Robotica (1997) volume 15, pp 63–73. © 1997 Cambridge University Press

SMAR: A Robot Modeling and Simulation System

S. Zeghloul, B. Blanchard, M. Ayrault

Université de Poitiers Laboratoire de Mécanique des Solides, U.R.A. C.N.R.S. 861, Bât SP2MI, Bd 3 Téléport 2, BP 179, 86960 FUTUROSCOPE Cedex (France)

SUMMARY

In this paper we present the SMAR CAD-robotics system (Système de Modélisation et d'Animation de Robots), which we developed at the University of Poitiers. This system allows its user to deal with a great number of robotics problems through the use of a graphic simulator. We will discuss the different parts which form the SMAR system. This includes the following:

—The modeler which allows the user to build a database, describing the robot and its environment. The database generated by the system is composed of the geometric description of the objects and the kinematics description of the environment.

—The simulator and the coordinates reverser, which simulate the robot's movements.

—The collision detection algorithms used to verify task accomplishment.

—A calculation algorithm in order to find optimal placement, which determines the relative position robot/task, allowing the robot to efficiently execute the assigned task.

—The collision free-path planning algorithm allowing the system to generate trajectories in a cluttered environment.

An example dealing with a complex robotized cell will also be presented in order to demonstrate the capabilities of the system.

KEYWORDS: CAD-Robotics systems; Collision detection; Optimal placement; Trajectories planning.

1. INTRODUCTION

The use of robots in industrial manufacturing systems has greatly increased. Because of the increased use of this tool, a great number of problems have surfaced.

These different problems can be effectively treated or possibly resolved by the use of CAD-Robotics systems. These systems offer powerful graphic capabilities, allowing for simple resolutions for problems like, robot selection, robot placement in its work station, and off-line programming of tasks.

Several robot simulation and programming systems have been developed. These systems can be classed into two categories: First, there are the commercial systems designed to effectively resolve most problems related to industrial robot utilization. These may include commercial systems such as: ROBCAD, SILMA, Unigraph-Place, ACT, ... The second category concerns the systems developed in university research labs that are used as a support for validating more advanced robotics algorithms.[1–7] Some of the algorithms originating from university research are used on industrial-type systems.

In this paper we will describe the CAD-Robotics system (SMAR) that we have developed in our research laboratory. Our aim is to make available a tool which supports and validates algorithms developed in the domain of CAD-Robotics. This tool is also used for educational robotics training. Many SMAR algorithms are used for industrial robotics applications.

In the sections that follow, we will describe the different parts of SMAR (CAD-Robotics system). Firstly, we will present a modeler which allows the user to build the database describing the robotics cell. Secondly, we will present the simulator. It has a movement manager module, a system for updating the database, as well as the capability to inverse coordinates, allowing the user to specify movements using Cartesian coordinates (task coordinates). The two systems that allow for collision detection are presented in section 4. We will briefly describe the first method based on an algorithm similar to those used in Boolean operations. It permits collision detection between both convex and non-convex solids. We will also present the second method based on a fast algorithm for distance calculation between convex objects. We will present an algorithm that calculates the optimal placement of robots in section 5. The suggested approach determines the relative position between the robot and the task in order to minimize an objective function integrating multiple criterion. This algorithm allows the treatment of different problems such as: robot assessibility to the task, and improving robot performances during the execution of the task. It is worth noting that this is one of our system's unique characteristics, since to our knowledge the approach we propose is not available in any other CAD-Robotics system. In section 6, we will present the method for collision-free path planning, which we have developed and introduced in the system. It is similar to local methods, but its formulation makes its possible to avoid dead-lock phenomenons which are caused by convergence towards local minimums, encountered in classical local approaches. Finally, in the last section, we will present the modeling of a welding cell for metal framed sections, and the simulation of welding tasks. This complex cell, with 38 degrees of freedom, is a well adapted example which allows us to validate the whole of SMAR modules in a real case simulation.

https://doi.org/10.1017/S0263574797000088 Published online by Cambridge University Press

The system is developed on SILICONGRAPHICS and uses OSF Motif and Open Inventor Libraries. It is important to mention that all the system's modules have a user-friendly interface based on the use of menus and 3D graphic representation. It offers the grahic capability of most of the CAD systems.

2. MODELER

The purpose of the CAD-Robotics modeler is to enable the user to create a database necessary to describe the robot and its environment. This database is next used by the simulator to reproduce the behavior of the cell. Therefore, the modeler must allow the user to define the overall data that the simulator algorithm requires. Two types of descriptions are used in the SMAR system to define a cell in the database. The first allows the user to define the geometry of the cell's different objects. The second description is used to define the links between these different environment objects.

2.1 *Geometric modeling*

Geometric modelization defines an approximation of the entire environment bodies. In SMAR we have used a polyhedral approximation like most of the CAD solids modelers.[8,9] To modelize an enrironment the SMAR system has different methods for defining objects: by primitives, swept volumes, or Boolean operations.

—Modelization by primitives allows one to define an object from a set of the standard forms available in the system, such as: cube, cylinder, etc . . . . . (Figure 1).

—The swept volume method defines an object by a Cartesian product of a facet and a given trajectory (Figure 1).

—Modelization using Boolean operations or CSG (Constructive Solid Geometry), allows one to define a complex object by a set of Boolean operations ($\cup, \cap, -$) applied on simple or complex shapes.

Note that the first two methods are fairly easy to implement; however, Boolean operations are difficult to implement and demand an intensive computational load.

Figure 1 shows the system's different modeling possibilities, as well as an environment model.

2.2. Links modeling

The second description available in the database of the SMAR system describes the links between the cell's objects. In the SMAR system, the cell is described as a branched chain composed of different links. In order to define this chain, three different links are used: rigid type links, temporary type links and joint type links.

The rigid type links do not allow any movement. They are used to define objects of the same link. The temporary type links are used to momentarily connect two objects. For example, to carry out grasp and place procedures. Finally, joint type links allow movement between two links. In this case the user introduces the joint axis, the joint type (prismatic or revolute), as well as the following parameters: joint, velocity and acceleration limits.

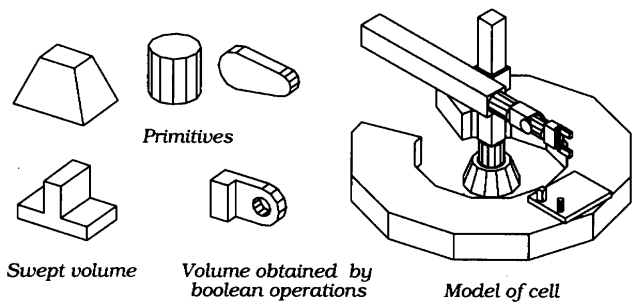

Fig. 1. SMAR modeling capabilities.

3. SIMULATOR

The simulator of the SMAR system uses the database generated by the modeler to simulate the user's requests (tasks) on a graphics screen. In order to do this, the simulator is equipped with a set of routines that assure the following: automatic updating of the database, cell animation, an algorithm to solve inverse kinematics problems for tasks coordinates control, called the reverser, trajectory management and a certain number of post-processors for robot off-line programming.

