Feasibility Study and Porting of the Damped Least Square Algorithm on FPGA

Carlo Sau, Tiziana Fanni, Claudio Rubattu, Luca Fanni, Luigi Raffo, Francesca Palumbo

IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received August 3, 2020, accepted September 14, 2020, date of publication September 21, 2020, date of current version October 6, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3025367

Feasibility Study and Porting of the Damped Least Square Algorithm on FPGA

CARLO SAU[1], (Member, IEEE), TIZIANA FANNI[2], CLAUDIO RUBATTU[2,3], (Member, IEEE), LUCA FANNI[2], LUIGI RAFFO[1], (Member, IEEE), AND FRANCESCA PALUMBO[2], (Member, IEEE)

[1]Department of Electrical and Electronic Engineering (DIEE), University of Cagliari, 09123 Cagliari, Italy
[2]Department of Chemistry and Pharmacy, University of Sassari, 07100 Sassari, Italy
[3]IETR UMR CNRS 6164, INSA Rennes, University of Rennes 1, 35700 Rennes, France

Corresponding author: Carlo Sau (carlo.sau@unica.it)

This work was supported by the European Union's Horizon 2020 Research and Innovation Programme under Grant 732105. The work of Francesca Palumbo was supported by the University of Sassari through the Fondo di Ateneo per la Ricerca 2019.

ABSTRACT Modern embedded computing platforms used within Cyber-Physical Systems (CPS) are nowadays leveraging more and more often on heterogeneous computing substrates, such as newest Field Programmable Gate Array (FPGA) devices. Compared to general purpose platforms, which have a fixed datapath, FPGAs provide designers the possibility of customizing part of the computing infrastructure, to better shape the execution on the application needs/features, and offer high efficiency in terms of timing and power performance, while naturally featuring parallelism. In the context of FPGA-based CPSs, this article has a two fold mission. On the one hand, it presents an analysis of the Damped Least Square (DLS) algorithm for a perspective hardware implementation. On the other hand, it describes the implementation of a robotic arm controller based on the DLS to numerically solve Inverse Kinematics problems over a heterogeneous FPGA. Assessments involve a Trossen Robotics WidowX robotic arm controlled by a Digilent ZedBoard provided with a Xilinx Zynq FPGA that computes the Inverse Kinematic.

INDEX TERMS Cyber physical systems, damped least square, design automation, embedded systems, field programmable gate arrays, hardware, heterogeneous platforms, reconfigurable architectures, robotic arm controller, robot kinematics.

I. INTRODUCTION

Cyber-Physical Systems (CPSs) are complex platforms, composed of physical and computing parts deeply inter-wined, characterized by a strong interaction with environment and users [1]. This continuous interaction requires high flexibility and a certain degree of adaptivity, due to the numerous triggers CPSs are subjected to. The cyber part of a CPS often leverages on embedded computing platforms, which nowadays have become heterogeneous, integrating different types of computing units, e.g. different cores and coprocessors. Such kind of platforms is suitable to serve the needs of CPSs, being capable of acquiring large data-streams from a plethora of different sensors, and of processing them, when required, in parallel. Field-Programmable Gate Array (FPGA)-based platforms can play a crucial role in the described context: they have turned into heterogeneous architectures offering, at the same time, the flexibility and efficiency given by the integrated core(s), connected to custom hardware accelerators that can be reconfigured at run-time.

In general, the research activity in this context is flourishing [2]–[8]. In the H2020 CERBERO[1] European Project [8] we worked on providing a continuous design environment for CPSs that, at computational level, relies on many tools for FPGA-based CPSs, which have been accessed during the project time-frame on a *Planetary Exploration* use-case [9]. An adaptive embedded controller for a robotic manipulator has been implemented over an FPGA. Such a controller was supposed to work in stringent survival conditions (radiation and harsh environment), being able to meet the reliability constraints of a robotic exploration mission, as well as functional constraints such as a good accuracy over the requested tasks. Performance and accuracy are key requirements in robotic applications, not only in the space exploration field. For instance, also surgery [10] and industrial [11] application fields require an accurate trajectory control during their

The associate editor coordinating the review of this manuscript and approving it for publication was Vincenzo Conti.

[1]http://www.cerbero-h2020.eu/

tasks execution, even in the proximity of singularity points. At the same time, if online trajectory calculation is enabled, the time requested to evaluate it may be critical in achieving responsiveness.

Most of the commonly used methods for solving the faced problem, namely Forward Kinematic (FK) or Inverse Kinematic (IK), split the trajectory in smaller paths, each calculated singularly, and the higher is the number of smaller paths used to split the original trajectory, the more accurate will be the overall robotic arm movement. As a drawback, if the number of smaller paths grows the amount of calculations increases in turn. So that, to meet such colliding requirements different solutions are adopted, ranging from given algorithm modifications to the usage of different algorithms for different places within the workspace. Besides algorithm-oriented approaches, the explosion of computational complexity could be addressed by relying on modern computing platforms, which are suitable to support parallel computation and offer heterogeneity. Parallelization gives the possibility of executing different operations of the algorithms concurrently, while heterogeneity provides differentiation in terms of computing cores, shaping them according to the functionality they are dedicated to. Nevertheless, parallel and heterogeneous platforms have a cost in terms of design effort, since they are usually hard to program and control in an efficient way. Moreover, most of robotics algorithms are not easy to be parallelized, e.g. due to the dependencies between successive smaller paths calculations, and exploiting heterogeneity by shaping computing cores or by optimizing the execution on the different available ones, according to the involved operations, is not trivial.

This article focuses on the analysis of the Damped Least Squares (DLS) algorithm for IK solution, looking to a prospective hardware implementation on FPGA of a DLS-based hardware controller for a robotic arm. More in details, the contributions of this article are:

- the open-source sequential and parallel MATLAB implementations of the DLS [12]. These implementations have been released as open data of the CERBERO project together with a trajectory generator [13] for the workspace described in Section II.
- the analysis of the DLS algorithm, with the specific intent of understanding which parameters could influence a perspective hardware implementation. This analysis is reported in Section V where two different implementations, a traditional sequential one and a parallel one based on segmentation, are discussed and analyzed. To the best of our knowledge no attempt to parallelize the DLS has been done prior to this work.
- a preliminary FPGA-based implementation of the robotic arm controller capable of executing different DLS profiles (baseline and high-performance). As discussed in Section VI, an automated High-Level Synthesis (HLS)-based strategy has been used to reduce the designers effort needed to master the target heterogeneous platform. To the best of our knowledge, despite our implementation is just preliminary and the assessment is covering just the reconfigurable part of the FPGA-based arm controller, the present work is the first one porting DLS for IK problems solving on such heterogeneous substrate.

This article is organized as follows. In Section II, the context of the target robotic manipulator is presented. State-of-the-art IK solutions with a focus on the DLS algorithm are discussed in Section III. In Section IV, state-of-the-art works on hardware implementations of the algorithms solving the IK problem are proposed. In Section V, a detailed analysis of pros and cons in the usage of the DLS algorithm in the context of the CERBERO project is presented. Section VI is dedicated to the design flow and assessment of the proposed hardware implementation of the DLS algorithm. Section VII concludes the paper.

II. REFERENCE PROBLEM

This section provides an overview of the physical context of the proposed work. The target manipulator, the workspace it operates into, and the tasks considered for the execution within such workspace are described in the subsections below.

A. TARGET MANIPULATOR

Any robotic manipulator is composed of different parts, namely: 1) a base; 2) rigid links; 3) joints (each of them connecting two adjacent links); and 4) an end-effector. Figure 1 shows as an angle, θ_i, can be associated to each rotational joint.

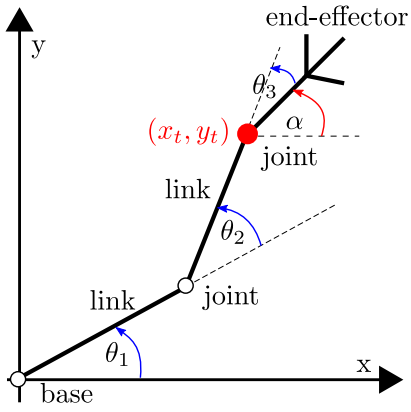

FIGURE 1. Robotic manipulator example.

Describing the movement of the manipulator in the space implies to define the trajectory of the end-effector from its initial given spatial coordinates, or from the origin $A(x_0, y_0, z_0)$, to the final ones, or to the destination $Z(x_1, y_1, z_1)$. It is possible to describe such a trajectory by:

- computing the spatial coordinates of the end-effector from all the joint angles, resolving a *FK* problem.
- computing the joint angles from a desired end-effector spatial coordinate, resolving an *IK* problem.

To implement an arm controller, the second type of problem has to be solved to derive the angles of the joints starting from a given desired position (or set of positions).

The characteristics/features of the manipulator are generally vendor specific. The target we have considered is a WidowX Robotic Arm by Trossen Robotics [14], characterized by four main Degrees of Freedom (DoFs) for wrist and clamp movements. This manipulator is equipped with six Dynamixel actuators, controllable in terms of angular position, speed and acceleration, and each of them provides a full control of the relative joint via a digital communication using a Universal Asynchronous Receiver-Transmitter (UART) protocol. The manipulator comes with an integrated Arduino-based ArbotiX-M controller, which can be used to control it directly, even though one of the goals of this article is to prove that it is possible to substitute the built-in controller with an FPGA based one.

B. WORKSPACE, TASKS AND REQUIREMENTS

The reachable coordinates depend on the chosen robotic arm and on its fabrication parameters that define the operating workspace, which has to include the set of points reachable by the end-effector, meant as both the initial and final spatial coordinates, as well as the positions along the trajectory. The workspace for a six-axis robotic arm is a six-dimensional space that consists of all possible combinations of values for the six degrees of freedom of the arm, e.g. the complete set of positions and orientations of the robot.

In this workspace, there might be also singularity points that are positions where the IK solver fails to find a convergent solution. Main ones are wrist, elbow and shoulder singularities. Passing close to a singularity results in high joint velocities, which are normally to be avoided since they might damage the arm itself. A properly built workspace could limit, or even prevent, the risks related to singularities.

In this work, we have adopted a singularity-free workspace defined by the Thales Alenia Space company for the chosen type of manipulator and made available as an open-data [15] of the CERBERO project [8]. The workspace is limited to a safe area of operation specified in cylindrical coordinates (see Equations 1, 2 and 3). In this region, given the equations and assumptions specified in [15], the end-effector can describe any type of trajectory without encountering singularities. Moreover, moving from $A(x_0, y_0, z_0)$ to $Z(x_1, y_1, z_1)$ it is needed to guarantee that $sgn(x_0) = sgn(x_1)$ or $sgn(y_0) = sgn(y_1)$, so that not direct displacement is produced from one quadrant of the XY plane to the opposite one.

$$1 + \frac{2}{5}\rho < z < 29 - \frac{2}{5}\rho \tag{1}$$

$$10 < \rho < 35 \tag{2}$$

$$\rho^2 = x^2 + y^2 \tag{3}$$

According to the described workspace, we have derived, and made available as an open-data [13] of the CERBERO project, a *trajectory generator*. The trajectory generator is a MATLAB script that generates 100 possible trajectories for the WidowX Robotic Arm. For each of them the initial and final position of the end-effector are defined as spatial coordinates within the workspace. By default, the initial point is fixed to $A(x_0, y_0, z_0) = \{35, 0, 15\}$ that is the idle resting position of the arm. Along with the script itself, a *trajectories.csv* file is also provided. It contains the set of trajectories that are used for the algorithm explorations reported in this article in Section V. They can be classified in three groups, *short*, *medium*, and *long*, on the basis of their length. Please note that, given the reference workspace, the minimum length of a trajectory is around 2 cm and the maximum is around 65 cm. The identified groups are composed as follow:

1) *small* - trajectories ID: [1 to 35]; range of length: [2.11 cm to 22.75 cm]
2) *medium* - trajectories ID: [36 to 73]; range of length: [23.05 cm to 39.76 cm]
3) *long* - trajectories ID: [74 to 100]; range of length: [41.23 cm to 64.79 cm]

The definition of the workspace has to be accompanied by other requirements to specify the reference problem. As it will become evident in Section V, the tasks that the robotic arm has to execute may indeed affect programmers' decisions when choosing/tuning the IK solver. In this article, we consider that the chosen algorithm should be capable of executing two different tasks, presenting different requirements and characteristics:

- Task 1: Retrieval of rock samples from planetary surface of Mars
  - Approximate distance of trajectories: 50 cm
  - Required positioning accuracy: 10 mm
- Task 2: Manipulation of biocontainment
  - Approximate distance of trajectories: 10 cm
  - Required positioning accuracy: 5 mm

These tasks are the same that have been used to demonstrate CERBERO technologies within the Planetary Exploration use case and have been provided by Thales Alenia Space [15].

