

# Advanced Motion Solutions Using Simple Superposition Technique

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



Paper presented at the  
1994 Incremental Motion  
Control Systems and  
Devices Symposium  
Copyright © 1994 Douloi Automation

## Abstract

An important strategy for addressing complexity is to decompose a problem into smaller pieces, solve the pieces, and recombine the answers to yield a solution. A simple yet powerful superposition technique is presented which uses this strategy to solve several types of advanced motion control problems. Multiple motion related descriptions are used on separate aspects of a problem to independently solve each aspect. These descriptions are then combined in real-time to yield a resultant motion which solves the combined problem. Three technique examples are presented including acceleration-bounded continuous path motion, backlash compensation, and satellite antenna pointing from a moving reference frame. The breadth of the example applications illustrates the general usefulness of this superposition technique. Motion controller requirements for accommodating superposition are discussed.

The equations which transform between Cartesian and joint space are called kinematic equations. Kinematic equations are generally single vector input, single vector output, closed form expressions.

To produce joint position time history we can command an imaginary Cartesian mechanism in Cartesian space. The time history of this imaginary mechanism is transformed by the kinematic equations into the corresponding joint position time histories of the desired Cartesian path. In this paper such an imaginary mechanism is called a "virtual" mechanism composed of "virtual axes". Virtual axes have the same motion profiling behavior as conventional servo axes however they have no physical motor attached to them. Some motion controller manufacturers call these axes "phantom axis". The motion control system now looks like figure 2.

## Introduction

### Movement Transformations

A common technique used in robotics to generate motion control trajectories involves describing a problem in a preferred space, for example Cartesian space, and transforming positions to the corresponding mechanism joint positions. This idea is represented by figure 1.

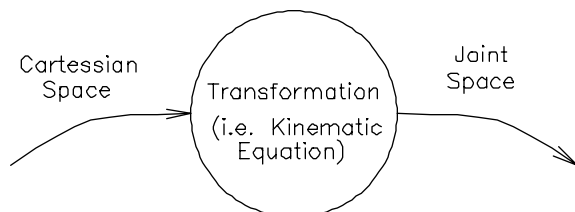


Figure 1. Single Input-Single Output Transformation

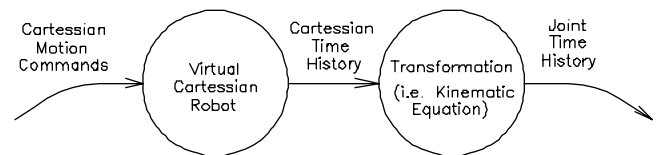


Figure 2. Virtual Mechanism Contribution

Movement commands come into the virtual Cartesian robot, time history of Cartesian position flows through the transformation to become joint positions for the actual mechanism.

### Multiple-Input, Single-Output Transformations

The single-input, single-output form of this transformation is useful. What additional benefits are gained by having multiple inputs? Figure 3 illustrates the concept.

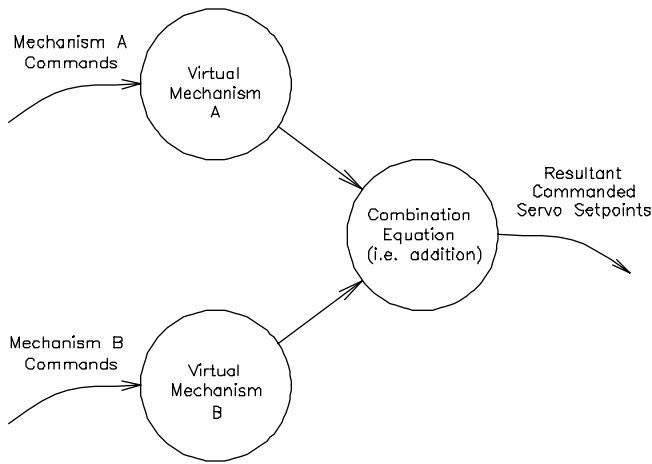


Figure 3. Multiple Input Transformation

Simulated movement is performed by independent virtual mechanisms responding to command or sensor information. The time history of these different trajectory sources is combined in real-time to produce a single result. In the work presented here the combination of inputs is simply addition although other combinations are possible. In a manner analogous to linear superposition, the summation of the independent solutions produces a combined result which solves the combined problem. In this paper the technique is referred to as motion superposition.

