Motion Controller Resident Collision Avoidance

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

Paper Presented at the
1998 Incremental Motion
Control Systems and
Devices Symposium

Abstract

Automated machines are being given greater responsibilities. Particularly in the semiconductor industry, a single work-in-process wafer may be more valuable than the machine handling it. A machine work area can be crowded with delicate and expensive tooling including microscope optics and fixtures. A single positioning mistake, such as an improperly trained point or incorrectly calculated destination, can produce a machine collision.

A collision damages expensive tooling and mechanics, wastes work in process, stops system development or production, and contaminates a clean environment. Measures must be taken to avert these outcomes. Regardless of the commands sent to the motion controller, the machine should never collide with tooling, fixtures, or mechanical limits.

Once a topic of abstract research, advances in motion control technology enable real-time collision avoidance to be resident on-board the motion control card itself. One or more geometric models describe safe movement areas. These models are used by a real-time monitor which anticipates collisions based on current machine position and velocity. The machine is stopped before a detected collision can occur. An example taken from industry will show the method of implementing on-board collision avoidance.

Collision Cost

Semiconductor wafers have grown in size from small disks up to the 300 mm standard being pursued today, about the size of a small pizza. An unprocessed 300 mm wafer costs about $1500 although some of this expense may be related to the still relatively small volume compared to 200 mm wafers. Upon completion, the same wafer may have a value of $90,000. A single wafer can easily cost more than the semiconductor processing tool currently holding and moving it. The cost of a collision that damages the wafer has dramatically increased.

New generations of semiconductor processing equipment are being developed at a rapid pace. During development software is being run for the first time and may well be generating incorrect move destinations. If this machine under development collided with a fixture several types of expenses occur. Mechanical parts will need to be rebuilt, replaced, and recalibrated. As well, the software testing effort may come to a halt while the mechanism is being restored. The system integration schedule slips because there is often just one prototype machine to work with.

At run-time the costs are even greater. A collision damages work in process, generates particle contamination in a clean environment, and potentially stops a production line at great expense.

Collision Causes

Why would a collision occur in the first place? Some collisions are caused by runaway conditions and other types of uncontrolled motor movement. The collisions being considered in this paper are produced when the controller is deliberately told to perform a move to an inappropriate destination. Destinations are often data driven. Position information can be calculated from CAD information, assembly and part models, or taught locations.

Ideally, the motion controller is safeguarded from bad destinations by error checking being performed in the host software. However, the host software is not necessarily finished or proven. The collision avoidance is needed before the host software is done because software development and debugging is a high-risk period for the machine.

There are also cases where the motion directives are not coming from the host, but coming from a joystick or sensor. Collision avoidance needs to be active in these cases also.

Behavioral Objective

The goal is to have a motion control system which is simply unwilling to do something that would damage the machine. This objective is not to be confused with "obstacle avoidance". As defined in this paper, obstacle avoidance includes the idea of motion path adjustment so as to maneuver around obstacles in the machine workspace. The behavior of a machine implementing obstacle avoidance in response to a foreseen collision is to change course and navigate around the obstacle. The behavior of a machine implementing collision avoidance is to simply come to a stop. There is no recovery plan for a collision situation. The machine has a structured environment. No legitimate command from good data would direct the machine into a collision situation. If a collision problem is detected then apparently something has gone very wrong. In this mind set, continuing is not a good idea. Operator or developer intervention is required.

One might make the critical point that adding collision avoidance will create unnecessary complexity and degrade system reliability because of that complexity. It is true that unnecessary complexity is part of the problem in deploying automation, not part of the solution. This appears to be a legitimate criticism. However the collision avoidance is isolated from the design of the host software being resident onboard the controller. It must be isolated, because part of its job is to safeguard against mistakes in the host software. The presence of collision avoidance on-board the motion controller has almost no impact on the host software with the exception of additional status indicating that a move stopped prematurely to avoid a collision. There is no increase in host software complexity.

Example Mechanism

An example mechanism that will be considered in the course of this paper is shown below.

