2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

A Parallel Evolutionary Solution for the Inverse Kinematics of Generic Robotic Manipulators

Siavash Farzan* and G. N. DeSouza**
Vision-Guided and Intelligent Robotics Lab – ViGIR
University of Missouri
Email: *SFarzan@mail.missouri.edu, **DeSouzaG@missouri.edu

***Abstract*—This paper is an improvement of our previous work [1]. It provides a robust, fast and accurate solution for the inverse kinematics problem of generic serial manipulators – i.e. any number and any combination of revolute and prismatic joints. Here, we propose further enhancements by applying an evolutionary approach on the previous architecture and explore the effects of different parameters on the performance of the algorithm. The algorithm only requires the Denavit-Hartenberg (D-H) representation of the robot as input and no training or robot-dependent optimization function is needed. In order to handle singularities and to overcome the possibility of multiple paths in redundant robots, our approach relies on the computation of multiple (parallel) numerical estimations of the inverse Jacobian while it selects the current best path to the desired configuration of the end-effector using an evolutionary algorithm. But unlike other iterative methods, our method achieves sub-millimeter accuracy in 20 iterations in average. The algorithm was implemented in C/C++ using POSIX threads, and it can be easily expanded to use more threads and/or many-core GPUs. We demonstrate the high accuracy and real-time performance of our method by testing it with five different robots including a 7-DoF redundant robot. Results show that the evolutionary implementation of the algorithm is able to reduce the number of iterations compared to the previous method significantly, while also finding the solution within the specified margin of error.**

***Keywords*: Inverse Kinematics, Evolutionary Algorithms, Inverse Jacobian, Serial Manipulators, Parallel Computing, Denavit-Hartenberg.**

I. Introduction

In robotics, controlling the movement of a robot in space so that it performs a desired task is known as motion planning. In the case of robotic manipulators, the motion planning requires the solution of two problems: the forward and the inverse kinematics. The former is concerned with mapping the configuration of the joints variables $\vec{Q}(t)$ onto the position and orientation (i.e. pose) of the end-effector $\vec{X}(t)$, i.e. $\vec{X}(t) = f(\vec{Q}(t))$, while the latter, the inverse kinematics, determines the required configuration of every joint in order to achieve a given pose of the end-effector, or $\vec{Q}(t) = f^{-1}(\vec{X}(t))$. These mappings are necessary because when it comes to controlling the velocity of the end-effector, most methods rely on the calculation of $\dot{\vec{X}}(t) = J(\vec{Q}(t))\dot{\vec{Q}}(t)$ using the Jacobian, $J(\vec{Q}(t)) = \frac{\partial f}{\partial \vec{Q}}$, to estimate the joint velocities from the Cartesian velocities of the end-effector, i.e. $\dot{\vec{Q}}(t) = J(t)^{-1}\dot{\vec{X}}(t)$.

Many approaches have been researched in order to solve the inverse kinematics problem. In our previous work [1], we presented a fast and accurate method using an iterative numerical approximation of the inverse Jacobian. In this paper, we improve our previous method by employing an evolutionary approach, while further reducing the number of iterations, keeping its real-time applicability to generic robot manipulators – i.e. any number and any combination of revolute and prismatic joints. In summary, the main advantages and contributions of our method are: 1) unlike other iterative methods, ours is indeed accurate and fast; 2) it works for any generic robotic manipulator – redundant or not – even at singular configurations of the joint variables; 3) it is naturally implemented in parallel, running as multi threads in a simple CPU or on modern GPUs; 4) it does not require any training or robot-dependent optimization function, as it is the case of recent evolutionary methods (e.g. neural networks [2], genetic algorithms [3], and swarm optimization [4]); and 5) it is guaranteed to statistically converge to a solution.

The evolutionary approach used in the proposed algorithm is based on a single level, real-coded genetic algorithm, where the fitness function evaluates the positional and the orientational error of the robot end-effector. We also investigate the effects of different parameters such as number of Jacobian estimations and the amount of exploration on the accuracy and the convergence of the algorithm.

II. Background and Related Work

Over the years, several methods for solving the inverse kinematics problems have been proposed. These methods can be divided in basically two classes of methods: closed-form or numerical solutions [5]. Some of the earliest closed-form solutions were provided by Liao et al. [6], and Lee and Liang [7], who proposed a resultant elimination procedure using complex number method and vector theory respectively. However, the geometric interpretation of their elimination procedure was not completely revealed due to its complexity, and also their solution was limited to 7R [6] or 7-link (6R1P) [7] mechanisms. Later, Raghavan and Roth [8] showed that the inverse kinematics problem for a general 6R manipulator can present at most 16 different solutions, for any given pose of the end-effector. This allowed for the derivation of a characteristic polynomial of order 16 and the derivation of a generic closed-form solution in real time for the inverse kinematics of any 6R robot manipulator [9], [10]. While the method proposed in [8] had a great impact in the area, it was still limited to 6-DoF robots, more specifically: 6R, 5R1P, 4R2P and 3R3P robots. Also, their method required a Newton iteration for improved accuracy on the calculation of the eigen-vector/values of a matrix derived from the A_i matrices in the D-H representation. However, when multiple solutions exist due to redundancy in the joint configurations (i.e. $n_Q > n_X$), or at singular configurations of the robot, this matrix becomes ill-conditioned. In all these situations, numerical methods are usually required. These methods include cyclic coordinate descent methods [11], the Levenberg-Marquardt damped least squares methods [12], [13], quasi-Newton and conjugate gradient methods [14], [11], [15], neural network and artificial intelligence methods [16], [17], [18], [19], [20], [21], genetic algorithms [22], pseudo-inverse methods [23] and Jacobian transpose methods [24], [25].

Oyama et al. [16] presented a learning approach for IK problems based on modular neural network architectures using DeMers method [26]. This method involves an expert selector, an expert generator, and a feedback controller to accommodate the nonlinearities in the kinematic system. The disadvantages of their approach are the high complexity of the procedure for the IK computation, and the low learning speed. Furthermore, the final accuracy achieved for the hand position is about 10 mm, which is regarded as a large error. Bingul et al. [21] presented a neural network approach using the back-propagation algorithm for the IK solution of industrial robotic manipulators. The limitation of their approach is clearly the large errors in the joint angles.

