EVALUATION BY SIMULATION OF INTERPOLATION AND ACCELERATION ALGORITHMS FOR STEPPER MOTORS

J.F. POLIAKOFF, Y.K. CHOW, P.A. ORTON, M. HOWSON, D. AL-DABASS

School of Computing and Informatics
Nottingham Trent University
Nottingham, NG1 4BU, UK.
janet.poliakoff@ntu.ac.uk, paul.orton@ntu.ac.uk, david.al-dabass@ntu.ac.uk

Abstract: Stepper motors are used to control CNC machines for many applications. As well as following the required path precisely, it is also important that the motion be smooth and that the surface speed be controllable. Improved interpolation algorithms for individual straight lines and circular arcs have been developed using distance as a parameter [Chow et al, 2002], [Chow, 2003]. The algorithms control the motor by means of pulses and the generation of the pulse timings is based on the geometry of the shape. For high speeds it is necessary to allow smooth acceleration at the beginning and similar smooth deceleration at the end. Thus, appropriate acceleration and deceleration algorithms have been developed for use with the new interpolation algorithms. This paper describes how simulation has been used to evaluate the new algorithms and compare them with previous algorithms. The algorithms are described for the 2D case but the principle can be extended to 3D.

Keywords: CNC, simulation, interpolation, motion generation, stepper motor, smooth motion, acceleration.

1. INTRODUCTION:

In machine tool control for CNC machines, it is very important to have both smooth continuous motion and precise following of the required path. The surface speed of the machine tool also needs to be controlled appropriately. Stepper motors are popular, because they have simple interface requirements and low cost. They normally operate without feedback and the required shape is followed by adjusting the speeds on the different axes. The path round the shape is generated by sending pulses to the motors for the axes at calculated times. In many cases all the steps are of the same size, typically 0.01 mm. We have assumed that this is the case in all our examples. When a complex path is to be followed, it is often approximated first by a combination of lower degree curves, such as lines and circular arcs, and interpolation is performed on these curves.

Previous algorithms suffer from lack of smoothness in the motion, errors in position and varying surface speed. In an attempt to address these problems, research in our group has investigated methods for smoothing the pulses after they have been generated [Steiger et al, 1994], [Stout et al, 1994].

More recently algorithms have been developed by our group to improve the interpolation of straight lines and circular arcs directly [Chow et al, 2002], [Chow, 2003], [Poliakoff et al, 2005]. Using simulation, these algorithms have been demonstrated to reduce errors in position, reduce unnecessary fluctuations in surface speed and achieve smoother motion. The algorithms are based on the geometry of the line or arc and use distance along the curve as a parameter to synchronise the axes. At the same time smoothness of motion is achieved by keeping the pulse rate constant (for lines) or adjusting it gradually (for arcs). Acceleration algorithms have also been developed for use with these algorithms to allow high speed machining. Smooth acceleration and deceleration are achieved by gradually increasing and decreasing the speed at the beginning and end of the motion, respectively. This paper describes how a number of simple simulations have been used to allow the new algorithms to be evaluated for both errors in position and smoothness of the motion. Section 2 describes some previously used interpolation algorithms. Then the simulation methods are explained in Section 3 before the new algorithms are presented in the later sections. We have investigated the algorithms for the 2D case but the ideas can be generalised to 3D.

2. INTERPOLATION ALGORITHMS:

Previous interpolation algorithms for lines and arcs include the Digital Differential Analyser (DDA) [Papaionnou, 1979], the Search-Step algorithm [Valentino and Goldberg, 2000] and the Direct Search algorithm [Massory and Koren, 1978]. The DDA interpolator performs digital integration on velocity to obtain the positions to be reached at fixed intervals of time (Δt) and is expressed by $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{v}_k \Delta t$, where the position at the start of the i^{th} interval is $\mathbf{x}_i$ and $\mathbf{v}_i$ is the corresponding velocity [Papaionnou, 1979]. Over the i^{th} interval the motion is generated in the direction of the tangent to the curve at the start of the interval. There is often some deviation from the required path, because in general the velocity is not constant over the interval. However, since the velocity is

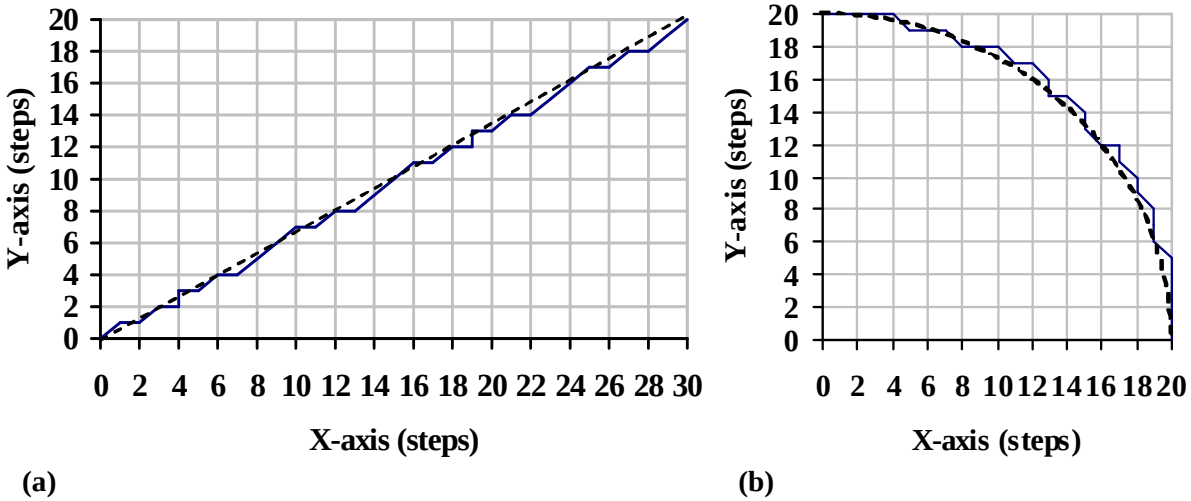

FIGURE 1. *The path plot from the Zero Order Simulation of the DDA interpolation at 0.3 m/min. for (a) the straight line from (0, 0) to (30, 20) and (b) the circular arc from (20, 0) to (0, 20) with centre (0, 0) (measured in steps of size 0.1mm.). The required path is shown dashed and the largest errors are 0.55 and 0.81, respectively.*

involved in the interpolation algorithm, the path will be machined at close to the required speed.

The Direct Search algorithm [Massory and Koren, 1978] is an improvement of the earlier Search-Step algorithm and both are based on the local geometry of the curve. They are used with curves defined in implicit form, where every point on the curve satisfies an equation of the form $f(x,y)=0$. For points lying off the curve $f(x,y)\neq 0$ and the magnitude of $f(x,y)$ increases with the error in position. The algorithm chooses the next interpolated point to be the one closest to the desired curve, i.e. the one for which $f(x,y)$ is closest to 0. For the Search-Step algorithm there are four possible moves at any stage, because it allows a move by one step but only in a single axis. For Direct Search a move of one step simultaneously in both axes is also allowed, so the number of possible moves is eight. Unlike the DDA algorithm, these two algorithms do not have full control of the surface speed during interpolation. Because a move can be of length 1 along a single axis or $\sqrt{2}$ when both axes are involved, the value of the resultant speed depends on the direction of the motion, i.e. the tangent to the curve. Thus, for interpolation of a 2D circular arc the speed can vary by up to a factor of $\sqrt{2}$.

