ and $\pi/2$ and stiffness is maximal at 0. This means that (stable) positioning accuracy is worst case $1/200^{\text{th}}$ of one revolution, or 1.8 degrees. Thus to improve this accuracy,



the motor is designed to provide as much torque as possible for a given current level, hence the extremely high K_e value. Please make sure not to confuse resolution and accuracy.

So in summary, stepper motors are, by design, quite different from brushless motors. In order to achieve good positioning capability, the motor stator windings are designed to provide a large torque-per-ampere value. This (and the

high pole count) in turn limits the maximum speed that can be achieved.

The choice between both motor types should be determined by the application and which motor attributes best match the requirements. Certainly, there exist applications where both may be a good fit. In such case, non-motor specific criteria, such as cost and implementation, may determine the final choice.

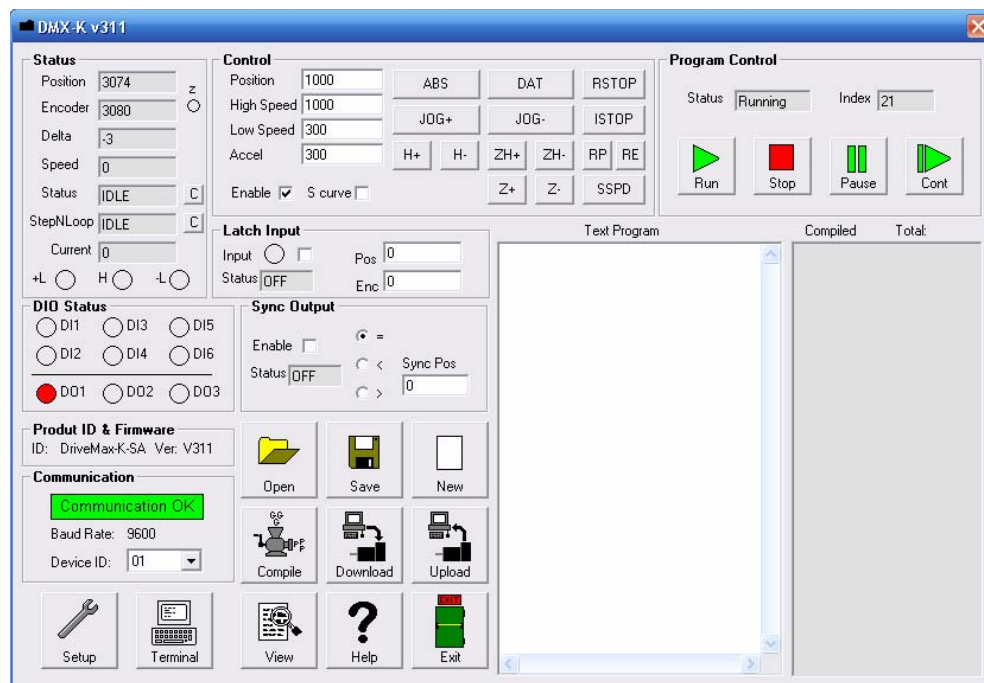
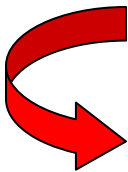
New K-SA Series from Arcus Technology.

Arcus Technology has released a new series of integrated stepper motor, driver and stand-alone controller. The K-SA Series is available in both a Nema17 and Nema23 package size and have the following common features:



- RS-485 or RS-232 ASCII communication
- Standalone programming using easy to use text based programming language
- Opto-isolated limit and home inputs
- Opto-isolated user configurable 6 digital inputs
- Opto-isolated user configurable 3 digital outputs
- High speed position capture function
- Position synchronized output function
- 1000 line incremental encoder (4000 counts/rev post-quadrature)
- StepNLoop closed-loop control
- S-curve and trapezoidal acceleration profile control
- Homing routine using home input, Z index encoder channel or both
- 16 micro-step driver with effective resolution of 3200 pulse/rev (with 1.8 degree motor)
- 12 to 35VDC supply
- Driver current from 100mA to 2.5A

Programming, setup and configuration can all be achieved via an easy to use Windows© based application:



