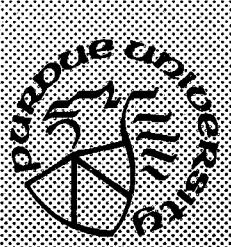
4-1-1990

Programming of Path Specific Robot Operations with Optimal Part Placement

Rick Guptill
Purdue University

Shaheen Ahmad
Purdue University



Programming of Path Specific Robot Operations with Optimal Part Placement

Rick Guptill
Shaheen Ahmad

TR-EE 90-24
April 1990

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

Submitted to IEEE Transactions on Systems, Man Cybernetics, November 1989.

PROGRAMMING OF PATH SPECIFIC ROBOT OPERATIONS WITH OPTIMAL PART PLACEMENT

Rick Guptill and Shaheen Ahmad
Real-time Robot Control Laboratory
School of Electrical Engineering
Purdue University
West Lafayette, IN 47907-0501, USA

Abstract

In this paper we describe a task level programming system for path specific robot operations. We define path specific tasks as those robot tasks in which the path the manipulator end effector has to follow is fixed and is given, such operations may include welding or sealant application. The initial path selection is made through a graphical interface using a pointing device (such as a mouse) to outline the desired path on a CAD model of the workpiece. The final result of the system is the part location, which enables the chosen manipulator to optimally perform the desired task. Optimality is based on maximizing the manipulability of the manipulator performing the task using a function of the jacobian. User defined constraints, joint limit constraints, and collision avoidance constraints are used to guide the optimal location selection. The workable

task is then executed using calls to a "C" language based motion control library outlined in [Guptill88] [Guptill & Stahura 87]. The usefulness of the system described in this paper is indicated by an example of two robotic devices performing a down-hand welding operation.

1. Introduction

The motivation behind developing a task level programming system is to allow the user to write a control program in the context of the task to be completed, independent of the robotic actions needed to carry out the task. In the task level programming system described in this paper, the task domain is path specific robot tasks, such as welding or sealant application. In this system, a user graphically specifies the desired path on a graphic representation of the related part. The programming system then finds the optimal location in the workspace to place the part to accomplish the desired path following in the workable robot workspace which is free of collision and other user imposed constraints. A manipulator level program which uses this information to perform the task is then executed.

The technique presented is general in that it will work with many different types of manipulators and many different types of path specifications. This system requires a inverse kinematics solution and the manipulator jacobian, neither of which need to be in closed form. It also requires a parameterized path specification. A method which is illustrated parameterizes a path by using a "drive transform[†]" between points allowing a series of points to define a path. Note that complete position and orientation of the end effector is required for a point on the path.

[†] See later for definition of the drive transform.

2. Previous Research

Previous work in task level programming for robotic devices has concentrated mainly in the assembly domain, and the subproblems associated with it. Common to each approach is a task level description in some form, and a target manipulator language to execute the task. The differences lie in input modes and how much is assumed about the environment.

Taylor [Taylor76] extracts equations relating object positions from a graph structure of the assembly task. He used these as constraint equations and he used linear programming to find the solution which satisfies all constraints. The use of linear programming allows and optimal solution to be found, given a criteria for optimality. The target manipulator level language used by Taylor was similar to AL [Mujtaba et al. 81].

RAPT [Popplestone et.al.78] starts with a part assembly description in the form of positional keywords, such as "Against" and "fits". Relationships between objects are converted into mathematical geometric constraint equations and then these equations are solved analytically to arrive at the location of the objects in the workspace. This introduction of a mathematical basis extracted from the task description is an important concept as it allows the program to obtain relational information without asking the user. The target manipulator language used by [Poplestone et al. 78] was POP-2.

AUTOPASS [Lieberman & Wesley79] used a set of keywords to describe an assembly task. It is not a natural language but rather a "task level" manipulator language. When an AUTOPASS program is compiled, the compiler asks the user for direction when ambiguities arise. No artificial intelligence reasoning or mathematical formulation was used. The target language generate was an Algol like language.

Inductive learning to robot programming was applied by [Dufay & Latombe84]. Execution traces were generated during a training phase. From these traces, the description of the class of robot task is induced and a manipulator level program is

synthesized. Typically, the system starts with an initial set of knowledge and acquires the information neeeded to complete a task in this initial set through teaching. The ability to represent knowledge is a very important part of this system. The target manipulator language is LM [Latombe & Mazer81].

PARR [Juan87] used an interactive CAD graphical interface to allow the user to assemble a part at a CAD workstation. The motions graphically performed by the user during the assembly are stored and converted into an RCCL program [Hayward & Paul 84] which then runs the PUMA robot. A world model is built from a solid model description. Information such as, the grasping position, the function of the object, and the part tolerances of the parts are stored with the world model. Lozano Pérez has addressed many aspects of task level programming system such as finding a collision free path and grasping etc. [Brady et al. 81].

The main areas usually addressed by an automatic programming system are: the findpath problem or gross motion planning, fine motion planning, and grasping. The problem domain is different in our task level system in that *the path is completely known, no grasping is required and our goal is the location of the part which is usually known* in most previous approaches to task level programming. A numerical optimization technique is devised to solve for the part location which allows collision free task execution while staisfying user specified constraints.

3. Organization

The task level programming system described below consists of five main components. The first component is a graphic modeling system. The robotic devices and the parts used to perform the desired task are modeled using a method which will allow the spatial relationships between them to be analyzed as the task progresses.

The second component in the programming system forms mathematical functions to describe the end effector path. The user specifies the path with respect to a coordinate frame located on the model of the part. An example of this is a seam weld robot has to make on a truck axle.

The third component of our system extracts constraints and produces a set of inequalities which bound the solution set of the position of the workpart. These constraints are combined with additional user constraints to limit the optimal solution space. An objective function is then formed to find the optimal location within this solution space.

The forth component consists of a numerical optimizing algorithm which solves the optimization problem. By virtue of being a numeric algorithm, substitution of other robots requires only a change in function calls, no symbolic manipulation is required. The output of the optimization is the position of the work part. This position is such that the manipulator manipulability is optimized through out the task, while joint limit constraints and user constraints are not violated, and collision free path execution is achieved. Final part of the system converts the numerical solution into a manipulator level language by calling a number of motion control libraries written in 'C' language [Stahura & Guptill 87] [Guptill 88]. Graphical viewing of the task execution is also possible.

4. Modeling Objects in the Workspace

In selecting a set of graphic primatives to use to model the objects in the work environment, we will put emphasis in two areas, ease of modeling, and ease of minimum distance calculation. The basic use of the models is to determine if any collisions occur while performing the task, and in many cases there may be no collisions at all.

Collision avoidance for our purposes is implemented by finding the minimum distance between all objects moving relative to one another in the work environment. This information is used to plan the next part location. Many procedures exist to find the minimum distance between geometric objects in three-dimensional space. Gilbert et al. [Gilbert et. al. 87] use an iterative algorithm to find the minimum distance between polytopes in three dimensional space. Cameron et al. [Cameron & Culley86] used a search procedure to find the minimal translation distance between two convex polyhedra. Khatib [Khatib86] used a time-varying artificial potential field to force the manipulator to move while under the effect of all obstacles' region of influence.

For our purposes, we considered spheres and rectangular parallelopipeds. It is computationally simple to find the distance between two spheres, and because many robotic links are rectangular in nature, it is fairly easy to model the links using multiple overlapping rectangular parallelopipeds. Also, minimum distance determination between rectangular parallelopipeds are relatively straight forward. The fact that the faces are rectangular speeds up this process. [+]

Modeling with spheres, however, presents special problems. The procedure for computing how many and what locations to place the spheres to model an arbitrary object is difficult to determine. Even for a simple object such as a rectangular robot link, the modeling procedure is not well defined. Therefore, we will combine the modeling benefit of rectangular parallelopipeds and the easy distance calculation for spheres. By using the minimum bounding sphere representation for a given rectangular parallelopiped and finding no collisions present, we know no collisions would result if we used

[+]More exact surface and or boundary representation may be employed for precise calculations. Our reasoning behind using sphere's and rectangular parallelopipeds is the ease of computation and system programming. Our main objectives in this paper is to show nonlinear optimization methods can be used to solve robot programming in path specific applications. Notice also that our whole system was written in 'C' language by the authors without using a commerical solid modelling package.

the rectangular parallelopiped. If a collision does result with the minimum bounding sphere we will use the rectangular parallelopiped for the distance finding procedure. In this way, we take advantage of the low computational costs of the sphere, while still maintaining accuracy with respect to graphic modeling.

