

An Evolutionary + Local Search Algorithm for Planning Two Manipulators Motion

M.A. Ridao¹, J. Riquelme², E.F. Camacho¹ and M. Toro²

¹ Dpto. Ingeniería de Sistemas y Automática. Universidad de Sevilla
{ridao, riquelme}@cartuja.us.es

² Dpto. Lenguajes y Sistemas Informáticos. Universidad de Sevilla
{riquelme, toro}@lsi.us.es

Abstract: A method based on the union of an Evolutionary Algorithm (EA) and a local search algorithm for obtaining coordinated motion plans of two manipulator robots is presented. A Decoupled Planning Approach has been used. For this purpose, the problem has been decomposed into two subproblems: path planning, where a collision-free path is found for each robot independently of the other, only considering fixed obstacles; and trajectory planning, where the paths are timed and synchronized in order to avoid collision with the other robot. This paper focuses on the second problem. 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. A hybrid technique with EA and local search methods has been implemented.

1 Introduction

The problem is to plan a collision-free motion (obstacles and other robots), from an initial configuration to a goal configuration. The most extended approach to this problem is to decompose it into two subproblems: path planning and trajectory planning. Many algorithms to solve this problem can be found in the literature [1], [2], [3], [4], [5].

The solution obtained by most of these algorithms is a robot trajectory. 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.

Different methods are presented in [6], [7] and [8] to minimize the total motion time of the robots along their paths, avoiding collision regardless of the accuracy of the dynamic model used. These methods are based on Evolutionary Algorithm (EA) to find a collision-free motion plan for two robots. Better results were achieved with integer and variable-length chromosome codification. EA, as numeric optimizers, are able to make fast search in broad spaces. However, their capability to realize local search is limited. In this paper, we propose an algorithm composed of an EA and a local search.

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 environments obstacles and themselves. The use of a Decoupled Planning approach needs a fixed obstacle collision-free path to be previously obtained for each of the robots. The paths which the robots are expected to follow are assumed to be given as a parameterized curve in the joint space, where λ parameter is the distance along the path. The *Coordination Space (CS)* is defined as the R^2 region:

$$CS = \{ (\lambda^1, \lambda^2) / 0 \leq \lambda^j \leq \lambda_{max}^j \text{ with } 1 \leq j \leq 2 \} \quad (1)$$

Any path from $(0,0)$ to $(\lambda_{max}^1, \lambda_{max}^2)$ determines a coordinated execution of the two paths, and it is called a *Coordination Path (CP)*. The *Collision Region (CR)* is defined as the set of points in CS where a collision between the two manipulators is produced. In order to reduce the search space in CS, a discretization of each path has to be made, so the path is divided into several equal intervals. Let's number the intervals of each path j from 1 to max_j and the ordered set of intervals is called Ω_j . A cell is defined as the subspace formed by one interval of the paths of each of the robots and is represented as the pair (n_1, n_2) . With these discretized paths, CS is transformed into an array of cells, the *Coordination Diagram (CD)*. A cell (n_1, n_2) is considered collision FREE if every point inside the cell does not belong to the collision region.

Robots can be synchronized using *Synchronization Points (SP)*, that is, a point in CD, which any CP will necessarily pass through. When the robot arrives at that place on its path, it will stop until the other robot arrives at its respective point. To avoid a collision it is possible to alter the CP defining the number and position of the SP. This constraint motion is very easy to implement using any robot programming language [6].

Let's consider a rectangle formed by free cells in CD 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 CD will always be a collision-free CP. This class of rectangles is going to be called *Free Rectangles*.

Let's 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 CD, and the upper right corner of the last rectangle is the upper right corner of CD. This set of rectangles is a *Free Rectangle Sequence*, and the intersection points between two rectangles will be the SP. The problem can be stated as that of finding a Free Rectangle Sequence that minimizes the total execution time necessary for the robots to complete their whole path. The main variables used to find this sequence are the number of SP and the position of these points in CD. A complete description of the method can be found in [6] and [7].

3 The Evolutionary Algorithm

As it was mentioned above, the optimization algorithm is decomposed in two parts: an evolutionary algorithm, that it is explained in this section, and a local search algorithm, that it will be study in the next section.

3.1 Chromosomic representation of individuals

Each individual is represented by an increasing SP sequence with an integer codification [9]. Chromosomes representing each solution have variable length as it is proposed in [10]. If n is the length of a chromosome, then a SP sequence will be determined by $n+2$ synchronization points, where $(x_0, y_0) = (0, 0)$ and $(x_{n+1}, y_{n+1}) = (max_1, max_2)$ and with $\{P_i = (x_i, y_i) / \text{for all } i \ x_i \leq x_{i+1} \text{ and } y_i \leq y_{i+1}\} \ 0 \leq i \leq n$ where x_i, y_i are intervals corresponding to the paths of the first and second robot respectively.

