A Direct Approach to Solving Trajectory Planning Problems Using Genetic Algorithms with Dynamics Considerations in Complex Environments

Fares J. Abu-Dakka[1], Francisco J. Valero[*], Jose Luis Suñer[*] and Vicente Mata[*]
[*] *Centro de Investigación en Tecnología de Vehiculos, U.P.V. Valencia, 46022, Spain*

Abstract

This paper presents a new genetic algorithm methodology to solve the trajectory planning problem. This methodology can obtain smooth trajectories for industrial robots in complex environments using a direct method. The algorithm simultaneously creates a collision-free trajectory between initial and final configurations as the robot moves. The presented method deals with the uncertainties associated with the unknown kinematic properties of intermediate via points since they are generated as the algorithm evolves looking for the solution. Additionally, the objective of this algorithm is to minimize the trajectory time, which guides the robot motion. The method has been applied successfully to the PUMA 560 robotic system. Four operational parameters (execution time, computational time, end-effector distance travelled and significant points distance travelled) have been computed to study and analyze the algorithm efficiency. The experimental results show that, the proposed optimization algorithm for the trajectory planning problem of an industrial robot is feasible.

Keywords: Robotics, Trajectory Planning, Obstacles Avoidance, Genetic Algorithms

1. Introduction

Minimum time trajectory planning for industrial robots has been addressed by numerous researchers motivated by the direct relationship between tasks being executed in minimum time and productivity in manufacturing systems. The resolution of efficient trajectory planning with prevention of collisions for robots in complex environments requires computationally costly algorithms that prevent their industrial application.

A clear difference exists between path and trajectory planning, as well as the algorithms used to solve these problems. In the case of path planning, the algorithms try to obtain a sequence of configurations between the initial and final ones, fulfilling some kinematic and geometrical constraints [1]. On the other

[1] Corresponding author. Address: CITV, Universidad Politécnica de Valencia, c/camino de vera s/n, 46022, Valencia, Spain. Tel: 0034 652972454; Fax: 0034 963877629.

E-mail addresses: fares.abudakka@gmail.com, fvalero@mcm.upv.es, josuner@mcm.upv.es, vmata@mcm.upv.es

Robotica / Volume 33 / Issue 03 / March 2015, pp 669-683

DOI: http://dx.doi.org/10.1017/S0263574714000393 (About DOI), Published online: 10 March 2014

hand, the algorithms for trajectory planning try to obtain a temporal history of the evolution of robot joint coordinates, by minimizing aspects such as time, power, and/or energy consumption.

Essentially, the trajectory planning has been analyzed using two different approaches: direct (global) and indirect (decoupled) approaches. Indirect approaches, first seek a path in the configuration space and then the trajectory adjusts subject to the dynamic constraints of the [2-9]. In the direct approaches, on the other hand, the search takes place in the system's state space. These approaches involve optimal control and numerical optimization [10-13]. The method used in this paper is oriented towards the attainment of an algorithm that avoids obtaining a prior path.

The main difficulties in finding an optimum trajectory are due to the system complexity, where the analytical solution may be intractable. While enumerative search methods are overwhelmed by the size of the search space, the genetic algorithm (GA) provides a more robust approach. GA was first introduced by Holland [14]. GA-based search and optimization techniques have recently found increasing use in machine learning, robot motion planning, scheduling, pattern recognition, image sensing, etc. [15].

Numerous implementations of GAs in the field of robot trajectory planning have been carried out by several researchers in recent decades. Toogood, et al., [16] developed a GA to find collision-free trajectories for the 3R robot with specific start and goal joint configurations, among known stationary obstacles. A new method for time-optimal motion planning based on improved GA was proposed, which incorporates kinematics, dynamics and control constraints of the robotic manipulator [17]. Rana and Zalzala [18] described a method to design a near time-optimal, collision-free motion in the case of multi-arm manipulators. The trajectory planning is carried out in the joint space, and the path is represented by knots connected through cubic splines. Monteiro and Madrid [19] have used GA to plan the stages of the trajectory of a robot arm called Jeca III. They have proposed the use of GA to plan a trajectory with obstacle avoidance and to implement joint space using classical GA. This is achieved in two stages: initial positioning, which locates the end-effector of the robot arm in the first point of the trajectory, and incremental positioning which moves the end-effector to the next point of the trajectory. Pires and Machado [20] proposed an algorithm containing a GA and a pattern search to design the optimal point-to-point trajectory planning for a planar 3-DOF manipulator. L. Tian and Collins [21] proposed a GA using a floating point representation to search for optimal end-effector trajectory for a manipulator. One year later, in 2004, they extended their work by developing a novel GAs for point obstacles avoidance trajectory using a cubic interpolation function [22]. Pires et al. [23] proposed a multi-objective GA, when

considering up to five simultaneous objectives, to generate manipulator trajectories and for obstacle avoidance. The authors in [24-26] use two evolutionary mutli-criteria algorithms, NSGA-II and MODE, to get optimal trajectory planning by minimizing travelling time, actuators energy and penalty for obstacle avoidance. A parallel populations GA procedure was presented to obtain a minimum time trajectory for a given sequence of configurations [27].

Direct methods usually start from knowledge of the initial state ($\mathbf{q}^{init}$ and $\dot{\mathbf{q}}^{init}$) and final state ($\mathbf{q}^{final}$ and $\dot{\mathbf{q}}^{final}$) of the robotic system. $\mathbf{q} = (q_1, q_2, \dots, q_i \dots, q_{dof})$ is the vector of the joint coordinates representing a generic configuration of the system, $\dot{\mathbf{q}}$ is the vector of their respective velocities, and i = 1, 2, 3, …, *dof* (degrees of freedom of the robot). Direct methods are about the acquisition of the trajectory, avoiding collisions and considering dynamic constraints of the robot, usually minimum and maximum torques $[\tau_i^{min}, \tau_i^{max}]$ in the actuators, setting the minimization of the execution time or energy consumed as the objective. The solution results in the time history of the joint coordinates $\mathbf{q}(t)$ and torques $\boldsymbol{\tau}(t)$ in the actuators.

The authors in [10] the authors proposed to directly parameterize the joint evolution vector $\mathbf{q}(t)$ using clamped cubic spline functions between an initial configuration, C^{init}, and a final one, C^{final}, using free nodes uniformly distributed between them and transforming the problem into a parametric constrained optimization problem which is solved for the unknown transfer time T and the unknown parameters of $\mathbf{q}(t)$ and positions of nodes. A concatenation of fifth-order polynomials to provide a smooth trajectory between two way points with jerk limits has been described [28]. The method requires the computation of the quintic control points. A minimum-jerk 3D model is also used to obtain the desired path in Cartesian space, which is widely used in the prediction of human reach movement. Instead of inverse kinematics, a direct optimization approach is used to predict each joint's profile (a spline curve) [11]. In these last examples, the obstacles are not addressed since the motion is path-constrained, that is, the whole path is parameterized previously (cubic splines, 5th polynomials, B-splines) in terms of passing points and the corresponding method is used not only to optimize the jerk, the time required or the energy consumed but also to obtain the unknown characteristics of the path (coefficients and positions of via points).