3.1. Movement management

In the database, the state of the cell is described as a set of configuration vectors q, which represent the configurations of different articulated chains of the cell. These vectors are introduced either by the user, in the case of direct control, or obtained by the reverser, for Cartesian coordinates control (task coordinates control).

3.1.1. Movements in direct control. In this control mode, the user introduces a configuration vector corresponding to the targeted configuration. The simulator automatically generates the trajectory of the manipulator using the algorithm's control codes. Two types of velocity profiles are available, they correspond to the constant acceleration movements whose constraints are found by using the following:

$$\left|\frac{dv}{dt}\right| \leq \Gamma_{\max} \tag{1}$$

$$|v| \leq V_{\max} \tag{2}$$

or to the constant "Jerk" which has the same constraints and the following:

$$\left|\frac{d^2v}{dt^2}\right| \leq c \tag{3}$$

where v is the joint velocity, remember that this data is acquired at the time of modelization. These two types of velocity profiles are described in Figure 2. The trajectory travel time can be introduced by the user or automatically calculated by the simulator. In the automatic mode, the trajectory is calculated in such a way that the travel time is minimal and the axes are synchronized.

v_i and v_f denote respectively the initial and the final velocity. The trajectories calculated in this way are

https://doi.org/10.1017/S0263574797000088 Published online by Cambridge University Press

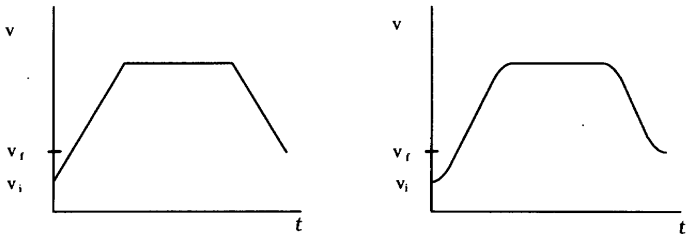

Fig. 2. Velocity profiles.

represented on a graphics screen. Of course, the number of images displayed for the intermediate positions depends on the computer's capabilities. For each position the system generates the transformation matrix T_{OJ} describing the location of each object *j,* relative to the absolute reference in order to update the database. The matrix T_{OJ} is given by:

$$T_{OJ} = \left[\begin{array}{c|c} R_{OJ} & P_{OJ} \\ \hline 000 & 1 \end{array}\right] \tag{4}$$

where R_{OJ} and P_{OJ} are respectively the matrices of rotation and the translation of the object. When your working with an object of a kinematic chain, T_{OJ} is obtained by:

$$T_{OJ} = \prod_{i=l}^{J} T_{i-1,i} \tag{5}$$

where the object $i-1$ denotes the father of the object i in the hierarchy.

3.1.2. Movements in Cartesian coordinates. In this mode the movement is specified by Cartesian coordinates (task coordinates). The targeted pose is given by the position and orientation of end effector, in which there are three Cartesian coordinates of the end effector and the orientation of the end effector specified by ''Euler angles'' or ''Roll-pitch-Yaw angles''.

This data, as well as the structure of the robot are used by the coordinates reverser to calculate the configuration of the robot. Next, the movement between the starting position and the final position is generated in the same way as that of the direct control mode (cf. 3.1.1).

In SMAR we have installed an explicit coordinates reverser (closed-formed inverse solutions). It treats most existing robot structures. The architectures that the system accepts are the robots with six degrees of freedom, composed of a wrist having concurrent axes and a non-redundant arm (regional structure) with three degrees of freedom. The different arm classes that the system processes are represented in Figure 3. The advantage of the reverser algorithm is to give all possible configurations for a given end-effector pose as shown in Figure 4.

3.2. Task management and off-line programming

Two post-processors were developed for loading the tasks in the robot's control devices. These post-processors correspond to the two robots available in our research labs (an industrial robot TH8, ACMA-Renault and an educational robot ERICC, Barras-Provence). For task programming, the user has, along with the graphic abilities and simulation functions, task editing functions and collision detection available in order to check accessibility (cf. section 4). Task loading on the two controls is done by using serial ports.

4. COLLISION DETECTION

During movement simulation, two algorithms are used to detect collisions between the objects of the environment. The first is based on the calculation of intersection and it deals with convex as well as non-convex objects. The second is based on distance calculation, and it only deals with convex objects.

4.1. General algorithm for collision detection

In principle, this algorithm uses an approach similar to that of the Boolean operations algorithm. Its purpose is to check the intersection of any two objects. In order to do this it calculates the intersections between the facets of each object. The process is stopped when an intersection exists between the two facets. Therefore to reduce the calculation time, each object is described not only by a polyhedral description, but also by an encompassing sphere and a parallelepiped. So, the collision test is first of all, applied to the encompassing primitives before considering the polyhedral description. The same principle is applied to the levels of the facets at

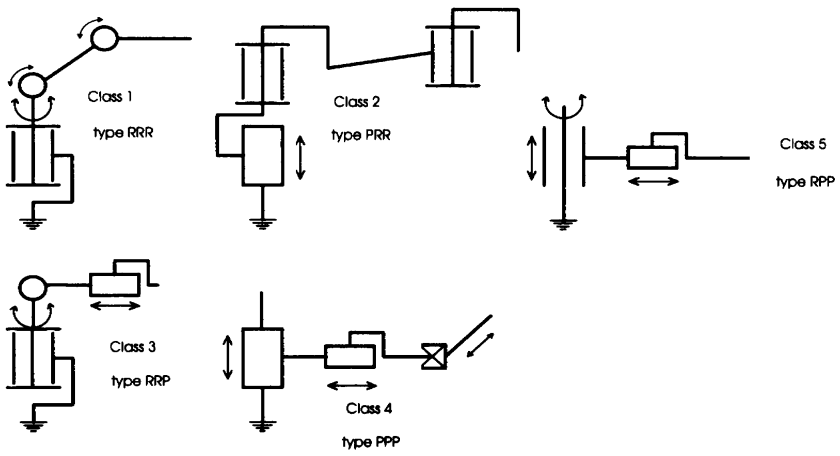

Fig. 3. Regional structures processed by the reverser.

https://doi.org/10.1017/S0263574797000088 Published online by Cambridge University Press

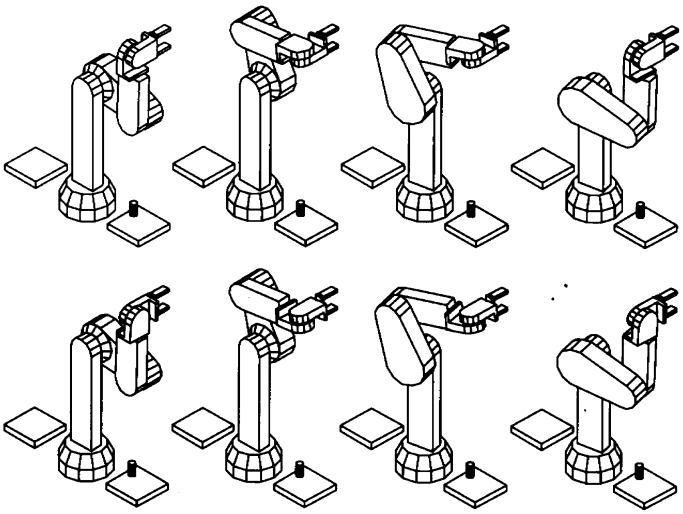

Fig. 4. Eight inverse solutions of 6R robot.

the time of the intersection calculation between an object and a facet. The algorithm considers first of all, the intersection between the facet and the primitives (sphere, parallelepiped) encompassing the object.