A genetic algorithm approach to solve the IK problem was presented by Tabandeh et al. [22]. They used a minimizing genetic algorithm based on adaptive niching and clustering to find the joint angles with smallest positioning error of the end-effector. The fitness function was the end-effector error and a modified filtering and clustering step was added to the algorithm to identify and process the outputs of the genetic algorithm. The algorithm was tested using a 3-DoF robotic manipulator and the average error were determined to be approximately 5 mm to 20 mm for different experiments.

For the methods relying on Jacobian matrix, the Jacobian can be indirectly estimated using pseudo-inverse [27], optimization [28], and evolutionary algorithms [3], [2], [29], [30]. In the case of evolutionary methods, a simple genetic algorithm was employed in [30] for a four-joints redundant robot using a fixed number of iterations and limiting the search for a solution on the position of the end effector without considering its orientation. In order to apply the genetic algorithm on a redundant robot, the maximum joint displacement was used as an additional constraint and the fitness function was selected as a combination of the arm positioning error and the joint angle displacement from the initial position. As it will be shown in the next sections, this approach [30] led to poor results. So, Aguilar et al. [3] proposed a parallel implementation of the genetic algorithm to find a solution for both the position and orientation. The Denavit-Hartenberg representation was used to model a kinematic chain, and the chromosomes encoded the set of values for each joint angle (characteristics). Therefore, mutations were performed to create small random rotations within a range of the joint angles of the robot's manipulator. Finally the fitness function was defined as the distance between the end-effector current and desired positions. However, one major problem with the evolutionary approaches is the randomness during mutation and crossover, which are hard to "tune" and can increase the number of iterations needed to find the solution.

A major problem that arises during the motion of the robot is when it passes through singular configurations [31]. A number of authors (see [32]) avoid singular configurations by using the null-space method [33] and maximizing Yoshikawa's manipulability measure [34], [35]. Maciejewski and Klein [36] expanded this idea and proposed an approach to also avoid obstacles by defining task space vectors to critical points, with which the robot is directed away from the obstacle. Baillieul [32] proposed a more sophisticated null-space method, called the extended Jacobian method. In this method, a local minimum value of a secondary objective function is tracked. The purpose of this secondary function is to represent a set of constraints and also to take into account some objective functions (e.g. manipulability), which applies on the whole structure instead of physically-based constraints for each joint. The null-space method has also been used to assign different priorities to different tasks (see [37], [38]). Unfortunately, a thorough literature survey of this topic reveals that while many methods can indeed handle specific cases – e.g. 6R robots [39], [40], [10], [8], [9] – when it comes to redundant robots [28], [41], [27] and other robots at singular configurations [31], none of these methods achieved both accuracy and high performance at the same time. Besides, most numeric methods require either training, optimization of robot-dependent objective functions, or time to achieve reasonable accuracy [2]. Even methods that rely on massively parallel architectures to reduce time complexity require hundreds of iterations to achieve the specified error – e.g. the work in [3], where a NVidia GPU running 512 CUDA threads require over 42ms to converge.

The proposed method – which will be explained in the next section – does not require any previous training and it can produce the solution for the inverse kinematics problem for any robot, provided only its D-H table.

III. Proposed Algorithm

In this section, we introduce a new parallel evolutionary method for inverse kinematics. The method relies on estimates of the actual Jacobian matrix, eliminating the need to find an analytical solution for the inverse Jacobian, which for many robots is not even feasible. However, unlike our previous *greedy* method [1], this parallel evolutionary method addresses the problem of selecting one single estimate at each iteration of the algorithm. Instead, by evaluating an entire generation of individuals in each iteration, the evolutionary method proposed here can better handle non-monotonic Forward/Inverse Kinematics functions while trying to achieve the fewest possible iterations to get end-effector to the desired position.

A. Inverse and Pseudo-Inverse Jacobian

Let the pose of the end-effector $\vec{X}(t)$ be described by its three linear and three angular dimensions – i.e. (x, y, z) for its position in space, and (ϕ_r, ϕ_p, ϕ_y) the roll, pitch and yaw angles for its orientation. In addition, the joint configuration of the robot is described using the joint variable $\vec{Q}(t)$, with $q_i = d_i$ for the prismatic-joint lengths and $q_i = \theta_i$ for revolute-joint angles. The Jacobian matrix is then defined based on the forward kinematics equation $\vec{X}(t) = f(\vec{Q}(t))$; where $f(\vec{Q}(t))$ is readily obtained for q_i as a function of time in the D-H representation. Where,

$$\vec{X}(t) = \begin{bmatrix} x(t) = f_1(\vec{Q}(t)) \\ y(t) = f_2(\vec{Q}(t)) \\ z(t) = f_3(\vec{Q}(t)) \\ \phi_r(t) = f_4(\vec{Q}(t)) \\ \phi_p(t) = f_5(\vec{Q}(t)) \\ \phi_y(t) = f_6(\vec{Q}(t)) \end{bmatrix}, \quad \vec{Q}(t) = \begin{bmatrix} q_1(t) \\ q_2(t) \\ q_3(t) \\ q_4(t) \\ \vdots \\ q_n(t) \end{bmatrix} \tag{1}$$

and $f_i(\vec{Q}(t)) = f_i(q_1(t), q_2(t), ..., q_n(t))$
then

$$\dot{\vec{X}}(t) = J(\vec{Q}(t))\dot{\vec{Q}}(t) \tag{2}$$

where

$$\dot{\vec{X}}(t) = \frac{\partial \vec{X}(t)}{\partial t} = \begin{bmatrix} \frac{\partial x(t)}{dt} \\ \frac{\partial y(t)}{dt} \\ \frac{\partial z(t)}{dt} \\ \frac{\phi_r(t)}{dt} \\ \frac{\phi_p(t)}{dt} \\ \frac{\phi_y(t)}{dt} \end{bmatrix}, \quad \dot{\vec{Q}}(t) = \frac{\partial \vec{Q}(t)}{\partial t} = \begin{bmatrix} \frac{\partial q_1}{dt} \\ \frac{\partial q_2}{dt} \\ \vdots \\ \frac{\partial q_n}{dt} \end{bmatrix},$$

$$J(\vec{Q}(t)) = \begin{bmatrix} \frac{\partial f_1(\vec{Q}(t))}{\partial q_1} & \frac{\partial f_1(\vec{Q}(t))}{\partial q_2} & \cdots & \frac{\partial f_1(\vec{Q}(t))}{\partial q_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_6(\vec{Q}(t))}{\partial q_1} & \frac{\partial f_6(\vec{Q}(t))}{\partial q_2} & \cdots & \frac{\partial f_6(\vec{Q}(t))}{\partial q_n} \end{bmatrix} \tag{3}$$

