Chameleon Motion Controller Speeds Machine Upgrades

J. Randolph Andrews
Douloi Automation, Inc.
740 Camden Avenue Suite B
Campbell, CA 95008-4102
(408) 374-6322

Paper Presented at the
1997 Incremental Motion
Control Systems and
Devices Symposium

Abstract

Most designs for automatic machines do not begin with a clean sheet of paper. Previous versions of an automatic machine represent investments of mechanical, electrical, and particularly software engineering. Minimizing changes while pursuing machine improvements reduces engineering rework and relearning, retains a track record of proven performance, and reduces time to market. How can an improved motion control system be placed into an already present machine architecture with minimum system impact?

A motion control approach is described which has a chameleon-like ability to conform to the surrounding system. For example, the chameleon can be configured to electrically impersonate a predecessor controller's hardware interface as well as software command set while delivering improved controller performance and value. Engineering investment, including staff learning curve for a new controller, is reduced saving time and money.

Introduction

The objective is to reduce machine deployment time by minimizing system impact of the motion controller. The motion controller in an automatic machine might be regarded as a puzzle piece that fits into a total machine architecture. The controller puzzle piece has relationships with the various elements around it and with the developers who use it. On one side are the attributes that define the user's view of the controller. The user view includes elements such as:

- Machine structure
- Controller Command Set
- Communication Hardware

On the other side are controller attributes that relate to the machine's view of the controller. The machine view includes:

- Amplifier/Driver control signals
- Position encoders, sensors, and discrete I/O

The goal is to have the motion controller accommodate preexisting user and machine views. This is particularly important when there is significant engineering investment behind the creation of these views. If the controller cannot display a chameleon like behavior to blend into the existing system, the user must introduce additional engineering effort to bridge any gaps.

Considering the possible risk and expense of changes, why are changes pursued? Changes are made to:

- Improve system performance
- Change motor type/technology
- Reduce Costs
- Move from in-house to off-the-shelf solutions

Approach

One approach to support different motion controller roles is to mimic a Swiss army knife and have built-in to the controller all possible methods of use. This approach is not realistic as there are as many different methods of use as prospective users.

An alternate approach is to have a basic "native" control system which is augmented by resident application programs. These application programs can replace or supplement native control functions to create new capabilities.

The "native" system constitutes the controller behavior as it comes "out of the box". Native components include:

Encoder Hardware

Description: Physical electronics that reads position sensors and reports motor position.
Inbound: Electrical signals from position sensors
Outbound: Actual motor position

Communication Hardware

Description: Physical hardware that electrically connects host and motion control card.
Inbound: Electrical signals from host
Outbound: Raw data available to motion controller

Command Interpreters

Description: Software component that establishes the grammar of communication, the letters, words, or codes that comprise the command set.
Inbound - Raw data from communication hardware
Outbound - Specific function requests

Profilers and Coordination

Description: Converts abstract motion commands into detailed trajectories indicating desired or "commanded" motor positions every controller sample period, i.e. every 500 microseconds.
Inbound: Abstract movement commands
Outbound: Motor commanded positions presented at sample rate intervals

Control Laws

Description: Calculations that use profiler information and discrepancy between actual and commanded positions to produce corrective action.
Inbound: Profiler commanded position, actual motor position, and possibly additional profiler information
Outbound: Torque request

Amplifier Hardware

Description: Physical hardware which changes abstract torque request into physical signals that direct the servo amplifier.
Inbound: Torque request
Outbound: Voltage and possibly other related signals

Some of the native components are hardware components and some are software components. Information generally flows through native motion controller components in the order indicated by Figure 1.