This description method makes it possible to avoid useless calculation and as a result reduces the time spent in the calculation process. As we have shown above, this algorithm works with all types of objects, but it does not measure the proximity of objects which is made possible by using distance calculation algorithm.

4.2. Distance algorithm

We will now describe the distance calculation algorithm in the section which follows:

Let us state the problem of distance computation. Let O_1 and O_2 be two convex polyhedrons defined respectively by k_1 and k_2 half-spaces. We can designate an object O_i as the collection of points x which satisfy the following:

$$\langle x, n_{ij}\rangle \leq d_{ij} \qquad j = 1, 2, \ldots, k_i; \qquad i = 1, 2; \tag{6}$$

Where $\langle\ \rangle$ designates an inner product, x is any point in Euclidean three-space, n_{ij} is a unit vector normal to plane j, and d_{ij} is the perpendicular distance from plane j to the reference origin. We obtain:

$$O_i = \bigcap_{j=1}^{k_i} \{x \mid \langle x, n_{ij}\rangle \leq d_{ij}\} \tag{7}$$

So, the distance calculation problem can be stated as the following minimization problem: Find x_1 in O_1 and x_2 in O_2 such that:

$$f(x_1, x_2) = \|x_1 - x_2\|^2/2 \tag{8}$$

is minimal ($\|\ \|$ representing the Euclidean norm) subject to the linear constraints:

$$g_{ij}(x_i) = \langle x_i, n_{ij}\rangle - d_{ij} \leq 0 \tag{9}$$

In order to solve the above problem (minimal distance calculation), the computational scheme used is based on a direct approach to minimizing the distance function which produces a succession of optimal search directions along the boundary of the objects. This algorithm combines the gradient projection method[10] and an additional optimal search direction when the gradient projection method leads to a zigzagging phenomenon.

4.2.1. Iterative scheme. Let us define briefly the iterative scheme of the method to obtain the minimal distance between two convex polyhedrons, (for more details concerning this section refer to reference 11).

For the iteration $k+1$ the points $x_{i,k+1}$ on each convex set O_i are found by using:

$$x_{i,k+1} = x_{i,k} + \beta_i S_{i,k} \tag{10}$$

where $x_{i,k}$ are the starting points for the $(k+1)$-*th* iteration, β_i are the step lengths and $S_{i,k}$ are the search directions. In the general method of Rosen, the search directions are obtained by the following equations:

$$\begin{aligned} S_{i,k} &= -P_{i,k}\nabla f_i / \|P_{i,k}\nabla f_i\| \\ P_{i,k} &= I - N_{i,k}(N_{i,k}^T N_{i,k})^{-1} N_{i,k}^T \end{aligned} \tag{11}$$

where $N_{i,k}$ is the matrix whose columns are given by the gradients of the active constraints at $g_{ij}(x_{i,k}) = 0$ and ∇f_i is the gradient of the objective function at $x_{i,k}$. The matrix $P_{i,k}$ is called the projection matrix.

However, for the problem that interests us, we do not have to compute this projection matrix because $P_{i,k}$ is given simply by considering the geometrical properties of a polyhedron (see section 4.2.4). These search directions are obtained by projecting the gradient of the objective function (equation 8) onto the active constraints at $x_{i,k}$ (i.e. $g_{ij}(x_{i,k}) = 0$).

The iterative process is stopped when the Kuhn–Tucker Conditions (**KTC**), are satisfied for both objects. Consequently, the search directions are null vectors. This means that the closest points (minimal distance) between the two convex polyhedrons have been reached. The minimum found will be global because the problem is convex. The **KTC** are given on each object by:

$$\begin{aligned} -\nabla f_i = \sum_{j=1}^{m_i} \alpha_{ij}\nabla g_{ij} \quad \text{with} \quad \alpha_{ij} \geq 0 \\ \text{and} \quad -\nabla f_1 = -\frac{\partial f}{\partial x_{1,k}} = x_{2,k} - x_{1,k} \\ -\nabla f_2 = -\frac{\partial f}{\partial x_{2,k}} = x_{1,k} - x_{2,k} \end{aligned} \tag{12}$$

where m_i is the number of active constraints (i.e. constraints for which $g_{ij}(x_{i,k}) = 0$) and ∇f_i, ∇g_{ij} are respectively the gradients of f and g_{ij}.

Geometrically, the **KTC** mean that the gradient lies within the region defined by the normals, as shown in Figure 5.

So, we have to check the **KTC** for each object according to the following relations:

$$-\nabla f_1 = x_{2,k} - x_{1,k} = \sum_{j=1}^{m_1} \alpha_{1j} n_{1j} \quad \text{with} \quad \alpha_{1j} \geq 0 \tag{13}$$

$$-\nabla f_2 = x_{1,k} - x_{2,k} = \sum_{j=1}^{m_2} \alpha_{2j} n_{2j} \quad \text{with} \quad \alpha_{2j} \geq 0 \tag{14}$$

https://doi.org/10.1017/S0263574797000088 Published online by Cambridge University Press

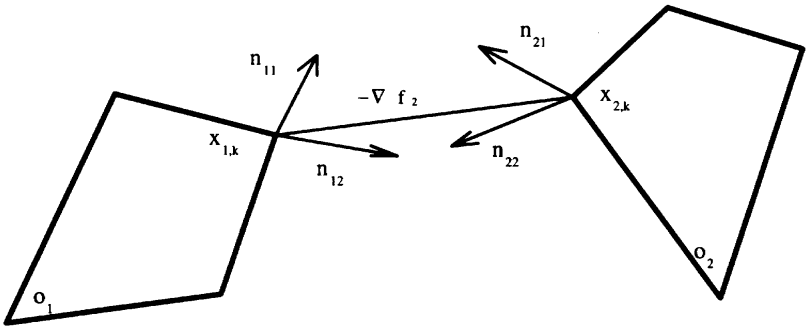

Fig. 5. Kuhn-Tucker conditions at a global minimum.

As previously stated, the search direction is a null vector when the KTC are satisfied. However, when the computation of the minimum distance between two solids is carried out, the **KTC** are frequently satisfied on only one object.[12] On this object, the method gives a zero vector as search direction, which leads to a zigzagging phenomenon as shown in Figure 6. For these critical situations, in order to avoid the zigzagging phenomenon, it is necessary to compute a new search direction on the object, where the KTC are satisfied, which should be different from the zero vector.

Let O_2 be the object satisfying the **KTC**, we have shown[11] that the optimal search direction $S_{2,k}$ on O_1 is given by the projection of $S_{1,k}$ on the gradients of the active constraints at the point $x_{2,k}$ ($S_{1,k}$ being the non-null search direction vector on object O_1).