Also, for simplicity of notation, hereafter we will replace the time dependency from all the terms in the equations above with the subscript "t". Similarly, we will omit the dependency on $\vec{Q}_t$ in the Jacobian $J(.)$, but it should be made clear here that a Jacobian can only be fully defined at the current configuration $\vec{Q}_t$ of the robot. Finally, it is assumed that the initial position ($\vec{X}_{t_0}$) and the initial joints configuration ($\vec{Q}_{t_0}$) are known – e.g. can be obtained by reading the current values of robot encoders.

As the equation (3) implies, the Jacobian matrix J can be numerically estimated by causing small changes $\partial \vec{X}$ while applying arbitrarily small and individual perturbations to ∂q_j's at the current pose $\vec{Q}_t$. For example,

$$J_c = \frac{\partial \vec{X}}{\partial q_c} = \vec{X}_t - f(\vec{Q}_t + [\ldots, 0, 0.01, 0, \ldots]) \tag{4}$$

where the subscript c indicates the column of the Jacobian and t is the iteration step. As it will be described in the next section, more Jacobians will be estimated in the algorithm using the mutation procedure.

Also, in order to move the end-effector toward its final position, the next joint configuration can be calculated using the inverse of the Jacobian matrix:

$$\triangle \vec{Q}_t = J_t^{-1} * \alpha_t(\vec{X}_{final} - \vec{X}_t) \tag{5}$$

where $\alpha_t \in (0, 1)$ is an attenuation factor which can affect the path of the end-effector, and also the time for the process to converge. For example, a fixed $\alpha_t \simeq 1$ can cause the end-effector to jump back and forth over the desired $\vec{X}_{final}$. On the other hand, a small α_t may slow the convergence process and it can also cause $\vec{X}_t$ to only asymptotically reach the $\vec{X}_{final}$. Using $\alpha_t = 1$ at the beginning and slowly decreasing it towards the end of the process guarantees fast convergence at the same time that it avoids over stepping $\vec{X}_{final}$.

When the number of joints, n_Q, is either smaller or greater than the number of degrees of freedom of the workspace, n_X – i.e. $n_Q < n_X$ or $n_Q > n_X$, – the equation (5) can be re-written, respectively, using the left $((J_t^T J_t)^{-1} J_t^T)$ or the right $(J_t^T (J_t J_t^T)^{-1})$ pseudo-inverses of the Jacobian. Otherwise, the equation is solved using the normal inverse, J_t^{-1}, as already indicated.

The current position, $\hat{\vec{X}}$, of the end-effector for the new joint configuration is computed using the forward kinematics applied to the addition of $\triangle \vec{Q}_t$ to the current joint configuration $\vec{Q}_t$. This and other aspects of the algorithm used in the proposed evolutionary algorithm will be explained in the following section.

B. Evolutionary Methodology

Since there is no guarantee that any single estimate of the Jacobian can lead to the final solution, the parallel evolutionary method proposed here originated from the assumption that by creating multiple estimates of the Jacobian matrix, the process can be sped up and an optimal path to this same final pose can be found with fewer iterations. With that in mind, we devised an evolutionary algorithm to produce a generation of Jacobian matrices which will evolve over time through the selection of the current best individuals in the path. In order to handle the several Jacobian matrices at each iteration, we resort to parallel computation through the use of multiple threads.

For a robotic manipulator, the individual in a population would be represented by the real value of the joint variables as $\{q_1, q_2, ..., q_n\}$. The evolutionary approach used in this work thus falls in the category of a real-coded genetic algorithm, which includes initialization, iterative selection made on the basis of fitness, mutation and termination.

The algorithm can be summarized in the following major steps: At each iteration t, first, the best u individuals (poses) are selected based on a fitness function explained in Section III-B2 and the Jacobian matrices at those positions are estimate: i.e. $\frac{m}{u}$ estimations are computed by the mutation procedure for each selected individual. Next, the inverses of the calculated Jacobians are derived using the pose of the robot given by each individual. These inverses are used to find the

Algorithm 1 : Proposed Parallel Evolutionary Algorithm

```
procedure IK($joints\ configuration:\ \vec{Q}$)
    $n \leftarrow number\ of\ joints$
    $\vec{Q}_{t_0} \leftarrow joints\ configuration$
    $\vec{X}_{t_0} = f(\vec{Q}_{t_0})$
    $\vec{X}_t = \vec{X}_{t_0}$
    $\vec{Q}_t = \vec{Q}_{t_0}$
    while ($\|\vec{X}_t - \vec{X}_{final}\| > \varepsilon_r$) do
        create $m$ threads
        thread-do
            $l \leftarrow threadID\ divided\ by\ (\frac{m}{u})$
            $k_l \leftarrow threadID\ mod\ (\frac{m}{u})$
            $k \leftarrow threadID$
            for $each\ joint\ c \in [1 \dots n_Q]$ do
                $J^l_c = \frac{\partial X}{\partial q_i} = \vec{X}^l{}_t - f(\vec{Q}_t + \partial q_c)$
            end-for
            $J^l_t = \vec{X}^l_t - f(\vec{Q}^l_t + \partial q_t)$
            $J^{k_l}_t = J^l_t + \aleph(0, \Sigma_J)$
            if $n = 6$
                $\triangle\vec{Q}^{k_l}_t = (J^{k_l}_t)^{-1} * \alpha_t(\vec{X}_{final} - \vec{X}^l_t)$
            else
                $\triangle\vec{Q}^{k_l}_t = (J^{k_l}_t)^{-P} * \alpha_t(\vec{X}_{final} - \vec{X}^l_t)$
            end-if
            $\vec{Q}^{k_l}_t = \vec{Q}^l_t + \triangle\vec{Q}^{k_l}_t$
            $\vec{X}^{k_l}_t = f(\vec{Q}^{k_l}_t)$
        end-thread
        for each $l \in [1 \dots u)$ and $k_l \in [1 \dots \frac{m}{u})$ do
                $\vec{X}^l_{t+1} = \{\vec{X}^{k_l}_t \mid \|\vec{X}_{final} - \vec{X}^{k_l}_t\|\ is\ minimal\}$
        end-for
        $\vec{X}_{t+1} = min\{\vec{X}^l_{t+1}\}$
    end-while
end-procedure
```

