A Dataflow Implementation of Inverse Kinematics on Reconfigurable Heterogeneous MPSoC

Luca Fanni[1], Leonardo Suriano[2], Claudio Rubattu[1,3], Pablo Sánchez de Rojas[4], Eduardo de la Torre[2], and Francesca Palumbo[1]

[1] University of Sassari, 07100 Sassari, Italy
[2] Univesidad Politécnica de Madrid, 28006 Madrid, Spain
[3] Univ Rennes, INSA Rennes, IETR UMR CNRS 6164, 35700 Rennes, France
[4] Thales Alenia Space España, 28760 Madrid, Spain

Abstract. This paper describes the activities related to the implementation of a robotic arm controller based on the Damped Least Square algorithm to numerically solve Inverse Kinematics problems over a heterogeneous MPSoC platform.

Keywords: Inverse Kinematics · Damped Least Square · heterogeneous MPSoC · FPGA · CERBERO project.

1 Introduction

The H2020 CERBERO [7] project is developing a continuous design environment for Cyber Physical Systems (CPS), leveraging on a large set of tools that support both run-time and design-time issues of CPS [21]. One of the main outcomes of the project, beside the continuous design framework, is the extensive and efficient support for run-time adaptation [22]. The algorithm and demonstration set-up presented below is compliant with CERBERO Planetary Exploration (PE) use-case scenario, which is built using CERBERO technologies and aims at assessing heterogeneous and reconfigurable Multi-Processor System on Chip (MPSoC) solutions in space applications. The final demonstrator of this scenario will be the controller of a robotic arm implemented over a Field Programmable Gate Array (FPGA) device, with advanced self-monitoring and self-adaptive processing capabilities to ruggedize the systems under stringent survival conditions (radiation and harsh environment) and meet the reliability constraints of a robotic exploration mission. In this work we present the preliminary implementation of such controller using the Damped Least Square algorithm [9,10].

Demonstrator Setup: The setup used for testing purposes is composed by: **(1)** a Digilent/Xilinx ZedBoard, **(2)** a Trossen Robotics WidowX robotic arm, **(3)** Power supplies for both the robotic arm and the ZedBoard and **(4)** a Personal Computer.

The manipulator is a WidowX Robot Arm by Trossen Robotics, characterized by four main Degrees of Freedom (DOF) plus other two DOF for wrist and clamp movements. The manipulator is equipped with six Dynamixel actuators, and

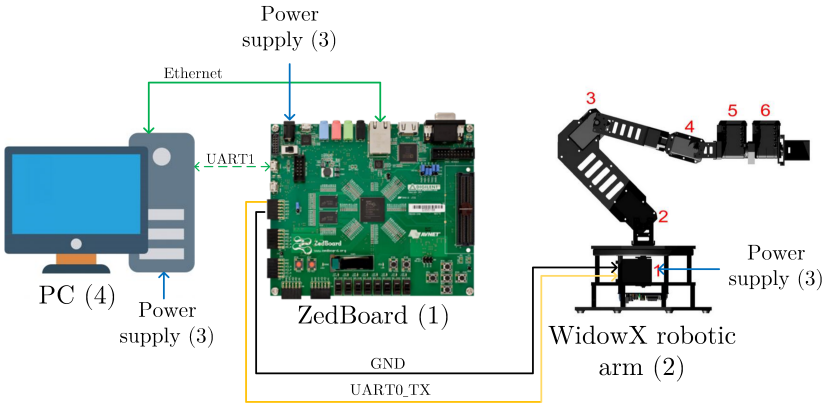


Fig. 1: Demonstrator Setup.

each of them provides a full control of the relative joint via a digital communication using an Universal Asynchronous Receiver-Transmitter (UART) protocol. For each actuator, several parameters can be controlled, such as the angular position, speed and acceleration.
The external controller is based on a Digilent/Xilinx ZedBoard equipped with a Zynq®-7000 MPSoC. A Linux-based Operating System (OS) distribution running on the Zynq ARM processor provides the environment to compile and execute the Inverse Kinematics (IK) algorithm and to control the manipulator actuators.
The Personal Computer (PC) is used to design the IK application and, if necessary, compile and upload its executable on the MPSoC via Ethernet connection. The numerical control of the robotic arm is entirely delegated to the ZedBoard.

