Exploiting Multi-Level Parallelism for Run-Time Adaptive Inverse Kinematics on Heterogeneous MPSoCs

Leonardo Suriánó, (Member, IEEE), Andrés Otero, (Member, IEEE), Alfonso Rodríguez, (Graduate Student Member, IEEE), Manuel Sánchez-Renedo, and Eduardo de la Torre

1Centro de Electrónica Industrial, Universidad Politécnica de Madrid, 28006 Madrid, Spain
2Thales Alenia Space España, 28760 Tres Cantos, Spain

 Corresponding author: Leonardo Suriánó (leonardo.surián@upm.es)

This work was supported by the European Union’s Horizon 2020 research and innovation programme under Grant 732105. The work of Leonardo Suriánó was supported by the Universidad Politécnica de Madrid under Grant RR01/2015 (Programa Propio).

Abstract This paper presents a run-time solver for the inverse kinematics of a robotic arm implemented on a heterogeneous Multi-Processor System-on-Chip (MPSoC). The solver has been formulated as an optimization problem, in which two levels of algorithmic parallelism are proposed: i) the Nelder-Mead derivative-free method used as the optimization engine is modified to allow the evaluation of the cost function in multiple vertices simultaneously, ii) the trajectory is divided into non-overlapping segments, in which all the points are solved concurrently. Algorithmic parallelism is supported by a variable number of parallel instances of a custom hardware accelerator, which speeds up the computation of the forward kinematics equations of the robot required during the resolution of the inverse kinematics. This adaptable scheme provides run-time scalability in terms of trajectory accuracy, logic resources, dependability, and execution time. New design methodologies are used to unify the modeling of the software and hardware partitions of the controller while transparently providing adaptability. They are based on the dataflow Model of Computation (MoC), supported by the PREESM prototyping tool. This tool has been extended to support the use of dynamically reconfigurable hardware accelerators implemented using the ARTico3 framework. The proposal has been validated with a python-based robotic arm simulator. Experimental results show how the proposed parallelism, combined with hardware acceleration, enables the run-time resolution of the trajectory with adaptable performance using a Xilinx Zynq UltraScale+ MPSoC device.

Index Terms Inverse kinematics, robot arm, nelder-mead, MPSoC, parallelization, hardware acceleration.

I. INTRODUCTION

Inverse Kinematics (IK) is a well-known problem in robotics, which aims at determining the joint angles that make the end-effector of the robot reach a given position. If the robot operates in a space without obstacles and the application does not require the arm to follow a specific trajectory, it would be enough to solve a single instance of the IK to obtain the set of angles that brings the end-effector to the target point in space. However, in various applications, such as surgery [1], [2] or industrial scenarios [3], [4], it is necessary to meticulously control the trajectory as explained in [5], [6].

In those cases, the expected path is discretized, and multiple IK calculations are carried out to guarantee the end-effector goes through every intermediate point on its way from the origin to the destination. Moreover, robots may operate in dynamic environments, in which the trajectory cannot be planned offline. Hence, all the intermediate IK calculations must be carried out while executing each movement. The higher is the number of points sampled from the trajectory, the lower is the error compared to the desired movement, but also higher is the number of computations to be carried out at run time.

Different approaches have been reported in the literature to solve the inverse kinematics problem. In this paper, we propose considering it as a mathematical optimization problem,
whose goal is to determine the values for the joint angles that minimize the difference between the expected and the real position of the end-effector. Therefore, the forward kinematics (FK) robot equations, which return the position of the end-effector using the joint angles as the input, is the cornerstone part of the cost function to be optimized. It should be noticed that the FK function is usually non-linear due to the presence of trigonometric functions. In this regard, and aiming at increasing the numerical stability of the solution near the singularities of the function, we have adopted the derivative-free Nelder-Mead method [7] as the optimization engine. The Nelder-Mead is an iterative numerical optimization method based on the simplex concept, which has been modified in this paper to enable the evaluation of the cost function in multiple vertices of the simplex simultaneously. Moreover, a novel strategy is proposed to define the initial conditions for each IK problem, which allows executing multiple points of the trajectory in parallel. This number can be changed at run time, trading it off with the roughness of the movements and power consumption.

Solving the inverse kinematics problem at run time with a high accuracy requires intensive computations, which have to be performed by an embedded system attached to the robot. In this regard, we propose using a heterogeneous Multi-Processor System-on-Chip (MPSoC) as the IK controller. Heterogeneous MPSoCs combine in a single device different types of computing fabrics, among which there are Central Processing Units (CPUs), Graphic Processing Unit (GPU) and Field Programmable Gate Array (FPGA) [8]. This combination generally results in higher computing performance, lower power consumption, and higher flexibility when compared to homogeneous multi-core solutions. These benefits come at the cost of an increase in design complexity, which results in higher time to market. For this reason, new tools and design flows are required to benefit from the advantages offered by MPSoCs. These tools simplify the management, coherency, and synchronization between the different device fabrics.

In this work, we propose the use of PREESM [9] as a rapid prototyping tool for multi/many-core embedded systems using a dataflow programming language as the entry point, extended with support for dynamically reconfigurable accelerators using the ARTICo\textsuperscript{3} framework [10]. ARTICo\textsuperscript{3} is a run-time reconfigurable processing architecture that enables hardware-accelerated high-performance embedded computing. The proposed solution improves the performance per watt of the system by offloading computationally intensive tasks to hardware accelerators that run on the FPGA fabric. Moreover, these accelerators can be adapted at run time when deployed on SRAM-based FPGAs, increasing the flexibility of the solution. In this context, a custom reconfigurable hardware accelerator has been developed to speed up the computation of the cost function. The accelerator has been described using the model-based design features provided in the Simulink programming environment, which has been integrated with the ARTICo\textsuperscript{3} toolchain. Algorithmic parallelization strategies have been devised to increase the use of multiple concurrent hardware accelerators.

The original contributions of this paper can be summarized as follows:

- A two-level parallelization strategy for an inverse kinematics solver based on the Nelder-Mead algorithm that provides inherent scalability.
- An adaptive hardware-based acceleration approach that supports the real-time scalability demanded by the algorithms in heterogeneous MPSoCs.
- The integration of PREESM, ARTICo\textsuperscript{3}, and Simulink tools for increased productivity in heterogeneous Multi-Processor System on Chip (MPSoC) designs.

The rest of the paper is divided as follows. In Section II, the IK problem and the Nelder-Mead method are presented. After a brief discussion on the state of the art in Section III, the proposal of the novel strategy is provided in Section IV. Thanks to the use of this optimization algorithm, a strategy for the simultaneous evaluation of multiple trajectory points is discussed in Section V. Descriptions of the hardware accelerator and the dataflow implementation exploiting such acceleration are reported respectively in Sections VI and VII. After the presentation of the results in Section VIII, conclusions and future work are discussed in Section IX.

II. TECHNICAL BACKGROUND

The inverse kinematics problem and the Nelder-Mead optimization algorithm are presented in this section.

A. THE INVERSE KINEMATICS PROBLEM

In several applications, the completion of the task assigned to a robotic arm requires the execution of a specific motion prescribed to the end-effector of the manipulator [11]. This makes it necessary to determine the joint parameters (i.e., the angles in the planes of rotation) that make the robot attain the desired position. This is known as inverse kinematics problem [12], in contrast with the forward kinematics problem [13], where the goal is to compute the position of the end-effector from the set of specified values for the joint angles.

