[1]Automatic Generation of Collision-Free Programs for Multiple Manipulators Using Evolutive Algorithms

MIGUEL A. RIDAO[1], J. RIQUELME[2], E.F. CAMACHO[1] AND M.TORO[2]
[1]Dpto. Ingeniería de Sistemas y Automática [2]Dpto. Lenguajes y sistemas Informáticos
Universidad de Sevilla
Escuela Ingenieros. Camino de los Descubrimientos s/n 41092 - SEVILLA
SPAIN
[1] ridao@cartuja.us.es [2] riquelme@lsi.us.es

Abstract: A method based on Evolutionary Algorithms for obtaining coordinated motion plans of multiple manipulator robots using a Decoupled Planning approach is presented. The problem has been decomposed in two subproblems: path planning of each robot independently of the other robots and trajectory planning, where the paths are synchronized. This paper is focused on the second problem. An evolutionary algorithm is proposed to generate free collision robot programs that minimize the total motion time of the robots along their paths

Key-words:- Evolutive algorithms, multirobot systems, motion planning

1 Introduction

Motion planning is an important problem in multi-robots systems. The plan has to take into account, not only the workspace obstacle, but the other robots. So, the motion planner has to consider the possibility of a collision among them and has to plan the movement of each robot in such a way that all collisions are avoided.

Several approaches have been proposed to solve this problem [1], but the most popular approach is the Decoupled Planning method [2]. This method divided the motion planning problem into two subproblems:

a) Free-collision path for every robot just considering fixed obstacles: A classic path-planning problem is solved considering only one robot at a time, and fixed obstacles (other robots are not considered as obstacles at this stage). [1]. The work of this paper is not centre in this subproblem, so we suppose that a path for every robot has been previously obtained.
b) Interactions among robot paths: The second subproblem determines the way the robots have to executed the movement along their paths without collisions among them

This decomposition reduces the complexity of the problem but this gain results in a loss of completeness. Several techniques have been proposed to solved this second problem for manipulator systems [3],[4]. The type of solutions given by the algorithm is an important practical issue. Most of the above algorithms provide robot trajectories (a time component associated with each point of the path). Such trajectories are very difficult to implement in most industrial robots, driven by robots program.

In this paper a method for automatic generation of robot programs is presented. The method can be applied to more than two robots. The obtained programs will execute the coordinated motion of several manipulators without collisions. The programs can be easily written in most industrial robots programming language. They will include the movement code and the synchronization code and the coordination will be designed in such a way that minimizes the total motion time of the robots.
The optimization problem will be solved using Evolutive Algorithms [5]. Evolutive are search procedures based on natural selection mechanism and genetic concepts.

2 A decoupled planning method

Let's consider a robotic system composed by R robots sharing a common workspace with known fixed obstacles. A robot j will have N_j degree of freedom. The problem is to plan the movement for every robot from an initial configuration to a final configuration avoiding collisions

[1] This work was supported in part by the Spanish research agency CYCIT under grant TAP 96-0884 and TAP 98-0541

with fixed obstacles and themselves. Initial and final configuration are previously known.

Let's consider that a fixed obstacle collision-free for each of the robot has been previously obtained. These paths are assumed to be given as parameterised curves in the joint space.

$P^j = \Phi^j(\lambda) \quad 0 \le \lambda \le \lambda^j_{\max} \quad$ with $\quad \Phi^j : \mathbf{R} \to \mathbf{R}^{N_j}$

and $\quad 1 \le j \le R$

The Coordination Space is an *R*-dimensional verifying:

$$CS = \left\{ \left(\lambda^1,...,\lambda^R\right) / \, 0 \le \lambda^j \le \lambda^j_{\max} \quad \text{with} \quad 1 \le j \le R \right\}$$

A point in *CS* represents a robotic system configuration, determining the position of the articulations of each robot.

The Collision Region (*CR*) is the set of points of the *CS* where at least, two robots are colliding. A two robot system with their prescribed paths is represented in Figure 1-a. Its Coordination Space with the Collision region appears in Fig. 1-b.