next m joint configurations of the end effector. Finally, using these joint configurations, m current positions, $\hat{\vec{X}}^k_t$, of the end-effector are computed using the forward kinematics applied to the addition of $\triangle\vec{Q}^k_t$ to the current joint configuration $\vec{Q}^k_t$ of the individual. The process iterates until the termination condition is satisfied.

In the rest of this section, we explain further these steps and the different terms used in the proposed evolutionary algorithm, including *mutation*, *selection* and *termination*.

1) Mutation Procedure: In order to create multiple estimates of the Jacobian matrix at the current position $J_t(\vec{Q})$ – i.e. the next generation of solutions from selected individuals – a mutation method is employed. After selecting the best u individuals based on the selection process – which will be explained in the next section – $\frac{m}{u}$ initial Jacobians $J^{k_l}_t(\vec{Q})$ are created for each of the u selected individuals, and $\frac{m}{u}$ matrices from a white-noise distribution $\aleph(0, \Sigma_J)$ are added to each Jacobian as a mutation operator. That is, the total number of m estimations of the Jacobians, J^l_t, are created by:

$$J^{k_l}_t = J^l_t + \aleph_k(0, \Sigma_J)\ \ for\ \ k_l = [0, \frac{m}{u})\ \ and\ \ l = [0, u) \quad (6)$$

It is important to mention that for the first iteration (i.e. in the initialization step), m different estimations are also created, but at that time, for the single, initial position of the robot.

Next, by using the inverse (or pseudo-inverse) of the m estimates of the Jacobian matrices, m joint configurations can be calculated in parallel so that all $J^{k_l}_t$'s are pointing toward $\vec{X}_{final}$. That is, all Jacobians should cause the end-effector to move towards the final pose, but each one with a different "slope".

$$\triangle\vec{Q}^{k_l}_t = (J^{k_l}_t)^{-1} * \alpha_t(\vec{X}_{final} - \vec{X}^l_t) \quad (7)$$

2) Selection: During each successive generation of m joint configurations, a proportion of the existing population is selected to breed a new generation. Individual joint configuration are selected through a fitness-based process

$$fitness = \alpha * err_s + \beta * err_o \quad (8)$$

where $err_s = ((x_f - x_i)^2 + (y_f - y_i)^2 + (z_f - z_i)^2)^{\frac{1}{2}}$ and $err_o = ((\phi_{a_f} - \phi_{a_i})^2 + (\phi_{b_f} - \phi_{b_i})^2 + (\phi_{c_f} - \phi_{c_i})^2)^{\frac{1}{2}}$.

α and β are factors in order to adjust the importance of the space (Cartesian) error and the orientation error in different applications. In the current implementation, $\alpha = 1$ and $\beta = 0.5$.

This selection method rates the fitness of each solution and preferentially select the best solutions as the next parent individuals.

As eq (8) indicates, the fitness of each solution is evaluated based on its distance to the desired end-effector position. However, instead of choosing a single individual for the position of the end-effector, $\hat{\vec{X}}$ – as in [1] – u individuals are now chosen. The entire process above is performed in parallel, using:

$$\vec{X}^{k_l}_t = f(\vec{Q}^l_t + \triangle\vec{Q}^{k_l}_t)\ \ for\ \ k_l = [0, \frac{m}{u})\ \ and\ \ l = [0, u) \quad (9)$$

It is important to mention that other constraints can be imposed to the fitness function in order to avoid obstacles, select linear paths, scape local minima, etc.

3) Termination: After selecting the u best fitted individuals, m new estimations of the Jacobian are created using m threads, where each set of $\frac{m}{u}$ threads processes the u positions reached by the parents. The process iterates until the following termination condition has been reached for any of the $k_l = 1 \dots \frac{m}{u}\ and\ l = 1 \dots u$ elements in each generation – i.e. the $k = 1 \dots m$ solutions for the inverse kinematics computed.

$$\vec{X}_t = \{\vec{X}^k_t \mid \|\vec{X}_{final} - \vec{X}^k_t\|\ is\ minimal\} \quad (10)$$

The pseudo-code of the proposed process is presented in more detail in Algorithm 1.

IV. Experimental Results and Discussion

In this section, we present three experiments that were performed. First, we ran our algorithm using four different robots and compared the proposed parallel evolutionary method with the method presented in the previous work [1]. For the second experiment, we performed a comparison between the proposed method and two Genetic Algorithm approaches presented in [3] and [30], which were also discussed in Section II. Finally, in the last experiment, we studied the effects of different parameters of the proposed algorithm on the convergence to a solution. These parameters are including total number of threads as well as number of selected individuals at each iteration, the desired accuracy and the amount of exploration.

Our tests were performed on an Intel(R) E8400 CPU running at 3.00GHz. In average, the algorithm performed extremely well, returning a solution with sub-millimeter accuracy in about 20 iterations. The typical speed-up obtained by the parallel evolutionary approach was about two times the speed of the previous parallel method, while keeping approximately the same error.