In this paper, a direct method is presented using GAs to obtain minimum time trajectories for industrial robots (at least 6 *dof*) working in complex environments and in which the intermediate configurations are unknown, i.e., no assumptions are previously made for the path. For example, to move the robot throughout the workspace (by means of adjacent configurations (AC)); a GA technique is used to ensure

continuous connections with minimum time between via points. This algorithm evaluates the time t_j for each step j in which the problem is solved. These times are progressively added up, so that when the final configuration is reached, the complete trajectory also will be defined. The minimum time t_j between two ACs is obtained by solving an optimization problem using steady state GAs, in which a fundamental role is played by the parameterization functions and the inverse dynamic problem, see Section 4. This approach deals with two facts: the obstacles in the workspace and unknown intermediate configurations. These two facts lead to uncertainties about the kinematic characteristics of intermediate points, highlighting that the knowing of these kinematic characteristics are indispensable to solving the inverse dynamic problem.

The algorithm in this paper works on a discretized configuration space which is generated simultaneously as the direct procedure evolves, demanding less computational effort than the corresponding indirect procedure [29].

The paper consists of eight sections. Section 2 briefly presents the GA procedure. Section 3 introduces the robot and workspace modeling. After that, the acquisition of the ACs with dynamic compatibilities is presented in section 4. The trajectories generation is covered in section 5. In section 6, a detailed explanation about the GA operators is introduced. Results of the experimental evaluation are given in section 7, followed by final remarks and conclusions in section 8.

2. Genetic algorithm procedure

In principle, GAs are stochastic search algorithms analogous to natural selection. They combine survival of the fittest among the string structures with a structured yet randomized information exchange to form a search algorithm with the innovative flair of natural evolution. GAs have proven their robustness and usefulness over other search techniques because of their unique procedures that differ from other normal search and optimization techniques in four distinct ways:

1. GAs work with coding of a parameter set, not the parameters themselves.
2. GAs use probabilistic transition rules, not deterministic rules.
3. GAs can be used when no information is available about the gradient of the function at the evaluated points. The function itself does not need to be continuous or differentiable.
4. The function is not examined at a single place, constructing a possible path to the local maximum or minimum, but many different places are considered simultaneously. The function must be calculated for all elements of the population. The creation of new populations also requires

additional calculations. In this way the optimum of the function is sought in several directions simultaneously and many paths to the optimum are processed in parallel.

The calculations required for GAs are obviously much more extensive than for a simple random search. However, compared to other stochastic methods, GAs have the advantage that they can be parallelized with little effort. Since the calculations of the function on all points of a population are independent from each other, they can be carried out in several processors [30]. A clear improvement in performance can be achieved with them in comparison to other non-parallelizable optimization methods. Compared to purely local methods (e.g., gradient descent) GAs have the advantage that they do not necessarily remain trapped in a suboptimal local maximum or minimum of the target function. Since information from many different regions is used, a GA can move away from a local maximum or minimum if the population finds better function values in other areas of the definition domain [31].

The search technique in GAs consists of generating an initial population of strings at random. Each solution is assigned a numerical evaluation of its fitness by an objective function, which is a mathematical function that maps a particular solution to a single positive number that is a measure of the solution's worth. During each iteration (generation), each individual string in the current population is evaluated using this measure of fitness. New strings (children) for the next generation are selected from the current population of strings (parents) by a process known as selection. A random selection process is used with a higher probability given for strings with higher fitness values. Such a selection scheme systematically eliminates low-fitness individuals from the population from one generation to the next. New generations can be produced either synchronously, so that the old generation is completely replaced, or asynchronously, so that the generations overlap.

In this paper, two optimization processes using GAs are involved. One optimization process is constructed to obtain the ACs [32] and the other one is used to obtain the overall trajectory simultaneously that ensures continuous connections for velocities and accelerations between intermediate configurations. The GA for ACs uses the technique of steady-state reproduction without duplicates. This technique creates a certain number of children to replace the parents in the population, but discards children which are duplicated by the current individuals in the population. On the other hand, the GA for the trajectory uses parallel populations with migration technique. The GA has multiple, independent populations. It creates the populations by cloning the genome or population that is passed to it when it is created. Each population evolves using the steady-state GA, but each generation some individuals migrate

from one population to another. The migration algorithm is deterministic stepping-stone; each population migrates a fixed number of its best individuals to its neighbor. The master population is updated each generation with the best individual from each population.

Two genetic operators, crossover and mutation, are probabilistically applied to create a new population of individuals. Parent individuals are selected as candidates for crossover or mutation using the roulette-wheel selection method. This method is based on the magnitude of the fitness score of an individual relative to the rest of the population. The higher score, the more likely it is that a given individual will be selected. GAs are domain independent because they require no explicit notion of a neighborhood. Hence, crossover and mutation may not always produce feasible solutions. Therefore, the feasibility of a newly created individual is ascertained before inserting it in the population to replace a parent string.

In the GA-based solution procedure, a number of new individuals are created at each iteration. The remaining individuals are obtained by deterministically copying the individuals with the top fitness from the previous generation.

3. Robot & workspace modeling

The robot has been modeled as a wire linkage [4]. This model involves rigid links joined by the corresponding kinematic joints. Furthermore, the robot configuration has been modeled as a function of joint variables $C(\mathbf{q})$.

The collision detection process between the robot and the obstacles will be realized in Cartesian coordinates. In this manner, the configuration C of a robot with *dof* degrees of freedom will be determined without ambiguity in Cartesian coordinates for a minimum of M points. These points are called significant points $\alpha_m(\mathbf{q})$ where m = 1, 2, …, M. Besides the significant points, in order to improve the efficiency of the algorithm some other points P_k called interesting points will be used, where k = 1, 2, …, K. Interesting points are useful to define the robot's links and their number K is dependent on the geometric characteristics of the robot. The coordinates of interesting points are obtained from the significant points and the geometric characteristics of the robot. All these points have been modeled as a function of generalized coordinates and expressed in Cartesian coordinates. Therefore, the robot configuration $C(\mathbf{q})$ has been converted to the Cartesian coordinates $C(\alpha_m, P_k)$ to facilitate the collision avoidance technique.

As an application example, the wired model of the PUMA 560 robot is shown in Figure 1 and also its four significant points $C(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ together with another four points of interest.

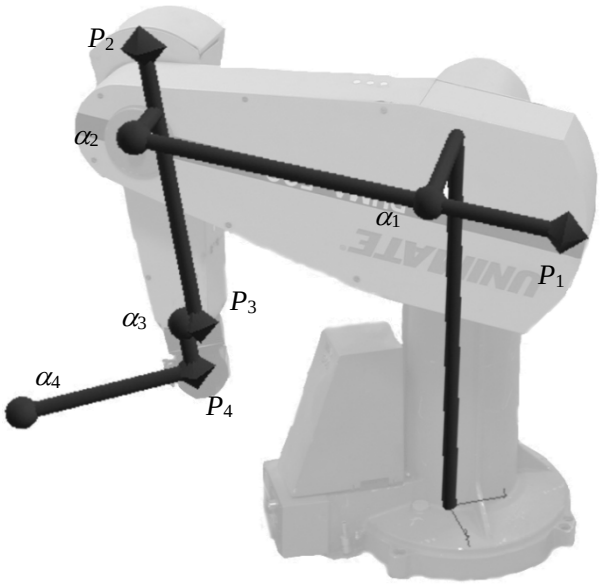

Figure 1: Robot wired model.

