University of California at Davis
California Department of Transportation

THE BIG ARTICULATED STENCILING ROBOT (BASR)*

Volume IV

Phillip W. Wong, P.E.
Professor Bahram Ravani
Richard Blank
Jeff Hemenway
Richard McGrew
Ulrich Mueller
Dr. Walter Nederbragt
Robert Olshausen
Ken Sprott

AHMCT Research Report
UCD-ARR-98-01-15-01

Final Report of Contract
RTA-65X936

January 15, 1998

*This work was supported by the California Department of Transportation (Caltrans) Advanced Highway and Maintenance and Construction Technology Program at UC-Davis and by the Federal Highway Administration (FHWA).

DISCLOSURE STATEMENT

Design information, processes and techniques discussed within this report may be patent pending. Do not disclose to other agencies, persons, companies, or entities.

The Contractor grants Caltrans and the FHWA a royalty-free, non-exclusive and irrevocable license to reproduce, publish or otherwise use, and to authorize others to use, the work and information contained herein for government purposes.

DISCLAIMER STATEMENT

The research report herein was performed as part of the Advanced Highway Maintenance and Construction Technology Program (AHMCT), within the Department of Mechanical And Aeronautical Engineering at the University of California, Davis and the Division of New Technology and Materials Research at the California Department of Transportation. It is evolutionary and voluntary. It is a cooperative venture of local, state and federal governments and universities.

The contents of this report reflect the views of the author(s) who is (are) responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the offical views or policies of the STATE OF CALIFORNIA or the FEDERAL HIGHWAY ADMINISTRATION and the UNIVERSITY OF CALIFORNIA. This report does not constitute a standard, specification, or regulation.

Algorithms and Robotic Hardware Improvements for Painting of Roadway Markings

by

Richard M. Blank

B.S.M.E (California State University, Chico) 1994

THESIS

Submitted in partial satisfaction of the requirements for the degree of

Master of Science

in

Mechanical Engineering

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY of CALIFORNIA

DAVIS

Approved:

Bahram Ravani

Jan G. Velinsky

Committee in Charge

1997

Acknowledgements

I would like to thank my advisor Professor Bahram Ravani, for his invaluable guidance and support throughout my work for this thesis. Without him, this would not have been possible.

I would like to give thanks to Ken Sprott, Phil Wong, Walter Nederbragt, Richard McGrew, and Robert Olshausen for their incredible help and extensive knowledge.

I would like to thank Jeff Hemenway for spending much of his time working in the Student Shop manufacturing many components of the robot linkage.

I would like to offer my thanks to Professor Steve Velinsky and Professor Andy Frank for their valuable time and consideration in reviewing my thesis.

I would like to thank Ty Lasky for endless support of solutions to my thousands of questions.

I especially want to thank my parents for their love and support during my work at UC Davis.

Contents

List of Figures

List of Tables

Richard M. Blank
June 1997
Mechanical Engineering

Algorithms and Robotic Hardware Improvements for Painting of Roadway Markings

Abstract

This document is separated in two parts, Chapter 2 summarizes the research and development of the trajectory planning technique used on a robot in a roadway marking spray painting application, and Chapter 3 summarizes the mechanical redesign of the Stenciling Robot for improved accuracy and repeatability during the application of markings.

In Chapter 2, the trajectory planning technique provides the means for calculating robotic movement parameters that are necessary to paint all of the standard markings. The trajectory planning technique is generic in nature and can be adapted to many different fonts and markings. This thesis covers the basic concepts of development without the in-depth detail of the programming algorithms needed for implementation.

In Chapter 3, the redesign of the parallel/retraction mechanism is discussed in detail. Stiffness and weight were major concerns throughout the design. A comparison between the previous mechanism and the new design has shown an extreme increase in stiffness with a slight decrease in weight.

Chapter 1

Introduction

The painting of roadway markings is a slow, burdensome process and in many instances, it can be dangerous. The current process consists of closing down a lane of traffic, laying down stencils for a desired message, and coating the road surface and stencils with paint from a spray gun. This process must be repeated many times per year depending on the traffic for that area. The crew is exposed to traffic hazards each time this process is repeated.

To improve the safety of the work crew and speed the process of painting of roadway markings, a robotic system to automate this process is under way at the Advanced Highway Maintenance and Construction Technology Research Center (AHMCT) located at the University of California, Davis. Spray painting of roadway markings is a unique application for robotics. To produce quality markings a spray gun must be positioned precisely through out a preset trajectory.

This thesis presents the development of algorithms for trajectory generation and mechanical improvements of a robotic manipulator for painting of roadway markings.

1.1 Previous Research

Research in the area of robotics for use in spray painting applications has been ongoing for many years. At AHMCT, research on robotic path planning for painting of roadway markings has been in progress since 1992. A completed and successful project, the Stenciling Robot for aerial surveying premarks created by AHMCT, is such an example to show uses of this previous research. The robot was designed to paint 4 foot square premarks along highways for aerial surveying. [10] The premarks are combinations of straight black and white segments as shown in Figure 1.1.

Research in painting alphanumeric symbols has also been ongoing in parallel to the aerial premark. The previous path planning techniques can be viewed in detail in a conference paper by H. Kochekali and B. Ravani called *A Feature Based Path Planning and Referencing for Robotic Stenciling of Roadway Markings* see [5]. Some techniques are similar to the techniques described later in this thesis but can be best compared after reading both.

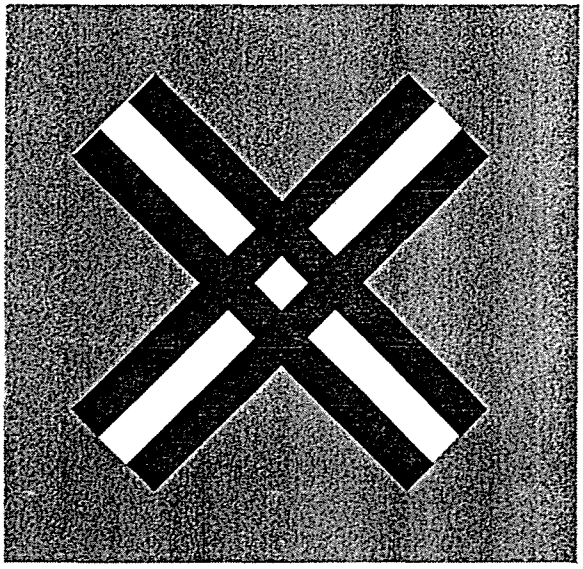

Figure 1.1: Aerial premark marking

1.2 Current Methods of Producing Roadway Markings

There are a variety of markers displayed on roadways to warn and inform drivers of incoming events. There are two standard materials used to produce these messages currently, paint and thermoplastic, both requiring the use of stencils. To produce a marker a maintenance worker lays down a stencil aligning it to an existing marker, then coats the stencil and roadway with paint or thermoplastic. The stencil is removed and drying time is given before vehicles are allowed to pass over. Application of thermoplastic material requires a crew of five whereas painting requires two work-

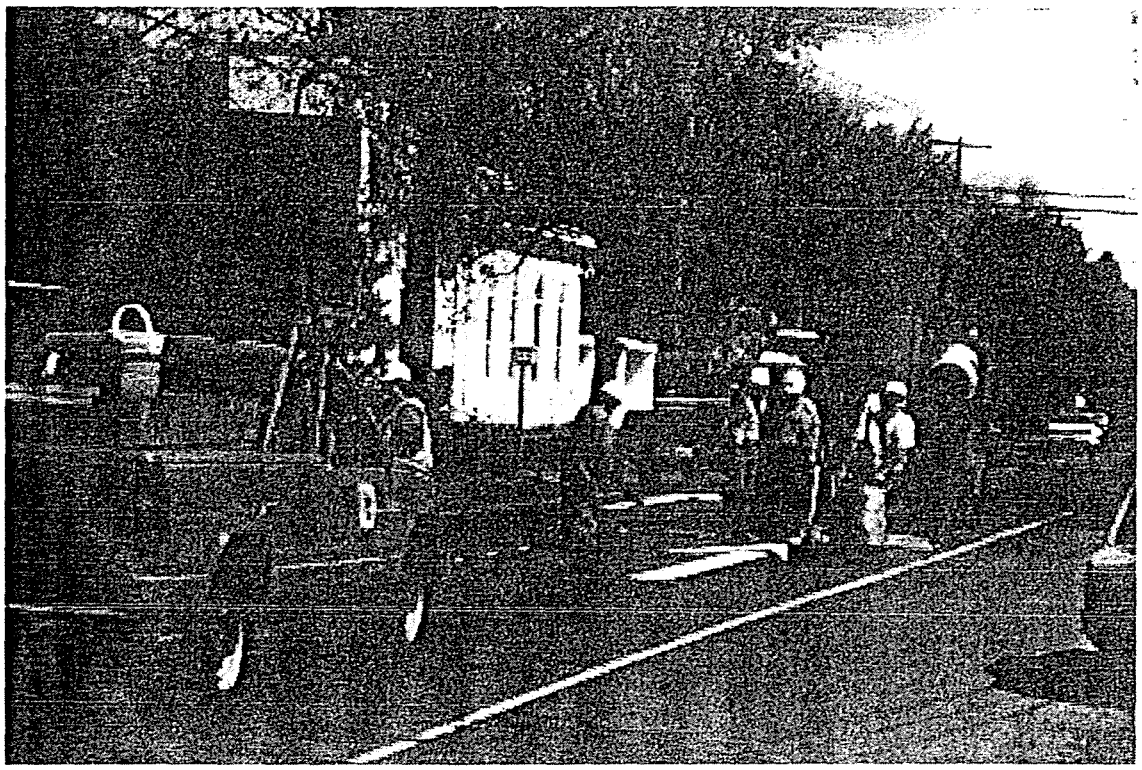

Figure 1.2: An example of producing roadway markings using thermoplastic

ers [6]. Equipment has to be unloaded and operated by personnel exposed to fast moving traffic. An example of producing roadway markings is shown in Figure 1.2. Our goal at AHMCT is to reduce hazards to workers and improve the efficiency of roadway maintenance We can accomplish this by applying robotics and automation to highway maintenance and construction. thereby keeping workers in the safety of their vehicles.

Chapter 2

Trajectory Planning

2.1 Description of Roadway Markings

The roadway markings being painted are specified by "Standard Alphabets for Highway Signs and Pavement Markings" printed by the U.S. Department of Transportation. They consists of all 26 letters of the alphabet. These letters are typically required to be applied at heights which are not shorter than 2.44 Meters (8 feet). This allows a roadway marking to be read clearly from a sharp angle. A standard letter appears elongated in the vertical direction with a ratio of 25 units long to 4 units wide. Fig. 2.1 shows a sample of the roadway markings which consists of four letters on 25 by 4 grids as presented by the U.S. Department of Transportation [2].

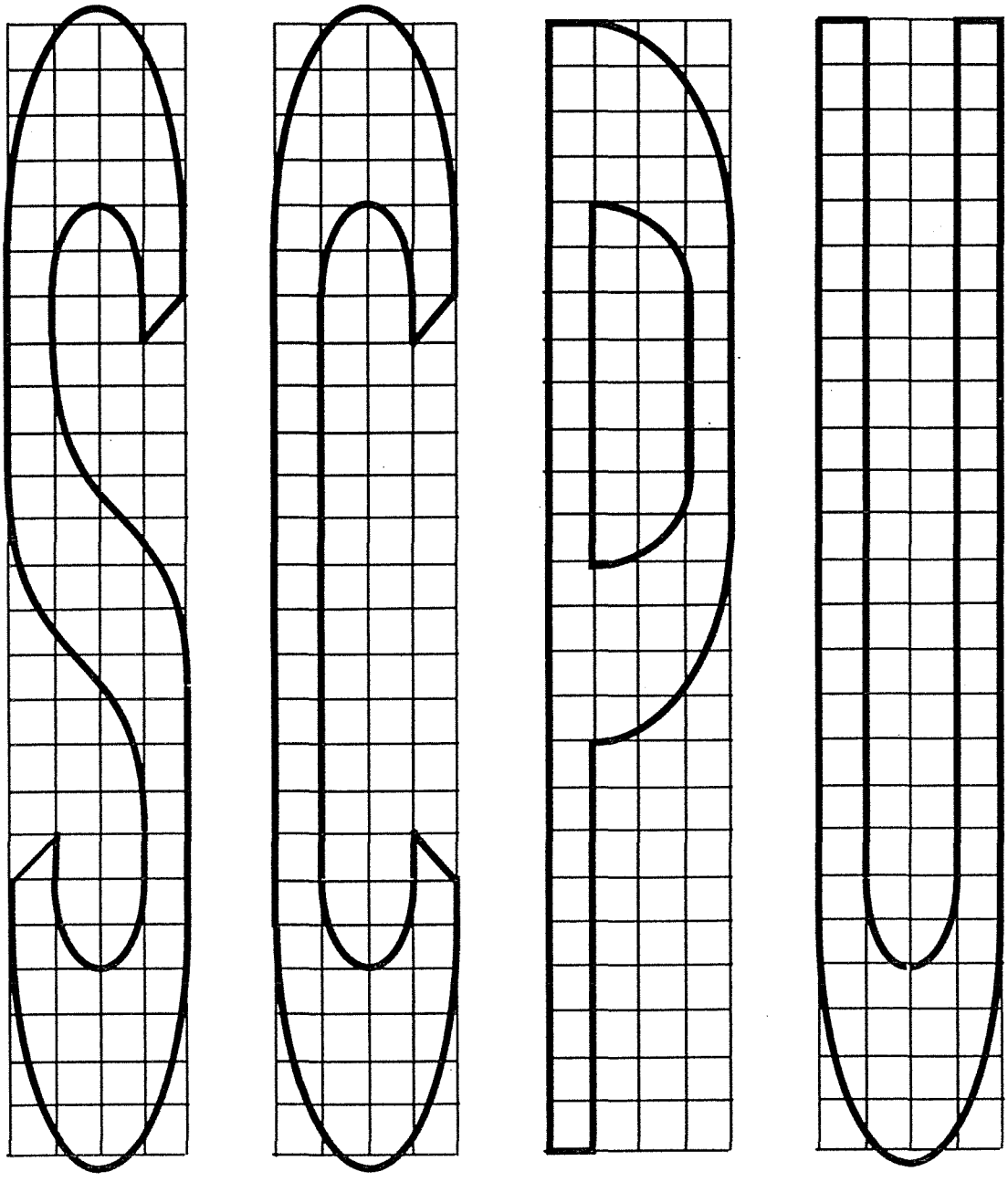

Figure 2.1: A sample of the standard markings

2.2 Trajectory Planning

A standard trajectory planning method was developed and tested using high pressure painting equipment (described later in the testing section). This method is generic in nature and can be applied to many markings including all 26 letters of the alphabet. There are many steps to the actual implementation of this trajectory planning technique, therefore, to simplify the readers understanding, a flow chart is used to highlight the major steps as shown in Figure 2.2. It should be noted that there are some steps that will not be discussed but can be viewed in the source code located in Appendix A.

A typical manipulator allows the user to input cartesian transformations describing the end effector positions and orientations. They typically allow for user input of velocity and accelerations of the end effector. Since this application is intended for use in the Stenciling Robot pantograph-style manipulator (see Figure 2.3), the specific trajectory data has been developed with regard to its limiting-performance characteristics in mind [7]. These characteristics are boundary conditions set forth by the robots structural design and control system.

The Stenciling Robot is simple in nature, the robot should not experience any singularities in the application of the trajectory generation algorithm developed here. Using other robot geometries, one has to pay special attention to make sure trajectories are singularity free.

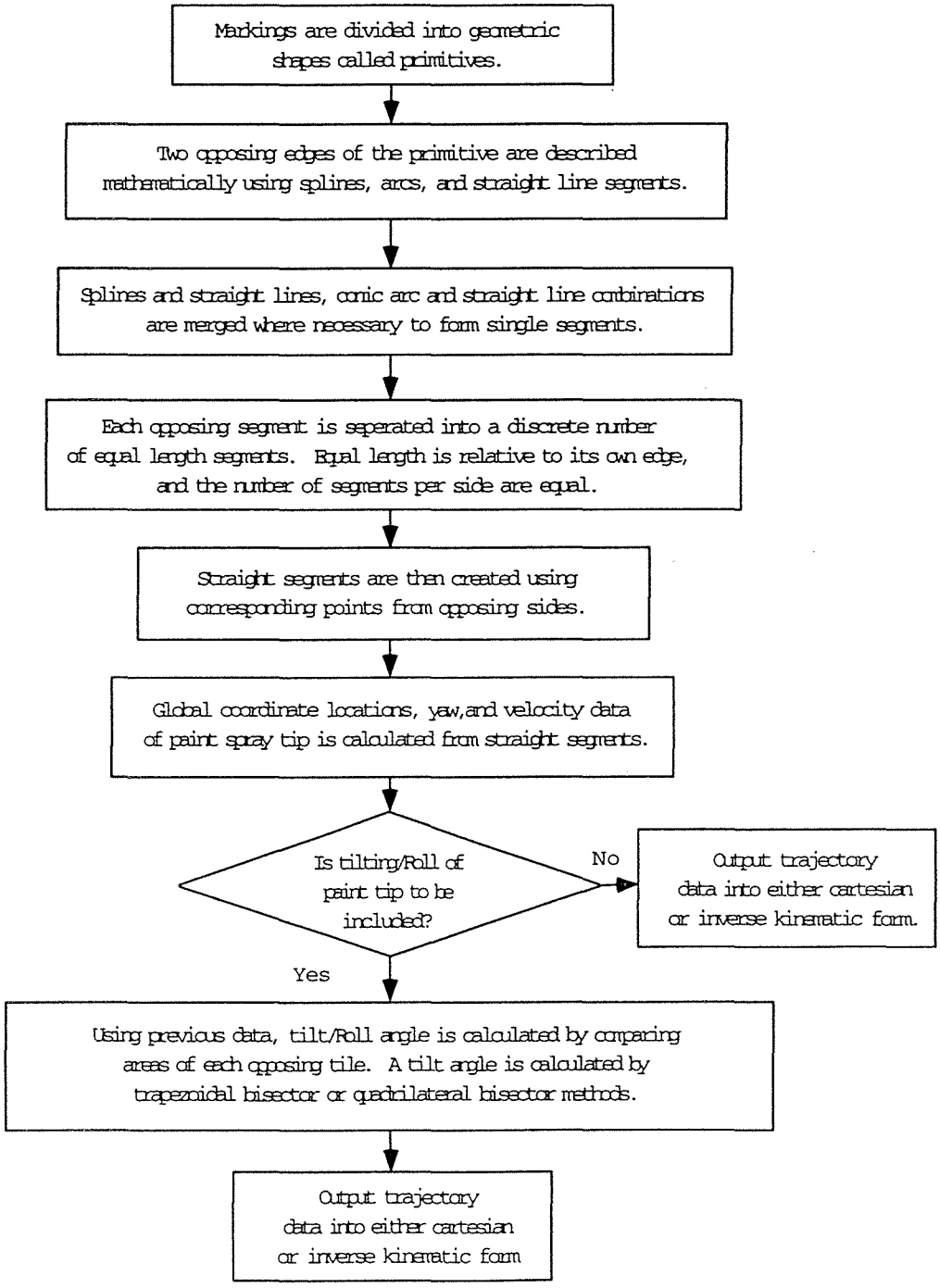

Figure 2.2: Flow chart of the trajectory planning technique

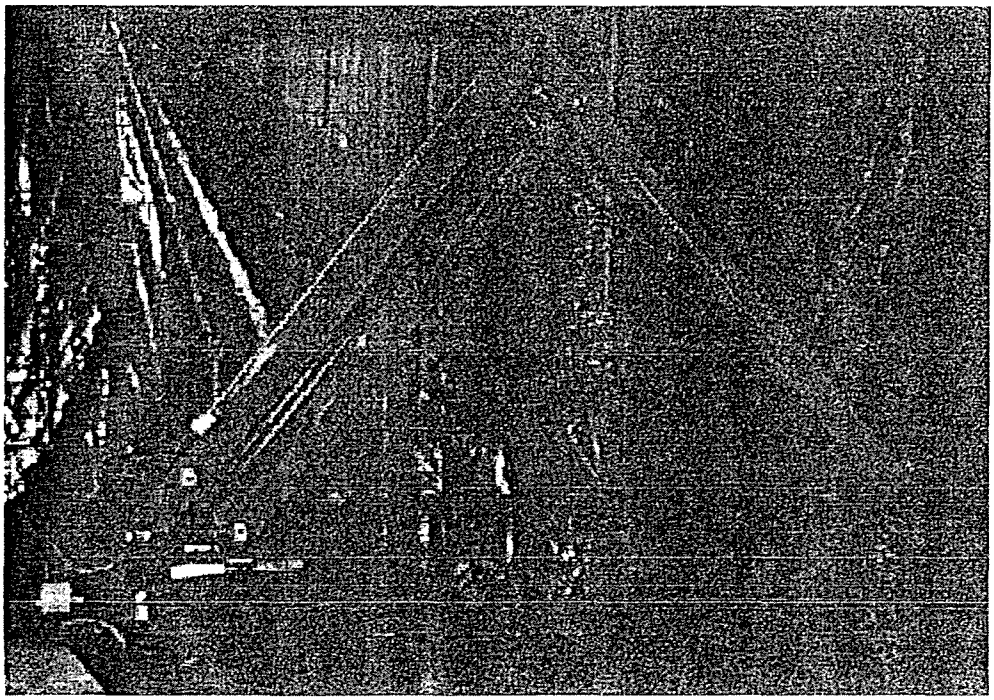

Figure 2.3: The pantograph style UC Davis Stenciling Robot

2.3 Description of Standard Alphanumeric Markings

All of the alphanumeric markings share similar characteristics to each other. All 26 letters can be formed using five standard shapes. Kochekali and Ravani (see [5]) describe these five standard shapes as primitives. Figure 2.4 shows the five primitives used to make all 26 letters. The primitives must be rotated and flipped, as necessary, to form each different letter; but the overall characteristics are the same. Table 2.1 shows the breakdown of each letters use of the five primitives. Many markings consist of all five primitives as shown in Fig. 2.5. The intersections of the interior

