An Evolutionary and Local Search Algorithm for Motion Planning of Two Manipulators

M. A. Ridao*, E. F. Camacho
Departamento Ingeniería de Sistemas y automática. Escuela Superior de Ingenieros Universidad de Sevilla, Spain

J. Riquelme, M. Toro
Departamento Lenguajes y Sistemas, Facultad de Informática y Estadística. Universidad de Sevilla, Spain

A method for obtaining coordinated motion plans of robot manipulators is presented. A decoupled planning approach has been used; that is, the problem has been decomposed into two subproblems: path planning, where a collision-free path is found for each robot independently only considering fixed obstacles, and trajectory planning, where the paths are timed and synchronized to avoid collisions with other robots. This article focuses on the second problem. The proposed plan can easily be implemented by programs written in most industrial robot programming languages. The generated programs minimize the total motion time of the robots along their paths. The method does not require accurate dynamic models of the robots and uses an evolutionary algorithm followed by a local search which produces near optimal solutions with a relatively small computational cost.

1. INTRODUCTION

One of the main problems encountered when operating a multirobot system is that of obtaining collision-free motion plans. These motion plans have to take into account environment obstacles and other robots. Finding collision-free motion planning algorithms in multirobot systems has been the focus of research in many works.[1] These algorithms can be classified into three categories: centralized, prioritized, and decoupled planning.

In centralized planning all robots are considered simultaneously and the problem is solved by find-

*To whom correspondence should be addressed; e-mail: ridao@cartuja.us.es.
Contract grant sponsor: CICYT.
Contract grant numbers: QUI199-0663, TAP-98-0541, 1FD97-0836, and TIC-99-0351.

ing a collision-free path in a space with dimension equal to the sum of the degrees of freedom of each of the robots. Algorithms based on centralized planning require a large amount of computation because of the size of the search space and only methods for solving specific problems have been proposed.[2–4]

In prioritized planning, an order is established among the robots. The motion of the robots is planned in this order, only one robot at a time. The motion of the robot is computed as if the robot were moving among stationary obstacles and the previous robots are considered as moving obstacles.[5] This is one way of reducing the computational complexity, but this approach may fail to find a solution, even if there is one.

In the third category, decoupled planning, the path of each robot is planned independently of the others, and afterwards the paths are synchronized to avoid collisions among the robots. These algorithms reduce the computational complexity considerably, but this gain results in a loss of completeness. Different methods have been presented to solve the interactions among the paths. Kant and Zucker[6,7] and Fujimura[8] used the technique for mobile robots. Lee and Lee[9] applied the technique to two manipulators by fixing the speed of one of the robots and modifying the other robot to find an obstacle-free plan. The collisions are studied using a bidimensional graph (collision map) where the path of the second robot is represented versus time and collision regions are included and approximated by rectangles. The robots are modeled with spheres and the movement of the robots was restricted to straight-line paths. Chang et al.[10] proposed an extension of this method using robots approximated by polyhedra and determining the minimal delay time value needed for collision avoidance.

O'Donnell and Lozano-Pérez[11] used the coordination diagram, where all the possible collisions between both robots appear. Applying the concept of SW-closure, they proposed an algorithm which did not require searching to find a solution. Bien and Lee[12] proposed a method to plan a trajectory that ensures collision-free and time-optimal motions. The main drawback of the method is that it can only be used when there is a single collision region in the coordination space. Lee[13] extended this procedure to situations in which there are more than one collision region.

Fraile et al.[14] presented a distributed planning and control architecture for multi-manipulator systems. A distributed trajectory planning approach based on artificial potential fields was used. Finally, Pérez-Francisco et al.[15] presented a method for the coordination of two robots when they are grasping objects of unknown shape and position from a conveyor belt, under time constraints.

The solution obtained by most of these algorithms is a robot trajectory; that is, they associate a time component with the points of the paths. These trajectories are very difficult to implement in most industrial robots, because they require the internal controller of each articulation to be fully available to the user to synchronize their movement with the movements of other robots' joints. Also, the selection of an accurate robot dynamic model is critical, because a discrepancy with the robot behavior can cause collisions between robots.

In this article, a method is presented to minimize the total motion time of two manipulators along their paths, avoiding collision regardless of the accuracy of the dynamic model used. The method uses the coordination diagram concept and provides trajectories that can easily be implemented in most industrial robot programming languages. Selection of the dynamical model is not a critical aspect, because a collision will never occur, even with an inaccurate dynamic model. However, since the algorithm tries to minimize the total motion time, precise models are required to obtain optimal results. This method is based on defining *synchronization points*[16] as spots on the path of each robot where it will stop until the other robot arrives at its respective point. Once both robots are at these points, motion can continue. This problem can be solved by an A* algorithm.[16,17] Additionally, the problem complexity is considerably reduced by using decoupled planning and the concept of the synchronization point. However the computational burden (in terms of memory needed and time required to solve it) can be prohibitive in some complex cases.

This article shows how evolutionary algorithms can be used to obtain collision-free motion plans in a multi-robot environment. Global optimization algorithms imitating certain principles of nature such as simulated annealing and the field of evolutionary algorithms (EAs) have proved to be useful tools for the optimization of high-dimensional and highly nonlinear problems. The EA is based on biological observations[18,19]: the means of natural evolution (natural selection and natural genetics) and the survival of the fittest.

The evolutionary algorithm maintains a population of problem solutions. First, the best individuals are selected to form the offspring using crossover

and mutation. Finally, the worse individuals of the population are replaced by the new offspring to form a new population.

The *fundamental intuition* of genetic algorithms[20] explains why the EA performs an effective search. The EA presents a continual improvement using the pair selection + mutation, working as a local search where the mutation operator slightly modifies a solution. If this new solution is better than previous ones, it will be accepted with a high probability by the selection mechanism. On the other hand, the pair selection + crossover avoids the process being trapped in a local minima, executing an intelligent jump to another search space region.

The realization of fast search in wide space is the main quality of genetics algorithms as function optimizers. However, their capability for complete local search is limited. Holland[21] suggested that genetic algorithms should be used as a preprocessor to perform the initial search, before tuning the search with local methods. Likewise, Grefensette[22] pointed out that genetic algorithms are qualified to identify high performance regions of the search space and he recommends: ''it may be useful to invoke a local search routine to optimize the members of the final population.'' A local search using a hill-climbing strategy is proposed by Syrjakow and Szczerbicka.[23] If the local search does not obtain a optimum value, the authors apply a backtracking process to execute a new genetic simulation.

The article is organized as follows: Section 2 states the problem and describes the coordination diagram concept. Section 3 introduces the evolutionary algorithms and defines individual structures, genetic operators, and other parameters used in the algorithms. In Section 4 some results are presented to demonstrate the feasibility of the method. Finally, concluding remarks are given.

