Evolutionary Path Planning Algorithm for Industrial Robots

FARES J. ABU-DAKKA, FRANCISCO VALERO and VICENTE MATA

Centro de Investigación de Tecnología de Vehículos, Edificio 5E, Universidad Politécnica de Valencia,
C/camino de vera s/n, 46022, Valencia, Spain. faab@doctor.upv.es, fvalero, vmata@mcm.upv.es.

Abstract

This paper proposed a new methodology to solve collision free path planning problem for industrial robot using genetic algorithms. The method poses an optimization problem that aims to minimize the significant points traveling distance of the robot. The behaviour of more two operational parameters – the end effector traveling distance and computational time – is analysed. This algorithm is able to obtain the solution for any industrial robot working in the complex environments, just it needs to choose a suitable significant points for that robot. An application example has been illustrated using robot Puma 560.

Keywords: Industrial Robots, Adjacent configurations, Path planning, collision avoidance

1. INTRODUCTION

In the last few decades, the number of robots has grown in many areas. Upon industrial applications (pick and place operations, assembly tasks, spray-painting, and many other tasks), robots also are used in surgery, agriculture, underwater, and for transportation. In some cases, it is required to control and program the robots in real-time. On the contrary, to meet demands on flexibility, quality, and efficiency in industrial systems, off-line programming is required.

Usually, path planning is distinguishes from trajectory planning: the first ones try, essentially, to obtain a sequence (a path) of robot configurations between an initial and goal configurations that fulfills some conditions, mainly, collision avoidance, geometrical constraints, etc. Whereas, the
second ones try to obtain a temporal history of the evolution for the robot joint coordinates, by minimizing aspects, such as; the required time or the energy consumption.

Path planning has important applications in many areas, for example, industrial robotics, autonomous systems, assembly planning and virtual prototyping [1], computer graphics simulations [2], and computer-aided drug design [3].

Path planning algorithms can be classified into two aspects [4]: global algorithms assume that the robot’s environment is completely known. Therefore, their strength is global path planning. On the other hand, local algorithms use only a small fraction of the world model to generate robot control.

One of the most general and simple ways to arrange the path planning problem is based on the utilization of artificial potential fields [5]. Kavraki and Latombe [6,7] introduced an approach for many degrees of freedom robots moving in static environments. Valero et al. [8,9] proposed a technique for collision-free path planning in complex environments based on the concept of adjacent configurations. Valero et al. [10] introduced a planner in which all the workspace were analyzed. In probabilistic roadmap [11], random configurations are generating and attempting to connect pairs of nearby configurations with a local planner that will connect pairs of configurations. Kavraki and Latombe [12] proposed a randomized method, which has been successfully applied for solving path planning problem for robots with 3–6 degrees of freedom operating in static environments. LaValle [13] proposed a new probabilistic technique for path planning called rapidly exploring random tree (RRT). LaValle and Kuffner [14] introduced a RRT-based approach to path planning that generated and connected two RRTs in a state space, which generalizes configuration space. Abu-Dakka et al. [15] introduced a method using genetic algorithm (GA) for such problem. Rubio et al. [16] presented an approach in which the search of the path is made in the state space of the robotic system, and it makes use of the information generated about the characteristics of the process, introducing graph techniques for branching. This paper results will be compared with Rubio results.
With the growing interest for more flexible and autonomous industrial robots, the need for automatic path planning and robust obstacle avoidance algorithms is becoming apparent. Several different procedures have been suggested as mentioned above. Here, a history of techniques for obstacle avoidance for path planning based on GA will be introduced. GAs were first introduced by Holland [17] based search and optimization techniques have recently found increasing use in machine learning, robot motion planning, scheduling, pattern recognition, image sensing, etc.

A GA technique was introduced to solve the inverse kinematic problem [18]. A path planning problem for a mobile manipulator system using GAs has been addressed [19]. The path planning problem has been examined in two-dimensional space by developing an off-line approach that use Cartesian space and a planner with three degrees of freedom [20]. In another interesting application, GAs have been applied to the learning of local robot navigation behaviors for a reactive control system [21]. The method was applied to a mobile robot simulation in a two-dimensional world with stationary obstacles and known start and goal positions. Vadakkepat et al. [22] combined GAs with the artificial potential field to derive optimal potential field functions. Tian and Collins [23] applied GA for path planning on two degree of freedom robotic system operates in two dimensional workspace with one point obstacle. A very interesting GA procedure although not corresponding exactly to our problem is presented by Vannoy and Xiao [24]. They aim to plan the motion of mobile manipulators in dynamic environments. Their goal is to reach a region, not a configuration. They work in very large workspaces and the time needed to run their trajectories is very high. In contraposition, this paper is oriented to manufacturing cells and highly controlled environments where time is of great importance for the cycle and the accuracy of operations performed by the robot.