Product Feature: Understanding multi-tasking in CTC controllers

Introduction

Since the late 1960's, the control of automated machinery has been accomplished via PLCs and their newer cousins, PACs. Although not as dominant as it once was, relay ladder logic (RLL) is still a widely used method for creating logic code in PLCs. The purpose of RLL was to provide a method to mimic the functionality of the electromechanical relays and their representation in wiring schematics as a "ladder"- diagram. The actual execution of such a diagram consists of a cyclical scanning of the entire diagram, evaluating the current state of all contacts and then determining which coils should be active at that moment. The contacts associated with the active coils will, on the next scan, assume their active state which may in turn affect the status of other coils, etc. In many modern high speed automation control applications, however, more than simple I/O sequencing is required. Today it is not uncommon to see PLC/PACs tightly

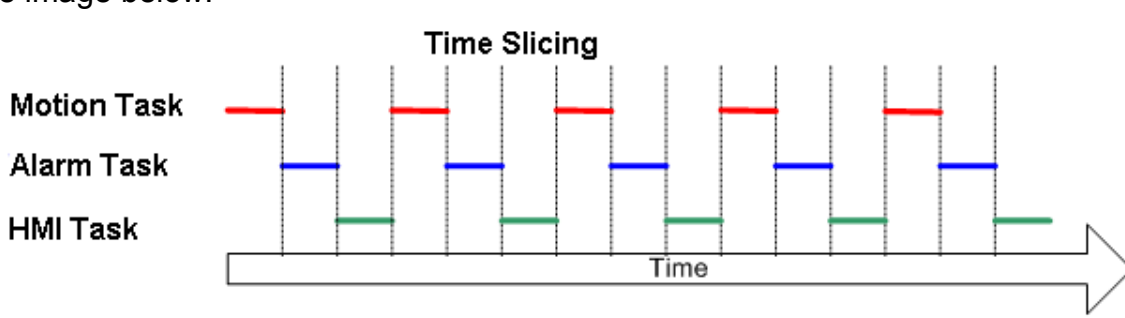
control digital I/O, analog devices, stepper and servo motors, interface with hardware or software HMI, collect/analyze data and interface with a SCADA package. As the quantity and complexity of the hardware grows in an automation environment, it becomes desirable to the code programmer to "break up" or modularize the operation of the machine or process into manageable groups or sequences, which correspond to actual functioning parts of the machine or process. Often these "sequences" or tasks must be executed and controlled simultaneously and asynchronously for the most efficient machine/process operation. To use RLL alone to combine these tasks into a single, one way execution of events can increase complexity of the code, make it more difficult to understand, and carry significant performance penalties in maximizing machine and code execution speed. Thus the need for "multi-tasking".

Definition

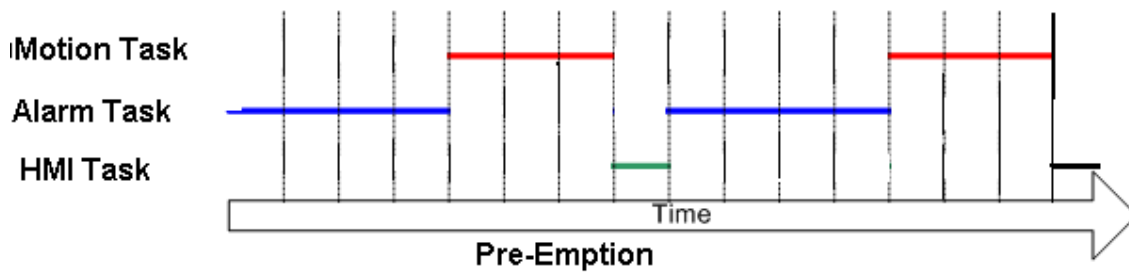
In its simplest form, multi-tasking can be thought of as the control of multiple instances of the sequences (tasks) that make up a machine or process running at the same time. Examples of concurrent machine tasks would be: alarm monitoring, data acquisition, motion control, servicing the HMI and general machine sequencing. Events such as these generally need to be executed in a quasi-parallel manner if a complex machine is going to operate at

the highest level of efficiency. Prioritizing and scheduling tasks is the job of one or more CPUs to insure that not only the code, but also the machine operates in the same manner in the same amount of time every cycle. This "repeatability" is called determinism. Generally, the use of multi-tasking by one or more CPUs to achieve deterministic machine control or process control is achieved in one of two ways: "Time Slicing" or "Pre-Emption".