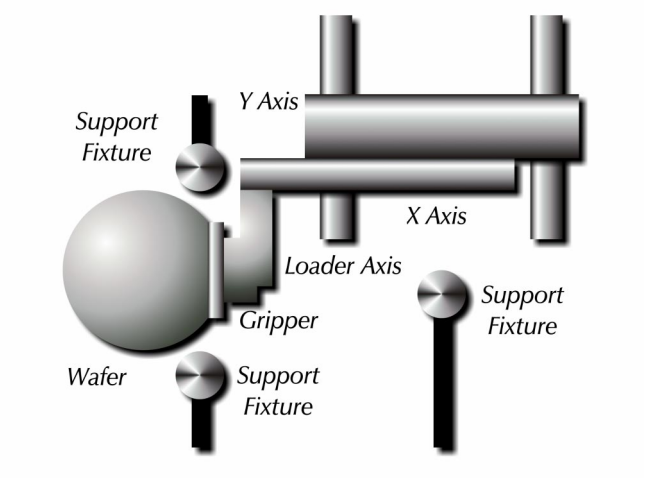

Figure 1. Example Mechanism

Figure 1 shows an example of a mechanism with a gripper operating in a workspace. A number of collision features are in the workspace including two support fixtures on the left that the gripper reaches between and a support fixture on the right of the machine. These support fixtures are required to support the film on which the wafer is mounted. This particular mechanism contains a primary X and Y stage along with a loader attached to the X stage which also moves in the X direction. In Figure 1, the loader is to the extreme left in its travel on top of X axis. The gripper is located on the end of the loader. Material is retrieved and deposited in a magazine on the left side of the machine. The wafer passes over the support fixtures, however the gripper and sections of the X axis slide can collide into them.

Figure 2 shows the mechanism in the processing position having retrieved a wafer. The loader has moved to the right by the length of the X axis member. The X axis has moved slightly to the right and the Y axis has moved up.

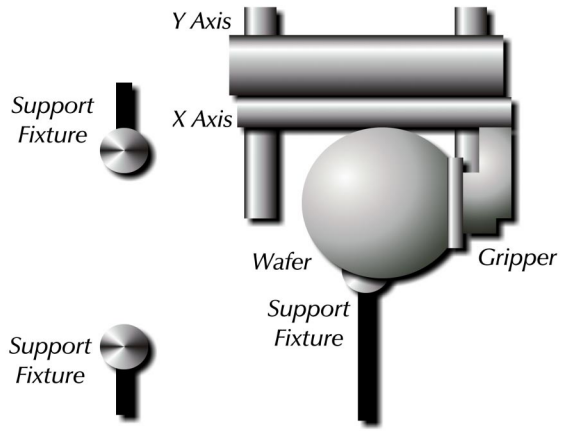

Figure 2. Mechanism in Normal Run Position

X and Y movement of the primary stage positions the wafer during processing. Moving from the Figure 1 load position to the Figure 2 run position requires moving in a deliberate path around the right support fixture. A straight line path, for example, would cause a collision.

Proactive Collision Avoidance

There are several ways to approach collision avoidance. One approach uses a proactive style. Proactive avoidance considers the consequences of a proposed motion and rejects the command without even starting the move if the move would cause a problem.

One common example of proactive collision avoidance is simply the “soft limits” found on many motion controllers. Software limits are depicted in Figure 3. A positive limit and negative limit is sent to the controller for a particular axis. If the axis is asked to move outside these limits, the command is rejected before any motion occurs. In this case, the collision avoidance operation is simply a comparison of the proposed coordinate with the limits to insure that the destination is safe.