III. SOLUTIONS TO THE INVERSE KINEMATIC PROBLEM

Solving an IK problem means computing the joint angles of a body starting from a desired end-effector spatial coordinate. This is a non-linear problem that requires to solve complex trigonometric functions. Indeed, an IK problem can have multiple solutions, only one or even none at all. In particular, the number of solutions depends on the desired end-effector coordinates and the number of DoFs. For instance, a point could result to be non-reachable and thus the problem would be not solvable. On the contrary, two different sets of angles may represent a solution when considering the elbow-up or elbow-down poses of an arm, but possibly only one of them could be physically possible for a certain considered arm. Thus, before solving an IK problem, it is necessary to take in consideration any joint angles limitation and singularities-related problem, and which are the out-of-reach end-effector spatial coordinates. In this regard, it is crucial to properly define the workspace, identifying the out-of-reach trajectories to be avoided by constructions. Similarly,

singulatity matters could be avoided by excluding specific points from the workspace, as done by Thales Alenia Space in [15].

The IK problem has been deeply addressed in literature, mainly in the robotic and in the computer graphic fields. The computer graphic field is the one that offers more flexibility in studying the solvers for IK problems, and is the one considered by Aristidou *et al.* [16] in their classification of the possible solutions for IK. In particular, they consider four main categories of solvers, that present different behaviors:

1) *Analytic solutions*: considering the length of the arm, the initial position and the physical constraints, the analytic methods are used for finding all the possible solutions to the problem. They guarantee fast computation and the best solution. They generally do not suffer from the singularity problem and are highly reliable. However, they are suitable only for mechanisms with low DoFs. The problem is not easily scalable, the bigger is the chain of links and the more are the DoFs, the more difficult is the solution of the IK problem adopting an analytic solver.
2) *Numerical solutions*: based on a cost function to be minimized, this family includes the methods that require various iterations to converge over a solution. In general, solutions belonging to this family are better capable of mastering more DoFs and multiple end-effectors (e.g. fingers of a hand or arms of a body). Many algorithms, based on numerical solutions, exist: *Heuristic* methods and *Cyclic Coordinate Descendant* ones [17], *Newton-based* methods exploiting the second-order Taylor expansion, and *Jacobian-based* ones adopting inversed, pseudo-inversed and transposed Jacobian matrices [18].
3) *Data-Driven methods*: learning-based techniques are the foundation for these methods that require having available large datasets and pre-learned postures to match the position of the end-effector. These characteristics make them suitable for complex problems such as human-like structures.
4) *Hybrid methods*: they are based on the simplification of the IK problem, by decomposing it in a combination of analytical and numerical components and exploiting the related algorithms for the solution. As data-driven methods, the hybrid methods are suitable for complex problems, such human-like structures or multiple end-effectors ones.

Due to the peculiarities of the numerical solutions, that are highly suitable for the problem we address in this article, we focus our attention on this family. Indeed, given the chosen target (see Section II-A), where multiple end-effectors or body-like mechanisms are not there, complex data-driven or hybrid methods can be excluded. Having available six DoFs (four for the hand effector positioning, one for wrist and one for clamp), numerical methods seems to be the most straightforward choice. In our studies, we opted for the DLS IK solver, which falls into the Jacobian-based family, due to computational complexity considerations as explained below in Sections III-A and III-B.

A. JACOBIAN-BASED METHODS

Jacobian methods are based on the Jacobian matrix [19], defined as:

$$f : \Omega \subseteq \mathbb{R}^n \rightarrow \mathbb{R}^m$$

$$J_f(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x) & \frac{\partial f_1}{\partial x_2}(x) & \dots & \frac{\partial f_1}{\partial x_n}(x) \\ \frac{\partial f_2}{\partial x_1}(x) & \frac{\partial f_2}{\partial x_2}(x) & \dots & \frac{\partial f_2}{\partial x_n}(x) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1}(x) & \frac{\partial f_m}{\partial x_2}(x) & \dots & \frac{\partial f_m}{\partial x_n}(x) \end{bmatrix} \in \mathbb{R}^{m,n}$$

Such a matrix represents a transformation between two time derivative-related spaces, the Cartesian space and the velocity space. The latter can be related to the velocity joints space. Solving an IK problem requires finding the joint angles of the body from the desired spatial coordinates. Therefore we need to compute the joint space vector, $\Delta\vec{\Theta}$, as:

$$\Delta\vec{\Theta} = J^{\dagger}\Delta\vec{e} \tag{4}$$

where $\Delta\vec{e}$ is the error or displacement vector and $J^{\dagger}$ is computed from the Jacobian matrix, J, accordingly with the chosen Jacobian-based solver.

The $\Delta\vec{\Theta}$ and $\Delta\vec{e}$ vectors are expressed as:

$$\Delta\vec{\Theta} = [\Delta\theta_1 \quad \Delta\theta_2 \quad \dots \quad \Delta\theta_n]^T \tag{5}$$

and

$$\Delta\vec{e} = \begin{bmatrix} \Delta x & \Delta y & \Delta z & \delta x & \delta y & \delta z \end{bmatrix}^T \tag{6}$$

where $[\Delta x \ \ \Delta y \ \ \Delta z]^T$ defines a linear displacement, while $[\delta x \ \ \delta y \ \ \delta z]^T$ a rotational one.

Figure 2 illustrates the general flow of Jacobian-based IK algorithms, no matter of the chosen specific solver. The trajectory itself is specified by initial and final end-effector coordinates. The former are computed solving a FK problem, which requires to know the angles assumed by the manipulator in the initial position; while the latter correspond to

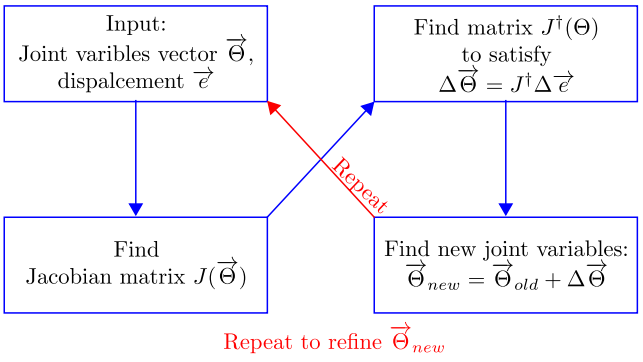

FIGURE 2. Jacobian-based IK algorithms flow diagram.

the desired position to be reached. The way of computing $J^\dagger$ depends on the chosen Jacobian-based IK solver. In any case, an iterative process to compute the new joint variables is required.

Prior to execute the algorithm, the displacement vector must be defined according to Equation 6. The following simplification hypothesis $[\delta x \ \ \delta y \ \ \delta z]^T = [0 \ \ 0 \ \ 0]^T$ has been taken, meaning that we will consider linear displacements only. Therefore, trajectories are computed just on the basis of the values assigned to $[\Delta x \ \ \Delta y \ \ \Delta z]^T$, and the displacement vector is expressed as:

$$\Delta\overrightarrow{e} = \left[\frac{x_{fin} - x_{init}}{i+1} \quad \frac{y_{fin} - y_{init}}{i+1} \quad \frac{z_{fin} - z_{init}}{i+1} \quad 0 \quad 0 \quad 0 \right]^T$$

where x_{init}, y_{init}, and z_{init} represent the initial point coordinates, x_{fin}, y_{fin}, and z_{fin} the coordinates of the final position that the end-effector must reach, and i is the iteration parameter.

In our setup, the joint space is composed only by angles, since we are using rotational-only joints. The process to obtain integral angles from the joint space, is iterative:

$$\overrightarrow{\Theta}_{new} = \overrightarrow{\Theta}_{old} + \Delta\overrightarrow{\Theta} \tag{7}$$

The number of repeated steps can be set by an *iteration* parameter of the algorithm. At each iteration, the last calculated $\overrightarrow{\Theta}_{new}$ is used as $\overrightarrow{\Theta}_{old}$ to compute the next $\overrightarrow{\Theta}_{new}$. This procedure is equivalent to the calculation of the defined integral of thetas along the trajectory.

The first mathematical solution for Equation 4 is computing $J^\dagger$ as J^{-1} [18]. However, J might be neither square nor invertible. Therefore, several alternative solutions have been studied to solve this problem. The Jacobian transpose [18] computes $J^\dagger$ using the transpose of J as $J^\dagger = \alpha J^T$, while the Pseudo-inverse methods [18] leverage on $J^\dagger = J^T(JJ^T)^{-1}$. The computation of $J^\dagger$, given by the transpose-based solutions, presents low complexity, but transpose-based solutions fail in handling singularities [20], and in many cases null space methods have been adopted to avoid singular configurations [21], [22]. The Single Value Decomposition (SVD) method [23] has been proposed to deal with the singularities. It generalizes the eigen-decomposition of a square normal matrix, decomposing the matrix J as $J = UDV^T$, where U is an $m \times m$ unitary orthogonal matrix, D is an $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal, and V^T is an $n \times n$ unitary orthogonal matrix. This solutions is based on the calculation of the eigenvalues and eigenvectors, at high computational costs. The Damped Least Square (DLS) algorithm [24], computes $J^\dagger$ as $J^\dagger = (J^TJ + \lambda^2 I)^{-1}J^T$. This algorithm handles the singularity problem by using a damping factor λ, and does not require the computation of eigenvalues and eigenvectors. Therefore, in our studies we opted for this latter to have the possibility of porting the designed hardware controller in different workspaces, not necessarily singularity-free ones, and to maintain the computational load of the robotic arm controller to be implemented as low as possible.

B. THE DAMPED LEAST SQUARE ALGORITHM: MATHEMATICAL FORMULATION

The DLS algorithm is also known as *Levenberg—Marquardt* algorithm [24], and computes $J^\dagger$ as:

$$J^\dagger = \left(J^TJ + \lambda^2 I\right)^{-1} J^T \tag{8}$$

so that, using Equation (8), the Equation (4) can be expressed as:

$$\Delta\overrightarrow{\Theta} = \left(J^TJ + \lambda^2 I\right)^{-1} J^T \Delta\overrightarrow{e} \tag{9}$$

The damping factor, λ, is a non-zero constant that needs to be large enough to make the solution behave well near singularities, but not too large to make the convergence rate excessively slow. This factor can be set as a static parameter, which has to be defined a-priori before starting the computation, or as a dynamic one to be calculated at run-time, between two subsequent iterations. In our hardware implementation, as discussed in Section V-E, a static λ is used.

To obtain the Jacobian J matrix, prior to the application of the DLS, the mathematical model of the chosen manipulator has to be derived. Commonly, the *Denavit-Hartenberg* (DH) parameters [25] are used. These are a set of four parameters composed of two angles (α and θ) and two displacements (a and d), defined for each link-joint group and obtained from the manipulator mechanical dimensions and joint orientations, as shown in Figure 3. Values related to the chosen set-up/manipulator are reported in Table 1.

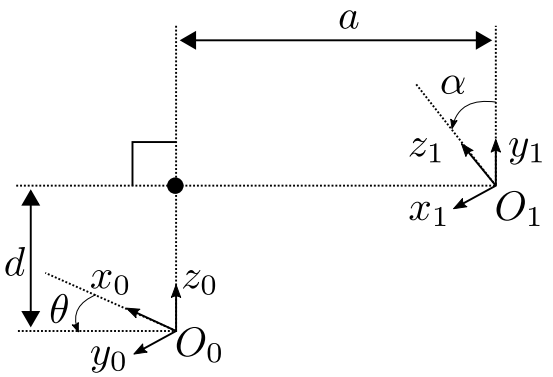

FIGURE 3. Graphical representation of Denavit-Hartenberg parameters.

TABLE 1. Denavit-Hartenberg parameters of the chosen manipulator.