The workspace has been modeled in Cartesian coordinates. It is discretized according to increases (step size) into a rectangular prism with its axis parallel to the Cartesian reference system. The step size of the discretization should be smaller than the smallest obstacle size in the workspace. The end points of the diagonal of the prism represent the initial and final position of the robot end-effector of the initial and final configurations respectively [29]. The obstacles contained in the workspace are modeled in Cartesian coordinates, using a combination of three simple obstacle patterns: spheres, cylinders, and planar quadrilaterals since they are computationally simple [33]. Any types of obstacles can be modeled approximately using these simple patterns. This will help in the calculation of the distances between robot's links and obstacles and to avoid collisions. The growing obstacles technique has been used in order to obtain the actual dimensions of the robot [34]. The robot kinematics have been modeled in a recursive way based on a modified form of Denavit-Hartenberg's notation.

An important role is played by the generation of ACs using techniques described by [4, 32, 33]. The inverse dynamic problem has been solved using recursive Newton-Euler formulation to obtain the joint torques for a given set of joint angles, velocities, and accelerations [35].

4. Obtaining adjacent configurations with dynamic compatibility

The configuration space is generated by means of ACs. A new feasible configuration C^{j+1} is said to be adjacent to a given one C^j once it fulfills the following three conditions and has the proceeding property.

Conditions:

1. The position of the end effector of C^j is a one unit increment distance from the corresponding C^{j+1} and is located at some point in the discretized workspace.
2. No obstacle can be placed between the configurations C^j and C^{j+1}.
3. It minimizes the following multi-objective expression (1),

$$\left\|C^{j+1}-C^{j}\right\| = A\cdot t_{j,j+1}^{2} + B\cdot\sum_{i=1}^{dof}\left(q_i^{final}-q_i^{j+1}\right)^2 + C\cdot\begin{pmatrix}\left(\alpha_m^{j+1}-\alpha_m^{j}\right)_x^2 + \\ \sum_{m=1}^{M}\left(\alpha_m^{j+1}-\alpha_m^{j}\right)_y^2 + \\ \left(\alpha_m^{j+1}-\alpha_m^{j}\right)_z^2\end{pmatrix} \quad (1\textbf{Error! Bookmark not defined.})$$

Subjected to:

$$\text{Joint Torques: } \left|\tau_i(t)\right| \le \tau_i^{\max} \qquad i=1,\cdots,dof$$

where *A*, *B*, and *C* are three weighted empirical coefficients. In case of PUMA 560 robot, *dof* = 6 and *M* = 4. The first term is to minimize the time needed to move the robot from configuration C^j to C^{j+1}. The second term is to minimize the distance between the generalized coordinates of the obtained configuration C^{j+1} and the final one C^{final} to make it converge. The third one is to minimize the distance between the significant points of the current configuration C^j and the obtained one C^{j+1} and make them close.

Property:

The dynamic compatibility between the configurations C^j and C^{j+1} must be presented. To verify this property, the trajectory is adjusted by means of polynomial functions. This trajectory is subjected to the actuators constraints of the robot. The problem of obtaining the minimum time $t_{j,j+1}$ between ACs, is solved [32, 33]. In this paper, the trajectory has been modeled using cubic polynomials as shown in the following equation.

$$q_i^{j,j+1}(t) = a_i^{j,j+1} + b_i^{j,j+1}t + c_i^{j,j+1}t^2 + d_i^{j,j+1}t^3 \;\; ; \;\; \forall t \in \left[0, t_{j,j+1}\right]. \quad (2)$$

where $a_i^{j,j+1}$, $b_i^{j,j+1}$, $c_i^{j,j+1}$, and $d_i^{j,j+1}$ are the polynomial coefficients to move robot joints form configuration *j* to *j* + 1. $t_{j,j+1}$ is the time needed to move the robot form configuration *j* to *j* + 1.

The order in which the ACs are generated will condition the space of configurations generated and, therefore, the trajectory to be obtained, for more details about ACs readers may refer to [32, 33].

In this paper, the proposed procedure was applied to a PUMA 560 robot. The verification of the maximum torque in each actuator is done by dividing the interval $t_{j,j+1}$ and solving the corresponding inverse dynamic problem (IDP) [35].

5. Obtaining the trajectory

The determination of the overall trajectory from C^{init} to C^{final} is undertaken in this section. First of all, starting from C^{init}, a random search algorithm has been applied to look for the next AC as explained in section 4, then after the second AC a continuous connection between every two adjacent cubic polynomials takes place until completing the whole trajectory. Like this, a continuous trajectory of minimum time is adjusted directly.

The objective is to minimize the travelling time *T* between C^{init} and C^{final}.

$$T = \sum_{j=1}^{final} t_{j,j+1} \tag{3}$$

In each step of the evolution of the trajectory let's consider the trajectory fragment *j*-1 (between configurations $C^{j-1}(\mathbf{q})$ and $C^{j}(\mathbf{q})$) represented by Eq. (5) and the fragment *j* (between configurations $C^{j}(\mathbf{q})$ and $C^{j+1}(\mathbf{q})$) expressed in joint variables, Eq. (3)

$$q_i^{j-1,j}(t) = a_i^{j-1,j} + b_i^{j-1,j} t + c_i^{j-1,j} t^2 + d_i^{j-1,j} t^3 \;\; ; \;\; \forall t \in [0, t_{j-1,j}]. \tag{4}$$

where *j* is the *j*th configuration in the trajectory and *i* = 1, 2, …, *dof*. $t_{j-1,j}$ is the time needed to move the robot joints form configuration *j* – 1 to *j*.

To guarantee the smoothness of the whole path, the following conditions are imposed:

- Position;

$$\begin{cases} q_i^{init}(0) = a_i^{init} = q_i^{init} \\ q_i^{j-1,j}(t_{j-1,j}) = q_i^{j,j+1}(0) \\ q_i^{final}(t_{final-1,final}) = q_i^{final} \end{cases} \tag{5}$$

- Velocity;

  The velocity at the beginning (C^{init}) and the end (C^{final}) of the trajectory must be zero, while between the intermediate configurations, the velocity at the end of the fragment *j* – 1 must be coincident with the initial of the following one.

$$\begin{cases} \dot{q}_i^{init}(0) = 0 \\ \dot{q}_i^{j-1,j}(t_{j-1,j}) = \dot{q}_i^{j,j+1}(0) \\ \dot{q}_i^{final}(t_{final-1,final}) = 0 \end{cases} \tag{6}$$

- Acceleration;

Throughout intermediate configurations, the final acceleration of the fragment *j* – 1 must be coincident with the initial acceleration of the following one.

$$\ddot{q}_i^{j-1,j}(t_{j-1,j}) = \ddot{q}_i^{j,j+1}(0) \tag{7}$$

Figure 2 shows how the kinematic parameters (position, velocity and acceleration) and the torques in actuators 1, 2 and 3 evolve. The graphs correspond to the example No. 1 (the case with zero obstacles), which is detailed in Section 7. The torques in the actuators are limited due to the following values: $\tau_1 \in [-140, 140]$ N.m, $\tau_2 \in [-180, 180]$ N.m, $\tau_3 \in [-140, 140]$ N.m, $\tau_4 \in [-80, 80]$ N.m, $\tau_5 \in [-80, 80]$ N.m, $\tau_6 \in [-40, 40]$ N.m.