Let us consider a robotic arm with n rigid joints, as shown in Figure 1, with three Degrees of Freedom (DoF) [14]. The position of the joints can be completely described by giving the value of the column vector $\theta = (\theta_1, \theta_2, \ldots, \theta_n)^T$ (where $\theta_i$ are joint angles). The position $s = (x, y, z)$ of the end-effector in the 3D space can be expressed as a function of the joint angles:

$$ s = f(\theta) \quad (1) $$

The function $f$ is the so-called FK. However, the goal of the IK problem is to determine the specific set of joint angles $\theta$ that brings the end-effector to the desired position $s$:

$$ \theta = f^{-1}(s) \quad (2) $$

While the FK has a unique solution and its success depends on whether the joints are allowed to perform the desired
transformations [11], [15], the IK may have zero, one or multiple solutions [14]. When the IK target point cannot be reached, the problem has no solution (meaning that a set of joint angles does not exist for that specific point), and it is said to be over-constrained. When multiple solutions are possible for the same target point, the problem is said to be under-constrained or redundant. It is worth noticing that the FK function can be divided into four main categories, as proposed by Aristidou in [14]:

1) **Analytical Solutions**: the forward equation is inverted to find a closed-form expression of the IK [16]. Finding this analytic solution can be very complicated in practice, due to the non-linearities existing in the forward form of the equation: it exists only for a restricted set of cases. An example of this approach can be found in [17], for the robot Pioneer 2, and in [18] for the humanoid robot Aldebaran NAO.

2) **Numerical Solutions**: all the iterative methods to solve the IK problem fall within this family. They require a certain number of iterations to converge into a satisfactory solution. They aim to minimize a cost function that is typically expressed by the Euclidean distance between the target point \( s_T \) and the reached point \( s_R \), with \( s_T = f(\theta_T^*). \)

In turn, the output of the algorithm is a set of arm parameters \( \theta_T^* = (\theta_{T1}, \theta_{T2}, \ldots, \theta_{Tn})^T \) that bring the end-effector as close as possible to the wanted 3D space coordinates:

\[
d(s_T, s_R) < \xi
\]

where the function \( d(s_T, s_R) \) is the Euclidean distance between \( s_T \) and \( s_R \).

Beyond its usage in robotics, the IK problem is also applied in other fields, such as computer graphics ([14], [27]), where human bodies are modeled as a set of rigid joints and links, and the IK techniques are used to animate them in the 3D space.

### B. NELDER-MEAD SIMPLEX ALGORITHM

The Nelder-Mead algorithm is an iterative method for mathematical optimization proposed in 1965 [7]. It belongs to the class of direct search methods [30]. Their main property is that they do not require computing or estimating the function’s derivatives to be optimized. Instead, they rely exclusively on the values of the objective function. The goal of the method is to solve the unconstrained problem:

\[
\min f(x)
\]

where \( x \in \mathbb{R}^n \), being \( n \) the number of parameters of the cost function to be optimized, that is defined as \( f : \mathbb{R}^n \to \mathbb{R} \).

More specifically, the Nelder-Mead is a simplex search method. Let us start the description of the algorithm by defining what a simplex and a vertex are.

**Definition 1**: A simplex \( S \) in \( \mathbb{R}^n \) is the convex hull of \( n + 1 \) vertices \( v_i \):

\[
S = \{v_i\}_{i=1,n+1}
\]

where every \( v_i \in \mathbb{R}^n \) is a vector of \( n \) coordinates within the domain space, also called vertex.
Algorithm 1: Original Nelder-Mead Algorithm [7]

Input: Initial simplex $S = \{x_i\}_{i=1,n+1}$

Output: The minimum of the objective function under test

1. Compute an initial simplex $S_0$
2. $S \leftarrow S_0$
3. while $\alpha(S) > tol$ do
   4. Sort the vertices of $S$ (Eq. 7)
   5. Compute $x_0$ (Eq. 8)
   6. Compute $x_r$ (Eq. 10)
   7. $f_r \leftarrow f(x_r)$
   8. if $f_r < f(\epsilon r)$ then
      9. Compute $x_\epsilon$ (Eq. 11)
      10. if $f_r < f(x_\epsilon)$ then
          11. Substitute $x_{n+1} \leftarrow x_\epsilon$ {Accept Expansion}
      else
          12. Substitute $x_{n+1} \leftarrow x_r$ {Accept Reflection}
      end
   else if $f_r < f_{n+1}$ then
      13. Substitute $x_{n+1} \leftarrow x_r$ {Accept Reflection}
   else if $f_r < f_{n+1}$ then
      14. Compute $x_{co}$ (Eq. 12)
      15. $f_{co} \leftarrow f(x_{co})$
      16. if $f_{co} < f_r$ then
          17. Substitute $x_{n+1} \leftarrow x_{co}$ {Accept Contraction}
      else
          18. Compute $x_i = x_1 + \delta(x_i - x_1)$ for $i = 2, \ldots, (n+1)$
             (Eq. 14)
          19. $f_i \leftarrow f(x_i)$ for $i = 2, \ldots, (n+1)$
      else
          20. Compute $x_i$ (Eq. 13)
          21. $f_i \leftarrow f(x_i)$
          22. if $f_i < f_{n+1}$ then
              23. Substitute $x_{n+1} \leftarrow x_i$ {Accept Contraction}
          else
              24. Compute $x_i = x_1 + \delta(x_i - x_1)$ for $i = 2, \ldots, (n+1)$
                 (Eq. 14)
              25. $f_i \leftarrow f(x_i)$ for $i = 2, \ldots, (n+1)$
          end
      end
   end
5. Update simplex $S$
6. return $x_1$ the best vertex of the last simplex

The simplex can be considered as a set of inputs for the function to be optimized. Each vertex $v_i$ is associated with a function value:

$$f_i = f(v_i) \text{ for } i = 1, n + 1$$  (6)

The iterative optimization process starts with an initial simplex $S$, which constitutes a set of possible guessed solutions for the problem. Then, the cost function in each of the $n + 1$ vertices of $S$ is calculated, and the obtained values are sorted:

$$f(v_1) \leq f(v_2) \leq \ldots \leq f(v_n) \leq f(v_{n+1})$$

$$\Rightarrow f_1 \leq f_2 \leq \ldots \leq f_n \leq f_{n+1}$$  (7)

where the best point (i.e., vertex) of the simplex is defined as $v_1$ while the worst point of the vertex is $v_{n+1}$.

Some basic operations used to update the simplex during the subsequent iterations of the algorithm are defined at this point.

**Definition 2:** Given a finite set of $k$ points $x_1, x_2, \ldots, x_{k-1}, x_k \in \mathbb{R}^n$, the centroid is defined as

$$x_0 := \frac{x_1 + x_2 + \ldots + x_{k-1} + x_k}{k}$$  (8)

From now on, the vertices of a simplex will be identified with the symbol $x$ to remark that such vertices are also possible solutions to the stated problem. The definition of the reflected point of the simplex is done with respect to the centroid point.

**Definition 3:** Given a simplex defined as $S = \{x_i\}_{i=1,n+1}$, only the first $n$ points/vertices are considered for computing its centroid:

$$x_0 := \frac{x_1 + x_2 + \ldots + x_{n-1} + x_n}{n}$$  (9)

Then, the reflected point is defined as follows:

$$x_r := x_0 + \alpha(x_0 - x_{n+1})$$  (10)

where $\alpha > 0$ is a parameter of the Nelder-Mead algorithm (typically $\alpha = 1$ as justified by Singer in [31], [32]).

Similarly, the expansion, contraction, and shrink points are defined:

**Definition 4:** Given a simplex $S = \{x_i\}_{i=1,n+1}$, its centroid $x_0$ and its reflected point $x_r$, the expanded point is defined as follows:

$$x_e := x_0 + \gamma(x_r - x_0)$$  (11)

where $\gamma > \alpha$ is a parameter of the Nelder-Mead algorithm (typically $\gamma = 2$).

**Definition 5:** There are two different kinds of contraction with respect to the centroid:

1) Contraction Outside:

$$x_{co} := x_0 + \beta(x_r - x_0)$$  (12)

2) Contraction Inside:

$$x_{ci} := x_0 + \beta(x_{n+1} - x_0)$$  (13)

where $0 < \beta < 1$ is a parameter of the Nelder-Mead algorithm (typically $\beta = \frac{1}{2}$).

**Definition 6:** Given a simplex $S = \{x_i\}_{i=1,n+1}$, the $n$ new vertices for the following iteration are calculated by shrinking it as follows:

$$\bar{x}_j := x_1 + \delta(x_j - x_1) \text{ for } j = 2, \ldots, n + 1$$  (14)
where $0 < \delta < 1$ is a parameter of the Nelder-Mead algorithm (typically $\delta = \frac{1}{2}$).