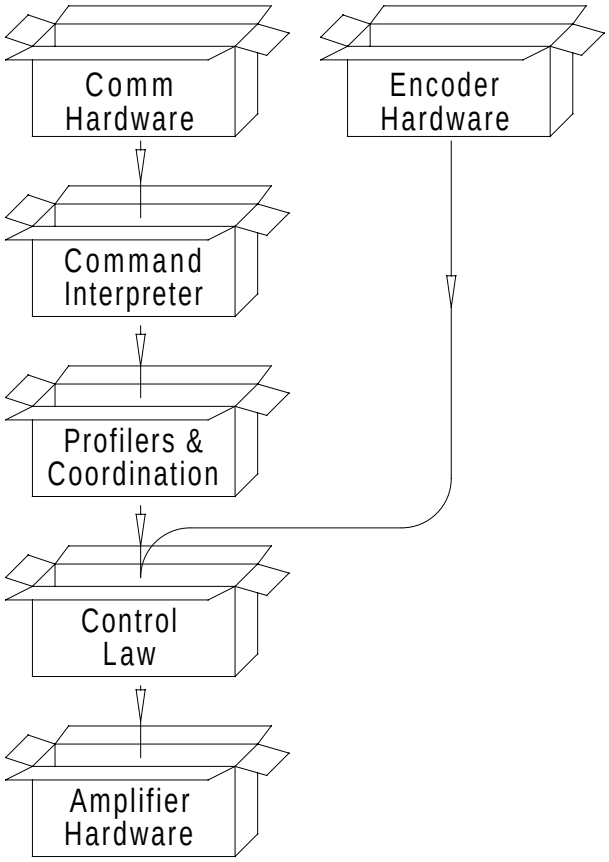

Figure 1. Native Information Flow

The figures are open boxes indicating functions that are available "out of the box". Note that the lines connecting boxes might represent multiaxis vectors of information, not just a single axis. In a motion controller there may be several such chains controlling independent mechanisms.

Adjacent to the native system is a user application software level. This user level allows components to be constructed that operate at sample-rate speeds and in sync with the motion control native system. User level components are represented as boxes with rounded corners that are fashioned for an application specific need. These user level components can be used to alter normal information flow through the native level. Figure 2 shows an example user component which is replacing the native profiler with a user level profiler.

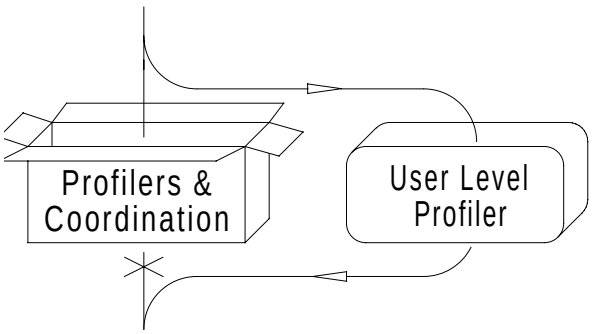

Figure 2. User Level Component

In additional to user-level software components the physical hardware itself may need to change to accommodate the system. Field Programmable Gate Arrays (FPGAs) are a very flexible means of reconfiguring hardware to meet application specific needs. A new hardware structure can be "downloaded" into a standard motion controller product to realize a "hardware special" that solves an application specific problem. In a manner similar to user components in software, the bulk of the hardware remains unchanged while one section is spliced or substituted with new application specific behavior.

The ability to synthesize new hardware functions will most likely not be in end-user hands for several years due to the software tools and background needed to describe the new hardware. However a motion controller vendor is in a good position to rapidly respond with an application specific FPGA design change.

Topologies

Multiple user components can be present in a system. User components can have various relationships to native components.

Substitution

Substitution replaces a native control component with a user level component. Information flow connections are made on each side of the new user component. Figure 2 is an example of a substitution. All of the remaining components of the native control system continue to operate as normal. Only a particular native component has been changed. This helps localize change and minimizes the amount of work that must be done to realize a different controller behavior.

Splicing

Splicing a user box into the information flow of the native controller augments or changes the information in some manner but continues to use all of the native elements. A user component can redirect information from the native system, modify it, and replace it in the original flow.

Concatenation/Redirection

In some cases it is beneficial to have user components create connections between independent motion groups to realize some concatenated, or "compound" behavior.

Examples

The following examples illustrate different user and machine views and how user components augment the native controller capabilities to provide these views.

New Amplifier Format

Servo amplifiers are commonly driven by +/- 10 volt signals representing the desired motor current. However some amplifiers have different formats. A chip-level amplifier, for example, has a current magnitude signal and a current direction signal as a digital input. What should be done to accommodate this amplifier?