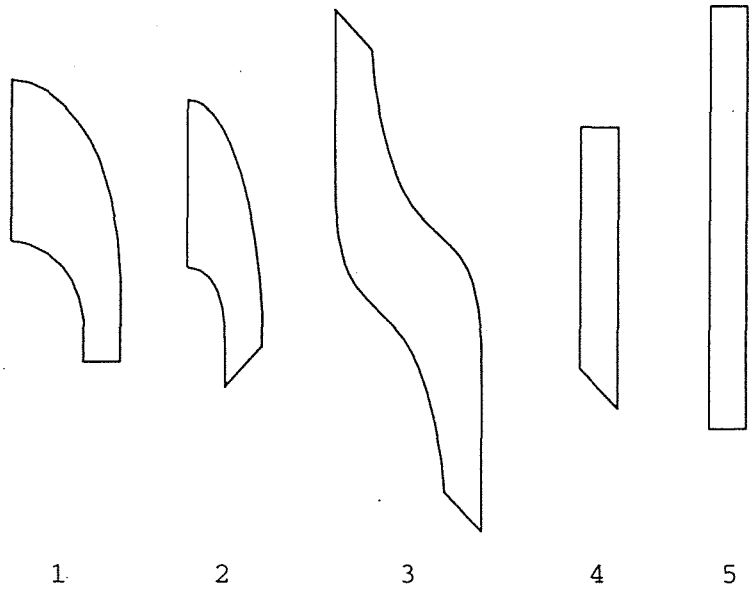

Figure 2.4: Five basic primitives used for all 26 letters

Letter	Primitives	Letter	Primitives	Letter	Primitives
A	4,5	J	2,4,5	S	2,3
B	1,5	K	4,5	T	5
C	2,4	L	5	U	2,4
D	1,5	M	4,5	V	4
E	5	N	4,5	W	4,5
F	5	O	2,4	X	4
G	2,4,5	P	1,5	Y	4,5
H	5	Q	2,4	Z	4,5
I	5	R	1,4,5		

Table 2.1: Usage of primitives for each letter

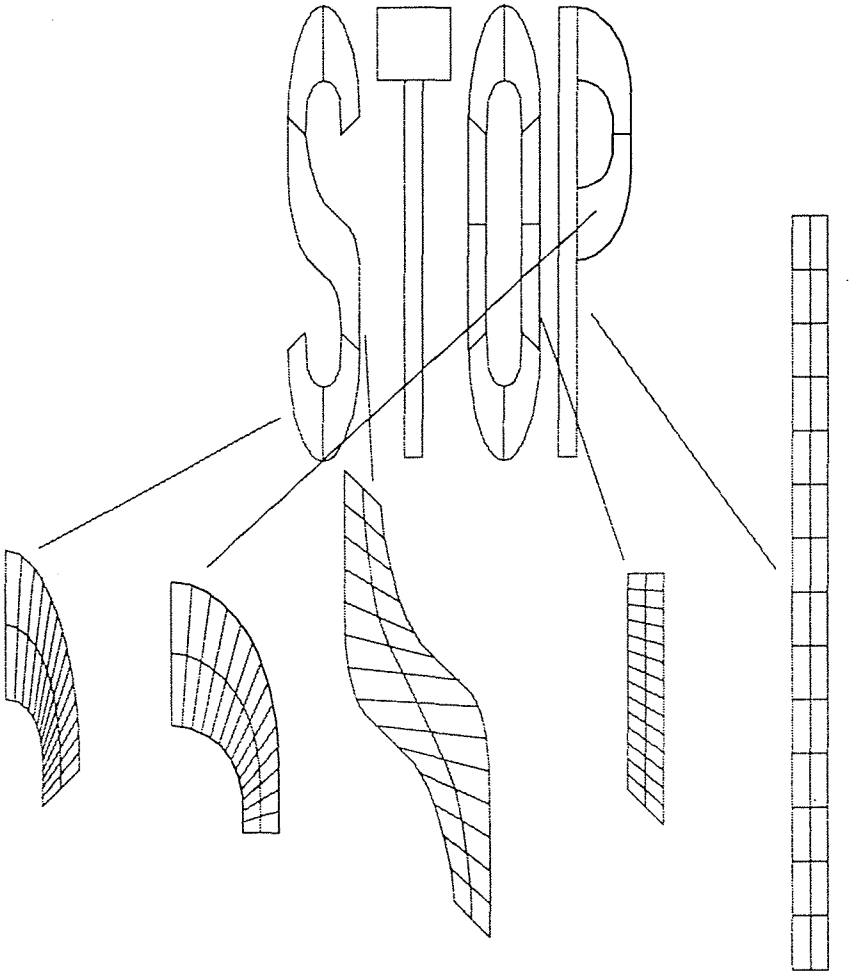

Figure 2.5: Typical symbol using all five primitives

lines of the primitives shown in Fig. 2.5 indicate the location of the center of the spray gun. The number of intersection points will generally be much greater. The method of generating these intersections will be discussed in detail to follow.

2.4 Primitive Trajectory Planning Generation

Each of the primitives edges have been described analytically using splines, arcs and straight-line segments. This unpublished work was completed by a previous researcher, Ian Broverman; for completeness, the overall concept will be described. As shown earlier, each of the letters are presented by the U.S. Department of Transportation [2] on grids of uniform size. Using the grid, the curves could be matched closely with spline, arc, and line segments. The grid gave exact starting and ending locations and good intermediate locations of the curves. Cardinal Splines [8], are used in primivites 1, 2, and 3. Conic Arcs were used in primitives 1 and 2 [3]. Straight lines are used in all five primitives.

The Cardinal spline is a planar curve described using this form

$$V(t) = \boldsymbol{f}^T \boldsymbol{M}_{cs} \boldsymbol{q}_{cs} \tag{2.1}$$

where $\boldsymbol{f}^T$ is the polynomial basis function of the form

$$\boldsymbol{f}^T = [\, t^3 \quad t^2 \quad t \quad 1 \,]\,. \tag{2.2}$$

M_{cs} is the basis matrix of this form

$$M_{cs} = \begin{bmatrix} -a & 2-a & -2+a & a \\ 2a & -3+a & 3-2a & -a \\ -a & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{2.3}$$

and q_{cs} is the control vector representing the four control points. The two opposing edges of primitives 1 and 2 are composed of combinations of spline and straight lines, and conic arc and straight lines. Primitive three has a combination spline and straight line for both opposing edges which are actually rotated 180 degrees apart. The edges need to be continuous; therefore, the combinations were merged together to form a single edge, as shown in figure 2.6. The merging was implemented numerically representing this format.

$$X = \begin{cases} f^T M_{cs} q_{cs} & \text{for } 7 \geq Y > 2 \\ 4 & \text{for } 2 \geq Y \geq 0 \end{cases}$$

For a continuous path through the primitive, a method dividing the two opposing edges into series of equally spaced points was implemented. These points or segments, depending on how one views it, are equally spaced along the edge and also are equal in number per opposing edge. A numerical method using linear interpolation and iteration was used to allow any number of points, ≥ 2, to be spaced equally along the edge, composed of a Cardinal Spline and straight line (see figure 2.7). The number of points determines its resolution of the trajectory. When implementing, one would choose a number of points that can be efficiently processed by the robot controller.

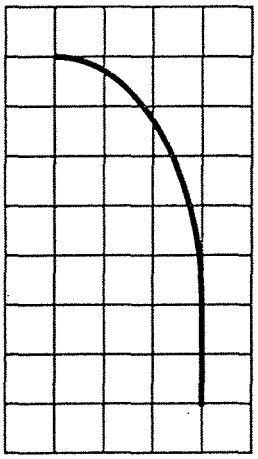

Figure 2.6: Spline and straight line merging

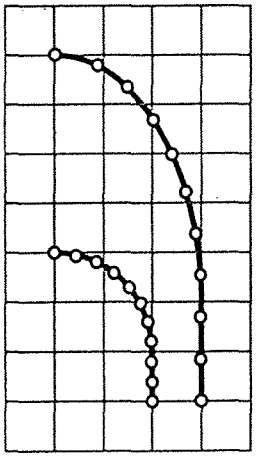

Figure 2.7: Division of edges into equally spaced points

The next step is to join the corresponding opposing points with straight lines. The lines generated represent the spray-nozzle yaw angle ψ as it would traverse its path (see figure 2.8). ψ is calculated by [1]

$$\psi = tan^{-1}\left(\frac{Y_2 - Y_1}{X_2 - X_1}\right). \tag{2.4}$$

The midpoint of each line segment determines the center location of the path during non-tilting of the spray nozzle and can be easily calculated using the equation for a midpoint of a line. The height of the spray gun Z is dependent on the line length and spray-nozzle fan angle α and can be written as

$$Z = \frac{\sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2}\ /2}{\tan(\alpha/2)}. \tag{2.5}$$

The tangential velocity at each knot point is inversely proportional to the length of the line. This is to help distribute the volume of paint evenly. The equation for the tangential velocity V_t is

$$V_t = \frac{K}{\sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2}} \tag{2.6}$$

where K, a constant, is a function of the surface texture, paint viscosity, and desired finish.

This trajectory generation method is done similarly in each primitive. Therefore, its necessary to have some method of numerically describing the edges of each marking. The painting system uses an airless fan spray nozzle. Figure 2.9 shows the pattern of a fan spray nozzle. This type of nozzle pattern is needed for this path

planning method. The geometry begins and ends with straight edges which is simply accomplished with this nozzle pattern.

Paints are abrasive, and tip wear can cause inaccuracies in the output. Therefore, to prevent rounded corners, the manufacturer recommends replacing a tip after 113 to 190 liters (30 to 50 gallons), depending on the abrasiveness of the paint.

2.5 Advanced Trajectory Planning

The previous section discussed the procedure of path planning while holding the spray nozzle in a horizontal orientation. The next section discusses the procedure of path planning when including roll of the spray tip ϕ.

When holding the spray tip horizontally while traversing through a path, the distribution of paint volume in some instances will be uneven. This is due to curvature in the trajectory and the change of yaw angle ψ of the spray tip. This can best be seen in primitives 1 and 2 where the path is composed of a sharp radius. The paint will tend to be heavy near the inner edge and light on the outer edge. In Figure 2.10, the simulation shows the contour of the paint distribution. To calculate this distribution the original path was divided into many small areas which will be called tiles (see Figure 2.11). The area of the individual tile on each side of the divided middle axis is compared to the sum of both of the tiles on both sides of the axis. This comparison shows the difference in volume of paint applied per tile. Two assumptions have been made as follows:

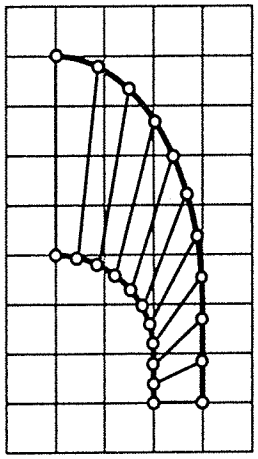

Figure 2.8: Opposing points are joined with straight lines

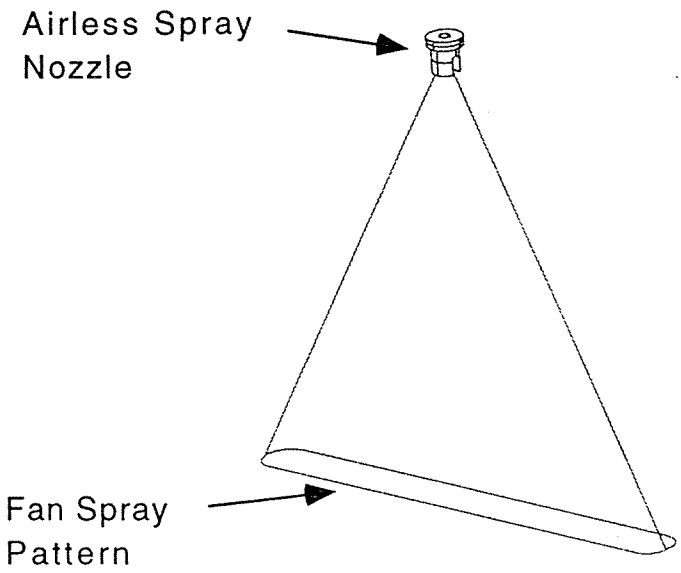

Figure 2.9: Airless Nozzle Spray Pattern

- The distribution of paint dispensed from the spray nozzle is even from edge to edge.
- The velocity is inversely proportional to the line length.

From this we can deduce that on either side of the center line of the spray, the paint volume is equal, but the area being applied may not be equal when rounding an arc.

The area of the tile is calculated by dividing the tile into two triangles and summing them together. The area ratio U_i between the individual tiles and sum of the two tiles A_T can be expressed as

$$U_i = \frac{A_i}{A_T} \quad and \quad U_j = \frac{A_j}{A_T} \tag{2.7}$$

where

$$A_T = A_i + A_j \tag{2.8}$$

$U_{i,j}$, the area ratio, can represent the relative paint distribution of the primitive. A value of 50 is the medium paint volume per unit-surface area. A $U_{i,j}$ value greater than 50 is a smaller paint volume per unit-surface area, since the equivalent amount of paint is spread over a greater surface area. Likewise a $U_{i,j}$ value less than 50 is a heavier distribution of paint. The contour plot shown in figure 2.10 gives a good graphical representation of the distribution of paint. The upper edge of primitive 1 has a value of 58 where the lower edge has a value of 42. This wide distribution range is considered undesirable in most instances. An ideal distribution would have a value 50 throughout, meaning that the paint thickness is at average thickness. To get closer to the ideal distribution, tilting of the paint nozzle will be utilized.

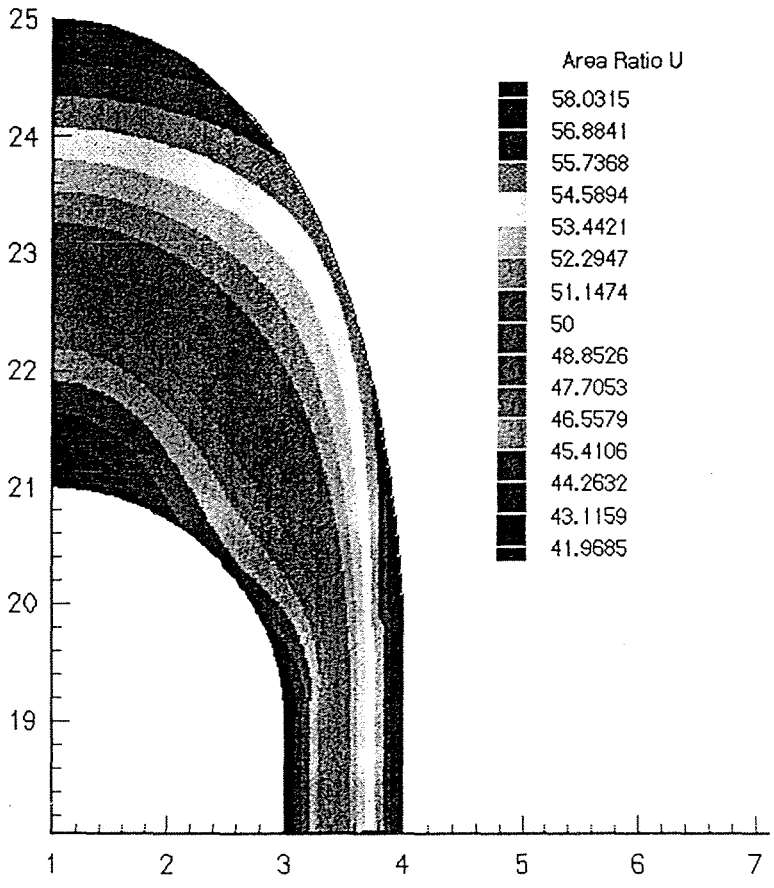

Figure 2.10: Contour Plot of Primitive 1's Paint Distribution

Tilting of the spray nozzle, as previously mentioned, will improve the distribution of paint volume. To illustrate this concept, figure 2.12 shows a tilted paint nozzles spray characteristics. The model represents a fan spray nozzle where the angle of spray α is divided in half. The volume of paint dispersed is equal on both sides of the intersecting line. Therefore, if the spray nozzle is tilted, the volume of paint will differ per side of the intersecting line. To further expand this concept a relationship between tilt angle β, and tile areas A_i and A_j will be derived to tilt the spray nozzle for best results.

To optimize an even distribution of paint, the area of each opposing tile needs to be equal and is presented in this form

$$A_i = A_j. \tag{2.9}$$

As previously mentioned, the tile-dividing line is coincident with the endpoint of the intersecting line of the spray angle. For the opposing tile areas of a non-parallelogram to be equal, the tile dividing line will need to be translated. For this translation to occur, the spray tip is tilted an angle β which can be calculated as

$$\beta = \frac{1}{2}\gamma - \arctan\left[\frac{x_i \cos\gamma - x_j}{x_i \sin\gamma}\right]. \tag{2.10}$$

Two different methods have been developed for calculating a tilt angle β which may satisfy the constraint of equation 2.9. The methods are the trapezoid-bisector method and the quadrilateral-bisector method.

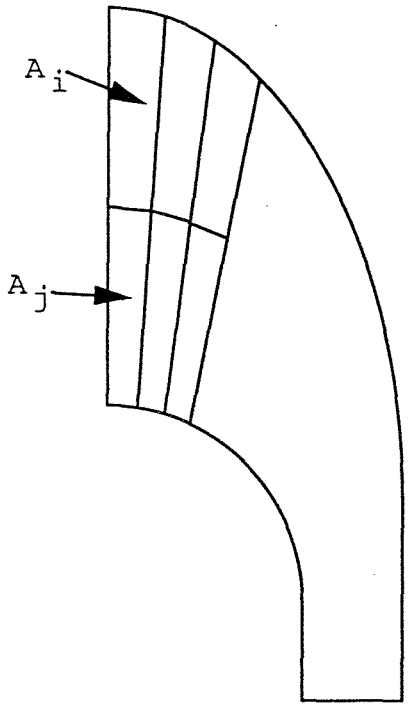

Figure 2.11: Tiles of primitive 1

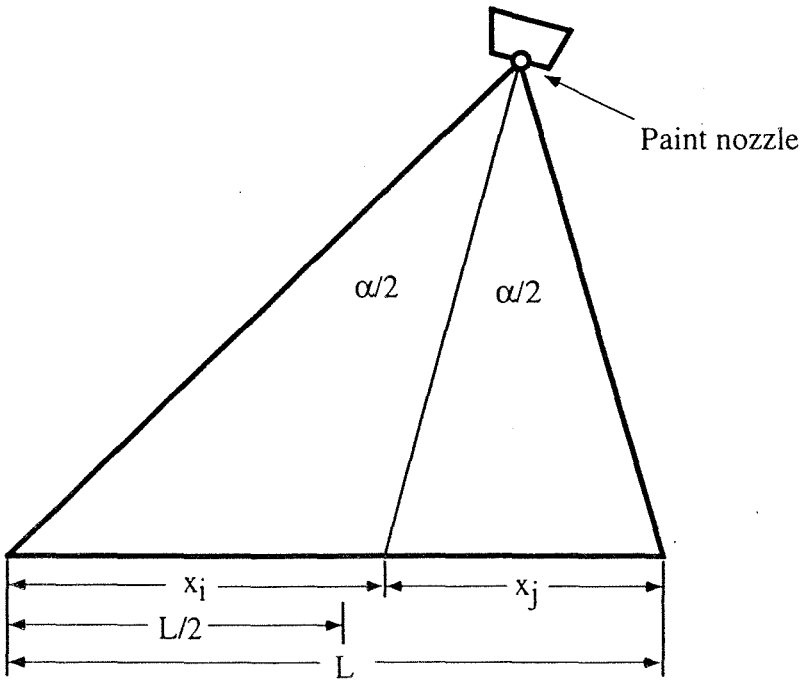

Figure 2.12: Spray characteristics during tilting

2.5.1 Trapezoid-Bisector Method

The trapezoid-bisector method is likely the most simple method to approximate the tilt angle β. This method can be effectively utilized when a constant tilt angle β is sufficient for the trajectory. For example, when painting using primitive 1, the trajectory will always involve a mirror image of that primitive, thus never producing a discontinuity of tilt angles. The trapezoid-bisector method is simply described as: given a trapezoid, find a line parallel to the other two parallel sides that divides the trapezoid into two equal areas. Where

$$x_1 = \begin{bmatrix} \frac{1}{(2(-2b1+2b2))}(-4b1 + 2\sqrt{2}\sqrt{b1^2 + b2^2}) \\ \frac{1}{(2(-2b1+2b2))}(-4b1 - 2\sqrt{2}\sqrt{b1^2 + b2^2}) \end{bmatrix} \tag{2.11}$$

and

$$x_2 = 1 - x_1. \tag{2.12}$$

Figure 2.14 shows the results using the trapezoid-bisector method. As the figures shows, the area ratios U_i and U_j now range from 47 to 52, a much better paint distribution than with no tilt. The trapezoid-bisector method works very well when the cell being divided has two edges that are parallel or almost parallel. As the edges get farther from being parallel the method is less valid. The distribution of paint is excellent during the beginning and ending, but deviates slightly during the middle section. This can be attributed to the large change in line lengths from cell to cell, thus causing non-parallel edges.