A helpful mechanical analogy for motion superposition is to consider the class of problems that could be solved if you mechanically attached on top of a multi-axis mechanism a second independent mechanism. Each mechanism can be commanded to move independently. The physical attachment of the two mechanisms results in the positions of each being added together to produce the final position. Motion superposition as a control technique accomplishes the same result without the mechanical redundancy. The following examples show the usefulness of motion superposition.

### Example 1: Acceleration Bounded Continuous Path Motion

#### Problem Description

When performing robotic assembly it is important to avoid certain obstacles while getting to the destination in the

minimum amount of time. Obstacle avoiding trajectories can be described by “corner points” and an associated radius inside which the robot is allowed to “cut the corner”. Inside the radius the shape of the trajectory is not important. Outside the radius the trajectory needs to be on the line between the neighboring corner points. An example path is shown in figure 4.

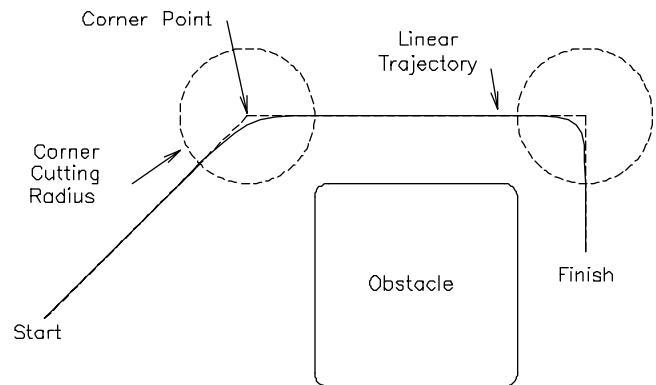


Figure 4. Corner Cutting Trajectory

The obvious solution to this problem is to move along a trajectory composed of straight lines and circular arcs. However this approach has a problem. Motion controllers will usually attempt to move along such a curve description at constant path velocity. As the corner-cutting radius is reduced there is a point where the acceleration limit of the motors is exceeded. In the limiting case of a corner cutting radius of 0 the robot is being asked to make a sharp corner at high speed resulting in discontinuous velocity. The solution being sought should gracefully handle the case of 0 corner cutting radius. A different approach is needed that allows corner-cutting radii to be arbitrary values without producing excessive acceleration.

#### Solution Strategy

Consider the following specific case where a Cartesian robot is asked to move along a corner-cutting trajectory composed of segments which are aligned with the robot’s axis. This case is shown in figure 5.