In the proposed method, the positioning problem of the manipulator linkages will be considered and a global off-line algorithm has been implemented to minimize the distance between the initial and final configurations of the robot. The GA appears here to solve such problem by minimizing the
traveling distance of the end-effector and the significant points (Section 2.1) between the initial and final points avoiding obstacles. The workspace will be modeled in such way to provide a discrete configuration space based on the positions of the end-effector between the initial and final configurations.

In this procedure, two optimization processes using GAs are involved. The first one is for the obtaining of the adjacent configurations (Section 3). The order in which the adjacent configurations are generated will condition the space of configurations generated and, therefore, the path to be obtained. Second one, for the obtaining of the path which consist of a set of adjacent configurations.

2. ROBOT & WORKSPACE MODELLING

2.1. Robot modeling

The robotic system has been modelled as function of generalized coordinates and considered as a wired model. This model consists of rigid links joined together by the corresponding kinematics joints. Although, the robot configuration has been modelled as a function of joint variables \( C(q_i) \), the workspace and obstacles have been modelled in Cartesian coordinates to facilitate the definition of the whole collision avoidance process.

![Robot wired model](image)

Figure 1. Robot wired model.

The robot configuration can be expressed in Cartesian coordinates as a set of points called significant points \( y_{m}^{i}(q_i) \) and interesting points \( \lambda_{i}^{i}(q_i) \), see Figure (1). Significant points have been
modeled as a function of generalized coordinates and expressed in Cartesian coordinates to facilitate the formulation of the collision avoidance process. The selection of these points is made based on the degrees of freedom of the robot in the way that they should be minimum as much as possible to define sin ambiguity the configuration of the robot. It is important to emphasize that they do not constitute an independent set of coordinates. To improve the efficiency of the obstacle detection algorithm, interesting points \( \lambda_i(q_i) \) have been modeled as function of generalized coordinates and expressed in Cartesian coordinates. The interesting points’ coordinates are obtained from the significant points and the geometric characteristics of the robot. The robot configuration \( C'(q_i) \) has been converted to the Cartesian coordinates \( C'(j_{m}, \lambda_{k}) \) to facilitate the collision avoidance technique.

In the Figure (1), an application example is shown for robot Puma 560 with four significant points \( \gamma_m(q_i) \Rightarrow \{\gamma_{1}', \gamma_{2}', \gamma_{3}', \gamma_{4}'\} \) and four interesting points \( \lambda_k(q_i) \Rightarrow \{\lambda_{1}', \lambda_{2}', \lambda_{3}', \lambda_{4}'\} \).

2.2. Workspace modeling

The workspace is that space that contains at least a set of robot configurations obtained based on a discrete set of end-effector’s positions. To achieve that definition, let’s consider a rectangular prism between the initial \( C^i \) and goal \( C^f \) robot configurations. The end points of the prism’s diagonal (represented by \( \gamma_4^i \) and \( \gamma_4^f \) in Figure (2)) are corresponding to the positions in Cartesian coordinates of the end-effector of the initial \( C^i \) and final \( C^f \) respectively. The prism edges are parallel to the Cartesian reference system.

A uniform grid of points is considered inside the prism. These points are far a magnitude small enough (\( \Delta x, \Delta y, \Delta z \)) to prevent the existence of obstacles between two adjacent points in the grid. Thus, the workspace contains a discrete set of configurations such that the position of the end-effector for each configuration must belong to the previously defined grid. The set of positions that can be occupied by the robot’s end-effector inside the prism are restricted finite number of points provided by discretizing the prism according to the following increments:
\[ \Delta_x = \frac{\gamma_x^f - \gamma_x^i}{\text{Pts}_x - 1}; \quad \text{where } \text{Pts}_x = 1 + \text{ceil}\left(\frac{\gamma_x^f - \gamma_x^i}{D}\right) \]  

(1)

\[ \Delta_y = \frac{\gamma_y^f - \gamma_y^i}{\text{Pts}_y - 1}; \quad \text{where } \text{Pts}_y = 1 + \text{ceil}\left(\frac{\gamma_y^f - \gamma_y^i}{D}\right) \]  

(2)

\[ \Delta_z = \frac{\gamma_z^f - \gamma_z^i}{\text{Pts}_z - 1}; \quad \text{where } \text{Pts}_z = 1 + \text{ceil}\left(\frac{\gamma_z^f - \gamma_z^i}{D}\right) \]  

(3)

where \( \text{ceil}(\text{number}) \) returns the smallest integer value that is not less than that number. \( D \) is less than the size of the smallest obstacle in the workspace or less than the smallest robot’s link diameter (depends which is smaller). \( \{\text{Pts}_x - 1, \text{Pts}_y - 1, \text{Pts}_z - 1\} \) are the number of points steps that discretize the prism. The points \( \{\gamma_x^i, \gamma_y^i, \gamma_z^i\} \) and \( \{\gamma_x^f, \gamma_y^f, \gamma_z^f\} \) are the coordinates of the end-effector positions of the initial and final configuration. The workspace dimensions can be modified depends on the need.