2. PROBLEM STATEMENT

The problem can be stated as: Given two robots R_1 and R_2, a set of known fixed obstacles, and the initial and final configurations of R_1 and R_2, find a coordinated motion plan for the robots from their initial configuration to their final configuration avoiding collisions with environment obstacles and themselves.

To use the decoupled planning approach, it is necessary that a fixed obstacle collision-free path for each of the robots was previously obtained. These paths are obtained by considering only the fixed obstacles and not taking into account the other robot. There are many algorithms described in the literature[1,8] which can solve this problem. The proposed algorithm is independent of the algorithm used for obtaining the fixed obstacle collision-free path for each robot, although some details of the method used for the applications presented are given in Section 4. The paths which the robots are expected to follow are assumed to be given as a parameterized curve in the joint space

$$P = \Phi(\lambda), \qquad 0 \le \lambda \le \lambda_{\max} \text{ with } \Phi\colon \mathbf{R} \to \mathbf{R}^n$$

where n is the number of degrees of freedom of the robot and λ is the distance along the path. Let P^1 and P^2 denote the two collision-free paths generated for each of the manipulators. Let $\lambda^1_{\max}$ and $\lambda^2_{\max}$ denote the length of both paths. The P^1P^2-space is defined as the following $\mathbf{R}^2$ region:

$$P^1P^2 - \text{space} = \left\{(\lambda^1, \lambda^2)/0 \le \lambda^1 \le \lambda^1_{\max} \text{ and } 0 \le \lambda^2 \le \lambda^2_{\max}\right\}$$

A point (λ^1, λ^2) in the P^1P^2-space represents a snapshot of the robot's configurations [robot i articulation positions are $\Phi_i(\lambda^i)$]. Any continuous path from $(0,0)$ to $(\lambda^1_{\max}, \lambda^2_{\max})$ determines a coordinated execution of the two paths and is called a *coordination path*.

Let the *collision region* (CR) be defined as the set of points in the P^1P^2-space where a collision between the two manipulators occurs. Normally, the CR consists of several connected subsets, with complex shapes. If a coordination path does not cross through the collision region, it is called a *collision-free coordination path*.

To reduce the search space in the P^1P^2-space, a discretization of each path has to be made, so the path is divided into several equal-length intervals. Let the intervals of both paths be numbered from 1 to $\max_1$ and 1 to $\max_2$, respectively. Let δ^i_k be the kth interval along the path of the manipulator i. Now, a path can be defined as an ordered set of intervals; that is,

$$\Omega_i = \{\delta^i_k/1 \le k \le \max_i\}$$

A cell is defined as a pair (δ^1_m, δ^2_m) where $\delta^1_m \in \Omega_1$ and $\delta^2_n \in \Omega_2$. For simplicity, a cell will be represented as (m, n) and $\Omega_i = \{k/1 \le k \le \max_i\}$. With these discretized paths, the P^1P^2-space is transformed into an array of cells. This array of cells is

the *coordination diagram*. Cell $(1,1)$ corresponds to the first interval of each path, that is, the lower left-hand cell of the coordination diagram, and $(\max_1, \max_2)$ is the upper right-hand corner, the last interval of each path. A cell $(m, n) = \delta_m^1 \times \delta_n^2$ is considered as a *free cell* if it satisfies the following condition:

$$\forall \lambda^1 \in \delta_m^1,\ \lambda^2 \in \delta_n^2 \Rightarrow (\lambda^1, \lambda^2) \notin \mathrm{CR}$$

If the cell does not satisfy the above condition, it is considered as a *collision cell*. A correspondence between a path interval and robot configuration has to be established. A point λ_k, which is normally the middle point, is associated to each interval k.

Obtaining the coordination diagram is not a trivial problem. To evaluate whether the robots in a particular configuration are going to collide, it is necessary to determine if any link of one of the robot collides with any of the other robot's links. A main aspect of this problem is the geometrical forms used in links modeling. Many types of geometrical shapes, such as convex polyhedra, spheres, or convex spherical polyhedra have been proposed in the literature to facilitate the task of detecting robot collisions.[24] To determine the collision state of each cell is a more complex problem because the collision problem should be solved for each point in the cell. Only approximate solutions can be found. The technique used here was proposed by Lozano-Pérez[25] and consists of determining the maximum displacement for each of the robot's joints in the interval and increasing the size of the corresponding links accordingly. By using this technique, the resulting solid will include the volume swept by the moving link. Finally, in this article, a sweep-line algorithm[11] has been used to obtain the coordination diagram.

The size of the cells is an important factor. On the one hand, if the size of the cells chosen is too small, the computational time required by the algorithm may be prohibitive. On the other hand, the cell size has to be kept small because (1) the collision region will be oversized by the somewhat conservative sweep-line method if big cells are used and (2) the synchronization points are chosen to be the middle point of each cell. Large cells will decrease the size of the solution space and the optimality of the solution.

A collision-free coordination path will be composed of a sequence of free cells. To implement a trajectory in the coordination diagram, the motion of both robots must be synchronized; that is, both robots have to be simultaneously on points of the path corresponding to free cell coordinates. This synchronization can be implemented in an open loop manner (defining timed trajectories for both robots and assuming that the trajectories will be executed in the prescribed time). The open loop synchronization strategy is not very realistic, because collisions may be produced if one of the robots is not able to move at the programmed speed in one part of the trajectory; so to ensure that all robots are able to follow the trajectories, slow speeds have to be prescribed. Furthermore, timed trajectories cannot be implemented in many industrial robot programming languages.

Robots can be synchronized by a closed loop strategy based on synchronization points.[17] A synchronization point is a point in the coordination diagram which the robots have to reach; that is, any coordination path will necessarily pass through it. So, when a robot reaches a synchronization point, it waits for the other robot to reach the synchronization point before prosecuting its planned motion. This requires some sort of communication between robots. This type of synchronization has been used here and it is easy to implement because it is only necessary to connect a digital input with a digital output signal on each one of the controllers. Figure 1 shows a general structure of the system.