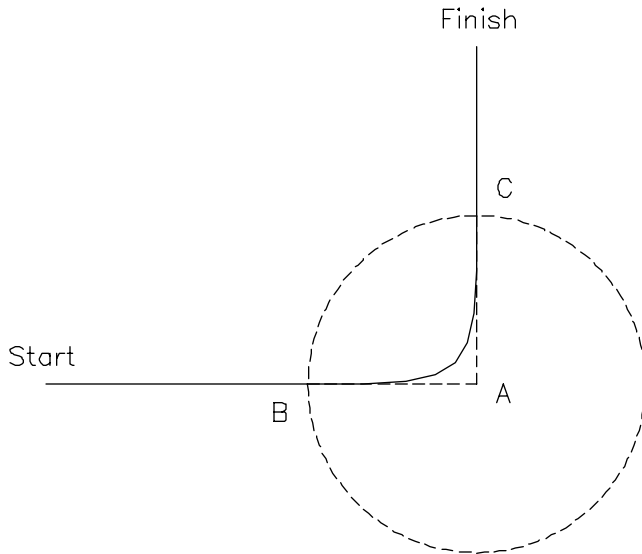


Figure 5. Aligned Segment Special Case

This particular case has an easy solution. Rather than treating the X and Y axis as a coordinated pair, instead treat them as independent axes. Start by initiating X movement towards point A. When the X axis passes point B and enters inside the corner cutting radius, simply start the Y axis before the X axis motion has completed. If the accel and decel values for X and Y are the same, the Y axis will just be getting to point C as the X axis gets to point A. Note that the "launch" of the second axis is dependent on when the first axis enters the corner cut radius, not on when the first axis begins to decel. These two motion attributes are independent. If the corner-cutting radius is reduced to 0 the movement degenerates to stopping at the corner, the correct behavior for 0 radius. Clearly the path is acceleration bounded because each independent axis is performing a trapezoidal motion profile. This solution has all the qualities we want but the unfortunate restriction that the trajectory can only be composed of segments aligned with the axes of the machine.

In this particular case the movement of the mechanism is the result of adding together the movements of the X and Y axis. By overlapping the moves we save time. This solution can be carried into a more general form.

## General Solution

Let's consider the more general corner-cutting trajectory shown in figure 6.

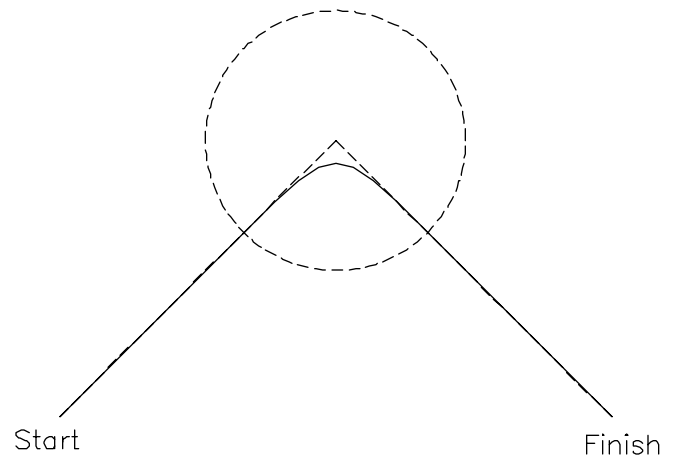


Figure 6. Coupled Segment General Case

This trajectory is composed of segments not aligned with the axis of the machine. Performing the linear section of the segment requires coordination of the two axes.

To solve this case two virtual XY mechanisms will be used. The first mechanism will be called XYA and the second XYB. These two mechanisms will perform the role of the independent axes in the orthogonal case. XYA will be responsible for the first segment. XYB will be responsible for the second segment. XYA will begin motion first. After XYA gets within the corner-cutting radius, XYB will begin movement before XYA finishes. The vector sum of the movement of the two virtual mechanisms is calculated in real-time to provide the commanded setpoints for the physical servo system.

## Implementation

The program used to implement the strategy is shown in figure 7. The first set of statements set the XYA and XYB virtual mechanisms to 0 position. The next statement schedules a task to execute every millisecond. This task is named "Superimpose" and performs the motion superposition calculation. A data logging and display task is started next. A 2

```

Editor
Finish Edit Search Configure
procedure click;
begin
  {initialize virtual mechanisms}
  XYA.SetCommandedPosition(0,0);
  XYB.SetCommandedPosition(0,0);

  {begin 1 millisecond motion superposition}
  ScheduleTask(TaskAddr(Superimpose),1);

  {begin data collection}
  BeginTask(TaskAddr(LogAndPlot));

  {identify corner point}
  CornerPoint.Init(1000,1000);

  {begin first segment}
  XYA.BeginMoveToVector(CornerPoint);

  {wait until XYA is within corner cut radius}
  repeat
    yield;
    XYA.GetCommandedPositionVector(XYAPos);
    Delta:=CornerPoint-XYAPos;
  until Delta.Length < CornerCutRadius;

  {overlap second virtual mechanism}
  XYB.MoveTo(1000,-1000);
end;

```

Figure 7. Corner Cutting Implementation

dimensional CornerPoint is established at coordinate 1000,1000. XYA then begins movement towards the corner point. While on the way there a continuous check is made to see how close XYA is to the corner point. When the distance to the corner is less than the CornerCutRadius then XYB starts its movement contribution.