J_i	a_i (cm)	d_i (cm)	α_i (rad)	θ_i (rad)
1	0	0	$\frac{\pi}{2}$	θ_1
2	15	0	$-\pi$	θ_2
3a	5	0	0	$\frac{\pi}{2}$
3	15	0	0	θ_3
4	15	0	0	θ_4

For each set of parameters a transformation matrix is defined as:

$$ {}^{n-1}T_n = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ t_{41} & t_{42} & t_{43} & t_{44} \end{bmatrix} \tag{10} $$

where

$$\begin{aligned} t_{11} &= \cos\theta_n \\ t_{12} &= -\sin\theta_n \cos\alpha_n \\ t_{13} &= \sin\theta_n \sin\alpha_n \\ t_{14} &= a_n \cos\theta_n \\ t_{21} &= \sin\theta_n \\ t_{22} &= \cos\theta_n \cos\alpha_n \\ t_{23} &= -\cos\theta_n \sin\alpha_n \\ t_{24} &= a_n \sin\theta_n \\ t_{32} &= \sin\alpha_n \\ t_{33} &= \cos\alpha_n \\ t_{34} &= d_n \\ t_{31} &= t_{41} = t_{42} = t_{43} = 0 \\ t_{44} &= 1 \end{aligned}$$

This transformation matrix represents the position and the orientation of the n^{th}-joint with respect to the previous one. For an N-joints manipulator there will be N transformation matrices. The general transformation matrix is defined as the chain product of each transformation matrix:

$$ T = {}^0T_1 {}^1T_2 \ldots {}^{N-1}T_N = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11} $$

This matrix represents the end-effector spatial coordinates and orientation with respect to the manipulator base coordinates. The derived DH parameters, used in Equation (10), allow to determine the general transformation matrix orientation coefficients (n_x, n_y, n_z, o_x, o_y, o_z, a_x, a_y, and a_z) and the position coefficients (p_x, p_y, and p_z).

$$\begin{aligned} n_x &= -\sin(\theta_3 - \theta_2 + \theta_4)\cos\theta_1 \\ o_x &= -\cos(\theta_3 - \theta_2 + \theta_4)\cos\theta_1 \\ a_x &= -\sin\theta_1 \\ n_y &= -\sin(\theta_3 - \theta_2 + \theta_4)\sin\theta_1 \\ o_y &= -\cos(\theta_3 - \theta_2 + \theta_4)\sin\theta_1 \\ a_y &= \cos\theta_1 \\ n_z &= -\cos(\theta_3 - \theta_2 + \theta_4) \\ o_z &= \sin(\theta_3 - \theta_2 + \theta_4) \\ a_z &= 0 \end{aligned}$$

p_x, p_y and p_z are the end-effector spatial coordinates with respect to joint angles, defined throughout the manipulator FK over the final desired position.

$$\begin{aligned} p_x &= 5\cos\theta_1\,[3\sin(\theta_2 - \theta_3) - 3\sin(\theta_3 - \theta_2 + \theta_4) \\ &\quad + 3\cos\theta_2 + \sin\theta_2] \\ p_y &= 5\sin\theta_1\,[3\sin(\theta_2 - \theta_3) - 3\sin(\theta_3 - \theta_2 + \theta_4) \\ &\quad + 3\cos\theta_2 + \sin\theta_2] \\ p_z &= -15\cos(\theta_2 - \theta_3) - 15\cos(\theta_3 - \theta_2 + \theta_4) \\ &\quad - 5\sqrt{10}\cos(\theta_2 + \arctan 3) \end{aligned}$$

Once n_x, n_y, n_z, o_x, o_y, o_z, a_x, a_y, and a_z, and p_x, p_y, and p_z are known, according to the theory in [19], Equation (11) can be used to derive J as follow:

$$ J = \begin{bmatrix} j_{11} & j_{12} & j_{13} & j_{14} \\ j_{21} & j_{22} & j_{23} & j_{24} \\ j_{31} & j_{32} & j_{33} & j_{34} \\ j_{41} & j_{42} & j_{43} & j_{44} \\ j_{51} & j_{52} & j_{53} & j_{54} \\ j_{61} & j_{62} & j_{63} & j_{64} \end{bmatrix} \tag{12} $$

where

$$\begin{aligned} j_{11} &= -5\sin\theta_1[3\sin(\theta_2 - \theta_3) - 3\sin(\theta_3 - \theta_2 + \theta_4) \\ &\quad + 3\cos\theta_2 + \sin\theta_2] \\ j_{12} &= 5\cos\theta_1[3\cos(\theta_2 - \theta_3) + 3\cos(\theta_3 - \theta_2 + \theta_4) \\ &\quad + \cos\theta_2 - 3\sin\theta_2] \\ j_{13} &= -5\cos\theta_1[3\cos(\theta_2 - \theta_3) + 3\cos(\theta_3 - \theta_2 + \theta_4)] \\ j_{14} &= -15\cos(\theta_3 - \theta_2 + \theta_4)\cos\theta_1 \\ j_{21} &= 5\cos\theta_1[3\sin(\theta_2 - \theta_3) - 3\sin(\theta_3 - \theta_2 + \theta_4) \\ &\quad + 3\cos\theta_2 + \sin\theta_2] \\ j_{22} &= 5\sin\theta_1[3\cos(\theta_2 - \theta_3) + 3\cos(\theta_3 - \theta_2 + \theta_4) \\ &\quad + \cos\theta_2 - 3\sin\theta_2] \\ j_{23} &= -5\sin\theta_1[3\cos(\theta_2 - \theta_3) + 3\cos(\theta_3 - \theta_2 + \theta_4)] \\ j_{24} &= -15\cos(\theta_3 - \theta_2 + \theta_4)\sin\theta_1 \\ j_{31} &= j_{41} = j_{51} = j_{62} = j_{63} = j_{64} = 0 \\ j_{32} &= 15\sin(\theta_2 - \theta_3) - 15\sin(\theta_3 - \theta_2 + \theta_4) + \\ &\quad + 5\sqrt{10}\sin(\theta_2 + \arctan 3) \\ j_{33} &= 15\sin(\theta_3 - \theta_2 + \theta_4) - 15\sin(\theta_2 - \theta_3) \\ j_{34} &= 15\sin(\theta_3 - \theta_2 + \theta_4) \\ j_{42} &= \sin\theta_1 \\ j_{43} &= j_{44} = -\sin\theta_1 \\ j_{52} &= -\cos\theta_1 \\ j_{53} &= j_{54} = \cos\theta_1 \\ j_{61} &= 1 \end{aligned}$$

C. THE DAMPED LEAST SQUARE ALGORITHM: SUMMARY OF THE FEATURES AND CHARACTERISTICS

As already said, the DLS presents a singularities tolerance, which is a crucial desirable characteristic when the workspace is not singularity-free. The DLS is capable of passing through workspace singularities: the algorithm allows tuning the solution around singularities by means of the damping factor,

resulting in a smoothed movement. Moreover, it does not require to compute eigenvalues or eigenvectors, leading to a reduced computational complexity more suitable for hardware implementation. Furthermore, the DLS algorithm can achieve a good trade-off between computation time and accuracy even if complex application scenarios, where full body reconstruction are considered [26].

Generally speaking, computation time and accuracy strongly depend on two main parameters of the DLS: the number of iterations, which are the points along the trajectory from source to destination, and the damping factor λ, which define the behavior near the singularities. In particular, fixing the number of iterations per trajectory it is possible to predict computation time and accuracy of the DLS execution. This time predictability is a distinctive feature of the DLS that can drive the choice towards this IK solver under certain constrained application scenarios, such as in critical applications where the algorithm convergence has to be guaranteed within a hard deadline in time.

IV. RELATED WORKS

The literature available on the IK solvers is generally mainly focused on algorithms optimization when calculated offline. Not many state-of-the-art works address the implementations of IK solvers leveraging on hardware acceleration platforms.

With the advent of CPSs and Internet of Things (IoT), devices are every day more power limited, while being requested to be highly computational efficient. Dedicated hardware allows for the efficient execution of complex algorithms even in constrained contexts, where finding trade-offs among system aspects of interest, e.g. power consumption versus quality of service, may be fundamental [27]. Profiling the algorithms to find the sub-parts that can be accelerated is surely necessary, as well as the identification of parallelism, since parallel off-the-shelf or dedicated devices could improve both timing and power performance. In particular, while off-the-shelf devices force designers to shape the parallel execution over the available concurrent resources, dedicated platforms can be customized to support the native parallelism of the algorithm. Dedicated hardware also enables pipelining of resources, which can be seen as a further hardware-level degree of parallelism, additional to the algorithm-level one.

The two subsections below provide, first of all, an overview of the state-of-the-art regarding the available hardware implementations of IK solvers and, then, a closer look into what has been done specifically on the DLS is provided.

A. HARDWARE IMPLEMENTATIONS OF INVERSE KINEMATIC SOLVERS

The hardware acceleration of IK solvers has been only partially explored by the scientific community. Moreover, while pipelining is often applied, algorithm-level parallelization is not. Analytical IK solvers have been accelerated with FPGAs, mainly addressing industrial robotics applications [2], [28]. Other works proposed the adoption of algorithms not commonly used for solving IK problems, but intrinsically parallel. A behavior-based solver has been adopted by Köpper and Berns [6]. Such solver has an intrinsically parallel nature; therefore, it turns out to be suitable for an FPGA implementation: calculation of the different DoFs can be parallelized, resulting in a substantial speed-up with respect to general purpose systems, even if the accelerator has not been optimized yet. Hildebrand *et al.* [29] instead adopted a conformal geometrical algebra approach to solve the IK problem addressing computer graphics application with FPGA support. Both pipelining and algorithm-level parallelization methods are exploited on the reconfigurable logic of the FPGA, resulting in extremely shorter execution time with respect to a full software processing. Other studies adopted common IK solvers, where algorithm level parallelism is more difficult to be extracted. Yu *et al.* [30] considered the generalized inverse Jacobian method adopted for FPGA acceleration by means the parallel calculation of matrix multiplications. Suriano *et al.* [31] instead adopted the iterative Nelder-Mead optimization method to solve the IK problem, investing effort in parallelizing the application in different ways (intra- and inter-iteration). The resulting parallel solutions are then ported on an FPGA fabric tightly coupled with a multi-core system: the application is split between general purpose and dedicated computing units, which are provided in several instances enabling multi-parallelism on the hardware acceleration side.

Table 2 summarizes the described IK solvers hardware implementations together with the approach proposed in this work. The analyzed state-of-the-art works adopt algorithms that are not suitable for the manipulator that is used in this work [2], [28], or algorithms that are meant for application fields far from the one considered in this article [29], or even algorithms that might not converge [31]. As already said, we adopted the DLS that has a predictable convergence time for a given trajectory length, and it is capable of tolerating singularities in the workspace. As will be detailed in Section VI, the proposed robotic arm controller will be implemented in hardware on a Xilinx Zynq FPGA platform. In this study we propose also a possible parallelization of the chosen IK solver, following a segmentation-based approach.

B. DAMPED LEAST SQUARE FOR INVERSE KINEMATICS

To the best of our knowledge, considering IK problems there are neither works in literature that address the hardware implementation of the DLS, nor works that analyze the DLS characteristics for a prospective hardware implementation. Works available in literature rarely use a pure DLS approach, meaning a purely numerical solution according to Aristodou *et al.* [16] classification. They are mainly preliminary studies combining also other behaviors, such as a system for robotic thumbs where both kinematics (DLS for movements) and dynamics (force model for grasping) [32], or full body reconstruction [26].

Other works focus on DLS drawbacks: being an iterative method, where each step (iteration) depends on the previous

TABLE 2. Hardware Implementation and Parallelization of Inverse Kinematics Solvers. [The DoFs number below refers to those of the chosen target in the different works; in our case, despite the manipulator has 6 available DoFs we are considering just 4 of them (see Section II-A), leaving out the wrist and clamp related ones].

Work	Solver Class [16]	Solver Algorithm	DoFs	Application Field	Applied Parallelization	Target Board
[2]	analytic	-	N/A	industrial robotics	hardware	Cyclone IV
[28]	analytic	-	4	industrial robotics	hardware	Cyclone IV
[6]	data-driven	behavior-based control	6	industrial robotics	algorithm/hardware	Cyclone IV
[29]	analytic	conformal geometrical algebra	N/A	computer animation	algorithm/hardware	Virtex 2
[30]	numerical	generalized inverse Jacobian	7	space missions	algorithm/hardware	Virtex 5
[31]	numerical	Nelder-Mead	4	space missions	algorithm/hardware	Zynq Ultrascale+
THIS WORK	numerical	DLS	4	space missions	algorithm/hardware	Zynq

TABLE 3. Damping least square for inverse kinematics overview.