To avoid a collision it is possible to alter the coordination path defining the number and position

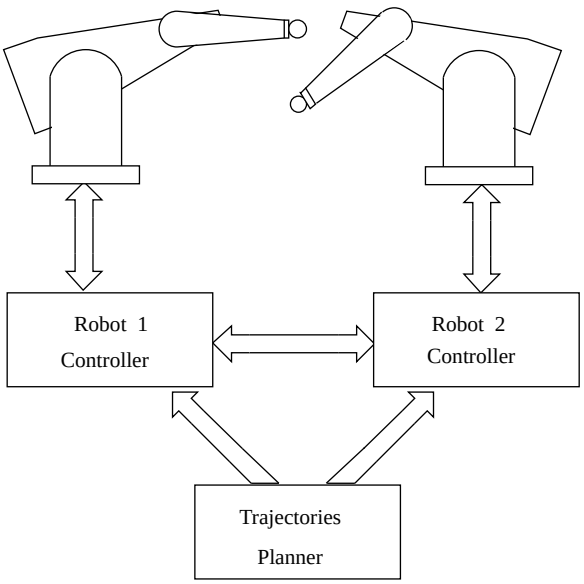

Figure 1. A general structure of the trajectory planner system.

of the synchronization points. Figure 2 shows a coordination diagram, where the dark regions are the collision regions, and it illustrates how a collision-free coordination path can be found by using synchronization points for the case where robot 1 moves along its associated path faster than robot 2.

A collision-free coordinated motion of two robots can be found by searching for a synchronization point sequence that minimizes the total coordinated motion time. The object of this article is to determine this synchronization point sequence.

Let us consider a rectangle formed by free cells in the coordination diagram and let us consider the motion of the robots from the lower left corner cell to the upper right corner cell. Any trajectory defined for each robot between these two points in the coordination diagram will be a collision-free coordination path; that is, the robots may move from the starting point to the end point at any speed and a collision between them will never occur. This class of rectangles is going to be called *free rectangles*.

Let us now consider a set of free rectangles, connected in such a way that the upper right corner of one rectangle is the lower left corner of the next. Furthermore, the lower left corner of the first one is the lower left corner of the whole coordination diagram, and the upper right corner of the last rectangle is the upper right corner of the coordination diagram, as can be seen in Figure 3. This set of rectangles is a *free rectangle sequence*, and the intersection points between two rectangles will be the synchronization points.

Given a free rectangle sequence, any coordination path constrained to pass over every synchronization point of the sequence will be a collision-free coordination path. This constraint is very easy to implement using most robot programming languages. The set of synchronization points will divide the path of each robot into several sections. Any section of the path between two synchronization points will be followed by every robot independently of the other, but a synchronization operation must be implemented at the end of the section, that is, at a synchronization point. The following is part of a program with the synchronization instructions written in VAL II, for the case of two robots.

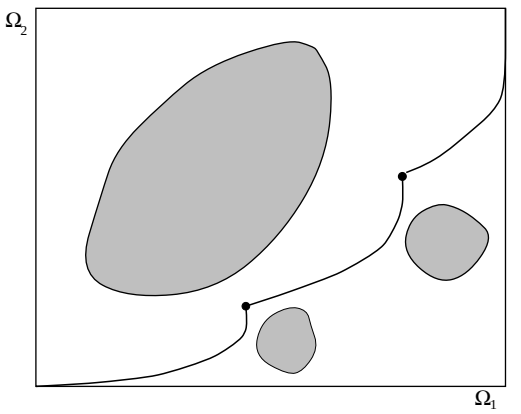

Figure 2. Free coordination path using synchronization points.

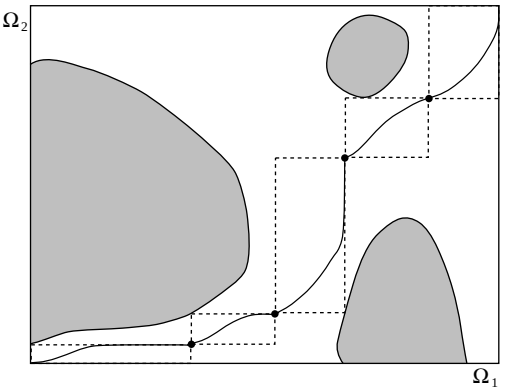

Figure 3. A free rectangle sequence.

```
ROBOT 1 PROGRAM
FOR i = 0 TO N
   MOVE #pts[i]
END
SIGNAL 1
WAIT SIG(1001)
SIGNAL −1
```

With this program, it is possible to execute the coordinated motion of the robots from an initial position to a goal position, without collision, either with fixed obstacles in the environment or with the other robot.

The problem now is to find a free rectangle sequence, that is, a synchronization point sequence that minimizes the total execution time necessary for the robots to complete their whole paths. The main variables used to find this sequence are the number of synchronization points, which depend on the collision region shape, and the position of these points in the coordination diagram. Notice that a delay is produced by adding a new synchronization point, because one of the robots has to stop to wait for the other robot.

This optimization problem can be solved by an A* algorithm as shown by Ridao and Camacho[17] although the great number of successors of each cell makes the searching tree too big, even with the pruning techniques suggested in ref. 17. To give an idea of the complexity of the problem, let us consider a coordination diagram of $m \times n$ cells and $p + 2$ synchronization points, with two of them fixed at positions $(1, 1)$ and (m, n). The number of potential solutions of this problem can be defined as the number of sequences with a number of synchronization points less than or equal to $p + 2$ and can be computed as follows:

Each solution of this problem can be obtained as two nondecreasing successions of p coordinates $\{x_1, \ldots, x_p\}$ and $\{y_1, \ldots, y_p\}$ in such a way that a synchronization point is the pair (x_i, y_i). Taking into account that coordinates x_i and y_i are independently chosen, the dimension of the search space is $t_p(m) \times t_p(n)$, where $t_p(k)$ is the number of different sequences of p integers $\{i_1, i_2, \ldots, i_p\}$ such that $1 \le i_1 \le i_2 \le \cdots \le i_p \le k$. Clearly, $t_p(k)$ is the number of combinations with repetition taken in a group of p elements

$$t_p(k) = \binom{k + p - 1}{p}$$

and the number of different synchronization point sequences is clearly given by

$$t_p(m) \times t_p(n) = \binom{m + p - 1}{p} \times \binom{n + p - 1}{p}$$

For the case of a coordination diagram of 100×100 and $p = 10$, there are around 10^{23} possible solutions, while for a coordination diagram of 150×150 cells and $p = 20$ there are around 10^{47} possible solutions.

3. THE PROPOSED ALGORITHM

EAs are basically search algorithms that start with some population of structures, called individuals, that are initial solutions to the problem and repeatedly perform the following cycle of operations until a termination condition is satisfied:

- *Evaluation*: Evaluate each individual in the population.
- *Selection*: Depending on the fitness of the individuals, select some of them for reproduction. These individuals are called parents.
- *Reproduction*: With some reproductive strategies, called genetic operators, generate some number of new individuals (offspring) from selected parents.
- *Replacement*: Replace some or all of the original population with the offspring, forming a new generation of individuals.