2.3. Obstacles modeling

A generic obstacle models have been constructed in terms of a combination of three basic patterns: Spheres, cylisphere, and quadrilateral planes since they are computationally simple. Very little information has to be stored in order to fully define such elements. Any type of obstacle can be modelled one or set of these elements. The minimum distance is obtained among these basic elements and the robot’s links. In this paper, the obstacles are considered to be static, which means, the position and orientation no change with time. Moreover, growing obstacles technique has been [25].
3. ADJACENT CONFIGURATIONS FOR PATH PLANNING

In this scope, the process of generating a discrete space of configuration is presented. This space of configurations is based on the obtaining of adjacent configurations concerning kinematics compatibility and feasibility with collision avoidance regardless the dynamics concerns.

3.1. Adjacent Configurations Definition

The configuration $C^k$ is adjacent to a given configuration $C^p$, if they are feasible and the three following conditions are satisfied:

a) The end-effector position $\gamma_4$ corresponds to a point of the discrete workspace. In addition, it is one increment far from the point corresponding to the $C^p$ configuration, so it is said that, the two configurations are neighboring and there must be a given increment between them less than the smallest obstacle size in the workspace.

b) Verify the absence of obstacles between adjacent configurations $C^k$ and $C^p$. Also, to verify that the distance between significant points meet the following condition,

$$\left|\gamma_i^p - \gamma_i^k\right| \leq 2 \cdot \min(r_j); \quad i = 1,2,3; \quad j = 1,2,...$$  \hspace{1cm} (4)

Figure 3: The obstacle three basic elements.
where $r_j$ is the minimum characteristic dimension of the obstacles in the workspace.

c) $C^k$ should be such as to minimize the function:

$$
\|C^k - C^p\| = A \cdot \sum_{j=1}^{4} \left( \left( \gamma^k_j - \gamma^p_j \right)^2 \right) + B \cdot \sum_{i=1}^{6} \left( q_i^f - q_i^k \right)^2
$$

where $A,B$ are coefficient and the expression is expressed in Cartesian coordinates, which aims to minimize the distance between significant points and the distance between the joints values of the current configuration and the final global one.

Adjacent configurations for path planning concern in finding a set of via points that constructed the path. This path can be tracked to find an optimal time scaling subjected to the robot dynamics.

3.2. Obtaining the Configuration $C^k$

In the building process of the path a random search procedure will be applied to search from the $C^i$ for the next adjacent configuration and so on till it reaches the $C^f$. The mainly concerns in this part is finding a sequence of robot configurations between the initial and final configurations that fulfils the early listed conditions. A methodology of two distinct routines has been constructed to obtain a robot configuration $C^k$ adjacent to $C^p$. In first place, the inverse kinematic problem [26] will be used to fined the $C^k$ for a given $\gamma_4$. If the new configuration $C^k$ doesn't fulfill the condition, a genetic algorithm procedure will be used to solve the problem.

GA maintains a population of solutions or individuals throughout the search. It initializes the population with a pool of potential solutions to the problem and seeks to produce better solutions by combining the better of the existing ones through the use of genetic operators. Individuals are selected at each iteration through a selection scheme depends on the fitness value for each individual.

A Steady State Genetic Algorithm (SSGA) procedure is used to obtain a robot configuration $C^k$ adjacent to a given one $C^p$ considering the three conditions mentioned above. A SSGA use
overlapping populations. This means, the ability to specify how much of the population should be replaced in each generation. Newly generated offspring are added to the population, and then the worst individuals are destroyed.

- **Individual**

The individual or the chromosome represents the robot configuration. Each individual consists of six genes; the robot generalized coordinates \( q_i; i=1,2,...,6 \).

\[
\text{Individual} = \{q_1, q_2, q_3, q_4, q_5, q_6\}
\]  

(6)

The initial population consists of a defined number of individuals, in this case are 30. The initial values of each gene in the individual are selected randomly between the two limits of the generalized coordinates for that gene. For example:

\[
gene(i) = RV(q_{i,\text{min}}, q_{i,\text{max}}); \quad i = 1 \rightarrow 6
\]  

(7)

where \( RV = \) Random Value (between low and high)

In fact, this way of generating the genes value and then checking the validity of the resulting individual is computationally expensive. To improve that, by looking to the workspace modeling and the conditions to produce adjacent configurations, it will be concluded that the movement between the two configurations is small. Because of that the previous Equation can be modified. Consider that \( q_i^p \) is the given configuration and \( \Delta q \) is a small increment. Therefore, the new interval for each \( q \) can be calculated as the next flow chart indicates:
Figure 4: Joints Interval determination.