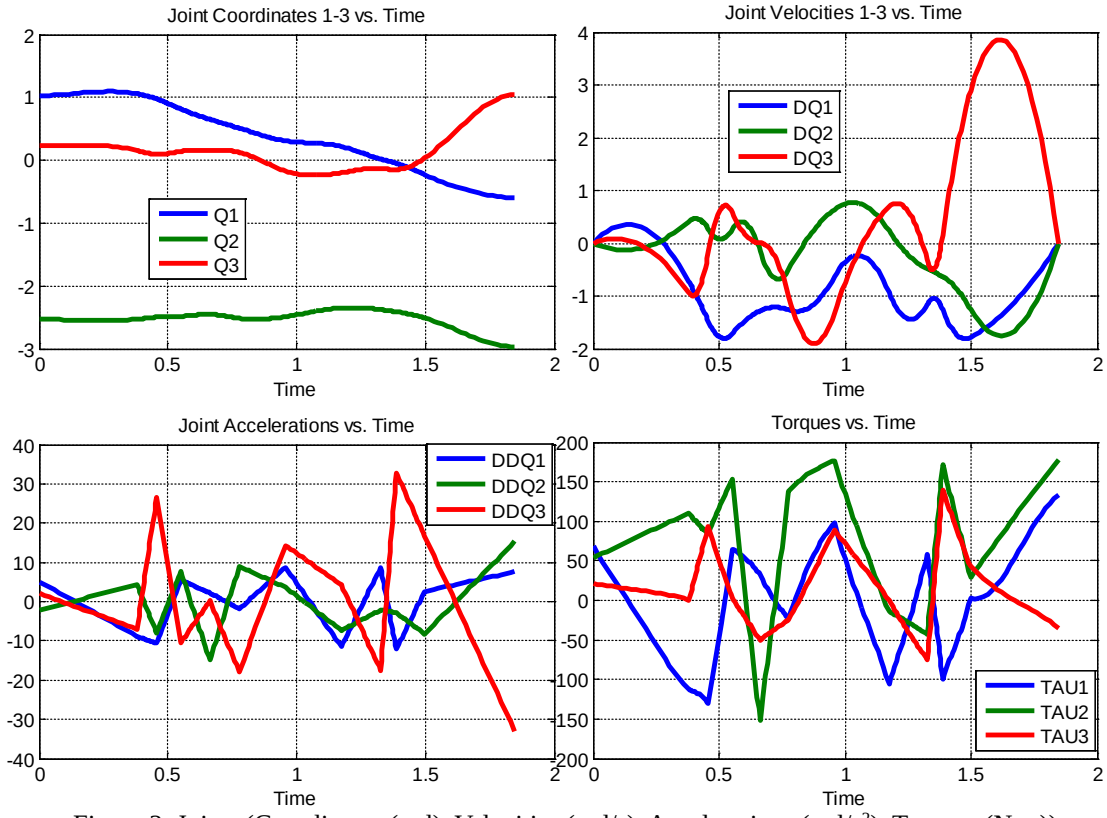

Figure 2. Joints (Coordinates (rad), Velocities (rad/s), Accelerations (rad/s^2), Torques (N.m)) vs. Execution Time (s).

6. Genetic algorithms operators and parameters

In this paper, the GA uses real representation scheme to encode the trajectory variables. The main operators and characteristics in the exposed GA are explained as the following:

- **Individual:**

The individual or the chromosome is a complete trajectory between C^{init} and C^{final}. Each chromosome is composed of a set of genes. Each gene contains the robot configuration $C^j(\mathbf{q})$, and the time needed to move the robot to this configuration. See Figure 3.

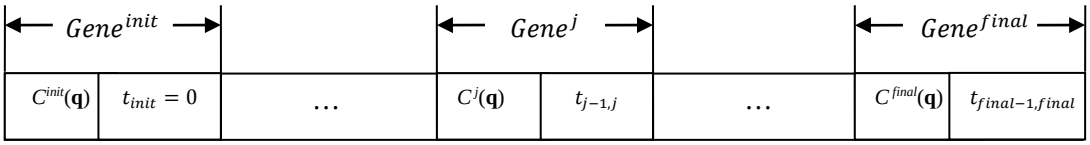

Figure 3: GA Chromosome

The first gene of each individual $Gene^{init}$ contains the initial configuration C^{init} data. Then the ramification to construct the chromosome will be started by selecting randomly the next gene; based on the random search algorithm; by calling the AC builder algorithm, and so on until the final configuration C^{final} reached, represented by $Gene^{final}$. The ramification can be done without repetition in 7 directions; *X*, *Y*, *Z*, *XY*, *XZ*, *YZ*, or *XYZ* directions. In this paper, chromosomes can be with different lengths.

In chromosome construction process, if the algorithm can't find the next AC due to obstacles or dynamic incompatibility, a retuning back recursive technique will be applied. This technique tracks back looking for the last possible configuration in which the robot can continue from it. If the tracking back drives the search to the initial configuration, this means that there is no possible trajectory in the workspace. In this case, the algorithm extends the workspace and starts again.

- **Objective function:**

In this paper, as mentioned above there are three optimization algorithms: one deals with optimizing the trajectory between ACs; detailed in [32]. The second deals with optimizing the time trajectory composed of a set of optimized trajectories between ACs. Third, optimize the result of the second optimization procedure using clamped cubic spline. The objective is:

$$minimize \sum_{j=1}^{final} t_{j,j+1} \tag{8}$$

Subject to:

$$\text{Joint Torques: } |\tau_i(t)| \leq \tau_i^{\max} \qquad i = 1, \cdots, dof$$

- **Crossover:**

The crossover occurs randomly and only with some probability p_{cross}. The crossover is made through the exchange of a part of the trajectory (chromosome) between two selected trajectories through the selection operation mentioned earlier. This is done by searching groups of individuals that have been selected for crossover, and then, select pair of individuals randomly. In each pair, the algorithm searches the genes of each individual for the intersection configurations. The intersection in this case is to find a configuration p $\left(C_{Dad}^{p}\right)$ in the first trajectory (let's call it *Dad*) that can be adjacent to a configuration k (C_{Mom}^{k}) in the second trajectory (let's call it *Mom*). The algorithm looks for all possible intersections

between two selected chromosomes (trajectories) for crossover. i.e. given two trajectories: *Dad* with length *n* and *Mom* with length *m*.

$$Dad = C_{Dad}^{1} \cup C_{Dad}^{2} \cup \cdots \cup C_{Dad}^{p} \cup \cdots \cup C_{Dad}^{n} \tag{9}$$

$$Mom = C_{Mom}^{1} \cup C_{Mom}^{2} \cup \cdots \cup C_{Mom}^{k} \cup \cdots \cup C_{Mom}^{m} \tag{10}$$

$$Dad \cap Mom = \left\{ \left(C_{Dad}^{p}, C_{Mom}^{k}\right)_1, \left(C_{Dad}^{p}, C_{Mom}^{k}\right)_2, \cdots, \left(C_{Dad}^{p}, C_{Mom}^{k}\right)_l \right\} \tag{11}$$

where l = 0, 1, 2, …, n – 2 in case of *Dad* or m – 2 in case of *Mom*, is the number of ACs found.

The algorithm then will select one intersection randomly in case of many are found satisfying these criteria. The new offspring (trajectory) will be like this:

$$Offspring = C_{Dad}^{1} \cup C_{Dad}^{2} \cup \cdots \cup C_{Dad}^{p} \cup C_{Mom}^{k} \cup \cdots \cup C_{Mom}^{m} \tag{12}$$

If there are no such points, the crossover will be cancelled.