2 Forward and Inverse Kinematics

Any robotic manipulator is composed of different parts, namely: *(i)* a base; *(ii)* rigid links; *(iii)* joints (each of them connecting two adjacent links); and *(iv)* an end effector. As shown in Figure 2, physically an angle can be associated to each rotational joint, or a displacement for a prismatic joint. The end effector can be characterized by specific spatial coordinates. To determine the end-effector spatial coordinates or the angles for each joint, you could leverage on two different techniques: computing the end-effector spatial coordinates from all the joint angles, or computing the joint angles from a desired end-effector spatial coordinate. The former is known as *Forward Kinematics* problem (FK), the latter as *Inverse Kinematics* problem (IK). In the following we will focus on IK. Indeed,

implementing an arm controller you need to derive the angles of the joints of the chosen arm to move it in a desired position.

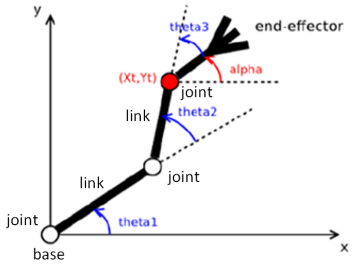


Fig. 2: A generalized robotic manipulator.

Inverse Kinematics problem: Inverse Kinematics typically requires to solve a complex problem, since: (i) more than a solution can be found for a desired end-effector position, for example elbow-up or elbow-down poses could both lead to reach the same spatial coordinate; (ii) there could be joint angles limitations; and finally, (iii) out-of-reach end-effector spatial coordinates could be experienced. These limitations define the manipulator workspace, and an attempt to get over them can lead to a singularity in the IK problem. Many solutions, numerical and non-numerical ones, are present in literature [8]. Numerical methods require various iterations to converge over a solution, but are better capable of dealing with DOF, and multiple end effectors (e.g. fingers of a hand or arms of a body) with respect to analytic solutions and data-driven methods. Among the numerical solutions we can list: Heuristic methods and Cyclic Coordinate Descendant ones [32]; Newton-based methods (exploiting second-order Taylor expansion), and Jacobian-based methods (exploiting inverted, pseudo-inverted and transposed Jacobian matrices) [9,10]. In this work we opted for the latter.

3 Jacobian-based methods

3.1 Fundamentals

Jacobian-based methods are based on the Jacobian matrix [17], defined as:

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

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. Considering the following equation:

$$\Delta\overrightarrow{\Theta} = J^{\dagger}\Delta\overrightarrow{e} \tag{1}$$

$\Delta\overrightarrow{\Theta}$ is the joint space vector, $\Delta\overrightarrow{e}$ is the error or displacement vector and $J^{\dagger}$ (see Section 3.3) is a matrix computed from the Jacobian matrix J. The $\Delta\overrightarrow{\Theta}$ and $\Delta\overrightarrow{e}$ vectors are defined as: $\Delta\overrightarrow{\Theta} = [\,\Delta\theta_1\ \Delta\theta_2\ \ldots\ \Delta\theta_n]^T$ and $\Delta\overrightarrow{e} = \left[\Delta x\ \Delta y\ \Delta z\ \delta x\ \delta y\ \delta z\right]^T$. Please note that $[\,\Delta x\ \Delta y\ \Delta z\,]^T$ defines a linear displacement, and $[\,\delta x\ \delta y\ \delta z\,]^T$ defines a rotational displacement. In our case we consider linear displacements only with $[\,\delta x\ \delta y\ \delta z\,]^T = [\,0\ 0\ 0\,]^T$. The trajectory will be calculated from the values assigned to $[\,\Delta x\ \Delta y\ \Delta z\,]^T$.