This means that the Equation (7) will be as follows:

\[
gene(i) = RV( q_{i,\text{min}}^{\text{new}}, q_{i,\text{max}}^{\text{new}}), \quad i = 1 \rightarrow 6
\]

where \( RV = \) Random Value (between \( low \) and \( high \))

- **Selection:**

A roulette-wheel selection method is applied to select individuals for crossover and mutation. This method based on the magnitude of the fitness score of an individual relative to the rest of the population. The higher score, the more likely an individual will be selected.

- **Crossover:**

The crossover operator defines the procedure for generating a child from two selected parents. A single point crossover used in this procedure, see Figure (5).

- **Mutation:**

The mutation operator defines the procedure for mutating each genome. In this procedure, an offspring will be selected randomly then a gene will be selected randomly from that offspring. This gene will be mutated with respect to the following equation.
\[ \text{gene}(i) = \text{gene}(i) + \text{RV}(q_{i,\text{min}}, q_{i,\text{max}}) \times [\text{RV}(q_{i,\text{min}}, q_{i,\text{max}}) - \text{RV}(q_{i,\text{min}}, q_{i,\text{max}})] \]  \tag{9}

where \( \text{RV} \) = Random Value (between low and high), \( i = 1 \rightarrow 6 \).

**Objective:**

Minimize Equation (5).

**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 final parameter values used in the computational experiments for the GA procedure are summarized in Table (1).

<table>
<thead>
<tr>
<th>Description</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>popsize</td>
<td>30</td>
</tr>
<tr>
<td>Generation number</td>
<td>ngen</td>
<td>100</td>
</tr>
<tr>
<td>Crossover rate</td>
<td>pcross</td>
<td>0.9</td>
</tr>
<tr>
<td>Mutation rate</td>
<td>pmut</td>
<td>0.1</td>
</tr>
<tr>
<td>Number of solutions replaced by new generation</td>
<td>nReplacement</td>
<td>0.25</td>
</tr>
</tbody>
</table>

4. **GA PROCEDURE FOR PATH PLANNING**

The search technique 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 on 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 selection scheme systematically eliminates low-fitness individuals from the population of one generation to the next.

New generations can be produced either synchronously, so that the old generation is completely replaced, or asynchronously, in which the generations overlap.

The genetic algorithm for path planning in this paper uses parallel populations with migration technique. The genetic algorithm has multiple, independent populations. It creates the populations by cloning the genome or population that you pass when you create it. Each population evolves using steady-state genetic algorithm, but at 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 best individual from each population.

Crossover and mutation operators are probabilistically applied to create a new population of individuals. Parents are selected as candidates for crossover or mutation using the roulette-wheel selection method. This method based on the magnitude of the fitness score of an individual relative to the rest of the population. The higher score, the more likely an 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.

4.1. GAs operators and parameters

The main operators and characteristics in the exposed GA are:

● Individual

The individual is composed of set of intermediate points (end-effector positions) including end points (initial and final position of the end-effector). Each individual represents a complete path between initial and final configurations. Each triplet cells consisting one point (the Cartesian coordinates of the end-effector) in the individual and considered as a gene.

\[
\text{Individual} = \{ (X_1, Y_1, Z_1)_1, (X_1, Y_1, Z_1)_2, \ldots, (X_1, Y_1, Z_1)_i, \ldots, (X_1, Y_1, Z_1)_f \} 
\]  

(10)

The first point of each individual is the initial position of the end-effector of \( C^i \). The second point will be selected randomly from the discretized workspace without repetition in one of seven directions; \( X, Y, Z, XY, XZ, YZ, \) and \( XYZ \) direction. This strategy will be repeated for the next point and so on till the goal position is achieved. This definition is based on the number of intermediate points that consisting the path, which means that paths have not equal lengths, which leads to more complexity in crossover and mutation.

● Objective function:

The objective of this optimization problem is to find the optimal path between initial and final configurations of the robot. The shortest path will be calculated by minimizing the sum of the straight line segments of the corresponding significant points of the robot from the initial to the final point.

\[
\text{Minimize} \left\{ \sum_{i=1}^{n-1} \sum_{j=1}^m \sqrt{(x_{j+i+1} - x_j)^2 + (y_{j+i+1} - y_j)^2 + (z_{j+i+1} - z_j)^2} \right\} 
\]  

(11)

where: \( j \) is the number of the significant points of the robot, and \( m = 4 \) for Puma 560 robot; the case demonstrated in this paper. \( i = 1, 2, \ldots, n \) is the number of robot configurations included in the path.

● Selection:
The selection operation is made using the roulette-wheel method.