Given the previous definitions, the Nelder-Mead algorithm can be seen as a systematic update of the worst vertex of the simplex, following the rules described in Algorithm 1. The new calculated simplex is the input to the subsequent iteration of the recursive algorithm.

The simplex update process ends when any of the following conditions are met:

1) the best solution/vertex falls below a predefined quality threshold.
2) the simplex reaches a minimum size limit, in which the vertices of the simplex are too close.
3) a maximum number of iterations is reached without achieving one of the two previous conditions.

When the Nelder-Mead algorithm terminates with the condition (3) of the previous list, some authors (Butt in [33], Kelly in [34], and O’Neil in [35]) propose the restart of the algorithm. This technique and its implications are analyzed in section V of this paper.

A geometrical interpretation of the simplex is given in Figure 2 for a simplex $S = \{x_1, x_2, x_3, x_4\}$ defined in $\mathbb{R}^3$. According to the previous definitions, $x_0$ is the centroid of the green triangle made by the first best three points $x_1, x_2$, and $x_3$. $x_e$ is the reflection of the worst point with respect to the centroid. $x_e, x_{co}, x_{ci}$ are respectively the expansion, the contraction outside and the contraction inside. The smaller tetrahedron on the right side is the shrunk simplex.

A new strategy to parallelize the Nelder-Mead algorithm is proposed in this paper, making it possible to exploit hardware acceleration when evaluating the cost function, as described in Section IV.

### III. RELATED WORKS

A selection of works related to the implementation of IK solvers and the parallelization of the Nelder-Mead algorithm are reported in this section. Then, the main differences with our proposal are highlighted.

### A. PARALLEL INVERSE KINEMATICS SOLVER

The earliest solvers proposed for the inverse kinematics problem were variations of Jacobian-based methods running sequentially on microprocessors. Later on, the release of GPUs and MPSoC platforms enabled solutions that exploit parallelism at different levels, as discussed next.

Concerning GPU-based solutions, it must be mentioned the work in [36], where authors propose solving the IK problem adopting a parallel version of the Damped Least Squares (DLS) optimization algorithm. Xiaoyan et al. follow the same approach in [37], where an IK solver based on a parallel version of the Cycle Coordinate Descent (CCD) optimization algorithm is described. In this case, authors assume that the joint positions can be calculated independently from their previous positions, thus allowing the parallel calculation of all the positions simultaneously. This method introduces an error in the trajectory, which is mitigated by a damping factor. The number of threads executed in parallel is, at most, equal to the maximum number of joints of the arm. In [38], Aguilar et al. propose a CUDA-based approximate solver based on a parallel genetic algorithm. An annealing particle filter that allows the exploration of several IK solutions in parallel is presented in [39]. An alternative approach is described by Farzan et al. in [40]. Their strategy is to inject white noise on the Jacobian matrix of the robot and then evaluate all the possible evolutions of the matrices selecting the best path to reach the desired configuration of the end-effector. Solutions based on GPUs are capable of exploiting the parallelism of the IK algorithms, but with a lower power efficiency when compared to FPGAs. In the particular case of the works selected in this section, high-end GPUs have been used, as shown in Table 1. These devices are not appropriate to be used as platforms for embedded systems.

In the literature, some other works propose using custom accelerators on FPGAs to implement the computing demanding sections of IK solvers. A good example is shown in [41], where an analytical solver based on Conformal Geometric Algebra (CGA) is proposed. CGA is a mathematical framework widely used in computer graphics.
TABLE 1. Comparison of Existing Works on IK Parallelization.

| Method                     | IK Solver   | Platform          | Parallelism      | Dynamic Scalability | DoF |
|----------------------------|-------------|-------------------|-------------------|---------------------|-----|
| Harish et al. [36]         | Modified DLS| GPU AMD FirePro D300 | Algorithmic      | No                  | > 600 |
| Xiaoyan et al. [37]        | Modified CCD| GPU GeForce 8300GT | Algorithmic      | No                  | 4   |
| Aguilar et al. [38]        | Genetic Alg.| GPU Nvidia Tesla C1060 | Algorithmic      | No                  | 6   |
| Lehment et al. [39]        | Annealing   | GPU Nvidia GTX275 | Algorithmic      | No                  | -   |
| Farzan et al. [40]         | Jacobian    | GPU Intel Xeon E5520 | Algorithmic      | No                  | 6   |
| Hildenbrand et al. [41]    | CGA         | FPGA Virtex 2VP70 | Hardware          | No                  | 2   |
| Gac et al. [42]            | Analytic    | FPGA Altera Cyclone IV | Hardware          | No                  | 3   |
| Köpper et al. [43]         | iB2C        | FPGA Altera Cyclone IV-E | Hardware          | No                  | 7   |
| Proposed Strategy          | Nelder-Mead | Zynq UltraScale+  | Hardware (Multiple) / Algorithmic | Yes | 5   |

The proposed accelerator executes dataflow graphs extracted from the IK analytical equations. A similar approach is followed in [42], where the authors propose a hardware co-processor to accelerate computing-intensive mathematical operations in the IK analytical solution. Also, Köpper and Berns in their proof-of-concept work [43] propose the realization of a hardware accelerator for solving the IK using the method called Integrated Behaviour-Based Control (iB2C). In all these works, parallelism is limited to the spatial distribution of the solver datapath throughout the programmable logic. However, neither algorithmic parallelization nor multi-accelerator solutions have been exploited in FPGA-based solutions.

Table 1 summarizes the main features of the state-of-the-art works described in this section and their comparison with the proposed system. It is worth noting that, in this paper, we bring together multiple parallelization strategies: the intrinsic hardware parallelism offered by the FPGA is extended with a multi-accelerator arrangement, and combined with a two-level algorithm parallelization. To the best of author’s knowledge, the proposed solution is the only one providing adaptability at run time. Our implementation targets specifically MPSoC, which are more suitable for the embedded domain than high-end GPUs.

**B. NELDER-MEAD PARALLELIZATION**

Different parallelization strategies have been proposed in the literature for the Nelder-Mead algorithm. Most significant examples are reported next, highlighting the main differences with the approaches proposed in this paper.

A parallel version of the Nelder-Mead algorithm was proposed by Lee et al. in [44], targeting multiprocessor implementations. Essentially, given a function with \( j \) variables (i.e., vertices) and \( p \) processors (with \( j > p \)), it assigns to each processor the calculation of one vertex for the new simplex. This way, every algorithm iteration updates the simplex with \( p \) new vertices (instead of one, as it happens in the classical sequential algorithm). To make it possible, the authors propose considering only \( j - p \) vertices to calculate a new vertex by each processor. This strategy alters the original algorithm and introduces an error that must be evaluated for the specific function to be optimized. This parallelization strategy was improved in [45] by proposing the update of a local copy of the worst vertex for each processor, thus reducing the communication cost in a context of distributed memory architectures. Differently, the parallel version of the Nelder-Mead algorithm proposed in this work does not alter the algorithm’s original behavior. Additionally, we provide a solution that targets heterogeneous MPSoCs: clusters of powerful processors are not suitable for data processing in the embedded domain.

Mariano et al. propose in [46] the speculative execution of some essential functions of the algorithm upon FPGA. Fundamentally, in the hardware version proposed in [46], all the "decision paths" of the algorithm are computed in parallel. Then, at the end of the computation, a multiplexer selects the right output based on a proposed classification of the reflection point quality, called the Reflection Vertex Test. It consists in classifying the reflection point as **good**, **very good**, **weak**, **very weak**. This preliminary step produces an output that selects the Nelder-Mead path with a multiplex. In their proposal, not only the cost function, but the whole iteration of the algorithm is implemented in HDL targeting its implementation on FPGA. However, in the experimental results reported in [46], the critical path on the FPGA logic leads to a maximum clock frequency of 3.6 MHz, which prevents its usage in real applications. Differently to this work, we propose rearranging some of the Nelder-Mead operations to allow the parallel evaluation of the cost function of multiple vertices at the same time. This way, only the cost function is offloaded onto the FPGA, instead of the whole Nelder-Mead algorithm.