Meyer [Meyer86] presented an algorithm to find the minimum distance between two rectangular parallelopipeds (RPP), he finds the minimum distance between all edges of the closest face with the other RPP. He then divides the regions around the RPP into four types. From this information, the minimum distance is extracted. We modified Meyer's minimum distance algorithm to suit our graphic representation scheme.

5. Specifying The Task and The Path Graphically

To initiate the task description a graphic representation of the part, which was presumed to have been designed on or at least transferred to a CAD system, is displayed. By allowing multiple views of the part as well as "zoom in" and "zoom out" capability, the user can move a coordinate frame around on the screen using a mouse and select the desired path relative to a part coordinate frame.

Figure 1 shows a part with a possible task represented as a series of coordinate frames which describe the path the manipulator is to follow. The points are numbered in the order of task execution. Extracting surface information from the part, we can find a desired orientation of the tool with respect to the surface of the part if it is

required.

5.2. Mathematical Representation of the Path

Homogeneous transforms are used to describe the position and orientation relationships between objects in the workcell. Using the homogeneous transforms defined in Figure 1, we can form a generic position equation.

$$[\text{robot}]\ [\text{tool}] = [\text{part}]\ [\text{path}] \tag{1}$$

The [robot] transform ($\in \mathbb{R}^{4\times4}$) represents the position and orientation of the last link of the manipulator with respect to its first axis origin. The [tool] ($\in \mathbb{R}^{4\times4}$) transform describes the tool mounted on the robot end effector. The [part] ($\in \mathbb{R}^{4\times4}$) transform is undefined initially because it describes the location we wish to find optimally. The [part] transform can be written in terms of its roll (ϕ), pitch (θ), and yaw (ψ) angles and a translation of (p_x,p_y,p_z) with respect to the world reference frame. We can define a vector $\vec{\Omega} = [p_x, p_y, p_z, \phi, \theta, \psi]^t \in \mathbb{R}^{6\times1}$, and the part transform as:

$$[\text{part}\ (\vec{\Omega})] = \text{Trans}(p_x,\ p_y,\ p_z)\ \text{Rot}(z,\ \phi)\ \text{Rot}(y,\ \theta)\ \text{Rot}(x,\ \psi) \tag{2}$$

We will let the path be a series of N points on the part which specify the desired path similar to those specified in Figure 1.

For each path segment, we can determine a drive transform [Paul & Zhang85], which moves the tool of the manipulator along the desired path in a controlled straight line. By multipling the drive transform with the path defining transforms, we arrive at a set of $(N-1)$ position equations which describe the entire task. The path segments may be straight lines or curves.

The general form of the task position equation is given as

$$[\text{robot}]\,[\text{tool}] = [\text{part}]\,[\text{point(k)}]\,[\text{D(r)}] \qquad k = 1,\ N \tag{3}$$

where the drive transform $[D(r)] = \text{Trans}(x(r),y(r),z(r))\text{Rot}(\vec{U},\phi(r)) \in \mathbb{R}^{4\times4}$, it represents a translational of displacement of $(x(r),y(r),z(r))^t$ followed by a rotation of $\phi(r)$ about axis $\vec{U}$ (see Paul [Paul81] or Craig [Craig86] for further detail). The scalar variable $r\in[0,1]$ and as r goes from $(0\longrightarrow1)$ the end effector moves from [point(k)] to [point(k+1)]. When the robot is referenced with respect to [point(k)] and $(r = 0)$ we have

$$[\text{robot(k,r} = 0)] = [\text{part}][\text{point(k)}][\text{D(0)}][\text{tool}]^{-1} \tag{4}$$

when $(r = 1)$ robot end effector is located at [point(k+1)] and,

$$\begin{aligned}[\text{robot(k,r=1)}] &= [\text{part}][\text{point(k)}][\text{D(1)}][\text{tool}]^{-1} \\ &\equiv [\text{part}][\text{point(k+1)}][\text{tool}]^{-1}\end{aligned} \tag{5}$$

This gives us the transform [D(1)] along path segment (k,k+1) as:

$$[\text{D(1)}] = [\text{point(k)}]^{-1}[\text{point(k+1)}] \tag{6}$$

solving this equation allows us to find $(x(1),y(1),z(1))^t$ and $(\vec{U},\phi(1))$. During the path segment (k,k+1) motion the robot joint angles and end effector position is determined by:

$$[\text{robot(k,r)}] \equiv \mathscr{F}([\text{point(k)}],[\text{point(k+1)}],r) \tag{7}$$

where the function $\mathscr{F}(\,) = [\text{part}][\text{point(k)}]D(r)[\text{tool}]^{-1} \in \mathbb{R}^{4\times4}$, this is also an implicit function of [point(k+1)]. Notice the path the end effector describes is determined by the functions $x(r),y(r),z(r),(\vec{u},\phi(r))$.

6. Developing Task Constraints

We need to convert the (N—1) equations into joint space. This is done by symbolically solving for the vector of joints angles ($\vec{\theta}$) of the given manipulator using the inverse kinematics function $K^{-1}()$:

$$\vec{\theta}(k,\ r) = K^{-1}([robot(k,\ r)]) \ \in \mathbb{R}^{n_J \times 1} \tag{8}$$

where $k = 1,\ldots N-1$ and n_J is the number of joints. The (N—1) equations are functions of r and produce curved paths in joint space.

6.1. The Objective Function

The manipulator jacobian matrix $\mathbf{J} \in \mathbb{R}^{6 \times n_J}$ is the locally linearized transformation matrix which maps incremental changes in the manipulators joint variables to the corresponding incremental changes in Cartesian position and orientation $\vec{dx} \in \mathbb{R}^{6 \times 1}$ and is given by:

$$\vec{dx} = \mathbf{J}\ d\vec{\theta} \tag{9}$$

To enable the manipulator to respond equally well in all Cartesian directions, the jacobian transformation should be as homogeneous as possible. The range of $\vec{dx}$, when $d\vec{\theta}$ takes values satisfying $||d\vec{\theta}||_2 \leq 1$ is an ellipsoid which is called the manipulability ellipsoid [Yoshikawa85]. The closer the ellipsoid to a sphere, the more homogeneous the Jacobian transformation. For non-redundant manipulators, the volume of the ellipsoid is directly proportional to $\left|\det \mathbf{J}\right|$ [Uchiyama et. al.85].

When the manipulator jacobian is singular, the manipulator loses capability of moving in a certain direction and is least dextrous. The determinant of the jacobian is a good measure of working point manipulability [Klein & Blaho 87].

Manipulator singularities are joint angles in which the jacobian is singular, i.e. its determinant is equal to zero. When a manipulator tries to move in cartesian coordinates

through a singular point, excessive joints rates results and control of the manipulator is degraded. These singular points should be avoided. To magnify the effect of being close to a singular point, and avoid joint configurations close to the singularity, one over the absolute value of the determinate of the jacobian can be minimized. This is done along each path and summed over all paths k = 1,...N−1. Thus the manipulability objective function is;

$$f_m(\vec{\Omega}) = \sum_{k=1}^{N-1} \left[\int_0^1 \left| \frac{1}{\det J(k, r)} \right| dr \right] \tag{10}$$

6.2. Joint Motion Limits

The manipulator joint limits provide inequality constraints which bound the solution set in joint space.

$$\theta^-_l \leq \theta_l(k, r) \leq \theta^+_l \quad l = 1,... n_J , \quad k = 1, N-1 \tag{11}$$

where n_J is the number of joints of the manipulator and $\vec{\theta}(k,r) = (\theta_1(k,r),...,\theta_{n_J}(k,r))^t$.