The motion superposition equation, which executes every millisecond, is shown in figure 8.

```

Editor
Finish Edit Search Configure
Procedure Superimpose;
begin
  XAxis.SetCommandedPosition(
    XYA.XAxis.CommandedPosition+
    XYB.XAxis.CommandedPosition);

  YAxis.SetCommandedPosition(
    XYA.YAxis.CommandedPosition+
    XYB.YAxis.CommandedPosition);
end;

```

Figure 8. Motion Superposition Equation

This equation calculates the vector sum of the XYA and XYB virtual mechanisms and supplies the vector to the XY physical servos as commanded setpoints.

### Experimental Results

The commanded setpoint trajectory generated by executing this program is shown in figure 9.

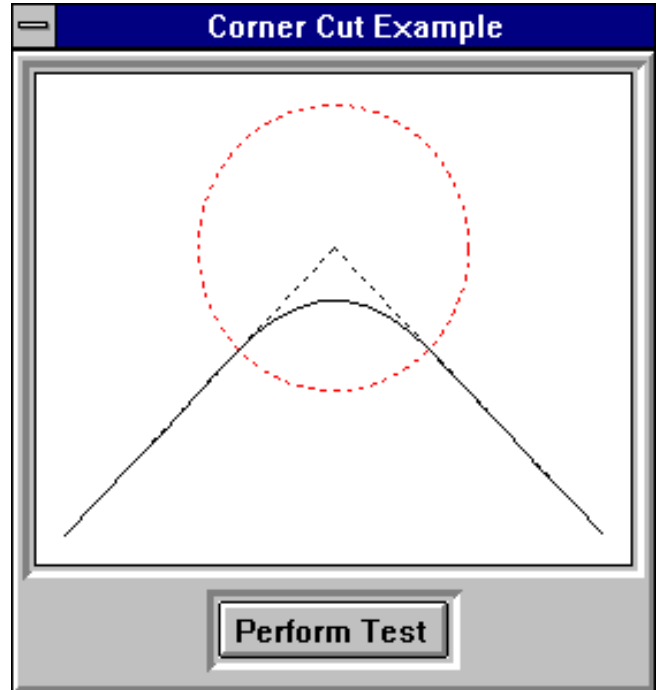


Figure 9. Corner Cutting Experimental Results

The trajectory contains the attributes desired including linear motion outside of the corner cut radius, and a smooth transition between segments within the radius.

How well is this approach solving the acceleration bound requirement? As an extreme case consider what happens if XYB's movement "folds back" completely on XYA's movement. In this worst case, the acceleration at the turnaround point would be double the acceleration of the individual virtual mechanisms. By setting the accel of XYA and XYB to be half the system accel this approach handles all cases, but that's a bit wasteful. A more optimum approach would be to set the XYA and XYB accels based on the angle in between the segments they will be performing.

By using a “double buffer” style technique the XYA and XYB virtual mechanisms can “leapfrog” one another to direct movement along a multiple segment trajectory.

### Interesting Attributes

One of the interesting attributes of this solution is that we never explicitly specified the shape of the trajectory. However we do know because of the profiling boundaries of XYA and XYB that we are satisfying the constraints of the problem.

## Example 2: Backlash Compensation

### Problem Description

We would like to specify the position of a mechanism which has backlash. In order to monitor the actual position of the mechanism a second encoder is provided. The motor should not be commanded to move beyond its acceleration limits. Conventional dual-loop solutions that could be used to combine information from two encoders do not necessarily respect the acceleration limits of the motor.