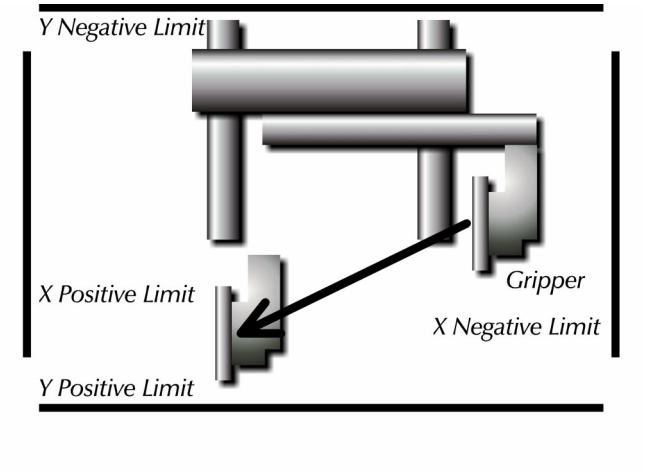

Figure 3. Software Positive and Negative Limits

Proactive collision avoidance becomes more complicated when there are machine features in the work area. A destination may be in a safe zone, but the path may go through a feature. This case is shown in Figure 4. A proactive approach needs to check for intersections between the moving members and the machine fixtures.

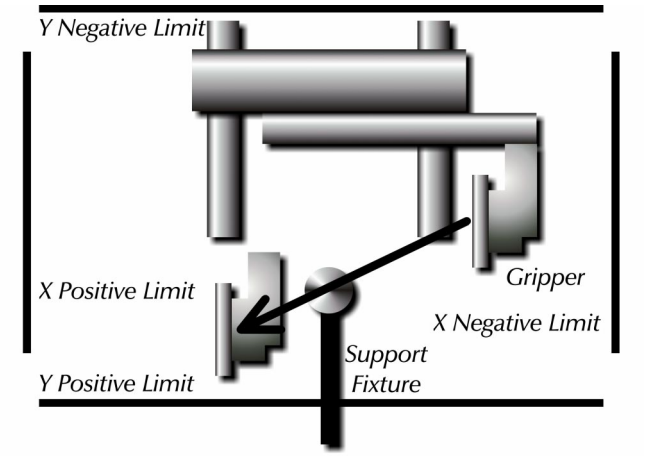

Figure 4. Safe Destination with Collision Midway

An additional consideration is that a proactive approach presumes that the controller has a specific start location and destination location for the next move. The path planner would be the most appropriate place to locate a proactive collision evaluator. It’s possible to run the controller with a joystick, or electronic-gearing hand wheels. In this

case the motion is “sensor-driven” rather than path-planning driven. Destinations are calculated every sample period based on the sensors and do not even go through the path planner in the motion controller.

There might be a reason to ask for a motion controller to begin moving to an unsafe destination with the requirement that the motion is stopped early while the machine is still in the safe area. Some joystick methods use this approach. A proactive approach would reject this technique since it would be unwilling to start the move initially.

Reactive Collision Avoidance

In contrast to proactive collision avoidance, reactive collision avoidance is willing to try any move that is asked of the controller but evaluates whether the move is remaining safe in an ongoing, real-time manner. If the move is heading towards an unsafe condition the motion controller stops motion before the collision occurs.

The reactive approach can be more general than the proactive approach making real-time decisions on the fly. The reactive approach is able to handle the problematic cases for the proactive collision avoidance method. In this paper the reactive method is used.

The Geometric Model

The first step in avoiding collisions is to have a clear understanding of where the safe-movement zones are in the machine. This paper is not considering sensor-based collision detection. Instead, collision detection will be based on a geometric model that describes safe areas of movement.

Tomas Lozano-Perez showed that a collision problem between an object and a work environment can be re-mapped into a collision problem between a point and an adjusted work environment. Figure 5 shows the physical arrangement of a gripper around machine features.