This means that the resulting trajectory (offspring) will consist of two parts: a part from *Dad* (from the initial configuration until the selected Configuration C_{Dad}^{p}), and a part from *Mom* (from C_{Mom}^{k} until the final configuration). This way for crossover doesn't need equal chromosomes lengths. This process is illustrated in 2-D in Figure 4:

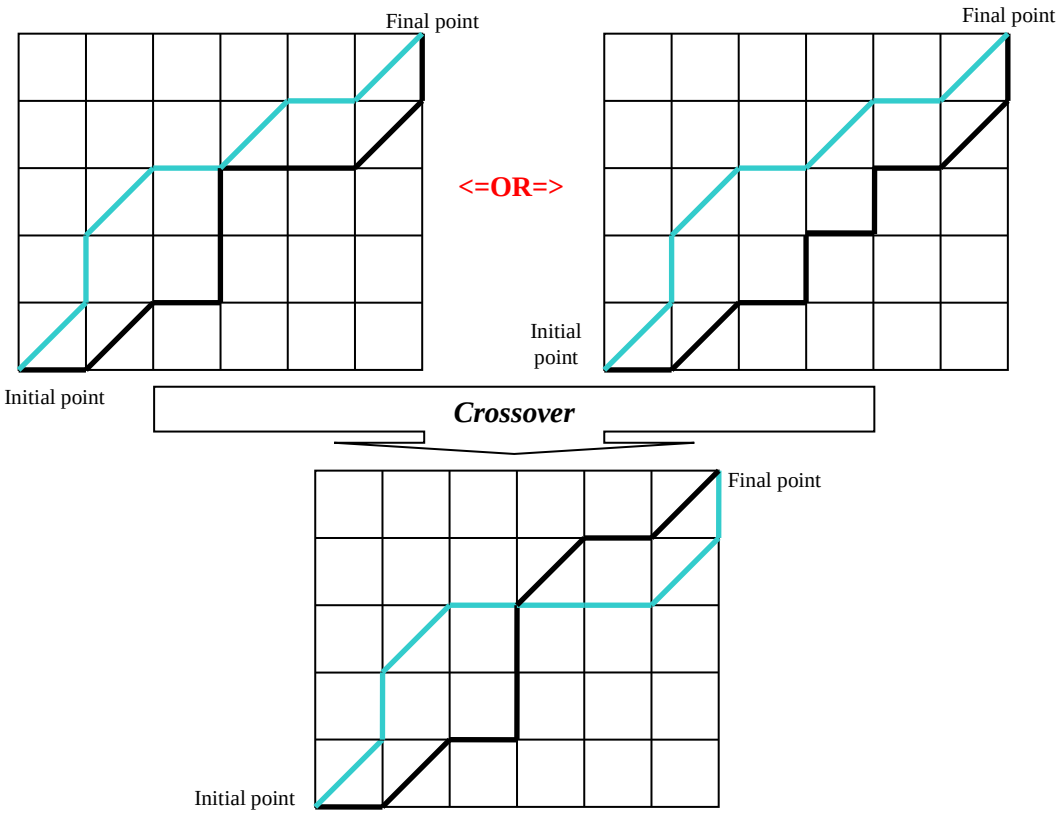

Figure 4: Crossover between two robot trajectories.

- **Mutation:**

Mutation is done by selecting a configuration (gene) randomly from a selected trajectory (chromosome). The first and the final configurations are not considered for mutation. The configuration is then compared to the previous and next configurations in the trajectory. All the possible changes with which the trajectory will remain incremental and quantum are applied to the configuration. To illustrate, let us consider three consecutive robot configurations C^{j-1}, C^{j}, C^{j+1} (three consecutive genes) in which their end-effector has the positions (0, 0, 0), (1, 0, 1), (1, 1, 2) with a step value of 1 in the *x*, *y* and *z*-coordinates. If mutation is to be applied on the C^{j}, where its end-effector position lies at (1, 0, 1), the algorithm will consider how each of the coordinates changed. The *x*-coordinate changed from 0 (previous position) to 1 and remained 1 in the next position. It is clear that changing the *x*-coordinate from 1 to 0 will not has the step size since the positions will become (0, 0, 0), (0, 0, 1), (1, 1, 2); i.e. *x*-coordinate changed from current to next while remained the same when going from the previous position to the current one. The same thing can be said about the *y*-coordinate, since it has not changed when going from the previous position to the current one while changed when going to the next position. The mutation will cause the *y*-coordinate to change from 0 to 1. Finally, the *z*-coordinate can't be modified since it changed from 0 to 1 to 2. If the mutation would change the *z*-coordinated to 0 or 2, the step would be greater than the predefined step. The mutation will not affect the coordinates that has not changed at all, for example the *x*-coordinate in (0,0,0),(0,0,1),(0,1,1) since any changes will result in the trajectory being invalid. For this new position, the ACs algorithm will take places to move the robot from the position (0, 0, 0) to (0, 0, 1) and then to (1, 1, 2).This process is illustrated in Figure 5.

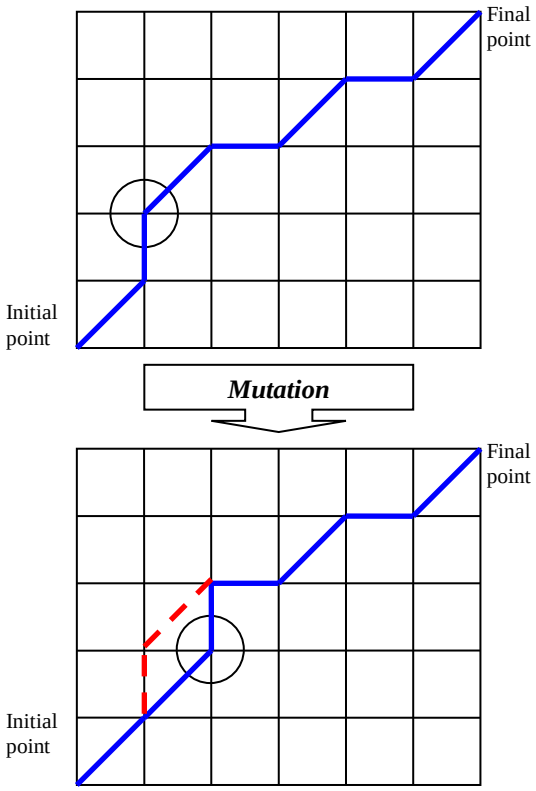

Figure 5: Mutation.

- **GA parameters:**

The control parameter values and terminating conditions used in the GA were selected based on several preliminary runs with alternate control parameters and terminating conditions on different instances of the problem. The next Figure 6, can demonstrate the necessary number of generations. The graph corresponds to example No. 1 (without obstacles), which is detailed in section 7.

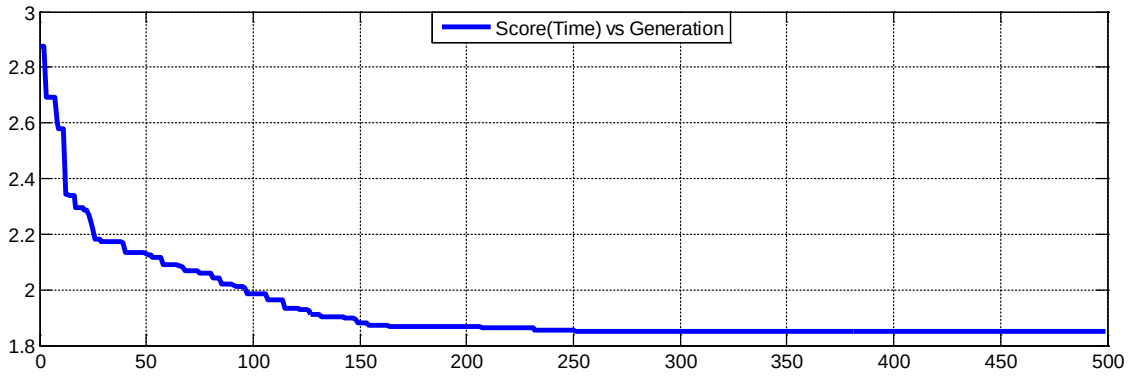

Figure 6: Objective function – time in (s) vs. No. of generations.