In the next section we use the DDA and Direct Search algorithms to illustrate the simulation methods before the new algorithms are introduced. For evaluation of interpolation algorithms it is important to investigate both errors in position and the speed on the individual axes. Three different simulation methods have been used. None of them is completely realistic but, between them, they allow us to make comparisons between the chosen algorithms.

3. SIMULATION METHODS:

For accuracy of path following we have used two simulation methods, Zero Order and Second Order, and have found that they are useful in different ways. For simulation of speed we have used a first order method which is effectively an estimate of the pulse rate.

3.1. Simulation of Path Following:

Simulation of path following has enabled us to evaluate errors in position [Chow, 2003] and we present two simulation methods here. The first method, Zero Order Simulation, does not take into consideration the response time of the stepper motor, i.e. the stepper motor is assumed to move by one step instantaneously when a command pulse is received. Thus, in detail this simulation is very far from the motion of a real machine but it does allow the sequence of pulse timings on the two axes to be seen, as shown in Figure 1, where the simulation is shown for one line and one arc. Some moves are in a single axis, giving a vertical or horizontal line. In other cases there is a move in both axes simultaneously, i.e. a diagonal line. Thus, the plot from Zero Order simulation does not depend much on the speed but provides an overview of the expected path of the machine and has been used to provide a simulator to check the path before cutting [Henrich et al. 2005]. Effectively it indicates only the number of pulses in each axis and the sequence in which the pulses are sent to the two motors. We explain later how it can be misleading at a detailed level when errors are considered.

The second method, Second Order Simulation, is likely to be closer to the behaviour of a real system than the previous one, because it takes into account the fact that the motor cannot respond

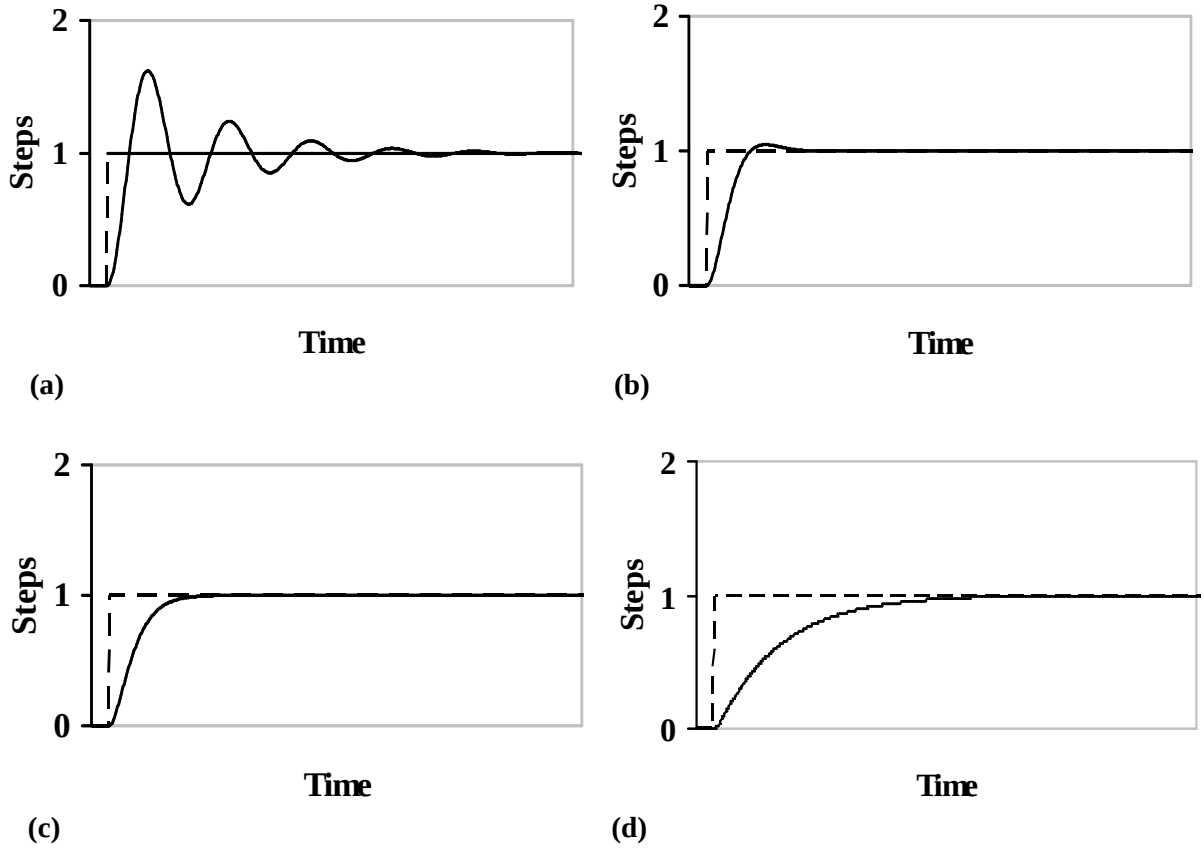

FIGURE 2. *Examples of the Second Order Simulation of one pulse on a single axis for different values of the damping factor: (a) 0.15 , (b) 0.7, (c) 1.0 and (d) 2.5.*

instantaneously [Christodoulou, 2000]. This simulation system uses as input the Zero Order simulation of each axis separately (including the timings for every pulse) before they are combined to give the simulated path. The Second Order simulator is still not entirely realistic but it is nevertheless useful, because it is expected to have a smoothing effect similar to that of the motor and drive circuits. For the Second Order simulation two parameters are needed: the natural frequency and the damping factor. We have chosen the natural frequency to be 100 Hz, which is a typical value for a stepper motor, and a damping factor of 0.7. Figure 2 shows the importance of the damping factor in determining the resulting motion. Examples of the result of Second Order Simulation for one pulse are shown for four different values of the damping factor. For a low value of damping factor 0.15 in (a) there is a large overshoot followed by a number of oscillations which gradually settle down, whereas for a large value 2.5 in (d) there are no oscillations but it takes a considerable time to complete the step. The value 1.0, in Figure 2(c), is known as critical damping and is ideal, because it is the smallest value for which there are no oscillations and thus takes the shortest time to complete the step without overshooting. In practice it is difficult to achieve critical damping and a compromise has to be reached between avoiding overshoot and not taking too long to complete the step. A value of 0.7, in Figure 2(b), has been chosen to represent this situation and has an almost negligible overshoot.

As an example of the simulations, one line and one arc have been interpolated at surface speed 500 steps/sec, i.e. 0.3 m/min. The line is from (0,0) to (30, 20) and the arc is from (20,0) to (0, 20) with the arc centre at (0,0) (all distances given in steps, which are of length 0.01mm.). Figures 1 and 3 show the Zero Order simulation of the DDA and Direct Search algorithms, while the Second Order simulations are given in Figures 4 and 5. The simulation of axis speed is described in the following section. In Figures 1(b) and 4(b) it can be seen that the direction of the motion tends to lag behind a little, so the path is mostly outside the arc. This is because the DDA method assumes that the velocity is constant over the time interval.