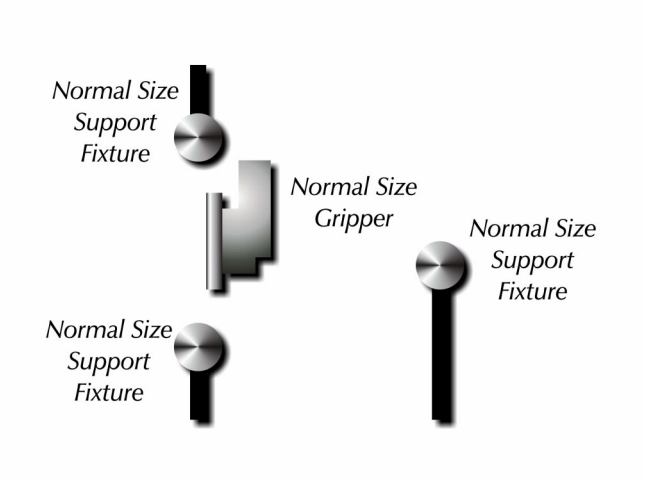

Figure 5. Normal Sized Machine Features

Figure 6 shows how the gripper has been reduced to a point and the machine features enlarged to represent the same physical region of movement.

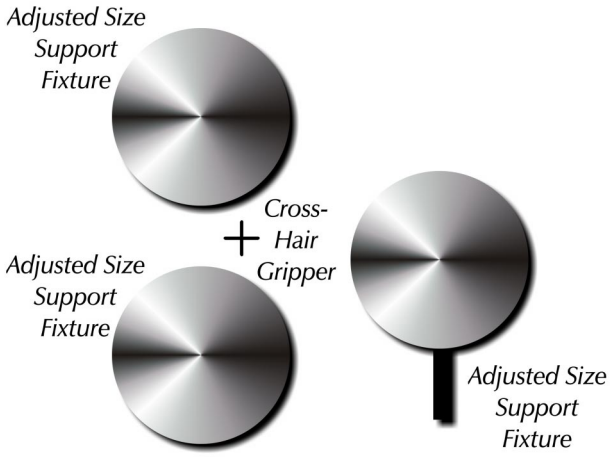

Figure 6 - Adjusted Features with Point Gripper

This re-mapping is convenient because it simplifies collision detection calculations. The re-mapping needs to be done just once. Questions can now be asked about the point being inside or outside a feature, a simpler problem than asking if different shapes representing the gripper and machine features overlap. This re-mapping is similar to radial tool offset compensation however the adjustment in the X direction is not necessarily the same as the adjustment in the Y direction. In practice, the model is learned through teaching and the mapping is never explicitly performed.

Polygons can now be used to represent a “safe zone” of movement. If we describe the polygon with lines aligned with axes we have an easy job determining if the gripper point is inside the inflated polygon shape. It is also possible to use other shapes based on the geometry of the features being avoided such as circles or arcs. In this paper only polygons were used with polygon edges aligned to axes of motion.

Beyond collision risk of the gripper there is also the possibility of colliding other parts of the machine into fixtures including intermediate stages. There may be several, different concurrent models based on how many ways there are for a collision to occur. In the example mechanism there is one particular model for collisions between the gripper and surrounding tooling. There is a second model that is used to detect collisions between the X stage and the tooling, separate from considerations of the gripper. The collision outcome of these different models can be combined to produce a single answer, collision or no collision.

Model Training

How are geometric models taught? The easiest method is by explicit training. A training interface, such as shown in Figure 7, can be used to establish safe-zone edge locations.

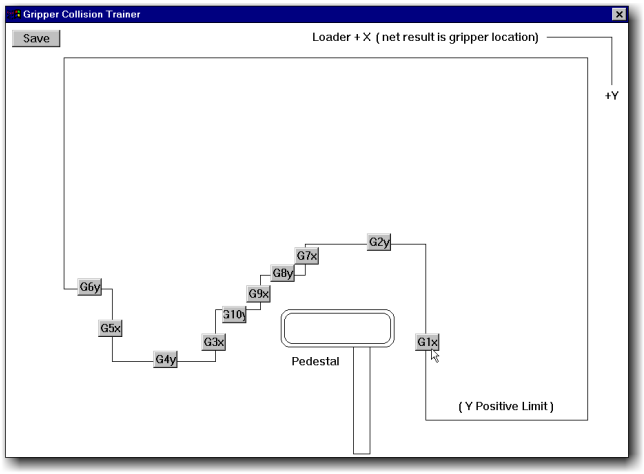

Figure 7. Gripper Collision Trainer

This trainer is for the gripper. The stair-step geometry describes connector and corners in the gripper that must be avoided although the gripper does move in closely to the left of the right-most support fixture.