Work	Associated Algorithm	Customizable Parameters	DoFs	Parallelization	Target
[26]	no	none	4/6	no	N/A
[32]	no	none	4	no	SW
[33]	numerical	error	7	no	SW
[34], [35]	machine learning	step	6/7	no	SW
[10]	machine learning	lambda	8	no	SW
[36]	genetic	lambda	6	no	N/A
[37]	analytical	lambda	6	no	SW
THIS WORK	no	step,lambda	4	yes	HW

one, the error is cumulative and it may result in a considerable distance between the actual and the desired end-effector position, especially when several DoFs are involved. To reduce the error, the number of iterations (see Section V-B) can be increased at the expenses of the computation time, which become longer. Another option to reduce the error is to couple DLS with other methods, such as the Newton-Raphson one that limits the cumulative error of DLS when it becomes too large, improving accuracy without increasing the DLS iterations [33].

Most of DLS implementations, in combination with other methods, play on the algorithm knobs: number of iterations and damping factor. Such a combination mainly speeds-up computation and achieves accuracy improvements. Wang *et al.* [34], [35] focused on the number of iterations, adjusting it by means of supervised learning techniques, considering different DoFs and different target platforms for implementation, among which an embedded multi-core system.

Other works focused more on the damping factor, which also impacts on accuracy and computation time of the DLS algorithm. In particular, addressing snake-like robots employed in micro-surgery applications, where hard accuracy and time requirements are present, Omisore *et al.* [10] adopted a deep-learning approach to optimally adjust the damping factor for every target point of the DLS, obtaining speed-up and accuracy improvements. Damping factor optimization is also desired for enhancing behavior of the DLS algorithm when passing close to singularities. Some works also aimed at such goal, by optimizing it by means of genetic algorithms [36] or closed-loops implementations, where the singularity problem and the Jacobian matrix are taken into consideration for damping factor adjustment [37].

Table 3 summarizes the works adopting DLS for IK solving and their main features and characteristics. Most of these works focus on the algorithm optimization under certain metrics, as computing time and accuracy, by coupling DLS with other algorithms or by playing with its parameters. None of those works focus on implementation details of the DLS, such as parallelization or optimization for execution on specific devices. Considering the addressed application fields, implementation aspects should be considered in future due to the stringent constraints and autonomous behavior possibilities, which are already envisioned for some of them, like for space and micro-surgery purposes. Based on this analysis, to the best of our knowledge, the proposed work is the first attempt to study parallelization and hardware acceleration of DLS algorithm for IK solving.

V. DAMPED LEAST SQUARES ALGORITHM: ANALYSIS

The goal of this analysis is to verify features, pros and cons of the DLS algorithm, when parameters vary. In particular we analyze two different MATLAB implementations that we made available as an open-data [12] of the CERBERO project. We will refer to these implementation as:

- *classic* - intended as fully sequential, where prior to start a new iteration of the algorithm the current computation of Equation 7 should be completed. In this case, programmers can play with two parameters: the damping factor and the number of iterations.
- *parallel* - This implementation is composed of two steps. In the first one the DLS is used to obtain coarse points over the whole trajectory; the outcome is a segmentation of the whole trajectory. In the second step, the DLS is run again over each segment, using as starting and final points the sequence of adjacent points calculated in the first step. In this case, programmers can play with

three parameters: the damping factor, the number of iterations in the first DLS step and the number of iterations in the second DLS step.

Figure 4 illustrates an example of the operation of the two implementations.

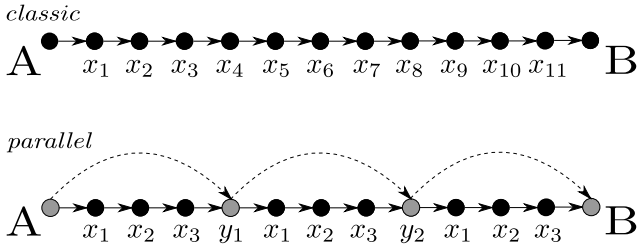

FIGURE 4. DLS: *classic* versus *parallel* computation.

The *parallel* implementation has been considered to understand the potential benefits of implementing the DLS over a parallel computing infrastructure, as the one used in [31]. Clearly, due to physical hardware limitations, we cannot assume an infinite parallel factor. As an example, dividing a trajectory into 50 different segments would require the availability of as many parallel computing units, which is highly unlikely in practice. For this reason, in the experiments below, we have considered up to 4 parallel computing units (that could be prospectively different cores or dedicated co-processors) to estimate the execution time.

Our explorations focus mainly on the error reaching the desired position and the elapsed execution time needed to compute the trajectories in [13]. All the raw data are available in [12]. Time measurements are obtained by using the "tic/toc" MATLAB functions, thus they are machine-dependent.[2]

In the *classic* case the provided estimation time is straightforward, while in the *parallel* one the estimation is given by summing up the time needed to compute step 1 (t_{s1}), plus the time needed to compute step 2 (t_{s2}) weighted with respect to the available parallel computing units. Equation 13 provides an example of the formula used to estimate the elapsed execution time needed to compute N different parallel segments over k parallel computing units.

$$time_{par} = t_{s1} + \frac{N}{k} * t_{s2} \tag{13}$$

Regarding the error, considering $Z(x_1, y_1, z_1)$ as the final desired position of the end-effector, it is possible to define the error among the desired target and the actual reached position as computed by the chosen implementation. Equation 14 is representative of the error with respect to Z when using the *classic* implementation of the DLS, while Equation 15 when using the *parallel* implementation of the DLS.

$$\Delta err_{cl} = \sqrt{(x_{1cl} - x_1)^2 + (y_{1cl} - y_1)^2 + (z_{1cl} - z_1)^2} \tag{14}$$

[2]In all the experiments reported below the following machine has been adopted: *PC Intel Core i7 8550U / 1.8GHz / 8 MB Cache / 16 GB DDR4 memory*

$$\Delta err_{par} = \sqrt{(x_{1par} - x_1)^2 + (y_{1par} - y_1)^2 + (z_{1par} - z_1)^2} \tag{15}$$

In the subsections below the 15 different explorations listed in Table 4 are discussed in details.

TABLE 4. Scenarios. [Please note that in the case of *parallel* design implementations N(M) stands for N iterations in the first step, with M iterations in the second step].

Scenario	Implementation	# of Iterat.	λ
Scenario 1 (SC #1)	*classic* *parallel*	1100 100(10)	0.5
Scenario 2 (SC #2)	*parallel*	100(10)	0.1, 0.2, 0.5, 0.8
Scenario 3 (SC #3)	*classic* *parallel*	1100 1650 100(10) 150(10)	0.5
Scenario 4 (SC #4)	classic	50,100, 250 500,1000	0.5

A. SCENARIO 1: FIXED ITERATIONS AND FIXED DAMPING FACTOR

This analysis refers to Scenario 1 in Table 4. All the results are obtained with a fixed damping factor $\lambda = 0.5$. For the *parallel* implementation, we have chosen 100 iterations in the first step of the algorithm, which leads to have 100 segments. Then, for each segment, 10 iterations of the DLS are executed. Therefore, the DLS is executed $100 + 100 * 10 = 1100$ times. To allow a fair computational comparison, *classic* is run with 1100 iterations too. Figure 5 reports on the error among the desired target and the actual reached position for each trajectory in [13]. This figure includes two curves: *Classic* is representative of Equation 14 and *Parallel* is representative of Equation 15. The trend of the curves is the same and the error increases with the trajectory ID that, as already stated in Section II-B, are ordered by length. From Figure 6 to Figure 8 the analysis of the different groups of trajectories in [13] is shown. Trajectories are grouped according to their length as described in Section II. In case of *small* trajectories both DLS implementations perform well. It is never the case that the error is above the worst given tolerance limit, which is set to 5 mm according to the requirements presented in

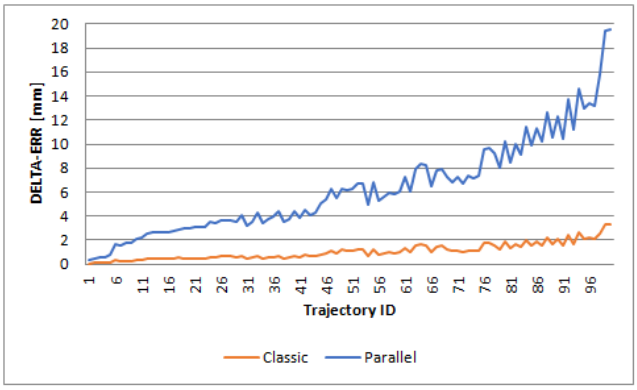

FIGURE 5. SC #1 - Error among the desired target and the actual reached position.

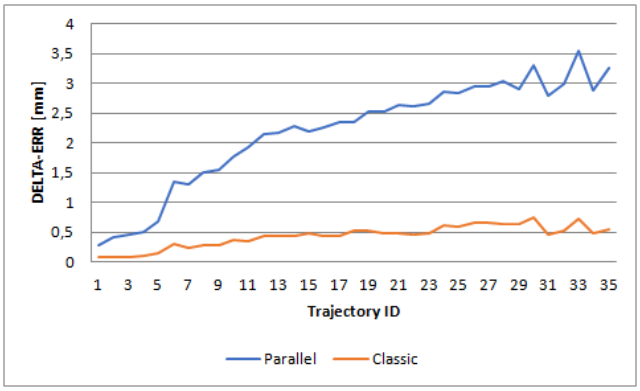

FIGURE 6. SC #1 - Focus on short trajectories: Error among the desired target and the actual reached position.

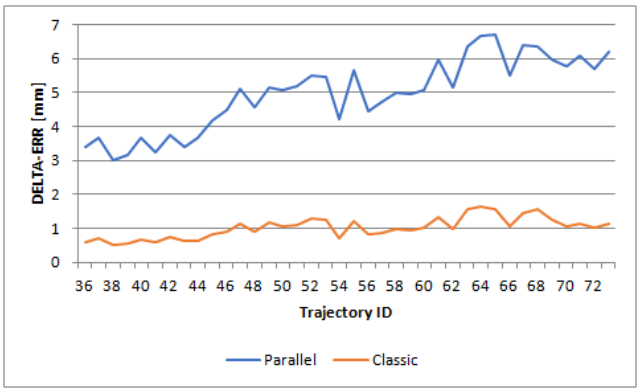

FIGURE 7. SC #1 - Focus on medium trajectories: Error among the desired target and the actual reached position.

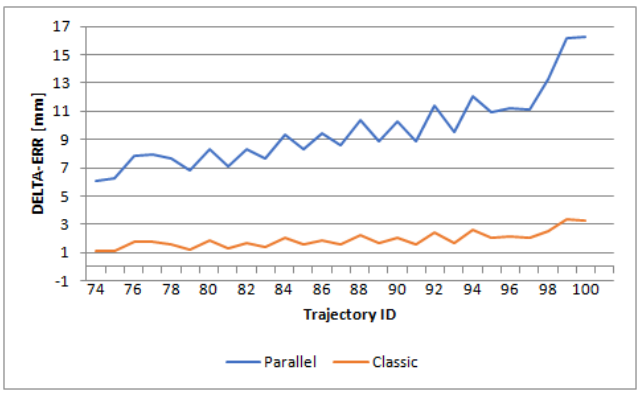

FIGURE 8. SC #1 - Focus on long trajectories: Error among the desired target and the actual reached position.

Section II-B. There is a good margin to even reduce the number of iterations in both the *classic* and the *parallel* case. For *medium* trajectories the situation changes. The *classic* implementation still guarantee that the requirement is met, the error is always below the given tolerance limit and there is still a good margin to reduce the number of iterations, maximum error is 1.65 mm. In the *parallel* case, instead, those trajectories with a length longer than 28 cm start to present an error with respect to the target that is larger than the required accuracy. For *long* the situation is the same: the *classic* implementation allows meeting the requirement with a maximum error of 3.03 mm, while this is never the case when the *parallel* one is adopted.