4.2.2. Search directions and KTC evaluation. In order to define the search directions, three cases must be distinguished according to the location of the points on the object O_i. If $x_{i,k}$ lies on a plane (one active constraint), the search direction $S_{i,k}$ is obtained by projecting the gradient on a plane (Figure 7). Let n_{ij} be the plane normal, $S_{i,k}$ is given by:

$$S_{i,k} = -\nabla f_i - \langle -\nabla f_i, n_{ij} \rangle n_{ij} \tag{15}$$

If $S_{i,k} = 0$, then the KTC are satisfied which means that the gradient ∇f_i is parallel to n_{ij}.

A similar approach is used to check and evaluate the **KTC** and to obtain the search directions, when the point $x_{i,k}$ lies on an edge or on a vertex (for more details please refer to reference 11).

4.2.3. Step length. As previously stated, the process is stopped if the *KTC* are satisfied for both objects. If this is not the case, using $x_{2,k}$, $x_{1,k}$ and the search directions $(S_{1,k}, S_{2,k})$ determined in the last section, then we compute the step lengths (β_1, β_2) in such a way that $(x_{1,k+1}, x_{2,k+1})$ gives the minimal distance between the two following segments (Sg_1, Sg_2):

$$\begin{aligned} Sg_1 &= [x_{1,k},\ x_{1,k} + \beta_{1u} S_{1,k}] \\ Sg_2 &= [x_{2,k},\ x_{2,k} + \beta_{2u} S_{2,k}] \end{aligned} \tag{16}$$

where β_{iu} is the limit value of β_i for which the segment lies in the solid O_i.

4.2.4. Initial points. Finally, in order to start the iterative process, a pair of starting points $(x_{1,0}, x_{2,0})$ must be given. In general, the algorithm uses the centroïd of each object as a starting pair. Then, $x_{1,1}$ and $x_{2,1}$ are obtained, as shown in Figure 8, by computing the intersection between the segment $[x_{1,0}, x_{2,0}]$ and each object.

When computing the distance for small discrete step movements of objects, the starting points considered are those obtained in the previous step. However, the starting points must be transformed as the objects move in space. In this case the algorithm needs only one or two iterations for the convergence.

In general, the number of iterations required for the convergence is in the range of 2 to 5. The results of numerical experiments[11] show the practical efficiency of the proposed algorithm which is linear in the total number of half-spaces which define the boundary of the solids.

5. OPTIMAL PLACEMENT

The optimal placement algorith of the SMAR system deals with the following problem: It finds the relative position robot/task, for any given robot and its

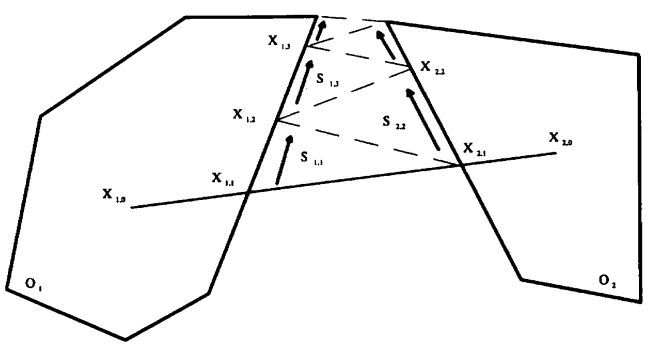

Fig. 6. Illustration of zigzagging in two dimensions.

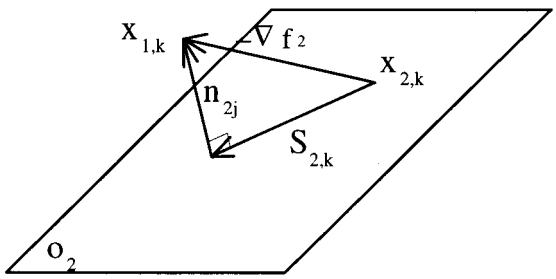

Fig. 7. Search direction with one active constraint.

https://doi.org/10.1017/S0263574797000088 Published online by Cambridge University Press

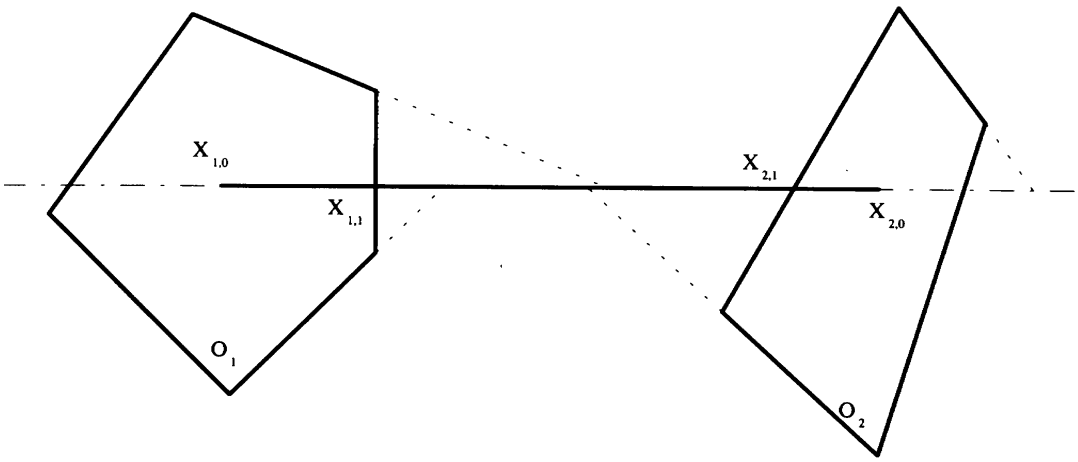

Fig. 8. Starting points.

environment, in order to complete the assigned task. The SMAR system uses an optimization approach in order to resolve placement problems. The task considered is defined by a set of positions and orientations of the end-effector referred to an absolute coordinate system $\sum o$. Let $\sum$ c be the coordinates system attached to the manipulator base. The placement problem can be stated as follows:

Find the position and orientation of the robot base defined by $\sum c$ which minimizes an objective function f under a task constraint.

Figure 9 describes the coordinate systems and as well as the different task points *Si.* In order to complete the formulation we have to define the problem's constraints and the objective function.

5.1. Problem constraints

In general, the constraints limit the evolution of the Design Vector, which is defined in this case by the position and orientation of the robot's reference $\sum c$. In our approach the design vector is defined by the position of $\sum c/\sum o$ in the Cartesian coordinates. It is written as r_{ox}, r_{oy} and r_{oz} and the orientation of $\sum c/\sum o$ given by the pitch, roll and yaw angles written as λ, μ and ν.

A distinction is made between the explicit constraints and the implicit constraints. The explicit constraints are directly applied to the components of the design vector and are expressed by (equation 17), while the implicit constraints are applied to variables that depend implicitly on the design vector.

$$\begin{aligned} \lambda_1 \le \lambda \le \lambda_u \qquad & r_{ox_1} \le r_{ox} \le r_{ox_u} \\ \mu_1 \le \mu \le \mu_u \qquad & r_{oy_1} \le r_{oy} \le r_{oy_u} \\ \nu_1 \le \nu \le \nu_u \qquad & r_{0z_1} \le r_{oz} \le r_{oz_u} \end{aligned} \tag{17}$$

where the subscripts l and u denote respectively lower and upper bounds, which can be determined from the available space at the workstation. These constraints introduce some primary boundaries, however they do not guarantee the avoidance of interference between the manipulator and obstacles nor joint limits.