EAs usually have two different ending conditions: (1) the maximum number of generation and (2) a fixed number of generations without changes in the best individual. The first one is applied in this article. Some possible reproductive strategies are combinations of replicate or copies of some parents, crossover or recombination, where each new individual is constructed with inherited characteristics from its parents and mutation or change of an attribute or characteristic (gene) of an individual.

The main aspects that define the application of evolutionary algorithms for global optimization problems are chromosomic structures undergoing adaptation, ways to create an initial population, a fitness measure that evaluates the structures, genetic operators to modify the structures, constraints on offspring, and parameters to control the process.

3.1. Chromosomic Representation of the Individual

Each individual will represent a nondecreasing synchronization point sequence. Most of the problems solved using EAs represent individuals using fixed length chromosomes, because it is easier to implement genetic operators in this way. However, the number of synchronization points is a variable parameter in this application.

The solution adopted here is to consider variable-length chromosomes. The length of the chromosome defines the number of synchronization points of the sequence. Let n be the length of the chromosome. The initial point of the sequence is the path origin of each one of the robots $(1, 1)$ and the last point is $(\max_1, \max_2)$. Between these two vertices, an individual will be a set of synchronization points $\{P_i\}$ with $1 \le i \le n$ where $P_i = (x_i, y_i)$ and x_i, y_i are points corresponding to the paths of the first and second robots, respectively. A robot is not allowed to move back along its path, so the codification of an individual is therefore constrained by this fact. This constraint is that the coordinates of successive synchronization points should be monotonically increasing; that is, $x_i \le x_{i+1}$ and $y_i \le y_{i+1}$, for all i. Finally, all the points should be in the coordi-

nation diagram; i.e., $x_1 \geq 1$, $y_1 \geq 1$, $x_n \leq \max_1$, and $y_n \leq \max_2$. Considering that the coordination diagram is a discretized space, every synchronization point can be represented by a pair of integer numbers, so every individual must verify

$$\{(x_i, y_i) \in \Omega_1 \times \Omega_2 | \forall i,\ x_i \leq x_{i+1} \text{ and } y_i \leq y_{i+1}\} \quad 1 \leq i \leq n$$

A synchronization point sequence will be considered to be determined by $n+2$ points, where $(x_0, y_0) = (1, 1)$ and $(x_{n+1}, y_{n+1}) = (\max_1, \max_2)$.

An individual has been defined as an acceptable individual if it is a monotonically increasing succession of synchronization points. However, two classes of acceptable individuals can be distinguished:

- *Valid individuals*: An individual is valid if it forms an increasing sequence of free rectangles, and therefore defines a collision-free coordination path.
- *Nonvalid individuals*: An individual is nonvalid if there is at least one nonfree rectangle in the sequence. Then it is possible that a collision between the robots may take place. These individuals are not considered as a solution to the problem.

Finally, when a genetic operator is applied, it is possible for a decreasing synchronization point sequence to be obtained. It is not strictly an individual, but in this article, it will be called a *nonacceptable individual*.

3.2. Individuals in the Initial Population

The initial population is selected randomly. The following procedure was proposed to obtain a initial population with a wide diversity of solutions in a first approach, taking into account that the number of synchronization points of individuals is variable:

A maximum number of points *NMAX* is established (only for the initial population), and the number of synchronization points of the individuals of the initial population is distributed in a random uniform way between 1 and *NMAX*. In this way, a similar number of individuals with one, two, and so on until *NMAX* synchronization points are generated.

However, this uniform distribution of the number of points between 1 and *NMAX* was proved to be inappropriate in the tests. The reason is that nonoptimal solutions (with few points) can constitute local minima, and the population quickly converges toward them, losing genetic information. Therefore, the method improves when the individuals of the initial population have available a greater quantity of information, i.e., more points in the coordination diagram. For this, a increasing probability distribution from 1 (minimum) to *NMAX* (maximum) was selected.

Once the number of points n of an individual is selected, its coordinates are obtained as follows: two sets of n random values in $[0, 1]$ are generated; then they are ordered in a increasing way and projected on $[1, \max_1]$ and $[1, \max_2]$ intervals, respectively.

3.3. Fitness Measure

The objective is to minimize the cost function (f), so that the fitness function must increase as the cost function decreases. There are many ways in which this can be accomplished, for example, by making the fitness function equal to a constant minus the cost function[19] or by making the fitness function equal to $1/(1+f)$,[26] which has been used here. The probability of selection of each individual has been made proportional to the fitness function, which has been normalized by dividing it by the sum of the fitness functions of all individuals in the population. The selected fitness function guarantees that the probability of selection of an individual with a low cost in relation to the rest of the individuals will not be too great at the beginning, whereas in a more advance phase of the process, the difference between individuals with similar values of the cost function will be enlarged.

The evaluation of the cost function will consider the two different kinds of individuals: valid and nonvalid ones. In fact, two different cost functions will be used. For valid individuals, the cost function measures the total execution time needed by the robots to complete their paths when the synchronization points are placed at the positions defined by the individual specifications. Thus, to define the cost function for these individuals, it is necessary to use a model describing the robot motion. Dynamic models of the robot of different levels of complexity can be used for this purpose. That is, trajectory time can be computed from a simple model characterized by maximum acceleration and velocities of each joint, to a full dynamical model of the robot consid-

ering inertia, Coriolis force, friction, load, and other nonlinearities such as saturation or back slashes.

The paths used for the examples described in the article are composed of straight lines in the configuration space. This type of path has been chosen because it can easily be implemented on most industrial robots, although this is not an inherent feature of the method which can be used with other types of paths. The model used for this type of path is based on the works of Paul,[27] Luh and Lin,[28] and Tondu and El-Zorkany.[29] The following assumptions are made for each joint:

- The motion in each segment is made with constant speed, smooth transitions between segments, and a continuous acceleration profile. Maximum acceleration is reached in each transition.
- The transition is carried out symmetrically. Maximum acceleration is produced at the middle point of the transition.
- At each segment, the motion of each joint is synchronized with the movement of the rest of the joints. That is, the speed of each joint is computed to ensure that all joints finish their movements simultaneously.

Therefore, for each joint, during the constant speed phase, the joint position grows linearly with time. A fourth degree polynomial is used to ensure continuity in the position, speed, and acceleration during the transition phases.[29] The trajectory is specified with two parameters: the desired speed for each segment and maximum acceleration. These parameters can be obtained from the robot manufacturer or experimentally. In the case of the PUMA robot, the parameters given by Tondu and El-Zorkany[29] were considered, while for the applications presented with other robots, these parameters were obtained experimentally in our laboratory.