3.2 Generation of the initial population

Obviously, the initial population is selected randomly. Since the number of SP of individuals is variable and to obtain a initial population with a wide diversity of solutions, the following procedure was proposed: a maximum number of points $NMAX$ is established (only for the initial population), and the number of SP of the individuals of the initial population is distributed in a random increasing probability distribution from 1 (minimum) to $NMAX$ (maximum way). 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 $[0, max_1]$ and $[0, max_2]$ intervals respectively.

3.3 Fitness measure

For valid individuals, the fitness function gives the total execution time needed by the robots to complete their paths, when the SP are placed in the positions defined by the individual specifications (See [6]). The fitness function for non-valid individuals must measure how far it is from a valid individual. 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 rectangle sequence.

3.4 Genetic operators

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

A SP of S^1 called P is randomly selected. Then, Q another synchronization point of S^2 is selected. Q is the first randomly obtained SP 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 SP sequence. It is possible that no point of S^2 verifies this limitation, then the

child is returned as a copy of S^1 . If any point Q exists, a new individual is formed by the first points of S^1 (P inclusive), followed by the points of S^2 from Q (inclusive) to the end of S^2 . After, all the synchronization points of the resulting individual with the same coordinate values are reduced to a single SP. That is, the same individual but with a more simplified structure.

Mutation Operator. Given an individual S formed by a sequence of n SP, another individual is formed by altering the genetic information of S . Two groups of mutation operators are proposed in this paper: *Slight Operators*, and *Strong Operators*, which modify the individual in a more substantial way, even the structure of the solution.

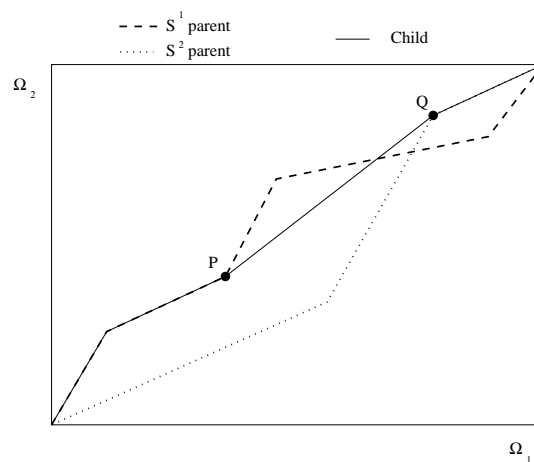


Figure 1.- Crossover Operator

Experimental results show that the most important improvements are obtained with the slight mutation. Nevertheless, the strong mutation plays a very important role in the process, avoiding the algorithm to remain trapped in local minima.

Slight mutations. A SP (x_k, y_k) of individual S is randomly selected and also an integer m between $MUTMAX$ and $-MUTMAX$ is chosen too, where $MUTMAX$ is the maximum permitted mutation. The mutation consists of substituting the point (x_k, y_k) by (x_k+m, y_k+m) . This mutation has two variants. In the first one, different m values for each coordinate are selected, while in the second one, only one of the coordinates is modified.

Strong mutations. Several strong mutations have been implemented and tested with satisfactory results. The following mutations are considered in this paper:

Synchronization Point Modification. A SP (x_k, y_k) from sequence S is randomly selected. The mutation consists of substituting that point by another one chosen in the rectangle

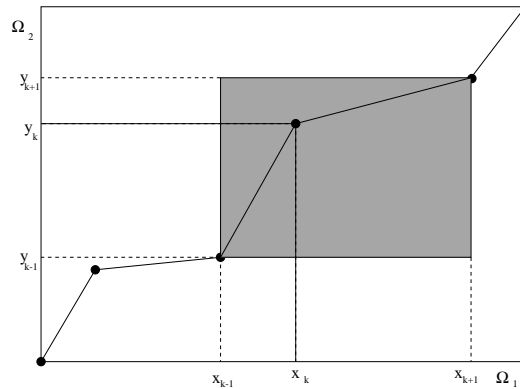


Figure 2.- Synchronization Point Modification

defined by the previous SP as the lower left corner and the following one as the upper right corner (if any of them does not exist, $(0,0)$ or (max_1, max_2) are considered respectively). That is, a pair of integers (x,y) is randomly chosen so that $x \in [x_{k-1}, x_{k+1}]$ and $y \in [y_{k-1}, y_{k+1}]$. Finally (x_k, y_k) is substituted by (x,y) , a point inside the gray area in Figure 2.