We must consider then the two sets of implicit constraints in order to avoid interference and the joint limits by using:

$$(d_{hk})_i \ge d_{hkl} \qquad \begin{matrix} h = 1, 2, \ldots, n_e \\ i = 1, 2, \ldots, m \\ k = 1, 2, \ldots, n \end{matrix} \tag{18}$$

where $(d_{hk})_i$ is the shortest distance between the $h-th$ object of the environment and the $k-th$ link of the manipulator for the $i-th$ task point, with the number of objects in the environment being defined as n_e. The distance $(d_{hk})_i$ for any task point is bounded by a minimum admissible value d_{hkl}. The computation of the distance $(d_{hk})_i$ is carried out using the algorithm described in section 4.2. The parameters m and n are respectively the number of task points and the number of joint variables of the manipulator.

The manipulator joint constraints can be expressed as:

$$q_{k_1} \le q_{ki} \le q_{k_u} \qquad \begin{matrix} k = 1, 2, \ldots, n \\ i = 1, 2, \ldots, m \end{matrix} \tag{19}$$

where the q_{ki} is the $k-th$ component of the joint vector q_i of the manipulator's configuration corresponding to the $i-th$ path point, and the subscripts l and u denote respectively lower and upper bounds.

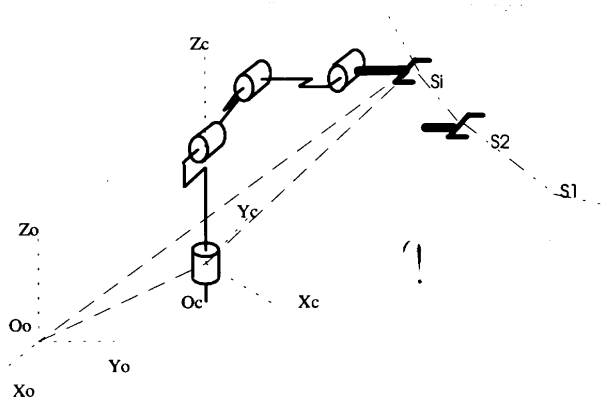

Fig. 9. Task and coordinate systems $\sum o$ and $\sum c$.

5.2. Objective functions

In the SMAR system many functions can be used in the placement problem. Some of these functions consider

https://doi.org/10.1017/S0263574797000088 Published online by Cambridge University Press

only one criterion to be optimized and others integrate multiple criterion.

5.2.1. Travel time. As an objective function, one can use the travel time of the trajectory defined by the points S_i. In this case the optimization process calculates the placement that leads to the minimal trajectory travel time, respecting the problem constraints.

Travel time is obtained using one of the two velocity profiles defined in section (3.1.1.).

5.2.2. Joint limit avoidance. This function, defined by Equation 20 determines the robot's placement, allowing the robot to execute the task with the configurations that keep the joint of robots as far as possible from their limits. The objective of the optimization process is to minimize the following function, proposed by Pamanes.[13]

$$\phi = \bar{k} + k_{\bar{\sigma}} \tag{20}$$

where $\bar{k}$ is the mean, and $k_{\bar{\sigma}}$ the standard deviation of the $m \times n$ values of the ratio k_{ij} defined by:

$$k_{ij} = \left[\frac{\Delta q_{ij}}{\Delta q_{i\,\max}}\right]^2 \qquad \begin{matrix} j = 1, \ldots, m \\ i = 1, \ldots, n \end{matrix} \tag{21}$$

where:

Δq_{ij}: is the deviation of the $i-th$ joint variable, with respect to the center of its permissible range, at the $j-th$ task point,

$\Delta q_{i\,\max}$: is the maximum deviation permissible of the $i-th$ joint variable.

This criterion, here termed "joint limits avoidance", can be used to measure the ease of a manipulator to access to a certain task.

5.2.3. Multi-criterion function. With the SMAR system it is also possible to determine the placement that minimizes many performance criterion at the same time. In this situation we assign a criterion to each task point S_i. In this approach we have used certain performance criterion available in the literature such as:

* The condition number of the Jacobean transpose matrix J^T of the manipulator which is given by:

$$C(A) = \|J^T\| \, \|J^{-T}\| \tag{22}$$

where $\| \ \|$ denotes any norm. This index is used in order to minimize the error propagation from input torques or velocities to output forces or velocities.

* The manipulability measure introduced by Yokhikawa[14] is quantified by the index w:

$$w = \sqrt{\det\,(JJ^T)} \tag{23}$$

The manipulability is a measure of the ease to arbitrarily change the position and orientation of the end effector. Thus, its maximization would be appreciated in task zones where relatively large deviations in the prescribed motion of the end effector are likely.

* The compatibility index introduced by Chiu[15] allows one to optimize the magnitude and accuracy of force and velocity of the manipulator on preferred displacement direction u. The compatibility index is based on the transmission ratios of force α and of velocity β in the considered direction. These ratios are given by:

$$\begin{aligned} \alpha &= [u^T(JJ^T)u]^{-1/2} \\ \beta &= [u^T(JJ^T)^{-1}u]^{-1/2} \end{aligned} \tag{24}$$

where u is a unit vector in the direction of interest. If the magnitude of force or velocity is considered, α and β must be maximized. If the accuracy of force or velocity is considered, then α^{-1} or β^{-1} must be maximized.

In the multi-criterion approach we choose p points ($p \leq m$), from the m points defining the task, to which kinematic criteria are assigned, and we express as k_j the index of the kinematic criterion assigned to some point S_i. The objective of the optimization is to locate the manipulator in such a way that the p indices k_j are kept as great as possible. However, since generally the orders of values of these indices are different, they can't be compared effectively in an optimization process. We may avoid this difficulty by introducing the following normalized index C_j:

$$C_j = k_j / k_j^* \tag{25}$$

where the normalization factor kj^* is the maximum value that can be reached by k_j. Thus, C_j is bounded as follows:

$$0 \leq C_j \leq 1$$

Next in order to complete the formulation, we may define a small typical element of the set of p indices C_j as:

$$C = \bar{C} - C\sigma \tag{26}$$

where $\bar{C}$ and $C\sigma$ are respectively the mean and the standard deviation of the set. It is evident that the optimal set, i.e. the one with all its elements as great as possible, must have a C maximum. Next, if the optimization problem is to be solved as one of minimization, we can define the objective function as:

$$f = C\sigma - \bar{C} \tag{27}$$

5.3. Optimization algorithm

The minimization of the objective functions (cf. sections 5.2.1, 5.2.2 and 5.2.3) subject to constraints (cf. section 5.1) can be carried out using some classical methods of non-linear programming. We have applied the Box method,[16] which does not require the derivatives of the objective function in order to solve the placement problem.