A. Test for General Robot Manipulators

In this section, the results of applying the proposed inverse kinematics solution to four different robotic manipulators are presented. Over 100 random final positions of the end-effector were used for test of the non-redundant Puma 560, Kuka and Scara and the redundant 7-Dof Mitsubishi PA10-7C robotic manipulators. The D-H parameters of these robots can be found in Table I(a)-(d). All angles are in degree and all lengths in millimeters. It is important to point out that the initial position of the end-effector can affect the results as well as the path to the final position. Due to limitation of space, here we only report some of the most typical results and the total average. A spreadsheet with the remaining results can be found at http://vigir.missouri.edu/parallel-programming. Also, for the experiments reported here, the initial positions of the robots were set to the manufacturer-defined *home* position.

The first two robots have six revolute joints, while the Mitsubishi robot has seven revolute joints, and Scara robot has three revolute and one prismatic joints. The algorithm was implemented in C/C++ using POSIX threads. For the current tests, the number m of threads used for each iteration was 24 and the number u of threads picked at each iteration was 4. (In Section IV-C, we discuss this choice in detail).

Table I
D-H PARAMETERS OF THE TESTED ROBOTS

#	θ	d	a	α
1	θ_1	0	0	-90
2	θ_2	149.09	431.8	0
3	θ_3	0	-20.3	90
4	θ_4	433.07	0	-90
5	θ_5	0	0	90
6	θ_6	56.25	0	0

(a) Puma 560

#	θ	d	a	α
1	θ_1	700	750	-90
2	θ_2	0	1250	0
3	θ_3	0	-55	-90
4	θ_4	1500	0	90
5	θ_5	0	0	90
6	θ_6	-230	0	180

(b) Kuka robot

#	θ	d	a	α
1	θ_1	317	0	-90
2	θ_2	0	0	90
3	θ_3	450	0	-90
4	θ_4	0	0	90
5	θ_5	480	0	-90
6	θ_6	0	0	90
7	θ_7	70	0	0

(c) Mitsubishi PA10-7C

#	θ	d	a	α
1	θ_1	0	250	0
2	θ_2	0	350	180
3	0	d_3	0	0
4	θ_4	114.5	0	0

(d) Scara robot

As it can be seen in Tables II(a)-(d), in most of the tests reported here, the algorithm was able to find the inverse kinematics solution in less than 20 iterations within the specified margin of error. As the average values indicate, the number of iterations shown in the tables are typical also for the tests not reported here. The error column is the same for both the proposed method and the parallel method since the termination conditions for both cases were set to $\varepsilon_{r_{Trans}} < 1mm$ and $\varepsilon_{r_{Rot}} < 0.5^o$. For all cases above, in the third column of the tables, we report the number of iterations required by the proposed parallel evolutionary method presented in section III versus the parallel method in [1].

B. Comparison with other works

For the second set of experiments, we present a comparison between the results of the proposed method and two genetic algorithm approaches presented in [3] and [30], discussed in previous sections.

In [3], Aguilar and Huegel applied their serial and parallel genetic algorithm method to a Puma 500 robot. They reported the results for two different positions. The comparison between those results and the results of the new proposed method for the same robot and positions is presented in Table III(a). In this test, each algorithm was allowed to iterate freely while the termination condition was fixed ($< 1mm$ and $< 0.5^o$).

Parker et al. in [30] applied their genetic algorithm method to a redundant robot for a fixed number of iterations, i.e. 50 iterations. The new proposed algorithm was also applied to the same redundant robot for the same 50 iterations. The results are shown in Table III(b).

As it can be seen in the results, although in both cases the genetic algorithm solutions, specially the parallel one (Table III(a)), have reasonable outputs, the new proposed method showed a much better performance with less than $\frac{1}{10}$ of the number of iterations.

C. Effects of Different Parameters On Convergence

In this section, the effects of different parameters of the proposed algorithm on the convergence to a solution are studied. These parameters are the total number of estimated Jacobians, the number of selected individuals, the accuracy (i.e. termination condition), and the exploration parameter, i.e. the amount of white noise added to the Jacobians. As we mentioned before, for all the results shown in the previous sections, the termination condition was set to 1 millimeter for the position and 0.5 degree for the orientation of the end-effector. Furthermore, the number of estimated Jacobians were 24 with selection of 4 best individuals, and the standard deviation of the distribution used for exploration was set to 0.5. Here, we must justify these choices.

The first question that arises from the previous tests is regarding the best number of threads (i.e. number of Jacobians) and also the best number of selected individuals to be used. These choices are constrained by the hardware employed (i.e. number of CPU cores, memory, etc.). In order to minimize the influence of the hardware, we measure this parameter versus the number of iterations. Figure 1 combines these two parameters in one graph: the effect of using different number of Jacobians on the convergence time, and also keeping

Table II
RESULTS FOR TESTED ROBOTS

End-effector position and orientation (x, y, z, ϕ_r, ϕ_p, ϕ_y)	Calculated joint configuration (θ_1, θ_2, θ_3, θ_4, θ_5, θ_6)	# of iterations (Proposed method / Parallel method)	Error Position / Orientation (mm) / (deg)
(164, 415, 165, 90, 30, 60)	(-85.4, -97.5, -23.19, 122.12, 92.59, 3.47)	11 / 46	0.51 / 0.41
(543, -587, 43, 45, -15, 60)	(143.52, -143.21, 30.13, -178.31, 5.07, -74.51)	9 / 22	0.72 / 0.42
(-280, 360, 700, 30, 15, -60)	(108.33, -33.85, 39.54, 3.92, 55.36, -73.34)	13 / 25	1.02 / 0.22
(140, 600, 560, -60, 0, -15)	(-87.95, -122.70, 70.64, 165.35, 120.04, 143.79)	10 / 25	1.01 / 0.26
Average of 20x20 tests		17.48 / 28.11	0.76 / 0.38

(a) Four arbitrarily chosen test cases and the average of all 400 trials for the Puma 560