Synchronization Point Elimination. This mutation eliminates a randomly selected SP of S , that is, the new individual is identical to S , but with one SP less.

Segment Mutation. This mutation chooses two consecutive SP point $P=(x_k, y_k)$ and $P'=(x_{k+1}, y_{k+1})$, and a slight mutation is applied to them with a probability. Thereafter, this mutation adds a new SP between both of them. For this purpose, the differences $d_x=x_{k+1}-x_k$ and $d_y=y_{k+1}-y_k$ are obtained. Then, another two integers are chosen randomly (v_x between 1 and d_x-1 and v_y between 1 and d_y-1). The new point is located in S after P with coordinates (x_k+v_x, y_k+v_y) . (See Figure 3).

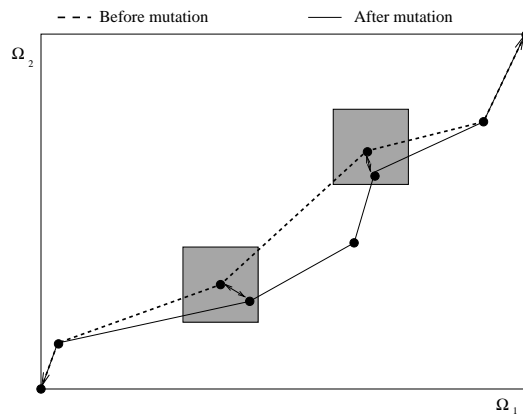


Figure 3.- Segment Mutation

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 $P=(x_k, y_k)$ is substituted by (y_k, x_k) .

3.5 Selection Mechanism

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 (also selected as a function of its fitness), and after, one of above defined mutation operators is applied with a random probability.

4 Local Search Algorithm

The realization of fast search in wide space is the main quality of Genetics Algorithms as function optimizers. However, their capability of complete local search is limited. Holland [11] suggested that genetic algorithms should be used as a preprocessor to perform the initial search, before tuning the search with local methods. Likewise, Grefenstette [12] point out that genetic algorithms are qualified to identify high performance regions of the search space and he recommend: "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 in [13]. If the local search do not obtain a optimum value, the authors apply a backtracking process to execute a new genetic simulation.

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

```
Procedure Random-Walk
  Generate(CurrentSolution)
  BestSolution = CurrentSolution
  REPEAT
    CurrentSolution •
  GenerateNeighbour(CurrentSolution)
    IF Objective(CurrentSolution) <
      Objective(BestSolution)
    THEN BestSolution • CurrentSolution
  UNTIL StopCriterium
```

We have used the previously defined mutation operators in order to implement the GenerateNeighbour 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 finding optimal solutions even starting with solutions with a non-optimum number of SP.