The cost function for nonvalid individuals is completely different. The execution time cannot be used as a cost measure, because this individual is not a solution of the problem. The function must measure how far it is from a valid individual. Obviously, the cost value for this kind of individual must be higher than any value for a valid individual. The function used is

$$f(N) = K + nco$$

where K is a high value with respect to the value associated with the valid individuals, and nco is the number of collision cells inside the rectangle sequence. This cost function makes sense when finding initial valid solutions is difficult because of the complexity of the collision regions. By using a cost function such as the one we suggest, the algorithm is able to find valid solutions.

3.4. Genetic Operators

Bearing in mind the special configuration of the individuals in this problem, modifications to the basic genetic operators are needed. The different genetic operators used in the algorithm are presented in this section. This includes the crossover and mutation operators. There have been many types of crossover and mutation operators proposed in the literature. Some of the genetic operators used here are extensions or modifications of previously proposed operators.

3.4.1. Crossover Operator

Given two individuals S^1 and S^2 formed by sequences of n and m synchronization points, respectively, the idea is to obtain another sequence, through genetic information exchange from parents S^1 and S^2. The following method is proposed (Fig. 4).

A synchronization point of S^1 called P is randomly selected. Then, Q, another synchronization point of S^2, is selected. Q is the first randomly obtained point with the values of both coordinates x and y greater than the respective coordinates of P, so the resulting child will be always an increasing sequence. It is possible that no point of S^2 verifies this limitation, then the child is returned as

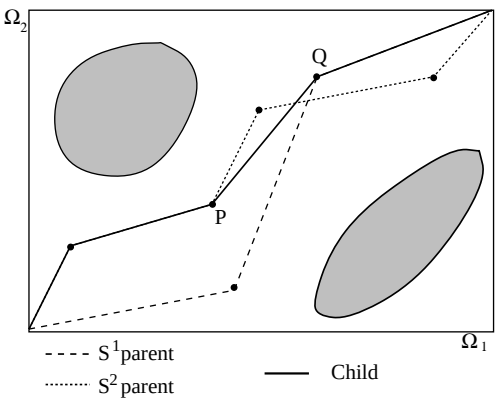

Figure 4. Crossover operator.

a copy of S^1. If any point Q exists, a new individual is formed with the first points of S^1 (P inclusive), followed by the points of S^2 from Q (inclusive) to the end of S^2. The number of SP of the child goes from two, when P is the first SP of S^1 and Q the last of S^2, to $n+m$ when P is the last point of S^1 and Q the first one of S^2.

Afterward, all the synchronization points of the resulting individual with the same coordinate values are reduced to a single synchronization point, that is, the same individual but with a more simplified structure.

3.4.2. Mutation Operators

A mutation produces a change in a gene of an individual. The proposed mutation types are described in the following paragraphs.

- **Slight mutation:** Given an individual $S = \{(x_i, y_i)/1 \le i \le n\}$, two integer values are randomly chosen; the first one k so that $1 \le k \le n$ and the second one m with $-MUTMAX \le m \le MUTMAX$, where $MUTMAX$ is the maximum permitted mutation. Then, this mutation consists of substituting point $P_k = (x_k, y_k)$ by $(x_k + m, y_k + m)$. This mutation has three variations:
  - — Double mutation: This is the result of selecting two different values of m for each coordinate.
  - — Single mutation: This second possibility is the result of applying the mutation to only one of the coordinates of P_k (Fig. 5).
  - — Nonuniform mutation: The third variant is obtained when $MUTMAX$ varies as a function of the number of the executed generations.

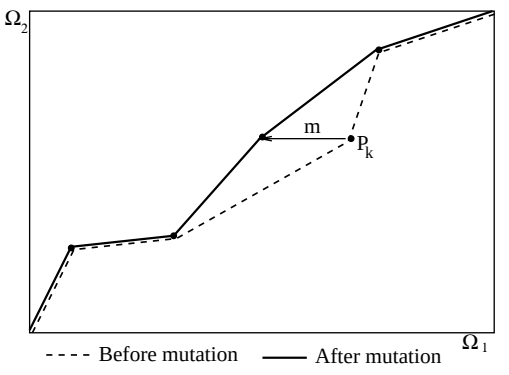

Figure 5. Slight mutation (single mutation).

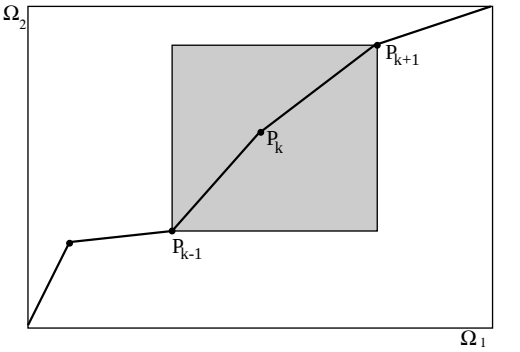

Figure 6. Proportional mutation.

The individuals obtained through these mutations can be nonacceptable ones, and so, some action must be taken to obtain acceptable individuals.

- **Strong mutation:** Several strong mutations have been implemented and tested with satisfactory results. The following mutations are considered in this article:
  - — *Proportional mutation*: Given an individual $S = \{(x_i, y_i)/1 \le i \le n\}$, an integer value k is randomly chosen so that $1 \le k \le n$. Then, the coordinates of the point $P_k = (x_k, y_k)$ will be changed by new values in such a way that they will be randomly distributed over the rectangle defined by the lower left vertex (x_{k-1}, y_{k-1}) and the upper right vertex (x_{k+1}, y_{k+1}); that is, P_k will be substituted by a new point inside the gray area in Figure 6. Every individual generated by this mutation is an acceptable one. This mutation has a variation consisting of modifying only one of the coordinates of the point, chosen randomly. This variation has been called the single proportional mutation.
  - — *Synchronization point elimination*: This mutation eliminates a randomly selected synchronization point of S; that is, the new individual is identical to S, but with one SP less.
  - — *Segment mutation*: This mutation chooses two consecutive synchronization points $P_k = (x_k, y_k)$ and $P_{k+1} = (x_{k+1}, y_{k+1})$, and a slight mutation is applied to them with a probability. Thereafter, this mutation adds a new point between both of them. For this purpose, the differences $d_x = x_{k+1} - x_k$ and

$I_y = y_{k+1} - y_k$ are obtained. Then, another two integers are chosen randomly (v_x be-tween 1 and $d_x - 1$ and v_y between 1 and $d_y - 1$). The new point is located in the individual after P_k with coordinates $(x_k + v_x, y_k + v_y)$ (Fig. 7).