6.3. Collision Avoidance Constraints

To perform collision avoidance, we need to find the minimum distance between all combinations of the two modeling primatives we selected. If we name our shortest distance function **distance**, then given two objects, object1 and object2, we can find the shortest distance between them by passing the object descriptions as arguments, or;

$$\text{min\_dist} = \textbf{distance}\ (\text{object1, object2}) \tag{12}$$

Given two algebraic spheres in space, the minimum distance between them is found by the following formula.

$$\text{min\_dist} = \left\| (\vec{\text{center1}} - \vec{\text{center2}}) \right\|_2 - (\text{radius1} + \text{radius2}) \tag{13}$$

where $\vec{\text{center1}}$ and $\vec{\text{center2}}$ are vectors in cartesian coordinates of the sphere centers with respect to the origin. Notice that $|| \; ||_2$ is the Euclidean norm of a vector.

Given a rectangular parallelopiped and an algebraic sphere we can find the minimum distance between the two by the following formula.

$$\text{min\_dist} = \left\| (\vec{\text{closest point on the RPP}} - \vec{\text{center}}) \right\|_2 - \text{radius} \tag{14}$$

where the vector (closest point on the RPP) is in cartesian coordinates and it is found using the algorithm described in [Guptill88] min_dist returns the minimum distance between the RPP and the sphere.

6.4. User Specified Constraints

There are two types of constraints the user may invoke, inequality constraints, used to bound the solution space of the part to a desired region, and equality constraints which are used to assign one of the part locating variables.

For example, if we wanted to find the optimal table height on which to locate the desired part, we could set the pitch and yaw angles of the part locating vector $\vec{\Omega}$ to zero. Assuming the coordinate frame on the part was aligned such that a stable resting surface was parallel to the x-y plane, this equality constraint would force a solution which allowed the part to be rested on a table.

A set of constraints such as z = 400.0 mm, pitch angle = 0.0, yaw angle = 0.0, -100.0 mm < x < 100.0 mm, and 0.0 < y < 250 mm, would confine the solution space to that of a pre-existing table top, for example.

the terminating point is also feasible.

7.1.1. Joint Limit Penalty Function

In order to impose a penalty when the joint limits of a robotic device are exceeded, it is necessary to construct a function which is smooth, monotonically increasing away from joint limits, and goes to zero as the joints move to inside their limits. We will use a penalty function which is zero if no limits are exceeded and the difference between the joint value and the joint limit squared if the joint limit is exceeded, or

$$f_j(\vec{\Omega}) = \sum_{k=1}^{N-1} \int_0^1 P_j(k, r)dr = \sum_{k=1}^{N-1} \left\{ \int_0^1 \left\{ \sum_{l=1}^{n_J} (\theta_l - \theta_l^{\pm})^2 \delta(l) \right\} dr \right\} \tag{16}$$

Notice that ($\delta(l) = 0$) if the joint limit is not exceeded and ($\delta(l) = 1$) if the joint limit is exceeded. The objective function is now given as:

$$f(\vec{\Omega}) = f_m(\vec{\Omega}) + \beta_{joint} f_j(\vec{\Omega}) \tag{17}$$

where β_{joint} is a monotonically increasing scalar function.

7.1.2. Collision Avoidance Penalty Function

In our system, we modeled every link as a union of rectangular parallelopipeds and algebraic spheres. As each links movement is a function of **r**, $\in[0,1]$ over the current path segment. Notice if a manipulator joint limit is not violated, there may still be collision between two or more links of the manipulator. In such cases, the links of the robot which can collide with itself can be modeled in sections. Those link section always in contact are not checked for collision, only those portions of the link where an undesirable collision could occur are checked. We will include the tool as being part of the manipulator. Therefore, to check for collisions, we need only check if the links

(including the tool) of the manipulator collide with the part and other objects in the workspace. We are also required to check for manipulator self collisions. Hence we have the following distance function;

$$\text{min\_dist}\ (l,\ k,r) = \textbf{distance}\ (\text{link}(l),\ \text{part}) \left\{ l = 1,\ n_J{+}1{+}T\ ,\quad k = 1,\ N-1 \right\} \quad (18)$$

This is useful for checking collision between the objects and the manipulator. Manipulator self collisions can determined from:

$$\text{min\_dist}(j,l,r) = \text{distance}\ (\text{link}(l),\text{link}(j))\ \text{ and }\ j = 1,\ n_J{+}1{+}T \quad (19)$$

$$\text{and}\ \ l \neq j\ ;\ l = 1,\ n_J{+}1{+}T$$

where n_J is the number of joints and T is the number of rectangular parallelopiped which make up the tool.

As long as all links of the manipulator, including the tool, maintain a user selectable safe distance, which we will call SD, from the part during the path execution, then the part location is acceptable as far as collision avoidance. The exception to this safe distance is the end of the tool which is required to follow the path.

The penalty function for collision avoidance is selected as;

$$f_{ca}(\vec{\Omega}) = \sum_{k=1}^{N-1} \sum_{\substack{l=1 \\ j=1}}^{T+n_J+1} \left[\int_0^t (P_{ca}(\vec{\Omega},\ r,\ l,\ k) + P_{cs}(\vec{\Omega},r,j,l,k))\ dr \right] \quad (20)$$

where $P_{ca}()$ monitors collision between the robot linkages and the environment and $P_{cs}()$ monitors the robot self collison. $P_{ca}()$ is given as:

$$P_{ca} = 0 \quad \text{if } (\text{min\_dist}(l,\ k,\ r) \geq\ SD)$$

$$P_{ca} = (\text{min\_dist}(l,\ k,\ r) - SD)^2 \qquad \text{if } (0 < \text{min\_dist}(l,\ k,\ r) < SD)$$

$$P_{ca} = \left(\left\| \vec{\text{center1}} - \vec{\text{center2}} \right\|_2 - \text{radius1} + \text{radius2} \right)^2 + SD^2 \qquad (21)$$

$$\text{if } (\text{min\_dist}(l,\ k,\ r) < 0)$$

where radius1 and radius2 are the distances from the center to the intersection points for both objects respectively. Figure 2 illustrates the collision avoidance penalty function. The penalty function $P_{cs}(\vec{\Omega},r,j,l,k)$ is given by an expression similar to equation (21).

7.1.3. User Constraints Penalty Function

To form penalty functions for the user constraints, we again follow the convention of forming a monotonically increasing function which returns a larger value for the more extreme violation of the constraint. Thus with inequality constraints such as $(y < 100)$ we form the function;

$$\text{if}(y < 100) \qquad f_u = 0 \qquad (22)$$

and if $(y \geq 100)$, $f_u = (y - 100)^2$

Adding the user inequality penalty function onto the objective function yields;

$$f(\vec{\Omega}) = f_m(\vec{\Omega}) + \beta_{joint} f_j(\vec{\Omega}) + \beta_{ca} f_{ca}(\vec{\Omega}) + \beta_{user} f_u(\vec{\Omega}) \qquad (23)$$

where β_{user} β_{joint} and β_{ca} are monotonically increasing scalar functions.

Equality constraints, on the other hand, are used directly in forming the unknown $\vec{\Omega}$ vector of the part. That is, if $(z = 400)$ is a user equality constraint, then the

optimization problem is solved with ($\mathbf{z} = 400$).

7.2. Numerical Methods

Numerical optimization method we employed can be found in literature [Gill et al. 81] [Dennis et al. 83].

Normally, in an unconstrained optimization problem we do not give special consideration to the choice of a starting location for the search. However, the presence of singularities in our manipulability objective function and the limited range of the $K^{-1}()$ function of the robot force us to determine an acceptable initial location.

As the forward and inverse kinematic maps are not one to one, a degeneracy condition exists, as more than one set of joints angles maps to the same homogeneous transform. Control is not degraded by a degenerate condition, but a choice needs to be made as to what set of joint values (i.e. manipulator configurations) are used to position the manipulator. The point at which a given arm configuration changes from one degenerate configuration to another is a sufficient condition for the determinant of the manipulator jacobian to go to zero.