Both *classic* and *parallel* implementations would be fine for Task 2 (see Section II), which is characterized by trajectories having an approximate length of 10 cm belonging to the *short* group identified above. Therefore, in both cases the accuracy stay always below the 5 mm required by Task 2. If we consider the computation time, we can appreciate the advantage of the *parallel* solution with respect to the *classic* one. Figure 9 presents three curves reporting the elapsed time to compute *short* trajectories, two of them refer to the *parallel* implementation of the DLS. The *Ideal Parallel* considers having available an infinite number of parallel computing units; therefore, in Equation 13, $\frac{N}{k} = 1$ and $time_{par} = t_{s1} + t_{s2}$. The *Parallel Over* 4 *Slots* sets $k = 4$; therefore, in Equation 13, $\frac{N}{k} = 25$ and $time_{par} = t_{s1} + 25 * t_{s2}$. In general, the advantage of segmentation is clear. The average processing time for the *classic* implementation is 11.23 ms with a standard deviation of 2 ms, while for the realistic parallel execution over 4 slots the average processing time is 4.11 ms with a standard deviation of 1.61.

SHORT TRAJECTORIES

Parallelization via segmentation has positive effect on *short* trajectories, the error is kept within the tolerance. Timing advantage is there, being on average nearly 1/3 of the *classic* one. In practice, this advantage would strongly depend on the communication overhead, and target dependent results are expected.

In general, there are margins to reduce the number of iterations, since the difference between the computed errors and the expected accuracy is large. This is particularly true for the *classic* DLS implementation, which is discussed more in details in Section V-D. Nevertheless, reducing the number of iterations may impact on the smoothness of the movement itself. Therefore, the identification of the minimum number of iterations to be considered has to be defined also according to the physical impact on the movement.

The other two groups, *medium* and *long*, are relevant for Task 1, which contains trajectories that can be up to 50 cm long. In this case the accuracy is set to 10 mm. This means that the *parallel* implementation would fit when executing Task 1 over *medium* trajectories, being the actual error in this group always less than 7 mm. While for the *long* ones this is not the case: trajectories around 50 cm present an error around 10 mm (i.e Trajectory 87 has a length of 50.63 cm and an error of 8.61 mm, and Trajectory 88 has a length of 50.70 cm and an error of 10.4 mm). Trajectories from 88 to 100 have even larger error values, and are theoretically reachable by the arm, but not relevant for Task 2. In terms of elapsed execution

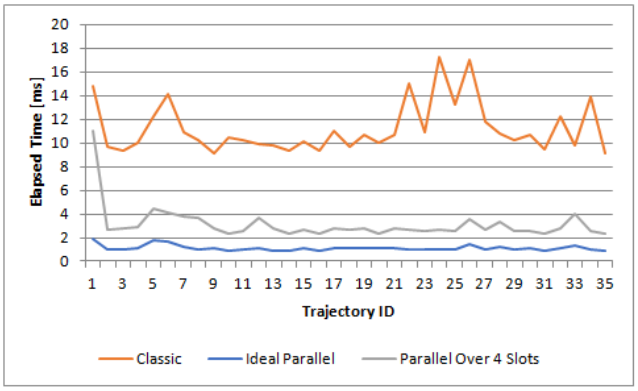

FIGURE 9. SC #1 - Focus on short trajectories: Elapsed time.

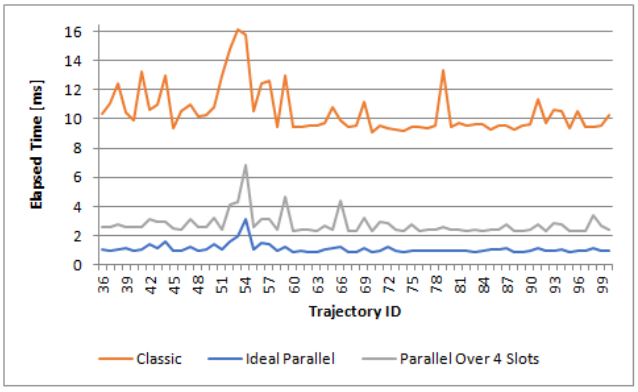

FIGURE 10. SC #1 - Focus on medium and long trajectories: Elapsed time.

time, the trend of the curves presented in Figure 10 is the same of Figure 9 and the theoretical saving is still approximately 1/3 in favor of the *parallel* implementation of the DLS with respect to the *classic* one.

MEDIUM TRAJECTORIES

Parallelization via segmentation has positive effects on *medium* trajectories too: the error is kept always within the tolerance and execution time is lower. Therefore, the same considerations made for the *short* case stands here too.

LONG TRAJECTORIES

Parallelization via segmentation does not have a generally positive effect on *long* trajectories. Tolerance boundaries are, in some cases, either exceeded or no margins are there. Viable solutions to overcome this issue may be the following:

- experiment different values of the damping factor, for which you should refer to Section V-B.
- change the number of segments, for which you should refer to Section V-C.

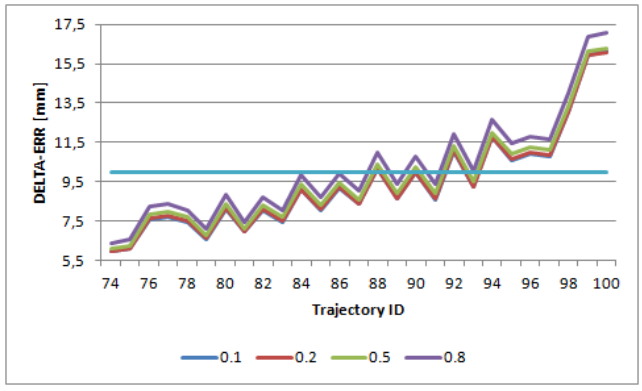

FIGURE 11. SC #2 - Focus long trajectories: Error among the desired target and the actual reached position.

TABLE 5. SC #2 - Focus on long trajectories: Samples example of error among the desired target and the actual reached position.

Trj ID	Length (cm)	Δerr_{par} [mm]			
		$\lambda = 0.1$	$\lambda = 0.2$	$\lambda = 0.5$	$\lambda = 0.8$
85	48.69	8.05	8.09	8.30	8.70
86	50.20	9.19	9.22	9.45	9.93
87	50.63	8.34	8.38	8.61	9.03
88	50.70	10.19	10.20	10.40	10.97

B. SCENARIO 2 - FOCUS ON PARALLEL: FIXED ITERATIONS AND VARIABLE DAMPING FACTOR

This analysis refers to Scenario 2 in Table 4. It focuses on the *parallel* implementation of the DLS and all the results are obtained for a fixed number of iterations (trajectories of 1100 points), while varying the damping factor λ that can assume the values 0.1, 0.2, 0.5 and 0.8. The damping factor λ influences not only the behavior around singularities, but also accuracy [38]: smaller λ should provide, in general, more accurate solutions. Therefore, choosing a small λ could solve the *long* trajectories issues discussed in the previous section. Figure 11 reports on this analysis for the *long* trajectories of the given workspace only, since they are those affected by accuracy issues when a *parallel* implementation of the DLS is chosen. The trend of the error for all the chosen damping factor value is the same and, as clearly demonstrated also in Table 5, the variation of λ, when we are so close to the workspace boundaries, is not sufficient to guarantee that the requirement on the accuracy is safely met.

LONG TRAJECTORIES

Parallelization via segmentation, even considering a variable λ, does not have a generally positive effect on *long* trajectories. Tolerance boundaries are still, in some cases, either exceeded or no margins are there.

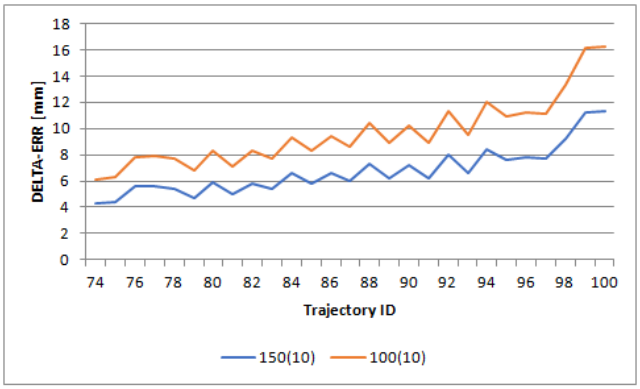

FIGURE 12. SC #3 - Focus long trajectories: Error among the desired target and the actual reached position.

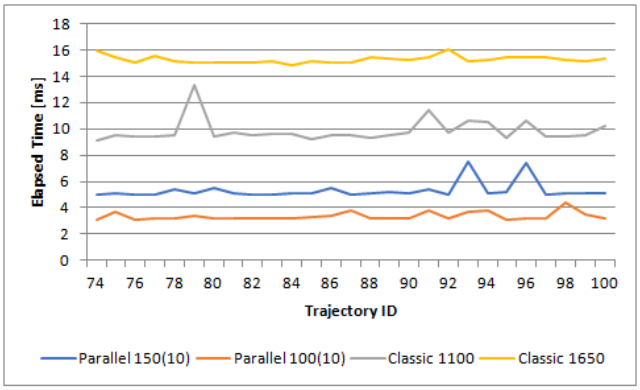

FIGURE 13. SC #3 - Focus on long trajectories: Elapsed time.

C. SCENARIO 3 - FOCUS ON PARALLEL: VARIABLE ITERATIONS IN THE 1ST DAMPED LEAST SQUARE STEP AND FIXED DAMPING FACTOR

The third considered scenario is meant to assess what would happen by changing the number of iterations in the first step of the *parallel* DLS implementation. Two different runs are compared, where the number of points is 100 (as in SC #1), and 150. The number of iterations per segment in the second step is kept equal to 10 and the damping factor is 0.5. The focus, as shown in Figure 12, is kept on the *long* trajectories of the given workspace only, since the issue of their accuracy when adopting a *parallel* DLS implementation is still there. The trend of the error curves is the same in both *100(10)* and *150(10)* cases, but increasing the overall number of segments has a positive effect: a reduced number of trajectories (98 to 100) is at the limit or exceeds the given accuracy requirement for Task 1. Nevertheless, having a closer look to them, as reported in Table 6, they all have a length above 60 cm, while Task 1 presents lengths around 50 cm. Therefore, despite being included in the workspace, they are not relevant for the considered task. The boundaries of Task 1 are provided by trajectories 85 to 88, which error is well below (maximum error is 7.34 mm, when up to 10 mm are admitted) the required accuracy when 150 iterations in step 1 are chosen. Figure 13 reports on the elapsed time,

TABLE 6. SC #2 - Focus on long trajectories: Samples example of error among the desired target and the actual reached position.

Trj ID	Length (cm)	Δerr_{par} [mm]	
		150(10)	100(10)
85	48.69	5.80	8.30
86	50.20	6.63	9.45
87	50.63	6.01	8.61
88	50.70	7.34	10.40
98	61.50	9.26	13.30
99	63.97	11.26	16.12
100	64.80	11.32	16.27

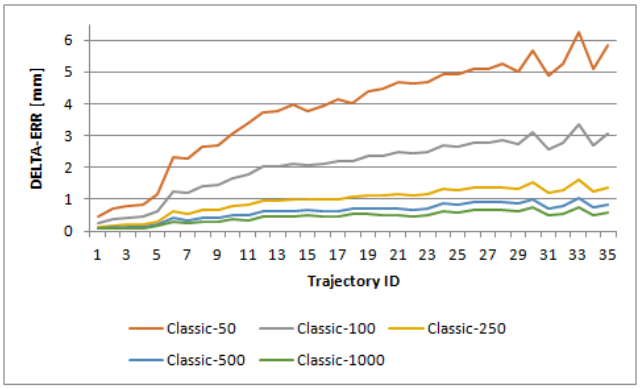

FIGURE 14. SC #4 - Focus on short trajectories: Error among the desired target and the actual reached position.

when 4 parallel computing units are chosen. Clearly, the more iterations are chosen, the higher is the elapsed time for the computation. In this figure also the elapsed time values for two runs of the *classic* DLS implementation are shown: Classic 1100 has as many iterations as Parallel 100(10), while Classic 1650 has as many iterations as Parallel 150(10). Still parallel implementations are preferable, in terms of elapsed time, than their corresponding classic version. Moreover, even considering 150 iterations in the first step, the elapsed time is smaller than in the classic implementation with fewer number of points.

LONG TRAJECTORIES Parallelization via segmentation is positively affected by an increased N, number of points in the fist iteration of the DLS. Accuracy is met with a good margin. In the reference problem considered in this article, there are no limitations on timing or on the number of parallel computing units that can be used, so that we can conclude that playing with N leads to keep the accuracy under control also on Task 1.