- **Crossover:**

The crossover is made through the exchange of a part of the path (individual) between two selected paths through the selection operation mentioned earlier; being that it is executed only if the probability of the crossover is satisfied. 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 \((C_j^p)\) configuration \(p\) in path \(i\) that can be adjacent to a \((C_j^k)\) configuration \(k\) in the path \(j\).

The search for the adjacent configuration occurs in the positive direction. The algorithm looks for all possible intersections between two selected individuals (paths) for crossover. I.e., given two paths: 

\[
\text{Dad with length } n \text{ and } \text{Mom with length } m.
\]

\[
\text{Dad} = C_{\text{Dad}}^1 \cup C_{\text{Dad}}^2 \cup \ldots \cup C_{\text{Dad}}^i \cup \ldots \cup C_{\text{Dad}}^n
\]

\[
\text{Mom} = C_{\text{Mom}}^1 \cup C_{\text{Mom}}^2 \cup \ldots \cup C_{\text{Mom}}^j \cup \ldots \cup C_{\text{Mom}}^m
\]

\[
\text{Dad} \cap \text{Mom} = \left\{ C_{\text{Dad}}^1, C_{\text{Mom}}^p \right\}_h \cup \left\{ C_{\text{Dad}}^2, C_{\text{Mom}}^p \right\}_2 \cup \ldots \cup \left\{ C_{\text{Dad}}^i, C_{\text{Mom}}^p \right\}_l
\]

\[
\text{Mom} \cap \text{Dad} = \left\{ C_{\text{Mom}}^1, C_{\text{Dad}}^p \right\}_h \cup \left\{ C_{\text{Mom}}^2, C_{\text{Dad}}^p \right\}_2 \cup \ldots \cup \left\{ C_{\text{Mom}}^i, C_{\text{Dad}}^p \right\}_l
\]

where \(\left\{ C_{\text{Dad}}^1, C_{\text{Mom}}^p \right\}_h\) are adjacent configurations, \(l_1 = 0, 1, 2, \ldots, n-2\) in case of \(\text{Dad}\). \(l_2 = 0, 1, 2, \ldots, m-2\) in case of \(\text{Mom}\) number of adjacent configurations found.

This way of intersection means that \(\text{Mom} \cap \text{Dad}\) and \(\text{Dad} \cap \text{Mom}\) are not necessary to have equal lengths which leads to the possibility to produce only one offspring rather than two in some cases.

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

\[
\text{Offspring1} = \text{sis} = C_{\text{Dad}}^1 \cup C_{\text{Mom}}^2 \cup \ldots \cup C_{\text{Dad}}^i \cup C_{\text{Mom}}^j \cup \ldots \cup C_{\text{Mom}}^m
\]
Offspring \(2 = \text{bro} = C^1_{\text{Mom}} \cup C^2_{\text{Mom}} \cup \cdots \cup C^i_{\text{Mom}} \cup C^j_{\text{Dad}} \cup \cdots \cup C^n_{\text{Dad}} \) \hspace{1cm} (17)

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

This means that the resulting path (offspring) will consist of two parts: a part from Dad (from the initial configuration until the selected Configuration \(C^i\)), and a part from Mom (from \(C^j\) until the final configuration). This way for crossover doesn’t need equal individuals’ lengths. This process is illustrated in 2-D in Figure (6):

![Crossover diagram](image)

Figure 6: Crossover between two robot Paths.

**Mutation:**

Mutation is done by selecting a configuration (gene) randomly from a selected path (individual). The first and the final configurations are not considered for mutation. The configuration is then compared to the previous and next configurations in the path. All the possible changes with which the path will remain incremental and quantum are applied to the configuration. To illustrate, let us consider three consecutive robot configurations \(C^{i-1}, C^i, C^{i+1}\) (three consecutive genes) in which their end-effector have 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 \(C^i\), 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 affect the validity of the path since the positions will become (0, 0, 0), (0, 0, 1), (1, 1, 2); *i.e.* $x$-coordinate changed from current position 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 $z$-coordinate 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 invalid path. For this new position, the adjacent configuration 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 (7).

![Figure 7: Path GA Mutation.](image)

- **GA parameters:**

The final parameter values used in the computational experiments for the GA procedure are summarized in Table (2).

<table>
<thead>
<tr>
<th>Description</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>popsize</td>
<td>10</td>
</tr>
<tr>
<td>Nº of populations</td>
<td>numpop</td>
<td>3</td>
</tr>
</tbody>
</table>

- 16 - *Advanced Robotics (RSJ)*
5. APPLICATION EXAMPLES

In this Section, application examples “particularized” for robot PUMA 560 have been implemented and analysed to validate the efficiency of the proposed algorithms. The introduced procedure has been executed using a computer with Intel Xeon CPU E5440 @ 2.83 GHz, 7.97 GB of RAM.