3.2. Simulation of Speed:

For simulation of axis speed we have estimated the speed at the time of each pulse based on the time between that pulse and the next. Thus, at time $t_x(n)$, the time of the n^{th} pulse, x-axis the speed is given by

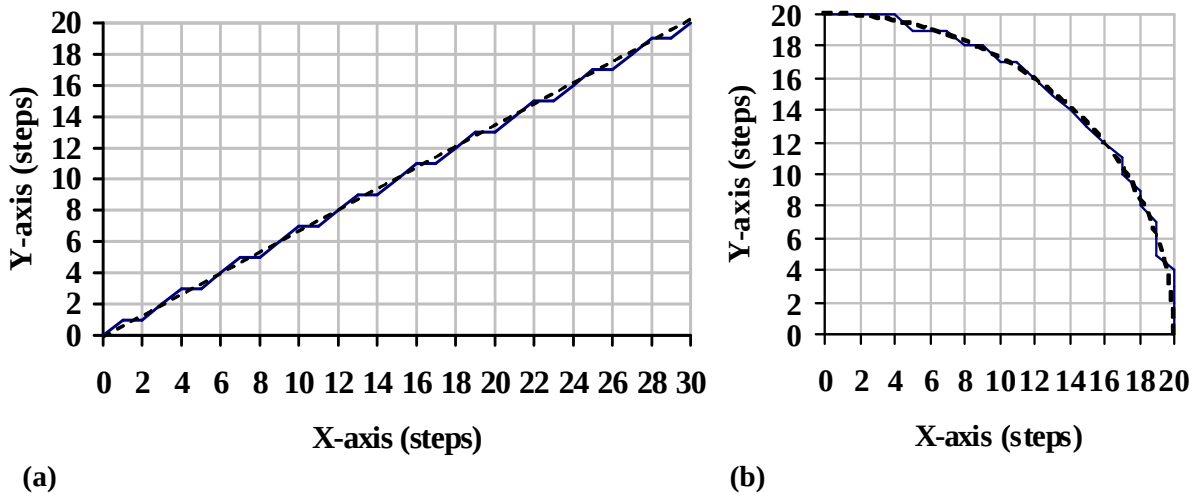

FIGURE 3. *Path plots from the Zero Order Simulation of the Direct Search interpolation for the line and arc from Figure 1. The largest errors are 0.28 and 0.40, respectively.*

$\frac{L}{(t_x(n+1)-t_x(n))}$ (and similarly for *y*), where *L* is the step length [Chow et al, 2002], [Chow, 2003]. This is effectively a first order simulator. Again it is not very realistic in detail and is actually estimating the pulse frequency. A real motor will allow some smoothing but will often suffer from vibrations, exacerbating the situation, especially when the pulse rate fluctuates. Therefore it is the fluctuations in the pulse rates that need to be detected. Figure 6 shows the speed plots for the *y*-axis with the DDA algorithms. Only the *y*-axis speed is shown, because the speed plot is similar for *x* in each case (although there are more fluctuations in the linear case and for the arc the speed increases rather than decreases). The *y*-axis speed plots for the Direct Search algorithms are shown in Figure 7. For the arc the *x*-axis speed plot is again similar but for the line it is constant (at 500 steps/sec). Thus, for both interpolation algorithms in these examples at least one axis has sudden changes or fluctuations in speed.

Ideally, for smooth motion along a straight line the speeds on both axes would be constant, while for an arc they would each follow the appropriate part of a sine wave. Our new algorithms are able to generate pulse trains close to the ideal and without any sudden fluctuations in speed.

4. THE NEW INTERPOLATION ALGORITHMS:

We have developed new linear and circular arc interpolation algorithms to reduce the problems of lack of smoothness in the motion and varying surface speed [Chow, 2003], [Poliakoff et al., 2005]. At the same time it is important that errors in position are not increased. On a particular axis the pulses are generated with timings which allow the machine to follow the required path according to the geometry of the path, while moving at the required speed. Unlike the previous methods, we have used a parameter-based method to synchronise the individual axes. For both lines and arcs, the surface speed can be calculated from the distance, *s*, along the curve, so this distance has been found to be a suitable parameter. Indeed for straight lines and circular arcs the other parameters normally used are proportional to the distance along the curve, e.g. *x* for a line or the angle of turn for an arc. The calculation of the pulse timings are based on the path geometry and the required surface speed. For interpolation we assume the overall speed is constant. Then the acceleration algorithms can be used when changes in speed are required, as described in Section 6.

The initial idea was to imagine a point travelling along the curve at the required speed and then generate a pulse to the *x*-axis motor every time a step in the *x* direction has been completed (and similarly for the *y*-axis). For 2D linear interpolation, a straight line is given in parametric form as:

$$x(s) = x_0 + s\cos\theta\,,\quad y(s) = y_0 + s\sin\theta\,, \qquad (1)$$

where *s* is the distance along the line, (x_0, y_0) is the start point and θ is the (constant) angle between the line and the *x*-axis. If *L* is the length of one motor step, then the distance in *x* corresponding to the n^{th} pulse is *nL* and the corresponding distance along the curve $s_x(n)$ can be calculated. Thus for both *x* and *y* we have:

$$s_x(n) = \frac{nL}{\cos\theta}\,,\quad s_y(n) = \frac{nL}{\sin\theta}\,. \qquad (2)$$

If a constant speed *V* is required, the corresponding timings of the n^{th} pulse for each axis is given by:

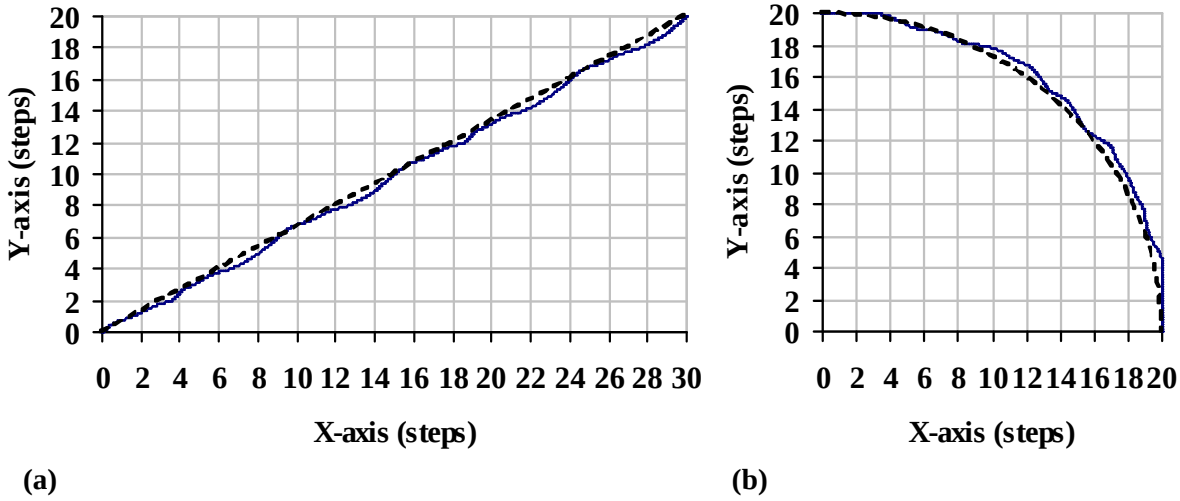

FIGURE 4. *Path plots from the Second Order Simulation of the DDA interpolation from Figure 1. The largest errors are 0.38 and 0.63, respectively.*