A typical solution is to create an additional piece of electronics that presents the absolute value of the conventional +/- 10 volt signal as well as the direction based on comparing the original voltage to 0 volts. However this represents additional engineering work and creates a packaging problem. Where is this new piece of electronics placed?

An alternative solution is to create a user component that performs the absolute magnitude and direction calculation in software rather than hardware. This is done by splicing the calculation in between the control law and the amplifier hardware on the motion control card as shown in Figure 3.

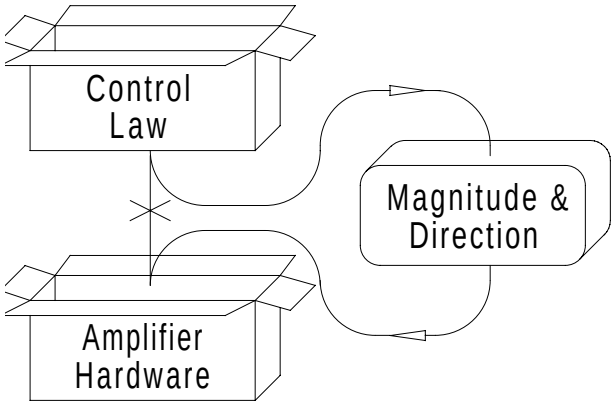

Figure 3. Magnitude/Direction Amplifier Splice

The user software component that performs this splice for the X axis is shown below.

```
Procedure ConvertAmplifierSignals;
  begin
  while true do
    begin
    yield;
    XAxis.SetDac(
      abs(XAxis.CommandedTorque));

    if XAxis.CommandedTorque>0 then
      XAxis.SetCompareBit(true)
    else
      XAxis.SetCompareBit(false);
    end;
  end;
```

The procedure is started as an independent activity and performs an ongoing loop. The first command inside the loop is "yield" which causes the procedure to wait until the next controller sample period. The absolute value of the X axis commanded torque is then assigned to the X axis digital to analog converter. The compare bit, an output bit available for the axis, is set to true if the commanded torque is greater than 0 and false if the commanded torque is less than 0. The magnitude/direction function has been realized in software.

A different kind of amplifier problem is brushless commutation. Here, torque amplitude information must be distributed to multiple motor coils based on motor position through a trigonometric calculation. This can be performed by a user component that makes use of additional DACs on the controller card to provide the additional analog channels for the brushless motor. This technique requires that the software sample rate be much faster than the commutation frequency. This is commonly true in linear brushless motors but might not be true for higher speed rotational brushless motors.

Control Law Substitutions

A conspicuous candidate for change in the native system is the motion control law itself. The native control law is a PID filter. Alternate filters can be built and implemented to realize improved performance for a particular application or to emulate a predecessor controller. Replacing the control law is a substitution as shown in Figure 4.

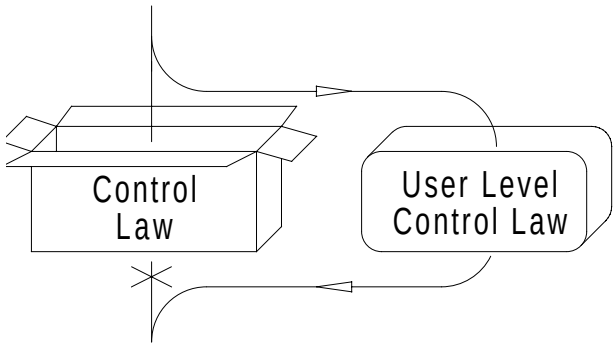

Figure 4. User Level Control Law

Control Law Extensions

Rather than replace the control law it might be desirable to augment it instead. The most common augmentation is related to feed-forward techniques. Here the native PID law calculates desired motor torque. Added to this torque, after the control law, is additional torque based on the motion profile or motion of other axes that have known geometric and dynamic coupling to the controlled axis.

Different Communications

Motion controllers often receive commands from a host PC computer. How these commands are provided vary. Even for commands being sent through the host computer's ISA bus there are various methods including IO registers, IO FIFOs, and dual-port memory.

Through the use of FPGA hardware connected to the ISA bus, the motion controller's communication interface can be altered to these various forms. Along with the new hardware is most likely a new command structure and method of sending information. A user component can be built to implement this new command set which is received through the new command hardware. This is shown in Figure 5.