Recently, machine learning techniques were also applied to accelerate the Nelder-Mead method by parallel predictive evaluation, such as the work in [47]. Here, the authors use a Gaussian process regression model [48] as a surrogate of the real target function, and then perform a Monte Carlo Simulation to determine the candidate points to be speculatively evaluated. Thus, the approach adopted by Ozaki et al. is not suitable to be implemented in embedded systems due to the high computational capability required by the Monte Carlo Simulation.

In our work, a new version of the Nelder-Mead is proposed. Differently from other state-of-the-art alternatives, we demonstrate that the result is not altered by the rearrangement of the speculative execution of the basic algorithm operations. Additionally, we provide a high-level description of it by using the dataflow MoC. A custom hardware
accelerator is designed using commercial tools to implement the cost function to be optimized. The tests are performed on an embedded MPSoC, providing run-time adaptation and scalability.

Table 2 provides a summary of the discussed existing methods.

### IV. PARALLELIZATION OF THE NELDER-MEAD METHOD

The first strategy proposed in this paper to parallelize the inverse kinematics solver is application-independent and focuses on the computation of the Nelder-Mead method. In particular, we propose the modification of the algorithm to evaluate the cost function in the multiple vertices of the simplex in parallel.

In the baseline Nelder-Mead algorithm (described in Algorithm 1), most of the computing time is spent in evaluating the cost function in every single vertex of the simplex. The more complex the function to be optimized is (like in the forward kinematics), the higher the computational cost of the algorithm becomes. The rest of the operations of the algorithm are straightforward, including comparisons and equations (10) and (13), which have a complexity of \(O(n)\), where \(n\) is the number of elements in the vertex.

It is not possible to predict the latency of each outer loop iteration of the algorithm, as well as the number of cost function evaluations. This is due to the different execution paths the algorithm may follow. In the best case, the expanded point calculated is accepted after just two function evaluations (line 12 of Algorithm 1). In the worst case, a shrink of the whole simplex may be performed; it implies four cost function evaluations followed by a massive contraction of \(n\) vertices and, consequently, \(n\) extra function evaluations. Therefore, the highest computational cost (in terms of numbers of cost-function evaluations) occurs when the shrink operation has to be performed (shown in Algorithm 2).

By focusing on the shrink operation, it must be noticed that every iteration within the loop (line 1 of Algorithm 2) does not depend on the previous one. In other words, every iteration of this loop can be performed independently of the others. Thus, the algorithm can be rewritten, as shown in Algorithm 3.

Further optimizations can be derived from the following observations:

1) The reflected point (see equation (10)) depends on the centroid and the vertices of the initial simplex.

2) The expanded point also depends on the centroid and the vertices of the initial simplex. In fact, replacing equation (10) within equation (11):

\[
x_{r} = x_{0} + \gamma(x_{f} - x_{0})
\]

\[
x_{r} = x_{0} + \gamma(x_{f} + \alpha(x_{0} - x_{n+1}) - x_{f})
\]

\[
x_{r} = x_{0} + \gamma\alpha(x_{0} - x_{n+1})
\]

(15)

3) The contracted points also depend on the centroid and the vertices of the initial simplex. Replacing (10) with the original expression in (12):

\[
x_{co} = x_{0} + \beta(x_{f} - x_{0})
\]

\[
x_{co} = x_{0} + \beta(x_{f} + \alpha(x_{0} - x_{n+1}) - x_{f})
\]

\[
x_{co} = x_{0} + \beta\alpha(x_{0} - x_{n+1})
\]

(16)

Based on the previous considerations, the new parallelization strategy is proposed. It consists of changing

### Algorithm 2 Shrinking Operation in the Original Nelder-Mead Algorithm (lines 23-24 and 31-32 of Algorithm 1)

**Input:** Initial simplex \(S = [x_{i}]_{i=1,n+1}\)

**Output:** A new simplex where all the vertex are shrunk towards the best point \(x_{1}\)

1) \(for\ i \leftarrow 2\ to\ n + 1\ do\)

2) \(\text{Compute } x_{i} \leftarrow x_{1} + \delta(x_{i} - x_{1})\)

3) \(f_{i} \leftarrow f(x_{i})\)

4) \(return\ Shrunk\ Simplex\)

### Algorithm 3 Proposed Parallel Shrink Operation for the Nelder-Mead Algorithm

**Input:** Initial simplex \(S = [x_{i}]_{i=1,n+1}\)

**Output:** A new simplex where all the vertex are shrunk towards the best point \(x_{1}\)

1) \(\text{do in parallel}\)

2) \(\text{Compute } x_{2} \leftarrow x_{1} + \delta(x_{2} - x_{1})\)

3) \(\ldots\)

4) \(\text{Compute } x_{n-1} \leftarrow x_{1} + \delta(x_{n-1} - x_{1})\)

5) \(\text{Compute } x_{n} \leftarrow x_{1} + \delta(x_{n} - x_{1})\)

6) \(\text{do in parallel}\)

7) \(f_{2} \leftarrow f(x_{2})\)

8) \(\ldots\)

9) \(f_{n+1} \leftarrow f(x_{n+1})\)

10) \(f_{n} \leftarrow f(x_{n})\)

11) \(return\ Shrunk\ Simplex\)
the order of the algorithm’s operations to compute a priori all the possible new vertices in Algorithm 1. Therefore, the centroid, the reflected point, the expanded point, the contraction-outside point, and the contraction-inside point are all calculated over vertices of the initial simplex \( S = \{v_i\}_{i=1,...,n+1} \), independently from each other. Hence, all these operations can be potentially executed in parallel. Once the initial evaluations are carried out, the rest of the iteration is reduced to computing simple operations, such as comparisons and substitutions. Finally, the simplex is updated at the end of the iteration. This way, it is possible to rewrite the Nelder-Mead Algorithm, as shown in Algorithm 4.

It is worth noting that the proposed strategy is applicable whenever the Nelder-Mead algorithm is used, resulting in particularly convenient implementations when combined with hardware acceleration. It does not depend on the particular function to be optimized, so it is not restricted to the inverse kinematics problem tackled in this work.

V. PARALLELIZATION AT TRAJECTORY LEVEL

The second level of parallelism proposed in this work to accelerate the execution of the IK solver is application-specific and targets the computation of multiple points of the trajectory simultaneously.

Let us remark that the problem consists in bringing the end-effector of the robotic arm from a point \( A \) to a point \( B \) in a 3D space, as shown in Figure 3. Whenever the arm is moving in a space where no collisions are possible, and the followed trajectory does not matter, the simpler solution consists of solving the IK to find the right set of angles \( \theta_B^* \) such that \( \mathbf{s}_B = f(\theta_B^*) \).

![Figure 3. Robotic arm correctly moving along a generic trajectory from a point A to a point B.](image)

However, in many applications, it is necessary to control the whole trajectory meticulously. In that case, the trajectory is sampled to extract intermediate points. Afterward, the IK problem is solved for each of these intermediate points, obtaining the combination of joint angles to position the end-effector in all the sampled points of the trajectory. The smaller the distance between two adjacent sampled points is, the higher the accuracy of the movement will be. However, it also increases the computational costs and latency. The described scenario is summarized graphically in Figure 4 where every \( A_i \) is a point in a 3D space \((x_i, y_i, z_i)\). For the sake of simplicity, the trajectory drawn in this figure is a straight-line segment, but it can be any curve.