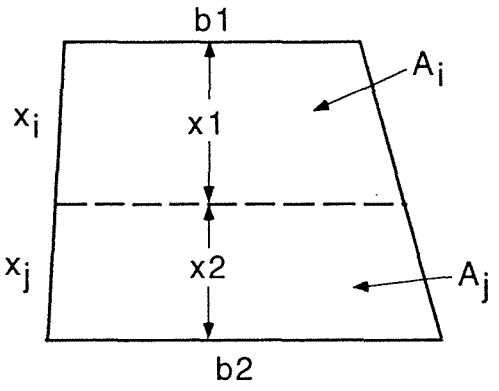

Figure 2.13: Trapezoid-Bisector method

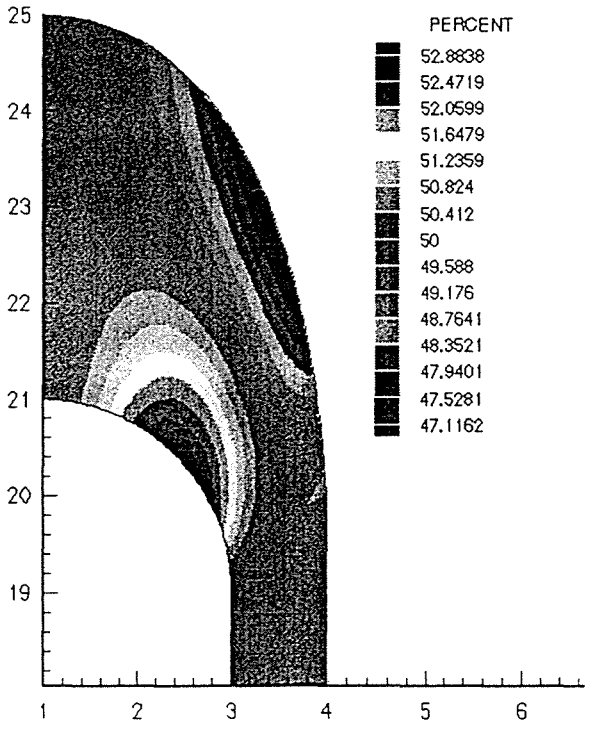

Figure 2.14: Contour plot of primitive 1's paint distribution with a tilt angle calculated using the trapezoid-bisector method.

2.5.2 Quadrilateral Bisector Method

As previously discussed, the cell of interest is not necessarily a trapezoid and can typically only be considered a four sided polygon. The second method is called the Quadrilateral Bisector method because it refers to dividing a four sided polygon with no parallel edges into two equal tiles. This method will give a much better approximation for the tilt angle β, thus providing a more even distribution of paint. Figure 2.15 shows a general cell that may be encountered while generating a trajectory. The Quadrilateral Bisector method is composed of four equations to calculate the two unknown cartesian points (g_x, g_y) and (h_x, h_y). The constraint equations are

$$A_i = A_j \tag{2.13}$$

$$\frac{l_{bh}}{l_{hc}} = \frac{l_{ag}}{l_{gd}} \tag{2.14}$$

$$h_x, h_y = f(\overline{bc}) \tag{2.15}$$

$$g_x, g_y = f(\overline{ad}). \tag{2.16}$$

In order to calculate A_i and A_j, the tiles are split into triangles as shown in figure 2.16. The area of a triangle $\triangle$ in a plane having vertices $A(a_x, a_y)$, $B(b_x, b_y)$, and $H(h_x, h_y)$ is

$$area\ \triangle = \pm \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ h_x & h_y & 1 \end{vmatrix}. \tag{2.17}$$

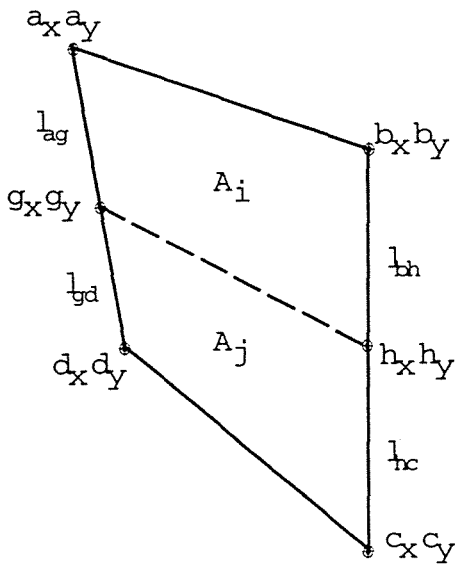

Figure 2.15: General cell division using Quadrilateral Bisector method

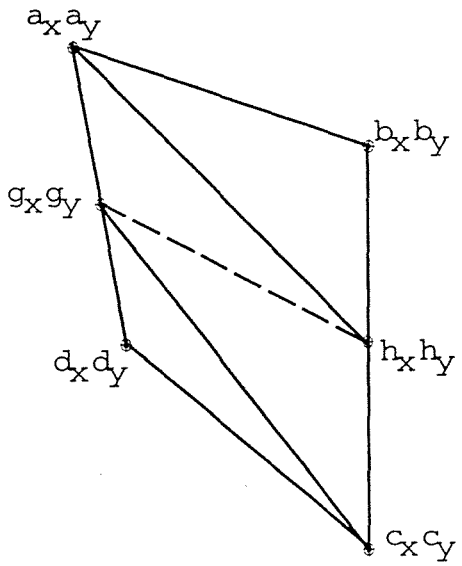

Figure 2.16: Splitting cells into triangles for area calculation

Equation 2.14 states that the points g and h must be proportionally spaced between the corresponding vertices. Equation 2.14 can be expanded in this form

$$\frac{\sqrt{(c_x - h_x)^2 + (c_y - h_y)^2}}{\sqrt{(b_x - h_x)^2 + (b_y - h_y)^2}} = \frac{\sqrt{(d_x - g_x)^2 + (d_y - g_y)^2}}{\sqrt{(a_x - g_x)^2 + (a_y - g_y)^2}}. \tag{2.18}$$

Equations 2.15 and 2.16 specify that the points g and h must lie on the line segments determined by the two sets of points (a,d) and (b,c), respectively. These equations can be expanded in this form

$$g_y - d_y = \frac{a_y - d_y}{a_x - d_x}(g_x - d_x) \tag{2.19}$$

$$h_y - c_y = \frac{b_y - c_y}{b_x - c_x}(h_x - c_x) \tag{2.20}$$

where (g_x, g_y) and (h_x, h_y) are unknowns. The equations can be reduced to two second order equations and two unknowns. The solution will converge quickly with a few iterations using a computer numerical solver.

2.5.3 Trajectory Recalculations Due to Tilting

If tilting of the spray nozzle is utilized, then a new set of equations for positioning of the end-effector in the Cartesian coordinate frame is necessary. Figure 2.17 shows the general configuration of the spray-nozzle characteristics in relation to the desired output. Following are the equations required for calculating new values of X, Y, and Z:

$$x_o = -\frac{1}{2}L\frac{sin(-90 + 1/2\alpha + \beta)sin(-\beta + 1/2\alpha) - sin(\beta + 1/2\alpha)sin(-90 + 1/2\alpha - \beta)}{sin(-90 + 1/2\alpha + \beta)sin(-\beta + 1/2\alpha) + sin(\beta + 1/2\alpha)sin(-90 + 1/2\alpha - \beta)} \tag{2.21}$$

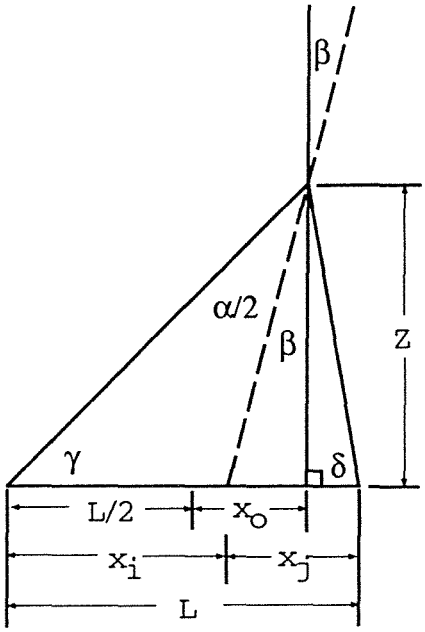

Figure 2.17: General configuration of a spray nozzle tilted

where

$$X_{new} = X + x_o cos\psi \tag{2.22}$$

$$Y_{new} = Y + x_o sin\psi \tag{2.23}$$

$$Z_{new} = \frac{sin(90 - \alpha/2 - \beta)(L/2 + x_o)}{sin(\alpha/2 + \beta)}. \tag{2.24}$$

2.6 Joint Level Trajectory Planning

Most industrial manipulator controllers allow the input of trajectories in Cartesian space, and so far, that is all that has been discussed. For the Stencil Robot, the

trajectory will be input at joint level. To transform the trajectory from Cartesian space to joint space, inverse kinematics will need to be performed. The kinematics for the first two joints of the Stencil Robot is simple in nature and is derived from the relationship of rectangular coordinates to polar coordinates described by these equations

$$X = Rcos\theta$$
$$Y = Rsin\theta \tag{2.25}$$

where R is the extension length of the arm and θ is the rotation angle of the arm with respect the X-Y plane. The robot structure is configured as a pantograph mechanism, where the extension length R is amplified $8.33\overline{3}$ times the actuator movement. The inverse kinematics is of this form

$$r = \frac{\sqrt{X^2 + Y^2}}{8.33\overline{3}}$$
$$\theta = \tan^{-1}\left(\frac{Y}{X}\right) \tag{2.26}$$

where r is the linear actuator position. A complete trajectory for a letter S, shown in figure 2.18, will be used to illustrate the inverse kinematics for a single joint of the manipulator. A plot of joint 1 versus time, is presented in figure 2.19, with the corresponding velocity and acceleration shown in figures 2.20 and 2.21, respectively.

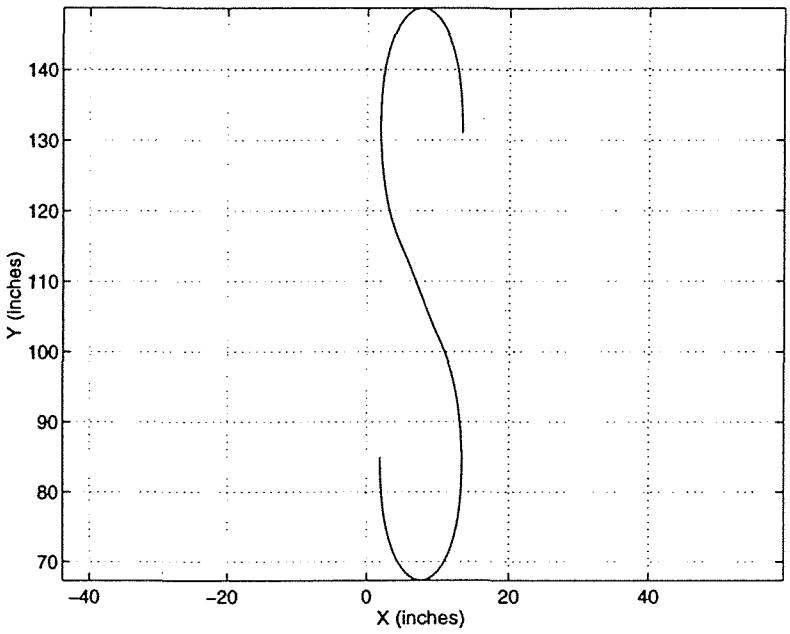

Figure 2.18: Trajectory in X,Y coordinates

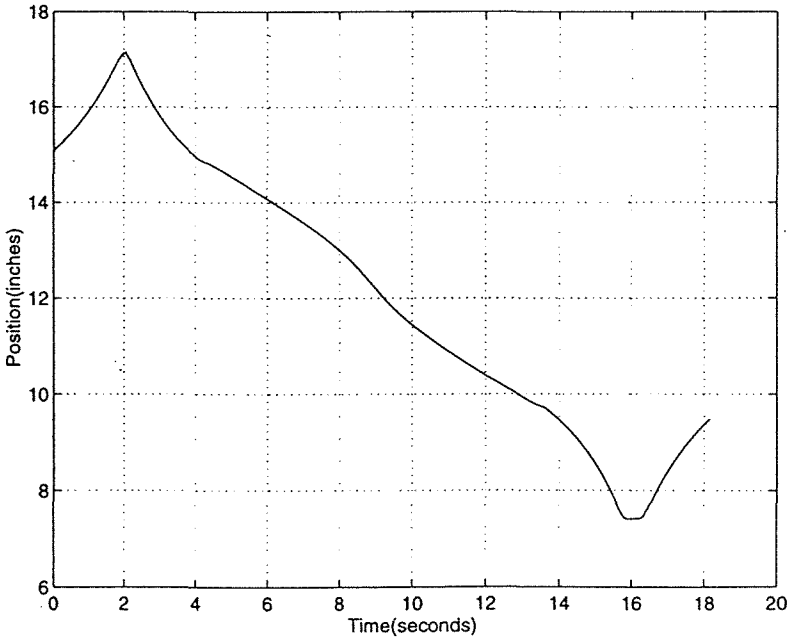

Figure 2.19: Single-joint trajectory corresponding to fig. 2.18

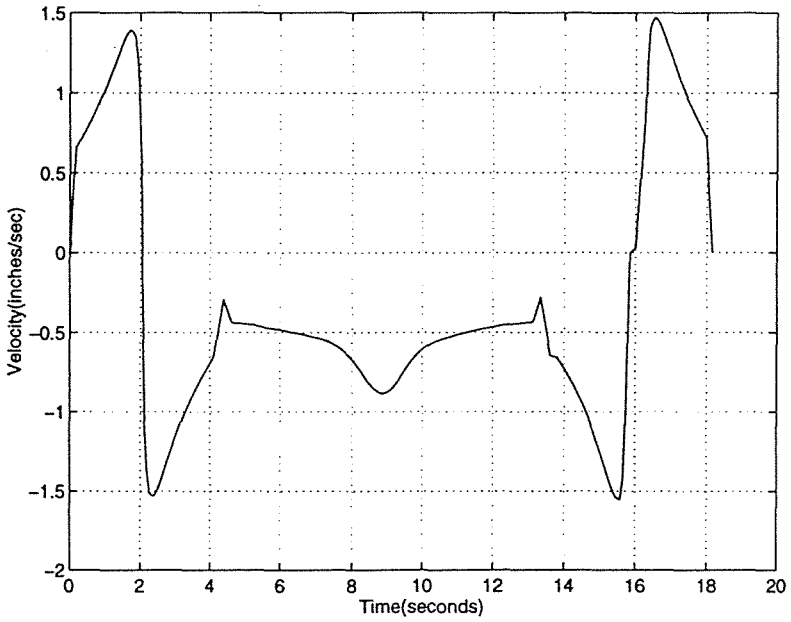

Figure 2.20: Velocity corresponding to trajectory of fig. 2.19

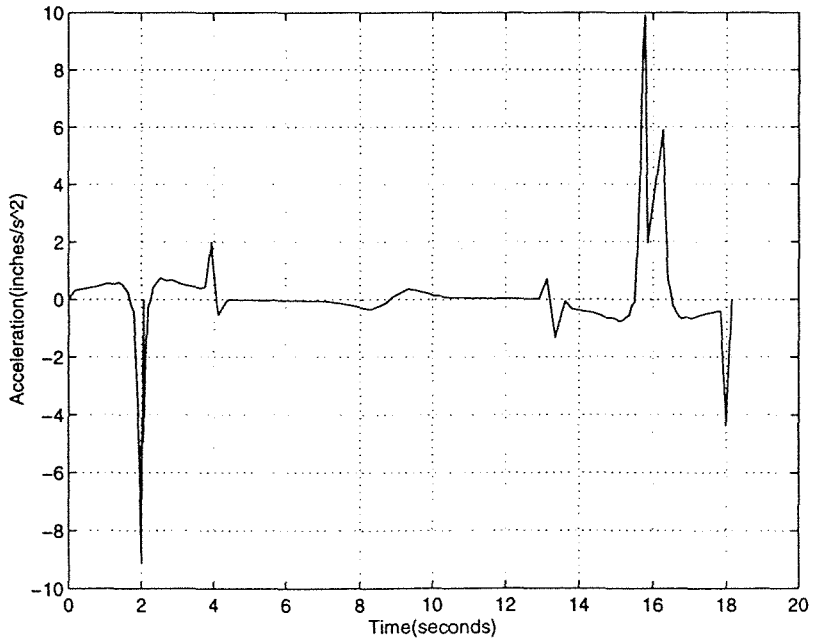

Figure 2.21: Acceleration corresponding to trajectory of fig. 2.19

2.7 Testing of the Trajectory Planning Technique

This section discusses the testing of the trajectory planning technique as described in previous sections. The first subsection will describe the experimental setup, and the second subsection will discuss the results of countless hours of spray painting.

2.7.1 Experimental Setup

To verify the trajectory planning technique, an industrial robot was fitted with an end-effector similar to one that will be utilized on the Stenciling Robot. The industrial robot, an Adept Three SCARA manipulator, was utilized to help develop the path algorithms in parallel to the development of the Stenciling Robot. This would reduce the research and development time for the overall completion of the project.

The Adept Three manipulator comes standard with four degrees of freedom. To simulate the Stenciling Robot, a fifth DOF had to be built and interfaced to the robot controller. This fifth DOF, better known as the tilting axis or Roll ϕ, was integrated into the end-effector. A schematic of the robot and end-effector is shown in Figure 2.22. The end-effector was designed to accommodate a high-pressure airless paint system. The spray gun was positioned to utilize the full length of travel of the Z joint without possibility of crashing into the floor. The tilting axis has been positioned so the center of rotation is coincident with the focal point of the paint nozzle. This will reduce the trajectory offset in the X-Y plane caused by tilting. This

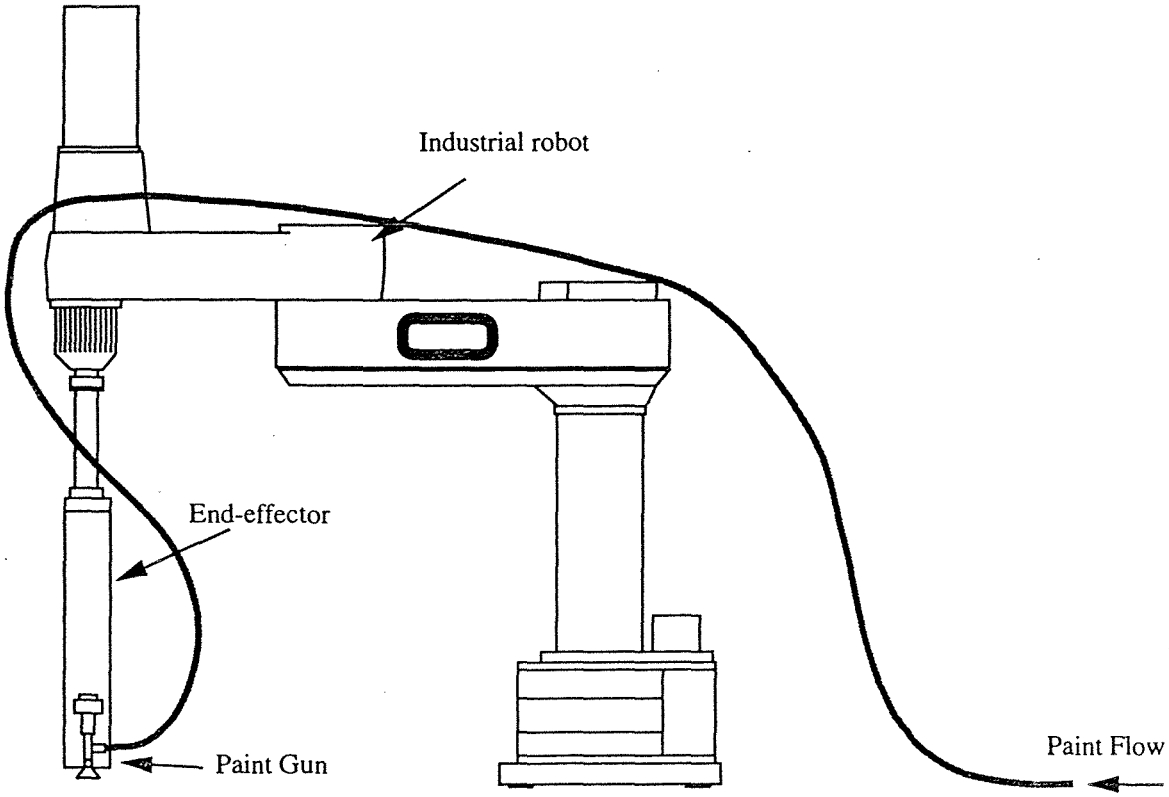

Figure 2.22: Adept robot with a spray painting end-effector

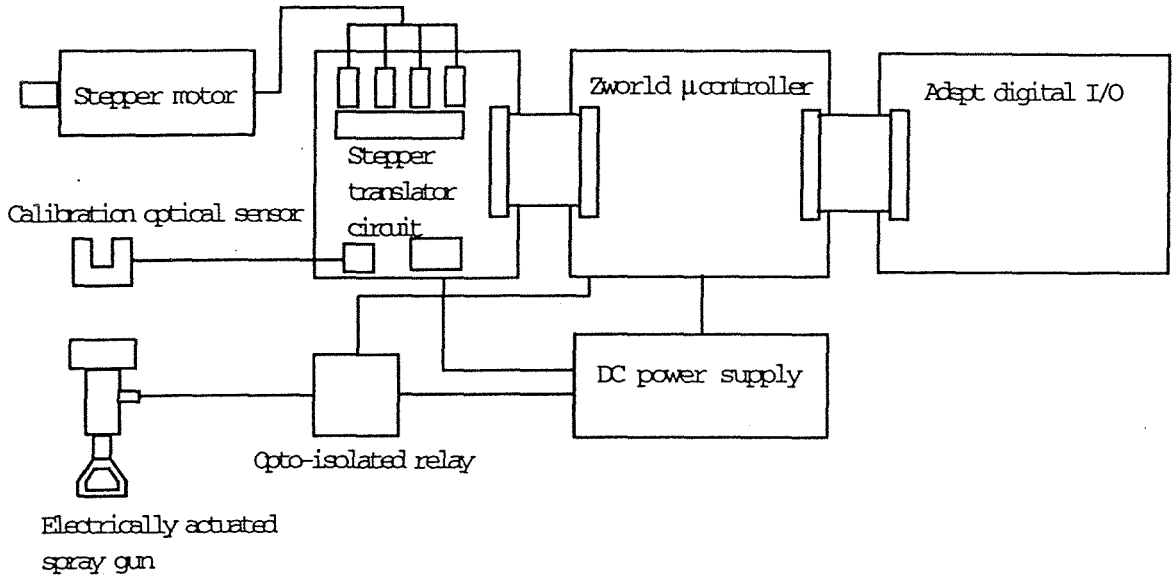

Figure 2.23: Electrical schematic for the Adept end-effector

is the recommended position of the tilting axis for the Stenciling Robot end-effector.