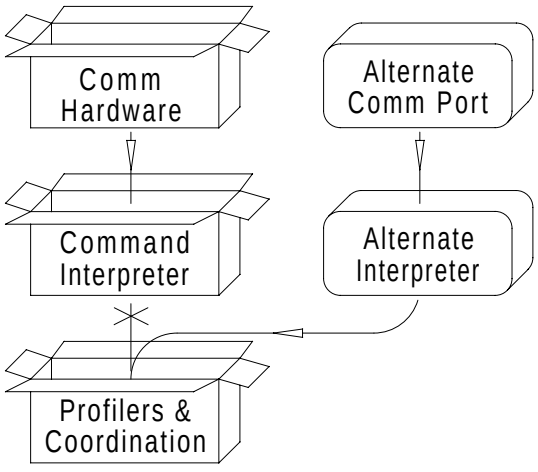

Figure 5. Alternate Communication

Kinematics

Most multiaxis motion controllers are designed for connection to Cartesian style machines. Coordinated "vector" style motion produces straight lines in a Cartesian mechanism.

Mechanical aspects of applications may make other mechanism geometries advantageous, such as the popular "SCARA" form or mechanisms that use pivoting, rather than translating members.

Even though the mechanism is not Cartesian it is desirable to present a Cartesian user view. This can be done by placing kinematic equations into the native motion controller. The simplest place to put the kinematics is in between the profiler and the control law of a Cartesian coordinated group. The kinematics transforms the sample rate Cartesian commanded positions into joint positions. These joint positions then flow into the control law. This splice is illustrated in Figure 6.

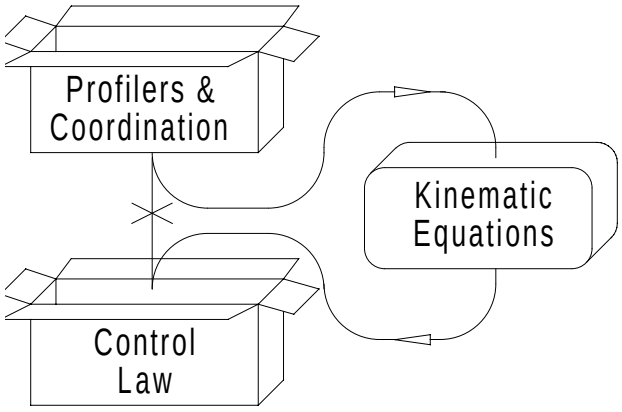

Figure 6. Kinematic Splice

This technique is most easily implemented by performing the kinematics at the controller sample rate. Fortunately hardware floating point coprocessors available in modern motion controllers make real-time kinematic calculations convenient.

A kinematic example is shown in the following program listing.

```
Procedure PerformKinematics;

  var C,J:T2SingleVectors;

  begin
  while true do
    begin
    yield;
    C.Init(
      Robot.XAxis.CommandedPosition,
      Robot.YAxis.CommandedPosition);

    CartesianToJoint(C,J);

    Mechanism.SetCommandedPosition(
      J.LongintX,J.LongintY);
    end;
  end;
```

As in the amplifier splice, the software is performing an ongoing loop. The first step is again "yield" which waits for the next sample period. "Robot" is the name of the axis group being directed in Cartesian space. The current position of Robot is collected. A procedure is called, CartesianToJoint, which maps the positions into joint space through geometry. The commanded positions of the mechanism joints are then set. In the current embodiment, commanded positions cannot be both read and set for the same axis group, so two different axis groups are involved.

Conveyor Tracking

Conveyor tracking is the ability to perform a motion operation, such as acquiring a part of painting a surface, on an object which is not stationary but moving relative to the mechanism. In common practice moving objects are on a translating conveyor. The mechanism's operation travels along with the part on the conveyor.

Conveyor tracking can be implemented by combining the motion of two separate native coordinated groups. One group is performing the machine operation in a stationary frame as if the subject part was not moving. The second coordinated group performs just the translation motion. The user component combines the sample-rate generated motor positions from each of these two sources to constitute a new, superimposed motion that is expressed through the physical target mechanism. This is illustrated in Figure 7.