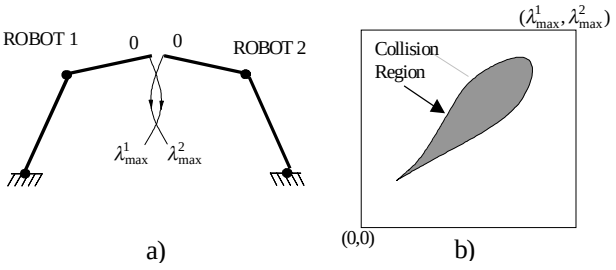

Figure 1 a) Two manipulator system with their prescribed paths. b) Coordination Space and Collision region

A Free-Collision Coordinated Path (*CP*) is a continuous path in the Coordinated Space from point (0,...0) to $\left(\lambda^1_{\max},...\lambda^R_{\max}\right)$ where no collision among robots is produced .

The proposed method are based on a search in the Coordination Space. So, in order to reduce the complexity of the problem, each robot path is discretised into several equal intervals. The intervals of the path of robot *j* will be numbered from 1 to max_j, and δ^j_k will note the interval number *k* of the path of robot *j*. A discretised path of robot *j* is described as:

$$\Omega_j = \left\{ \delta^j_k \, / \, 1 \le j \le max_j \right\}$$

A Cell *C* is defined as the following *CS* subspace:

$$C = \delta^1_{n_1} \times \delta^2_{n_2} \times ... \times \delta^R_{n_R} \quad / \quad \delta^j_{n_j} \in \Omega_j$$

A cell is collision-free if verifies:

$$\forall \lambda^j \in \delta^j_{n_j} \quad \text{with} \quad 1 \le j \le R \quad \Rightarrow \quad \left(\lambda^1,\ldots,\lambda^R\right) \notin CR$$

In order to simplify the notation, an interval of a path will be noted by its ordering number, that is $\Omega_j = \{ k \, / \, 1 \le k \le max_j \}$. Now, a cell is defined as a *R*-tuple $C=(n_1,...,n_R)$ and the Coordination Space is transformed into an array of cells named Coordination Diagram (*CD)*. Extending the previous concept, a Coordination Path in *CD* is defined as non-decreasing secuence of consecutive free cells from the initial cell (1,...,1) to the final cell $(max_1,..,max_R)$.

In order to obtain *CD,* the number of cells to be evaluated in a system with *R* robots is $max_1 \times ... \times max_R$. Nevertheless, it is not necessary to evaluate every cell. The whole *DC* can be obtained in a simpler way by computing several two-dimensional coordination diagrams. Let's consider a three-robot system. If robots 1 and 2 collide in a point of their respective paths, this collision condition persists independently of position of robot 3. So, the set of two-dimensional *CD* computed with every pair or robots (*i*,*j*) with $1 \le i,j \le R$, will provide information to evaluate the collision state of every *R*-dimensional cell.

A free collision coordination path will be composed of a sequence of free cells. In order to implement a trajectory in the coordination diagram, the motion of the robots must be synchronized, that is, the robots have to be simultaneously on points of the path corresponding to free cell coordinates. Robots can be synchronized by a closed loop strategy based on synchronization points [6],[7]. 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 others to reach the synchronization point before prosecuting its planned motion.

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

Let's consider an hypercube formed by free cells in the coordination diagram and let's 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 generate a collision-free coordination path. This class of hypercubes is going to be called *Free Hypercubes*.

Let's now consider a set of free hypercubes, connected in such a way that the upper right corner of one hypercube is the lower left corner of the next, as can be seen in Figure 2. 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 hypercube is the upper right corner of the coordination diagram. This set of hypercubes is a *Free Hypercube Sequence*, and the intersection points between two hypercubes will be the synchronization points.