End-effector position and orientation (x, y, z, ϕ_r, ϕ_p, ϕ_y)	Calculated joint configuration (θ_1, θ_2, θ_3, θ_4, θ_5, θ_6)	# of iterations (Proposed method / Parallel method)	Error Position / Orientation (mm) / (deg)
(-560, 730, 1300, 45, -30, 30)	(123.12, -163.63, 68.77, 159.31, 120.49, 74.31)	9 / 33	0.84 / 0.54
(560, 720, 1400, 30, 15, -60)	(40.04, 174.40, 65.23, -71.62, -61.03, 142.04)	14 / 28	0.02 / 0.00
(280, 1200, 1120, -60, 0, -15)	(78.96, -136.20, 70.48, 11.59, -104.05, -44.84)	11 / 51	0.52 / 0.02
(-920, -430, 822, 75, -15, 45)	(-161.98, 131.60, 99.47, 147.37, 87.36, 104.31)	16 / 31	0.54 / 0.01
Average of 20x20 tests		15.49 / 33.84	0.68 / 0.21

(b) Four arbitrarily chosen test cases and the average of all 400 trials for the Kuka robot

End-effector position and orientation (x, y, z, ϕ_r, ϕ_p, ϕ_y)	Calculated joint configuration (θ_1, θ_2, θ_3, θ_4, θ_5, θ_6,θ_7)	# of iterations (Proposed method / Parallel method)	Error Position / Orientation (mm) / (deg)
(450, 80, 120, 90, 15, -60)	(158.49, -177.72, -35.12, 124.77, 158.94, 69.23, 92.62)	14 / 27	0.02 / 0.54
(-180, -260, 200, -90, -30, 90)	(108.97, -74.95, -64.13, -131.99, 57.32, -144.37, -152.85)	11 / 24	0.94 / 0.05
(-140, 275, 140, -90, 0, -15)	(68.16, 74.34, 36.42, 130.90, 69.87, 125.83, 71.63)	23 / 32	0.75 / 0.11
(-170, 100, 60, -30, 30, 15)	(83.80, -116.41, -37.95, -148.13, -21.30, 59.07, -46.61)	12 / 40	0.75 / 0.07
Average of 20x20 tests		26.57 / 42.28	0.82 / 0.28

(c) Four arbitrarily chosen test cases and the average of all 400 trials for the Mitsubishi PA10-7C

End-effector position and orientation (x, y, z, ϕ_r, ϕ_p, ϕ_y)	Calculated joint configuration (θ_1, θ_2, d_3, θ_4)	# of iterations (Proposed method / Parallel method)	Error Position / Orientation (mm) / (deg)
(350, 250, 120, 75, 0, 0)	(-18.89, 89.98, -34.50, -3.07)	10 / 18	0.18 / 0.28
(-175, -250, -200, 90, 0, 0)	(157.57, 121.64, 85.50, -170.03)	17 / 29	0.14 / 0.26
(300, -475, -140, 30, 0, 0)	(-33.48, -41.41, 25.50, 75.38)	9 / 16	0.87 / 0.09
(175, 450, 400, -30, 0, 0)	(24.65, 74.05, -314.50, 128.63)	11 / 21	0.72 / 0.02
Average of 20x20 tests		11.79 / 22.75	0.56 / 0.18

(d) Four arbitrarily chosen test cases and the average of all 400 trials for the Scara robot

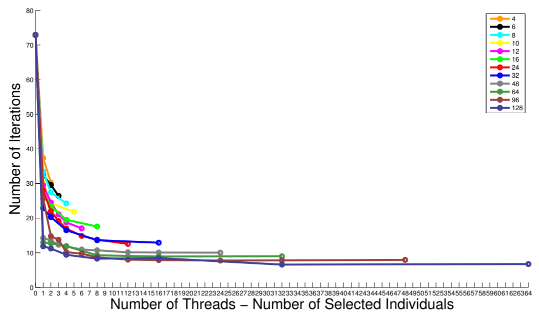

Figure 1. Convergence vs. Threads Configuration for the Puma 560

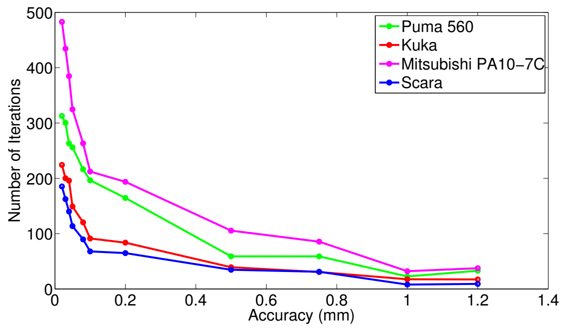

Figure 2. Convergence vs. Accuracy

different number of individuals as the parents for next iteration. It is important to mention that keeping only 1 Jacobian makes

Table III
COMPARISON WITH OTHER WORKS

End effector pose	Calculated joint configuration	Number of Iterations		
(x, y, z, ϕ_r, ϕ_p, ϕ_y)	(θ_1, θ_2, θ_3, θ_4, θ_5, θ_6)	Serial GA	Parallel GA	Proposed Method
(29, -163, 343, -5.6, 1.8, 0)	(-19, 116, -64, 7, -53, 9)	395	260	25
(-284, -803, 186, -0.5, 1.5, 0)	(-99, 124, 162, 178, -75, -80)	211	239	18

(a) Comparison between the proposed method and the method presented in [3]

End effector position and orientation	Calculated joint configuration	Genetic Algorithm		Proposed Method		
(x, y, z, ϕ_r, ϕ_p, ϕ_y)	(θ_1, θ_2, θ_3, θ_4)	Position Error (mm)	Number of Iterations	Position Error (mm)	Orientation Error (deg)	Number of Iterations
(685.8, 152.4, 660.4, 0, 30, -90)	(0, -39, 86, -20)	7	50	0.87	0.46	21
(-25.4, 355.6, 635, -60, 30, 90)	(-61, 158, 91, 142)	4.6	50	0.72	0.54	16

(b) Comparison between the proposed method and the method presented in [30]

algorithms 1 and the one proposed in [1] identical. As it can be seen in the figure, generating more threads, helps, and keeping more threads also improves the performance of the algorithm. In the case of total number of threads, using only two Jacobian estimations rather than one has a tremendous effect on the number of iterations. However, there is not much difference after about 48 Jacobians being created. Apparently, this shows that the probability of finding the closest $\vec{X}_{final}$ from 48 estimations is large enough to eliminate the need for more estimations. In the case of number of selected individuals, as Figure 1 indicates, keeping more Jacobians/individuals has also a positive effect on the convergence. For example, having total number of 24 threads/Jacobians, selecting 2, 3, 4, 6, 8, and 12 individuals can decrease the number of iterations from 28 to 22, 19, 17, 15 and 13, respectively.