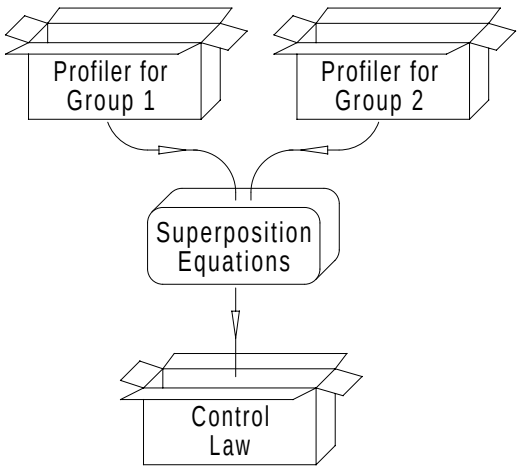

Figure 7. Conveyor Tracking Superposition

This conveyor tracking example is similar to the kinematic example except there are two profiler sources instead of just one feeding the user level component. The combined result is analogous to having the mechanism mounted on top of a moving platform.

The user component that performs this conveyor tracking superposition is shown below:

```
Procedure PerformConveyorTracking;
  begin
  while true do
    begin
    yield;

    Mechanism.SetCommandedPosition(

      Robot.XAxis.CommandedPosition +
      Conveyor.XAxis.CommandedPosition,

      Robot.YAxis.CommandedPosition +
      Conveyor.YAxis.CommandedPosition
    );
    end;
  end;
```

In this software component the Robot is performing the basic operation. The conveyor, a separate coordinated group from the robot, is independently being directed to follow the physical conveyor. Performing conveyor tracking involves adding these two motions together. This is done by setting the mechanisms position to be the sum of the X and Y movements of the two axis groups contributing to the motion. This addition is done every controller sample period. Because the motion is composed of profiled contributions, the super-imposed result is profiled and smooth.

Space and Time Filters

It is sometimes necessary to perform smoothing and filtering operations on motion. Depending on the needs of the application this may occur in one of several places. One opportunity to smooth a curved trajectory is to splice spline calculations in between the commands directing motion and the profiler. A small number of vectors expands to a higher number of finer resolution vectors which is submitted to the profiler for constant speed motion along the curve. This is performing spatial filtering, the removal of high "frequency" corners from a curved path, as shown in Figure 8.

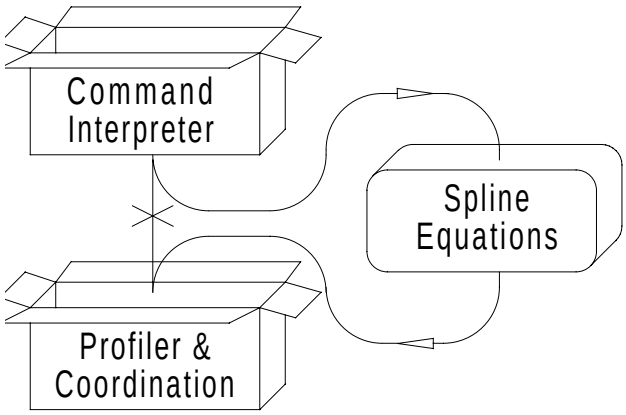

Figure 8. Spline Splice Before Profiler

An alternative place to perform smoothing is after the profiler and before the control law, as shown in Figure 9. These filters are causal in time as compared to the previous spatial, a-causal approach. Software filters can also be applied to noisy input signals to eliminate false triggers.

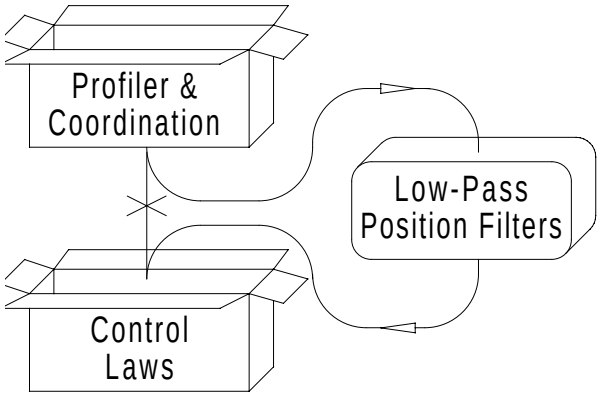

Figure 9. Position Filter Splice After Profiler

Total Replacement

For the hard-to-satisfy user there is always the possibility of Total Replacement as shown in Figure 10.