— *Reflection*: A set of synchronization points of the sequence is selected with a given probability. Every selected point is substituted by its symmetrical point in relation to the diagonal; that is, a point $P_k = (x_k, y_k)$ is substituted by (y_k, x_k).

3.5. Constraints on the Individuals

When these operators are applied, the resulting individuals may not pass the conditions necessary for being an acceptable individual. Various actions can be taken to resolve this problem.

- *Refusing a solution*: The individual is refused and substituted, by simply carrying out another crossover or mutation operation until an acceptable individual is obtained or by selecting another individual to cross or mutate it.
- *One-coordinate moving solution*: This consists of changing only the coordinate affected at each conflicting point P_k, modifying the sequence so that the coordinate of the following point equals the conflicting coordinate of point P_k. A variation of this solution is to modify the conflicting coordinate, making it equal to the respective coordinate of the following point.
- *Eliminating points solution*: The last solution consists of eliminating the conflicting point.

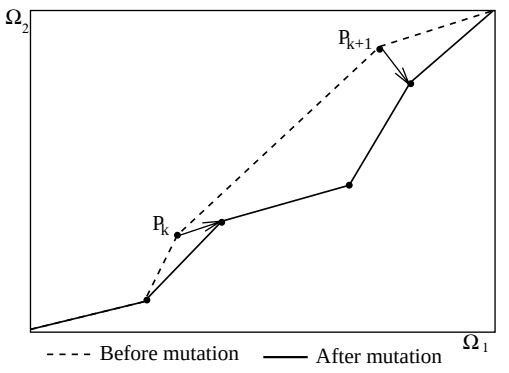

Figure 7. Segment mutation.

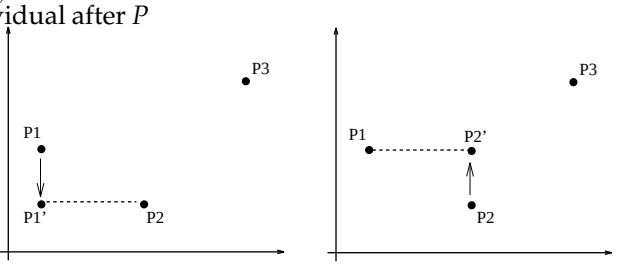

Figure 8. Resolution of the conflicts when nonacceptable individuals appear.

Figure 8 shows an illustrative example of the methods. P2 is the conflicting point, which can be solved by moving either P1 or P2. Suppose that during a crossover or a mutation execution a nonacceptable sequence defined by {P1, P2, P3} is generated. The *one-coordinate moving solution* would give rise to the path {P1′, P2, P3} or {P1, P2′, P3}, whereas the *eliminating solution* would give rise to the paths {P1, P3} or {P2, P3}.

3.6. Local Search Algorithm

In our work, the evolutionary algorithm gives an approximate solution, and starting from this solution, a heuristic search algorithm will find the optimum. The proposed local search procedure consists of a monotonous random walk search with the following structure:

```
Procedure Random-Walk
  Generate(CurrentSolution)
  BestSolution ← CurrentSolution
  REPEAT
    CurrentSolution ← GenerateNeighbor
        (CurrentSolution)
      IF Objective(CurrentSolution) <
          Objective(BestSolution)
      THEN BestSolution ← CurrentSolution
  UNTIL StopCriterion
```

We have used the previously defined mutation operators to implement the GenerateNeighbor subroutine. Notice that these operators, even the strong mutations, perform a local search, exploring for minima at the nearness of the previous solution. The strong operators allow us to find optimal solutions even when we starting with solutions with a nonoptimum number of synchronization points. In most cases, the proposed hybrid technique obtains a computational time reduction, in relation to a pure

EA, because the local search is performed on a more restricted space. However, the method would fail if the starting solution of the local search were not in the proximity of the optimum. The election between slight and strong mutations is made with a random procedure as the following:

```
Procedure Generateneighbor(Solution)
  Generate(ProbChange)
  IF ProbChange < ProbChangeLocal
    THEN RETURN SlightMutation(Solution)
  ELSE RETURN StrongMutation(Solution)
```

The StrongMutation function performs one of the previously describes mutation, selected by a random procedure.

3.7. Parameters of the Algorithms

There are many alternatives for the design of the EA. Three types of parameters must be defined in this application. First are the usual parameters of any EA, such as size of the population, number of generations, mutation probability, selection mechanism of the population to be reproduced, and ratio of duplicates in each generation in the elitist selection mechanism.

Second, the specific parameters needed for this EA application are the maximum number of points in the initial population *NMAX*, the ratio of the valid individuals in the initial population, the parameters associated to some types of mutation, and the solutions for converting a nonacceptable individual generated by a mutation operation into an acceptable one. Finally, parameters related to the local search are the number of iterations and the parameter ProbChangeLocal.

4. APPLICATION EXAMPLES

The proposed algorithm has been implemented and applied to several examples to test it and study its efficiency. The first example corresponds to the coordinated motion for two PUMA-560 robots. Figure 9a represents the coordination diagram with a single collision region and 105×82 cells. This is a very common coordination diagram in real multirobot applications. Initial and final configurations are shown in Figure 10. The optimal solution consists of a single synchronization point represented in Figure 10b. The second one is a more complex example (Figure 9b), with two SCARA-type robots and a

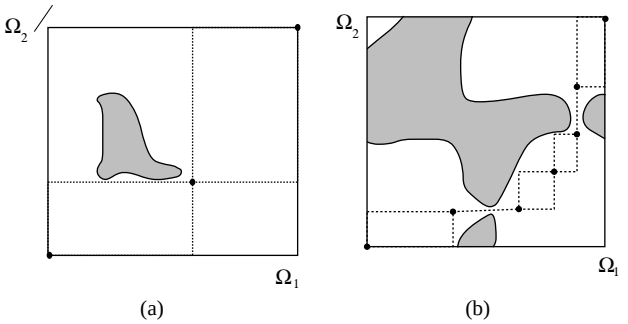

Figure 9. Coordination diagrams corresponding to examples 1 and 2.

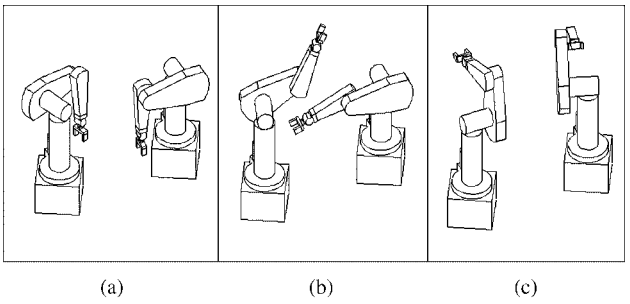

Figure 10. Example 1. (a) Initial position. (b) Position at the Synchronization point. (c) Goal.

coordination diagram with 82×68 cells. Any path must verify the constraint consisting in passing through two narrow corridors. Figure 11 shows the initial position, the goal, and some intermediate positions.