The same approach is applied when computing the normalization factors k_j^* (cf. section 5.2.3). In fact, the normalization factor k_j^* is the maximum value which can be obtained by k_j. This maximum is determined by solving an optimization problem.

The optimal placement algorithm has been applied to solve several problems such as: robot accessibility to the task, improving robot performance during the execution of the task and robotized cell design. In all cases the

https://doi.org/10.1017/S0263574797000088 Published online by Cambridge University Press

results obtained lead to the improvement of the objective functions. For more details concerning this section please refer to[7,13,17] where several examples are included that show the results obtained with this method.

6. PLANNING TRAJECTORIES WITHOUT COLLISION

Planning trajectories means finding trajectory without collision that brings the manipulator from an initial configuration to the final configuration q_f. In this paragraph we will present a local method introduced in SMAR that we propose for the planning of trajectories that avoid collision with obstacles in a workspace.[18]

6.1. Classic method

Classically, the local methods are based on an iterative procedure where the variation of the configuration vector is found by using the following formula:

$$q_i = q_{i-1} + \Delta q_i \tag{28}$$

where Δq_i is the variation of the configuration vector at the iteration i. It is calculated by using the potential field method[19] or according to the constraints method.[20] With the constraints method, the movement Δq is the one that minimizes the following function:

$$\tfrac{1}{2}\,\|\dot{\tau}(q) - \underline{\tau}(q)\|^2 \quad \text{with} \quad \begin{array}{l} \underline{\dot{\tau}}(q) = \dfrac{q - q_f}{\|q - q_f\|} \\ \tau(q) = q - q_f \end{array} \tag{29}$$

under constraints

$$\dot{d} \geq -\xi \frac{d - d_s}{d_i - d_s}\, dt \quad \text{for} \quad d < d_i \tag{30}$$

where:

- q and q_f are respectively the current configuration vector and the final configuration vector,
- ξ represents a coefficient in order to adapt the convergence,
- d is the distance between two convex solids,
- d_i is the influence distance from which a constraint becomes active,
- d_s is the security distance that must be respected,
- $\dot{d}$ is the time derivative of the distance which defines the anti-collision constraint. It is shown that this variation can be calculated in relation to the configuration parameter variation by using $\dot{d} = (J^t n \mid dq)$, J being the Jacobean matrix calculated at the point of minimum distance, n is the segment that connects these points and $\mid$ means the inner product.

If no constraints are activate then the trajectory is a straight line in the configuration space.

6.2. Proposed method

The problem with the methods mentioned above is in converging towards a local minimum due to the fact that in the vicinity of local minimums there are the same geometrical active constraints between two iterations (cf. distance calculation). Our method is designed to avoid these constraints (local minimum) through an analysis of the local geometry of the environment.

In order to avoid this local minimum, it would be sufficient to determine a movement that would tend to change from one set of constraints to another set. According to this idea, we will define the movement Δq as the following:

- if $d > d_i$ none of the constraints are active and the trajectory is a straight line. In this case, the displacement is given by:

$$\Delta Q = \frac{q - q_f}{\max_{j=1,\ldots,nddl} \|q_j - q_f\|} \tag{31}$$

- if $d < d_i$, such will be the case when one or several constraints are active. The displacement is obtained by using two displacements dq_{but} and dq_{lock}, according to the following relation:

$$\Delta Q = \sum_{k=1}^{N} (\alpha^k dq^k_{but} + (1 - \alpha^k) dq^k_{lock}) \tag{32}$$

with

- N as the number of active constraints,
- dq^k_{but} is the displacement calculated according to Equation 31,
- dq^k_{lock} is the displacement that allows us to change from one set of constraints to another,
- $\alpha^k = \dfrac{d^k - d_s}{d_i d_s}$ is a weighing coefficient of the two calculated displacements in relation to the distance d^k for the constraint k. If the distance d^k is close to the security distance, we give priority to the movement which allows us to change the constraint, whereas if d^k is close to the distance of influence, we give priority to the movement which would bring us closer to the final configuration.

dq^k_{lock} is obtained by using an inverse differential model by: $dq^k_{lock} = J^t(JJ^t)^{-1} \cdot V^k_i$. In order to change active constraints, we must define the movement V_i that will allow us to move towards another geometrical constraint (face, edge, vertex). Considering Figure 10, the local methods are classically in a blocked state. The minimum distance between the objects O_1 and O_2 is obtained at points x_1 and x_2 of planes F_{11} and F_{12}. This brings about the result that the constraint remains active on planes F_{11} and F_{12}. What this means is that the minimal distance which makes the constraint active is always obtained for two points belonging to facets F_{11} and F_{12}. It is this situation which leads to the blocked condition mentioned above. To avoid this problem, a displacement that leads to the modification of a series of constraints must be defined. Consequently, in order to find the displacement according to those points which give the minimal distance, they must be placed on those facets other than F_{11} and F_{12}. This is achieved by calculating distances between x_2 and the different adjacent facets F_{22}, F_{32}, F_{42}, F_{52}, on the obstacle. For each facet, there is a direction of displacement that is defined by $\vec{V}_1$, $\vec{V}_2$, $\vec{V}_3$, $\vec{V}_4$ which allows us to change the constraints on the obstacle. To switch the constraints on the object O_1 we must move along the direction $\vec{\bar{V}}_i$. This direction is obtained by projecting $-\vec{V}_i$ on the facet F_{11}. There are m displacement possibilities in order to

https://doi.org/10.1017/S0263574797000088 Published online by Cambridge University Press

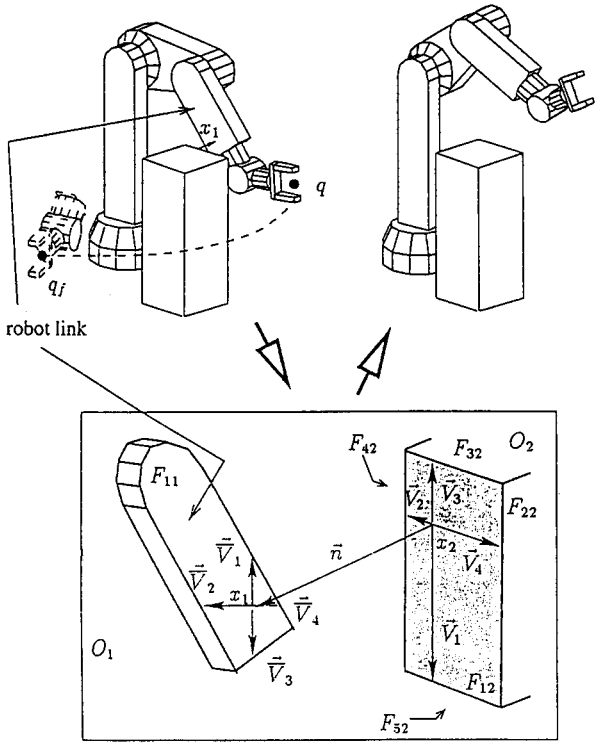

Fig. 10. Second displacement choice.