The tilting axis is controlled by a stepper motor attached to the base of the end-effector. The motor is coupled to the tilt axis through a timing belt configuration consisting of a 5:1 gear reduction for increased torque. The tilting axis has a resolution of 0.2 degrees with a range of +/- 25.4 degrees. The stepper motor controller was custom designed, built, and interfaced to the digital I/O port of the Adept controller. The schematic for this controller is shown in figure 2.23.

A Zworld engineering microcontroller is used as the interface between the Adept controller and stepper-motor translator. To control the angle of tilt, the Adept controller outputs an eight-bit number to the Zworld. The Zworld interprets the eight-bit number in a 2's compliment format as shown in Table 2.2 [4]. The Zworld outputs a stepping pulse and a direction to the stepper-motor translator which handles the

binary	*decimal*	*angle* (deg)
00000000	0	0
00000001	1	.2
01111111	127	25.4
10000000	128	-.2
11111111	255	-25.4

Table 2.2: Control signals from Adept to Zworld

sequencing of the phases of the motor. During start up, an optical sensor is used to calibrate the stepper motor. The process is to rotate the spray gun past the sensor, which is positioned at the zero-degree mark, and trigger a pulse to the Zworld to indicate the zero position. Since the actuator is a stepper motor, there is no feedback necessary to accurately control the position. This simplifies the system and is fine so long as the disturbances are not large enough to make the motor miss a step.

The paint gun is controlled through the digital I/O of the Adept controller also. A solid-state, optically-isolated relay is used to trigger the 24 volts needed to turn the paint gun on.

The Adept robot has a limited workspace for spray painting. For this reason a full-sized eight foot tall letter would be impossible to paint using this current system. However, it does provide a significant means of testing the trajectory method. Figure 2.24 shows the workspace available and a typical placement of a spray-painted symbol.

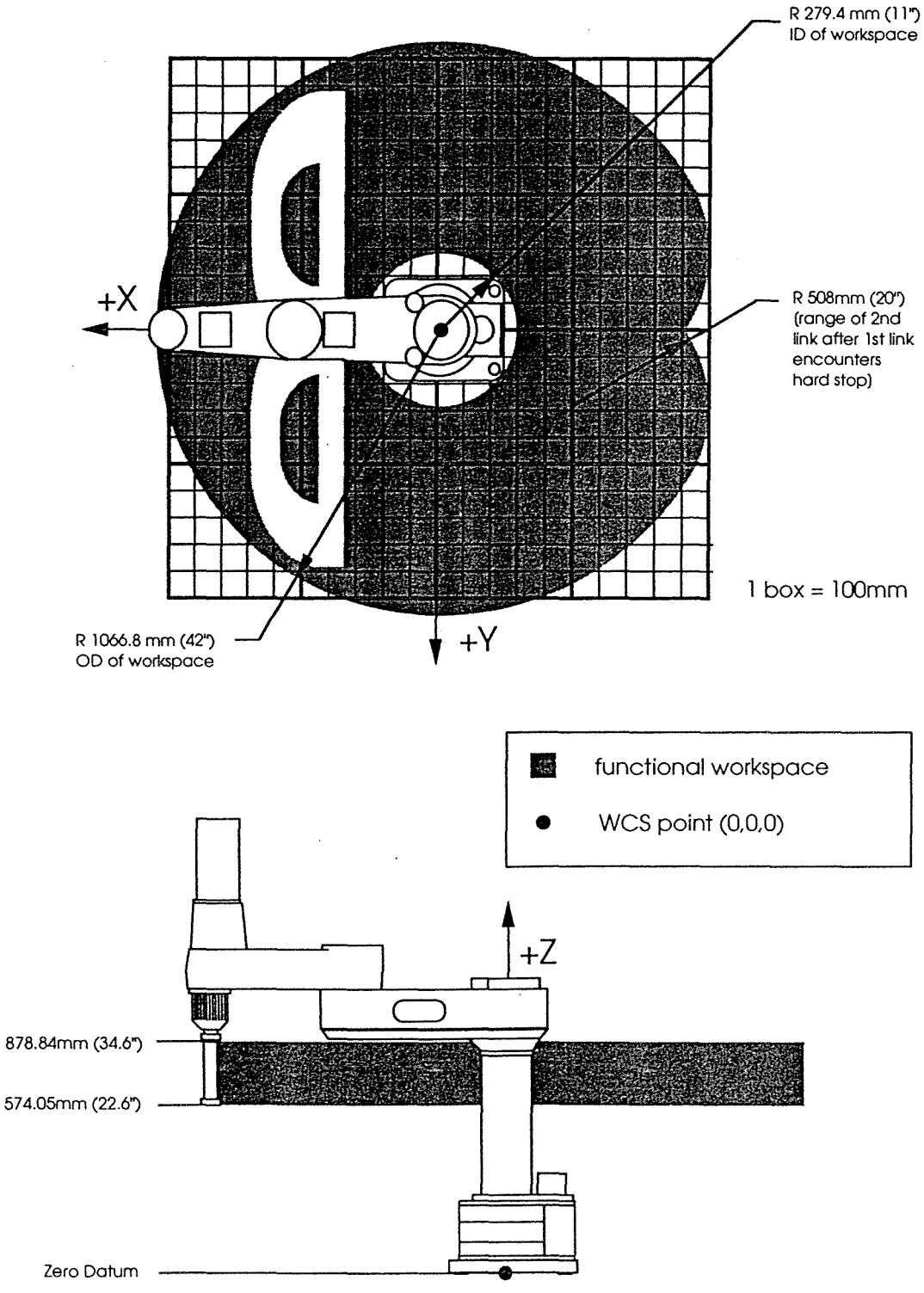

Figure 2.24: Adept workspace

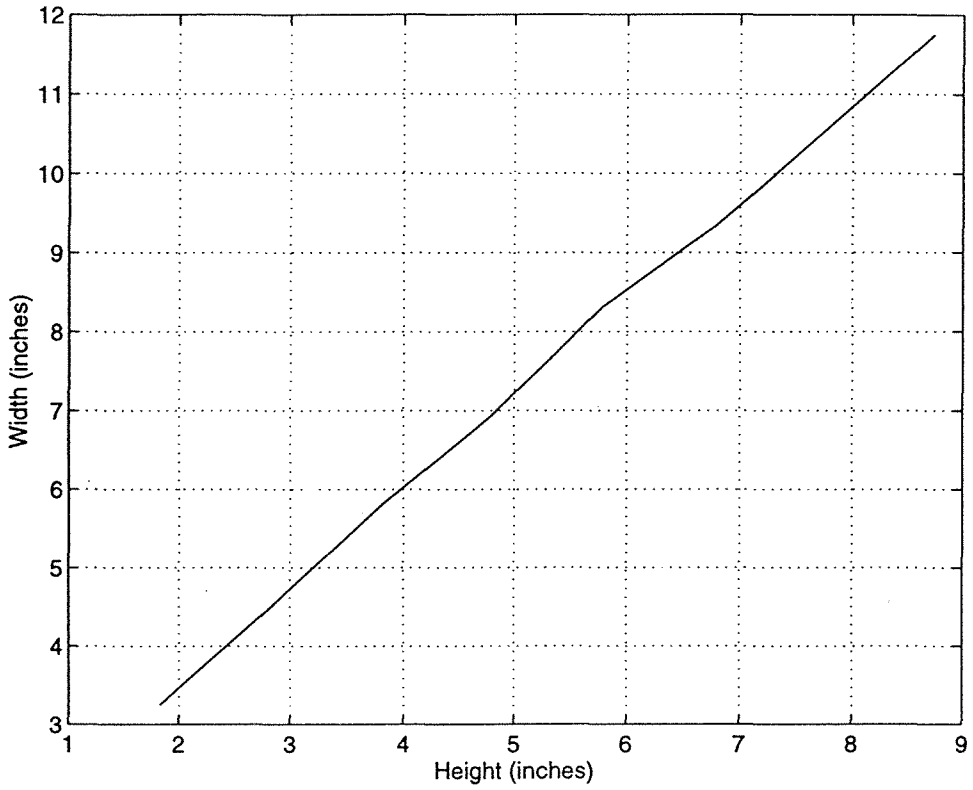

Figure 2.25: Results of testing the height to width relationship of an airless spray nozzle

2.7.2 Results of the Robotic Spray Painting

Testing was performed to evaluate the trajectory planning techniques. The first test was to evaluate the performance of the fan spray. This consisted of spraying a group of straight segments at various spray heights to determine the relation of height to width. The results reveal a linear relationship between the height of the spray nozzle to the spray width as shown in figure 2.25. The data was also used to adjust the height equation in the trajectory planning stage.

Subsequent tests were performed and the results were promising. Two main areas of interest of the tests were edge definition and paint distribution. Figure 2.26 shows

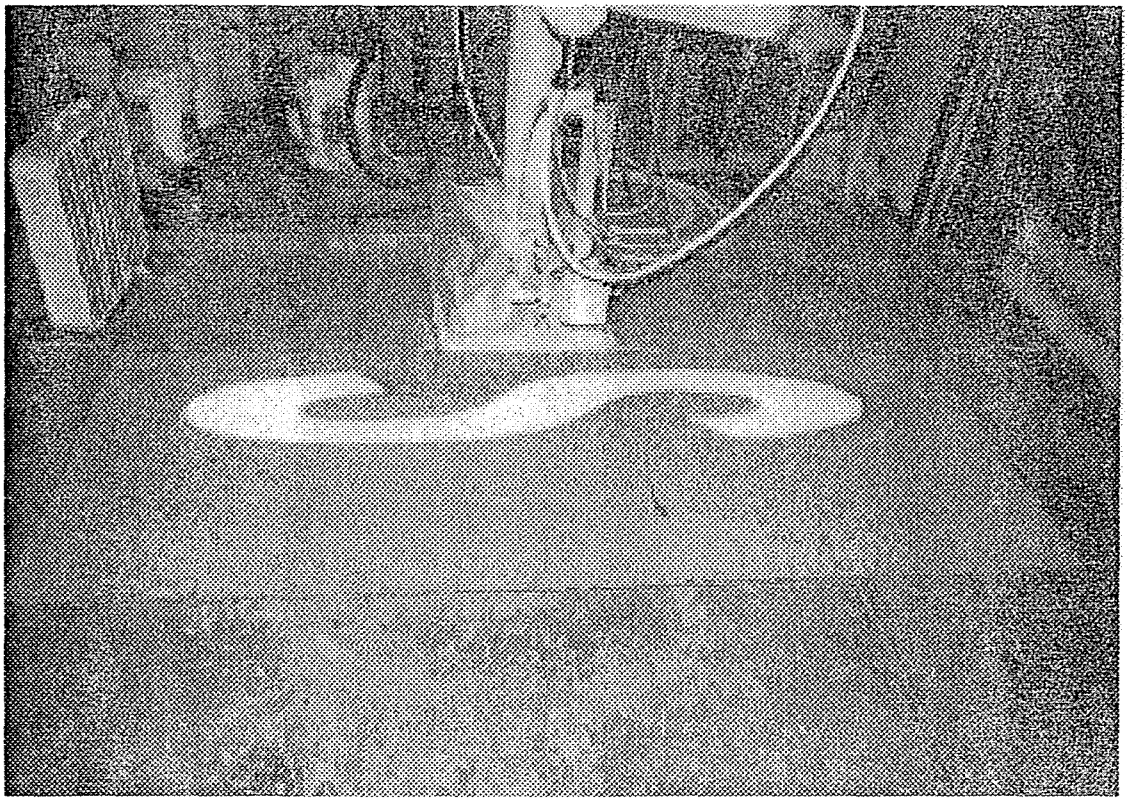

Figure 2.26: Results of testing the trajectory for the letter S.

the result of painting a complete trajectory for the letter S. The trajectory is the combination of primitives 2 and 3. This particular run was with a zero-degree tilt. The edge definition is excellent with no signs of excessive overspray. The paint distribution is good, but shows signs of thinning near the upper and lower portions of the letter. The letter P was tested with a 15° tilt angle during the arc segments as shown in figure 2.27. An area of interest in this test was the trajectory offset caused by tilting. As the photo shows, the curved section lines up well with the straight section at the top of the letter. The paint distribution is excellent in this test, with no detectable areas of thinning or flooding.

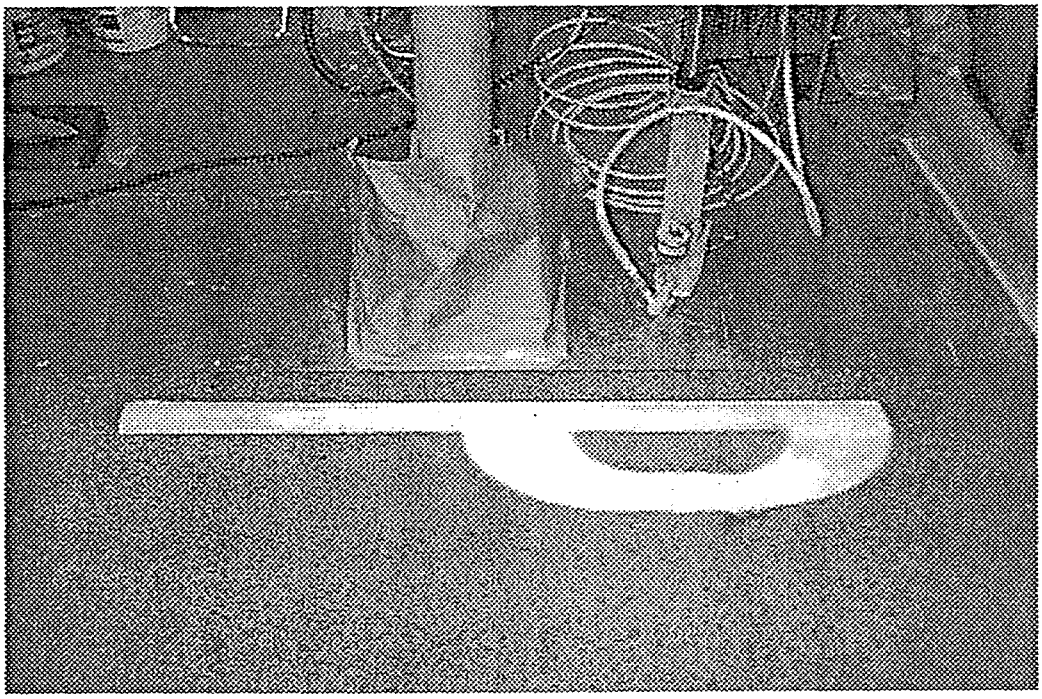

Figure 2.27: Results of testing the trajectory for the letter P with tilt.

Chapter 3

A Mechanical Redesign of the Stencil Robot Parallel Linkage

A key component to the success of the Stenciling Project is the robot. To produce well painted symbols, the robot needs to be accurate and repeatable. A weak link in the overall system can cause poor results. To address this we looked at many aspects of the robot structure and discovered a weak link in the parallel mechanism and retraction system. A redesign of this system was implemented to provide a higher stiffness to the robot end-effector and an increased accuracy to the painting system. This chapter will examine the problems with the previous design and will discuss the solutions of the replacement system.

3.1 Previous Parallel Linkage/Retraction System Design

There are two purposes for the Parallel Linkage/Retraction System: to orient the end-effector perpendicular to the ground at any location, and to retract the end-effector for stowage. The retraction system is designed as a component of the parallel linkage system.

3.1.1 The Parallel Linkage System

The use of a pantograph mechanism for the robot structure produces a large planar workspace with a simple actuation method. Both rotation and extension actuators are located at the base to reduce inertia and gravity effects. Since the specific use of the robot is for spray painting of roadway markings, the end effector need only be oriented 90 degrees to the planar surface for all applications. To simplify this operation, a mechanism was designed to maintain end-effector orientation without the use of any actuation. This in effect, can eliminate one controller needed for end-effector orientation.

The first implemented design of the parallel mechanism is shown in Figure 3.1. It consists of two long composite links that run parallel above the robot structure and three shorter links that run vertically; together, they create two parallelograms [7]. Since opposing edges of a parallelogram always remain parallel, the end-effector ori-

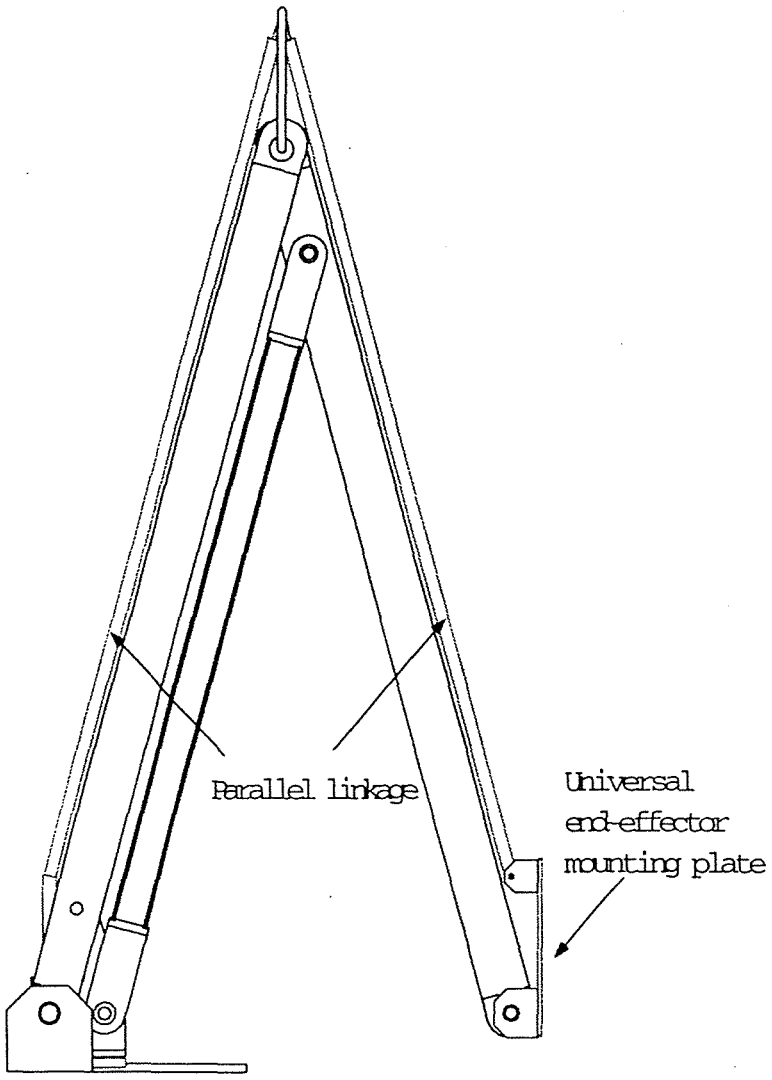

Figure 3.1: First implemented design of the parallel mechanism

entation will always match the orientation of the stationary link. A shortcoming of this design is the height addition to the robot of .45 meters (18 inches). This addition would put the total height of the robot in a retracted position at 4.11 meters (13.5 feet). A second shortcoming was a requirement that was placed after the design was implemented; it consisted of retracting the end-effector for stowage and travel.

To try and solve these problems, a second mechanism was designed and implemented. This design was similar to the first with the same parallelogram-style linkage, but now the linkage was located to the sides of the robot as shown in Figure 3.2. Integrated into this design was the retraction device which consisted of pneumatic cylinders inserted into the composite tubes. These cylinders would collapse and extend for retraction of the end-effector during stowage. This system now used four long composite links for symmetrical loading instead of two in the prior design. This design was approximately three times heavier and required three times the required parts than the prior design.