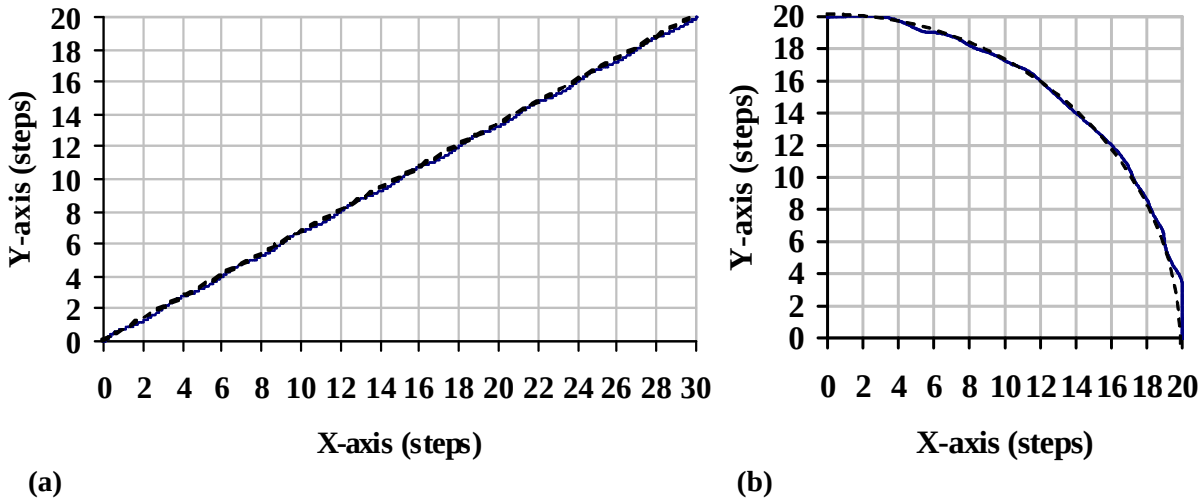

FIGURE 5. *Path plots from the Second Order Simulation of the Direct Search interpolation from Figure 3. The largest errors are 0.13 and 0.29, respectively.*

$$t_x(n) = \frac{nL}{V\cos\theta},\ t_y(n) = \frac{nL}{V\sin\theta}. \qquad (3)$$

However, we have found that, although smoothness was improved, errors in position could be exacerbated by this method. In Figures 8(a) and 9(a) it can be seen that the path is, on average, just below the required line, as is reflected in the value of the average error as 0.14 and 0.13, respectively. For the arc, Figures 8(b) and 9(b), the path starts to move outside the required arc, then follows it quite well but moves somewhat inside it near the end. For both line and arc each simulation has a rather high value for the largest error. After further investigation we have found that it is possible to obtain a reduction in the errors can be obtained without losing the smoothness of the speed plots.

If we assume that the motor moves instantaneously when a pulse is sent, then the initial method will always cause a delay in each axis, which will vary between 0 and L with an average value of ½L. This is because the algorithm waits for a complete step before generating a pulse. We have estimated the error in position for the case of a delay in both x and y of ½L. For a line at angle θ to the x-axis a point (x_1, y_1) on the line satisfies for some constant C:

$$x_1\sin\theta - y_1\cos\theta + C = 0. \qquad (4)$$

Then the distance from the line of a point $(x_1 - ½L, y_1 - ½L)$ representing the actual position is given by:

$$E = (x_1 - ½L)\sin\theta - (y_1 - ½L)\cos\theta + C. \qquad (5)$$

Substituting for C from equation 4 we obtain:

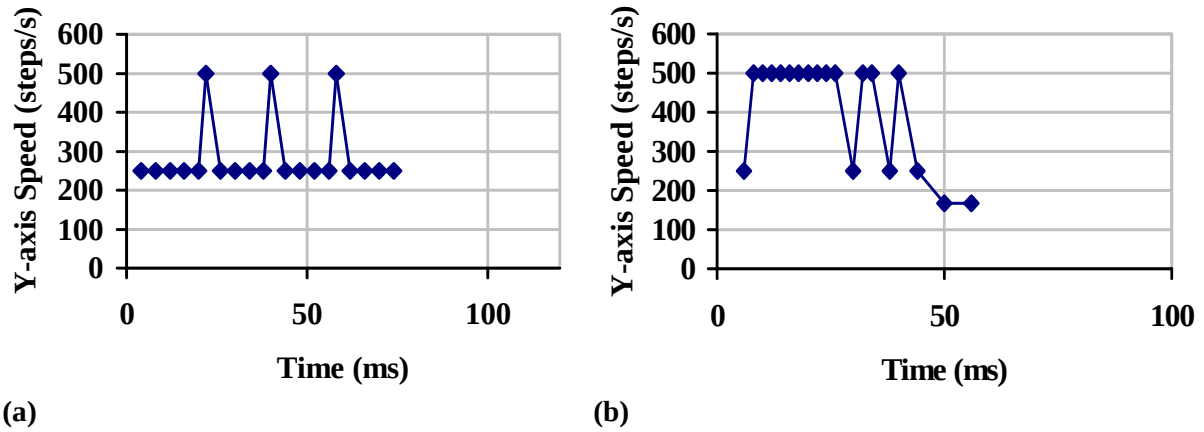

FIGURE 6. *Speed plots for y-axis from the DDA interpolation for the line and arc from Figures 1 and 4. (The plots for the x-axis are similar, except that for the arc it starts low and then increases to the higher value.)*

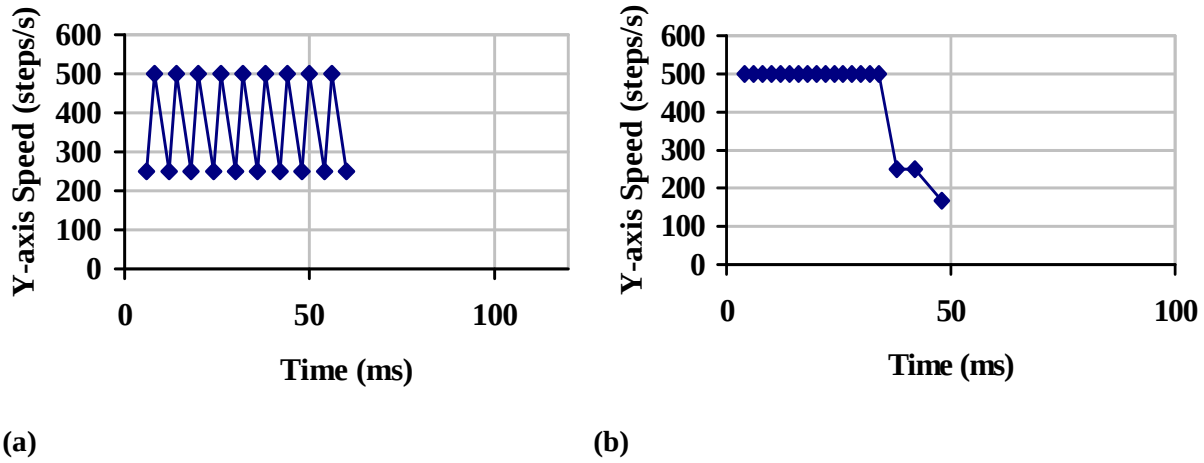

FIGURE 7. *Speed plots for y-axis from the Direct Search interpolation for the line and arc from Figures 3 and 5. (In the case of the arc the x-axis plot again is similar but starts low and then increases to the higher value. For the line the x-axis plot is constant.)*

$$E = \tfrac{1}{2}L(\cos\theta - \sin\theta). \qquad (6)$$

This applies when θ is strictly between 0 and 90° but the error is 0 when θ = 0 or 90°, because then only one axis is involved. For the line, we obtain E = 0.14, which agrees well with the average error from the simulations (0.14 from Zero Order and 0.13 from Second Order). If, however, the timing of the pulse is calculated when the imaginary point is mid-way between the two steps, the average error E will be 0. Thus we replace n with $n - \tfrac{1}{2}$ in the formulae for the n^{th} pulse and equation 2 becomes:

$$s_x(n) = \frac{(n-\frac{1}{2})L}{\cos\theta},\ s_y(n) = \frac{(n-\frac{1}{2})L}{\sin\theta}, \qquad (7)$$

and equation 3 becomes:

$$t_x(n) = \frac{(n-\frac{1}{2})L}{V\cos\theta},\ t_y(n) = \frac{(n-\frac{1}{2})L}{V\sin\theta}. \qquad (8)$$

A similar argument can be applied in the case of circular arcs or, more generally, to other curves, because at any instant the motion can be thought of as moving in a straight line along the tangent to the arc.