To find the initial location of the part so that the robot starts within a region surrounded by singular points, we perform three steps. First, we place those joints which do not determine a singular condition in the middle of their travel. Second, we place those joints which determine singular points in the middle of their range between the singular location and the joint limit on the same side of the singular point. Finally, using the initial joints in the above positions, we solve for the robot transform [robot] $= K(\vec{\theta}_{\text{initial}}, \text{configuration})$ and then solve for the part location [part] $=$ [path]$^{-1}$[robot][tool]

We have observed that this method of selection also guarantees the joint solution will remain within the allowable joint range. It also ensures that the joint penalty

function is not initially violated. Using this fact we can start the β_{joint} scalar at a high value to keep the robot within its work envelope and prevent the K^{-1} () function from returning an incorrect value. Given this initial position we have the objective function we wish to minimize in the form

$$F(\vec{\Omega}) = f_m(\vec{\Omega}) + \sum_{i=0}^{m} \beta_i P_i(\vec{\Omega}) \tag{24}$$

which [Fiacco & McCormick66] have shown produces convergence in the minimization of a convex function as $\beta_i \rightarrow \infty$. Using the numerical technique derived in [Dennis et.al 83], we can approximate the determinant of the jacobian as a convex function, as long as we stay within boundaries of the singular points. Thus, by dividing the entire feasible region into sections separated by singular points, we are guaranteed to find all optimal locations.

Once a solution has been found, we can check if any of the penalty functions have non-zero values along the path. If they do, we know the solution is not acceptable. At this point we divide the path into shorter paths and repeat the optimizing process, finding the optimal location for each subpath. This process is repeated until each subpath can be executed without violating the penalty functions.

8. Examples of Task Programming

As an example session of the task level programming system, a path is chosen on a box-like part shown in Figure 1. The Cybotech TH8 robot, is chosen to perform the task, and the optimizing process is appropriately initialized and started.

The TH8 has only one singular point, that being when joint angle five is zero. Therefore, a joint vector is chosen which places each joint in the center of its travel with the exception of joint five, which is placed near the center of zero and its limit of

95 degrees, or about 45 degrees. Joint five could also be placed at -45 degrees to obtain the optimal solution on the other side of the singularity. To determine the initial position and orientation vector of the part, the TH8 tool tip is placed in the middle point of the path and the transform equation is solved for the location of the part. This initial position is shown in Figure 3.

Figure 4 shows the optimal location as found by the optimizing algorithm. As shown, the final position is fairly close to the initial position. This is not surprising as joint five was chosen to be between a singular position and a joint limit, and the determinate of the jacobian of the TH8 is proportional to the sine of joint five.

Introducing the user constraints pitch angle $= 0$ and yaw angle $= 0$ we can force the optimization algorithm to find the optimal location such that the part can be placed on a flat stationary table. The initial position is the same as shown in Figure 3, and the final optimal location is shown in Figure 5.

If we extend the part shown in Figure 6 to provide a collision with the TH8 in its optimal location, we can see how a new optimal location is chosen to maintain a safe distance from the robot in Figure 7. The graphs in Figures 8 through 11 show the convergence toward an optimal solution with the safe distance reached on iteration five. Once this happens the optimal placement algorithm moves the part back slightly to optimize the objective function taking into account the collision avoidance boundary. The z variable does not change from its initial value because it does not cause a joint out of limit and joint two is not part of the determinant of the TH8 jacobian. Shown in Figure 7 is the part moving from its initial position to its optimal location which maximizes the manipulability of the TH8 while avoiding collisions.

8.1. Coordination of Multiple Robotic Devices for Down-Hand Welding

In down hand welding a TH8-robot manipulates a welding gun along a specified path on a part. The part is mounted on a two axis orienting table (a robotic motion device) which simultaneously moves with the robot to make the weld point accessible. Given the geometry of the orienting table, the robot and the part and the specified path we wish to determine the optimal location of the table with respect to the robot for successful task execution. During the welding operation the surface normal of the weld point is constrained to remain anti-parallel to the gravity direction. This is to prevent the weld plasma from flowing away. This constraint requires the robot to weld in a "down-hand" manner, i.e. the weld gun pointed approximately in the direction of gravity.

As an example task we wish to weld a "spiral" on a rectangular parallelopiped. The part is mounted on the table is as shown in Figure 12.

The constraints relating to the "down hand" welding for the table is obtained from the transform equation, (the transforms are defined in Figure 12):

$$[\text{table\_base}][\text{py\_table}][\text{part}][\text{weld\_path(h)}] = [\text{down\_hand}] \tag{25}$$

where $h \in [0,1]$ and at it is the parameterization for the weld path, ($h = 0$) at the start of weld, ($h = 1$) at the end of weld.

The rotational part of [down_hand] is defined * as;

$$\text{Rotational part of [down\_hand]} = \begin{bmatrix} C_\psi & -S_\psi & 0 \\ S_\psi & C_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{26}$$

This ensures the z axis of the surface normal at the weld point is anti-parallel to the gravity direction.**

**Kinematics of down-hand welding with redundant robots can be found in [Ahmad & Luo 89].

If we set [b] = [table_base], [DH] = [down_hand] and [PT] = [path][weld–path(h)]. Then we notice the (3,3) element of [py–table] = $[b]^{-1}[DH][PT]^{-1}$ is zero.

$$b_{a_x}(C_\psi p_{n_z} - S_\psi p_{o_z}) + b_{a_y}(S_\psi p_{n_z} + C_\psi p_{o_z}) + b_{a_z} p_{a_z} = 0 \tag{27}$$

where b_{ax} = (1,3) element of [b], b_{a_y} = (2,3) element of [b] and b_{a_z} = (3,3) element of [b]; p_{n_z} = (3,1) element of [PT], p_{o_z} = (3,2) element of [PT] and P_{a_z} = (3,3) element of [PT]. We can solve the above triangular equation for angle ψ as:

$$\psi = \tan^{-1}\left(\frac{k_1}{k_2}\right) + \tan^{-1}\left(\frac{-\sqrt{k_1^2 + k_2^2 - k_3^2}}{k_3}\right) \tag{28}$$

where $k_1 = b_{a_x} p_{n_z} + b_{a_y} p_{o_z}$; $k_2 = b_{a_y} p_{n_z} - b_{a_x} p_{o_z}$; $k_3 = -b_{a_z} p_{a_z}$.

Once the angle ψ is found, we can solve for [p_y–table] for a given value of h. We can then use the table inverse kinematic function to solve for the table joint angles. A penalty function which places a high cost on violating the joint limits, similar to the one used for the robot, is constructed to retain the table within the joint limits. The value of this penalty function is multiplied by a scaling factor and added to the objective function. This satisfies the second constraint by keeping the table joint angles within their limits. The objective function that is minimized in this application is the sum of the following; robot joint limit penalty, the table joint limit penalty, the penalty function measuring the collision between the robot and the table and the workpiece, and the robot manipulability measure (as defined in equation (10)). As before the optimal location is found by minimizing this objective function without activating any of the penalty functions.

Note $C_\psi = \cos\psi$ and $S_\psi = \sin\psi$ where ψ is some angle about the Z axis of the world reference.

The synchronization constraint, that of keeping the robot in a down hand position during the path execution, and maintaining robot and table trajectories synchronized is achieved by solving for the robot angles once the table angles have been determined. The following position equation describes the kinematic relationship of the robot, the table, and the part.

$$[\text{robot\_base}][\text{TH8}][\text{weld\_torch}] = [\text{table\_base}][\text{py\_table}][\text{weld\_path(h)}] \quad (29)$$

By solving for the [TH8] transform as,

$$[\text{TH8}] = [\text{robot\_base}]^{-1}[\text{table\_base}][\text{py\_table}][\text{weld\_path(h)}][\text{weld\_torch}]^{-1}$$

the TH8 joint angles are determined next using the K^{-1} () function.

To chose an initial feasible solution, the robot joint vector is chosen to align the z axis of the weld torch with gravity. The initial table position and orientation is found by solving equation (29) for the [table_base] transform.

The initial position is shown in Figure 12. The final optimized location is shown in Figure 13. Figure 14 shows the optimal location when the user constraints pitch and yaw angles are set to zero.

9. Manipulator Level Execution

A subroutine user_task () [Guptill & Stahura86], which runs on the same graphic workstation used to model the part and define the path, uses the numerical information generated by the optimal placement algorithm. The user_task () is written to read in the desired path and optimal part location, and then perform the task using position equation (1) by making subroutine calls to the motion control library described in [Guptill & Stahura 86]. The motion control library supports multiple robotic devices and

graphics and kinematic models for the robotic devices exists on the system device list.