The downside to this design was its stiffness. For spray painting of roadway symbols, it is important that the end-effector remain oriented vertically. Table 3.1 and figure 3.3 show the deflection of the end-effector joint (joint F as shown in Figure 3.2) under a constant torque throughout the robot's full range of extension. As both the table and figure depict, the deflection of the end-effector (joint F) is quite large during minimal extension distances. The majority of the deflection can be attributed to the distortion of the idler links at joint D (see Figure 3.2). The cam followers used to

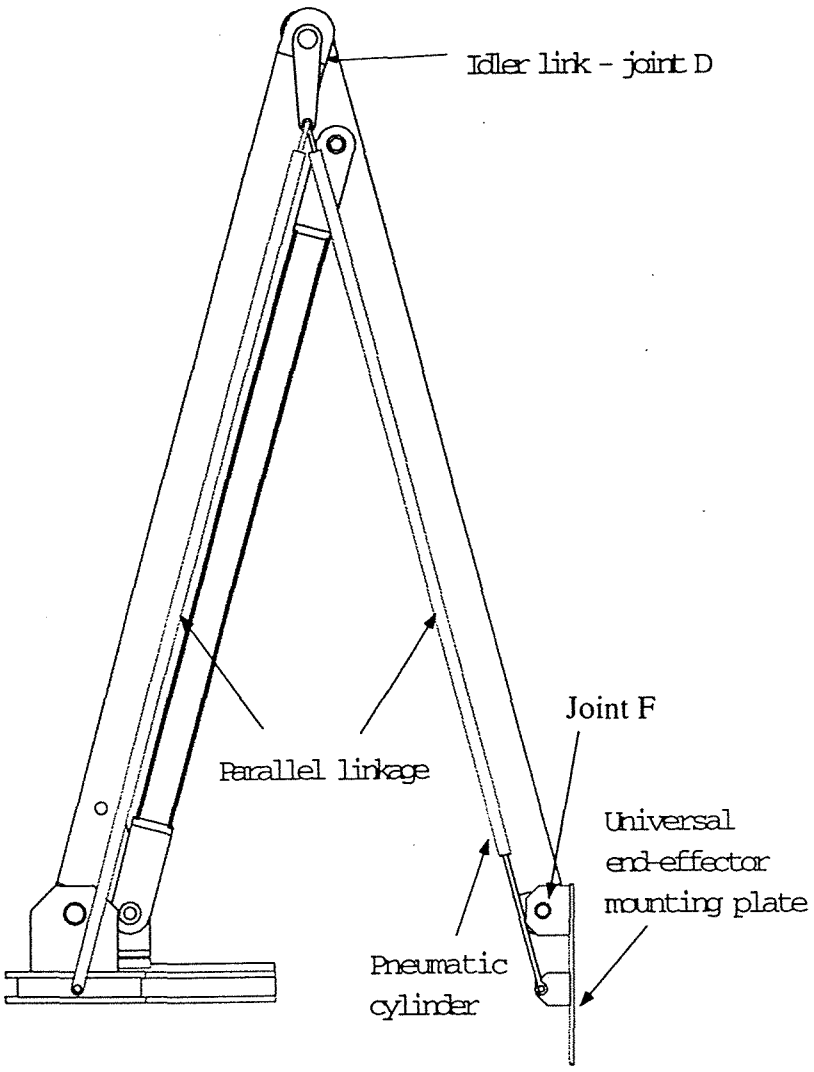

Figure 3.2: Second implemented design of the parallel mechanism

Robot link extension (mm)	Angular deflection of of joint F (degrees)	Deflection at nozzle height (mm)
1470	3.6	89
1765	2.3	56
2057	1.7	41
2362	1.4	33
2654	1.0	22
2959	0.8	20
3264	0.6	15
3567	0.5	13
3874	0.4	10
4178	0.3	8
4483	0.3	8
4788	0.2	5
5093	0.2	5

Table 3.1: Deflection of joint F under constant torque

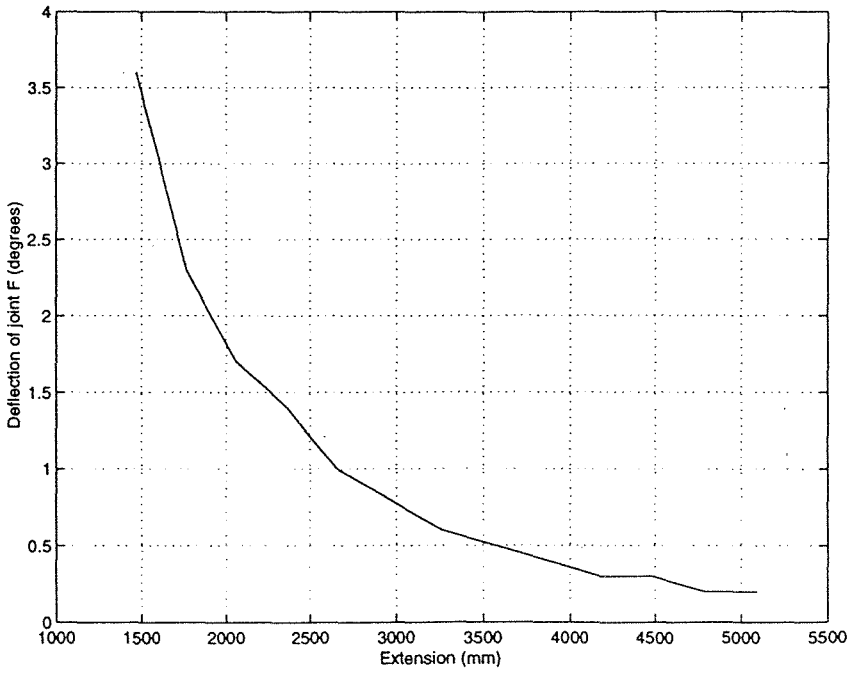

Figure 3.3: Deflection of joint F under constant torque

attach the idler links to the robot structure failed during normal testing conditions. Luckily a new system was already under development at this time.

3.2 Redesign of the Parallel Mechanism/Retraction Mechanism

The requirements of the parallel mechanism/retraction mechanism, as stated before, are to orient the end-effector perpendicular to the ground at any location and to retract the end-effector for stowage. These requirements can be simply solved separately, but the combination requires a more complex device.

A mechanism was developed using a configuration similar to a crank slider mechanism involving four revolute joints and one slider joint. Figure 3.4 shows the general configuration of this mechanism relative to the current robot structure. This design requires less links than the previous two designs, but involves one additional slider joint. This configuration will orient the end-effector but does not have the capability for stowage.

A combination of both the parallel and retraction mechanism requires a configuration which may be similar to Figure 3.5. This system is composed of a telescopic link that was designed to carry a moment load which could also retract. A pair of pneumatic cylinders actuate the retraction while the tightly coupled telescopic link carries the moment load. To retract the end-effector to a horizontal position, re-

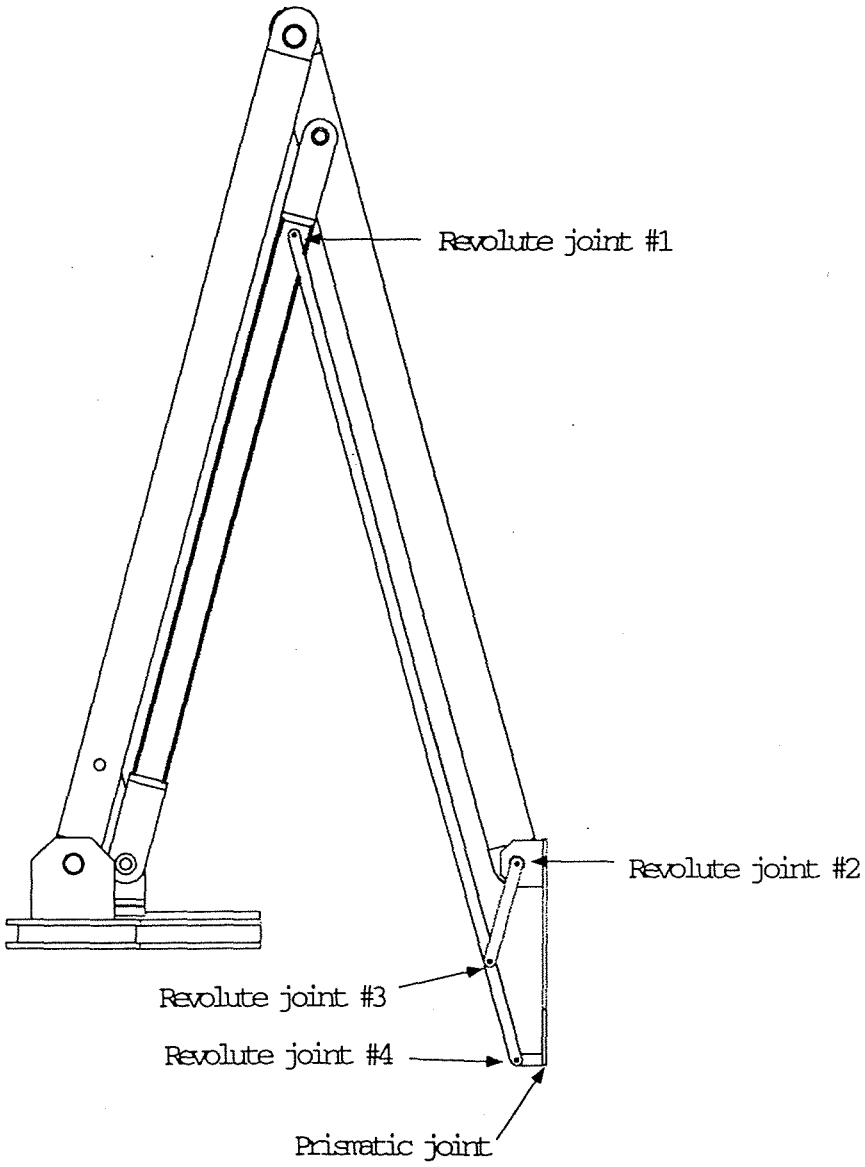

Figure 3.4: Conceptual design of the improved parallel linkage

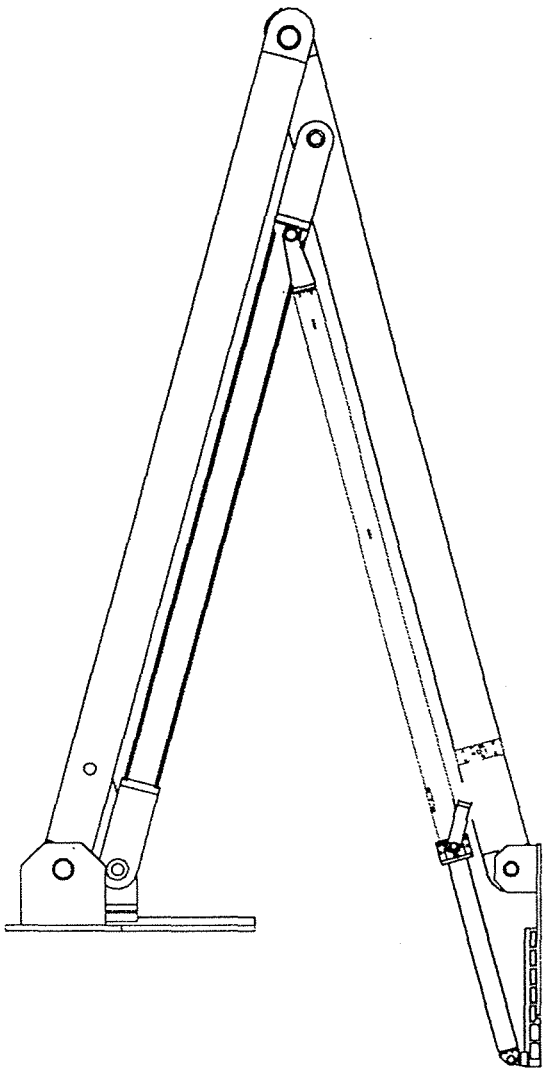

Figure 3.5: Actual design of the improved parallel linkage in extended position

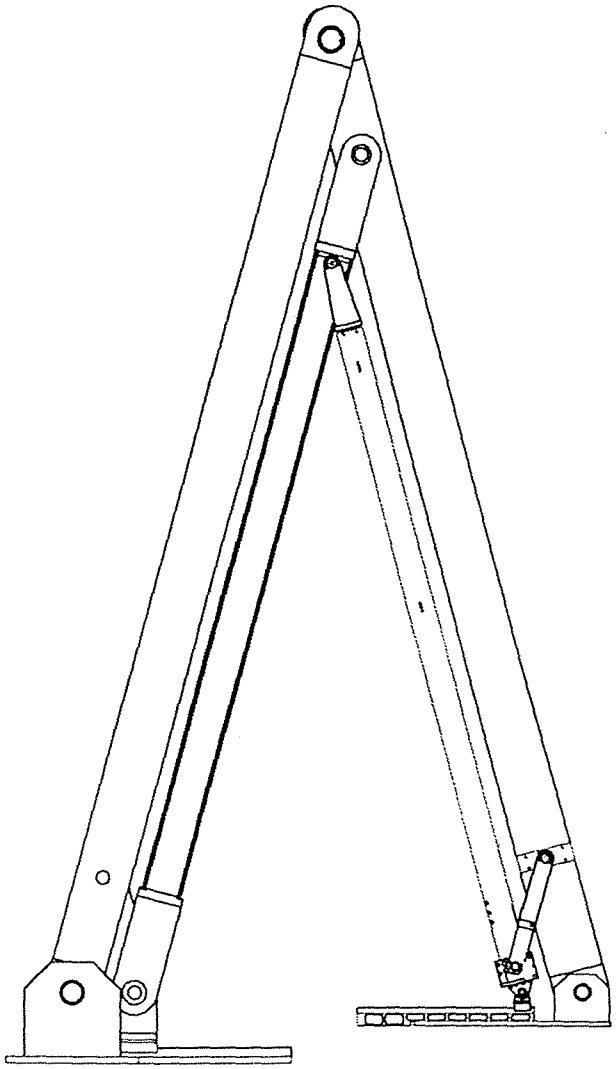

Figure 3.6: Actual design of the improved parallel linkage in retracted position

quired placing the cross member link higher along the robot structure link as shown in Figure 3.6.

3.3 Static and Dynamic Loading

The previous design of the parallel linkage had severe problems with stiffness. Under static loading the structure appeared capable of supporting the large mass of the end-effector, but during dynamic movements the end-effector would oscillate violently. To address this problem the new design had fewer links and larger cross sections . An analysis was performed to determine the loading this linkage would undergo. The force $\overline{F_R}$ acting on the linkage is necessary to determine resulting deflections and stresses. In this analysis we will be looking at the forces experience during an extension move. Figure 3.7 shows the end-effector as it mounts to the robot structure on the left and the corresponding free-body diagram of the end-effector on the right. From this free-body diagram, summing moments at A, $\sum M_A$, $\overline{F_R}$ can be determined by [9]

$$\overline{F_R} = \frac{mgd + F_{EE}b}{c} \tag{3.1}$$

where F_{EE} is the force due to the horizontal acceleration of the robot arm. This is calculated as the mass m of the end-effector and universal mounting plate, multiplied by the acceleration A_{EE}. This force is applied at the center of mass of the end-effector.

The resulting force F_R is graphed in Figure 3.8 where R is the extension distance

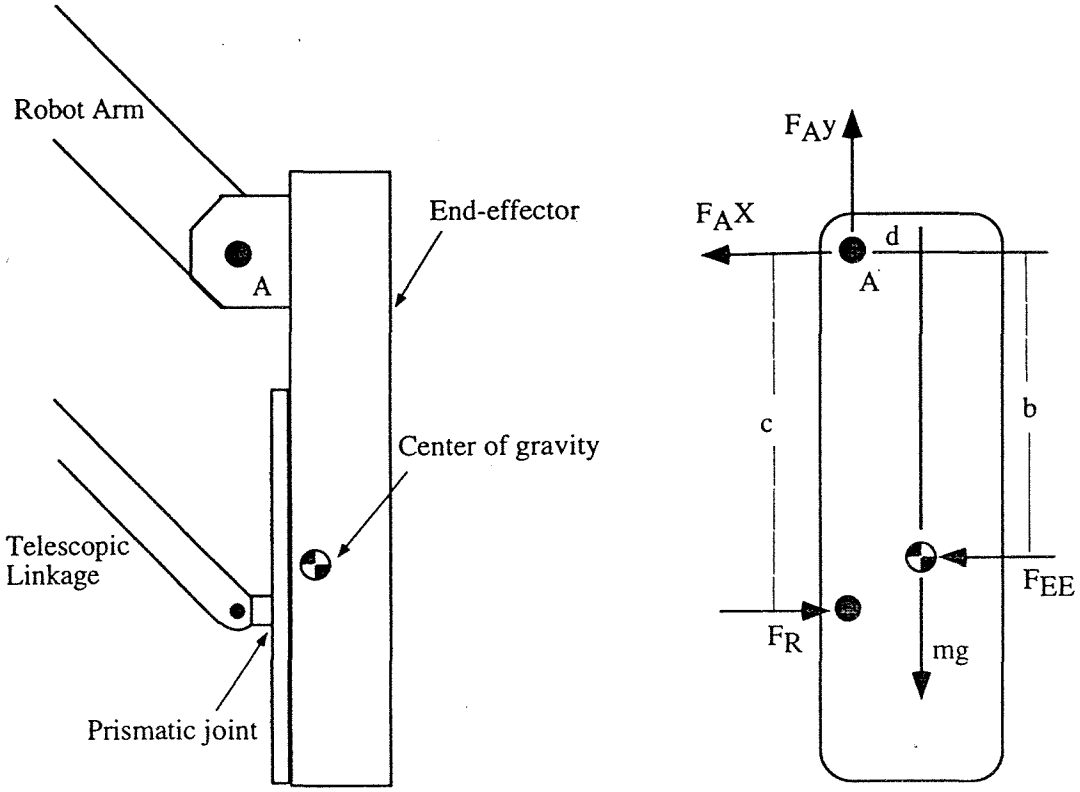

Figure 3.7: Free-body diagram of end-effector and support linkage

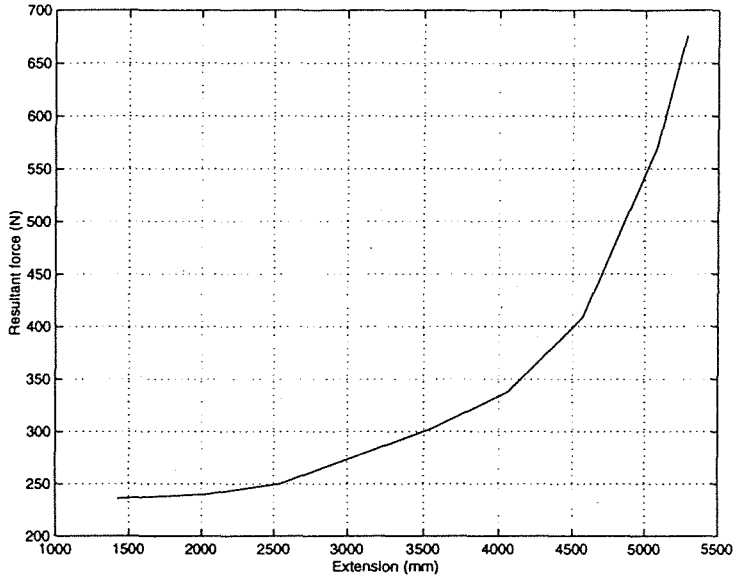

Figure 3.8: Resultant force acting on the parallel linkage

of the robot. The acceleration value of the end-effector used is twice the value of the acceleration calculated during a typical path. The resultant force can be broken into components F_x and F_y which correspond to an axial force on the linkage and a bending force respectively. F_x and F_y are shown in figures 3.9 and 3.10.

3.4 Analysis of the Telescopic Linkage

The deflection of the end-effector due to deflections of the telescopic linkage can be characterized by the sum of three sources. These three sources are considered to be the major contributors for which two basic assumptions have been made. These two assumptions are:

- Joint deflections can be considered very small and are neglected.

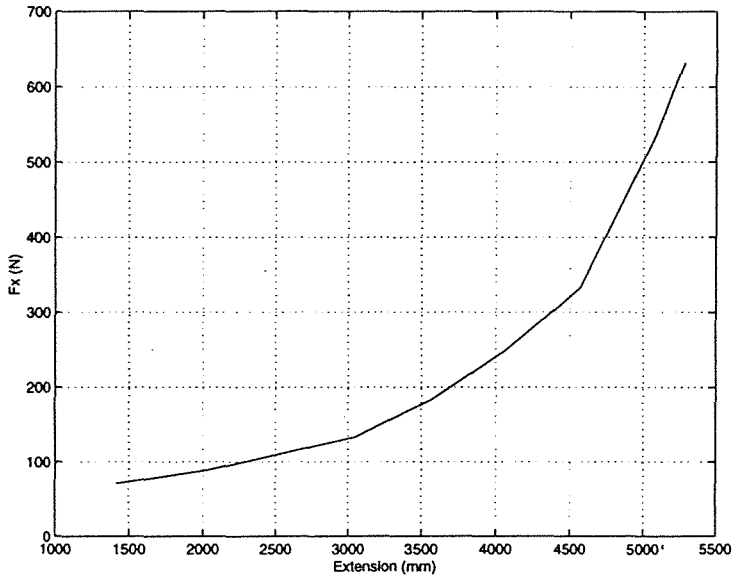

Figure 3.9: Axial F_x force acting on the parallel linkage

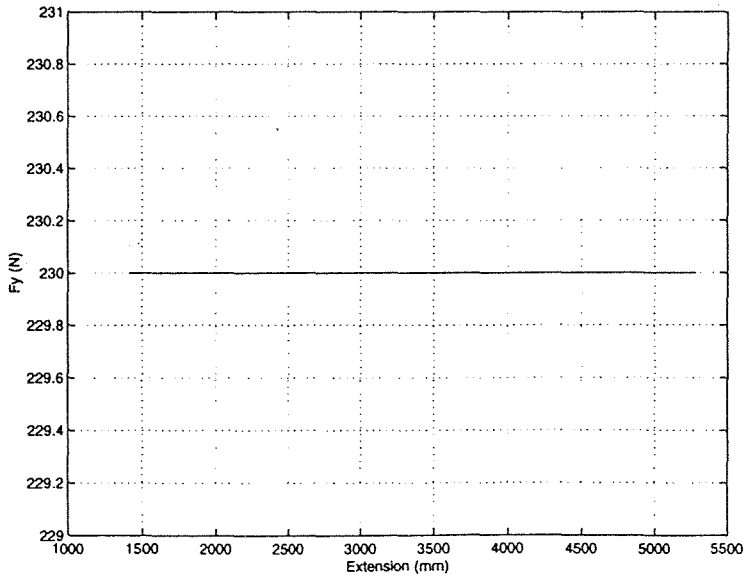

Figure 3.10: Bending F_y force acting on the parallel linkage

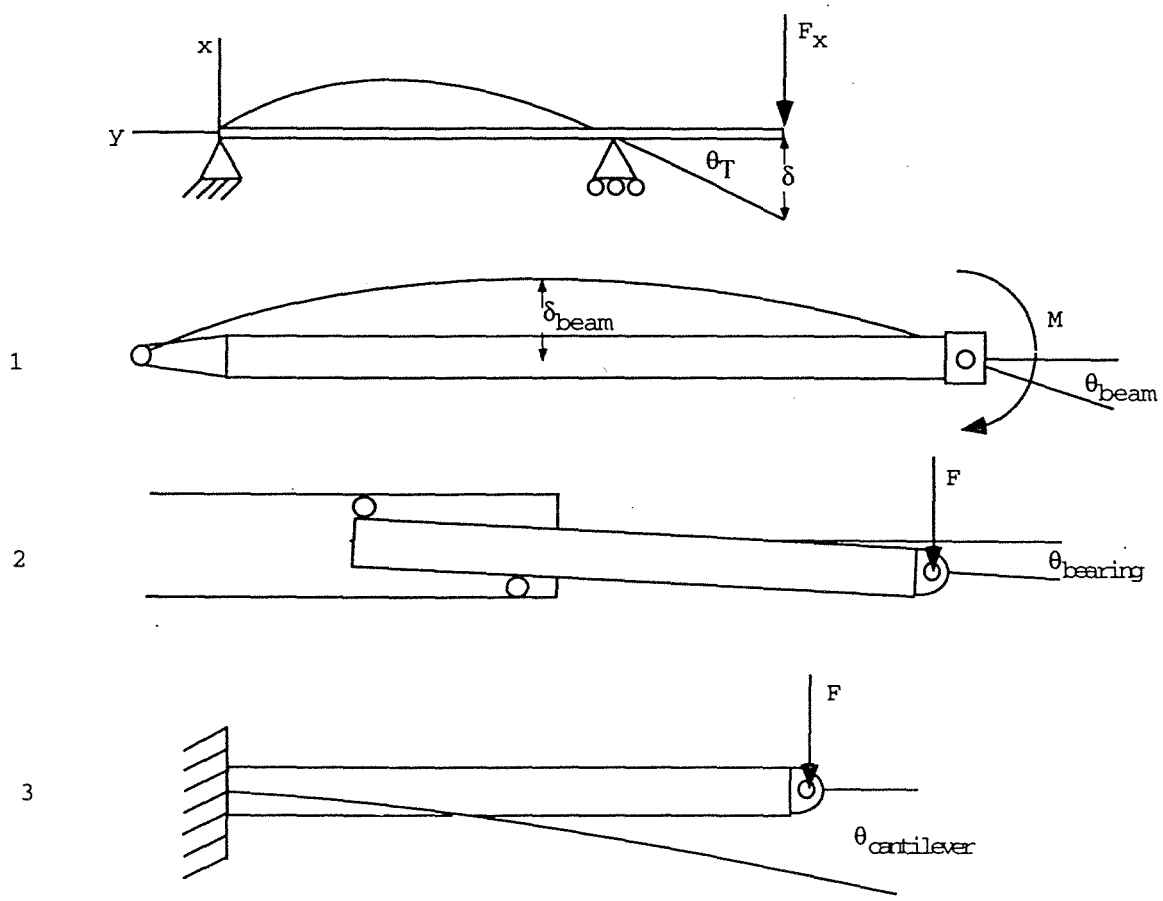

Figure 3.11: Three sources of deflection

• Material properties are considered to be isotropic.