Given a free hypercube 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 language. 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 others, but a synchronization operation must be implemented at the end of the section, that is, at a synchronization point. The required communication among robots is easy to implement connecting digital input and output among the controllers of the robots.

The problem now is to find a free hypercube 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.

This optimization problem can be solved by an A^* algorithm [6], although the great number of successors of each cell makes the searching tree too big .

3 The proposed evolutive algorithm

This section describes the proposed algorithm to find an optimal hypercube sequence. First, formal description of chromosome structure and operations over this representation is indicated. Thereafter, the main aspects of the evolutive algorithm are described.

3.1 General definitions.

3.1.1 Data structures.

Point: A point P in a R-dimensional space is defined as a tupla $(p_1,p_2,...,p_R)$ with $p_i \in$ N. A point defines a cell in the coordination diagram.

Order relationships:

1. Less or equal operator: $P \leq Q$, where P and Q are points, if $\forall i\ p_i \leq q_i$.
2. Less operator: $P<Q$ if $\exists j / p_j<q_j$ and $\forall\ i \neq j\ p_i \leq q_i$.

***Sequence*:** A sequence is defined as a tupla of points $S=<S_1,S_2,...,S_n>$. An hypercube sequence will be defined with this type of data.

***Sequence length*:** The length of a sequence L (noted as $\#L$) is defined as its number of points.

***Ordered sequence*:** $S=<S_1,S_2,...,S_n>$ is an ordered sequence if $S_i \leq S_{i+1}\ \forall i$.

***Multiple Point*:** Let's consider an ordered sequence S where $\exists i,j>0 / S_{i-1}<S_i=S_{i+1}=...=S_{i+j}<S_{i+j+1}$. The subsequence of points $<S_i,S_{i+1},...,S_{i+j}>$ is defined as a *Multiple Point.*

***Reduced ordered sequence*:** S is defined as a reduced ordered sequence if it is an ordered sequence and verifies $S=<S_1,S_2,...,S_n>$ with $S_i<S_{i+1}\ \forall i$. That is, an ordered sequence without Multiple Points.

***Presequence*:** Given S, presequence(S,m) with $m \leq \#S$ is defined as $<S_1,S_2,...,S_m>$. If $m>\#S$, presequence(S,m)= S.

***Postsequence*:** Given S, postsequence(S,m) with $m \leq \#S$ is defined as $<S_m,S_{m+1},...,S_n>$. If $m>$#S, postsequence(S,m)=$\varnothing$.

3.1.2 Operations.

***Concatenation*:** The concatenation (+ operator) of two sequences S and T is defined as the sequence $<S_1,...,S_{\#S},T_1,...,T_{\#T}>$.

Movement: Given an ordered sequence S, a natural m with $m \leq \#S$, and a point Q, movement(S,m,Q) is defined as the sequence $<S_1,S_2,...,S_m+Q,...,S_{\#S}>$ where S_m+Q represents a sum in the vectorial space of points.

***Elimination*:** Given a sequence S and a natural m with $0 \leq m \leq$ #S, elimination(S,m) operator gives the sequence $<S_1,S_2,...,S_{m-1},S_{m+1},...,S_{\#S}>$.

Insert: Given a ordered sequence S, a natural m with $0 \leq m \leq \#S$ and a point Q, insert(S,m,Q) is defined as the sequence $<S_1,S_2,...,S_m,Q,S_{m+1},...,S_{\#S}>$.

Order: An aditional operator to transform a not-ordered to ordered sequence has been defined. Let be S a sequence where a point S_m do not verify $S_{m-1} \leq S_m \leq S_{m+1}$. We define order($S$)=$S'$, where $S'=S$ except for S'_m defined as:

$$(S'_m)_j = \begin{cases} (S_{m-1})_j & \forall j \ / \ (S_{m-1})_j > (S_m)_j \\ (S_m)_j & \forall j \ / \ (S_{m-1})_j \leq (S_m)_j \leq (S_{m+1})_j \\ (S_{m+1})_j & \forall j \ / \ (S_{m+1})_j > (S_m)_j \end{cases}$$

3.2 Evolutionary algorithm

Many variants of EA have been defined in literature. Different implementations have been tested by the authors for the problem presented in this paper. Finally, the option

described in next sections has obtained best results. The main aspects defining an EA are: chromosomic structure of the individuals, generation of initial population, fitness measure to evaluate individuals, genetic operators to modify them and parameters to control the process.