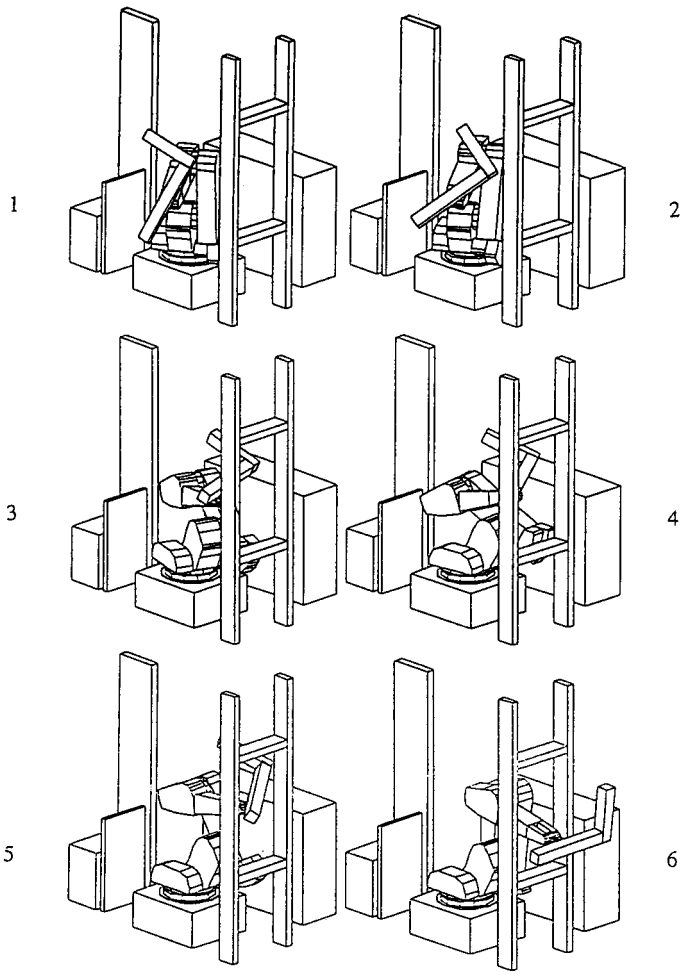

Fig. 11. Example for Commercy Cy2006 manipulator.

avoid the obstacle. We chose among these possibilities the one that permits switching rapidly from one set of constraints to another: this displacement is established along the following direction: $\min_i = 1, \ldots, m \, \|\vec{V}_i\| + \|\vec{\vec{V}}_i\|$. In the example shown in Figure 10, the method allows us to obtain a variation of the configuration vector such that the displacements along $\vec{V}_3$ lead to the possibility of avoiding the obstacle from the top. These methods have been briefly treated here, however, there are several particular cases which need to be further developed in our study. More explanations will be provided in reference 18. Thus, this obtained displacement ΔQ gives direction along which the robot must move in order to avoid the dead-lock situation. However, in order to avoid collisions, this displacement is used as a sub-goal to be reached in the optimization method under anti-collision constraints.

6.3. Example

This example (Figure 11) deals with a case where the classic local methods do not provide a solution. The manipulator must move an object of large dimension in a cluttered environment. The algorithm allows us to determine the trajectory from the initial configuration $q = \{\cdot \cdot \cdot\}$ towards the final configuration $q_f = \{\cdot \cdot \cdot\}$ in 124 iterations. The execution time necessary to determine the solution for this example using SILICONGRAPHICS is 70s. It corresponds to a nearly-exact real-time solution for these work stations.

7. EXAMPLE

Our application deals with the modelization of a robotized welding cell in a metal framed construction, as well as the simulation of the movements of the different welding tasks.

Taking the complexity of the cell and of the tasks into account, most of the functions available on the SMAR system have been used. In fact, the cell is composed of different kinematic chains which need the multi-robot management possibilities of the system. Therefore, this study is an interesting example in order to validate, in concrete terms, all the functions of the system. On the other hand, as shown in Figure 12, the environment is

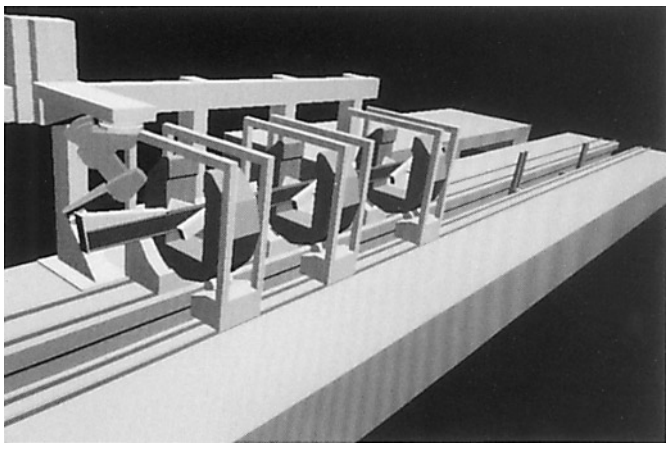

Fig. 12. Complex robotized welding cell.

https://doi.org/10.1017/S0263574797000088 Published online by Cambridge University Press

heavily cluttered, and this calls for collision detection and path planning tools, to ensure reasonable functioning time.

The cell studied must weld metal framed sections. It is made up of a manipulator of the sections needed to be soldered, and a welding suspended robot. The manipulator accomplishes all the transfers of framed sections, from their waiting position to the welding post, and vice versa. It has two transfer trolleys, and four gripping devices. The trolleys move in a gutter. They carry the metal sections between the loading-unloading zone and the welding zone of the cell. The gripping devices are part of the welding posts.

They move over the gutter to take a welding position. A procedure that checks the section bending and the reaction in the devices determines this position. Depending on the workpiece characteristics, two to four devices are useful to hold it during the welding. In order to respect welding process constraints, each device contains a wheel which has a u-shaped whole pointing downward in order to execute vertical section loading. The dimensions of the metal workpieces vary greatly. So, to maintain each type of section in each wheel, they are equipped with a centering device. Here, there are two rotating arms that ensure correct positioning of the metal framed section so that these remain stable in their movements.

The welding robot is placed on a two axis gantry which is make of a suspended rail. It can move over the welding areas to access all the assigned tasks. The second gantry axis and the wheel rotation are used in order to ensure proper task positioning, so that the robot may perform the tasks in the best conditions.

The tasks (except the transfer, placing, and gripping problems) consist of welding elements on workpieces. These elements are located on the extremity, and along the section. The welding trajectories, called beads, are shown in Figure 13.

The system simulates the complete cycle of the realization of a metal framed section. Chronologically, it simulates the loading of the workpiece on the trolleys, the transfer of the trolleys to the welding posts, and the vertical transfer of the workpiece in the wheels. Then, when the section is loaded, an automatic placement calculation using the algorithm of section 5, determines the three robot placement variables (both the gantry axis, and the wheel orientation), taking collision constraints between the robot and its environment into account. A result is shown in Figure 14. The robot task has several beads. Therefore, being careful about the cluttered robot's environment during the welding, we apply the path planning method, described in the previous section, to connect two successive joints.