A. Time Slicing : Based on the overall cycle time for the CPU to complete all its tasks, execute code and service resources, the scheduling of tasks such as alarm monitoring, motion control, etc. would be accomplished and serviced in equal amounts of CPU time. See image below:



B. Pre-Emption: In this form of multi-tasking, tasks are given priorities. The higher or more critical tasks not only take up more CPU time, but lower priority tasks can be "pre-empted or interrupted" to insure higher level tasks get the CPU time needed. See image below:



The choice of which type of multi-tasking to employ is based on a number of factors:

- Will there be one or multiple CPUs to manage the tasks?
- How large is the overall program code?
- What are the mechanical cycle time constraints if any on the machine?
- How fast can the CPU(s) execute specific program instructions in the code?
- Finally, how readily can the machine operation or process be broken up into logic functioning groups of code which mimic the actual flow?

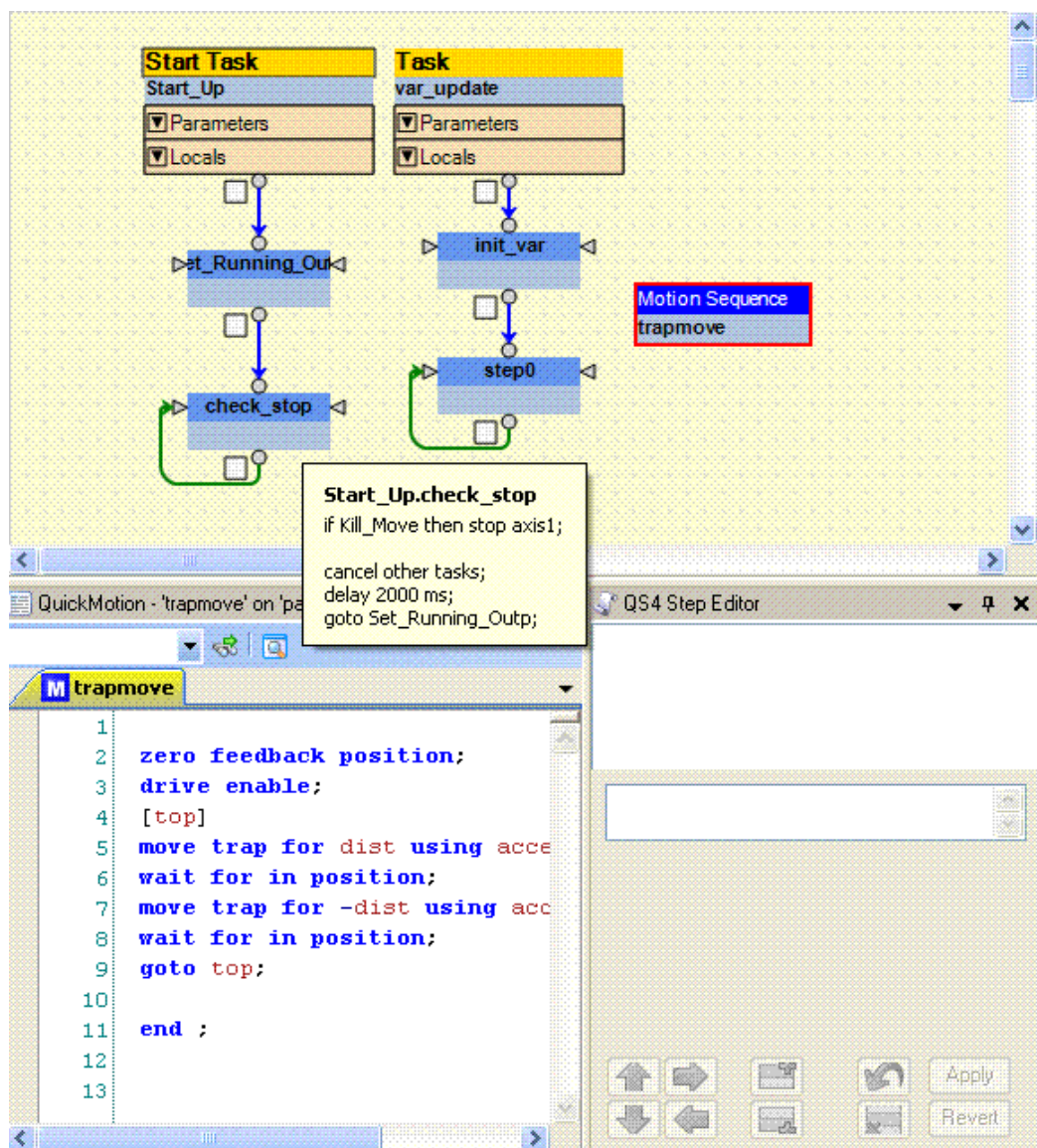
When these questions are thoughtfully answered and a multi-tasking approach to code development is chosen, the "main" program for a machine can become a short sequence of task calls, clearly representing the "overview" functionality of the machine as in a flow chart or function block diagram. This development approach has distinct advantages discussed below.