The Nelder-Mead algorithm uses the arm angles corresponding to the previous point in the trajectory \( (\theta_{i-1}) \) as the initial condition (i.e., the vertices where the initial simplex

| Algorithm 4 Proposed Parallel Nelder-Mead Algorithm |
|-----------------------------------------------|
| **Input:** Initial simplex \( S = \{x_i\}_{i=1,...,n+1} \) |
| **Output:** The minimum of the objective function under test |
| **while** \( \sigma(S) > tol \) **do** |
| 1. Sort the vertices of \( S \) {Eq. 7} |
| 2. Compute \( x_0 \) {Eq. 8} |
| **do in parallel** |
| 3. Compute \( x_f \) {Eq. 10} |
| 4. Compute \( x_e \) {Eq. 15} |
| 5. Compute \( x_{co} \) {Eq. 16} |
| 6. Compute \( x_{ci} \) {Eq. 13} |
| 7. Compute \( x_i = x_1 + \delta(x_i - x_1) \) for \( i = 2, \ldots, (n + 1) \) {Parallel Shrink Algo. 3} |
| **do in parallel** |
| 8. if \( f_r < f_1 \) **then** |
| 9. if \( f_e < f_r \) **then** |
| 10. Substitute \( x_{n+1} \leftarrow x_e \) {Accept Expansion} |
| 11. else |
| 12. Substitute \( x_{n+1} \leftarrow x_r \) {Accept Reflection} |
| 13. else if \( f_i < f_r < f_{n+1} \) **then** |
| 14. Substitute \( x_{n+1} \leftarrow x_r \) {Accept Reflection} |
| 15. else if \( f_n < f_r < f_{n+1} \) **then** |
| 16. if \( f_{co} < f_r \) **then** |
| 17. Substitute \( x_{n+1} \leftarrow x_{co} \) {Accept Contraction Outside} |
| 18. else |
| 19. Substitute \( x_i \) for \( i = 2, \ldots, (n + 1) \) |
| 20. else if \( f_{ci} < f_{n+1} \) **then** |
| 21. Substitute \( x_{n+1} \leftarrow x_{ci} \) {Accept Contraction Inside} |
| 22. else |
| 23. Substitute \( x_i \) for \( i = 2, \ldots, (n + 1) \) |
| 24. Update simplex \( S \) |
is computed) to compute the subsequent set of angles. The situation is schematically described in Figure 5.

![FIGURE 5. The IK problem at step i-th needs the solution of the IK of step (i-1)-th.](image)

Solving the previous IK problem before starting with the next point prevents the computation of multiple points in parallel. However, having the set of initial arm conditions \( \theta \) is not strictly necessary for the Nelder-Mead algorithm to find a correct solution. It is even possible to use a random vector \( \theta_r \) as the initial arm conditions since the end-effector will still reach the target point if it is in the scope of the robotic arm. This situation is graphically shown in Figure 6.

![FIGURE 6. The IK problem using a random set of initial conditions for every point.](image)

However, when the initial conditions of the angles are unknown, the robotic arm may reach the target position with a completely different combination of joint angles compared to the previous point (see Figure 7). It should be recalled that the IK problem can have multiple solutions for the same target point. As a consequence, this strategy may cause random abrupt and rapid movements of the joints between two consecutive trajectory points. These movements increase the power consumption of the robot, and they could eventually cause severe damage to the physical robot structure.

From the perspective of the optimization algorithm, it is worth remembering that the initial simplex of the algorithm is a cloud of \( n + 1 \) vertices built around the input vector \( \theta_i \) in an \( n \)-dimensional space. In line with [49], in this work, the simplex is initialized by considering \( \theta_i \) coming from a previously performed step as one of the initial vertices (as shown in Figure 5). Then, given \( \theta_i \), the cloud of points is generated as follow:

\[
\theta_i + \tau_j e_j \quad \text{for} \quad j = 1, \ldots, n
\]  

(17)

where \( e_j \) is the unit vector in the \( j \)th coordinate, and \( \tau_j \) is a parameter generally chosen as follow:

\[
\tau_j = \begin{cases} 
0.05 & \text{if } j = 1 \\
0.00025 & \text{if } j \neq 1 
\end{cases}
\]

(18)

For more information about the selection criteria for the initial simplex, see [50] and [51]. If \( \theta_i \) are the angles for the previous point, the application of Nelder-Mead will make this cloud evolve by getting closer to the expected minimum of the function. The result output \( \theta_{iC1} \) is more likely to be similar to the input \( \theta_i \). However, if the initial set \( \theta_i \) is a random set of values, the algorithm will make the simplex evolve until any minimum is reached, not necessarily close to the previous configuration of the arm.

As an alternative, we propose a trade-off solution to parallelize the computation of multiple points of the trajectory. The proposal is based on dividing the set of points of the trajectory in multiple subsets of \( N \) consecutive points, as shown in Figure 8. The same vector of angles \( \theta \) is used as the input for all the points in the subset, increasing the likelihood that the algorithm converges to close solutions. Since all the points in the subset share the same initial point, they can be computed in parallel independently from each other. In turn, the different subsets are computed sequentially, using the angles provided by the last IK problem in the previous subset.

![FIGURE 7. Robotic arm moving along a generic trajectory between two points A to B with random movements.](image)

![FIGURE 8. Parallel propagation of initial condition for subsets of \( N \) points.](image)

In section II-B, it was mentioned that the Nelder-Mead algorithm can reach a maximum number of iterations without finding a solution that satisfies the convergence criteria. This is a rare situation that happens when the worst vertex of
the simplex is relatively close to the centroid. Under this condition, the calculated new vertex is necessarily close to the worst and centroid vertices, since all the reflection, contraction inside and outside operations produce similar results. Thus, the simplex does not progress substantially in further iterations.

The solution proposed in the literature is to restart the algorithm using a different random simplex as the starting point [33]. This way, the Nelder-Mead algorithm tries to find another numerical path to reach the solution. The problem with this approach is the effect shown in Figure 7: the choice of a random set of joint parameters brings the end-effector to the target point but with random abrupt movements.

Instead, we propose to introduce in the equation a small random value \( \sigma \), which acts as a little perturbation noise:

\[
\theta_i + (\sigma \cdot \tau)e_j \quad \text{for } j = 1, \ldots, n \quad (19)
\]

with \( \sigma \) randomly generated with a continuous uniform distribution within the range \([-\frac{\pi}{2}, \frac{\pi}{2}]\). This approach modifies the numerical path followed by the algorithm to converge. At the same time, the initial set of joint parameters remains close enough to the original initial parameters to avoid the abrupt movements caused by the selection of totally different initial conditions. For this reason, a small enough value has been empirically selected for the parameter \( \tau \), as reported in Equation 18.

VI. HARDWARE ACCELERATION OF THE COST FUNCTION

Complementary to the two-level algorithmic strategies for the parallelization of the IK, we have designed a custom hardware accelerator to evaluate the cost function to be optimized by the Nelder-Mead algorithm (i.e., the direct kinematics function). The kinematics equations of the robotic arm are described here using the Denavit-Hartenberg (DH) representation (introduced in [52]), in which four parameters define every pair joint-link. In the case of WidowX [53], the robotic arm used as a demonstrator in this work, the corresponding DH parameters are summarized in Table 3. These values are available in its datasheet.

**TABLE 3. DH parameters for WidowX robotic arm.**

| \( i \) | \( a_i \) (cm) | \( d_i \) (cm) | \( \alpha_i \) (rad) | \( \theta_i \) (rad) |
|-------|-------------|-------------|----------------|----------------|
| 1     | 0           | 0           | 0              | \( \theta_1 \)   |
| 2     | 15          | 0           | \( -\pi \)     | \( \theta_2 \)   |
| 3     | 5           | 0           | 0              | \( \theta_4 \)   |
| 4     | 15          | 0           | 0              | \( \theta_5 \)   |

Given an open chain of \( n + 1 \) links connected by \( n \) joints, the position and the orientation of the frame \( n \) with respect to frame 0 is given by the equation:

\[
T_n^0(q) = A_i^0(q_1)A_2^0(q_2) \ldots A_n^0(q_n) \quad (20)
\]

where \( q_i \) is each of the sets of parameters shown in Table 3. In turn, the Transformation Matrix \( A_i^{i-1} \) brings the reference system from \( O_{i-1} \) to \( O_i \), being \( O_i \) the origin of a cartesian coordinate system centered on joint \( i \):

\[
A_i^{i-1} = \begin{bmatrix}
    c_{i \theta} & -s_{i \theta} & c_{i \theta}a_{i \theta} & c_{i \theta}c_{i \theta} - s_{i \theta}a_{i \theta} & a_{i \theta}c_{i \theta} \\
    s_{i \theta} & c_{i \theta} & s_{i \theta}a_{i \theta} & s_{i \theta}c_{i \theta} + c_{i \theta}a_{i \theta} & a_{i \theta}s_{i \theta} \\
    0 & 1 & 0 & 0 & 1
\end{bmatrix}
\]

(21)

where the notation \( c_x \) indicates \( \cos(x) \), while \( s_x \) indicates \( \sin(x) \).