Using a joystick control mode, the machine is manually moved to various positions representing the boundaries of the safe-zone polygon. In the interface, these polygons have buttons on the edges. When the machine has been manually guided to that edge, the button is pushed to record the machine coordinate for that polygon feature.

Figure 8 shows the much simpler trainer used to describe collisions between the X stage and the right most support fixture. In this case, a motor bracket can collide if the X stage is in the most positive location. This produces a notch in the safe area workspace shown in the lower left corner.

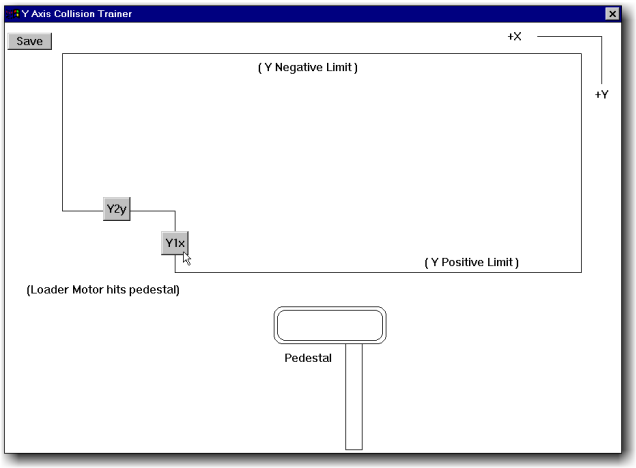

Figure 8. Collision Trainer for X Axis

The machine is guided around the limits of the work area with the edge-buttons being pushed along the way. In this implementation, the shape of the polygon is "hard-coded" into the training interface as is the inside-outside algorithm used to monitor for collisions. A more general solution would allow the polygon to be described interactively, such as is done by a CAD system, so as to accommodate changes to the machine structure without having to change the on-board software.

After showing each of the polygon edges, the geometric data can be saved in the host or saved in nonvolatile memory in the motion controller. If the information is saved in the host and downloaded into the controller, there must be some provision for insuring that no unsafe motion occurs prior to receiving the polygon safe-zone description. One way to accomplish this is to set all of the polygon coordinates to 0,0. This causes the motion controller to believe that the safe zone of operation is a mere point. Any motion directive at all is considered illegal and is rejected prior to receiving the safe-zone description.

Dynamic Considerations

The first step in performing collision avoidance is having a geometric safe-zone model. The second step is to use that model in a dynamic, real-time manner to anticipate that a collision will occur given the current state of the machine. When this is found to be the case the motion controller can avert the collision by stopping motion.

This dynamic consideration can be performed by using the current machine velocity vector and deceleration settings to calculate the current distance required to perform a controlled stop. A “safety margin” is added to this stopping distance to become the look-ahead distance. The controller then spatially looks-ahead of the current machine coordinate in the current-velocity direction. If the look ahead point is outside the safe-zone then the machine has just enough time, if it stops immediately, to avoid a collision. The current motion is aborted and error status is set in the controller indicating that the move finished early due to a collision avoidance stop.

This method assumes that the direction of the velocity vector is constant during the proposed stopping move, i.e. straight line vector style moves and not circular or curvilinear moves. For the application being described this was the case.

What should be done in light of such a stop? This question must be answered at the system design level. It may be that if all was well this collision avoidance safety net never should have engaged. If a collision was about to occur then there must be something fundamentally wrong with the positioning data or the host control software. On the other hand, if the collision stop was in response to a joystick command, the collision stop is simply a “safety bumper” around the machine feature begin protected. In this case there is no problem at all with the collision avoidance stop as the joystick control is operating normally.

It is interesting using joystick control to try to "strike a glancing blow" with the robot against one of the support struts. The gripper approaches at high speed, slows and stops against the invisible safe-zone polygon, but retains some velocity along the edge of the polygon (if it was a glancing, rather than direct blow). The gripper follows along the edge until the end of that polygon edge. At this point the gripper is free to continue. It immediately accelerates up to speed and continues as if "falling off the edge" of the polygon.