D. SCENARIO 4 – FOCUS ON CLASSIC ONLY: VARIABLE ITERATIONS AND FIXED DAMPING FACTOR

Having said that the number of iterations matters, this scenario is focused on exploring the relationship between the number of iterations and the accuracy when adopting the *classic* DLS implementation to solve IK problems. By looking at Figures 14 to 16 one may notice that, no matter

TABLE 7. SC #4 - Samples example of error among the desired target and the actual reached position (Δ) and density (D). For the sake of conciseness, $\Delta err_{cl} = \Delta$ and it is computed using Equation 14.

Task	Trj ID	Length (cm)	N=50		N=100		N=250		N=500		N=1000	
			Δ [mm]	D [mm]	Δ [mm]	D [mm]	Δ [mm]	D [mm]	Δ [mm]	D [mm]	Δ [mm]	D [mm]
2	6	10.45	2.31	2.00	1.26	1.02	0.62	0.41	0.41	0.21	0.31	0.10
	7	10.55	2.27	2.02	1.21	1.02	0.56	0.42	0.34	0.21	0.24	0.11
	8	11.56	2.64	2.22	1.41	1.13	0.66	0.46	0.41	0.23	0.30	0.12
1	85	48.69	15.10	9.33	7.95	4.76	3.66	1.93	2.26	0.97	1.60	0.49
	86	50.20	17.29	9.61	9.11	4.90	4.24	1.99	2.68	1.00	1.95	0.50
	87	50.63	15.70	9.71	8.26	4.95	3.78	2.01	2.32	1.01	1.63	0.50
	88	50.70	19.07	9.69	10.08	4.95	4.78	2.01	3.10	1.01	2.33	0.50

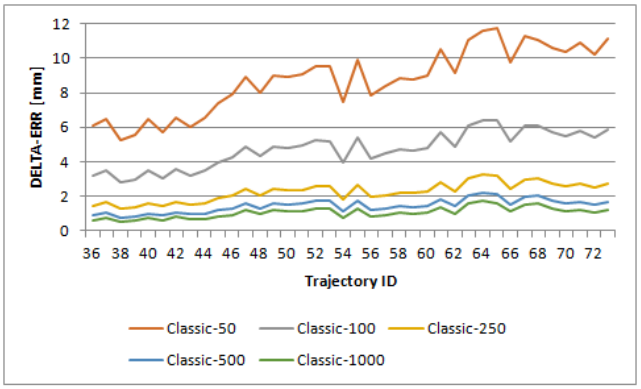

FIGURE 15. SC #4 - Focus on medium trajectories: Error among the desired target and the actual reached position.

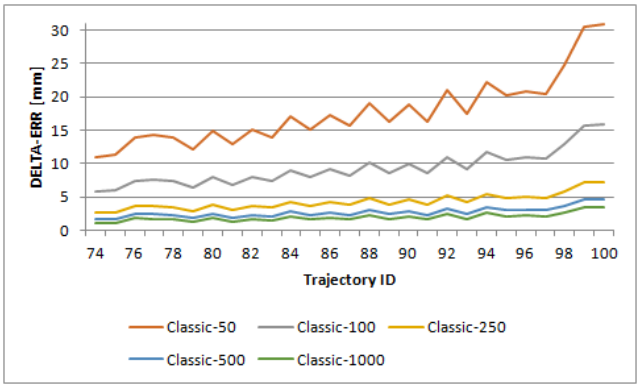

FIGURE 16. SC #4 - Focus on Long Trajectories: Error among the desired target and the actual reached position.

on the chosen number of iterations, the error trend is growing with the trajectory ID, that is with the trajectory length. Only for N = 500 and N = 1000 all the trajectories meet the worst case accuracy requirement (Task 2, 5 mm accuracy). By looking at Table 7, we can draw more considerations related to what happens at the boundaries of Task 1 and Task 2 in the considered workspace. Regarding Task 2, all the chosen N are fine and there is still a good margin to further reduce N since in the worst case an error of 2.64 mm is there, while the tolerance is set to 5 mm, so more or less 50%. This is not the case for Task 1. Given the chosen N values, we need at least N = 250 to guarantee that the tolerance of 10 mm is met: in the worst case an error of 4.78 mm is there, while the tolerance is set to 10 mm, so again more or less 50%. In both cases, you can assume a satisfactory margin when the density of points per trajectory (columns D in Table 7), defined as the length divided by the number of iterations, is around 0.2 cm.

ALL TRAJECTORIES

If we consider no constraint in terms of timing, a *classic* fully sequential implementation of the DLS IK algorithm can be chosen. Also in this case, it is important to properly choose the number of iterations to be performed. By looking at our reference problem, it can be empirically derived that, to have a considerable margin (50% as defined above) with respect to the reference requirement, a density of points per trajectory of 0.2 cm should be fine.

E. WRAP UP OF THE DAMPED LEAST SQUARE ANALYSIS

To summarize all the carried out analysis:

- a one-fit-to-all solution is not there. The DLS solver has to be characterized according to the executed task, to its requirements (e.g. accuracy) and to its features (e.g. approximate length of trajectories within a task).
- despite it is true that λ is fundamental to address the singularity problem, in a singularity-free workspace this parameter is not fundamental to meet the accuracy requirement, as the number of iterations turned out to be.

Given the fact that in our reference problem there are no timing limitations to be considered, which may lead to opt for a *parallel* solution, we decided to implement in hardware the *classic* algorithm. Indeed, the *classic* solver has proved to imply overall less iterations to meet the accuracy requirements with larger margins that, in turn, means less communication overhead. Please note that, a *parallel* implementation can be easily derived from the *classic* one, since it has only to be replicated while properly managing data flowing.

In the assessment below, implementations are characterized by: *i)* a static λ, which is fixed to 0.5, and *ii)* by a number of iterations per execution that is 1. Nevertheless, in all the proposed hardware implementations, λ can be changed at runtime, making them suitable for other contexts and workspaces which may involve singularities. Moreover, multiple runs of the hardware implementations can be done

according, e.g., to the empirical optimal density of points per trajectory derived in Section V-D, which is 0.2 cm. In summary, the proposed implementations can be potentially used with varying λ and number of iterations.

VI. HW IMPLEMENTATION

As already discussed in the introduction of this work, FPGA implementations could be extremely advantageous in the reference context since they are by construction flexible computing substrates, suitable to the reactive and dynamic behaviors characterizing CPSs, while being capable to efficiently support heavy computational workloads.

Therefore, in this section, we explore the possibility of implementing the DLS for the solution of IK problems with dedicated hardware. In Section VI-A we present the considered target architecture and the adopted design flow, in Section VI-B the DLS application is profiled and optimized, and finally in Section VI-C the obtained preliminary experimental results are discussed.

A. ARCHITECTURE AND DESIGN FLOW

Figure 17 illustrates the scenario setup addressed in this work. The Trossen Robotics WidowX Robotic aArm is controlled by an FPGA, in particular the one illustrated in figure is a Xilinx Zedboard, a Zynq FPGA that integrates the XC7Z020CLG484 chip. The reference architecture addressed in this work is a processor-coprocessor configuration where the processor runs most of the control flow tasks, while the coprocessor computes the onerous tasks, which in this case are related to the DLS calculation. The processor is the dual core ARM Cortex-A9 present on the addressed ZedBoard, while the coprocessor is fully custom and is generated with the Multi-Dataflow Composer (MDC) tool[3] [39], [40]. Processor and coprocessor communicate through an AMBA AXI4 system bus: the AXI4-Lite protocol is adopted for coprocessor configuration and control purposes, while the AXI4-full one is used for data transfers. This kind of coprocessing unit, being accessible through a commercial and widely adopted bus system, is suitable to different types of implementation. In the presented case, as discussed in Section V-E, there is no need of parallelization, but potentially the same type of coprocessor could be used in a parallel architecture too [5].

To deploy such a system, we can exploit a dataflow-based design flow, in which applications to be accelerated in the coprocessor are specified as dataflow models [41]. A dataflow is basically a functional programming language describing applications through directed graphs whose nodes are the operations of the applications, defined as the actors. Dataflows already revealed to be effective in addressing software and/or hardware design and mapping in heterogeneous systems [42]. Leveraging on dataflow specifications, it was possible to exploit a set of commercial and academic tools for a complete automated design flow. In particular,

[3]Available Open-Source at: https://github.com/mdc-suite/mdc

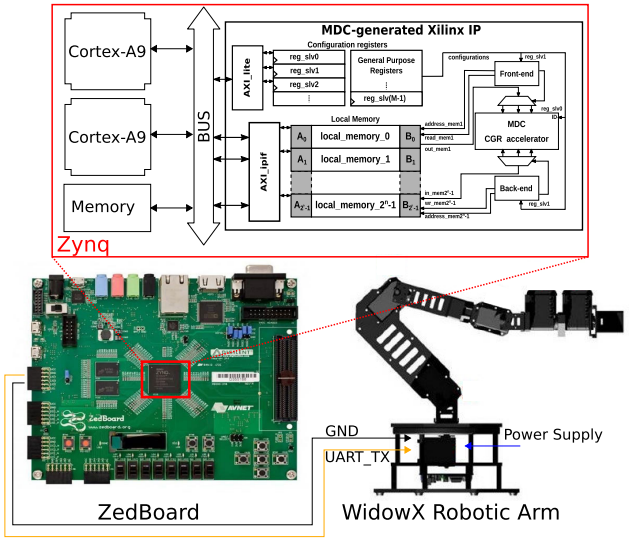

FIGURE 17. Overview of the robotic arm controller scenario and related processor-coprocessor architecture.

the MDC tool supports the coprocessor design and deployment. It provides a dataflow-based automated flow targeting reconfigurable coprocessors, which can implement i) different functionalities with some common operations, or ii) different profiles (e.g. in terms of timing performance, power consumption, etc.) of the same functionality. Actors can be specified using standard imperative languages (e.g. C), which can be used to derive the Hardware Description Language (HDL) specifications, necessary for hardware implementation of the coprocessor, by leveraging on HLS engines [43]. Once the functionalities are modelled through dataflows and the corresponding HDL actors are provided, MDC is capable of automatically deriving the corresponding reconfigurable coprocessor in which common actors are shared by the different functionalities/execution profiles, and reconfiguration is enabled by multiplexing resources in time. Please note that, the adoption of dataflows to model the DLS application and of the MDC tool to generate the system opens to the possibility of exploiting, in future, the advanced features offered by this tool, such as coprocessor topology optimization [44] and power management [45].

In the considered context, we used MDC reconfigurability support to derive a design able to execute different profiles for the *classic* implementation of the DLS algorithm. In particular, we explored a *baseline* (*BL*) profile with slow DLS calculation but limited power consumption, and a *high-performance* (*HP*) profile with fast DLS calculation but high power consumption. The capability of switching among different execution profiles is particularly useful in self-powered systems. The *HP* profile could potentially be enabled only when critical operations, with strict time constraints, have to be performed; while, in default conditions, the *BL* profile is enabled to preserve the batteries charge.

To port the above-mentioned profiles in hardware over an MDC-compliant coprocessor the following steps have

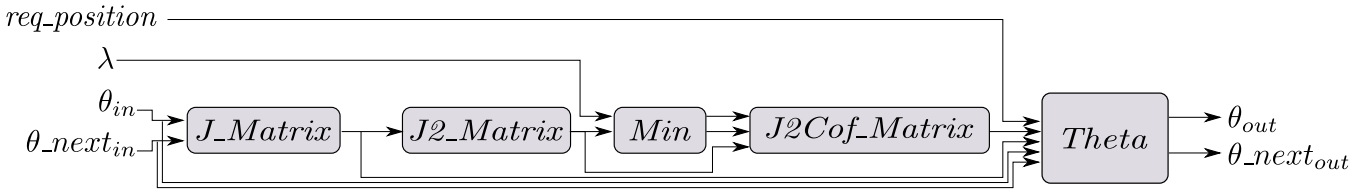

FIGURE 18. Graphical overview of DLS dataflow, in the *BL* version.

been performed. Starting from the MATLAB *classic* (see Section V) implementation of the DLS algorithm, the corresponding C code has been derived using the MATLAB Coder feature [46]. Once obtained an imperative C version, the DLS execution has been partitioned into five operations, which constitute the dataflow actors:

- *J_Matrix* calculates the Jacobian matrix J of the arm (see Equation 12);
- *J2_Matrix* calculates $\left(J^T J + \lambda^2 I\right)$;
- *Min* performs a first step for the calculation of $\left(J^T J + \lambda^2 I\right)^{-1}$;
- *J2Cof_Matrix* finalizes the calculation of $\left(J^T J + \lambda^2 I\right)^{-1}$;
- *Theta* performs final calculations of the new angles starting from the output of the other actors and according to the target position of the end effector (see Equation 9).