However, due to limitation on the number of cores in a CPU-based system, the total number of 24 Jacobians with 4 selected individuals at each iteration was used in this work.

Since different applications may require different accuracies in robot pose, an immediate question about the behavior of the proposed method vis-a-vis the required accuracy must be investigated. Figure 2 shows the effect of changing the accuracy – i.e. the termination condition – on the number of iterations required for the convergence of the algorithm. A total of 20 different final positions of the end-effector, running 20 times each, for each of the four robots were averaged to produce the results in this figure. As it was expected, the convergence takes longer as we increase the accuracy. The algorithm can achieve the accuracy of 20 micrometer in position and 0.02 degree in orientation of the end-effector in 312 iterations for Puma 560 robot. Based on this figure, one can now choose the most appropriate accuracy/number of iterations for each application.

Finally, the effect of different amount of exploration in the creation of Jacobians is studied. Since the proposed approach is a greedy algorithm and the Forward/Inverse Kinematics functions are non-monotonic, there can never be a guarantee that one specific amount of exploration will always work the best. As Figure 3 indicates, while the number of iterations seems to reach reasonable values for a standard deviation of 0.5, the rising number of iterations at the trailing edge of 3 out of 4 of the curves in Figure 3 (after $std \geq 1.2$) as well as the spike in number of iterations for the third curve between $0.8 \leq std < 1.2$ suggest that further investigation is warranted. For example, the effects of the covariance between the n-DoF of the robot could be taken into the account, which were assumed zero in this work.

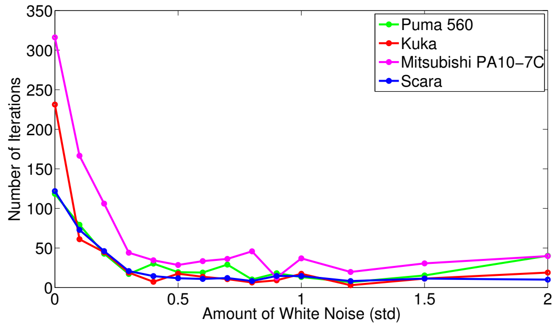

Figure 3. Convergence vs. White Noise

V. CONCLUSION AND FUTURE WORK

This paper presented an improvement of the previous work presented in [1] as a solution of the inverse kinematics problem for a general robotic manipulator. The method is an evolutionary approach implemented in parallel using C/C++ programming and POSIX threads. Our experimental results showed that the proposed method achieves 1mm accuracy in an average of 20 iterations. The high accuracy and the real-time performance of our method was demonstrated by testing it with five different robots, including a 7-DoF redundant robot. Furthermore, in addition of comparison with the previous work in [1], the algorithm is compared to two other evolutionary algorithms presented in [30] and [3].

While some choices of the parameters of the algorithm including the threads configuration and the desired accuracy were experimentally justified, others remain configurable depending on the desired constraints of the specific application. Yet one parameter (amount of exploration) requires further investigation. Moreover, the use of constraints, as to avoid obstacles, when selecting the best $\vec{X}$ towards the final position of the end-effector can be further investigated. Finally, expanding the algorithm to use many-core GPUs could lead to an impressive speed-up in the algorithm..

References

[1] S. Farzan and G. N. DeSouza, "From d-h to inverse kinematics: A fast numerical solution for general robotic manipulators using parallel processing," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, 2013, pp. 2507–2513.

[2] R. KöKer, "A genetic algorithm approach to a neural-network-based inverse kinematics solution of robotic manipulators based on error minimization," *Inf. Sci.*, vol. 222, pp. 528–543, Feb. 2013.

[3] O. A. Aguilar and J. C. Huegel, "Inverse kinematics solution for robotic manipulators using a cuda-based parallel genetic algorithm," in *Proceedings of the 10th Mexican international conference on Advances in Artificial Intelligence - Volume Part I*, ser. MICAI'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 490–503.

[4] D. Pham, M. Castellani, and A. Fahmy, "Learning the inverse kinematics of a robot manipulator using the bees algorithm," in *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*, july 2008, pp. 493–498.

[5] D. Pieper, "The kinematics of manipulators under computer control," *Ph. D. Thesis, Stanford University*, 1968.

[6] Q. Z. Liao and Q. Z. C. G. Liang, "A novel approach to the displacement analysis of general spatial 7r mechanism," *Chinese Journal of Mechanical Engineering*, vol. 22, no. 3, pp. 1–9, 1986.

[7] H.-Y. Lee and C.-G. Liang, "Displacement analysis of the spatial 7-link 6r-p linkages," *Mechanism and Machine Theory*, vol. 22, no. 1, pp. 1 – 11, 1987.

[8] M. Raghavan and B. Roth, "Inverse kinematics of the general 6R manipulator and related linkages," *Journal of Mechanical Design*, vol. 115, no. 3, pp. 502–508, 1993.

[9] D. Manocha. and J. F. Canny, "Real time inverse kinematics for general 6r manipulators," in *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*. IEEE, 1992, pp. 383–389.

[10] D. Manocha and J. F. Canny, "Efficient inverse kinematics for general 6r manipulators," *Robotics and Automation, IEEE Transactions on*, vol. 10, no. 5, pp. 648–657, 1994.

[11] L.-C. Wang and C. Chen, "A combined optimization method for solving the inverse kinematics problems of mechanical manipulators," *Robotics and Automation, IEEE Transactions on*, vol. 7, no. 4, pp. 489–499, 1991.