3.2.1 Chromosomic representation of individuals.

An individual of the population represents an hypercube sequence in the coordination diagram. An hypercube sequence with n synchronization points is going to be represented by an ordered sequence S of length n with $S_1=(1,1,..^{R}..,1)$ and $S_n= (max_1,max_2,...,max_R)$. Notice that n is variable, that is a variable-length codification will be used. An individual is admissible if it forms an increasing Sequence of Free Hypercubes, otherwise it is a non-admissible individual. Figure 2 shows the chromosomic representation of an hypercube sequence in a two-dimensional example.

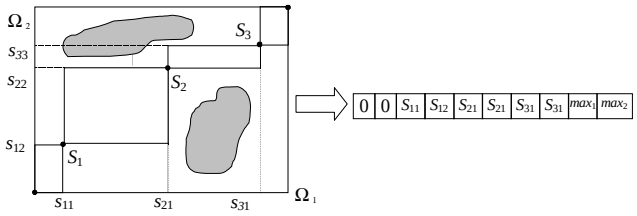

Figure 2 Chromosomic representation of an hypercube sequence

3.2.2 Generation of the initial population.

The initial population is randomly selected. Taking into account that the length of a sequence is variable, to obtain a initial population with a wide diversity of solutions, the following procedure is proposed:

A maximum number of points *NMAX* is established (only for the initial population), and the length of the sequences of the initial population is distributed with a random increasing probability between 1 and *NMAX*. Once the length of a sequence is selected, its constitutive points are obtained as follows:

> R sets of n random values in [0,1] are generated, then they are sorted in a increasing way and projected on $[0,max_1],...,[0,max_R]$ intervals .

3.2.3 Fitness measure.

The evaluation of the fitness measure will consider two different kind of individuals. If the hypercube sequence is a free collision one, a valid individuals is obtained, otherwise the individual is non-valid. In fact, two different fitness functions will be used. For valid individuals, the fitness function gives the total execution time needed by the robots to complete their paths, when the sinchronization points are placed in the positions defined by the individual specifications (See [Ridao, 1995] for a more detailed description).

The fitness function for non-valid individuals is completely different. The execution time cannot be used as a fitness measure, because this type of individual is not a solution of the problem. The function must measure how far it is from a valid individual. Obviously, the fitness value for this kind of individual must be higher than any valid individual value. The function considered is $f(N)=K+nco$, where K is a high value in respect to the value associated to the valid individuals, and nco is the number of obstacle cells inside the hypercube sequence.

3.2.4 Genetic operators

Four types of operators have been defined for the evolutionary process: a recombination operator or crossover, two types of mutation operators: local mutation (produce a slight modification of an individual), structural mutation (modify the individual structure) and finally, the reduction operator to eliminate multiple points of an individual. The following subsections formally describes these operators:

Crossover operator.

Given two individuals S and T :

crossover(S,T) = presequence(S,m) + postsequence(T,l)

with m and l randomly selected. Also, in order to obtain an admissible individual $S_m \leq T_l$ (Figure 3).

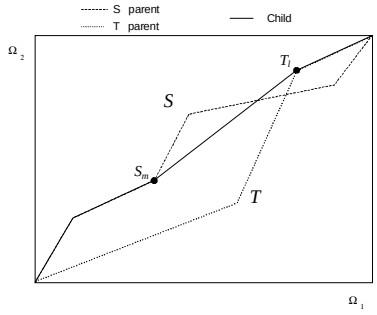

Figure 3 . Croosover operator

Local Mutation operator.