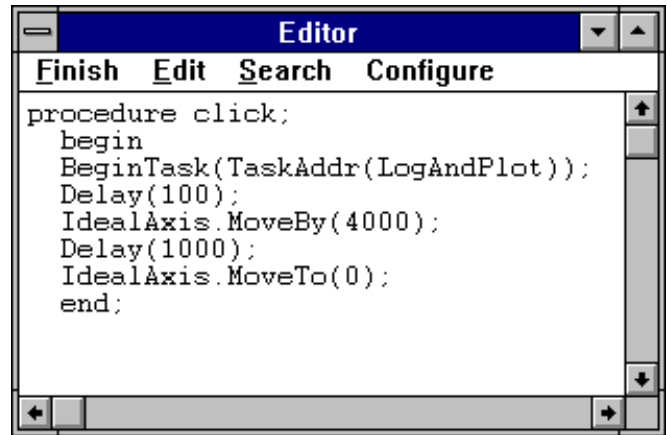
### Solution Strategy

A two motor solution could be used to address this problem. The first motor could provide the fundamental movement of the system while the second motor could “adjust away” any discrepancy between the actual position of the mechanism and the desired position. Each motor would continually operate with different criteria to solve that particular motor’s contribution to the solution. However mechanically and electronically such a structure would be wasteful since both motors produce movement in the same mechanical direction.

To solve this problem with superposition two virtual axes are used. The first axis is named IdealAxis and the second is named AdjustmentAxis. IdealAxis will be regarded as not having backlash. Motion commands will be sent to IdealAxis representing the desired motion of the mechanism. AdjustmentAxis will take on the responsibility of nullifying the backlash.

## Implementation

The movement of the mechanism for the experiment is prescribed with the program shown in figure 10.



```

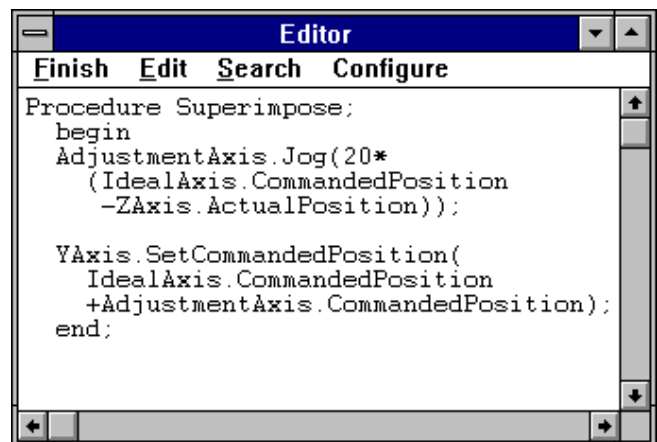
procedure click;
begin
  BeginTask(TaskAddr(LogAndPlot));
  Delay(100);
  IdealAxis.MoveBy(4000);
  Delay(1000);
  IdealAxis.MoveTo(0);
end;

```

Figure 10. Ideal Movement Program

The first statement starts a data collection activity. A short delay is introduced so as to see the beginning of the movement in the data. The IdealAxis is then told to perform a move to 4000, wait a second, and return to position 0.

The following program in figure 11 provides both the adjustment control law and the motion superposition equations. This program executes as a separate, concurrent program every millisecond.



```

Procedure Superimpose;
begin
  AdjustmentAxis.Jog(20*
    (IdealAxis.CommandedPosition
     -ZAxis.ActualPosition));

  YAxis.SetCommandedPosition(
    IdealAxis.CommandedPosition
    +AdjustmentAxis.CommandedPosition);
end;

```

Figure 11. Anti-Backlash Superposition Equation

The ZAxis is being used to read the encoder on the output of the mechanism. The YAxis is the physical motor of

the system. The first command tells the AdjustmentAxis to move so as to nullify any difference between the Actual output position of the mechanism and the ideal, desired position. This nulling is accomplished by jogging at a speed proportional to the error, a simple "P" control loop directing a velocity source.

The second statement in the program establishes the YAxis commanded position by simply adding together the ideal position and the adjustment position.

### Experimental Results

Performance of this anti-backlash strategy is graphed in figure 12.