Most of the cases, the proposed hybrid technique obtain 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 local search starting solution 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 Generateneighbour(Solution)
  Generate(ProbChange)
  IF ProbChange < ProbChangeLocal
    THEN RETURN LocalMutation(Solution)
  ELSE RETURN StructuralMutation(Solution)

```

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 to be trapped in local minima after 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.

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 coordination motion of two PUMA-type articulated manipulators (Figure 4-a). The coordination diagram is 105×80 cells and the collision region is formed by a single connected region. This class of coordination diagram is very typical in multirobot applications. The example shown in Figure 4-b corresponds to the coordinated motions of two SCARA-type manipulators. Any path must verify the constraint consisting of passing through a narrow corridor.

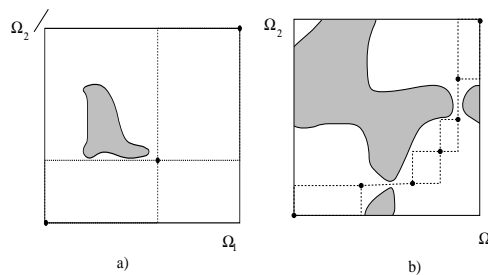


Figure 4.- Coordination Diagrams corresponding to examples 1 and 2

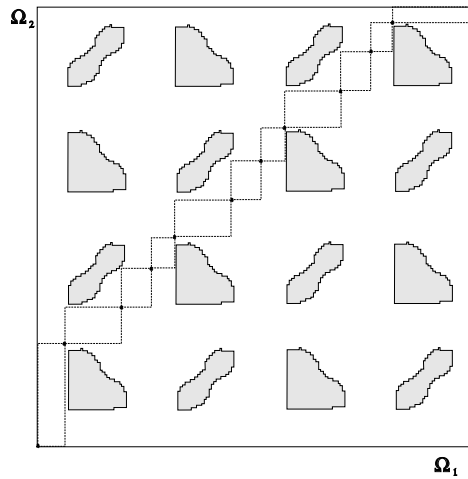


Figure 5.- Example 3 coordination diagram

The last example corresponds to the motion of two SCORBOT and 16 collision regions and 180×180 cells (Figure 5). It is an iterative motion represented in Figure 6. The motion from Figure 6-1 to Figure 6-2 and again to the initial configuration Figure 6-3, is repeated twice.

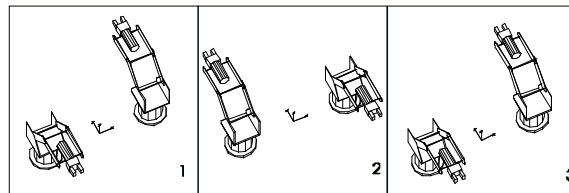


Figure 6.- Example 3 initial position (1), intermediate(2) and goal position (3)

The obtained results confirm the efficiency of the proposed approach. The found solutions reached for examples 1 and 2 can be observed in Table 1. The tests compare the results of the evolutionary algorithm without local search for 300 generations of 100 individuals, for 100 generations also without local search, and finally, the achieves values for a local search process beginning with the best individual of the generation 100. The stop criteria for random-walk were 5000 iterations. That is, 5000 calls to the evaluation function, equivalent to an additional computational cost of 50 generations of 100 individuals.

Table 1.- Example 1 and 2 Results

Ex.	Method	Avg.	•	Min.
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

The results are similar in example 1 and slightly better in example 2. Notice that GA 100 + rw process has a computational cost of the order of 50% of GA 300.

Table 2.- Example 3 results

Method	Avg.	•	Min.
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

The results for example 3 can be seen in Table 2. These values have been obtained for 100, 200, 300 and 500 generations of 100 individuals. After each evolutionary process, a local search has been executed with 5000 iterations starting with the provided best individual. These values clearly confirm the effectiveness of the proposed minimization

method. Notice that solution of GA 200 + rw is 10% better than GA 500, in spite of the fact that the computational cost are reduced in 50%

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. This hybrid technique gets better results than a pure evolutionary algorithm with a lower computational cost. Tests show that these benefits increase with the complexity of the problem

7 References

1. J.C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, 1991.
2. K. Kant and S.W. Zucker, *Toward Efficient Trajectory Planning: The Path-Velocity Decomposition*. The Int. Journal of Robotics Research, 5 (3), pp 72-89. 1986.
3. B.H. Lee and C.S.G. Lee, *Collision-Free Motion Planning of Two Robots*, IEEE Transactions on System, Man and Cyb., pp 21-32, Vol. SMC-17, no 1, Jan-Feb 1987.
4. P.A. O'Donnell and T. Lozano-Pérez, *Deadlock-Free and Collision-Free Coordination of Two Robots Manipulators*, Proc. of the IEEE I.C.Robotics Automation., pp 484-489, 1989.
5. Z. Bien and J. Lee, *A Minimum-Time Trajectory Planning Method for Two Robots*, IEEE Transactions on Robotics and Automation, pp 414-418, vol 8, no 3, June 1992.
6. M.A. Ridaó, *Generación Automática de Trayectorias Libres de Colisiones para Múltiples Robots Manipuladores*. Ph. D. Thesis. Universidad de Sevilla. 1995.
7. M.A. Ridaó, J. Riquelme, E.F. Camacho and M. Toro, *Coordinated Motion Planning of manipulators by evolution strategies*. Proceedings of the Tenth International Conference on Applications of Artificial Intelligence in Engineering. Udine (Italy). Julio 1995.
8. J. Riquelme, M.A. Ridaó, E.F. Camacho and M. Toro, *Using genetic algorithm with variable -length individuals for planning two manipulators motion*. Proceedings ICANNGA '97. Norwich (England) April, 1997.
9. Z. Michalewicz, *Genetic algorithm + Data structure = Evolution programs*. Second Edition, Springer-Verlag. 1994.
10. S.F. Smith, *A learning system based on genetic adaptive algorithms*. Ph. D. Thesis. University of Pittsburgh. 1980.
11. Holland J.H., *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, 1975.
12. Grefenstette J.J., *Incorporating problem specific knowledge into genetic algorithms*. Genetic Algorithms and simulated annealing, pp. 42-60, Ed. L. Davis, Morgan Kauffmann Publishers, 1987.
13. Syrjakow M. and H. Szczerbicka, *Optimization of Simulation models with REMO*. Proceedings of the conference on Modelling and Simulation, pp. 274-281, 1994.