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

a) The objective function: Equation (11), denoted by “$d_s$”.

b) End-effector travelling distance, denoted by “$d_e$”.

c) Computational time, denoted by “$t_c$”:

5.1. Example 1: Basic obstacle elements

In this example, the obstacles have been introduced in the workspace sequentially (from 0 Obstacle to 3 Obstacles). Each time the new obstacle as much as possible has to obstruct the path of the previous run. This example has been solved by Rubio [16]. Thus, a comparison results will be done.

The robot initial and final configurations are shown in Table (3). Obstacles are shown in Table (4), these obstacles are used to create 10 different environments, starting with the case without obstacles and then the cases of 1, 2, 3 obstacles for each obstacle type.

<table>
<thead>
<tr>
<th>Joint No.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial configuration</td>
<td>59.09°</td>
<td>-145.38°</td>
<td>13.03°</td>
<td>1.13°</td>
<td>31.68°</td>
<td>0.00°</td>
</tr>
<tr>
<td>Final configuration</td>
<td>-34.65°</td>
<td>-169.14°</td>
<td>58.56°</td>
<td>0.00°</td>
<td>15.78°</td>
<td>0.00°</td>
</tr>
</tbody>
</table>

Table 3: Initial and Final Configurations.
Table 4: Obstacles locations (in m).

<table>
<thead>
<tr>
<th></th>
<th>1st Spherical obstacle</th>
<th>2nd Spherical obstacle</th>
<th>3rd Spherical obstacle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centre</td>
<td>( C_1^{SO} = (-0.85,-0.40,0.50) )</td>
<td>( C_2^{SO} = (-0.75,0.00,0.50) )</td>
<td>( C_3^{SO} = (-0.60,0.20,0.30) )</td>
</tr>
<tr>
<td>Radius</td>
<td>( r_1^{SO} = 0.15 )</td>
<td>( r_2^{SO} = 0.15 )</td>
<td>( r_3^{SO} = 0.15 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1st Cylindrical obstacle</th>
<th>2nd Cylindrical obstacle</th>
<th>3rd Cylindrical obstacle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centre 1</td>
<td>( C_1^{Cyl} = (-0.85,-0.5,0.0) )</td>
<td>( C_2^{Cyl} = (-0.75,0.0,0.0) )</td>
<td>( C_3^{Cyl} = (-0.7,0.2,0.0) )</td>
</tr>
<tr>
<td>Centre 2</td>
<td>( C_1^{Cyl,2} = (-0.85,-0.5,2.0) )</td>
<td>( C_2^{Cyl,2} = (-0.75,0.0,2.0) )</td>
<td>( C_3^{Cyl,2} = (-0.7,0.2,2.0) )</td>
</tr>
<tr>
<td>Radius</td>
<td>( r_1^{Cyl} = 0.15 )</td>
<td>( r_2^{Cyl} = 0.15 )</td>
<td>( r_3^{Cyl} = 0.15 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1st Prismatic obstacle</th>
<th>2nd Prismatic obstacle</th>
<th>3rd Prismatic obstacle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point 1</td>
<td>( P_{11} = (-0.7, -0.35, 0.0) )</td>
<td>( P_{21} = (-0.5, 0.0, 0.0) )</td>
<td>( P_{31} = (-0.5, 0.3, 0.0) )</td>
</tr>
<tr>
<td>Point 2</td>
<td>( P_{12} = (-0.7, -0.35, 2.0) )</td>
<td>( P_{22} = (-0.5, 0.0, 2.0) )</td>
<td>( P_{32} = (-0.5, 0.3, 2.0) )</td>
</tr>
<tr>
<td>Point 3</td>
<td>( P_{13} = (-1.5, -0.35, 2.0) )</td>
<td>( P_{23} = (-1.3, 0.0, 2.0) )</td>
<td>( P_{33} = (-1.3, 0.3, 2.0) )</td>
</tr>
<tr>
<td>Point 4</td>
<td>( P_{14} = (-1.5, -0.35, 0.0) )</td>
<td>( P_{24} = (-1.3, 0.0, 0.0) )</td>
<td>( P_{34} = (-1.3, 0.3, 0.0) )</td>
</tr>
</tbody>
</table>

The next Figure (8) show the robot in the initial configuration for 3 different runs for the same example with environment modification. It’s shown clearly in the figures that the workspace dimensions can be modified as needed.

![Figure 8: The case of 3 Spherical, Cylispherical, Quadri-lateral Plane Obstacles.](image)

The numerical results of this example and the comparison with Rubio[16] results are shown in the next Table (6) and Figure(9). The column titled by Rubio [16] contains the results of 4 different
approaches: (1) In-direct algorithm: seq, (2) A*, (3) Uniform Cost (UC), and (4) Greedy (G). For more details about these approaches please refer to their article.