For a circular arc in the first quadrant with radius R, centre (x_c, y_c) and start angle α_0 the parametric form is:

$$x(s) = x_c + R\cos\left(\alpha_0 \pm \frac{s}{R}\right),$$
$$y(s) = y_c + R\sin\left(\alpha_0 \pm \frac{s}{R}\right) \qquad (9)$$

(with the signs corresponding to anticlockwise or clockwise motion, respectively). Thus, for the n^{th} pulse on each of the axes we obtain:

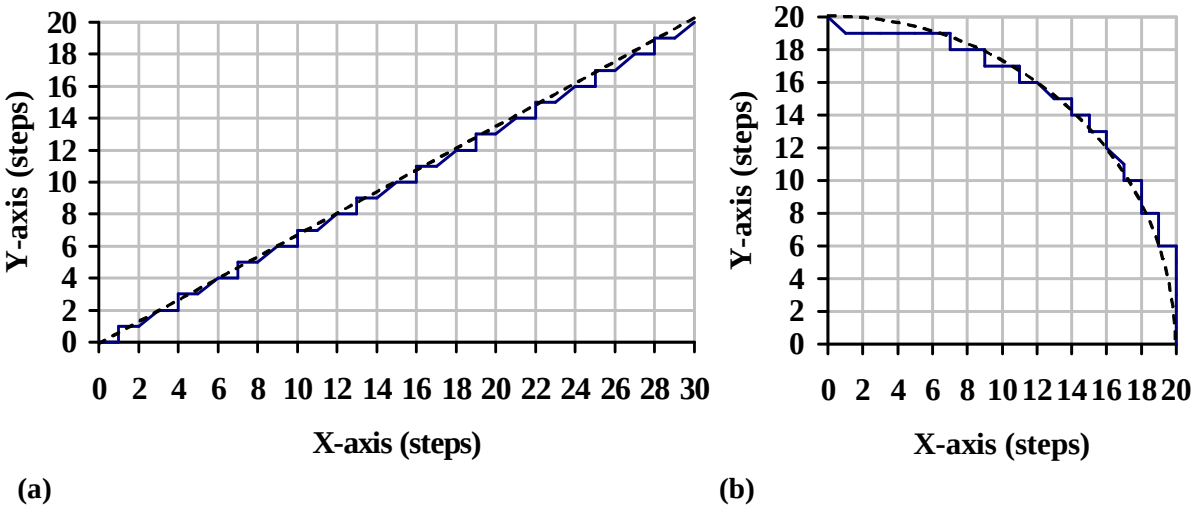

FIGURE 8. *Path plots from the Zero Order Simulation for the Initial New interpolation algorithms for the line and arc from Figure 1. The largest errors are 0.55 and 0.97, while the average errors are 0.14 and -0.03. (Errors are positive below the line or outside the arc, and negative otherwise.)*

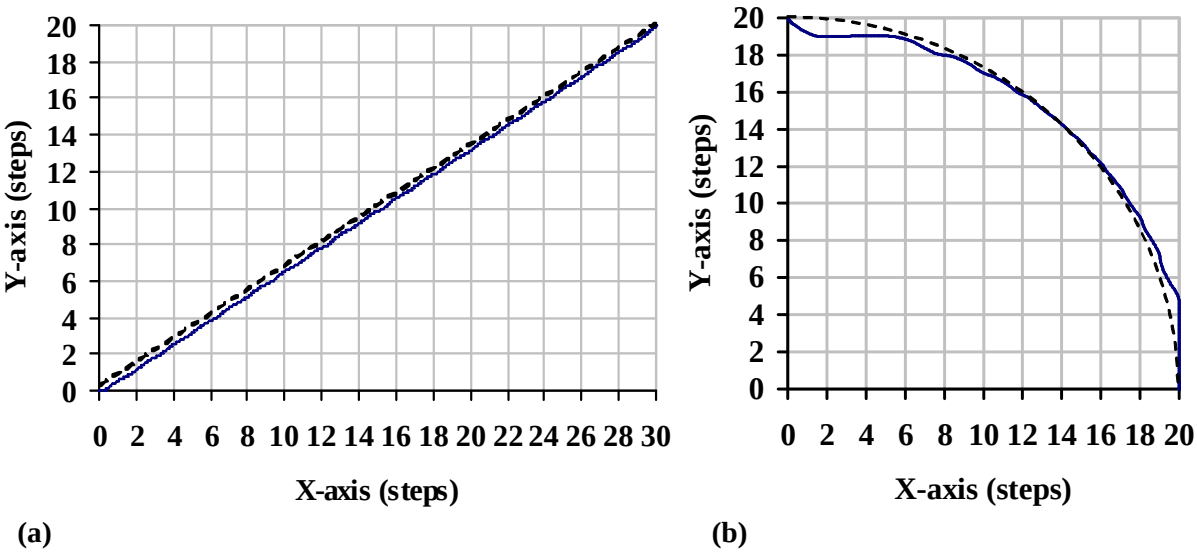

FIGURE 9. *Path plots from the Second Order Simulation for the Initial New interpolation algorithms from Figure 8. The largest errors are 0.19 and 0.93, while the average errors are 0.13 and -0.07.*

$$s_x(n) = \pm R\left[\cos^{-1}\left(\cos\alpha_0 \mp \frac{(n-\frac{1}{2})L}{R}\right) - \alpha_0\right],$$
$$s_y(n) = \pm R\left[\sin^{-1}\left(\sin\alpha_0 \pm \frac{(n-\frac{1}{2})L}{R}\right) - \alpha_0\right]. \qquad (10)$$

For constant speed V, as in the case for linear interpolation, the pulse timings can then be calculated by dividing by V. For the second, third and fourth quadrants similar expressions can be derived.

Figures 10 and 11 show the path plots for the new algorithms using Zero and Second Order simulation, respectively. For the line the average error is now 0.0 in both cases, and for the arc the value is much reduced (0.06 and 0.02, respectively). Again, the speed plots in Figure 12 are for the *y*-axis only but those for the *x*-axis are similar, although for the arc the speed increases rather than decreases. It can be seen that for both line and arc there are now no sudden fluctuations in speed. The speeds on the two axes are a great improvement on the previous algorithms and they are close to the ideal, i.e. constant or changing sinusoidally, respectively. Therefore with the new algorithms it is less likely that there will be undesirable vibrations of the motors.