A graphical overview of the resulting dataflow, showing the *BL* profile, is depicted in Figure 18. Here, it is possible to see the operations execution order and mutual dependencies. Please note that, in the resulting hardware implementation, all the operations will be executed asynchronously since they are performed by dedicated logic and First In First Out (FIFO) data buffers are placed between actors, allowing for a natural implementation of pipelining.

The HDL descriptions of the actors have been automatically derived through Vivado HLS, which is the HLS engine provided by Xilinx [47]. As common HLS engines, Vivado HLS allows the developer to tune the obtained HDL, in particular playing on the trade-off among resources and latency, through pragmas. In this way, it is possible to derive several versions of the actors/operations with different resources versus latency trade-offs, as explained here-under.

B. APPLICATION PROFILING AND OPTIMIZATION

Based on the profiling and performance of the *BL* profile, the *HP* profile has been derived, leveraging on the pragmas made available by Vivado HLS. The pragmas have been applied to the most computationally intensive actors within the *BL* dataflow. Table 8 depicts the latency required by the *BL* actors for executing one step of the DLS algorithm. The latency has been measured through a post-synthesis simulation of the actors performed with Vivado Simulator. *J_Matrix* is the most time consuming actor, being responsible of about 45% of the overall latency. Also *J2_Matrix* and *Theta* are quite computationally intensive: their execution requires more than 20% of the overall execution time. *J2Cof_Matrix* and *Theta* are instead requiring not more than 10% of total latency each.

TABLE 8. Profiling of the *BL* DLS hardware implementation actors.

Actor	Latency [cck]	Latency %
J_Matrix	2712	45
J2_Matrix	1275	21
Min	158	3
J2Cof_Matrix	610	10
Theta	1266	21
sum	6021	100

According to the profiled data, three actors have been identified for latency optimization: *J_Matrix*, *J2_Matrix* and *Theta*. *J_Matrix_HP*, *J2_Matrix_HP* and *Theta_HP* have been derived instrumenting the corresponding C specifications with pragmas, mainly aimed at pipelining and unrolling inner loops of the computation. So that, two different execution profiles of the DLS hardware implementation have been developed:

- *BL* profile, involving non optimized *J_Matrix*, *J2_Matrix* and *Theta*;
- *HP* profile, involving the optimized *J_Matrix_HP*, *J2_Matrix_HP* and *Theta_HP* actors.

Please note that *BL* and *HP* profiles share the remaining actors, *Min* and *J2Cof_Matrix*.

C. PRELIMINARY ASSESSMENT

In this section we assess the reconfigurable DLS implementation (*reconf*), derived using the design flow described in Section VI-A, and capable of performing both *BL* and *HP* profiles. Please note that this assessment focuses on a preliminary study of the reconfigurable DLS hardware datapath (*MDC CGR accelerator* in Figure 17). Therefore, we mainly focused on the study of the resource occupancy on board, which gives a first hint on the feasibility of the implementation, and on the latency and power data, which give hints on the possibility of achieving effective trade-off switching between the BL and HL configurations. The proposed implementation has been evaluated considering: *i)* resource occupancy data, coming from logic synthesis of the actors and the reconfigurable DLS HDL specification through the Vivado Synthesis Solution; *ii)* latency data, measured with post-synthesis simulations of the design running at 100 MHz and performed with the Vivado Simulator; *iii)* power consumption data, given by the Vivado Power Report considering

TABLE 9. Resource occupancy of the different actors involved actors and of the reconfigurable DLS implementation. In brackets the percentage of variations for *HP* actors with respect to the corresponding *BL* version.

Actor/Design	Resources				Profile	
	LUT	FF	DSP	BRAM	*BL*	*HP*
J_Matrix	7052	3852	18	15	x	
J_Matrix_HP	23707 (+236)	12627 (+228)	88 (+389)	17 (+13)		x
J2_Matrix	1697	986	5	0	x	
J2_Matrix_HP	2621 (+54)	1600 (+62)	11 (+120)	0 (+0)		x
Min	1108	579	5	0	x	x
J2Cof_Matrix	2662	1628	8	2	x	x
Theta	2041	1263	5	0	x	
Theta_HP	2513 (+23)	1578 (+25)	8 (+60)	0 (+0)		x
sum	43401	24113	148	34	x	x
reconf	21966	20034	148	14	x	x

TABLE 10. Execution latency and power consumption of the reconfigurable DLS implementation. In brackets the percentage of variations for *HP* profile with respect to the *BL* one.

Profile	latency		power	
	[cck]	[μs]	[mW]	dynamic [mW]
BL	6331	63.31	239	117
HP	3424(-46)	34.24 (-46)	261 (+9)	139 (+19)

the real switching activity of the system gathered during the same post-synthesis simulations running at 100 MHz.

Table 9 depicts resources occupancy details of the *reconf* design and of its standalone actors. The most time consuming actor, that is *J_Matrix* according to Table 8, is also the most resource hungry one. Such relationship is not always verified since *J2Cof_Matrix*, which is one of the less time consuming actors according to Table 8, is the second one in terms of resource occupancy. What is sure is that *HP* actors consume always more resources than the corresponding *BL* version. Such increase in resources demand is maximum for *J_Matrix*, where DSP touches +389%, and minimum for BRAM in *J2_Matrix* and *Theta*. In terms of reconfigurability, we can notice that the *reconf* design, which can execute both *BL* and *HP* profiles, takes less resources than the sum of single actors (see **sum** row in Table 9) synthesized stand-alone. This is due to optimizations made by the synthesizer. Indeed, resources partially employed in single actor cases are fully exploited, in shared manner, in the *reconf* one.

Table 10 shows execution latency and power consumption of the reconfigurable DLS implementation while operating under the different execution profiles. The desired trade-off is now clear: providing *HP* versions of *J_Matrix*, *J2_Matrix* and *Theta* actors made it possible to almost halving the execution latency in the *HP* profile. In this case a single DLS iteration terminates in a bit more than 34 μs, against the almost 63 μs of the *BL* one. As a drawback, the increased resources usage of the *HP* profile is reflected on the consumed power: *HP* consumes 9% more overall power, which reaches 19% if the only dynamic contribution[4] is considered.

VII. CONCLUSION

In the context of CPSs design, it turned out to be particularly important for the embedded computing platforms to support complex execution workloads without sacrificing flexibility, to be able to dynamically react to system- and environmental-driven triggers. Heterogeneous platforms, such as modern FPGAs, could easily serve the purpose and, lately, have been used in many new application fields, one of them is certainly the robotic one.

In this article we presented the first attempt of implementing the DLS algorithm, for IK problems solution, over an FPGA substrate. The implementation has been realized leveraging on commercial (MATLAB, MATLAB Coder, Vivado HLS, Vivado) and open-source academic (MDC) tools, to mitigate the designer effort and make the implementation more straightforward also for software programmers. The developed implementation is suitable for playing at runtime with the main algorithm parameters, such as number of iterations or damping factor. Preliminary results showed that the proposed implementation is capable of processing one DLS iteration in less than 0.1 ms and that different execution profiles, trading-off execution latency and power consumption, can be enabled.

The other important contribution of this article is the analysis of the DLS for a perspective parallel implementation. From our study on the algorithm, it turned out that segmentation could be used to reduce the execution time, at the price of adopting more iterations of the DLS with respect to a classical sequential solution. Such a price could worth to be paid in strictly timing constrained scenarios, and a parallel hardware implementation like the one presented in [31] for the Nelder Mead IK solver could be adopted. The coprocessing infrastructure presented in Figure 17 is already compliant [5] with the parallel FPGA-based architecture presented in [4] and used in [31] for robotic purposes. Therefore, in future, a parallel hardware implementation of the DLS could be certainly attempted.

[4]In FPGA this term is usually most important than the static one since this latter, being not dependent on the occupied resources, is always constant for a given operating condition.

REFERENCES

[1] K.-D. Kim and P. R. Kumar, "Cyber–physical systems: A perspective at the centennial," *Proc. IEEE*, vol. 100, no. Special Centennial Issue, pp. 1287–1308, May 2012.

[2] K. Gac, G. Karpiel, and M. Petko, "FPGA based hardware accelerator for calculations of the parallel robot inverse kinematics," in *Proc. IEEE 17th Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2012, pp. 1–4.

[3] A. Brant and G. G. F. Lemieux, "ZUMA: An open FPGA overlay architecture," in *Proc. IEEE 20th Int. Symp. Field-Programmable Custom Comput. Mach.* Toronto, ON, Canada: IEEE Computer Society, Apr. 2012, pp. 93–96, doi: 10.1109/FCCM.2012.25.

[4] 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 ARTICo3 framework," *Sensors*, vol. 18, no. 6, p. 1877, Jun. 2018.

[5] T. Fanni, A. Rodríguez, C. Sau, L. Suriano, F. Palumbo, L. Raffo, and E. de la Torre, "Multi-grain reconfiguration for advanced adaptivity in cyber-physical systems," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs, (ReConFig)*, D. Andrews, R. Cumplido, C. Feregrino, and D. Stroobandt, Eds. Cancun, Mexico: IEEE Press, 2018, pp. 1–8, doi: 10.1109/RECONFIG.2018.8641705.

[6] A. Köpper and K. Berns, "Behaviour-based inverse kinematics solver on FPGA," in *Advances in Service and Industrial Robotics*, C. Ferraresi and G. Quaglia, Eds. Cham, Switzerland: Springer, 2018, pp. 55–62.

[7] A. Vaishnav, K. D. Pham, and D. Koch, "A survey on FPGA virtualization," in *Proc. 28th Int. Conf. Field Program. Log. Appl. (FPL)*. Dublin, Ireland: IEEE Computer Society, Aug. 2018, pp. 131–138, doi: 10.1109/FPL.2018.00031.

[8] F. Palumbo *et al.,* "CERBERO: Cross-layer model-based framework for multi-objective design of reconfigurable systems in uncertain hybrid environments: Invited paper: CERBERO teams from UniSS, UniCA, IBM research, TASE, INSA-rennes, UPM, USI, Abinsula, AmbieSense, TNO, S&T, CRF," in *Proc. 16th ACM Int. Conf. Comput. Frontiers*. Alghero, Italy: ACM, Apr. 2019, pp. 320–325, doi: 10.1145/3310273.3323436.

[9] F. Palumbo, T. Fanni, C. Sau, A. Rodriguez, D. Madronal, K. Desnos, A. Morvan, M. Pelcat, C. Rubattu, R. Lazcano, L. Raffo, E. de la Torre, E. Juárez, C. Sanz, and P. S. de Rojas, "Hardware/software self-adaptation in cps: The cerbero project approach," in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, D. N. Pnevmatikatos, M. Pelcat, and M. Jung, Eds. Cham, Switzerland: Springer, 2019, pp. 416–428.

[10] O. M. Omisore, S. Han, L. Ren, A. Elazab, L. Hui, T. Abdelhamid, N. A. Azeez, and L. Wang, "Deeply-learnt damped least-squares (DL-DLS) method for inverse kinematics of snake-like robots," *Neural Netw.*, vol. 107, pp. 34–47, Nov. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608018302181

[11] 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)*. New York, NY, USA: Association Computing Machinery, 2012, pp. 95–100, doi: 10.1145/2393091.2393111.