This mutation operator produces a slight change in an individual. The operation consist of applying to an individual S the operation movement(S,m,Q) where $\exists^{*}\ j$ / $q_j \neq 0$. That is, the movement is accomplished to only one of the coordinates of point S_m (Figure 4).

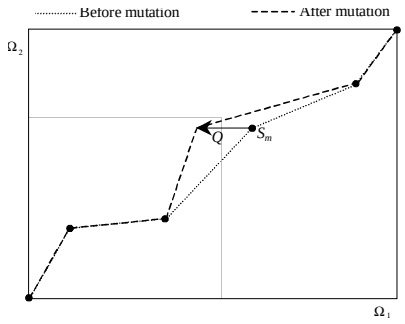

Figure 4 Local mutation

Structural mutation operators.
Structural mutations change the structure of individuals, that is, they change their number of points. Given an individual S and a position m, this operator gives a new individual S^f obtained with the following operations succession:

S'=movement(S,m,Q')
S''=movement(S',m+1,Q'')
S^f=insert(S'',m,Q)

where Q, Q' and Q'' are points randomly elected with the restriction $S''_m \leq Q \leq S''_{m+1}$ (Figure 5). Others mutations operator can be formed by the application of the insert or elimination operators to an individual.

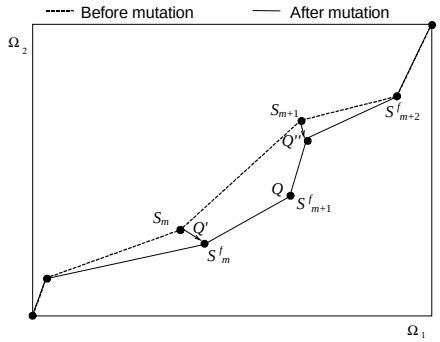

Figure 5 Structural mutation

Reduction operator.
The reduction operator acts on a ordered sequence, eliminating multiple points. That is, given a ordered sequence S and for every multiple point $<S_i,S_{i+1},...,S_{i+j}>$ the reduction operator transforms S into $S_R=<S_1,...,S_i,S_{i+j+1},...,S_n>$.

Control Parameters .
An elitist evolutionary algorithm has been used, where the best individual of each generation is replicated in the following one. A percentage of the offspring is obtained through parents mutations selected with a probability proportional to its fitness. The rest of the offspring is obtained through parents crossover , and then, one of above defined mutation operators is applied with a random probability.

3.3 Complete algorithm.
This section presents the complete proposed evolutionary algorithm:

```
PROCEDURE OPTIMIZATION
 Inicializate(population)
 FOR i=1 TO NumberOfGeneration
   Evaluate(population)
   population=evolution(population)
 ENDFOR
 Solution=Best(population)

 PROCEDURE EVOLUTION(P)
  Child={}
  Add Best(P) to Child
  FOR i=2 TO NumberOfReplica
     Add Select(P) to Child
  ENDFOR
  FOR i = NumberOfReplica TO SizeOfPopulation
     Add Crossover(Select(P),Select(P)) to Child
  FOR each C in Child
    IF IsThereMutation
       Substitute C in Child by
GenerateNeighbour(C)
      IF IsThereReduction
        Substitute C in Child by Reduction(C)
   ENDFOR
   RETURN(Child)

PROCEDURE GenerateNeighbour(Solution)
  Generate(ProbChange)
  IF ProbChange < ProbChangeLocal
    THEN RETURN LocalMutation(Solution)
     ELSE RETURN StructuralMutation(Solution)
```

5. APPLICATION EXAMPLES
The proposed algorithm has been implemented and applied to several examples in order to study its efficiency. The first example corresponds to the motion of two SCORBOT and sixteen collision regions and 180×180 cells (Figure 6).

Tests have been realised with a 100 individuals population. Other parameters are NumberOfReplica=10%, NMAX=10, mutation probabilities are 30% and reduction probability is 80%.

Results of the motion time for example 1 can be seen in Table 2. These values have been obtained for 100, 200, 300 and 500 generations of 100 individuals.