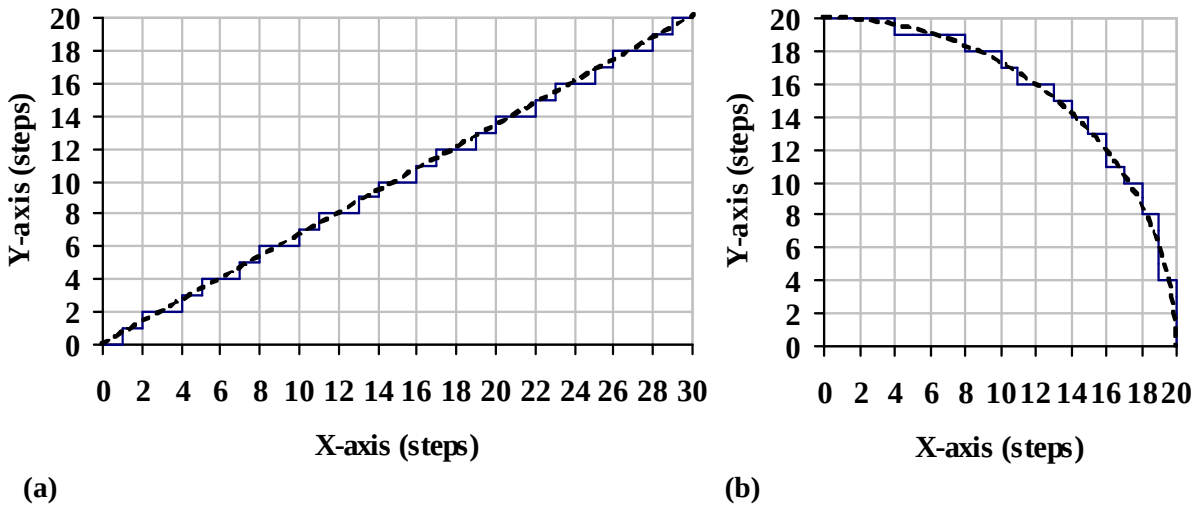

FIGURE 10. *Path plots from the Zero Order Simulation for the New interpolation algorithms for the line and arc from Figure 1. The largest errors are 0.55 and 0.62, while the average errors are 0.00 and 0.06.*

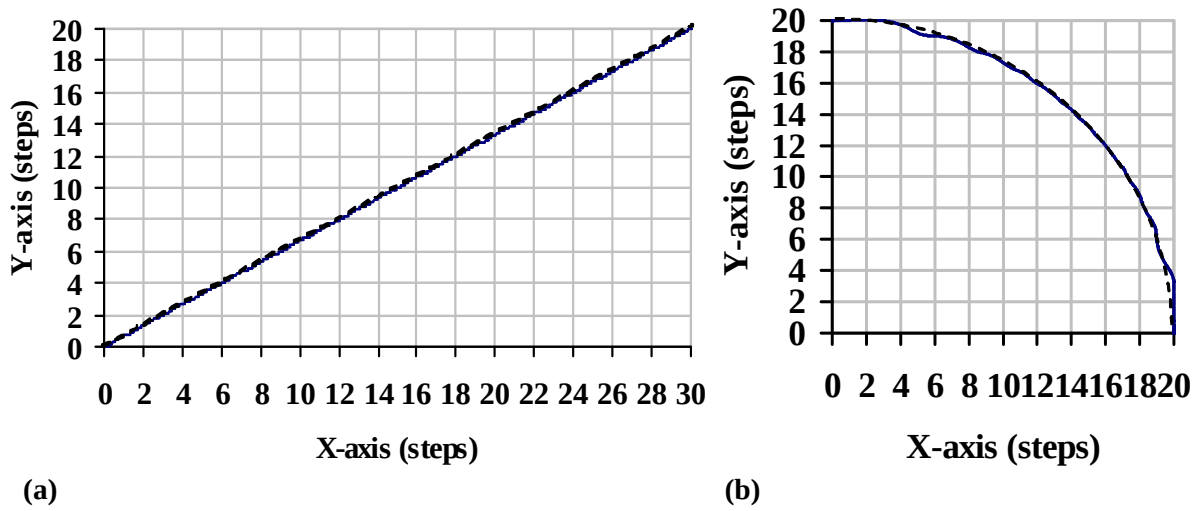

FIGURE 11. *Path plots from the Second Order Simulation for the New interpolation algorithms from Figure 10. The largest errors are 0.06 and 0.28, while the average errors are 0.00 and 0.02.*

5. PREVIOUS ACCELERATION ALGORITHMS:

Two types of acceleration technique commonly used for high-speed machining are linear and parabolic acceleration, in which the speed changes either linearly or parabolically with time. The minimum speed at which a motor can move depends on the rotor and load inertia [Palmin and Shlain, 1986] and this will also be the speed at which the motor can start moving from rest.

Linear acceleration results in slow acceleration and much of the available torque is not utilised [Palmin and Shlain, 1986]. This is because a stepper motor can achieve high acceleration at low speeds but the achievable acceleration decreases as the speed increases. Thus, for linear acceleration either the acceleration rate has to be limited to take this into account or the maximum speed has to be reduced. Both cases are disadvantageous, because the total machining time will then be increased, as can be seen in Figure 13. A parabolic acceleration algorithm allows a higher acceleration at low motor speed combined with a lower rate at high speed. With this method much more of the available motor torque can be utilised and therefore the stepper motors can be used effectively at higher speeds. Moreover, [Kim et al, 1994] have shown that machining accuracy is improved with parabolic acceleration in comparison with linear acceleration. This is likely to be because any linear acceleration algorithm involves sharp discontinuities in the acceleration, which tend to cause increased vibration and overshoot.

For a parabolic acceleration algorithm the equations for speed during the acceleration and deceleration phases are quadratic functions of time and were

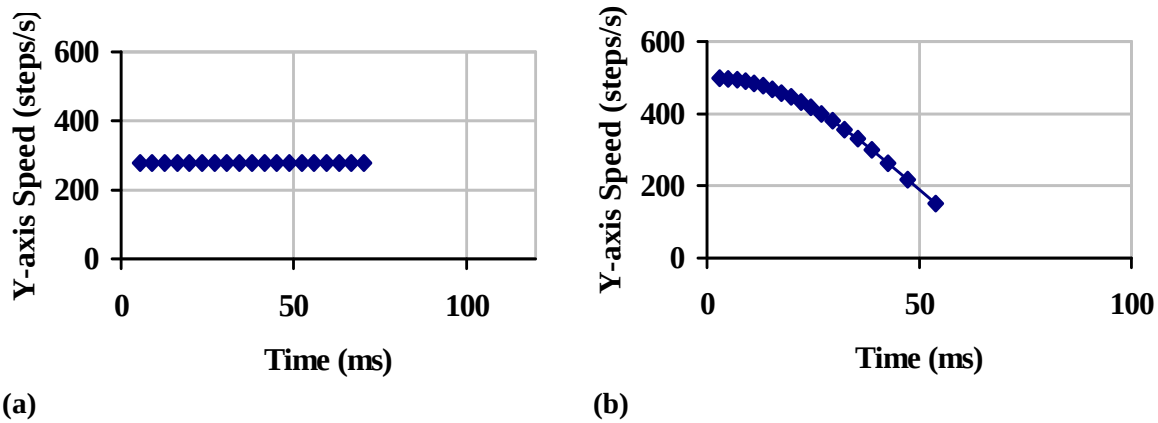

FIGURE 12. *Greatly improved speed plots for the y-axis from the New interpolation algorithms from Figures 10 and 11. (The plots for the x-axis are similar, except that for the line it is scaled up and for the arc the speed starts low and increases.)*

represented by Palmin et al. in terms of the acceleration (or deceleration) time T, the maximum speed V_m and minimum speed V_0 as follows:

$$V = pt^2 + qt + V_0, \tag{11}$$

where $p = \frac{V_0 - V_m}{T^2}$ and $q = -2pT$. Palmin et al. calculated the approximate timing t_n for every command pulse by assuming that:

$$V_n = V_{n-1} + a_{n-1}(t_n - t_{n-1}), \tag{12}$$

where V_n is the speed at time t_n and a_n is the acceleration at time t_n.