Implementation

The controller software must answer the question "Is the following described point in the machine's safe zone or not?". The pseudo code for this question is:

```
If coordinate is inside polygon then
   Coord would not cause collision
else
   Coord would cause collusion
```

This pseudo-code solution is implemented for the specific polygons used in the example mechanism with the following routine:

```
Function CoordinateWouldCauseCollision(
  XPos:longint; YPos:longint;
  LoaderPos:longint):boolean;

  var XPlusL:longint;

  begin
  if not CollisionAvoidanceEnabled then
   begin
   CoordinateWouldCauseCollision:=false;
   exit;
   end;
```

```
CoordinateWouldCauseCollision:=true;
if (XPos > Edge(ul_Y1x))
and (YPos > Edge(ul_Y2y)) then
 exit;

XPlusL:=XPos+LoaderPos;

if XPlusL < Edge(ul_G1x) then
 begin
 CoordinateWouldCauseCollision:=false;
 exit;
 end;
if (XPlusL < Edge(ul_G7x))
and (YPos > Edge(ul_G2y)) then
 exit;
if (XPlusL < Edge(ul_G9x))
and (YPos > Edge(ul_G8y)) then
 exit;
if (XPlusL < Edge(ul_G3x))
and (YPos > Edge(ul_G10y)) then
 exit;
if (XPlusL > Edge(ul_G5x))
and (YPos > Edge(ul_G6y)) then
 exit;
if YPos > Edge(ul_G4y) then
 exit;
CoordinateWouldCauseCollision:=false;
end;
```

This routine takes a proposed coordinate for the X axis, Y axis, and Loader axis. If collision avoidance is not enabled, then the routine returns false immediately. This is needed during safe zone training, for example. In certain situations the machine needs to be free to move if the collision data is unknown.

The answer to the question coordinate-would-cause-collision is set to true as the default answer. The positions submitted for checking then go through a series of comparisons with edges of the safe-zone polygon. The edges are noted with names such as ul_G7x, an X edge coordinate, or ul_G10y, a y edge coordinate. The first comparison is based on X and Y to check for collisions of the X stage against machine features. The remaining tests use the position XPlusL representing the sum of the X axis and loader axis which establishes the position of the gripper. A greater number of position comparisons are done with the gripper to determine if the gripper is inside or outside the safe-zone. Checking both the X axis and the gripper illustrate the use of two separate geometric models (although the X axis model is just the polygon notch).

This routine provides an answer to the static question regarding an X, Y and Loader coordinate. Now the dynamic question must be asked. The pseudo code for the dynamic collision avoidance case is expressed as follows:

```
|Stopping Vector|:= f(velocity)
  +CollisionMargin

angle(Stopping Vector):=angle(velocity)

LookAheadVector:=
   CurrentPositionVector+StoppingVector

If LookAheadVector coordinate would cause
collision then
   Stop all motion and
   Report collision stop
```

The following code implements the pseudo code for the dynamic motion connection to the safe-zone polygon:

```
Procedure WatchForCollision;

  var Delta:T3Vector;

  begin
  CollisionAvoidanceStop:=false;
  while (not XAxis.MoveIsFinished)
  or (not YAxis.MoveIsFinished)
  or (not LoaderAxis.MoveIsFinished) do
   begin
   yield;

   if XAxis.ProfileVelocity > 0 then
    Delta.x:=XDecelDistance
     +CollisionMargin
   else if XAxis.ProfileVelocity < 0
   then
    Delta.X:=-XDecelDistance
     -CollisionMargin
   else
    Delta.x:=0;
```

```
if YAxis.ProfileVelocity > 0 then
 Delta.y:=YDecelDistance
  +CollisionMargin
else if YAxis.ProfileVelocity < 0
then
 Delta.y:=-YDecelDistance
  -CollisionMargin
else
 Delta.y:=0;

if LoaderAxis.ProfileVelocity > 0
then
 Delta.z:=LoaderDecelDistance
  +CollisionMargin
else if LoaderAxis.ProfileVelocity <0
then
 Delta.z:=-LoaderDecelDistance
  -CollisionMargin
else
 Delta.z:=0;


if CoordinateWouldCauseCollision(
 XAxis.ActualPosition+Delta.x,
 YAxis.ActualPosition+Delta.y,
 LoaderAxis.ActualPosition+Delta.z)
 then
  begin
  CollisionAvoidanceStop:=true;
  XYAxis.BeginStop;
  XAxis.BeginStop;
  YAxis.BeginStop;
  LoaderAxis.Stop;
  while not XAxis.MoveIsFinished
  and YAxis.MoveIsFinished
  and LoaderAxis.MoveIsFinished do
   yield;
  end;
end;
```