3.2 Inverse Kinematics Calculation

In order to obtain the integral angles from the joint space, $\Delta\overrightarrow{\Theta}$, we need to add them to the previous thetas, $\overrightarrow{\Theta}_{old}$, to obtain: $\overrightarrow{\Theta}_{new} = \overrightarrow{\Theta}_{old} + \Delta\overrightarrow{\Theta}$. This step is normally iterated several times. At any iteration, once obtained the $\overrightarrow{\Theta}_{new}$ vector, this is used as starting vector to compute $\overrightarrow{\Theta}_{new}$ in the next step, thus becoming $\overrightarrow{\Theta}_{old}$. This procedure is equivalent to calculate the defined integral of thetas along the trajectory. The trajectory itself is defined 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 are defined as the desired position to be reached. The general flow of Jacobian-based IK algorithms is shown in Figure 3.

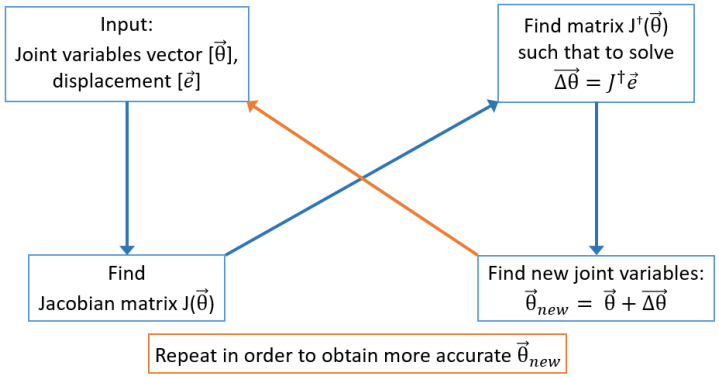


Fig. 3: Jacobian-based IK algorithms flow diagram

Prior to execute the algorithm, the displacement vector must be defined as: $\Delta\overrightarrow{e} = \left[\frac{x_{fin}-x_{init}}{i+1}\ \frac{y_{fin}-y_{init}}{i+1}\ \frac{z_{fin}-z_{init}}{i+1}\ 0\ 0\ 0\right]^T$, 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.

3.3 Computing the $J^\dagger$

There are several methods [9] to compute $J^\dagger$, such as: transpose of J, pseudo-inverse of J, SVD-based computing, Damped Least Squares (DLS). The first and the second ones cannot handle properly events such as singularities. The third is based on the eigenvalues and eigenvectors computing at high computational costs. Then we opted for the DLS.

Damped Least Square The *Damped Least Square*, also known as the *Levenberg—Marquardt* algorithm [10], computes the $J^\dagger$ as:

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

Using Equation (2), the Equation (1) can be expressed as:

$$\Delta\overrightarrow{\Theta} = \left(J^T J + \lambda^2 I\right)^{-1} J^T \Delta\overrightarrow{e} \tag{3}$$

Equation (3) uses a λ factor to handle singularities. 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 the second scenario, to choose dynamically the λ factor several algorithms [13]. At the moment we use a static approach to determine λ, but in future we plan to dynamically derive it using the Sugihara method [27].

4 Manipulator characterization

In order to obtain the Jacobian J and the $J^\dagger$ matrices a preliminary manipulator characterization is necessary. The manipulator has to be defined as a mathematical model, and from this model the Jacobian matrix J associated to the manipulator can be calculated and used in the DLS algorithm. A common way to define a mathematical model of a robotic manipulator uses the *Denavit-Hartenberg* (DH) parameters [26]. A set of four parameters composed of two angles (α and θ) and two displacements (a and d) is defined for each link-joint group. These parameters are obtained from the manipulator mechanical dimensions and joint orientations, as shown in Figure 4.

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

$$^{n-1}T_n = \begin{bmatrix} \cos\theta_n & -\sin\theta_n\cos\alpha_n & \sin\theta_n\sin\alpha_n & a_n\cos\theta_n \\ \sin\theta_n & \cos\theta_n\cos\alpha_n & -\cos\theta_n\sin\alpha_n & a_n\sin\theta_n \\ 0 & \sin\alpha_n & \cos\alpha_n & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

This transformation matrix represents the position and the orientation of the n^{th}-joint 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:

$$T = {}^0T_1 {}^1T_2 \ldots {}^{N-1}T_N = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