For the down hand-welding task, we use the same equations in the user_task () for execution of the task as we used in the optimizing process, that is equation (29). The same user_task () which runs in simulate mode can then be used to execute the task for a real workcell environment.

10. Summary

This paper described a programming system for path specific robot tasks. The input is a graphic representation of the desired path in part coordinates. The output is the location to place the part in the workcell of the chosen robot. The location is optimal in that it maximizes the manipulability of the robot while avoiding collisions and keeping the robot within its joint limits. The desired path can then be executed using a software package "user_task ()" running on the graphic workstation, or on an actual robot controller. The placement algorithm was implemented on a VAX 11/780 computer, the graphic simulator was run on an APPOLLO workstation. The robot task execution was graphically demonstrated on the APPOLLO workstation, files were generated which could have led to task execution on Purdue's RCCL* library or on Purdue's multirobot programming package [Guptill & Stahura 87].

References

[Ahmad & Luo 89] Ahmad S., Luo S., "Coordinated Motion Control of Multiple Robotic Devices for Welding and Redundancy Coordination through Constrained Optimization in Cartesian Space." IEEE Transaction of Robotics and Automation, August 1989, Vol.

*RCCL is a 'C' library package which enables robots to be programmed and controlled from a "C" language and UNIX operating system environment on a VAX 11/780.

5, No. 4, pp. 409-417.

[Brady et al. 82] Brady, M., Hollerbach, J. M., Johnson, T. M., Lozano-Pérez, T., Mason, M. T., "Robot Motion, Planning and Control." MIT Press, Cambridge, MA, 1982.

[Cameron & Culley 86] Cameron, S. A., and Culley, R. K., (1986). "Determining the minimum translational distance between two convex polyhedra," IEEE Conference on Robotics and Automation, San Francisco, pp: 591-596.

[Dennis & Schnabel 83] Dennis, J. E. Jr., and Schnabel, R. B., (1983). *Numerical Methods For Unconstrained Optimization and Nonlinear Equations,* Prentice-Hall, Inc., New Jersey.

[Dufay & Latombe 84] Dufay, B. and Latombe, J.C., (1984). "An approach to Automatic Robot Programming Based on Inductive Learning", Robotics Research, The First International Symposium, Edited by M. Brady and R. Paul, The MIT Press.

[Gilbert et al. 87] Gilbert, E. G., Johnson, D. W., and Keerthi, S. S., (1987). "A Fast Procedure For Computing The Distance Between Complex Objects in Three Space," IEEE Conference on Robotics and Automation, North Carolina, pp. 1883-1889.

[Gill & Murray 74] Gill, P. E., and Murray, W., (1974). *Numerical Methods For Constrained Optimization,* Academic Press, London.

[Guptill & Stahura 87] Guptill, R. L., & Stahura, P. A., (1987). "Multiple Robotic Devices: Position Specification and Coordination", IEEE Conference on Robotics and

Automation, North Carolina.

[Guptill 88] Guptill R., "Multiprocessor Based Control for Multiple Robots: Software and Kinematic Programming Methodology," Ph.D. Thesis, August 1988, Purdue University.

[Juan 87] Juan, Juan, (1987). "Planning Automatic Assembly for Robots (PARR)," PhD. Thesis, Purdue University.

[Khatib 86] Khatib, O., (1986). "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," The International Journal of Robotics Research, Vol. 5, No. 1, pp.90-98.

[Klein & Blaho 87] Klein, C. A., and Blaho, B. E., (1987). "Dexterity Measures for the Design and Control of Kinematically Redundant Manipulators,", The International Journal of Robotics Research, Vol. 6, No. 2, pp.73-83.

[Latombe & Mazer 81] Latombe, J. C., & Mazer, E., (1981). "LM: a high-level programming language for controlling manipulators," 11th International Symposium on Industrial Robots (ISIR), Tokyo.

[Liberman & Wesley 79] Lieberman, L. I., and Wesley, M. A., (1979). "AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly.", IBM Journal of Research and Development, No. 34.

[Meyer 86] Meyer, W., (1986). "Distance Between Boxes: Applications to Collision Detection and Clipping,", Proceedings 1986 IEEE International Conference on Robotics and

Automation, April 7-10, San Fransisco, pp. 597-602.

[Mujtaba et al. 81] Mujtaba S., Goldman R., "AL Users Manual," AIM-344, Stanford University, Dec. 1981.

[Paul & Zhang 85] Paul, R. P., & Zhang, H., (1985). "Robot Motion Trajectory Specification and Generation", Robotics Research: The Second International Symposium, MIT-Press Series in Artificial Intelligence.

[Popplestone et al. 78] Popplestone, R. J., Ambler, A. P., and Bellos, I., (1978). "RAPT: A Language for describing Assemblies", The Industrial Robot.

[Taylor 76] Taylor, R.H., (1976). "Synthesis of manipulator control programs from task-level specifications," Memo AIM 228, AI Lab.

[Uchiyama et al. 85] Uchiyama, M., Shimizu, K., and Hakomori, K. (1985). "Performance evaluation of manipulators using the Jacobian and its application to trajectory planning," Robotics Research: The Second International Symposium, eds. H. Hanafusa and H. Innoue, pp. 447-454, Cambridge: MIT Press.

[Yoshikawa 85] Yoshikawa, T., (1985). "Manipulability of robotic mechanisms," The International Journal of Robotics Research, Vol. 4, No. 2, pp. 3-9.