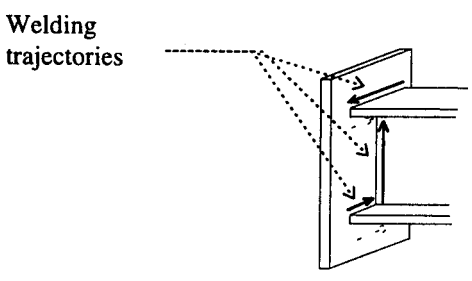

Fig. 13. Welding trajectories.

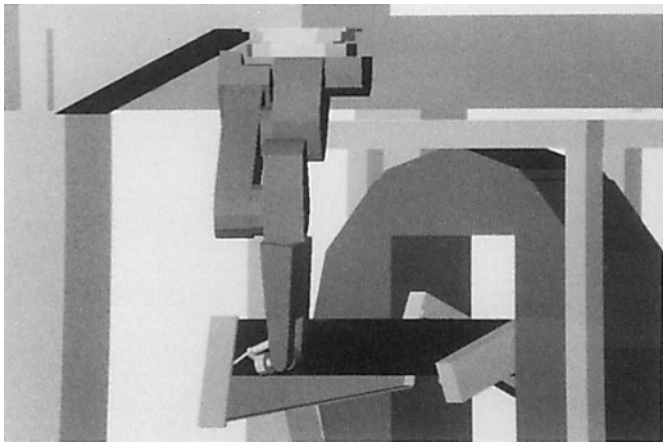

Fig. 14. Welding solutions.

The application that we just presented allows us to validate all the functions of the SMAR software. It also allows us to show its adapting capacities, in managing a 38 degrees of freedom robotized cell which contains 250 geometric objects. The simulation on workstation SILICONGRAPHICS has shown interesting functioning time, which is near real time simulation.

8. CONCLUSION

We have presented the possibilities of the SMAR CAD-robotics system. Although SMAR is not an industrial type system, it has enough functions to allow the user to deal with the problems encountered with robots usage, by the means of simulation. This system has a modeler, which describes the robot and its environment. This description is made up of the geometry and the kinematics definition of the environment. In order to define the geometry, the SMAR system modeler has primitives (cube, cylinders) for simpler objects, and Boolean operations for complex objects. The simulator provides task animation and task simulation. It's composed of the following:

—a reverser allowing for control in task coordinates and for us to find all of the compatible configurations with a position and orientation of the effector (multiple kinematic inverse solutions),
—a module for cycle time calculation.

To verify task accessibility without collision, the system has two internal algorithms. The first is based on an intersection calculation between objects, which allows collision detection between convex and non convex objects. The second one, is based on distance calculation between convex objects, allowing collision detection and the measurement of the objects proximity.

We have also presented in this study algorithms efficient in detecting collision, in calculating optimal placement as well as in establishing automatic creation of trajectories without collision. These different algorithms

https://doi.org/10.1017/S0263574797000088 Published online by Cambridge University Press

have shown their efficiency, especially throughout our study of a complex welded cell in metal framed construction, presented in the section 7.

The SMAR system is installed on different computers: μVAX, SUN SPARC 2, IBM-PC and SILICONGRAPHICS. The most complex version is installed on SILICONGRAPHICS. A user-friently interface in *C* language has been developed using Xlib, and also GL-Inventor.

References

1. S. Derby, ''Simulating Motion of General-Purpose Robot ARMS''. *Int. J. Robotics Research* **2,** No. 1, 3–12 (Spring 1983).
2. C. Laugier, ''Les apports respectifs des langages symboliques et de la CAO en programmation des robots''. *Séminaire CAO-Robotique, Etat de l'art,* La Grande Motte, Hermès (January, 1985) pp. 21–40.
3. M.L. Hornick and B. Ravani, ''Computer-aided off-line programming of robot motion''. *Int. J. Robotics Research* **4,** No. 4, 18–31 (Winter, 1986).
4. M.C. Leu, ''Elements of computer Graph Robot Simulation''. *Proceedings of the 1985 ASME International Computers in Engineering Conference.* Boston, Massachusett, (August 1985) pp. 65–72.
5. S. Zeghloul, J.P. Lallemand et D. Murguet, ''Modelling and Simulation of Mechanical Process in Hyperstatical Gripping with n-contact Points'', *Proceedings of six Romansy,* Cracow, Poland (September, 1986) pp. 139–147.
6. S. Zeghloul, ''Programmation hors-ligne multi-robots par simulateur graphique 3D'' *8ème Congrès Français de M1ecanique, Nantes (31 August 4 September, 1987)* ,,*Mécanique,* Nantes (31 August 4 September, 1987) pp. 314–315.
7. S. Zeghloul, ''Développement d'un système de C.A.O. robotique intégrant la planification de tâches et la synthèse de sites robotisés'' *PhD thesis* (Poitiers University, France, 1991).
8. A.A.G. Requicha & H.B. Voelcker, ''Solid modelling: a historical summary and contemporay assessment'' *IEEE, Computer Graphics and its Applications* (March, 1982) pp. 9–24.
9. R. Hillyard, ''The BUILD group of solid modelers'' *IEEE, Computer Graphics and its Applications,* (March, 1982) pp. 43–52.
10. S.S. Rao, *Optimization Theory and Applications* 2nd edition (Wiley Eastern Limited, USA, 1984).
11. S. Zeghloul, P. Rambeaud. and J.P. Lallemand, ''A fast distance calculation between convex objects by optimization approach'' *Procs. IEEE Int. Conf. Robotics Automation* (1992) pp. 2520–2525.
12. S. Zeghloul and P. Rambeaud, ''Comment on 'A Direct Minimization Approach for Obtaining the Distance between Convex Polyhedra' by James E. Bobrow'' *Int. J. Robotics Research* **11**, No. 5, 499–501 (1992).
13. J.A. Pamanes-Garcia, ''A criterion for optimal placement of robotic manipulators''. *IFAC Procs of the 6th Symp. on Information Control Problems in Manufacturing Technology* (1989) pp 183–187.
14. T. Yoshikawa, ''Manipulability of robotic mechanisms''. *Int. J. Robotics Research* **4,** No. 2, 3–9 (1985).
15. S.L. Chiu, ''Task compatibility of manipulators postures''. *Int. J. Robotics Research.* **7,** No. 5, 13–21 (1988).
16. M.J. Box. ''A new method of constrained optimization and a comparison with other methods'' *Computer J.* **8,** 42–52. (1965).
17. J.A. Pamanes-Garcia and S. Zeghloul, ''Optimal placement of robotics manipulators using multiple kinematic criteria''. *Procs of the 1991 IEEE Int. Conf. on Robotics and Automation.* (1991) **Vol. 1,** pp. 933–938.
18. B. Blanchard and S. Zeghloul, ''Planification de trajectoires sans collision pour robot manipulateur''. *Thèse in preparation* (1996).
19. O. Khatib, ''Real time obstacle avoidance for manipulators and mobile robots'' *Int. J. Robotics Research* **5**(1), 90–98 (Spring, 1986).
20. B. Faverjon, ''Hierarchical object models for efficient anti-collision algorithms'' *Int. Conf. Robotics and Automation,* IEEE (1989) pp. 333–340.

https://doi.org/10.1017/S0263574797000088 Published online by Cambridge University Press