The three sources of deflection are shown in figure 3.11. Where the total deflection δ_{Total} of the end of the telescopic linkage is the sum of the three sources of deflection in cases 1,2, and 3. Case 1 is the bending of the large beam due to the moment applied by the cantilever portion of linkage. Case 2 is the deflection resulting from contact stress of the roller bearings pressing against the structural tubing. Case 3 is the deflection resulting from the cantilevered portion of the linkage. The total

deflection can be calculated using equation 3.2,

$$\delta_{Total} = \sin(\theta_{beam} + \theta_{bearing}) L_{cantilever} + \delta_{cantilever} \tag{3.2}$$

where θ_{beam} is the externally created angular displacement of the large simply supported tubular beam at the end point where the cantilever beam is joined. $\theta_{bearing}$ is the externally created angular displacement of the cantilever section caused by local deformation of the tubular structure from the contact of the rollers. $L_{cantilever}$ is the overhanging length of the beam. $\delta_{cantilever}$ is the deflection of the cantilever portion of the beam. θ_{beam} can be calculated using this equation

$$\theta_{beam} = \frac{M L_{beam}}{6EI} \tag{3.3}$$

where M is the moment applied by the cantilever beam, and L_{beam} is the length of the large simply supported tubular beam.

The externally created angular displacement $\theta_{bearing}$ is

$$\theta_{bearing} = \tan^{-1}(\frac{\delta_1}{l_{bearing}}) + \tan^{-1}(\frac{\delta_2}{l_{bearing}}) \tag{3.4}$$

where δ_1 and δ_2 are the local deflections of the contact of the roller and tubular structure, and $l_{bearing}$ is the distance between contact points. Figure 3.12 shows a cutaway view of the telescopic linkage revealing the geometry of the rollers in contact with the structural tubing. To accurately solve for stresses and deflections, finite element models were used. Using SDRC's I-deas Masters Series Simulation software, a near exact replication of the problem can be modeled.

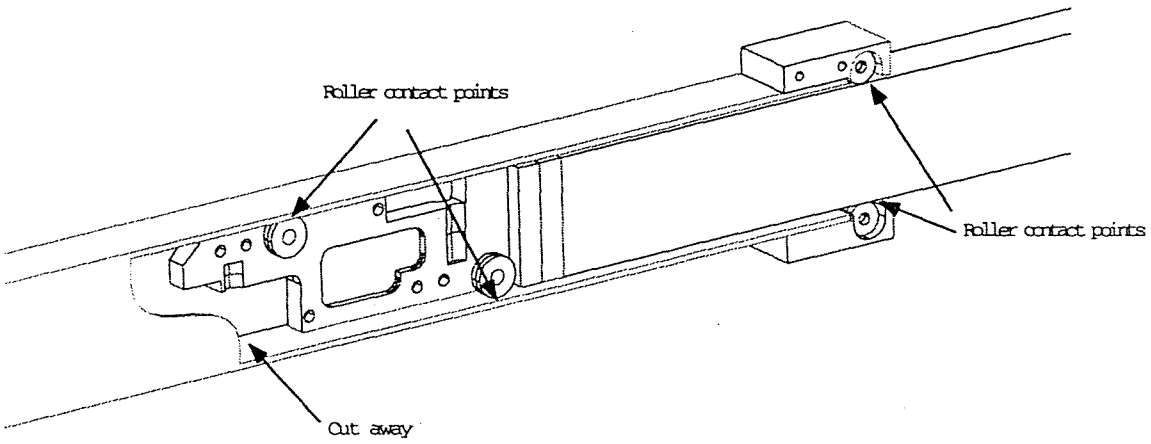

Figure 3.12: Cutaway view of the telescopic linkage

Figure 3.13 shows the process of using finite elements. It is assumed that the rollers will cause only local deformations; therefore, the size of the model can be reduced as the process shows. The upper picture is the original geometry of the problem showing the rollers in contact to the tubing. The middle picture is simplification of the problem showing the replacement of two rollers with two pressures. The lower picture is the geometry with elements and restraints added.

The results to the model is shown in Figure 3.14. The maximum displacement orthogonal to the beam is .0030 mm (.00012 in) and the maximum stress is 34.4 MPa (5 ksi).

A second model representing the rollers inside the tubing was created. The process is shown in Figure 3.15. The process is the same as previously stated. The results, as shown in Figure 3.16, show the maximum displacement being .010 mm (.0004 in) and maximum stress of 31.0 MPa (4.5 ksi). With the displacement values for both sets of rollers Equation 3.4 can be solved.

The third source of deflection can be attributed to the cantilever section of the linkage. This is easily solved using the deflection formula for a cantilever beam.

$$\delta_{cantilever} = \frac{FL^3}{3EI} \tag{3.5}$$

3.4.1 Results of the Analysis

The total deflection of the linkage can be closely calculated using equation 3.2 as mentioned previously. Using a force of 231 N (52 lbs) the components of Equation 3.2

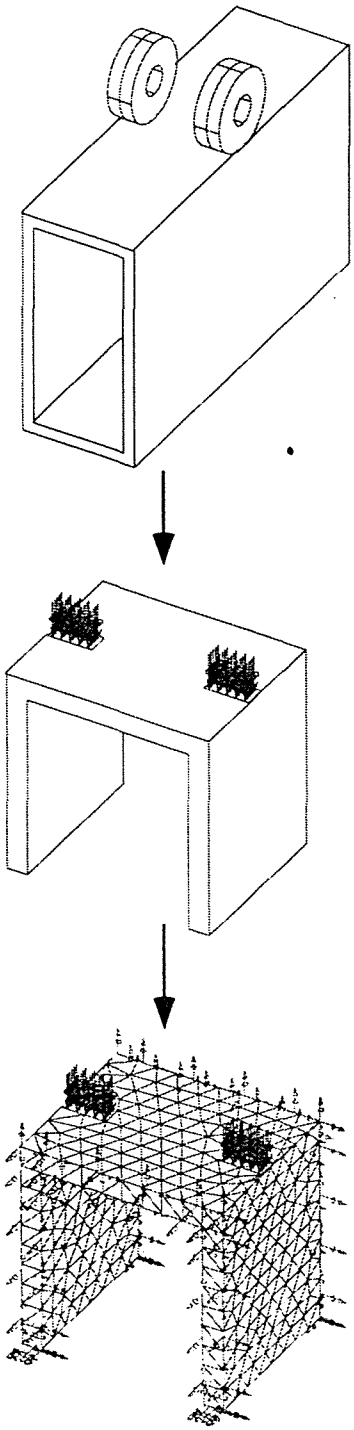

Figure 3.13: Method of solving for the deflection of the roller contact area

RESULTS:
STRESS - VON MISES MIN: 2.99E+01 MAX: 5.45E+03
DISPLACEMENT - MAG MIN: 0.00E+00 MAX: 1.21E-04

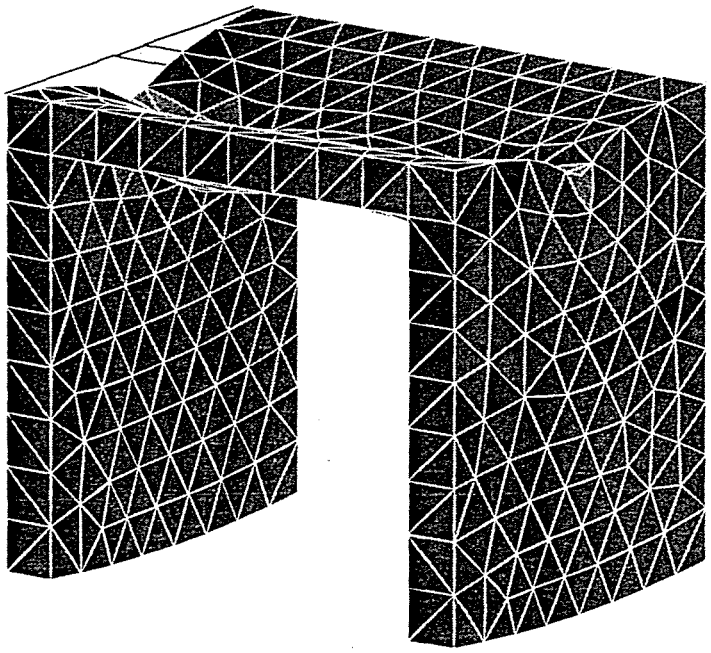

Figure 3.14: Results of the first finite element model

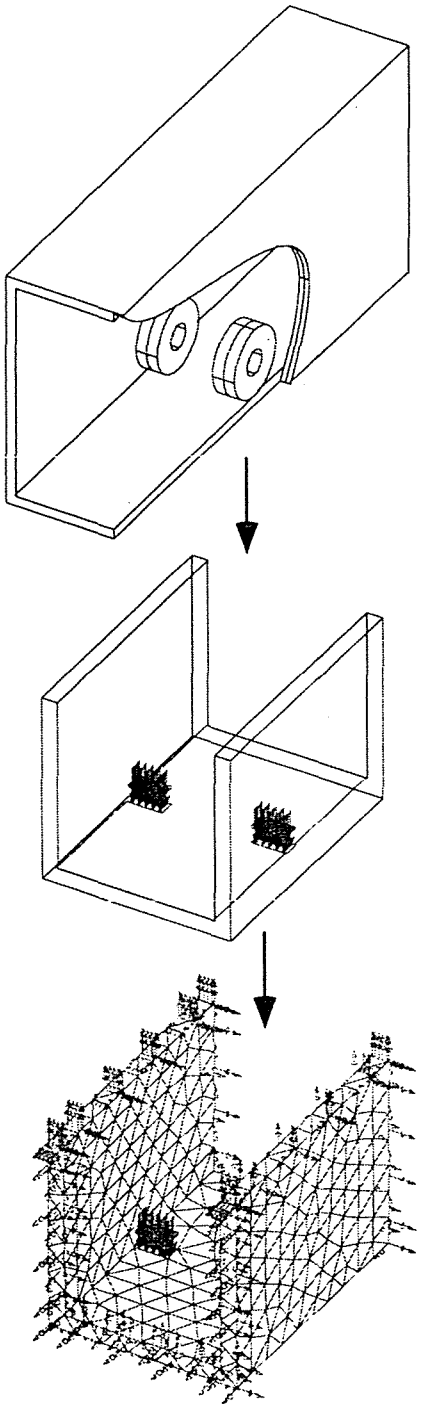

Figure 3.15: Method of solving for the deflection of the roller contact area

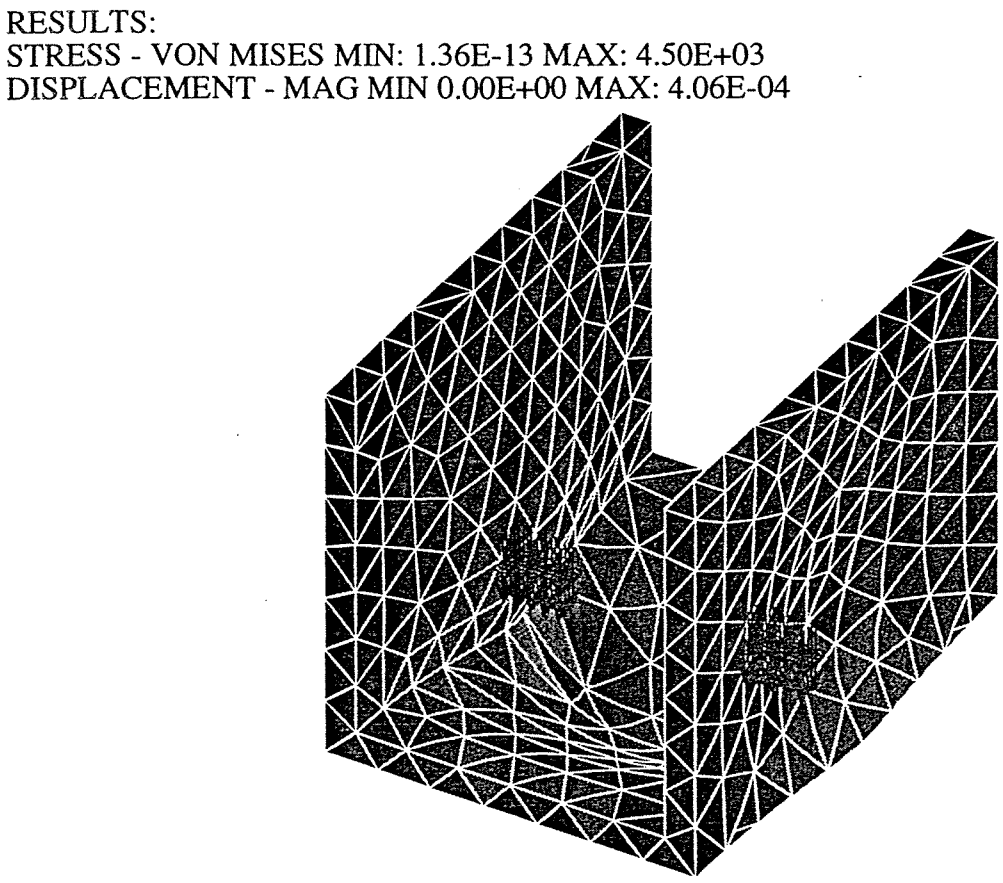

Figure 3.16: Results of the second finite element model

are calculated as follows:

$$\theta_{beam} = \frac{ML_{beam}}{6EI}$$

$$\theta_{beam} = \frac{(158.7N - m)(2057.4mm)}{(6)(68.9GPa)(5.03e05mm^4)}$$

$$\theta_{beam} = 0.0016\ radians \quad or \quad .089\ degrees$$

$$\theta_{bearing} = \tan^{-1}(\frac{\delta_1}{l_{bearing}}) + \tan^{-1}(\frac{\delta_2}{l_{bearing}})$$

$$\theta_{bearing} = \tan^{-1}(\frac{.010mm}{508.0mm}) + \tan^{-1}(\frac{.003mm}{508.0mm})$$

$$\theta_{bearing} = 0.0015\ radians \quad or \quad .085\ degrees$$

$$\delta_{cantilever} = \frac{FL^3}{3EI}$$

$$\delta_{cantilever} = \frac{(231N)(686.1mm)^3}{3(68.9GPa)(2.82e05mm^4)}$$

$$\delta_{cantilever} = .04826mm(.0019in)$$

$$\delta_{Total} = \sin(\theta_{beam} + \theta_{bearing})L_{cantilever} + \delta_{cantilever}$$

$$\delta_{Total} = \sin(0.0016\ radians + 0.0015\ radians)686.1mm + .04826\ mm$$

$$\delta_{Total} = 2.134mm(0.084in)$$

The deflection of the linkage at the endpoint due to the orthogonal force $F_x = 231N(52\ lbf)$ is shown to be approximately $2.134mm(0.084\ inches)$. This however is not the actual deflection at the spray gun height. The deflection at the spray gun

nozzle location can be calculated from this equation:

$$\delta_{spraygun} = \frac{H_{spraygun}}{635mm}\delta_{Total} \tag{3.6}$$

$$\delta_{spraygun} = \frac{914.4mm}{635mm}2.134mm$$

$$\delta_{spraygun} = 3.073mm(0.121in)$$

where $H_{spraygun}$ is the current distance between the spray gun nozzle and the end-effector revolute joint F.

3.5 Experimental Results

A test was performed to capture deflection data experimentally and compare it to the previous parallel linkage. The test consisted of applying a constant torque of 101.7 N-m (900 in-lbs) at joint F, and measuring angular displacement at defined extension increments. This test was also performed on the previous linkage which was discussed briefly in Section 3.1.1. The results of this test are tabulated in Table 3.2 and shown in Figure 3.17.

Robot link extension (mm)	Angular deflection of of joint F (degrees)		Deflection at nozzle height (mm)	
	previous	current	previous	current
1470	3.6	0.18	89	4.0
1765	2.3	0.18	56	4.0
2057	1.7	0.18	41	4.0
2362	1.4	0.18	33	4.0
2654	1.0	0.18	22	4.0
2959	0.8	0.19	20	4.5
3264	0.6	0.19	15	4.5
3567	0.5	0.19	13	4.5
3874	0.4	0.19	10	4.5
4178	0.3	0.19	8	4.5
4483	0.3	0.19	8	4.5
4788	0.2	0.19	5	4.5
5093	0.2	0.20	5	4.5

Table 3.2: Deflection of joint F under constant torque for the previous and current linkage

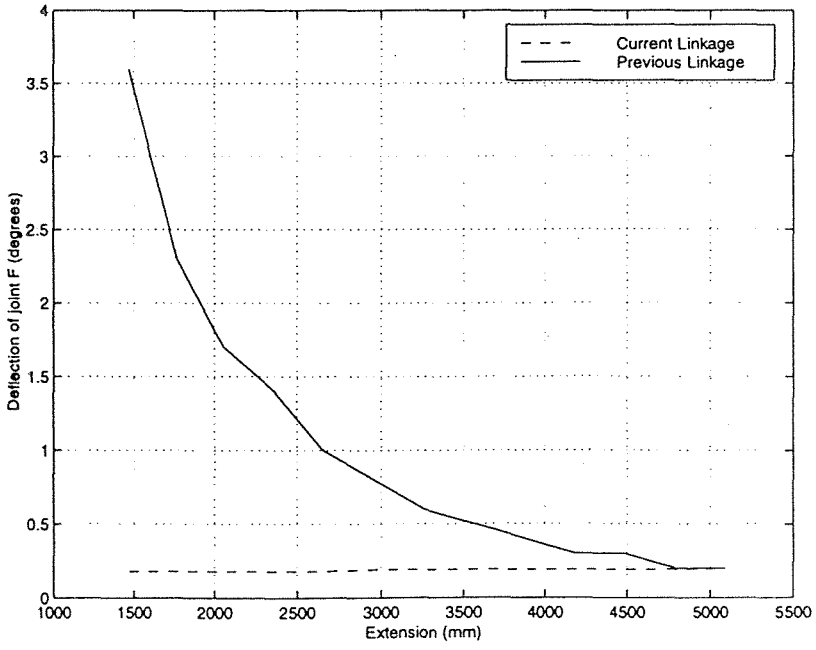

Figure 3.17: Deflection of joint F under constant torque

Chapter 4

CONCLUSIONS AND RECOMMENDATIONS

4.1 Conclusions

There are many stages to the development of a robot used for spray painting of roadway markings. This thesis discusses the development of the algorithms for trajectory planning and a mechanical redesign of the Stenciling Robot parallel linkage for improved performance.

Trajectory planning is an essential component of the total system needed for robotic spray painting of roadway markings. Although only letters were discussed here, the technique developed is adaptable to a wide range of symbols and markings. The complexity can vary depending on the desired paint distribution. The incorpo-

ration of tilt can introduce an almost seamless paint thickness. That is quite a feat using a single standard high pressure paint nozzle. The trajectory planning can be performed off-line, therefore the code necessary for generation of the trajectory does not need to be simple.

To improve the performance of the trajectory planning technique, a redesign of the parallel linkage was needed. To produce quality markings, the spray gun needs to be positioned precisely during its trajectory. Accelerations during motion make the inherently unbalanced end-effector want to deviate from its position. The parallel linkage is used to orient the end-effector vertically at all times when painting. The linkage is also fitted with an integrated retraction mechanism used to retract the end-effector for stowage. The combined responsibility of the linkage lends itself nicely to the telescopic style linkage developed for this purpose. The linkage showed an order of magnitude improvement over the previous linkage during the majority of the workspace in terms of deflection.