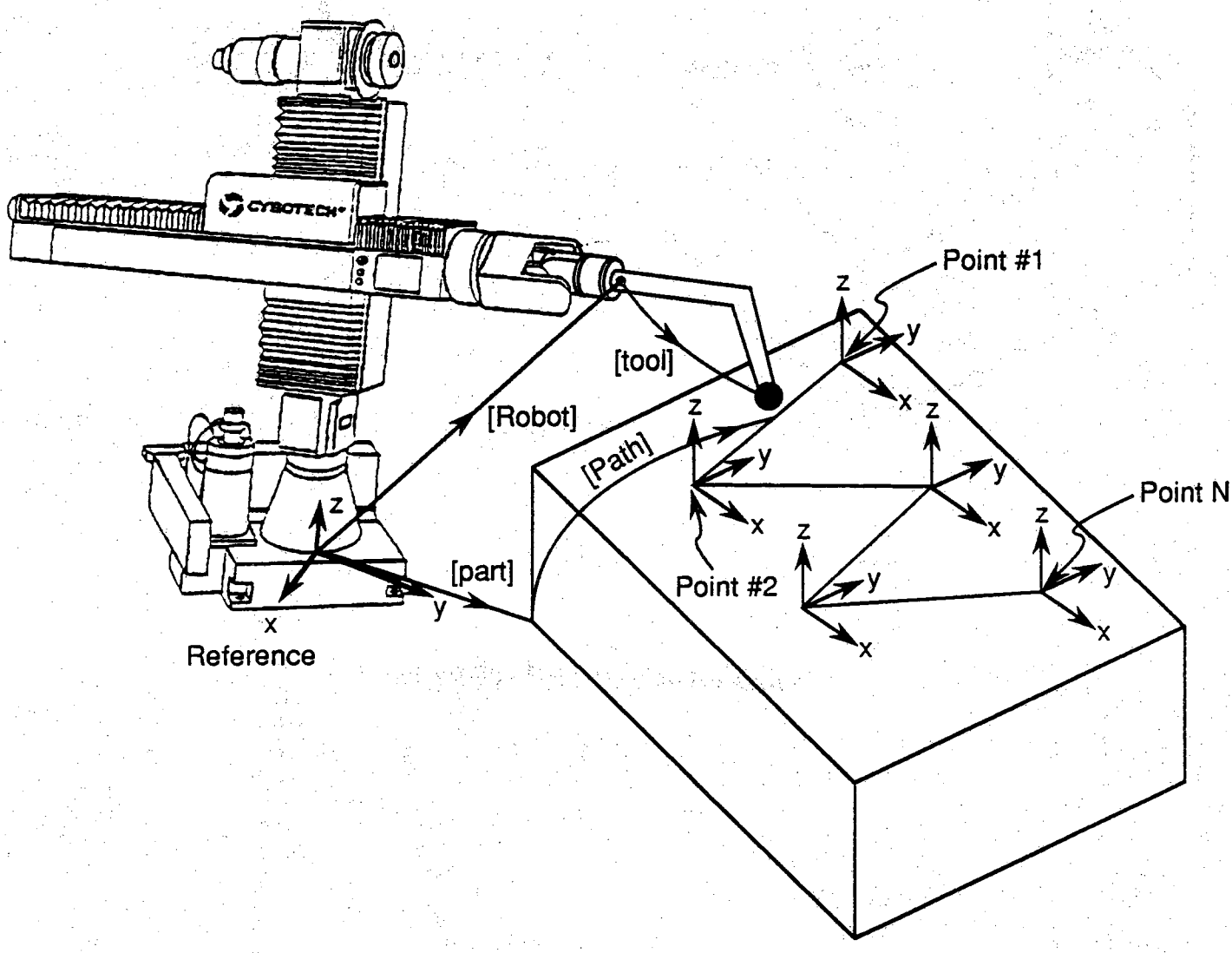


Figure 1
Homogeneous transforms and desired path

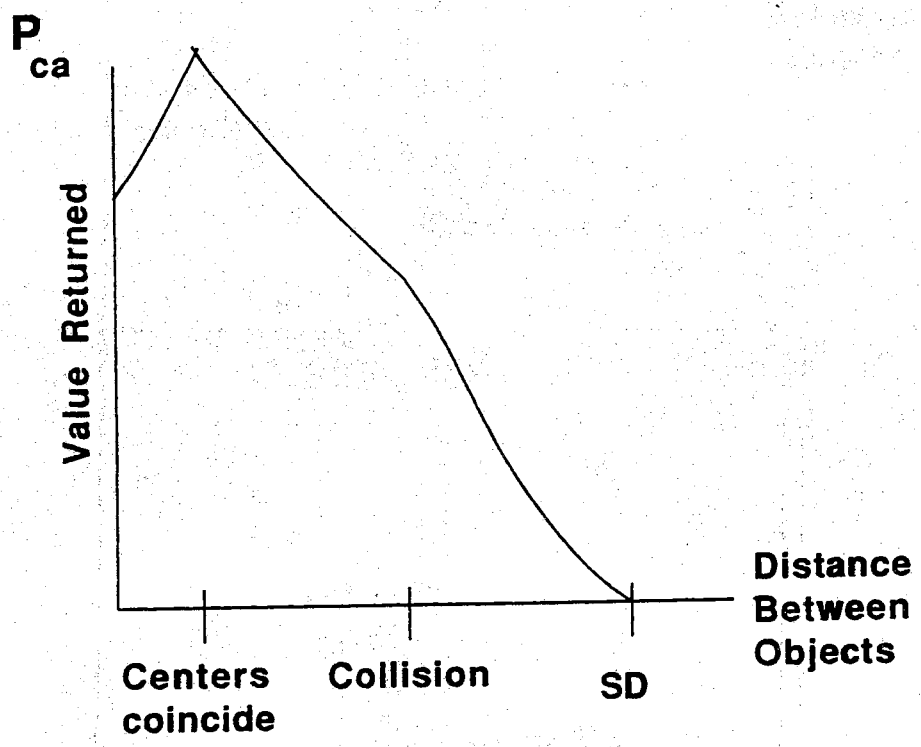


Figure 2
Collision avoidance penalty function

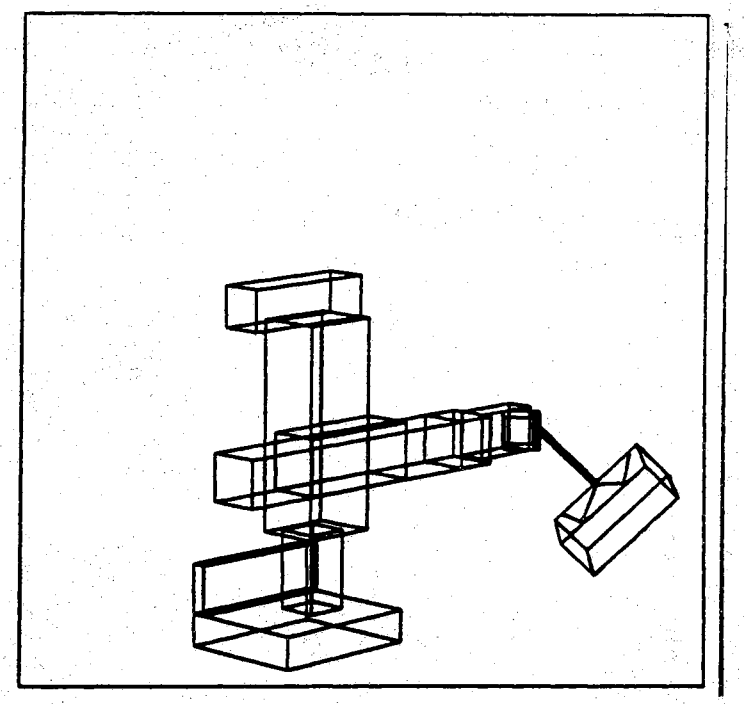


Figure 3
Initial position of TH8

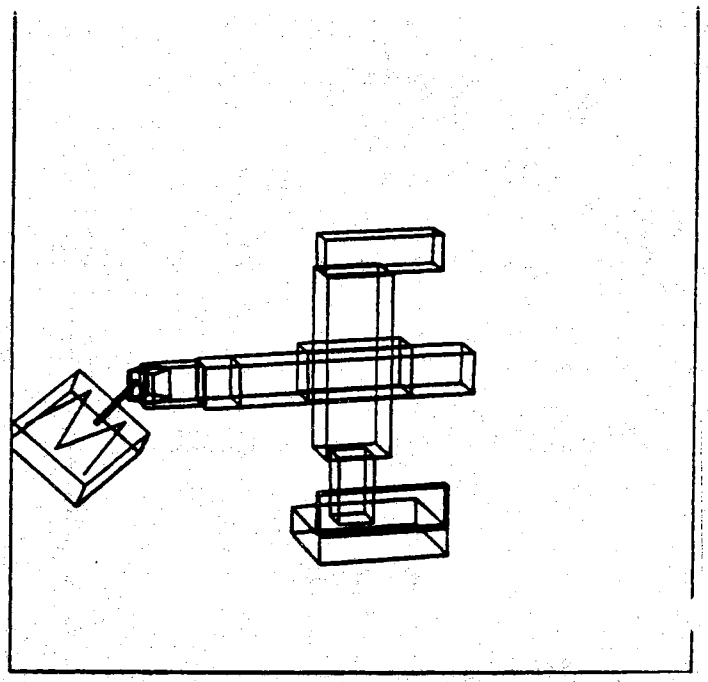


Figure 4
Optimal position of TH8

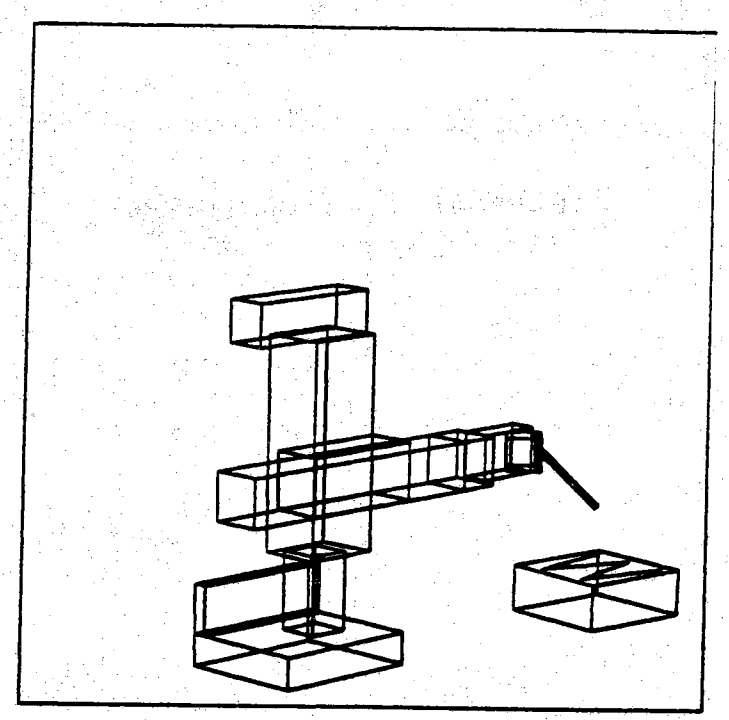


Figure 5
Optimal location with user constraints
of part being placed on a table