[12] *CERBERO Open Data—Damped Least Square: MatLab Implementations*. Accessed: Jul. 27, 2020. [Online]. Available: https://www.cerbero-h2020.eu/wp-content/uploads/2019/12/DLS-implementation.zip

[13] *CERBERO Open Data—Planetary Exploration Use Case: Trajectory generator*. Accessed: Jul. 27, 2020. [Online]. Available: https://www.cerbero-h2020.eu/wp-content/uploads/2019/12/Trajectory-Generator.zip

[14] *WidowX Robot Arm, Trossen Robotics*. Accessed: Jul. 8, 2020. [Online]. Available: https://www.trossenrobotics.com/widowxrobotarm

[15] *CERBERO Open Data—Planetary Exploration Use Case: Robotic Arm Workspace*. Accessed: Jul. 27, 2020. [Online]. Available: https://www.cerbero-h2020.eu/wp-content/uploads/2019/12/Open-Data-PE.zip

[16] A. Aristidou, J. Lasenby, Y. Chrysanthou, and A. Shamir, "Inverse kinematics techniques in computer graphics: A survey," *Comput. Graph. Forum*, vol. 37, no. 6, pp. 35–58, Sep. 2018.

[17] M. H. Wright, "Nelder, Mead, and the other simplex method," *Documenta Mathematica*, vol. 7, pp. 271–276, Aug. 2010.

[18] S. R. Buss, "Introduction to inverse kinematics with Jacobian transpose, pseudoinverse and damped least squares methods," *IEEE J. Robot. Automat.*, pp. 681–685, 2004.

[19] F. Lewis, D. Dawson, and C. Abdallah, *Robot Manipulator Control: Theory and Practice*, 2nd ed. New York, NY, USA: Marcel Dekker, 2004.

[20] L. Unzueta, M. Peinado, R. Boulic, and Á. Suescun, "Full-body performance animation with sequential inverse kinematics," *Graph. Models*, vol. 70, no. 5, pp. 87–104, Sep. 2008.

[21] A. Liegeois, "Automatic supervisory control of the configuration and behavior of multibody mechanisms," *IEEE Trans. Syst., Man, Cybern.*, vol. 7, no. 12, pp. 842–868, Dec. 1977.

[22] A. A. Maciejewski and C. A. Klein, "Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments," *Int. J. Robot. Res.*, vol. 4, no. 3, pp. 109–117, Sep. 1985.

[23] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C* (The Art Scientific Computing), 2nd ed. Cambridge, MA, USA: Cambridge Univ. Press, 1992.

[24] S. R. Buss and J.-S. Kim, "Selectively damped least squares for inverse kinematics," *J. Graph. Tools*, vol. 10, no. 3, pp. 37–49, Jan. 2005.

[25] M. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Dynamics and Control*, 2nd ed. Hoboken, NJ, USA: Wiley, 2004.

[26] P. Caserman, P. Achenbach, and S. Gobel, "Analysis of inverse kinematics solutions for full-body reconstruction in virtual reality," in *Proc. IEEE 7th Int. Conf. Serious Games Appl. Health (SeGAH)*, Aug. 2019, pp. 1–8.

[27] C. Sau, F. Palumbo, M. Pelcat, J. Heulot, E. Nogues, D. Menard, P. Meloni, and L. Raffo, "Challenging the best HEVC fractional pixel FPGA interpolators with reconfigurable and multifrequency approximate computing," *IEEE Embedded Syst. Lett.*, vol. 9, no. 3, pp. 65–68, Sep. 2017.

[28] W.-C. Chen, C.-S. Chen, F.-C. Lee, and Y.-S. Kung, "Digital hardware implementation of the forward/inverse kinematics for a SCARA robot manipulator," in *Proc. IEEE Int. Conf. Appl. Syst. Invention (ICASI)*, Apr. 2018, pp. 54–57.

[29] D. Hildenbrand, H. Lange, F. Stock, and A. Koch, "Efficient inverse kinematics algorithm based on conformal geometric algebra-using reconfigurable hardware," in *Proc. GRAPP*, 2008, pp. 1–8.

[30] J.-Y. Yu, D. Huang, C.-C. Liang, B. Wang, X.-D. Zhang, and L. Lu, "On-board real-time path planning design for large-scale 7-DOF space manipulator," in *Proc. IEEE Int. Conf. Mechatronics Autom.*, Aug. 2016, pp. 1501–1506.

[31] L. Suriano, A. Otero, A. Rodríguez, M. Sánchez-Renedo, and E. D. L. Torre, "Exploiting multi-level parallelism for run-time adaptive inverse kinematics on heterogeneous mpsocs," *IEEE Access*, vol. 8, pp. 118707–118724, 2020.

[32] S. Mukherjee and A. Mahapatra, "Compliant contact modeling of a humanoid robotic thumb for assisting grasp," in *Proc. IEEE Int. Conf. Intell. Syst. Green Technol. (ICISGT)*, Jun. 2019, pp. 94–943.

[33] J. Zhao, T. Xu, Q. Fang, Y. Xie, and Y. Zhu, "A synthetic inverse kinematic algorithm for 7-DOF redundant manipulator," in *Proc. IEEE Int. Conf. Real-time Comput. Robot. (RCAR)*, Aug. 2018, pp. 112–117.

[34] X. Wang, J. Cao, X. Liu, L. Chen, and H. Hu, "An enhanced step-size Gaussian damped least squares method based on machine learning for inverse kinematics of redundant robots," *IEEE Access*, vol. 8, pp. 68057–68067, 2020.

[35] X. Wang, J. Cao, L. Chen, and H. Hu, "Two optimized general methods for inverse kinematics of 6R robots based on machine learning," *Math. Problems Eng.*, vol. 2020, pp. 1–14, Jan. 2020.

[36] J. Wu, D. Bin, X. Feng, Z. Wen, and Y. Zhang, "GA based adaptive singularity-robust path planning of space robot for on-orbit detection," *Complexity*, vol. 2018, pp. 1–11, May 2018, doi: 10.1155/2018/3702916.

[37] P. Liu, D. Yu, and R. Li, "Research on singular robustness algorithm of robot inverse kinematics based on dynamic damping coefficient," in *Proc. IEEE 3rd Inf. Technol. Mechatronics Eng. Conf. (ITOEC)*, Oct. 2017, pp. 204–209.

[38] S. Chiaverini, B. Siciliano, and O. Egeland, "Review of the damped least-squares inverse kinematics with experiments on an industrial robot manipulator," *IEEE Trans. Control Syst. Technol.*, vol. 2, no. 2, pp. 123–134, Jun. 1994, doi: 10.1109/87.294335.

[39] F. Palumbo, D. Pani, E. Manca, L. Raffo, M. Mattavelli, and G. Roquier, "RVC: A multi-decoder CAL composer tool," in *Proc. Conf. Design Archit. Signal Image Process. (DASIP)*, Oct. 2010, pp. 144–151.

[40] C. Sau, P. Meloni, L. Raffo, F. Palumbo, E. Bezati, S. Casale-Brunet, and M. Mattavelli, "Automated design flow for multi-functional dataflow-based platforms," *J. Signal Process. Syst.*, vol. 85, no. 1, pp. 143–165, Oct. 2016, doi: 10.1007/s11265-015-1026-0.

[41] E. A. Lee and T. M. Parks, "Dataflow process networks," *Proc. IEEE*, vol. 83, no. 5, pp. 773–801, May 1995.

[42] L. Li, C. Sau, T. Fanni, J. Li, T. Viitanen, F. Christophe, F. Palumbo, L. Raffo, H. Huttunen, J. Takala, and S. S. Bhattacharyya, "An integrated hardware/software design methodology for signal processing systems," *J. Syst. Archit.*, vol. 93, pp. 1–19, Feb. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762118301735

[43] C. Rubattu, F. Palumbo, C. Sau, R. Salvador, J. Serot, K. Desnos, L. Raffo, and M. Pelcat, "Dataflow-functional high-level synthesis for coarse-grained reconfigurable accelerators," *IEEE Embedded Syst. Lett.*, vol. 11, no. 3, pp. 69–72, Sep. 2019.

[44] F. Palumbo, C. Sau, and L. Raffo, "DSE and profiling of multi-context coarse-grained reconfigurable systems," in *Proc. 8th Int. Symp. Image Signal Process. Anal. (ISPA)*, Trieste, Italy, Sep. 2013, pp. 744–749. [Online]. Available: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=6703836

[45] T. Fanni, C. Sau, P. Meloni, L. Raffo, and F. Palumbo, "Power and clock gating modelling in coarse grained reconfigurable systems," in *Proc. ACM Int. Conf. Comput. Frontiers (CF)*. New York, NY, USA: Association Computing Machinery, 2016, pp. 384–391, doi: 10.1145/2903150.2911713.

[46] *Matlab Coder*. Accessed: Jul. 27, 2020. [Online]. Available: https://www.mathworks.com/products/matlab-coder.html

[47] *Vivado High-Level Synthesis*. Accessed: Jul. 27, 2020. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html

CARLO SAU (Member, IEEE) received the degree in electronic engineering and the Ph.D. degree from the University of Cagliari, in 2012 and 2016, respectively. He is currently an Assistant Professor with the University of Cagliari. Since 2012, he has been involved in automated methodologies for dataflow-based reconfigurable platforms generation. His main research interests include reconfigurable system design and the development of code generation tools for advanced reconfigurable hardware architectures.

TIZIANA FANNI received the degree in electronic engineering and the Ph.D. degree from the University of Cagliari, in 2014 and 2019, respectively. From 2014 to September 2019, she stayed with the Department of Electrical and Electronic Engineering, University of Cagliari, where she was working on power saving methodologies for dataflow-based reconfigurable platforms. She is currently a Postdoctoral Researcher with the University of Sassari. Her main research interests include reconfigurable systems design and the development of code generation tools for low-power reconfigurable hardware architectures.

CLAUDIO RUBATTU (Member, IEEE) received the degree in electronic engineering from the University of Cagliari, Italy, in 2015. He is currently pursuing the Ph.D. degree with the INSA Rennes, France. In May 2015, he started a one year and six months research grant related to embedded system design in bio-medical applications at the University of Cagliari. He is also a Research Assistant with the University of Sassari, Italy. In particular, he also focused on the development of predictable models of architecture for multicore systems with hardware acceleration in the context of embedded systems and CPSs. His recent publications are related to dataflow-based design flows aimed at achieving predictability for systems that consider coarse-grained reconfigurable accelerators. His main research interests include reconfigurable systems design and the development of code generation tools for reconfigurable hardware/software architectures.

LUCA FANNI received the degree in electronic engineering from Roma Tre University, in 2018. He worked as a Research Fellow with the University of Sassari, up to January 2020.

LUIGI RAFFO (Member, IEEE) received the Laurea degree in electronic engineering and the Ph.D. degree in electronics and computer science from the University of Genoa, Italy, in 1989 and 1994, respectively. In 1994, he joined the Department of Electrical and Electronic Engineering, University of Cagliari, Italy, as an Assistant Professor, and an Associate Professor, in 1998, where he has also been a Full Professor of Electronics, since 2006. He teaches courses on system/digital and analog electronic design and processor architectures for the courses of studies in electronic and biomedical engineering. He was a Coordinator of the Project EU IST-FET-IST-2001-39266-BEST and the MADNESS EU Project. He was also a Unit Coordinator of the Project EU IST-FET-SHAPES—Scalable Software Hardware Architecture Platform for Embedded Systems. He is also a Local Coordinator of industrial projects in the field (among others: ST-Microelectronics—Extension of ST200 architecture for ARM binary compatibility and ST-Microelectronics—Network on chip). He is also responsible for the cooperation programs in the field of embedded systems with several other European universities. He was a Local Coordinator of the ASAM (ARTEMIS-JU) and ALBA projects (national founded project) and RPCT (regional founded project).

FRANCESCA PALUMBO (Member, IEEE) received the Laurea degree *(summa cum laude)* in electronic engineering from the University of Cagliari, in 2005, the master's degree in advanced embedded system design from the Advanced Learning and Research Institute, University of Lugano, in 2006, and the Ph.D. degree in electronic and computer engineering from the University of Cagliari. She is currently an Associate Professor with the University of Sassari. Her research interests include reconfigurable systems and to code generation tools and design automation strategies for advanced reconfigurable hardware architectures. She is also a Permanent Steering Committee Member of the ACM Conference on Computing Frontiers. For her studies in the fields of dataflow-based programming and hardware customization, she received two best paper awards at the Conference on Design and Architectures for Signal and Image Processing, in 2011 and 2015. She serves in several different technical committee of international conferences. She was the Scientific Coordinator of the CERBERO H2020 European Project on Smart Cyber Physical System Design and the Scientific Director of the Summer School entitled Designing Cyber-Physical Systems—From concepts to implementation. She is also an Associate Editor of the *Journal of Signal Processing Systems* (Springer) and the IEEE Embedded Systems Letters.