Advantages

A) *Modular approach to developing and maintaining program code*: multitasking can have usefulness beyond the simple creation of parallel, asynchronous tasks. It encourages and allows for programs to be organized into intuitive, modular groupings. These groupings or tasks become the main programming unit. Multitasking also encourages a "top-down" program flow, where the complex and detailed control code is at first hidden from view. To view the details of a given section of code, you move to the

appropriate task and the sequence of events for that portion of the program is stated in a clear and concise manner. This makes the program easier to understand, and acts as a “search tool” for finding the part of a complex program you wish to examine or modify.

An excellent example of this is Control Technology's QuickBuilder Development Environment for the 5300 series controllers. In the case of CTC, both the hardware and software are multi-tasking. In CTC's flow diagram below, it is easy to see two tasks running in parallel as the main routine. One can also view the lower level code for the motion sequence which is launched from the first task. In addition, the dialogue box contains the lower level code for the last step of the first task, which runs in parallel with the motion sequence continuously checking for conditions to halt the motion. Not only is it easy to understand the logic flow, but it can be “self documenting” if the task names and resources used are given intuitive names for what they represent on the machine.





B) *Faster execution and boot up times*: This is best illustrated with an example. Let's assume you want to monitor and update status on some analog or digital I/O during a move being made by a servo axis. If you work in a sequentially scanned program environment, generally the move would be commanded, some command would be issued to determine when the move was complete and then the I/O would be scanned. This entire sequence would not continue until the next loop cycle. Within a multi-tasking environment, there is no "waiting". The motion can be launched in parallel to the monitoring of the I/O. Thus two smaller control loops would run simultaneously instead of one larger one running sequentially.

C) *Simplicity*: multi-tasking in program development can be organized in a manner that most of a machine's functionality is embedded in tasks. Each "intuitive" task not only can correspond to a "mechanical module" or part of the machine, but also can be saved in a library of code "building blocks" for future use. An automation program then becomes easy to maintain and modify.

In summary, multi-tasking provides some clear benefits in overall machine control programming by organizing the code in tasks that can run in parallel. This in turn can shorten overall program processing and increase overall machine throughput. Although multi-tasking itself can be complex and intimidating (especially with a large task count and strict timing requirements), the CTC QuickBuilder graphical programming environment simplifies the actual implementation and provides the user with an intuitive yet powerful set of tools.

Application Solution: distributed motion control

The manufacturer of a medical imaging machine was in the process of developing a fairly large geometry imaging system, utilizing a centralized control topology. The motion control architecture consisted of an 8-axis motion controller and a PC running a real-time capable operating system. Some of the key requirements of the motion control system included:

- 7 axis of servo control
- Complex multi-axis path planning, including inverse kinematics due to non-linear motor versus load position relationship
- Reliability and safety due to direct patient interaction
- Scalable to accommodate different machine platforms
- Modular and expandable to accommodate additional axis and/or I/O
- Serviceability

It became clear after the first prototype system that the centralized controller topology presented some significant drawbacks:

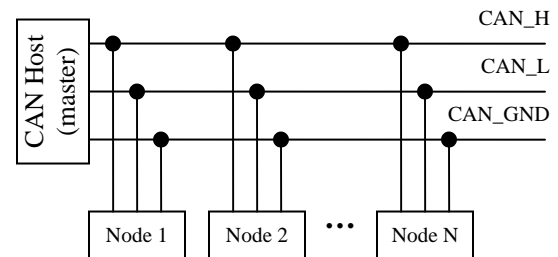
- Not scalable to the number of axis
- Not scalable to I/O
- Significant wiring cost and severely impacted reliability due to need to bring all motor feedback to a single control point
- Highly proprietary multi-axis controller
- Poor troubleshooting and diagnostics due to low level controller-to-drive interface (+/-10V)

To address these issues and also further enhance the control capabilities, a distributed topology utilizing CANopen as an embedded machine network was

considered. A single CANopen network was constructed as follows:

- CAN-interface card in the PC with proper drivers for the OS
- CAN network configured for 250kbits/s baud rate
- 7 CANopen servo nodes mounted within 1-2 feet of the motors
- 3 CANopen I/O modules for general purpose machine I/O

CANopen is an open standard protocol that is used in conjunction with the CAN physical layer network. CAN technology has been around for 20+ years. 100's of millions of CAN nodes are in operation in applications ranging from automotive (as in-vehicle network) to general factory automation (as an embedded machine network).