The variables in Table 3 only depend on the \( \theta_i \) angles so we can replace \( q_i \) with \( \theta_i \). Hence, the FK equation of the robotic arm used in this work can be written as:

\[
T_3^0(\theta) = A_1^0(\theta_1)A_2^0(\theta_2) \ldots A_5^0(\theta_5) \quad (22)
\]

This is the equation implemented in the accelerator, in which a 32-bits fixed point data representation is used. The custom hardware accelerator has been integrated within ARTICO3, a multi-accelerator architecture that provides transparent reconfiguration and scalable performance, dependability, and power consumption at run time. The ARTICO3 architecture is divided into a static part (i.e., the infrastructure that delivers data to the different accelerators) and several dynamic components (i.e., the hardware accelerators). The static part provides a DMA-capable data-path whose internal structure can be configured at run time to support different processing profiles. The dynamic part is divided into several reconfigurable partitions (or slots), where the hardware accelerators are loaded using Dynamic and Partial Reconfiguration (DPR) when required. The ARTICO3 infrastructure and the accelerators work at 100 MHz. The reader is referred to [10] and the ARTICO3 website [54] for further details on this open-source framework.

The dynamic scalability provided by ARTICO3 enables the use of multiple instances of the IK accelerator in the programmable logic available in the MPSoC at run time. The different evaluations of the cost function that are carried out during one Nelder-Mead iteration are submitted as a DMA burst to a single accelerator. In turn, the points in the sub-trajectories that are computed simultaneously are forwarded to different ARTICO3 accelerators. Hence, the proposed hardware acceleration scheme provides the support required by the two-level algorithmic parallelism proposed in this paper. Moreover, the dynamic scalability enabled by ARTICO3 also leads to outstanding flexibility. By dynamically changing the number of hardware accelerators working in parallel, the number of parallel evaluations of the cost function also changes, and so does time required to compute the trajectory.

Another important requirement for computer systems is dependability, especially in areas such as airspace, nuclear control, and biological medicine, as explained by Peng et al. in [55]. Two of the most commonly used methods for fault mitigation in a harsh environment are the Double Module Redundancy (DMR) and Triple Module Redundancy (TMR). Those techniques are mandatory when dealing with
SRAM-based FPGA [56] in harsh environments. They consist of using three (two for the DMR) physical-different but functionally-identical hardware systems processing the same data. At the end of the chain, a configurable-voter and error counters are introduced to detect potential faults, pick the correct results, and trigger a self-healing mechanism (when provided by the architecture). All these features are naturally supported by the ARTICO\textsuperscript{3} hardware architecture and by its runtime functions [54]. In section VIII-B, a Design Space Exploration (DSE) including the TMR and DMR fault-mitigation operating modes are included.

As opposed to previous works relying on the ARTICO\textsuperscript{3} infrastructure, the cost function accelerator’s description has been made using model-based design techniques. The model-based design increases abstraction in the design process, eliminating the use of text documents, such as source code or technical reports, as the primary mechanism to transfer project information. Visual inspection, simulation, and formal verification of models are used instead.

Model-based design methodologies required the support of specialized tools. In this work, we have used Simulink, extended with the capabilities offered by Xilinx System Generator for the implementation of hardware accelerators directly from graphic descriptions. Since System Generator does not provide native support for the generation of dynamically reconfigurable accelerators, we have integrated it with the ARTICO\textsuperscript{3} toolchain. The integration relies on a MATLAB script that automatically instantiates the input/output memory banks required by ARTICO-compliant accelerators, together with the required finite state machines that control data transfers between the local memory banks from the ARTICO\textsuperscript{3} kernel wrapper and the internal user logic. A detail of the cost function accelerator implemented in Simulink, together with the interface with the ARTICO\textsuperscript{3} kernel wrapper, is shown in Figure 9. It receives the coordinates of the destination point where the end-effector must be positioned and the joint angles as the inputs. The accelerator computes the position in space that corresponds to the input angles, and provides the distance from this computed position to the destination, as the cost function to be used by Nelder-Mead. We have consciously avoided including details on the accelerator datapath in Figure 9, for the sake of clarity. Internally, the accelerator datapath is composed of a set of CORDIC units that compute the trigonometric expressions, with registers, multiplexers, and arithmetic units in charge of computing the equation described above (Equation 20).

The accelerator can process sequentially the number of points indicated as a scalar input. All these points are received in a single DMA burst through the ARTICO\textsuperscript{3} infrastructure. Attached to each data port, we have included input and output First-In First-Out queue (FIFO), and the controllers used to store and sequence the multiple points received in a burst (see blocks with a red edge in Figure 9). This hardware wrapper has been designed to be compatible with the token buffers used by PREESM [9]. It must be noticed that these points are executed in a single ARTICO\textsuperscript{3} invocation from the embedded processor, but they are computed sequentially inside the accelerator. However, this is not a limitation since the time required to compute a single point in hardware requires orders of magnitude below the required time to transfer data to the hardware accelerators.

The custom accelerator described in this section is specifically developed for the robotic arm used in this work. However, its datapath could be easily modified to be used with robotic arms with different dimensions, and even with a different number of DoF.
VII. DATAFLOW IMPLEMENTATION WITH HARDWARE ACCELERATION

The proposed IK solver is composed of integrated software and hardware components, featured with dynamic adaptation capabilities. This approach makes necessary the use of novel techniques to describe both the software and hardware partitions of the algorithm in a unified manner. The approach proposed in this work is based on the use of dataflow MoCs. The rationale behind this decision is the natural expressiveness of parallelism and the high level of analyzability inherent to dataflow descriptions [57]. These are also the reasons why the dataflow MoC is commonly used to model algorithms in many areas such as video and audio processing [58], [59], computer vision algorithms [9], and telecommunications [60].

In a dataflow MoC, an application can be described by using actors that communicate through FIFOs. Every actor is associated with a specific task or job. The actor performs its task by processing the data available through the FIFO-in buffers and producing other data on the FIFO-out buffers. The triggering of an actor is known as firing, while the data-packet to process is a token. An actor fires when enough data tokens are available on the input ports.

In this work, we use an enhanced version of the PREESM tool that, apart from automatically generating code to deploy the application on a multi-core, it also may offload part of the computation onto the FPGA fabric by making use of the ARTiCo\textsuperscript{3} architecture.

By using a unique dataflow representation of the entire algorithm through the Parameterized and Interfaced Synchronous DataFlow (PiSDF) (a dataflow semantic which gives the possibility to use static and dynamic parameters to modify the actor firing rules [61]) and the description of the architecture using the System-Level Architecture Model (S-LAM) (high-level description of a generic hardware infrastructure [62]), PREESM can prototype any application rapidly. Further details on PREESM components can be found in [9].

In Figure 10, the PiSDF diagram of the IK problem solver, including the actors and their interconnections, is provided. Specifically, the actors of the dataflow graph were enumerated from 1 to 5. Below, we use the same enumeration to describe the graph reported in the figure:

1) This set of labeled actors are in charge of all the fundamental operations described from line 5 to 9 of the Algorithm 4;
2) This actor packs the vertices of the simplex to create the data structure (i.e., token) ready to be processed by the next actor of the network;
3) This actor is in charge of evaluating the cost function. The tool can recreate automatically as many instances as necessary of the same actor to fulfill the firing rules of the PiSDF.\textsuperscript{1} The line from 10 to 15 are processed here;
4) This actor dispatches the processed vertices of the simplex to the right input-FIFO of the Nelder-Mead algorithm;
5) The core of the algorithm is executed in this last actor, which performs just comparison and one substitution. The previous actors in the graph already completed all the cumbersome processes. Lines from 16 to 33 of our proposed Algorithm 4 are executed here.