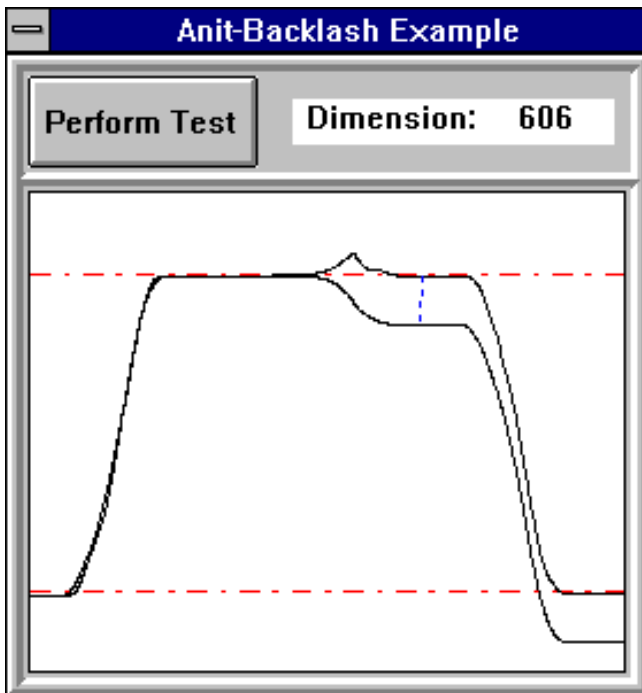


Figure 12. Anti-backlash Position Histories

The upper straight line represents a position of 4000 counts. The lower straight line represents a position of 0. The graph is composed of two position histories. The line which remains on top is the actual position of the mechanism output. The line which ends below is the position of the driving motor. Initially the motor and output agree since the backlash was initially engaged in the direction of motion. Half way through the dwell the load on the mechanism was reversed so as to

have the backlash contribute position error on the output encoder. This causes the anti-backlash control to take effect.

When the load reverses, the load position begins to increase beyond the target position of 4000 counts. At this point the driving motor compensates by dropping back and pulling the load back into position, returning the load to the upper line representing 4000. This offset in the drive motor is retained as it moves back to position 0. The distance that the drive motor compensated is measured to be about 600 counts, the amount of backlash in the system.

Note that this anti-backlash technique is different from the technique which adds or subtracts a backlash distance based on the direction of movement. The technique shown here corrects the backlash based on the reversal of the load itself as monitored through the output encoder, not on the direction of ideal movement. The amount of adjustment is not fixed but continually measured.

As was seen in the first example it is possible for a double-acceleration case to occur if the superposition routine asks the AdjustmentAxis to accelerate in the same direction as the IdealAxis is accelerating. This double-accel issue can be resolved by setting the accels of each virtual axis so as to sum to the acceleration limit of the physical system.

### Interesting Attributes

This technique is able to respect the motor's acceleration bound while at the same time being "continuous" in application and not just as "end-point" correction.

## Example 3: Antenna Pointing from Moving Reference Frame

### Problem Description

In this example we would like to enable a satellite antenna, mounted on a ship, to point to one of several satellites despite the ship's rolling movement. In this paper just the roll of the ship is considered although the concept could be applied to other rotational axes as well.

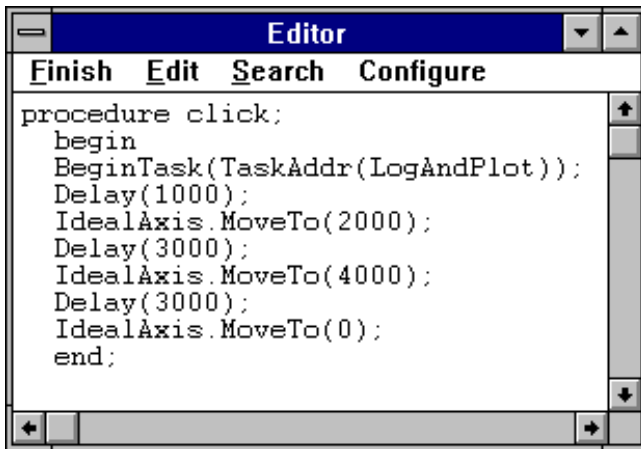
### Solution Strategy

Eliminating the roll and directing the antenna would be easy if the system had an intermediate platform. The platform would be positioned by one control system and the antenna attached to the top of the platform would be controlled by a separate control system. The platform control system would keep the platform stable by commanding the platform to null-out the ship's roll. The antenna control system would simply point the antenna with respect to the stable platform. However, as in the previous example, this approach is not good from a system point of view because of the mechanical and electronic duplication.