This architecture addressed the above mentioned issues as follows:

- Fully scalable: servo drives and I/O modules can be added or removed depending on the machine platform, without affecting the host hardware
- Motor and I/O related wiring is kept to a minimum due to proximity between drive and motor
- Avoidance of proprietary technology through the use of an open standard network protocol (CANopen)
- Extensive and detailed diagnostics through the CANopen interface

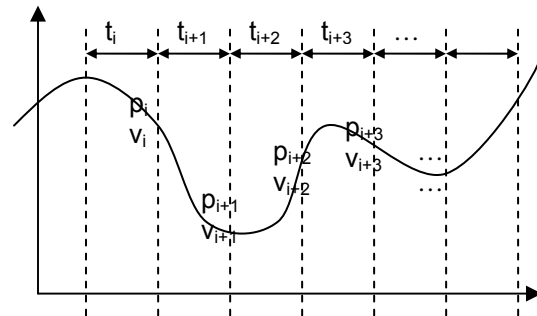
The CANopen servo drives from **Advanced Motion Controls** provide the following tasks:

- Close all servo loops (current, velocity and position)
- Perform homing routine
- Perform point-to-point positioning
- Perform 3rd order interpolation to achieve complex coordinated motion, provide a FIFO buffer for incoming set points
- Monitor and respond to all motion and axis related events
- Provide host with status on all pertinent variables

Most of the functionality can be handled by each servo drive on a per axis basis through a very efficient communication mechanism in CANopen called Process Data Objects (PDO's). This communication mechanism provides an interrupt driven messaging scheme where a node transmits a message based on specific criteria (e.g. value changed or amount of time elapsed). This way bandwidth-consuming polling is avoided. The coordinated motion requirement however requires a tight coupling between various axes. The combination of the PVT (Position-Velocity-Time) concept and the CANopen Time Stamp message allows realization of this requirement, while keeping CAN bus traffic to a minimum.

The PVT mechanism is a method that partitions a general position profile in segments defined by their end position, end velocity and segment time. This segment data can be sent down efficiently over the CAN bus to each servo node in a single package. Each servo node then "recreates" the original position profile by applying a 3rd order interpolation for each segment. The


"fidelity" of 3rd order interpolation is surprisingly high, even for relatively large segment time values. In the specific case described here, PVT points are generated with a 50 millisecond segment time, with no noticeable loss of position accuracy.



The PVT mechanism described above has one side effect that must be addressed in order to maintain synchronization between all axes. Since the overall (multi-axis) path planning is de-composed in individual axis position profiles, some form of synchronization must be ensured between axes. For example, if one were to describe a circle in an X-Y plane, by creating a sine and cosine position profile on each axis, loss of synchronization would result in an elliptic shape. CANopen provides a Time Stamp message that is broadcast to all nodes on the network, and contains a 48-bit time value. Since all nodes receive this message at the same time, a common time basis can be maintained on all nodes, and hence synchronization can be maintained indefinitely.

The end result thus is a CANopen network, running at a modest 250kbits/s baud rate, with a total of 10 nodes. The overall communication bus load is around 50%, and includes:

- 50 millisecc PVT messages for 7 servo nodes

- 
- Motor position reporting on all axis based on time-elapsed or position change
 - I/O reporting based on state change
 - General node status reporting based on status change
 - Fast local event handling with reporting back to the host

A few additional “bonus” benefits of this architecture are:

- Ability to make configuration changes on-the-fly
- Ability to update node firmware over the CANopen network
- Ability to diagnose a system, all the way down to the motor and sensor wiring.

For more information about any of the above topics or general questions or comments, please contact us:



Motion Designs is a technical sales and engineering company with extensive machine and motion control experience. We work with some of the best manufacturers in the industry as witnessed by our present line card:

- www.a-m-c.com : Advanced Motion Controls is a leading servo amplifier and drive manufacturer.
- www.arcus-technology.com: Arcus Technology manufactures stepper motor, drive and controller technology, providing USB, Ethernet and Mod-Bus connectivity.
- www.ctc-control.com : Control Technology is a leader in automation control technology with extensive network connectivity capabilities.
- www.esd-electronics.us : ESD Electronics is a leading manufacturer of CAN based hardware and software components, geared towards industrial equipment.
- www.magmotor.com : MagMotor is a leading servo motor manufacturer.