\textsuperscript{1}For more information, documentation, and tutorials on the use of PiSDF within PREESM, the reader is referred to the official website https://preesm.github.io/
VIII. EXPERIMENTAL RESULTS AND DISCUSSION

The solution proposed in this paper for the IK problem is adaptive since it enables the real-time modification of i) the number of trajectory points to be processed in parallel and ii) the number of hardware accelerators to be used in the system. No need for rewriting and rethinking the whole application is required to perform these changes. The integrated tools (i.e., the enhanced version of PREESM [9] and ARTICO³ [10]) will provide the automatic code generation from a unique dataflow application representation. These features are used in this section to provide a DSE of the architecture proposed for the IK.

A. EXPERIMENTAL SETUP

The proposed architecture has been implemented on the Xilinx Zynq UltraScale+ MPSoC included in the ZCU102 Evaluation Kit [63]. The Processing System (PS) in the Zynq UltraScale+ MPSoC features the ARM Cortex-A53 64-bit quad-core processor which, in turn, runs a Linux-based Operating System (OS). The same chip is equipped with programmable logic (i.e., an FPGA) for custom designs. Many other peripherals are included within the board, including a communication ethernet interface, a UART interface, and a variety of sensors. Among these is the TI INA 226 [64] current shunt and power monitor, which is accessible from the PS with a specific Linux driver, and provides power measurements of the Zynq UltraScale+ device. Results discussed in section VIII-B have been obtained using this mechanism.

Figure 11 shows the layout of the hardware acceleration infrastructure of the proposed IK system, where red boxes have been used to highlight the eight reconfigurable slots. The rest of the logic resources in the PL corresponds to the ARTICO³ data delivery module.

The WidowX Robot Arm Kit [53] developed by Trossen Robotic was chosen for testing the novel IK approach. This robot has five DoF. The lengths of the links are fixed and are not considered further DoF. The DH parameters of this specific robotic arm have been reported in Table 3 while discussing the hardware accelerator. Moreover, a python simulator was developed based on the model of the robotic arm. The simulator of the robotic arm was used for functional verification of the proposed system, i.e., to prove that the end-effector follows the planned trajectory when controlled with the proposed solver, since it allows the evaluation of novel IK strategies without mechanical risks. Figure 12 shows the arm simulator plotting in the 3D space, where the position of the arm for each point of the trajectory is represented. In turn, the experimental results shown in the rest of this section were directly measured with sensors (power and energy) and software timers (execution time) integrated on the MPSoC platform.

B. EXPERIMENTAL RESULTS

Figure 13 shows the time necessary to complete one Nelder-Mead iteration when the number of hardware accelerators in ARTICO³ increases, for different amounts of points computed in parallel. The time reported is normalized to the number of points processed in each experiment, so all the series can be compared directly.

Experimental results show three possible scenarios, depending on the ratio between the number of hardware accelerators and the number of trajectory points processed in parallel. When the number of accelerators is equal to the...
number of parallelized points, the time needed to compute one Nelder-Mead iteration reaches a minimum. Therefore, the most efficient implementation always corresponds to the number of concurrently processed points matching the number of hardware accelerators. When the number of accelerators exceeds the number of parallelized trajectory points, the execution time increases due to accelerator management overheads in the ARTiCo3 runtime. Some accelerators work on dummy data, increasing power consumption, and resource occupancy without improving workload processing efficiency. However, this is not a limitation for the proposed inverse kinematics solver since ARTiCo3 allows readjusting the number of accelerators at run time when the number of points to be processed in parallel is modified. In turn, if the number of parallel trajectory points is higher than the number of hardware accelerators, the ARTiCo3 runtime serializes the access to the accelerators. Hence, there is no performance gain in increasing the number of points processed in parallel above the number of hardware accelerators configured in the FPGA. The points beyond the optimal are shown as dashed lines in Figure 13 for each configuration. It would be possible to continue decreasing execution time by using a larger MPSoC in which more accelerators could be instantiated.

We consider now how parallelism affects the total time to process a trajectory. Clearly, this time is affected by the total number of iterations of the Nelder-Mead algorithm. As such, it should be noted that for the same trajectory (same initial arm configuration and same destination point in the 3D space), changing the number of parallel points to be concurrently processed results in a different number of total iterations. This occurs because the initial conditions for all the points of the trajectory (see Figure 8) are automatically modified when the number of points processed in parallel changes. Different initial conditions result in a different number of iterations for the Nelder-Mead algorithm to converge.

For this reason, the effect of changing the number of points computed in parallel cannot be measured for a single trajectory. Instead, it is statistically measured for 100 trajectories, randomly chosen within the workspace of the robotic arm, with 840 points each. Results are reported in Figure 14 as a box plot diagram, and the average values are reported in Figure 15. This figure shows how, on average, the more points are evaluated in parallel by an increasing number of custom hardware accelerators, the lower is the total number of iterations, and therefore, the total time needed to process the whole trajectory. In this experiment, the number of accelerators is equal to the number of points processed in parallel, up to the maximum number of eight accelerators.

The power consumption as well as the average execution time for 100 trajectories in different configurations (i.e., different number of ARTiCo3 slots) have also been measured and represented in Figure 16. As shown, increasing the number of hardware accelerators on the FPGA always
increases the power consumption of the whole system. However, it must be noted that the entire task is completed sooner: the system power request is limited to a smaller amount of time, thus decreasing the energy necessary to complete the process. The energy consumed for the different number of hardware accelerators is represented in Figure 17, where we can see that the trend mentioned before is not always respected: there is a minimum when four accelerators are used. Using more than four ARTiCo³ slots results in a too limited performance gain (see Figure 16) for the increase produced in power consumption.

FIGURE 17. Energy used by the programmable logic of the MPSoC by increasing the number of hardware accelerators (i.e., the number of points to be calculated in parallel).

Previous results show that there is no single solution that minimizes execution time and energy at the same time. Existing design points are represented in the two-dimensional diagram shown in Figure 18, in which the Pareto optimal solutions can be identified. By analyzing the operation conditions (such as the battery level, the required accuracy and speed of the trajectory, or the environment radiation level), a system adaptation manager, either autonomously or with a human-in-the-loop, must decide which is the optimal point in which the system must work. This decision results from the DSE involving energy consumption, hardware resource utilization, dependability, and execution time, which is detailed next.

The first parameter to be analyzed is resource utilization. This metric is proportional to the number of hardware accelerators configured in the FPGA. When performance is introduced in the analysis, it has already been discussed that optimal operation points correspond to the situation in which the number of hardware accelerators equals the number of parallelized points. However, this scenario changes when power consumption is introduced in the analysis. The blue curve labeled as No Redundancy in Figure 18 represents the operation points that are Pareto optimal in terms of power and performance for a given amount of hardware resources. Choosing a configuration on the blue curve ensures the system works in a Pareto-optimal configuration. Finally, dependability can be introduced in the analysis. In this regard, ARTiCo³ offers the possibility of introducing automatically dual or triple modular redundancy, respectively DMR and TMR. As explained in [10], these working-modes physically replicate two or three times the hardware for processing the same set of data, giving the possibility to detect and correct occasional hardware-faults caused by radiation. The price to pay, as can be noted by the diagram in Figure 18, is higher power consumption of the platform and, consequently, even higher energy consumption: some accelerators are not used for accelerating but to replicate processing. Despite more energy and less speed-up are achieved, this is the solution to be adopted in harsh environments such as nuclear power plants or space missions.