Rather than having a physical intermediate platform a virtual intermediate platform will be used. A roll sensor, such as an encoder monitored damped pendulum or gyro could be used to measure ship roll. This information will serve as a stable base on which ideal movement will be added.

### Implementation

The ideal motion requested of the antenna is shown in the figure 13.

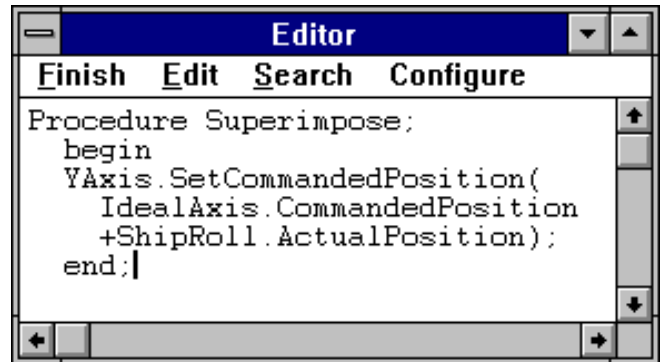


```
procedure click;  
begin  
  BeginTask(TaskAddr(LogAndPlot));  
  Delay(1000);  
  IdealAxis.MoveTo(2000);  
  Delay(3000);  
  IdealAxis.MoveTo(4000);  
  Delay(3000);  
  IdealAxis.MoveTo(0);  
end;
```

Figure 13. Ideal Antenna Pointing Program

As in the previous example, the first statement begins a data collection. The IdealAxis is then moved to several positions with dwells before returning to the original 0 position.

The superposition principle being applied in this example is shown in figure 14. This program is scheduled to execute every millisecond.



```
Procedure Superimpose;  
begin  
  YAxis.SetCommandedPosition(  
    IdealAxis.CommandedPosition  
    +ShipRoll.ActualPosition);  
end;
```

Figure 14. Antenna Superposition Equation

The YAxis is the drive motor for the antenna. IdealAxis is the virtual axis commanded to move the antenna as if it was on a stable platform. ShipRoll is a sensor indicating the negative of the Ship's angle with respect to horizontal. The superposition equation adds together the ideal desired behavior with the negative roll information to compensate out the roll movement.

### Experimental Results

The response of the system is shown on the graphs in figure 15. The top graph shows the ShipRoll sensor versus time and has a sinusoidal shape. The second graph in the middle indicates the commanded position of the IdealAxis with respect to time. This axis performs conventional trapezoidal motion as if there were no additional concerns or aspects to the problem. The bottom graph is the commanded position as sent to the drive motor and is the sum of the two graphs above it. The basic pointing direction is continually adjusted by the roll information to compensate the roll away.