6. THE NEW ACCELERATION ALGORITHMS:

Our new parabolic acceleration and deceleration algorithms have been developed for use with the new interpolation algorithms. They produce pulse timings which allow the stepper motors to accelerate or decelerate smoothly. We do not use the approximation given by Palmin et al. from equation 12 above.

In order to maintain the desired shape during path following, the new parabolic acceleration and deceleration need to be applied to the surface speed and not to the speed for an individual axis. Using Palmin's notation from equation 11, the distance travelled, s, is given, by:

$$s = \int V dt = \frac{pt^3}{3} + \frac{qt^2}{2} + V_0 t. \tag{13}$$

The distance, s, for every axis step movement is obtained by interpolation from equations 7 and 10. Every pulse timing can then be calculated by solving the cubic equation 13. We have implemented this using Newton-Raphson iteration.

For the evaluation of the acceleration algorithms we have chosen a desired surface speed 4.8 m/min., but with the step size 0.01 mm., as before. Both acceleration time and deceleration time were set to 0.15 sec. and the initial speed was 0.12 m/min. We have used a different line and arc, because they need to be longer to allow the maximum speed to be reached before the end and slow down again. The straight line is from (0, 0) to (3000, 2000) (in steps) and the arc is from (2000, 0) to (0, 2000) with centre (0, 0). We have chosen the deceleration time to be the same as acceleration time in our examples but they do not have to be the same; in practice they could be chosen to suit the properties of the motor used.

Figure 13(b) shows the speed plot for the *y*-axis of the line and it can be seen that the speed has a parabolic shape at the beginning and end during acceleration and deceleration and is constant in between. The speed plot for the *x*-axis is similar but the speed is scaled up in the ratio 3 to 2. Thus, the overall speed also follows the same shape (also scaled up). Figure 14 shows the speed plots for both axes with the arc. In this case it is harder to distinguish the acceleration and deceleration phases, because, although the overall speed is constant during the middle phase, the speed for the *y*-axis follows a sinusoidal shape similar to that in Figure 12(b) and for the *x*-axis it is another part of a sine wave. The plot of the overall speed is similar to that for the line but the total time is shorter, because the total distance along the arc is shorter.

7. RESULTS AND DISCUSSION:

When they are compared with the previous algorithms, the new algorithms have made a great improvement to the smoothness of the speed plots. The results of the simulations of path following for new algorithms can be summarised in Table 1 below.

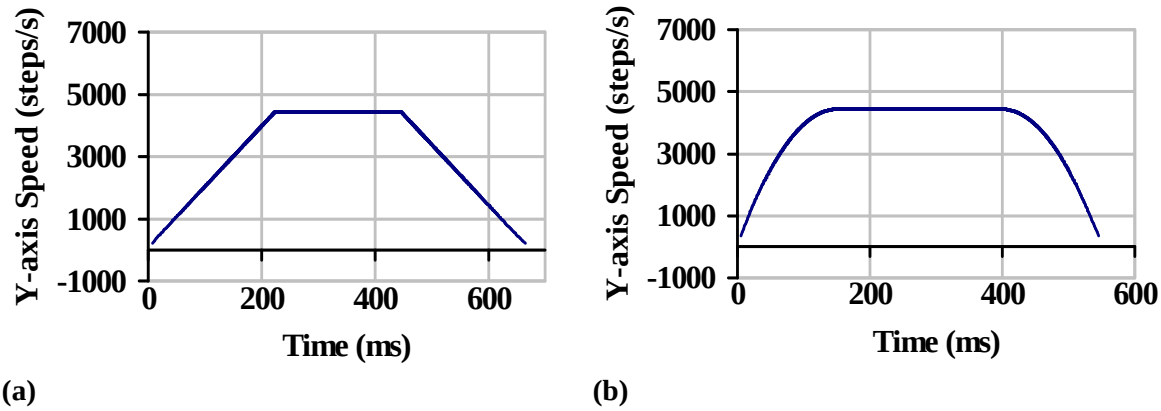

FIGURE 13. *The y-axis speed plot for the longer straight line from (0, 0) to (3000, 2000) using (a) linear acceleration and (b) parabolic acceleration. The maximum speed was 4.8 m/min. and the starting speed was 0.12 m/min. (The plots for the x-axis and verall speed in each case are similar but with the speed scaled up.)*

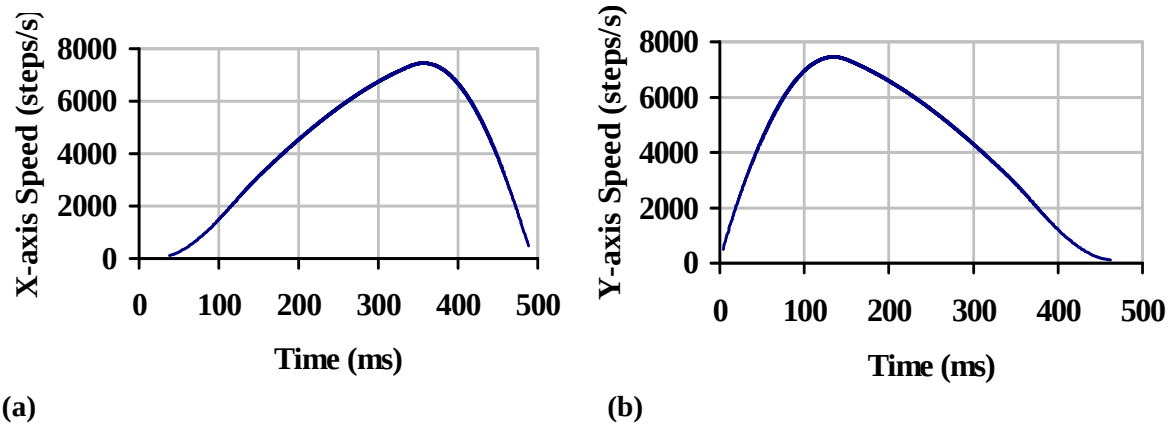

FIGURE 14. *The speed plots for the longer circular arc from (2000, 0) to (0, 2000) with centre (0, 0) using parabolic acceleration: (a) x-axis and (b) y-axis. Again the maximum speed was 4.8 m/min. and the starting speed 0.12 m/min. The acceleration stopped at 150 ms. and the deceleration started at 340 ms. (The plot for the overall speed is similar to that in Figure 13(b) but scaled up and the time at maximum speed is shorter.)*

Table 1: Position Errors for the Interpolation Algorithms using Two Simulation Methods

Simulation Type	Interpolation Algorithm	Largest Error (steps)	
		Line	Arc
Zero Order	DDA	0.55	0.81
	Direct-Search	0.28	0.40
	New Algorithm	**0.55**	**0.62**
Second Order 0.1 m/min	DDA	0.57	0.83
	Direct-Search	0.30	0.43
	New Algorithm	**0.28**	**0.42**
Second Order 0.3 m/min	DDA	0.38	0.63
	Direct-Search	0.13	0.29
	New Algorithm	**0.06**	**0.28**

From Table 1 it is clear that using the Second Order simulator gives a reduction in the largest error for the new algorithms compared with both the previous ones. This is not always the case when the Zero Order simulator is used. We have investigated the Zero Order simulator and concluded that it is not suitable for testing for small errors below the length of one step, as explained below.