4.2 Recommendations

The C code written for the trajectory planning gives an array of cartesian locations and corresponding yaw, tilt, and velocity parameters. The spacing between these points do not correspond to any set parameter. For implementation purposes this requires some internal code within the robot controller needed to interpolate between these locations. This is called the "Motion Interpolater". For increased performance

it would be desirable to generate points with the spacing proportional to the desired velocity and accelerations. This would allow direct position and velocity control directly from the off-line generated trajectory. During real-time implementation, the control system updates its trajectory at even time intervals, typically at 32 msec. Therefore, each cartesian position generated would be a point which is 32 msec away from the previous.

The C code written has produced paths that are based on a single pass to paint a marking. It can be desirable to have multiple pass trajectories in times of heavy winds. The multiple pass trajectory would cause the spray nozzle to be position much closer to the surface reducing overspray.

Bibliography

[1] H. Anton. *Calculus with Analytic Geometry.* John Wiley and Sons, 1984.

[2] California Department of Transportation. *Standard Plans*, 1994.

[3] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide.* Academic Press, Inc., Boston, Mass., 1989.

[4] J. g. Bollinger. *Computer Control of Machines and Processes.* Addison-Wesley Publishing, New York, New York, 1989.

[5] H. Kochiekali and B. Ravani. A feature based path planning system for robotic stenciling of roadway markings. In *American Society of Civil Engineers: Conference on Robotics for Challenging Environments*, pages 52–60, 1994.

[6] R. A. McGrew. A robotic end-effector for roadway stenciling. Master's thesis, Department of Mechanical and Aeronautical Engineering, University of California, Davis, June 1996.

[7] R. H. Olshausen. Development of an articulating robotic arm for spray painting on roadways. Master's thesis, Department of MAE, University of California, Davis, June 1996.

[8] B. Ravani. Class notes eme 255 computer-aided design and manufacturing. University of California, Davis, 1995.

[9] J. E. Shigley and L. D. Mitchell. *Mechanical Engineering Design, Fourth Edition.* McGraw-Hill, New York, New York, 1983.

[10] K. S. Sprott, P. W. Wong, W. Nederbragt, R. Olshausen, and B. Ravani. A description of the photogrammetery target project premark painting sytem. UCD-ARR-94-09-01, UC Davis, 1994.