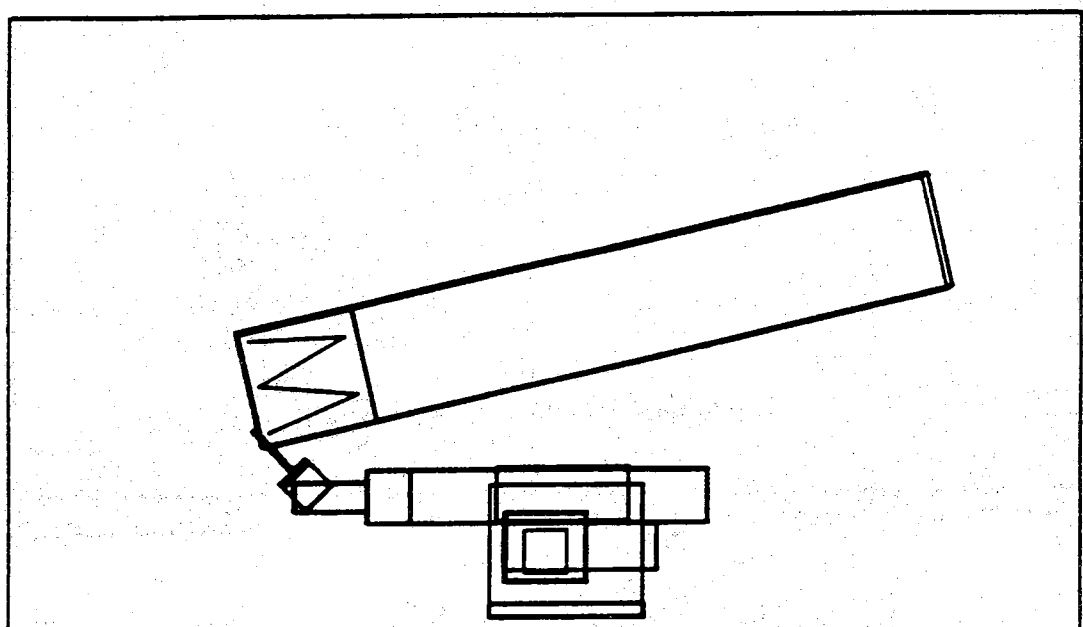


Figure 6
Initial position with extended part size

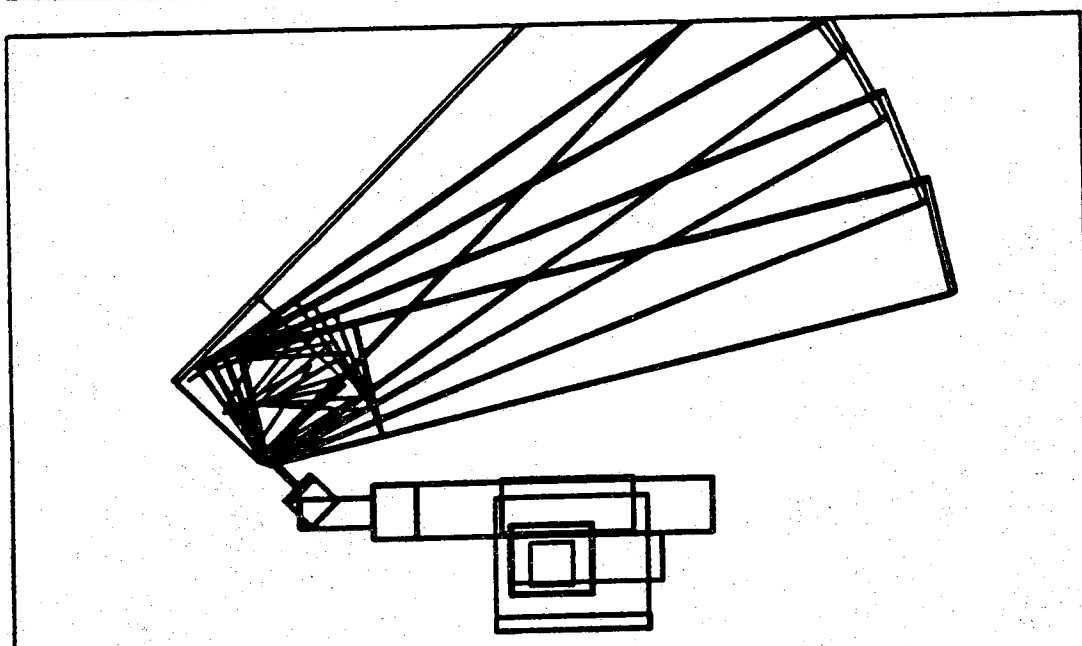


Figure 7
Motion of the part to find optimal solution

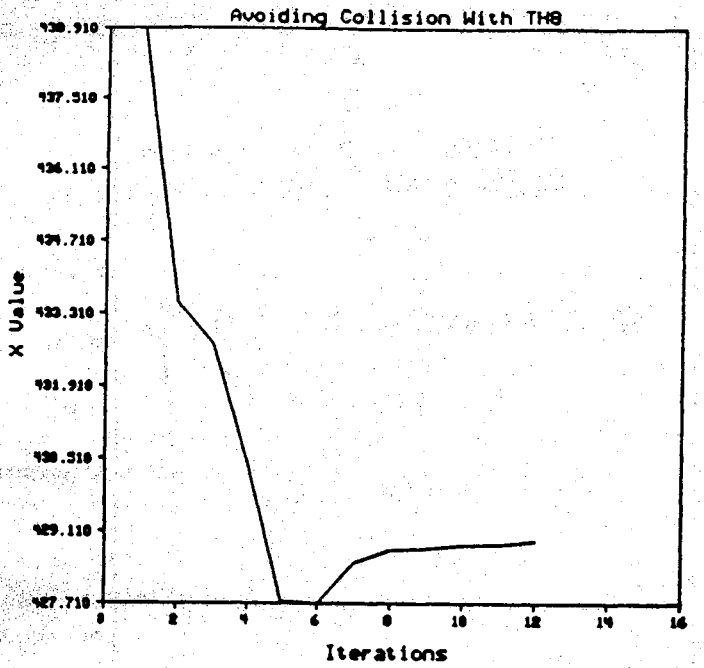


Figure 8

Convergence of the X-value of the part

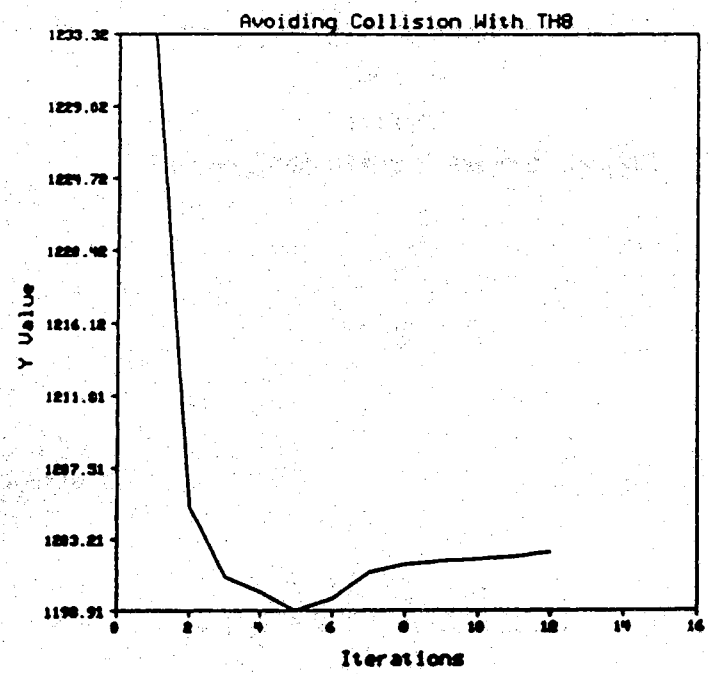


Figure 9

Convergence of the Y-value of the part

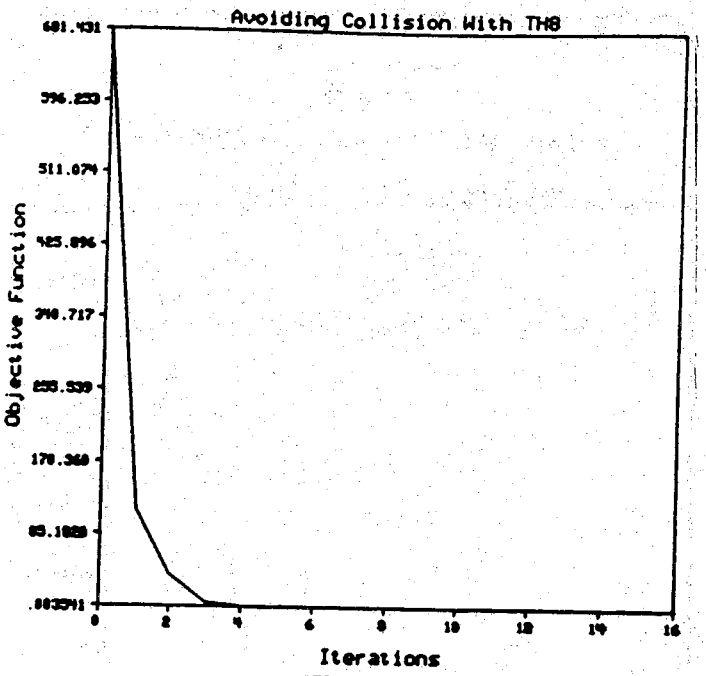


Figure 10