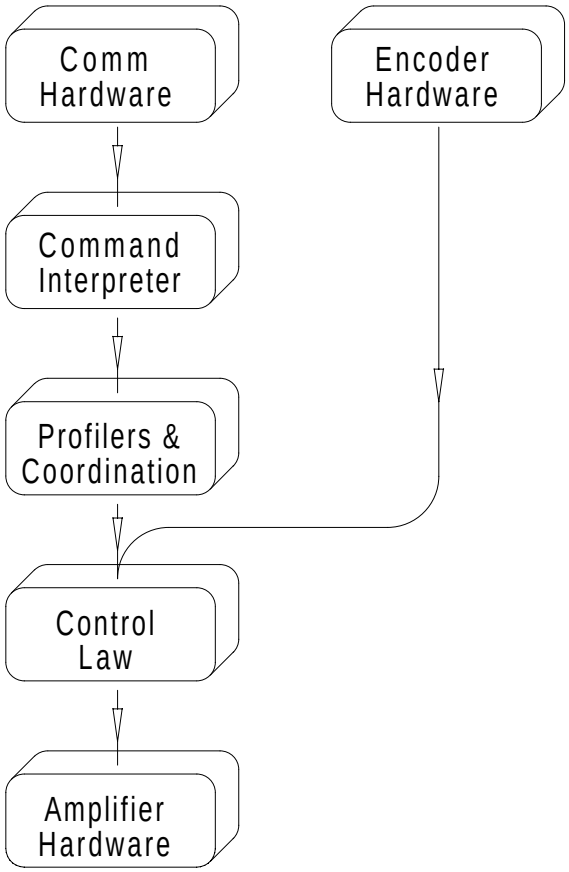

Figure 10. Total Replacement by User Level

In this case, no native components are used at all except for the software real-time framework hosting both native and user level components. Here all of the elements of the control system are provided at the application level and run at normal controller sample rate speeds. Because no native level components are being used this requires the greatest engineering contribution from the user. However this approach also affords the most flexibility.

Chameleon Controller Attributes

In order to realize these chameleon style system changes certain motion controller attributes are required. The user has to be able to write real-time application programs that operate at native sample-rate speeds. These programs must be able to coexist in a multithreading environment so as to make a contribution but not interfere with other software activities going on in the controller.

Making these types of changes has to be easy. Motion control users can't be expected to dive into deep and complicated motion controller internals. The native system must present opportunities for the user components to connect, and software tools must be available to make the job easy for the user, not just the motion control developer.

Creating these types of components requires having a clear view of the result. The software tools must support construction of diagnostic instruments and methods of studying and recording what is happening in the motion controller. Data collection must be done in a non-intrusive manner so that user components can be debugged during normal machine operation.

Summary

If each component in an automation system is functionally rigid and inflexible, a large amount of responsibility falls on the integrator to fill in the gaps and glue together the various automation pieces. Having a controller than can accommodate system variations, fill-in the gaps and conform to the problem at hand reduces the total engineering expense.

Bibliography

1) Andrews, J. Randolph: "Motion Server - A Next Generation Motion Controller Architecture", In *Proceedings of the Twenty Fifth Annual Symposium on Incremental Motion Control Systems and Devices, San Jose, CA,* 1996.

2) Andrews, J. Randolph: "Advanced Motion Solutions Using Simple Superposition Technique", In *Proceedings of the Twenty Third Annual Symposium on Incremental Motion Control Systems and Devices, San Jose, CA,* 1994.

3) Cox, Brad: *Object Oriented Programming: An Evolutionary Approach*, Addison-Wesley Publishing, 1986, 1991

4) Ellis, George: *Control System Design Guide*, Academic Press, San Diego, 1991

5) Franklin, Gene & Powell, David: *Digital Control of Dynamic Systems*, Addison-Wesley, Massachusetts, 1981

6) Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison Wesley Publishing, Massachusetts, 1995

7) Meyer, Bertrand: *Object-oriented Software Construction*, Prentice Hall, New York, 1988