The last example corresponds to the motion of two SCORBOT robots with 16 collision regions and 180×180 cells (Fig. 12). It is an iterative motion represented in Figure 13. The motion from Figure 13-1 to Figure 13-2 and again to the initial configu-

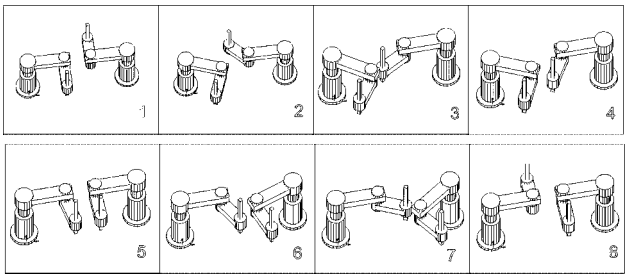

Figure 11. Example 2. Initial position (1), goal (8), and intermediate positions.

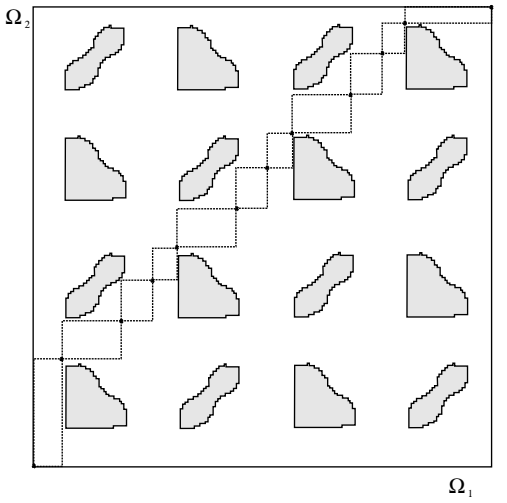

Figure 12. Example 3. Coordination diagram.

ration (Figure 13-3) is repeated twice. The best synchronization point sequence obtained is shown on every coordination diagram. To obtain the initial paths, the method proposed by Kondo[30] has been used. It is based on a decomposition in cells of the *configuration space* followed by a multistrategic bidirectional search. This method obtains a path given by a sequence of points in the configuration space. This path is approximated by a sequence of straight line segments.

The results obtained for these three examples are described in the following paragraphs.

In all the examples, the interval for path discretization is 4° (Discretization is carried out on the variable which measures the path length which, as indicated, is defined in the robot configuration space. Taking into account the fact that in the examples robots with revolution joints are used, interval sizes are given in degrees). Example 1 has been executed with a population of 25 individuals, while the population selected for the second and third examples

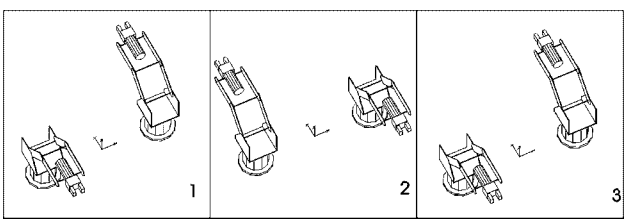

Figure 13. Example 3. Initial position (1), Intermediate position (2), and goal Position (3).

was 100 individuals. The selection of the population will follow an elitist model with selection probability proportional to fitness, 10% being the ratio of parents that will be duplicated in the next generation, the rest being offspring obtained by crossover and mutations of these. The nonacceptable individuals have been eliminated using the eliminating points solution. Different criteria have been used to obtain the initial population for the examples presented in the article. For the two first examples the initial population has been chosen with at least 15% valid individuals.

For the third example the initial population was chosen to have at least 15% almost-valid individuals (a sequence of rectangles containing less than 120 obstacle cells). The reason is that it takes a long time to generate valid individuals by a random generation procedure, whereas the evolutionary algorithm is able to generate valid individuals from almost-valid individuals by using the proper cost function. *NMAX* has been chosen as 10 and the mutation probability is 30%. These values have been obtained empirically after numerous tests. Typical CPU times on an IBM RISC-6000 320H are 1 min 30 s for 200 generations and 100-individual populations and 25 s for populations with 25 individuals.

The ProbChangeLocal value has been chosen after several tests. The tests show that the major improvement in the solution is due to slight mutation, but strong mutations are indispensable, because they avoid the solution being trapped in local minima after a few iterations. Therefore, the ProbChangeLocal must be relatively low (~ 0.1) to get a larger presence of strong mutations. All the probabilities of selecting a certain strong mutation have been made equal.

The results confirm the efficiency of the proposed approach. The solutions (motion time in seconds) found for examples 1 and 2 can be observed in Table I. The tests compare the results of the evolutionary algorithm without local search for 300 generations of 100 individuals (GA 300 in tables), for 100 generations also without local search (GA 100), and finally, the achieved values for a local search process beginning with the best individual of the generation 100 (GA 100 + rw). The stop criteria for random walk were 5,000 iterations. That is, 5,000 calls to the evaluation function, equivalent to an additional computational cost of 50 generations of 100 individuals. Notice that the GA 100 + rw process (15,000 calls to the evaluation function) has a computational cost of 50% of GA 300 (30,000 calls). This reduction occurs because the evaluation of the

Table I. Example 1 and 2 results.[a]

Example	Method	Avg.	σ	Minimum
1	GA 300 g	3.73	0.06	3.70
	GA 100 g	3.84	0.12	3.70
	GA 100 + rw	3.75	0.10	3.70
2	GA 300 g	7.71	0.09	7.58
	GA 100 g	8.27	0.62	7.83
	GA 100 + rw	7.62	0.13	7.51

[a]Robot's motion time in seconds. g, generations; rw, random walk.

fitness function is almost 90% of the total computational cost. Finally, the number of simulations for each test was 50.

Comparing the GA method with the GA + rw method, results are similar for example 1 and slightly better in example 2. For example, the average motion time in example 1 is 3.73 s (7.71 for example 2) for GA with 300 generations and 3.75 s (7.62 for example 2) for GA 100 + rw.

The results for example 3 can be seen in Table II. These values have been obtained for 100, 200, 300, and 500 generations of 100 individuals. After each evolutionary process, a local search was executed with 5,000 iterations starting with the best individual. These values clearly confirm the effectiveness of the proposed minimization method. Notice that the solution of GA 200 + rw is 10% better than that for GA 500, in spite of the fact that the computational cost is reduced by 50%.