These values were used for the algorithm testing. The final parameter values used in the computational experiments for the GA procedure are summarized in Table 1.

Table 1: Parameter values for the genetic algorithm procedure.

Description	Value
Population size	20
Number of populations	3
Generation number	500
Crossover rate	0.95
Mutation rate	0.05
Percentage of solutions replaced by new generation	0.25

7. Application examples

The introduced procedure has been applied to a PUMA 560 robot using a computer with Intel Xeon CPU E5440 @ 2.83 GHz, 7.97 GB of RAM. For GA, the MIT GA Library [36] are used and adapted to the problem.

Four operational parameters have been studied when the procedure was applied to a numerous different examples. The parameters are:

a) Execution time: The time need to move the robot from the initial to the final configuration.

b) Computational time:

c) End-effector travelling distance.

d) Summation of significant points travelling distance (eq (13)).

$$\sum_{j=1}^{final}\sum_{m=1}^{4}\sqrt{\left(\alpha_m^{j+1}-\alpha_m^{j}\right)_x^2+\left(\alpha_m^{j+1}-\alpha_m^{j}\right)_y^2+\left(\alpha_m^{j+1}-\alpha_m^{j}\right)_z^2} \quad (13)$$

Let's consider three different detailed experiments, where given information about initial and final configurations and the workspace are shown below:

- **Experiment 1:**

This experiment demonstrates the effectiveness of the mentioned algorithm. The next Figure (7) shows the robot in the final configuration for 3 different runs for the same example in different environments.

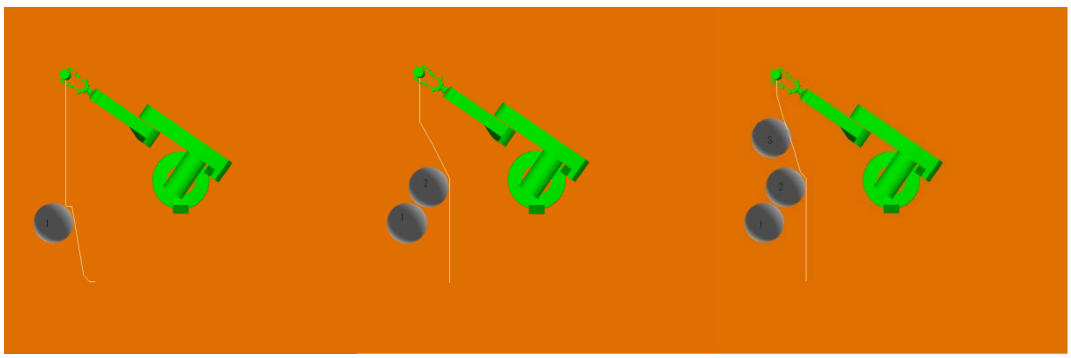

Figure 7: Case 1, top view of the workspace.

The robot initial and final configurations are shown in Table 2. Obstacles are shown in Table 3.

Table 2: Initial and Final Configurations for experiment 1.

Joint No.	Initial configuration	Joint No.	Final configuration
1	59.09º	**1**	-34.65º
2	-145.38º	**2**	-169.14º
3	13.03º	**3**	58.56º
4	1.13º	**4**	0.00º
5	31.68º	**5**	15.78º
6	0.00º	**6**	0.00º

Table 3: Obstacles locations (in m) for experiment 1.

0 Obstacle	1st Spherical obstacle	2nd Spherical obstacle	3rd Spherical obstacle
Centre	C_1^{SO} = (-0.90, -0.30, 0.50)	C_2^{SO} = (-0.75, 0.05, 0.50)	C_3^{SO} = (-0.85, 0.30, 0.50)
Radius	r_1^{SO} = 0.15	R_2^{SO} = 0.15	R_3^{SO} = 0.15

The results for this example in case of obstacle or without obstacles are shown in Table 4:

Table 4: Experiment 1 Results:

	Execution time (s)	Computational time (s)	End-effector travelling distance (m)	Significant points travelling distance (m)
0 Obstacles	1.84813	3792	1.5803	4.1060
1 Obs. Sphere	2.17046	3890	1.5904	4.0968
2 Obs. Spheres	2.41278	3800	1.5258	4.1385
3 Obs. Spheres	2.84737	4476	1.5803	4.6633

- **Experiment 2:**

Also, this experiment demonstrates the effectiveness of the mentioned algorithm. The next Figure (8) shows the robot in the final configuration for 2 different runs for the same. The left one is without obstacles and the right one is with a complex environment.

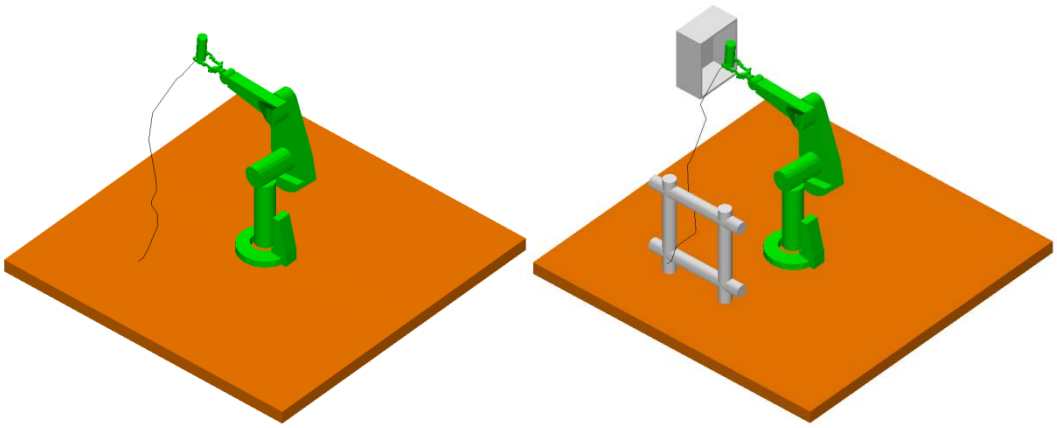

Figure 8. Case 2, workspace.

The robot initial and final configurations are shown in Table 5. Obstacles are shown in Table 6.

Table 5: Initial and Final Configurations for experiment 2.

Joint No.	Initial configuration	Joint No.	Final configuration
1	-7.50º	**1**	-95.10º
2	-174.80º	**2**	-101.20º
3	46.40º	**3**	15.59º
4	4.30º	**4**	0.00º
5	16.50º	**5**	0.00º
6	-6.50º	**6**	0.00º

Table 6: Obstacles locations (in m) for experiment 2.