Appendix A

Trajectory Planning Software

This appendix includes the software used to generate the trajectories for the letter S and B.

```
/*****************************************************************
  Title:      pathS.c

  Author:     Rich Blank

  Date:       August 13, 1996

  Function:  This program combines information from 2 external
     function files as shown in the include statements
     to calculate and output into a format needed by
             the Adept Robot.
*****************************************************************/

#include <stdio.h>
#include <math.h>
#include "primitiveS.ff"
#include "primitive3.ff"

#define pi 3.141592654
#define letterlength 1800 /*Total height of letter in inches*/
```

```
#define totaln 140
#define straightspeed 600 /*6 inches per second speed max*/
#define xpan 400          /*Pan location */
#define ypan -900
#define thetarotdeg 0.0 /*Rotation angle in degrees*/
#define xrotorg 0.0       /*X Rotation origin*/
#define yrotorg 0.0       /*Y Rotation origin*/
#define linklength 110.0
void primitiveS(int n,int startline,double sprayangle,
double line[][5]);
void primitive3(int flag,int n,int startline,double sprayangle,
double line[][5]);
void main(void)
{
  double line[totaln + 2][5],r[totaln + 2],theta[totaln + 2],
  time[totaln + 2];
  long int rencoder[totaln + 2],thetaencoder[totaln + 2];
  int i,j;
  double scalesize,thetarot,speedkfactor;
  FILE *fp;
  FILE *fp1;

for(i=0; i <= totaln +1;i++)
   for(j=0; j < 5;j++)
      line[i][j] = 0.0;


primitive3(1,20,0,63.5,line);
primitive3(2,20,21,63.5,line);
primitiveS(60,41,63.5,line);
primitive3(3,20,100,63.5,line);
primitive3(4,20,121,63.5,line);

/*This next section involves scaling ,panning, and rotations*/
scalesize=letterlength/25.0;

thetarot = thetarotdeg*pi/180.0;

speedkfactor= straightspeed*1.8/(.025);
for (i=0; i <=totaln;i++)
  {
    line[i][0]=scalesize*line[i][0];
```

```
    line[i][1]=scalesize*line[i][1];
    line[i][3]=scalesize*line[i][3];
    /*convert linelength to hieght "z" using binks 1360 ***/
    line[i][3]=509.3+(.878715*line[i][3]);
    line[i][4]=speedkfactor*line[i][4]/scalesize;
  }
for (i=0; i <=totaln;i++)
  {
    line[i][0]=((line[i][0]-xrotorg)*cos(thetarot))-
((line[i][1]-yrotorg)*sin(thetarot))+xrotorg;
    line[i][1]=((line[i][0]-xrotorg)*sin(thetarot))+
((line[i][1]-yrotorg)*cos(thetarot))+yrotorg;
    line[i][2]=line[i][2]+thetarotdeg;
  }

for (i=0; i <=totaln;i++)
  {
    line[i][0]=line[i][0]+xpan;
    line[i][1]=line[i][1]+ypan;
  }
/*inverse kinematics*/
for(i=0; i <=totaln;i++)
  {
    r[i] = sqrt((line[i][0]*line[i][0])+
(line[i][1]*line[i][1]));
    r[i] = acos((r[i]/2)/linklength);
    theta[i] = atan((line[i][1]*line[i][1])/
(line[i][0]*line[i][0]));

  }

/*encoder transformation*/
for(i=0; i <=totaln;i++)
  {
    rencoder[i] = (long int)(r[i]*44000.0/pi);
    /* r is now in radians rotation*/
    thetaencoder[i] = (long int)(theta[i]*44000.0/pi);
  }
for(i=0; i <=totaln-1;i++)
  {
  time[i] =(line[i][4]/((sqrt((line[i][0]-line[i+1][0])*
          (line[i][0]-line[i+1][0])+(line[i][1]-
```

```
            line[i+1][1])*(line[i][1]-line[i+1][1])))));
  printf(" time %f \n",time[i]);
  }

fp = fopen("dataS.out", "w");
fp1 = fopen("phil.out", "w");
for (i=0;i<=totaln;i++)
  {
  fprintf(fp,"move trans(%5.1f,%5.1f,(%5.1f+offset),
          0,180,%5.1f)\n"
  ,line[i][0],line[i][1],line[i][3],line[i][2]);
  fprintf(fp,"speed (%5.1f*scale), rotate MMPS \n",
  line[i][4]);
  fprintf(fp1,"%d, %d, %d, %f\n",i,rencoder[i],
  thetaencoder[i],time[i]);
  }
fclose(fp);
fclose(fp1);
}


/*************************************************************
  Title:      primitive3.ff

  Author:     Rich Blank

  Date:       August 13, 1996

  Function:  This program is written as a function file
     to supplement the path calculation for the
             letter S
*************************************************************/

#include <stdio.h>
#include <math.h>

void TSspline(double trvtx[],double trvty[],int n);
void arcs(double tlvtx[],double tlvty[],int n);

void primitive3(int flag,int n,int startline,double sprayangle,
        double line[][5])
{
```

```
double trvtx[n+2],trvty[n+2],tlvtx[n+2],tlvty[n+2],
       tempvtx[n+2],tempvty[n+2];

/* double line[n+2][5];line[certain point for letter] */
/*        line[][0] x location (midpt location)*/
/*        line[][1] y location     */
/*        line[][2] angle    */
/*                line[][3] height    */
/*        line[][4] speed                  */
     double midlinex[n+2],midliney[n+2],temp[n+2][5],
            linelength[n+2],angle[n+2],height[n+2],speed[n+2];
     int i;
     int speedfactor = 1;

     TSspline(trvtx,trvty,n);
     arcs(tlvtx,tlvty,n);
/*reverse point numbering, since the functions return point
  0 as the top of the arc. We want 0 at the bottom
  pos (3.5,18.5)*/
for(i=0;i <=n;i++)
   {
   tempvtx[i] = tlvtx[i];
   tempvty[i] = tlvty[i];
   }
for(i=0;i <= n;i++)
   {
   tlvtx[i] = tempvtx[n-i];
   tlvty[i] = tempvty[n-i];
   }
for(i=0;i <=n;i++)
   {
   tempvtx[i] = trvtx[i];
   tempvty[i] = trvty[i];
   }
for(i=0;i <= n;i++)
   {
   trvtx[i] = tempvtx[n-i];
   trvty[i] = tempvty[n-i];
   }


/*midpoint calculation */
```

```
     for (i=0; i<=n;i++)
       {
 midlinex[i]=(trvtx[i]+tlvtx[i])/2;
 midliney[i]=(trvty[i]+tlvty[i])/2;
       }
/*angle calculation*/
     for (i=0; i<=n;i++)
      {
        angle[i]=(atan2((trvty[i]-tlvty[i]),
                 (trvtx[i]-tlvtx[i])));
/*atan2(y,x);*/
        angle[i]=angle[i]*(180/pi);
       }
/*height calculation*/
     sprayangle=sprayangle*pi/180;
     for (i=0; i<=n;i++)
     {
     linelength[i]=sqrt(pow((trvtx[i]-tlvtx[i]),2)+
          pow((trvty[i]-tlvty[i]),2));
     }
/*speed */
     for (i=0; i<=n;i++)
 speed[i]=speedfactor/linelength[i];

/*data management */
     for (i=0; i<=n;i++)
       {
 line[i+startline][0]=midlinex[i];
 line[i+startline][1]=midliney[i];
 line[i+startline][2]=angle[i];
 line[i+startline][3]=linelength[i];
 line[i+startline][4]=speed[i];
       }

/*user input pan and rotation*/
     switch (flag)/*flag = 1 top right original pos.
                  flag = 2 left top pos.
             flag = 3 bottom  right pos.
                  flag = 4 left bottom position */
       {
       case 2: for (i=0; i<=n;i++)/*pan to location 1,18*/
       line[i+startline][0]=4.0-line[i+startline][0];
```

```
      /*renumber lines and remove first point*/
      for (i=0; i<=n;i++)
{
 temp[i][0]= line[i+startline][0];
 temp[i][1]= line[i+startline][1];
 temp[i][2]= 180.0-line[i+startline][2];
                /*NOTICE that minus will switch angle*/
 temp[i][3]= line[i+startline][3];
 temp[i][4]= line[i+startline][4];
}
      for (i=0; i<=n;i++)
{
  line[i+startline][0]=temp[n-1-i][0];
  line[i+startline][1]=temp[n-1-i][1];
  line[i+startline][2]=temp[n-1-i][2];
  line[i+startline][3]=temp[n-1-i][3];
  line[i+startline][4]=temp[n-1-i][4];
}
        printf("CASE 2");
        break;
      case 3:   /* for "bottom right pos."*/
      for (i=0; i<=n;i++)
{
line[i+startline][2]=180.0-line[i+startline][2];
line[i+startline][1]=25.0-line[i+startline][1];
}
printf("CASE 3");
        break;
      case 4:  /*bottom left position*/
             for (i=0; i<=n;i++)
        {
line[i+startline][0]=4.0-line[i+startline][0];
line[i+startline][1]=25.0-line[i+startline][1];
}
      /*renumber lines and remove first point*/
      for (i=0; i<=n;i++)
{
  temp[i][0]= line[i+startline][0];
  temp[i][1]= line[i+startline][1];
  temp[i][2]= line[i+startline][2];
              /*NOTICE that minus will switch angle*/
```

```
    temp[i][3]= line[i+startline][3];
    temp[i][4]= line[i+startline][4];
  }
       for (i=0; i<=n;i++)
  {
    line[i+startline][0]=temp[n-1-i][0];
    line[i+startline][1]=temp[n-1-i][1];
    line[i+startline][2]=temp[n-1-i][2];
    line[i+startline][3]=temp[n-1-i][3];
    line[i+startline][4]=temp[n-1-i][4];
  }
       /* pan next*/
         printf("CASE 4");
         break;
       }/*end switch-case statement*/


    }

void TSspline(double trvtx[],double trvty[],int n)
{
  float px1= 0.00,px2= 2.00,px3= 4.00,px4= 2.00,a= 1.0;
  float py1 = 6.00,py2 = -0.20,py3 = 6.00,py4 = 8.0;

  int i,j,count,ns;

  double t,newt[n+2];
  double vtbetween[n+2],totallength,vtsum[n+100],tt[n+2],
         increment[n+100],interpolate[n+100],sum,modtotal,
 seglength, ydist[n+2];
  totallength = 0.0;
 /*n=numofsegments              # of segments in curve */

  for (i=0; i<n+1; i++)
  {
    t=i*(1.0/(float)n);

   trvtx[i] =
     (((    -a*t*t*t)+(       2*a*t*t)+(-a*t)+0)*px1)+
     ((( (2-a)*t*t*t)+(     (a-3)*t*t)+( 0*t)+1)*px2)+
     ((( (a-2)*t*t*t)+( (3-(2*a))*t*t)+( a*t)+0)*px3)+
     (((     a*t*t*t)+(        -a*t*t)+( 0*t)  )*px4);
```

```
   trvty[i] =
     (((    -a*t*t*t)+(       2*a*t*t)+(-a*t)+0)*py1)+
     ((( (2-a)*t*t*t)+(     (a-3)*t*t)+( 0*t)+1)*py2)+
     ((( (a-2)*t*t*t)+( (3-(2*a))*t*t)+( a*t)+0)*py3)+
     (((     a*t*t*t)+(      -a*t*t)+( 0*t)  )*py4);
  }

/* these next lines will get distances between points*/
  for (i=0; i<=n-1 ; i++)
     vtbetween[i] =sqrt((pow((trvtx[i+1]-trvtx[i]),2.0))+
                   (pow((trvty[i+1]-trvty[i]),2.0)));

/* these next lines will add up the total length */
  for (i=0; i<=n-1 ; i++)
     totallength=totallength+vtbetween[i];

/* calculate the approximate segment length*/
     modtotal=totallength;
     seglength=modtotal/(float)n;

/* get sum of chord lengths*/
  vtsum[0] =vtbetween[0];
  for(i=1; i<=n-1 ; i++)
    vtsum[i]= vtsum[i-1]+vtbetween[i];

/*original values of t*/
 for(i=0;i <=n;i++)
   tt[i]=i*(1.0/(float)n);

/*number of points on straight segment*/
  ns=0/seglength;
  ns = 0;
/* calculation of increments  */
  increment[0] = seglength;
  for (i=1; i<=n-1 ;i++)
    increment[i] = increment[i-1]+seglength;

/* interpolation routine*/
    count = 0;
    i = 0;
 for (j=0; j<=n+10 ;j++)
```

```
        {
        if ( vtsum[i] >= increment[count])
         {
interpolate[count] = tt[i]+(((tt[i+1]-tt[i])/
            (vtsum[i]-vtsum[i-1]))*
    (increment[count]-vtsum[i-1]));

    if (vtsum[i] >= increment[count + 1]) i=i-1;
    count = count +1;
 }
        if (count > (n-ns-1) )  break;
        i=i+1;
        }

/*flag the interpolate numbers for new values of "t"  */
j=1;
for (i=0; i <= count;i++)
  {
  if (interpolate[i] > 1.0)
break;
  if (interpolate[i] > 0.0 )
    {
      newt[j]=interpolate[i];
      j=j+1;
    }
  }
/* recalculate (x & y) positions of spline */

 for (i=0; i<=n-ns; i++)
  {
    newt[0] =0.0;

    t=newt[i];

    trvtx[i]=
     (((    -a*t*t*t)+(       2*a*t*t)+(-a*t)+0)*px1)+
     ((( (2-a)*t*t*t)+(     (a-3)*t*t)+( 0*t)+1)*px2)+
     ((( (a-2)*t*t*t)+( (3-(2*a))*t*t)+( a*t)+0)*px3)+
     (((     a*t*t*t)+(        -a*t*t)+( 0*t)  )*px4);

    trvty[i]=
     (((    -a*t*t*t)+(       2*a*t*t)+(-a*t)+0)*py1)+
```

```
     ((( (2-a)*t*t*t)+(     (a-3)*t*t)+( 0*t)+1)*py2)+
     ((( (a-2)*t*t*t)+( (3-(2*a))*t*t)+( a*t)+0)*py3)+
     (((     a*t*t*t)+(        -a*t*t)+( 0*t)  )*py4);
  }
/*calculation of point locations on straight segment*/
 j=0;
  for(i=n;i >=n-ns  ;i--)
    {
      trvtx[i] = 4.0;
      trvty[i] = 6.0-(j*seglength);
      j=j+1;
    }
/*Mirror y locations for correct alignment*/
for(i=0; i<=n;i++)
  {
    ydist[i]= 25.0-trvty[i];
    trvty[i] = ydist[i];
  }

}
/*******************************************************/


void arcs(double tlvtx[],double tlvty[],int n)
/*n is # of segs in curve*/
{
int    i,j,count,ns;
double totalarclength,newt[n+2],vtbetween[n+2],
       vtsum[n+2],T[n+2],rx,ry,rrx,rry;
double increment[n+2],interpolate[n+2],modtotal,
       segmentlength,t,p,k;

ry=2.0;
rx=1.0;
totalarclength = 0.0;
j=0;
rrx=n;
rry=1.0;
for(i=n; i >= 0;i--)
{
 k=1-((float)i/(float)n);
 p=pow((rrx*rrx)*(1-((k*k)/(rry*rry))),0.5);
```

```
 T[j]=ry*p*(1.0/(float)n);
 tlvty[j] = T[j];

if (tlvty[j] == 2.0)
   {
     printf("value was 2 \n");
     tlvtx[j] = 0;
   }
 else
     tlvtx[j] = pow((rx*rx)*(1.0-((tlvty[j]*tlvty[j])/
                (ry*ry))),0.5);
 j++;
}
/*step 2 (find length of chords of the arc)*/

for(i=0;i <= n-1;i++)
   vtbetween[i] = sqrt((pow((tlvtx[i+1]-tlvtx[i]),2.0))+
                  (pow((tlvty[i+1]-tlvty[i]),2.0)));

/*step 3 (add up total length of arc)*/

for(i=0;i <= n-1;i++)
   totalarclength = totalarclength + vtbetween[i];

/*step 4(calculate approximate seg length
          including strght portion)*/

   modtotal = totalarclength + 1.0;
   segmentlength = modtotal/(float)n;

/*step 5 (get sum of chord lengths)*/

   vtsum[0] = vtbetween[0];
   for(i=1; i <=n-1; i++)
     vtsum[i] = vtsum[i-1]+vtbetween[i];

/*step 6 (get number points on straight line segment)*/

     ns = 1 /segmentlength;

/*step 7 (get increment -  sum of the segment lengths)*/
  increment[0] = 0;
```

```
  for (i=1; i<=n-1 ;i++)
    increment[i] = increment[i-1]+segmentlength;

/*step 8(interpolation routine:
         very confusing logic but it works!)*/
    count = 0;
    i = 0;
  for (j=0; j<=n+10 ;j++)
       {
       if ( vtsum[i] >= increment[count])
         {
 interpolate[count] = T[i]+(((T[i+1]-T[i])/
        (vtsum[i]-vtsum[i-1]))*
(increment[count]-vtsum[i-1]));
    if (vtsum[i] >= increment[count + 1]) i=i-1;

    count = count +1;
 }
       if (count > (n-ns-1) )  break;
       i=i+1;
       }

/*step 9(flag the interpolate numbers for new values
         of "t" (T))*/
j=0;
for (i=0; i <= count;i++)
  {
  if (interpolate[i] > 2.01)
break;
  if (interpolate[i] > 0.0 )
    {
      newt[j]=interpolate[i];
      j=j+1;
    }
  }
/*step 10 (calculate new (x & y) positions of arc) */

 for (i=0; i<= n-ns; i++)
    {
      t=newt[i];
      tlvty[i] = t;
      tlvtx[i] = pow((rx*rx)*(1-((tlvty[i]*tlvty[i])/
```

```
                (ry*ry))),0.5);
    }

 for(i=0; i<=n-ns;i++)
    tlvty[i] = tlvty[i]+1.0;

/*step 11 (calculation of pt locations on stght segment)*/
 j=0;
  for(i=n;i >=n-ns  ;i--)
    {
      tlvtx[i] = 1.0;
      tlvty[i] = (j*segmentlength);
      j++;
    }
/*step 12 (reposition x & y values)*/
  for(i=1; i<= n;i++)
    {
      tlvtx[i] = tlvtx[i] +2.0;
      tlvty[i] = tlvty[i] +18.0;
    }
  tlvtx[0] = 2.0;
  tlvty[0] = 21.0;
}



/*************************************************************
  Title:     primitiveS.ff

  Author:    Rich Blank

  Date:      August 13, 1996

  Function:  This program has been written to calculate a
             path used to spray paint the letter S.  This
             function is only valid for the middle section
             of the letter S.

*************************************************************/

#include <stdio.h>
#include <math.h>
```

```
#define pi 3.141592654
void midSspline(double mlvtx[],double mlvty[],int n);
void midSspline2(double mlvtx[],double mlvty[],
                 double mrvtx[],double mrvty[],int n);

void primitiveS(int n,int startline,double sprayangle,
                 double line[][5])
{
  double mlvtx[n+2],mlvty[n+2],mrvtx[n+2],mrvty[n+2];
  double midlinex[n+2],midliney[n+2],linelength[n+2],
         angle[n+2],height[n+2],speed[n+2];
  int i;
  float speedfactor = 1.0;

  midSspline(mlvtx, mlvty,n);
  midSspline2(mlvtx,mlvty, mrvtx, mrvty, n);

/*midpoint calculation */
     for (i=0; i<=n;i++)
       {
 midlinex[i]=(mrvtx[i]+mlvtx[i])/2;
 midliney[i]=(mrvty[i]+mlvty[i])/2;
       }
/*angle calculation*/
     for (i=0; i<=n;i++)
     {
      angle[i]=pi+(atan2((mrvty[i]-mlvty[i]),
               (mrvtx[i]-mlvtx[i])));
         /*atan2(y,x);*/
      angle[i]=angle[i]*(180/pi);
     }
/*height calculation*/
   sprayangle=sprayangle*pi/180;
   for (i=0; i<=n;i++)
   {
   linelength[i]=sqrt(pow((mrvtx[i]-mlvtx[i]),2)+
                 pow((mrvty[i]-mlvty[i]),2));
   }
/*speed */
     for (i=0; i<=n;i++)
 speed[i]=speedfactor/linelength[i];
```

```
/*data management */
     for (i=1; i<=n-1;i++)
       {
 line[i+startline-1][0]=midlinex[i];
 line[i+startline-1][1]=midliney[i];
 line[i+startline-1][2]=angle[i];
 line[i+startline-1][3]=linelength[i];
 line[i+startline-1][4]=speed[i];
       }
}
void midSspline(double mlvtx[],double mlvty[],int n)
{
  float mpx1= 2.00,mpx2= 4.00,mpx3= 1.00,mpx4= 3.00,
        a= 1.0;
  float mpy1 = 3.00,mpy2 = 10.00,mpy3 = 18.00,
        mpy4 = 25.00;
  int i,j,count,ns;

  double t,newt[n+2];
  double vtbetween[n+2],totallength,vtsum[n+100],tt[n+2],
         increment[n+100],interpolate[n+100],sum,
         modtotal,seglength,ydist[n+2];
         /*n= # of segments in curve */
  totallength = 0.0;

for (i=0; i<n+1; i++)
  {
    t=i*(1.0/(float)n);

   mlvtx[i] =
     (((    -a*t*t*t)+(       2*a*t*t)+(-a*t)+0)*mpx1)+
     ((( (2-a)*t*t*t)+(     (a-3)*t*t)+( 0*t)+1)*mpx2)+
     ((( (a-2)*t*t*t)+( (3-(2*a))*t*t)+( a*t)+0)*mpx3)+
     (((     a*t*t*t)+(        -a*t*t)+( 0*t)  )*mpx4);

   mlvty[i] =
     (((    -a*t*t*t)+(       2*a*t*t)+(-a*t)+0)*mpy1)+
     ((( (2-a)*t*t*t)+(     (a-3)*t*t)+( 0*t)+1)*mpy2)+
     ((( (a-2)*t*t*t)+( (3-(2*a))*t*t)+( a*t)+0)*mpy3)+
     (((     a*t*t*t)+(        -a*t*t)+( 0*t)  )*mpy4);
  }
/* these next lines will get distances between points*/
```

```
  for (i=0; i<=n-1 ; i++)
     vtbetween[i] =sqrt((pow((mlvtx[i+1]-mlvtx[i]),2.0))+
                  (pow((mlvty[i+1]-mlvty[i]),2.0)));

/* these next lines will add up the total length */
  for (i=0; i<=n-1 ; i++)
     totallength=totallength+vtbetween[i];

/* calculate the approximate segment length*/
     modtotal=totallength+4;
     seglength=modtotal/(float)n;

/* get sum of chord lengths*/
  vtsum[0] =vtbetween[0];
  for(i=1; i<=n-1 ; i++)
    vtsum[i]= vtsum[i-1]+vtbetween[i];

/*original values of t*/
 for(i=0;i <=n;i++)
   tt[i]=i*(1.0/(float)n);

/*number of points on straight segment*/
  ns=4/seglength;

/* calculate increment values  */
  increment[0] = seglength;
  for (i=1; i<=n-1 ;i++)
    increment[i] = increment[i-1]+seglength;

/* interpolation routine */
    count = 0;
    i = 0;
    for (j=0; j<=n+10 ;j++)
       {
       if ( vtsum[i] >= increment[count])
           {
    interpolate[count] = tt[i]+(((tt[i+1]-tt[i])/
        (vtsum[i]-vtsum[i-1]))*
(increment[count]-vtsum[i-1]));

    if (vtsum[i] >= increment[count + 1]) i=i-1;
```

```
    count = count +1;
 }
       if (count > (n-ns-1) )  break;
       i=i+1;
       }

/*flag the interpolate numbers for new values of "t"  */
j=1;
for (i=0; i <= count;i++)
  {
  if (interpolate[i] > 1.0)
break;
  if (interpolate[i] > 0.0 )
    {
      newt[j]=interpolate[i];
      j=j+1;
    }
  }
/* recalculate (x & y) positions of spline */

 for (i=0; i<=n-ns; i++)
  {
    newt[0] =0.0;
    t=newt[i];

    mlvtx[i]=
     (((    -a*t*t*t)+(       2*a*t*t)+(-a*t)+0)*mpx1)+
     ((( (2-a)*t*t*t)+(     (a-3)*t*t)+( 0*t)+1)*mpx2)+
     ((( (a-2)*t*t*t)+( (3-(2*a))*t*t)+( a*t)+0)*mpx3)+
     (((     a*t*t*t)+(        -a*t*t)+( 0*t)  )*mpx4);

    mlvty[i]=
     (((    -a*t*t*t)+(       2*a*t*t)+(-a*t)+0)*mpy1)+
     ((( (2-a)*t*t*t)+(     (a-3)*t*t)+( 0*t)+1)*mpy2)+
     ((( (a-2)*t*t*t)+( (3-(2*a))*t*t)+( a*t)+0)*mpy3)+
     (((     a*t*t*t)+(        -a*t*t)+( 0*t)  )*mpy4);
  }
/*calculation of point locations on straight segment*/
 j=0;
  for(i=n;i >=n-ns  ;i--)
    {
     mlvtx[i] = 1.0;
```

```
     mlvty[i] = 22.0-(j*seglength);
     j=j+1;
    }
/*adjust x & y locations for correct alignment*/
for(i=0; i<=n;i++)
  {
    mlvtx[i] = mlvtx[i]-1.0;
    mlvty[i] = mlvty[i]-3.0;
  }

}
void midSspline2(double mlvtx[],double mlvty[],
                 double mrvtx[],double mrvty[],int n)
{
  double tempx[n+2],tempy[n+2];
  int i;
for(i=0;i <= n;i++)
  {
    mrvtx[i]=4.0-mlvtx[i];
    mrvty[i]=25.0-mlvty[i];
  }

  /*flip values around*/
for(i=0;i <= n;i++)
  {
    tempx[i] = mlvtx[i];
    tempy[i] = mlvty[i];
  }
for(i=0;i <= n;i++)
  {
    mlvtx[i] = tempx[n-i];
    mlvty[i] = tempy[n-i];
  }
}
*********************************************************/

/*********************************************************
  Title:     pathB.c

  Author:    Rich Blank

  Date:      Octorber 5, 1995
```

```
  Function:  This program has been written to calculate
             a path used to spray paint the letter B.

************  ****************************************/
/*      Primitive #1 function
        items to input:
        primitive location(1,2,3,4) "flag"
        "speedfactor"
        number of segments(equals -1 number of points)"n"
        "startline" line number to begin with
        "spray angle" */
#include <stdio.h>
#include <math.h>
#include "arc.ff"
#include "interpolate3.c"

#define pi 3.141592654
#define letterlength 1900.0 /*height of letter in mm*/
#define totaln 100
#define straightspeed 600 /*strght line speed max*/
#define xpan 400.0          /*Pan location */
#define ypan -950.0
#define thetarotdeg 0.0 /*Rotation angle in degrees*/
#define xrotorg 0.0       /*X Rotation origin*/
#define yrotorg 0.0       /*Y Rotation origin*/
double line[100][5];

main()
{
int i;
double scalesize,thetarot,speedkfactor;
FILE *fp;

primitive2(0,0,1,1.0,0.5,0.0,25.0,63.5);
primitive1(1,20,2,1.0,63.5);
primitive1(2,20,23,1.0,63.5);
primitive1(3,20,43,1.0,63.5);
primitive1(4,20,64,1.0,63.5);

/*This next section involves scaling ,panning, and rot*/
scalesize=letterlength/25.0;
```

```
thetarot = thetarotdeg*pi/180.0;

speedkfactor= straightspeed*1.8/(.025);
for (i=0; i <=totaln;i++)
  {
    line[i][0]=scalesize*line[i][0];
    line[i][1]=scalesize*line[i][1];
    line[i][3]=scalesize*line[i][3]-20.828+528.145;
    line[i][4]=speedkfactor*line[i][4]/scalesize;
  }
for (i=0; i <=totaln;i++)
  {
    line[i][0]=((line[i][0]-xrotorg)*cos(thetarot))-
((line[i][1]-yrotorg)*sin(thetarot))+xrotorg;
    line[i][1]=((line[i][0]-xrotorg)*sin(thetarot))+
((line[i][1]-yrotorg)*cos(thetarot))+yrotorg;
    line[i][2]=line[i][2]+thetarotdeg;
  }

for (i=0; i <=totaln;i++)
  {
    line[i][0]=line[i][0]+xpan;
    line[i][1]=line[i][1]+ypan;
  }

fp = fopen("dataB.out", "w");
for (i=0;i<=123;i++)
  {
  fprintf(fp,"move trans(%5.1f,%5.1f,(%5.1f+offset),
  %5.1f,180,0)\n",line[i][0],line[i][1],line[i][3],
  line[i][2]);
  fprintf(fp,"speed (%5.1f*scale), rotate MMPS \n",
  line[i][4]);
  printf("%f  %f\n",line[i][0],line[i][1]);
  }
fclose(fp);
}

primitive1(flag,n,startline,speedfactor,sprayangle)

     int n,startline,flag;
```

```
     double speedfactor,sprayangle;

{

     double vtx[n+2],vty[n+2],svtx[n+2],svty[n+2];
    /* double line[n+2][5];line[certain point for letter]
   line[][0] x location (midpt loc)
   line[][1] y location
   line[][2] angle
   line[][3] height
   line[][4] speed*/
     double midlinex[n+2],midliney[n+2],temp[n+2][5],mirror,
 linelength[n+2],angle[n+2],height[n+2],speed[n+2];
     int i;

     arc(n,vtx,vty);
     spline(n,svtx,svty);

/*midpoint calculation */
     for (i=0; i<=n;i++)
       {
 midlinex[i]=(vtx[i]+svtx[i])/2;
 midliney[i]=(vty[i]+svty[i])/2;
       }
/*angle calculation*/
     for (i=0; i<=n;i++)
       {
      angle[i]=(atan2((svty[i]-vty[i]),(svtx[i]-vtx[i])));
/*atan2(y,x);*/
 angle[i]=angle[i]*(180/pi);
       }
/*height calculation*/
     sprayangle=sprayangle*pi/180;
     for (i=0; i<=n;i++)
       {
linelength[i]=sqrt(pow((svtx[i]-vtx[i]),2)+
pow((svty[i]-vty[i]),2));
height[i]=((linelength[i]/2)/tan(sprayangle/2));
       }
/*speed */
     for (i=0; i<=n;i++)
       {
```

```
 speed[i]=speedfactor/linelength[i];
       }
/*data management */
     for (i=0; i<=n;i++)
       {
 line[i+startline][0]=midlinex[i];
 line[i+startline][1]=midliney[i];
 line[i+startline][2]=angle[i];
 line[i+startline][3]=height[i];
 line[i+startline][4]=speed[i];
       }
/*user input pan and rotation*/
     switch (flag)/*flag = 1 top original pos.
                    flag = 2 flip top pos.
              flag = 3 bottom original pos.
                    flag = 4 flip bottom position */
       {
       case 2: for (i=0; i<=n;i++)
         {
   mirror=line[i+startline][1]-18;
   line[i+startline][1]=18-mirror;
 }
       /*renumber lines and remove first point*/
       for (i=0; i<=n;i++)
 {
  temp[i][0]= line[i+startline][0];
  temp[i][1]= line[i+startline][1];
  temp[i][2]=-line[i+startline][2];
/*NOTICE that minus will switch angle*/
  temp[i][3]= line[i+startline][3];
  temp[i][4]= line[i+startline][4];
 }
       for (i=0; i<=n;i++)
 {
   line[i+startline][0]=temp[n-1-i][0];
   line[i+startline][1]=temp[n-1-i][1];
   line[i+startline][2]=temp[n-1-i][2];
   line[i+startline][3]=temp[n-1-i][3];
   line[i+startline][4]=temp[n-1-i][4];
 }
         printf("CASE 2");
         break;
```

```
      case 3:/*pan to location (1,8) for "bottom org. pos."*/
      for (i=0; i<=n;i++)
{
      line[i+startline][1]=line[i+startline][1]-10.0;
}
printf("CASE 3");
        break;
      case 4:  /*flip and pan will occur next*/
             for (i=0; i<=n;i++)
        {
mirror=line[i+startline][1]-18;
line[i+startline][1]=18-mirror;
}
      /*renumber lines and remove first point*/
      for (i=0; i<=n;i++)
{
  temp[i][0]= line[i+startline][0];
  temp[i][1]= line[i+startline][1];
  temp[i][2]=-line[i+startline][2];
/*NOTICE that minus will switch angle*/
  temp[i][3]= line[i+startline][3];
  temp[i][4]= line[i+startline][4];
}
      for (i=0; i<=n;i++)
{
  line[i+startline][0]=temp[n-1-i][0];
  line[i+startline][1]=temp[n-1-i][1];
  line[i+startline][2]=temp[n-1-i][2];
  line[i+startline][3]=temp[n-1-i][3];
  line[i+startline][4]=temp[n-1-i][4];
}
      /* pan next*/
      for (i=0; i<=n;i++)
{
line[i+startline][1]=line[i+startline][1]-11.0;
}
        printf("CASE 4");
        break;
      }/*end switch-case statement*/
  }

primitive2(skip,startline,direction,speedfactor,x,y,
```

```
            length,sprayangle)

int skip,startline,direction;
double speedfactor,x,y,length,sprayangle;
{
switch (skip)
  {
  case 0:
    {
      line[startline][0]=x;
      line[startline][1]=y;

      line[startline+1][0]=x;
      line[startline+1][1]=y+(direction*length);


      line[startline][2]=0.0;/*angle set equal to zero*/
      line[startline+1][2]=0.0;

      line[startline][3]=((sin(pi-(sprayangle/2))*
(1/2))/sin(sprayangle/2));
      line[startline+1][3]=((sin(pi-(sprayangle/2))*
(1/2))/sin(sprayangle/2));
      line[startline][4]=speedfactor;
      line[startline+1][4]=speedfactor;
      break;
    }
  case 1:
    {
      line[startline][0]=x;
      line[startline][1]=y;
      line[startline][2]=0.0;/*angle set equal to zero*/
      line[startline][3]=((sin(pi-(sprayangle/2))*
 (1/2))/sin(sprayangle/2));
      line[startline][4]=speedfactor;
      break;
    }
  case 2:
    {
     line[startline+1][0]=x;
     line[startline+1][1]=y+(direction*length);
     line[startline+1][2]=0.0;
```

```
      line[startline+1][3]=((sin(pi-(sprayangle/2))*
   (1/2))/sin(sprayangle/2));
      line[startline+1][4]=speedfactor;
      break;
     }
   }
}
*****************************************************/


/************** ************************************
  Title:      interpolate3.ff

  Author:     Rich Blank

  Date:       October 5, 1995

  Function:   This program has been written as a
              function file to calculate the outside
              points of primitive #1
      used to spray paint the letter B.

*****************************************************/
/*approximation method
  this function will calculate points along a spline
  evenly spaced
  top spline of letter "B"                          */

#include <math.h>
#include <stdio.h>
#define px1 3.00
#define px2 1.00
#define px3 4.00
#define px4 2.00
#define py1 5.00
#define py2 0.00
#define py3 5.00
#define py4 15.2
#define a 1

spline(int n,double *vtx,double *vty)
 {
```

```
/* double px1,px2,px3,px4,py1,py2,py3,py4;*/
/* int a;*/

 int i,j,count,ns;

 double t,newt[n+2];
 double vtbetween[n+2],totallength,vtsum[n+2],tt[n+2],
        increment[n+2],interpolate[n+2],sum,modtotal,
        seglength,ydist[n+2];

 /*n=numofsegments          # of segments in curve */

 for (i=0; i<n+1; i++)
 {
   t=i*(1.0/(float)n);

   *(vtx+i)=
    (((    -a*t*t*t)+(       2*a*t*t)+(-a*t)+0)*px1)+
    ((( (2-a)*t*t*t)+(     (a-3)*t*t)+( 0*t)+1)*px2)+
    ((( (a-2)*t*t*t)+( (3-(2*a))*t*t)+( a*t)+0)*px3)+
    (((     a*t*t*t)+(        -a*t*t)+( 0*t)  )*px4);

   *(vty+i)=
    (((    -a*t*t*t)+(       2*a*t*t)+(-a*t)+0)*py1)+
    ((( (2-a)*t*t*t)+(     (a-3)*t*t)+( 0*t)+1)*py2)+
    ((( (a-2)*t*t*t)+( (3-(2*a))*t*t)+( a*t)+0)*py3)+
    (((     a*t*t*t)+(        -a*t*t)+( 0*t)  )*py4);
 }
 for (i=0;i<=n;i++)
    {
    printf("x %d %f y %d %f\n",i,vtx[i],i,vty[i]);
    }
/* these next lines will get distances between points*/
 for (i=0; i<=n-1 ; i++)
    {
    vtbetween[i] =sqrt((pow((*(vtx+i+1)-*(vtx +i)),2.0)+
  (pow((*(vty+i+1)-*(vty+i)),2.0))));
    }
/* these next lines will add up the total length */
 for (i=0; i<=n-1 ; i++)
    totallength=totallength+vtbetween[i];
```

```
/* calculate the approximate segment length*/
     modtotal=totallength+2.0;

     seglength=modtotal/(float)n;

/* get sum of chord lengths*/
  vtsum[0] =vtbetween[0];
  for(i=1; i<=n-1 ; i++)
    {
    vtsum[i]= vtsum[i-1]+vtbetween[i];
    }
/*original values of t*/
 for(i=0;i <=n;i++)
   {
   tt[i]=i*(1.0/(float)n);
   }
/*number of point s on straight segment*/
  ns=2/seglength;



/* interpolation  */
  increment[0] = seglength;
  for (i=1; i<=n-1 ;i++)
    {
    increment[i] = increment[i-1]+seglength;
    printf("increment %f \n",increment[i]);
    }

    count = 0;
    i = 0;
  for (j=0; j<=n+100 ;j++)
       {
       if ( vtsum[i] >= increment[count])
           {
    interpolate[count] = tt[i]+(((tt[i+1]-tt[i])/
         (vtsum[i]-vtsum[i-1]))*
 (increment[count]-vtsum[i-1]));
    if (vtsum[i] >= increment[count + 1]) i=i-1;
    count = count +1;
 }
       if (count > (n-ns-1) )  break;
```

```
        i=i+1;
        }

/*flag the interpolate numbers for new values of "t" */
j=1;
for (i=0; i <= count;i++)
  {
  if (interpolate[i] > 1.0)
break;
  if (interpolate[i] > 0.0 )
    {
      newt[j]=interpolate[i];
      printf("newt %f \n",newt[j]);
      j=j+1;

    }
  }
/* recalculate (x & y) positions of spline */

 for (i=0; i<=n-ns; i++)
  {
    newt[0] =0.0;
    t=newt[i];
    *(vtx+i)=
     (((    -a*t*t*t)+(       2*a*t*t)+(-a*t)+0)*px1)+
     ((( (2-a)*t*t*t)+(     (a-3)*t*t)+( 0*t)+1)*px2)+
     ((( (a-2)*t*t*t)+( (3-(2*a))*t*t)+( a*t)+0)*px3)+
     (((     a*t*t*t)+(        -a*t*t)+( 0*t)  )*px4);

    *(vty+i)=
     (((    -a*t*t*t)+(       2*a*t*t)+(-a*t)+0)*py1)+
     ((( (2-a)*t*t*t)+(     (a-3)*t*t)+( 0*t)+1)*py2)+
     ((( (a-2)*t*t*t)+( (3-(2*a))*t*t)+( a*t)+0)*py3)+
     (((     a*t*t*t)+(        -a*t*t)+( 0*t)  )*py4);
  }
/*calculation of point locations on straight segment*/
 j=0;
  for(i=n;i >=n-ns  ;i--)
    {

      *(vtx+i) = 4.0;
```

```
      *(vty+i) = 7.0-(j*seglength);
      j=j+1;
    }
/*Mirror y locations for correct alignment*/
for(i=0; i<=n;i++)
  {
    ydist[i]= 12.5-*(vty+i);
    *(vty+i)  = 12.5+ydist[i];
  }
}
*************************************************/


/*************************************************
  Title:     arc.ff

  Author:    Rich Blank

  Date:      October 5, 1995

  Function:  This program has been written as a
             function file to calculate the inside
             points of primitive #1 used to spray
             paint the letter B.
*************************************************/
/*This function calculates the necessary pts along
  an arc for primitive #1.  The only input is n the
  number of segments along this arc-straight line curve*/
#include <stdio.h>
#include <math.h>
#define pi 3.141592654
#define r 2.0

arc(int n,double *vtx,double *vty)
{
  double theta[n+2],distance,segmentlength,beta;
  int i,j,pts;
  /*calculate approximate arc length*/

  distance=r*.5*pi+1;

  segmentlength=distance/(float)n;
```

```
    beta=((pi/2)/(r*.5*pi))*segmentlength;

    pts=(1/segmentlength)+1;

    for(i=0; i<=(n-pts);i++)
    /*pts = number of segments on straight seg*/
        {
        theta[i]=(pi/2)-(i*beta);
        }
    for(i=0; i<=(n-pts);i++)
        {
       *(vtx+i)= r*cos(theta[i]);
       *(vty+i)= r*sin(theta[i]);
        }
    for(i=0; i<=(n-pts);i++)
        {
        *(vtx+i) = (*(vtx +i)+1);
        *(vty+i) = (*(vty +i)+19);
        }
    j=pts;
    for(i=(n-pts+1); i<=n ;i++)
        {
        j=j-1;
       *(vtx+i) = 3.0;
       *(vty+i) = 18.0+segmentlength*(j);
        }
}
*****************************************************/
```

Appendix B

Detail Drawings