![Figure 9: $d_s$, $d_e$, and $t_c$ vs. Obstacles.](image)

Table 5: $d_e$ Results

<table>
<thead>
<tr>
<th></th>
<th>0 O</th>
<th>1 SO</th>
<th>2 SO</th>
<th>3 SO</th>
<th>1 CO</th>
<th>2 CO</th>
<th>3 CO</th>
<th>1 PO</th>
<th>2 PO</th>
<th>3 PO</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_e$</td>
<td>1.5199</td>
<td>1.5199</td>
<td>1.5346</td>
<td>1.5199</td>
<td>1.6149</td>
<td>1.5917</td>
<td>1.5364</td>
<td>2.1387</td>
<td>1.8353</td>
<td></td>
</tr>
</tbody>
</table>

Table 6: $d_s$, $t_c$ Results

<table>
<thead>
<tr>
<th></th>
<th>Results of this paper</th>
<th>Rubio [16] Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$d_s$ ($m$)</td>
<td>seq</td>
</tr>
<tr>
<td>0 Obstacles</td>
<td>$t_c$ ($s$)</td>
<td>5532</td>
</tr>
<tr>
<td>1 Spherical Obstacle</td>
<td>$d_s$ ($m$)</td>
<td>3.8029</td>
</tr>
<tr>
<td></td>
<td>$t_c$ ($s$)</td>
<td>10567</td>
</tr>
<tr>
<td>2 Spherical Obstacle</td>
<td>$d_s$ ($m$)</td>
<td>4.0187</td>
</tr>
<tr>
<td></td>
<td>$t_c$ ($s$)</td>
<td>3806</td>
</tr>
<tr>
<td>3 Spherical Obstacle</td>
<td>$d_s$ ($m$)</td>
<td>4.1585</td>
</tr>
<tr>
<td></td>
<td>$t_c$ ($s$)</td>
<td>4013</td>
</tr>
<tr>
<td>1 Cylindrical Obstacle</td>
<td>$d_s$ ($m$)</td>
<td>3.7692</td>
</tr>
<tr>
<td></td>
<td>$t_c$ ($s$)</td>
<td>3932</td>
</tr>
<tr>
<td>2 Cylindrical Obstacle</td>
<td>$d_s$ ($m$)</td>
<td>4.1915</td>
</tr>
<tr>
<td></td>
<td>$t_c$ ($s$)</td>
<td>8139</td>
</tr>
<tr>
<td>3 Cylindrical Obstacle</td>
<td>$d_s$ ($m$)</td>
<td>4.3138</td>
</tr>
<tr>
<td></td>
<td>$t_c$ ($s$)</td>
<td>8366</td>
</tr>
<tr>
<td>1 Quadri-lateral Plane</td>
<td>$d_s$ ($m$)</td>
<td>3.9348</td>
</tr>
<tr>
<td>Plane Obstacle</td>
<td>$t_c$ ($s$)</td>
<td>10272</td>
</tr>
<tr>
<td>2 Quadri-lateral Plane Obstacle</td>
<td>$d_1$ (m)</td>
<td>4.5412</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td>$t_{c1}$ (s)</td>
<td>19523</td>
<td>198.45</td>
</tr>
<tr>
<td>3 Quadri-lateral Plane Obstacle</td>
<td>$d_2$ (m)</td>
<td>4.9628</td>
</tr>
<tr>
<td>$t_{c2}$ (s)</td>
<td>24322</td>
<td>1676.13</td>
</tr>
</tbody>
</table>

It’s shown clearly in the results that the objective function $d_s$ is increasing by increasing the complexity of the problem. In addition the capability of the algorithm to modify the dimensions of the workspace if it’s necessary can be observed clearly in Figure (8).

5.2. Example 2:

In this scope, a group of 50 examples with different initial and final configurations and different obstacles will be discussed. These examples have been solved by Rubio [16].

Figure 10: Travelled distance comparison between GA procedure and Rubio procedures.
Figure 11: Travelled distance comparison between GA procedure & Seq, A*, UC and G procedures produced by Rubio [16], respectively.

Figure 12: Computational time comparison between GA procedure and Rubio procedure.

5.3. Example 3: Industrial application

The robot initial and final configurations are shown in Table (6). Obstacles are shown in Table (7).

<table>
<thead>
<tr>
<th>Joint No.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial configuration</td>
<td>-7.50º</td>
<td>-174.80º</td>
<td>46.40º</td>
<td>4.30º</td>
<td>16.50º</td>
<td>-6.50º</td>
</tr>
<tr>
<td>Final configuration</td>
<td>-95.10º</td>
<td>-101.20º</td>
<td>15.59º</td>
<td>0.00º</td>
<td>0.00º</td>
<td>0.00º</td>
</tr>
</tbody>
</table>

Table 7: Initial and Final Configurations.