The name of the procedure is "Watch for Collision". This procedure is spawned as a separate thread of execution when a motion is started. As currently implemented it executes as long as there is motion occurring with the X, Y, or Loader axes. Once all the motion has stopped, the routine ends as there is nothing left to do. There may be some circumstances where it would be simplest just to always have the routine running.

The Delta vector is being used to hold look ahead information. The decel distances for each of the axes are found and oriented based on the current velocity of the axis along with a Collision Margin distance to provide a cushion around the machine features. The projected machine positions are calculated to be the current axis positions plus their corresponding look ahead delta values. If this projected point is not safe then the machine is stopped and status is recorded to indicate this was a collision avoidance stop.

The decel distances are calculated in 80 bit floating point hardware with the following math coprocessor using "reverse polish" style routines similar to this one:

```
Function XDecelDistance:longint;
    begin
    FInit;
    PushLongint(XAxis.ProfileVelocity);
    PushLongint(XAxis.ProfileVelocity);
    FMul;
    PushLongint(XAxis.Decel);
    FDiv;
    PushLongint(2);
    FDiv;
    XDecelDistance:=PopLongint;
    end;
```

Implementation Results

This collision avoidance task was described using Douloi Automation's Servo Application Workbench software tools. The routines were downloaded into a Motion Server control card which has a 128 MHz 486 processor and floating point coprocessor. The main looping body of the WatchForCollision routine was measured to take 35 microseconds of time.

In this particular application brushless motors were being used and the controller sample rate was set to 2 kHz. Of the 500 microsecond budget available to the controller, approximately 105 microseconds was being used for general motion control of the application including analog input monitoring and filtering for joystick control. Of the remain-

ing 395 microseconds, the 35 microsecond collision avoidance calculation time represents less than a 10% load of the controller's remaining time budget. The collision avoidance routine could run every controller sample period without seriously loading the controller. The calculation time would increase for more descriptive safe-zones such as diagonal lines, circles, and arcs.

In this particular application the host software was responsible for downloading polygon coordinates into the controller. This turned out to be troublesome as it required the software engineer to do some additional work interpreting the file and downloading the numbers into the controller before the collision avoidance benefit would be realized. In subsequent applications the safe-zone geometry will most likely be stored in on-board flash to minimize the amount of support host software designers must provide and to make the machine as independently safe as possible.

Summary

Machine control software is increasing in complexity. The cost of a machine crash is increasing in cost. Using motion controller resident collision avoidance, the risk of damage to a machine can be decreased both during development and run-time.

A motion controller having sufficient computational power, multithreading resources, and on-board application support tools can implement a reactive collision avoidance method. The incremental load on the motion controller can be less than 10% of the remaining available control resources after the controller has completed other time critical application tasks.

Isolating collision avoidance from host software provides a higher integrity safety-net as defects and defect patterns that might be present in the host software do not also corrupt collision avoidance.

Bibliography

1) Lozano-Perez, Tomas, "Spatial Planning: A Configuraton Space Approach", Artificial Intelligence Laboratory, Massachusetts Institute of Technology, December 1980

2) 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.

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.

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, Inc. to provide motion control hardware and software systems for use with Microsoft Windows. Douloi Automation also provides turnkey automation solutions for advanced automation applications.

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