[12] C. Wampler, "Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 16, no. 1, pp. 93–101, 1986.

[13] Y. Nakamura, "Inverse kinematic solutions with singularity robustness for robot manipulator control," *J. of Dynamic Systems, Mes. and Contr.*, vol. 108, pp. 163–171, 1986.

[14] J. Zhao and N. I. Badler, "Inverse kinematics positioning using nonlinear programming for highly articulated figures," *ACM Transactions on Graphics (TOG)*, vol. 13, no. 4, pp. 313–336, 1994.

[15] A. Deo and I. Walker, "Adaptive non-linear least squares for inverse kinematics," in *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*, 1993, pp. 186–193 vol.1.

[16] E. Oyama, N. Y. Chong, A. Agah, and T. Maeda, "Inverse kinematics learning by modular architecture neural networks with performance prediction networks," in *Robotics and Automation, Proceedings 2001 ICRA. IEEE International Conference on*, vol. 1, 2001, pp. 1006–1012.

[17] G. G. Lendaris, K. Mathia, and R. Saeks, "Linear hopfield networks and constrained optimization," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 29, no. 1, pp. 114–118, 1999.

[18] A. Ramdane-Cherif, B. Daachi, A. Benallegue, and N. Levy, "Kinematic inversion," in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 2, 2002, pp. 1904–1909 vol.2.

[19] G. Tevatia and S. Schaal, "Inverse kinematics for humanoid robots," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 1, 2000, pp. 294–299 vol.1.

[20] A. D'Souza, S. Vijayakumar, and S. Schaal, "Learning inverse kinematics," in *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, vol. 1, 2001, pp. 298–303 vol.1.

[21] Z. Bingul, H. Ertunc, and C. Oysu, "Applying neural network to inverse kinematic problem for 6r robot manipulator with offset wrist." Springer Vienna, 2005, pp. 112–115.

[22] S. Tabandeh, C. Clark, and W. Melek, "A genetic algorithm approach to solve for multiple solutions of inverse kinematics using adaptive niching and clustering," in *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, 2006, pp. 1815–1822.

[23] D. Whitney, "Resolved motion rate control of manipulators and human prostheses," *Man-Machine Systems, IEEE Transactions on*, vol. 10, no. 2, pp. 47–53, 1969.

[24] A. Balestrino, G. De Maria, and L. Sciavicco, "Robust control of robotic manipulators," in *Proceedings of the 9th IFAC World Congress*, vol. 5, 1984, pp. 2435–2440.

[25] W. Wolovich and H. Elliott, "A computational technique for inverse kinematics," in *Decision and Control, 1984. The 23rd IEEE Conference on*, vol. 23, 1984, pp. 1359–1363.

[26] D. DeMers and K. Kreutz-Delgado, "Solving inverse kinematics for redundant manipulators," *Neural Systems for Robotics*, pp. 75–112, 1997.

[27] D. Demers and K. Kreutz-Delgado, *Inverse Kinematics of Dextrous Manipulators*. 525 B Street, Suite 1900, San Diego, CA 92101-4495, USA: Academic Press, 1997, ch. 4.2, pp. 77–80.

[28] S. Kumar, N. Sukavanam, and R. Balasubramanian, "An optimization approach to solve the inverse kinematics of redundant manipulator," *International Journal of Information and Systems Sciences*, vol. 6, no. 4, pp. 414–423, 2010.

[29] P. Kalra, P. Mahapatra, and D. Aggarwal, "An evolutionary approach for solving the multimodal inverse kinematics problem of industrial robots," *Mechanism and Machine Theory*, vol. 41, no. 10, pp. 1213 – 1229, 2006.

[30] J. Parker, A. Khoogar, and D. Goldberg, "Inverse kinematics of redundant robots using genetic algorithms," in *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, may 1989, pp. 271 –276 vol.1.

[31] A. T. Hasan, N. Ismail, A. M. S. Hamouda, I. Aris, M. H. Marhaban, and H. M. A. A. Al-Assadi, "Artificial neural network-based kinematics jacobian solution for serial manipulator passing through singular configurations," *Advances Engineering Software.*, vol. 41, no. 2, pp. 359–367, Feb. 2010.

[32] J. Baillieul, "Kinematic programming alternatives for redundant manipulators," in *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, vol. 2, 1985, pp. 722–728.

[33] A. Liegeois, "Automatic supervisory control of the configuration and behavior of multibody mechanisms," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 7, no. 12, pp. 868–871, 1977.

[34] T. Yoshikawa, "Dynamic manipulability of robot manipulators," in *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, vol. 2, 1985, pp. 1033–1038.

[35] ——, "Manipulability of robotic mechanisms," *The international journal of Robotics Research*, vol. 4, no. 2, pp. 3–9, 1985.

[36] A. A. Maciejewski and C. A. Klein, "Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments," *The international journal of robotics research*, vol. 4, no. 3, pp. 109–117, 1985.

[37] S. Chiaverini, "Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators," *Robotics and Automation, IEEE Transactions on*, vol. 13, no. 3, pp. 398–410, 1997.

[38] P. Baerlocher and R. Boulic, "Task-priority formulations for the kinematic control of highly redundant articulated structures," in *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*, vol. 1, 1998, pp. 323–329 vol.1.

[39] S. Qiao, Q. Liao, S. Wei, and H.-J. Su, "Inverse kinematic analysis of the general 6r serial manipulators based on double quaternions," *Mechanism and Machine Theory*, vol. 45, no. 2, pp. 193 – 199, 2010.

[40] M. Husty, M. Pfurner, and H.-P. Schrocker, "A new and efficient algorithm for the inverse kinematics of a general serial 6r manipulator," *Mechanism and Machine Theory*, vol. 42, no. 1, pp. 66–81, 2007.

[41] O. Ivlev and A. Graser, "Resolving redundancy of series kinematic chains through imaginary links," in *in Proc. CESA 98 IMACS Multiconference. Computational Engineering in Systems Applications*, 1998, pp. 477–482.