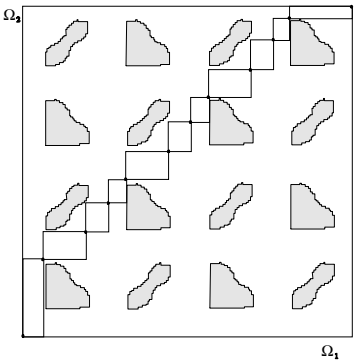

Figure 6 . Coordination Diagram of example 3

Table 1.- Example 1 results (in seconds)

Method	Avg.	Std.D	Min.
GA 100	44.98	1.34	42.35
GA 200	42.95	1.71	39.78
GA 300	41.19	1.16	38.63
GA 500	40.82	1.03	39.26

Finally, an application with three robots is presented. Initial and final configurations can be seen in Figure 7-1 and 7.5 respectively. The corresponding two-dimentional coordination diagrams are presented in Figure 8, and results are in Table 2 (same parameters as previous example). More then 200 individual do not significantly improve results. Both problems cannot be solved with an A^* algorithm.

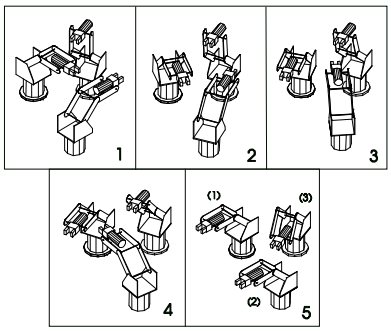

Figure 7 Initial (1), final (5) and intermediate position in example 4

6 CONCLUSIONS

This paper describes a method to generate collision-free coordinated motion plans in multirobots systems. The method tries to find a synchronization point sequence that minimizes the total execution motion time using an EA. The plans can be easily written in most industrial robot programming languages.

Table 2.- Example 2 results (in seconds)

Method	Avg.	Std.D	Min.
GA 100	8.95	1.12	8.37
GA 200	8.66	1.06	8.26

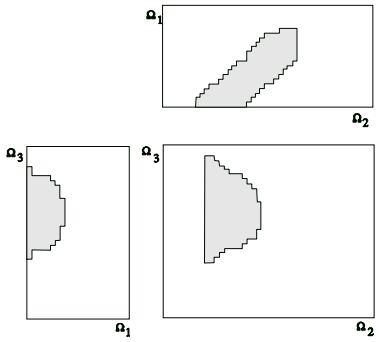

Figure 8 Two-dimentional coordination diagrams in example 4

7 REFERENCES

[1] J.C. Latombe, *Robot Motion Planning,* Klewer Academic Publishers, 1991.

[2] K. Kant and S.W. Zucker, *Toward Efficient Trajectory Planning: The Path-Velocity Decomposition.* Int. J. Robot. Research, 5 (3), pp 72-89. 1986.

[3] P.A. O´Donnell and T. Lozano-Pérez, *Deadlock-Free and Collision-Free Coordination of Two Robots Manipulators*, Proc. of the IEEE Int. Conf. Robotics and Automation, pp 484-489, 1989.

[4] J. Lee, *A dynamic programming approach to near minimun-time trajectory planning for two robots*. IEEE Trans.Rob.and.Autom.. Vol. 11 nº 1, pp 160-164. Febr 1995.

[5] D.E. Goldberg, *The design of innovation: Lessons from genetic algorithms, lesson for the real world.* Internal Report nº 98004, Illinois

[6] M.A. Ridao, *Generación Automática de Trayectorias Libres de Colisiones para Múltiples Robots Manipuladores*. Ph. D. Thesis. Universidad de Sevilla. 1995

[7] M.A. Ridao, J. Riquelme, E.F. Camacho and M.Toro. *An Evolutionary and Local Search Algorithm for Planning Two Manipulators Motion*. Lecture Notes in Artificial Intelligence 1416. Springer. 1998.