	1st Cylindrical obstacle	2nd Cylindrical obstacle	3rd Cylindrical obstacle	4th Cylindrical obstacle
Centre 1	C_1^{Cyl} = (-0.7, 0.5, 0.0)	C_2^{Cyl} = (-0.7, 0.0, 0.0)	C_3^{Cyl} = (-0.7,-0.15, 0.7)	C_4^{Cyl} = (-0.7,-0.15, 0.15)
Centre 2	C_1^{Cyl} = (-0.7, 0.5, 0.8)	C_2^{Cyl} = (-0.7, 0.0, 0.8)	C_3^{Cyl} = (-0.7, 0.65, 0.7)	C_4^{Cyl} = (-0.7, 0.65, 0.15)
Radius	r_1^{Cyl} = 0.15	r_2^{Cyl} = 0.15	r_3^{Cyl} = 0.15	r_4^{Cyl} = 0.15
	1st Prismatic obstacle	**2nd Prismatic obstacle**	**3rd Prismatic obstacle**	**4th Prismatic obstacle**
Point 1	P_{11} = (0.31, 0.79, 1.42)	P_{21} = (0.31, 0.79, 1.42)	P_{31} = (-0.03, 0.79, 1.42)	P_{41} = (-0.03, 0.79, 0.97)
Point 2	P_{12} = (0.31, 0.99, 1.42)	P_{22} = (0.31, 0.99, 1.42)	P_{32} = (-0.03, 0.99, 1.42)	P_{42} = (-0.03, 0.99, 0.97)
Point 3	P_{13} = (0.31, 0.79, 0.97)	P_{23} = (-0.03, 0.99, 1.42)	P_{33} = (-0.03, 0.99, 0.97)	P_{43} = (0.31, 0.99, 0.97)
Point 4	P_{14} = (0.31, 0.99, 0.97)	P_{24} = (-0.03, 0.79, 1.42)	P_{34} = (-0.03, 0.79, 0.97)	P_{44} = (0.31, 0.79, 0.97)

The results for this example in case of obstacle or without obstacles are shown in Table 7:

Table 7: Experiment 2 results:

	Execution time (s)	Computational time (s)	End-effector travelling distance (m)	Significant points travelling distance (m)
0 Obstacles	3.75705	2156	1.7205	4.5340
With Obstacles	4.51356	5692	1.7340	4.9652

- Experiment **3:**

The next Figure 9 illustrates the time evolution for three more examples, which have three different initial and final configurations, Table 8, in different environments (number of obstacles).

Table 8: Initial and Final Configurations for experiment 3.

	Example 1		Example 2		Example 3	
Joint No.	**Initial configuration**	**Final configuration**	**Initial configuration**	**Final configuration**	**Initial configuration**	**Final configuration**
1	59.0º	-34.0º	-35.0º	-20.0º	-34.0º	40.10º
2	-145.0º	-170.0º	-170.0º	-115.0º	-170.0º	50.20º
3	13.0º	60.0º	30.0º	89.0º	60.0º	15.59º
4	0.0º	0.0º	0.0º	0.00º	0.0º	24.0º
5	31.0º	0.0º	0.0º	0.00º	0.0º	-79.0º
6	0.0º	0.0º	0.0º	0.00º	0.0º	12.0º

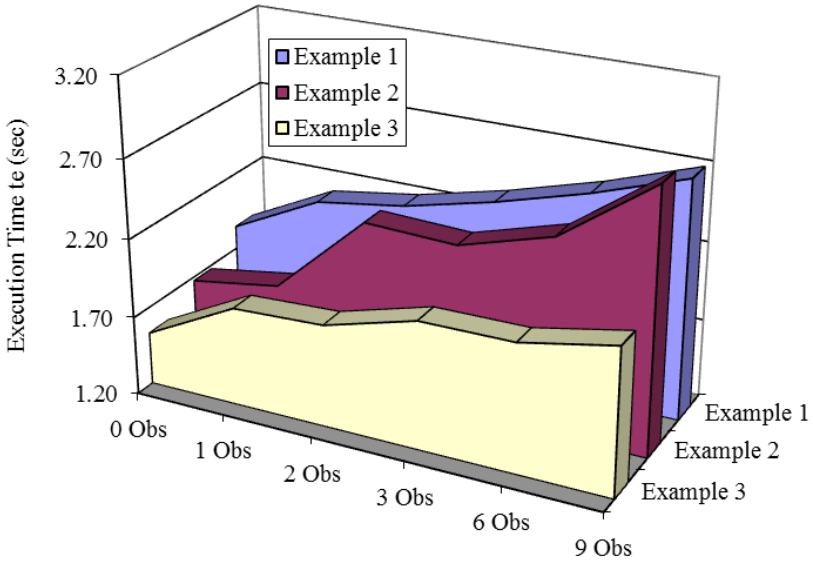

Figure 9: Time evolution in different examples with different environments.

The idea in this experiment is that we first run the algorithm when no obstacles in the workspace and then we recorded the trajectory and the four operational parameters. In the next run, we just added one obstacle colliding the path obtained from the first run and then recorded the trajectory and the four operational parameters. In the subsequent one, we added another obstacle colliding the new trajectory

solution generated in the second run and then we recorded the new trajectory and the four operational parameters. We repeated this until we added 9 obstacles in the workspace.

In the following Table 9, the four operational parameters are shown. Let's denote Execution time by "T" (sec), Computational time by "t" (sec), End-Effector travelling position by "d" (m), and significant points travelling position by "D" (m).

Table 9: Three different Examples with different environment conditions:

		Example 1	**Example 2**	**Example 3**
0 Obstacles	T (s)	1.84813	1.67483	1.53561
	t (s)	3792	3690	10667
	d (m)	1.5803	1.7223	1.3251
	D (m)	4.1060	3.4297	3.6164
1 Obstacles	T (s)	2.12692	1.76437	1.82668
	t (s)	3405	4111	15740
	d (m)	1.5904	1.7372	1.2901
	D (m)	4.0757	3.7056	3.6198
2 Obstacles	T (s)	2.21279	2.29341	1.85097
	t (s)	4719	4997	9382
	d (m)	1.6071	1.6707	1.2856
	D (m)	4.1867	4.3172	3.5747
3 Obstacles	T (s)	2.35064	2.26978	2.01012
	t (s)	6426	13704	17911
	d (m)	1.6784	1.7647	1.3686
	D (m)	4.3513	4.1418	3.8453
6 Obstacles	T (s)	2.51637	2.43976	2.01315
	t (s)	6101	15596	10593
	d (m)	1.8762	1.8674	1.3838
	D (m)	4.4731	4.2774	4.0285
9 Obstacles	T (s)	2.71562	2.86387	2.12885
	t (s)	5695	12874	15040
	d (m)	1.8678	1.9420	1.3838
	D (m)	4.5879	4.4529	4.0823

8. Conclusions

In this paper, a new methodology using GA has been presented for a direct approach trajectory planner in which the trajectory has been obtained simultaneously as the search algorithm evolves. The trajectories between ACs have been modeled in cubic polynomials, and discretized to solve the inverse dynamic problem and to validate the dynamics restrictions. Random search algorithms with new crossover and mutation operators have been introduced. The results obtained have been analyzed and studied on base of four operational parameters: (1) Execution time. (2) Computational time. (3) End-effector travelling distance. (4) Summation of significant points travelling distance, Eq. (13).

From the conducted analysis it's possible to conclude the follow:

a) The introduced algorithm provides a solution for the trajectory planning problem for industrial robots in complex environments.

b) The computational time is far greater than the execution time, which discarded the possibility of

using the algorithm in real time. However, the off-line trajectory planning is justified as a large number of robotic applications works in a repetitive manner. The main issue in these cases is to minimize the operation time of the robot as much as possible and so increase the production.

c) Observing Figure 2, the torques values are saturated which implies that the obtained time is near to optimal.

d) Moreover, it can be observed in Table 9 that the four operational parameters values are increasing by increasing the environment complexity.

References

[1] F. J. Abu-Dakka, F. Valero, and V. Mata, "Evolutionary Path Planning Algorithm for Industrial Robots," *Advanced Robotics*, vol. 26, pp. 1369-1392, 2012.