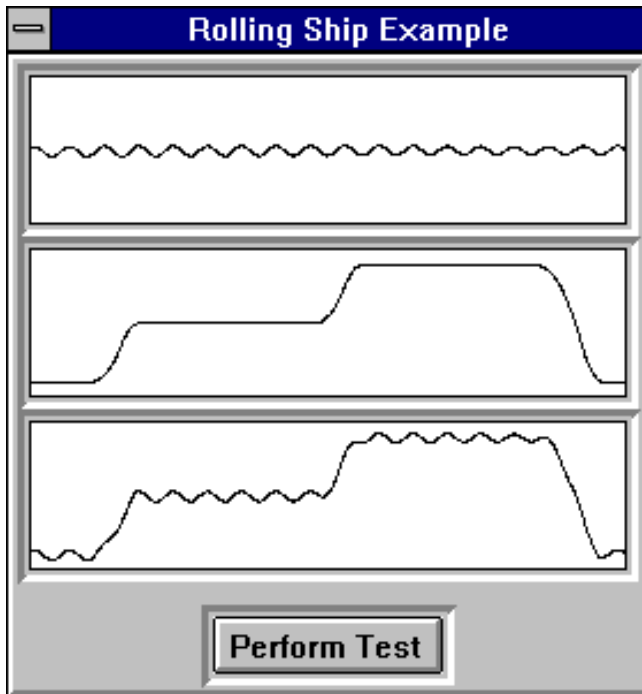


Figure 15. Antenna Experiment Position Histories

#### Interesting Attributes

This approach assumes the ship will not roll at an acceleration rate greater than the servos can handle. This should be true given the dynamics of a ship versus the dynamics of the servo. Additional axis could be provided to control the remaining degrees of freedom needed to point the antenna.

#### Controller Attributes Required for Superposition Technique

In order for a motion control system to employ this superposition technique several attributes are required. The first attribute is “virtual” or “phantom” axes with which to create virtual mechanisms. It is also necessary to be able to perform the superposition calculations at high speeds, preferably at the controller sample rate, so as to keep the system simple and not incur bandwidth limitations that can come with interpolation between lower-sample-rate commanded setpoints. It’s desirable to have high speed programs which can perform floating point math quickly. A floating point math coprocessor is also valuable, particularly for transformations involving trigonometry. Pre-emptive multithreading

is very important as an independent thread can manage the superposition equation while other threads manage general mechanism control. Finally, the application programming environment should be able to express these custom relationships easily and conveniently since the superposition equation is quite application specific.

#### Summary

From a mechanical viewpoint, these different problems have solutions which can be created by concatenating together two independent servo controlled mechanisms. Each mechanism responds to different control criteria to contribute its part to the total resulting motion. Being physically concatenated, the resulting motion is the vector sum of the independent mechanism movements. This motion superposition technique allows thinking in this partitioned mechanical manner, but perform the concatenation of the two independent servos in software rather than mechanical hardware. This combined solution, expressed through a single servo controlled physical mechanism, contains the behavior of both concepts but only the expense of one mechanism. Controllers equipped with suitable real-time performance can easily implement this technique to simplify mechanical design while accomplishing more advanced motion applications.

#### Bibliography

- 1) Andrews, J. Randolph: "A Software Component Library for Motion Control", In *Proceedings of the Twenty Second Annual Symposium on Incremental Motion Control Systems and Devices*, San Jose, CA, 1993.
- 2) Andrews, J. Randolph: "An Advanced Motion Control System Architecture Based on a 386 PC", In *Proceedings of the Twenty First Annual Symposium on Incremental Motion Control Systems and Devices*, San Jose, CA, 1992.
- 3) Andrews, J. Randolph: "A Servo Application Development Environment for Microsoft Windows", In *Proceedings of the Twenty First Annual Symposium on Incremental Motion Control Systems and Devices*, San Jose, CA, 1992.



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

5) W. Brogan, *Modern Control Theory*, Prentice Hall, Englewood, New Jersey, 1991

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

### **About the Author**

J. Randolph Andrews received his B.S. M.E. in 1981, B.S. E.E. in 1981 and M.S.M.E. in 1983 from the Massachusetts Institute of Technology. He participated in the MIT Mechanical Engineering Department's DeFlorez Design Competition each undergraduate year winning 1st place '78, 1st place '79, 1st place '80 and 1st and 2nd place '81.

Andrews spent 4 years at Hewlett Packard's corporate research laboratory in the Applied Physics Research Center as well as the Manufacturing Research Center.

The following 4 year period was spent with Galil Motion Control.

In July '91 Andrews founded Douloi Automation to provide motion control hardware and software systems for use with Microsoft Windows.

Professional interests include motion control, software/electrical/mechanical system design trade-offs, high abstraction programming and visual programming techniques and tools.