5. CONCLUSIONS

This article describes a method based on hybrid evolutionary algorithms to generate collision-free motion plans in multirobot environments. The plans can be written in most industrial robot programming languages and guarantee the coordinated motion of two robots without collision with the environment fixed obstacles or between the robots. The algorithm tries to find a synchronization point sequence that minimizes the total execution motion time.

The cost function, which is complex and has numerous local minima and flat behaviors, cannot be easily optimized by traditional optimization algorithms. The tests have demonstrated the capability of the proposed algorithm to find satisfactory solutions in few generations. The solutions found are close to the global minimum for all the examples treated. This hybrid technique gets better results than a pure EA, even with a lower computational cost. Tests show that these benefits increase with the complexity of the problem. Although the algorithm has been implemented for two robots, it could be extended to an environment with a greater number of robots.

Table II. Example 3 results.[a]

Method	Avg.	σ	Minimum
GA 100	44.98	1.34	42.35
GA 100 + rw	38.41	1.99	35.98
GA 200	42.95	1.71	39.78
GA 200 + rw	37.10	1.42	35.98
GA 300	41.19	1.16	38.63
GA 300 + rw	36.84	1.05	36.02
GA 500	40.82	1.03	39.26

[a]Robot's motion time in seconds. rw, random walk.

REFERENCES

1. J.C. Latombe, Robot motion planning, Kluwer Academic Publishers, Norwell, MA, 1991.
2. T.J. Schwartz and M. Sharir, On the piano movers problem. III. Coordinating the motion of several independent bodies: the special case of circular bodies moving amidst polygonal barriers, Int J Robotic Res 2 (1983), 45–47.
3. S. Fortune, G. Wilfong, and C. Yap, Coordinated motion of two robot arms, Proc IEEE Int Conf Robotics Automat, San Francisco, 1986, pp. 1216–1223.
4. M. Mediavilla, J.C. Fraile, and G.I. Dodds, Optimisation of collision free strategies in multi-robot systems, ICRA '98, 1998, pp. 2910–2915.
5. M. Erdman and T. Lozano-Pérez, On multiple moving obstacles, AI Memo 883, Artificial Intelligence Laboratory, MIT, 1986.
6. K. Kant and S.W. Zucker, Toward efficient trajectory planning: the path-velocity decomposition, Int J Robotics Res 5 (1986), 72–89.
7. K. Kant and S.W. Zucker, Planning collision free trajectories in time varying environments: a two-level hierarchy, Proc IEEE Conf Robotics and Automat, 1988, pp. 1644–1649.
8. K. Fujimura, Motion planning in dynamic environments, Springer-Verlag, Tokyo, 1991.
9. B.H. Lee and C.S.G. Lee, Collision-free motion planning of two robots, IEEE Trans Syst Man Cybernet SMC-17 (1987), 21–32.

10. C. Chang, M.J. Chung, and B.H. Lee, Collision avoidance of two general robot manipulators by minimum delay time, IEEE Trans Syst Man Cybernet 24 (1994), 517–522.
11. P.A. O'Donnell and T. Lozano-Pérez, Deadlock-free and collision-free coordination of two robot manipulators, Proc IEEE Int Conf Robotics Automat, 1989, pp. 484–489.
12. Z. Bien and J. Lee, A minimum-time trajectory planning method for two robots, IEEE Trans Robotics Automat 8 (1992), 414–418.
13. J. Lee, A dynamic programming approach to near minimum-time trajectory planning for two robots, IEEE Trans Robotics Automat 11 (1995), 160–164.
14. J.-C. Fraile, C.H. Wang, C.J.J. Paredis, and P.K. Khosla, Agent-based control and planning of a multiple-manipulator assembly system, Proc 1999 IEEE Int Conf Robotics Automat, Detroit, 1999.
15. M. Pérez-Francisco, A.P. del Pobil, and B. Martínez-Salvador, ''Coordinated motion of two robot arms for real applications,'' Task and methods in applied artificial intelligence, A.P. del Pobil, J. Mira, and M. Ali (Editors), Lecture Notes in Computer Science 1416, Springer-Verlag, Berlin, 1998, pp. 122–131.
16. M.A. Ridao, J. Riquelme, E.F. Camacho, and M. Toro, ''An evolutionary + local search algorithm for planning two manipulators robot,'' Task and methods in applied artificial intelligence, A.P. del Pobil, J. Mira, and M. Ali (Editors), Lecture Notes in Computer Science 1416, Springer-Verlag, Berlin, 1998, pp. 336–346.
17. M.A. Ridao and E.F. Camacho, Automatic motion programming of robots working in a colliding environment, Proc IMACS IEEE-SMC Multiconf Computational Eng Syst Applications, Lille (France), 1996.
18. J.H. Holland, Adaptation in natural and artificial systems, The University of Michigan Press, Ann Arbor, 1975.
19. D.E. Goldberg, Genetic algorithms in search, optimization and machine learning, Addison-Wesley, Reading, MA, 1989.
20. D.E. Goldberg, The design of innovation: lessons from genetic algorithms, lessons for the real world, Internal Report 98004, Illinois Genetic Algorithms Laboratory, Department of General Engineering, University of Illinois at Urbana-Champaign, Illinois, 1997.
21. J.H. Holland, Adaptation in natural and artificial systems, University of Michigan Press, Ann Arbor, 1975.
22. J.J. Grefensette, ''Incorporating problem specific knowledge into genetic algorithms,'' Genetic algorithms and simulated annealing, L. Davis (Editor), Morgan Kauffmann Publishers, San Francisco, 1987, pp. 42–46.
23. M. Syrjakow and H. Szczerbicka, Optimization of simulation models with REMO, Proc Conf Modeling Simulation, 1994, pp. 274–281.
24. K. Gupka and A.P. del Pobil (Editors), Practical motion planning in robotics, Wiley, New York, 1998.
25. T. Lozano-Pérez, A simple motion-planning algorithm for general robot manipulators, IEEE J Robotics Automat RA-3 (1987), 224–238.
26. J.R. Koza, The genetic programming: on the programming of computers by means of natural selection, MIT Press, Cambridge, MA, 1992.
27. R. Paul, Robot manipulators: mathematics, programming and control, MIT Press, Cambridge, MA, 1982.
28. J.Y.S. Luh and C.S. Lin, Optimum path planning for mechanical manipulators, ASME Trans 102 (1981), 142–151.
29. B. Tondu and H. El-Zorkany, Identification of a trajectory generator model for the PUMA-560 robot, J Robotic Syst 11 (1994), 77–90.
30. K. Kondo, Motion planning with six degrees of freedom by multistrategic bidirectional heuristic free-space enumeration, IEEE Trans Robotics Automat 7, (1991), 267–277.