[2] F. J. Abu-Dakka, F. Rubio, F. Valero, and V. Mata, "Evolutionary Indirect Approach to Solving Trajectory Planning Problem for Industrial Robots Operating in Workspaces with Obstacles," *European Journal of Mechanics - A/Solids*, vol. Available online 19 June 2013, 2013.

[3] S. F. P. Saramago and V. S. Jr., "Trajectory Modeling of Robot Manipulators in the Presence of Obstacles," *Journal of Optimization Theory and Applications*, vol. 110, pp. 17-34, 2001.

[4] F. Valero, V. Mata, J. I. Cuadrado, and M. Ceccarelli, "A formulation for path planning of manipulators in complex environments by using adjacent configurations," *Advanced Robotics*, vol. 11, pp. 33-56, 1996.

[5] L. J. d. Plessis and J. A. Snyman, "Trajectory-planning through interpolation by overlapping cubic arcs and cubic splines," *International Journal for Numerical Methods in Engineering*, vol. 57, pp. 1615–1641, 2003.

[6] A. Piazzi and A. Visioli, "A global optimization approach to trajectory planning for industrial robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems IROS '97*, Grenoble, 1997, pp. 1553-1559.

[7] A. Piazzi and A. Visioli, "Global minimum-jerk trajectory planning of robot manipulators," *IEEE Transactions on Industrial Electronics*, vol. 47, pp. 140-149, 2000.

[8] E. Bertolazzi, F. Biral, and M. D. Lio, "real-time motion planning for multibody systems," *Multibody System Dynamics*, vol. 17, pp. 119-139, 2007.

[9] S. Behzadipour and A. Khajepour, "Time-optimal trajectory planning in cable-based manipulators," *IEEE Transactions on Robotics*, vol. 22, pp. 559-563, 2006.

[10] T. Chettibi, H. E. Lehtihet, M. Haddad, and S. Hanchi, "Minimum cost trajectory planning for industrial robots," *European Journal of Mechanics - A/Solids*, vol. 23, pp. 703-715, 2004.

[11] K. Abdel-malek, Z. Mi, J. Yang, and K. Nebel, "Optimization-based trajectory planning of the human upper body," *Robotica*, vol. 24, pp. 683-696, 2006.

[12] D. Constantinescu and E. A. Croft, "Smooth and time-optimal trajectory planning for industrial manipulators along specified paths," *Journal of Robotic Syatems*, vol. 17, pp. 233-249, 2000.

[13] F. J. Abu-Dakka, "Trajectory planning for industrial robot using genetic algorithms," Ph.D., Ingeniería Mecánica y de Materiales, Universitat Politècnica de València, 2011.

[14] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*, Second (Fisrt Edition, 1975) ed. Cambridge, MA, USA: MIT Press, 1975/1992.

[15] Y. Davidor, *Genetic Algorithms and Robotics: A Heuristic Strategy for Optimization*: World Scientific, 1991.

[16] R. Toogood, H. Hao, and C. Wong, "Robot path planning using genetic algorithms," in *IEEE International Conference on Intelligent Systems for the 21st Century*, Vancouver, BC, 1995, pp. 489-494.

[17] W.-M. Yun and Y.-G. Xi, "Optimum motion planning in joint space for robots using genetic algorithms," *Robotics and Autonomous Systems*, vol. 18, pp. 373–393, 1996.

[18] A. S. Rana and A. M. S. Zalzala, "An evolutionary planner for near time-optimal collision-free motion of multi-arm robotic manipulators," in *International Conference on Control '96, UKACC*, 1996, pp. 29-35.

[19] D. C. Monteiro and M. K. Madrid, "Planning of robot trajectories with genetic algorithms," in *The First Workshop on Robot Motion and Control, RoMoCo '99*, Kiekrz, 1999, pp. 223-228.
[20] E. J. S. Pires, J. A. T. Machado, and P. B. d. M. Oliveira, "An Evolutionary Approach to Robot Structure and Trajectory," in *ICAR'01, 10th Internat. Conf. on Advanced Robotics*, Budapest, 2001.
[21] L. Tian and C. Collins, "Motion Planning for Redundant Manipulators Using a Floating Point Genetic Algorithm," *Journal of Intelligent and Robotic Systems*, vol. 38, pp. 297-312, 2003.
[22] L. Tian and C. Collins, "An effective robot trajectory planningmethod using a genetic algorithm," *Mechatronics*, vol. 14, pp. 455-470, 2004.
[23] E. J. S. Pires, P. B. d. M. Oliveira, and J. A. T. Machado, "Manipulator trajectory planning using a MOEA," *Applied Soft Computing*, vol. 7, pp. 659-667, 2007.
[24] R. Saravanan and S. Ramabalan, "Evolutionary Minimum Cost Trajectory Planning for Industrial Robots," *Journal of Intelligent and Robotic Systems*, vol. 52, pp. 45-77, 2008.
[25] R. Saravanan, S. Ramabalan, and C. Balamurugan, "Evolutionary multi-criteria trajectory modeling of industrial robots in the presence of obstacles," *Engineering Applications of Artificial Intelligence*, vol. 22, pp. 329-342, 2009.
[26] R. Saravanan, S. Ramabalan, C. Balamurugan, and A. Subash, "Evolutionary trajectory planning for an industrial robot," *International Journal of Automation and Computing*, vol. 7, pp. 190-198, 2010.
[27] F. J. Abu-Dakka, I. F. Assad, F. Valero, and V. Mata, "Parallel-Populations Genetic Algorithm for the Optimization of Cubic Polynomial Joint Trajectories for Industrial Robots," in *Intelligent Robotics and Applications*, 2011.
[28] S. Macfarlane and E. A. Croft, "Jerk-bounded manipulator trajectory planning: design for real-time applications," *IEEE Transactions on Industrial Electronics on Robotics and Automation*, vol. 19, pp. 42-52, 2003.
[29] F. Valero, V. Mata, and A. Besa, "Trajectory planning in workspaces with obstacles taking into account the dynamic robot behaviour," *Mechanism and Machine Theory*, vol. 41, pp. 525-536, 2006.
[30] M. Gorges-Schleuter, "ASPARAGOS an asynchronous parallel genetic optimization strategy," in *The third international conference on Genetic algorithms*, George Mason University, USA, 1989, pp. 422-427.
[31] R. Rojas, *Neutral Networks: A Systematic Introduction*: Springer-Verlag New York Incorporated, 1996.
[32] F. J. Abu-Dakka, F. Valero, and V. Mata, "Obtaining Adjacent Configurations with Minimum Time Considering Robot Dynamics Using Genetic Algorithm," in *The 17th International Workshop on Robotics in Alpe-Adria-Danube Region RAAD2008*, Ancona, Italy, 2008.
[33] F. J. Abu-Dakka, F. Valero, A. Tubaileh, and F. Rubio, "Obtaining Adjacent Configurations with Minimum Time Considering Robot Dynamics," in *The 12th World Congress in Mechanism and Machine Science, IFToMM*, Besançon, France, 2007.
[34] T. Lozano-Pérez and M. A. Wesley, "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," *Magazine Communications of the ACM*, vol. 22, pp. 560-570, 1979.
[35] J. J. Craig, *Introduction to Robotics Mechanics and Control*, Second ed.: Addison-Wesley Publishing Company, 2005.
[36] M. Wall. (1996). *GAlib, A C++ Library of Genetic Algorithm Components*. Available: http://lancet.mit.edu/ga