The plot from the Zero Order simulator must always pass through intermediate points with integer coordinates when measured in steps. Therefore, for a straight line at an angle θ to the x-axis, unless θ is an integer multiple of 45°, the errors in position can be considerable. Without diagonal lines in the simulated path the error may be up to $\frac{1}{2}\sqrt{2}$, i.e. about 0.71. However, diagonal lines will occur only when two pulses on different axes are calculated with *identical* timings. Even a very short time interval between the pulses could therefore increase the error by up to 0.71 in the simulation, whereas in a real situation the difference would be very small. Therefore, with Zero Order simulation an estimated error may itself have an error of up to 0.71 of a step.

Thus, small errors cannot be considered to be significant in judging the likely deviation from the required path.

If the speed were reduced more and more, the behaviour under Second Order simulation would become closer and closer to the Zero Order simulation. This explains why the errors are reduced as the speed increases. The Zero Order simulation will not normally change with increasing speed, unless the resolution of time causes two timings which are close but different a low speed to become identical at higher speed.

The speed plots in Figures 13 and 14 show that the pulse trains are also smooth when the new parabolic acceleration algorithms are used for the longer line and arc. Table 2 shows the errors obtained using the two simulation methods and the two acceleration algorithms.

Table 2: Position Errors for the Acceleration Algorithms using Two Simulation Methods

Simulation Type	Largest Error (steps)	
	Line	Arc
Zero Order (both linear and parabolic acceleration)	0.55	0.71
Second Order (both linear and parabolic acceleration)	0.59	0.53

From the detailed error plots (not shown) it has been found that the errors are largest at the beginning and end of the motion, particularly for the larger radius arc. We have found that the largest error for a straight line depends very much on the direction of the line. We have simulated the interpolation of a line at several different angles to the *x*-axis at 0.3 m/min. and found that the largest error ranges from 0 to at least 0.52. When the line is parallel to one axis, only one motor is involved, so the error is 0. For a line at 45° both motors will receive an identical stream of pulses, so the resulting path will follow the line exactly under our simulations. The line in our example is at 33.7° and the largest error is only 0.06. However, when the angle is close to 0, such as 0.57°, one motor is moving very slowly and then the error becomes 0.52. This is summarised in Table 3.

Table 3: Position Errors for Interpolation of a Line at Speed 0.3 m/min. (Second Order Simulation)

Line Angle	0°	0.57°	33.7°	45°
Largest Error (steps)	0	0.52	0.06	0

The algorithms have been tested on stepper motors and speeds of 7.5 m/min. have been reached. However, further work is needed to measure the motor behaviour in detail.

8. CONCLUSIONS:

The use of appropriate simulation methods has enabled the evaluation of the new interpolation and acceleration algorithms for stepper motors before testing on real motors. The Zero Order simulator is the simplest and is useful to obtain an overall view of the path. However, it is not suitable to investigate the detail of deviations from the path. The Second Order simulator has allowed us to compare the new algorithms with previous ones for accuracy of path following. The axis speed simulator has shown that the expected motion with the new algorithms is much smoother than for the previous ones and less likely to cause vibrations. Future work will involve extending the algorithms to other types of curves, such as splines.

9. ACKNOWLEDGEMENTS:

We thank Nottingham Trent University and Axiomatic Technology Ltd. for supporting this project.

10. REFERENCES:

Chow Y.K., Poliakoff, J.F. and Thomas P.D. 2002, "Interpolation and Acceleration Algorithms for Stepper Motors – A Parametric Approach". In *8th IEEE Int. Conf. on Methods and Models in Automation and Robotics*, Szczecin, Poland.

Chow Y.K. 2003, "Parametric Interpolation Algorithms for Motion Control". PhD Thesis, The Nottingham Trent University.

Christodoulou, N. 2000, *The Programmable Graphical Simulation of Stepping Motors Driven CNC 2-Axes Machine*. MSc Thesis, Department of Computing, Nottingham Trent University.

Henrich C., Poliakoff J.F., Orton P.A. 2005, "Simulation of 2D Cutter Paths for CNC Machines Controlled by Stepper Motors". In *Proc. 8th ICCMS (8th Int. Conf. on Computer Modelling & Simulation*, Oxford, UK.

Kim D., Song J. and Kim S. 1994, "Dependence of Machining Accuracy on Acceleration /Deceleration and Interpolation Methods in CNC Machine Tools". In EEE – IAS Annual Meeting, Denver, USA. Pp1898-1905.

Massory O. and Koren Y. 1978, "The Direct-Search Method In CNC Interpolators". In *ASME Winter Annual Meeting*, San Francisco, Calif. Pp2–8.

Palmin S. and Shlain V. 1986, "Stepper Motor Controller with Parabolic Velocity Profile allows Maximum Torque". In Control Engineering.

Papaioannou S. 1979, "Interpolation Algorithms For Numerical Control". In *Computers In Industry*, No.1. Pp27-40.

Poliakoff J.F., Chow Y.K., Orton P.A., Howson M. and Al-Dabass D. 2005, "Simulation for Development of 2D Interpolation Algorithms for Stepper Motors". In *Proc. 8th ICCMS (8th Int. Conf. on Computer Modelling & Simulation*, Oxford, UK.

Steiger W., Sherkat N. and Thomas P. 1994, "A Stepping Motor Control Algorithm For Smooth Continuous Path Motion". In *Proc. of the IEEE Int. Conf. on Control '94*, University of Warwick, UK, Vol.1, No. 389. Pp816-821.

Stout A.J., Orton P.A. and Thomas P.D. 1998, "Improving Continuous Path Motion Using Stepper Motors". In *Proc. 10th European Simulation Symposium & Exhibition*, The Nottingham Trent University, Nottingham, UK.

Valentino J.V. and Goldenberg J. 2000, *Introduction to Computer Numerical Control*. Prentice-Hall.

BIOGRAPHIES:

Dr Janet Poliakoff is a Senior Lecturer in the School of Computing and Informatics at Nottingham Trent University. After a BA and Part III Mathematics at the University of Cambridge, she worked for the Open University as a tutor in Mathematics before starting research under the supervision of Prof. Peter Thomas at Nottingham Trent University, where she received a PhD in Computing in 1993. Her thesis was entitled "The Digital Representation of Two-Dimensional Cutter Paths". Her research interests include: digital representation of (2- and 3-D) curves and surfaces; fairing of B-spline curves; measurement of surfaces using laser scanners; smooth motion generation for stepper motors; monitoring of rotary motion**.**

Dr. Yuan Kai Chow graduated from Nottingham Trent University in 1999. He then worked as a research student, also at Nottingham Trent University, under the supervision of Dr. Janet Poliakoff, Prof. Peter Thomas and Dr. Paul Orton. Dr. Chow was awarded his PhD in July 2004 and his thesis is entitled "Parametric Interpolation Algorithms for Motion Control". His research interests are: Computer Numerical Control (CNC), motion control systems, smooth path following and stepper motors.

Dr Paul Orton is a Senior Lecturer in the School of Computing and Informatics at Nottingham Trent University. After receiving a BSc in Automatic Control Engineering from Sussex University, he worked in the engineering industry for a number of years, producing several patented designs for machine tools. Dr Orton received a PhD from Nottingham Trent University in 1990 on "Adaptive Control and Instrumentation for Precision Grinding Machines". His research interests include: intelligent instrumentation, including condition monitoring and control; development of patented Incremental Motion Encoder technology (IME) for monitoring rotary motion; smooth motion generation for stepper motors.