The proposed scheme is also adaptive in terms of the roughness of the arm movement. When the trajectory parallelism increases beyond a limit, the proposed strategy for trajectory-level parallelism might cause the solver to find very different solutions for two consecutive points in the trajectory, as has been explained in Section V. As a consequence, the joints may describe abrupt and rapid movements. This situation is shown in Figure 19, where spikes appear when plotting the joint-angles as a function of the trajectory points. Here, the results correspond to a hundred-point trajectory when sixteen points are processed in parallel. This roughness is a measurement of the quality of the solutions that also have to be considered in the design space exploration, apart from
computing acceleration, energy, and power consumption. However, it must be highlighted that this is a rare situation that appears when the robotic arm end-effector needs to cover a long distance while the trajectory is segmented in few points and, at the same time, a large number of parallel points is chosen. This situation results in trajectory-points that are far away from each other: the generated initial simplex could so evolve to a local minimum of the cost function, which has the joint-angles values utterly different by the previous position. The randomness introduced by the Nelder-Mead restart procedure described in Section V hinders the formulation of an analytical expression predicting the appearance of spikes in the robotic arm movements. Instead, it can be evaluated empirically for a given combination of parameters.

It was already mentioned in section VIII-A that the use of ARTICo\(^3\) enables the use of DPR, as a mechanism to obtain adaptivity in the proposed IK solver. The reconfiguration time is an important characteristic to be evaluated in this context. Figure 20 reports the time necessary to upload the bitstreams on the FPGA. It can be noted that the time to perform the reconfiguration is non-identical for the eight slots. The reason is that the size of the reconfigurable regions (and thus the size of the partial bistream files) varies from one side of the FPGA to the other (see Figure 11), having two separate groups of 4 slots with similar reconfiguration times. This value is the time that the user must take into account when the number of accelerators working in parallel is modified, either for performance or dependability reasons.

FIGURE 20. Reconfiguration time per slot using the ARTICo\(^3\) architecture.

IN CONCLUSIONS AND FUTURE WORK

In this paper, we present a novel scheme for solving the IK problem at run time, targeting heterogeneous MPSoC devices. It relies on two-level algorithmic parallelism, combined with hardware-level parallelism. A variable number of dynamically reconfigurable hardware accelerators are used. They provide adaptation by trading-off among accuracy, resource occupancy, dependability, and execution time. Extensive experimental results show that our hypothesis is well suited in all the problems in which the robot end-effector path is discretized for accurate control of the arm movements.

The entire work additionally shows that academic tools and frameworks (in this case, PREESM and ARTICo\(^3\)), in combination with commercial tools (Simulink), can speed up the development of complex hardware/software systems thus reducing the time-to-market for new embedded systems. Moreover, the performed DSE shows, through the proposed Pareto, that a trade-off among performance, accuracy, energy-consumption, and dependability can be achieved by playing with the algorithm parameters and with the hardware resources at the same time.

Further effort will be provided in the future to increase the autonomy of the system by integrating a runtime manager in charge of acting in response to a stimulus and adapt itself to face new situations. This characteristic is commonly known as self-adaptation [65]: the capability to adjust its structure and behavior at run time and autonomously. In the proposed work, this feature is already enabled by the possibility of switching, dynamically, and at run time, between different operating modes. The self-adaptation itself requires a self-aware system that is capable of monitoring external events coming from the physical world as well as internal changes in the status of the hardware structure. The ARTICo\(^3\) is natively capable of overseeing its internal infrastructure thanks to the presence of ad-hoc performance monitoring counters interfaced with a standard hardware/software unified method [66]. The system will so guarantee not only fault-tolerance but also self-healing, making use of the Dynamic Partial Reconfiguration (DPR) for repairing eventually hardware damages caused by radiation [67].

ACKNOWLEDGMENT

Special thanks to Mr. T. Sebastián (formerly at Thales Alenia Space España) for his support on this paper.

REFERENCES

[1] H. Schreuder and R. Verheijen, “Robotic surgery,” BJOG, Int. J. Obstetrics Gynaecol., vol. 116, no. 2, pp. 198–213, 2009.
[2] M. Sugimoto, K. Tanaka, Y. Matsuoka, M. Man-i, Y. Morita, S. Tanaka, S. Fujiwara, and T. Azuma, “da Vinci robotic single-incision cholecystectomy and hepatectomy using single-channel GelPort access,” J. Hepato-Biliary-Pancreatic Sci., vol. 18, no. 4, p. 493, 2011.
[3] C. Heyer, “Human-robot interaction and future industrial robotics applications,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., Oct. 2010, pp. 4749–4754.
[4] J. Shi, G. Jimmerson, T. Pearson, and R. Menassa, “Levels of human and robot collaboration for automotive manufacturing,” in Proc. Workshop Perform. Metrics Intell. Syst. (PerMIS), 2012, pp. 95–100.
[5] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, “Trajectory planning in robotics,” Math. Comput. Sci., vol. 6, no. 3, pp. 269–279, 2012.
[6] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, Robotics: Modelling, Planning and Control. Cham, Switzerland: Springer, 2010.
[7] J. A. Nelder and R. Mead, “A simplex method for function minimization,” Comput. J., vol. 7, no. 4, pp. 308–313, Jan. 1965.
[8] W. Wolf, A. A. Jerraya, and G. Martin, “Multiprocessor system-on-chip (MPSoC) technology,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 27, no. 10, pp. 1701–1713, Oct. 2008.
[9] M. Pelcat, K. Desnos, J. Heulot, C. Guy, J.-F. Nezan, and S. Ardili, “Preesm: A dataflow-based rapid prototyping framework for simplifying multicore DSP programming,” in Proc. 6th Eur. Embedded Design Educ. Res. Conf. (EDERC), Sep. 2014, pp. 36–40.
[10] A. Rodríguez, J. Valverde, J. Portilla, A. Otero, T. Riesgo, and E. de la Torre, “FPGA-based high-performance embedded systems for adaptive edge computing in cyber-physical systems: The ARTICo\(^3\) framework,” Sensors, vol. 18, no. 6, p. 1877, Jun. 2018.
C. Park, J. Jung, and S. Ha, “Extended synchronous dataflow for efficient DSP system prototyping,” *Des. Autom. Embedded Syst.*, vol. 6, no. 3, pp. 295–322, 2002.

K. Desnos, M. Pelcat, J.-F. Nezan, S. S. Bhattacharyya, and S. Aridhi, “PMM: Parameterized and interfaced dataflow meta-model for MPSoCs runtime reconfiguration,” in *Proc. Int. Conf. Embedded Comput. Syst., Archit., Modeling, Simul. (SAMOS)*, Jul. 2013, pp. 41–48.

K. Desnos and J. Heulot, “PSDF: Parameterized & interfaced synchronous dataflow for MP-SoCs runtime reconfiguration,” in *Proc. 1st Workshop MEthods TOols Dataflow PrOgram. (METODO)*, Madrid, Spain: ECSI, Oct. 2014. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01075114/document

M. Pelcat, J. F. Nezan, J. Piat, J. Croizer, and S. Aridhi, “A system-level architecture model for rapid prototyping of heterogeneous multicore embedded systems,” in *Proc. Semantic Scholar*, 2009, pp. 1–9.

Xilinx Zynq UltraScale. MPSoC ZC702 Evaluation Kit. Accessed: Aug. 23, 2018. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/zc702-eval-board/.

JINA226, Datasheet, Texas Instrum., Dallas, TX, USA, 2011.

E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*. Cambridge, MA, USA: MIT Press, 2016.

L. Suriano, D. Madroñal, A. Rodríguez, E. Juárez, C. Sanz, and E. de la Torre, “A unified Hardware/Software monitoring method for reconfigurable computing architectures using PAPI,” in *Proc. 13th Int. Symp. Reconfigurable Commun.-Centric Systems-Ship (ReCoSoC)*, Jul. 2018, pp. 1–8.

A. Pérez, A. Rodríguez, A. Otero, D. G. Arjona, A. Jiménez-Peralo, M. A. Verdugo, and E. de la Torre, “Run-time reconfigurable MPSoC-based on-board processor for vision-based space navigation,” *IEEE Access*, vol. 8, pp. 59891–59905, 2020.