Convergence of the objective function

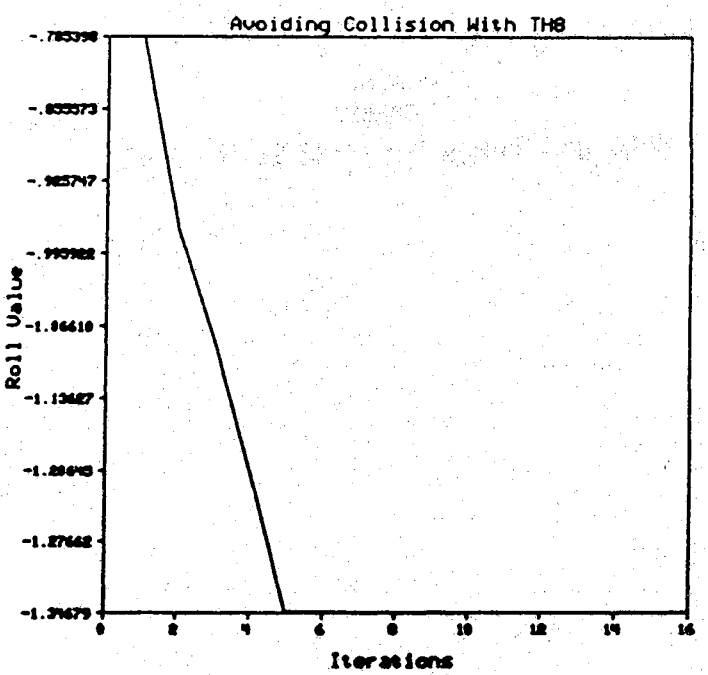


Figure 11

Convergence of the roll angle for the table

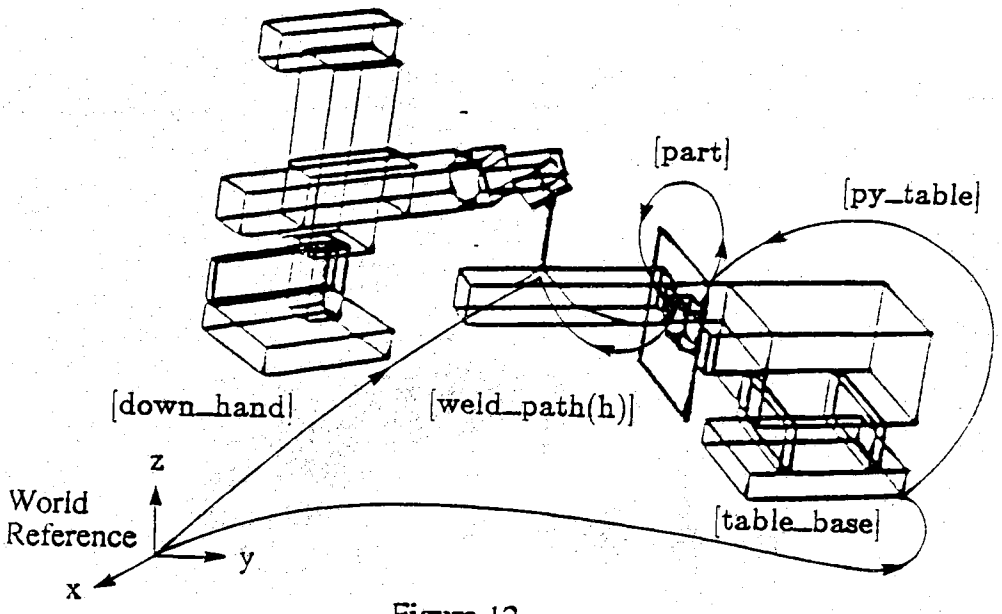


Figure 12
The down hand welding system initial location

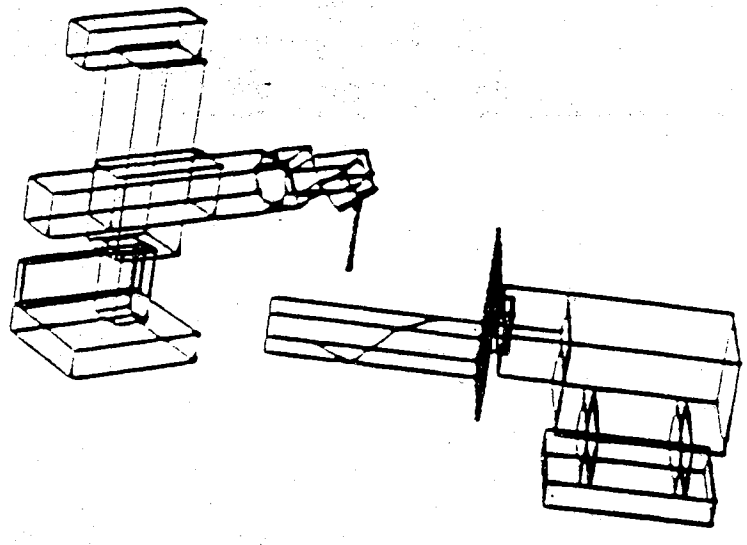


Figure 13
The down hand welding system optimal location
with pitch = 0 and yaw = 0

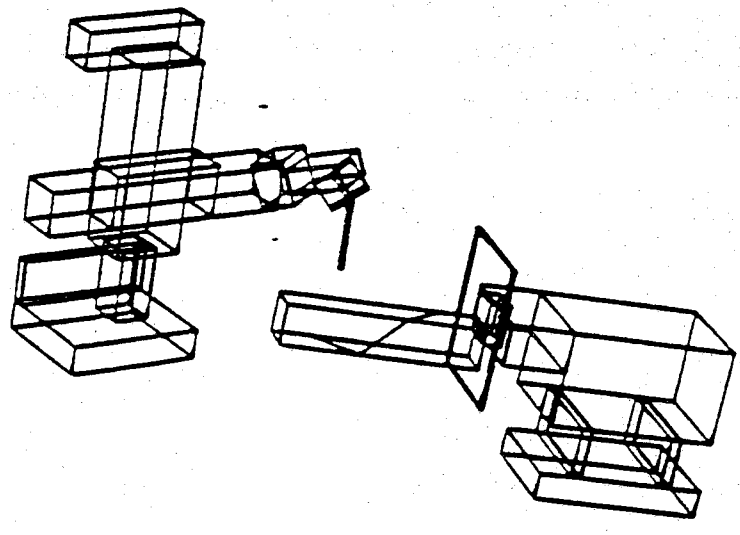


Figure 14
The down hand welding system optimal location