<table>
<thead>
<tr>
<th>Centre 1</th>
<th>1st Cylindrical obstacle</th>
<th>2nd Cylindrical obstacle</th>
<th>3rd Cylindrical obstacle</th>
<th>4th Cylindrical obstacle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centre 2</td>
<td>1st Cylindrical obstacle</td>
<td>2nd Cylindrical obstacle</td>
<td>3rd Cylindrical obstacle</td>
<td>4th Cylindrical obstacle</td>
</tr>
<tr>
<td>Radius</td>
<td>0.15</td>
<td>0.15</td>
<td>0.15</td>
<td>0.15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Centre 1</th>
<th>1st Cylindrical obstacle</th>
<th>2nd Cylindrical obstacle</th>
<th>3rd Cylindrical obstacle</th>
<th>4th Cylindrical obstacle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centre 2</td>
<td>1st Cylindrical obstacle</td>
<td>2nd Cylindrical obstacle</td>
<td>3rd Cylindrical obstacle</td>
<td>4th Cylindrical obstacle</td>
</tr>
<tr>
<td>Radius</td>
<td>0.15</td>
<td>0.15</td>
<td>0.15</td>
<td>0.15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1st Prismatic obstacle</th>
<th>2nd Prismatic obstacle</th>
<th>3rd Prismatic obstacle</th>
<th>4th Prismatic obstacle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point 1</td>
<td>0.31,0.79,1.42</td>
<td>-0.03,0.79,1.42</td>
<td>-0.03,0.79,0.97</td>
</tr>
<tr>
<td>Point 2</td>
<td>0.31,0.99,1.42</td>
<td>-0.03,0.99,1.42</td>
<td>-0.03,0.99,0.97</td>
</tr>
</tbody>
</table>

Table 8: Obstacles locations (in m).
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 next Figure (13) shows the path evolution from the initial robot configuration to the final configuration in a complex environment.

![Path evolution](image)

Table 9: Results:

<table>
<thead>
<tr>
<th></th>
<th>( d_i ) (m)</th>
<th>( d_f ) (m)</th>
<th>( t_e ) (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Obstacles</td>
<td>4.2936</td>
<td>1.8067</td>
<td>2156</td>
</tr>
<tr>
<td>With Obstacles</td>
<td>4.4467</td>
<td>1.7175</td>
<td>11366</td>
</tr>
</tbody>
</table>

6. CONCLUSIONS

In this paper, a new methodology using Gas has been presented for a global approach path planner in which the path has been obtained gradually. In working with a discrete configuration space, the inverse kinematic problem has been solved analytically and numerically using GA to find the robot configuration corresponding to a position in the workspace. This method consists of two algorithms;

1) GA for adjacent configurations generation for path planning.
2) GA for the obtaining of a complete path consisting of a set of adjacent configurations.

For path planning, (the second algorithm) in first please, the algorithm has been tested with one population of 30 individuals, and then it has been divided into three different parallel populations, with 10 individuals in each. Each of these populations will evolve using steady state GA. The advantage in this case that, as these populations evolve randomly and separately, there is more possibility to get better value than using one population. It’s clear if we increase the number of individuals per population and the number of populations, it may increase the probability of getting better results. However, the problem is by increasing the population size and the number of populations will increase the computational time. Thus, small population size has been used to minimize the computational time as possible.

The presented algorithms can be applied to any industrial robotic system. In this paper, application examples have been done using robot Puma 560 for testing the algorithms. An application program using object oriented C++ has been built to simulate the dynamics and kinematic Puma 560.

The results obtained have been analyzed and studied on base of the objective function $d$, and two other operational parameters. From the conducted analysis it’s possible to conclude the following:

- The introduced algorithm provides a solution for the path planning problem for industrial robots in complex environments. More over, the proposed procedure can be applied to any industrial manipulators just by selecting the appropriate significant and interesting points.
- The travelled distance by the significant points of the robot, are very acceptable, and by comparing the results with other works, in the most of cases, the travelled distance of the presented procedure is better than the ones of the other works.
- The comparison of results shows the efficiency of the proposed GA procedure over the four procedures (Seq, A*, UC, G) provided by Rubio[16]. Whereas, the GA procedure improved the results by an average of percentage 87.7%, 84.3%, 84.4%, and 68.6%, respectively.
• As shown in the illustrated examples, the computational time is relevantly high. Moreover, it increases as well as the restrictions increase. Furthermore, the number of individuals should be increased and so on the number of generations for more accurate results using the GA procedure, especially for big problems and big workspaces, which leads to an increasing of the computational time. This may consider as a disadvantage of the GA in general. However, as the industrial robots do a repetitive trajectories and paths, an offline planning normally takes place. This means that the computational time cost can be acceptable.

7. ACKNOWLEDGEMENTS

This paper has been made possible by the funding from the Spanish Ministry of MINISTERIO DE CIENCIA E INNOVACIÓN through the Project Research and Technological Development DPI2010-20814-C02-01.

8. REFERENCES