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

Table 1: DH parameters

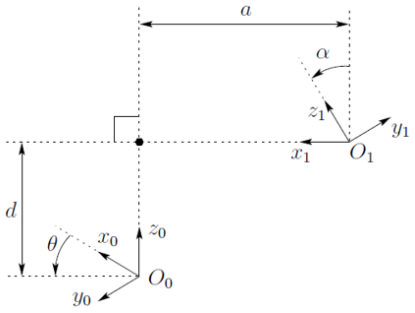


Fig. 4: Graphical representation of DH parameters

This matrix represent the end-effector spatial coordinates and orientation respect to the manipulator base coordinates. The derived DH parameters for our test manipulator are tabulated in Table 1, and can be used in Equation (4) and Equation (5) to obtain the general transformation matrix orientation coefficients for our manipulator (n_x, n_y, n_z, o_x, o_y, o_z, a_x, a_y, and a_z), the position coefficients (p_x, p_y, and p_z) and the manipulator Jacobian according to the theory in [17].

Please note that the coefficients p_x, p_y and p_z are representing the end-effector spatial coordinates with respect to joint angles, representing the manipulator FK. As stated in Section 3.2, FK is necessary to compute the initial point for the algorithm.

5 Porting the DLS over an Heterogeneous MPSoC

In the scope of the CERBERO project, one of the goal is to access Programmable Logic (PL) within FPGA technologies with the purpose of offloading computational intensive tasks by delegating them to custom hardware accelerators. Figure 5 shows the steps of the entire workflow. The steps within the graph are explained in details hereafter.

1. **Inverse Kinematics MATLAB Description**: The IK algorithm has been implemented and tested in a native matrix-oriented MATLAB® environment. It allows various optimized computations such as the Jacobian matrix automatically derived from the DH description. Using MATLAB, the entire error-prone process of generating C-code is speeded up. Result of the assessment for this step are presented in Section 6.
2. **C - code**: This first description of the DLS algorithm in C code is, here, automatically generated by using *MATLAB Coder* [1].
3. **Parameterized and Interfaced Synchronous DataFlow (PiSDF) Description [12]**: This is a key step and it is manually executed: the full robotic arm application is described using the PiSDF Model of Computation (MoC).

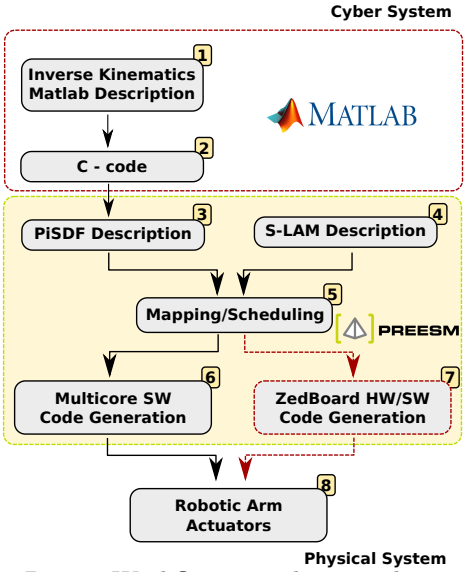


Fig. 5: Design Workflow: step-by-step description.

The previously generated C-code is used to described the internal behavior of its actors. Fig 6 provides a PiSDF description of the DLS.

4. **System-Level Architecture Model (S-LAM) description [24]**: The strength of PREESM is to give the possibility of independently describe (i) the algorithm and (ii) the architecture. Thus, PREESM will be in charge of finding an optimal solution by mapping and scheduling every instance of actors of the PiSDF onto the S-LAM.
5. **Mapping/Scheduling [23]**: This functionality is embedded in the PREESM toolflow: when the problem is correctly defined, the tool schedules the algorithm upon the architecture using heuristic methods [2].
6. **Multicore SW Code Generation**: The Directed Acyclic Graph (DAG) produced by PREESM is internally used by the workflow to automatically generate a C code with multiple threads (Pthreads of the C-standard). The code is, now, ready to be compiled and tested on the target architecture (in this case on the two A9 ARM cores of the Zynq MPSoC). The efficiency of the approach was demonstrated for shared-memory MPSoC [11] (as the case of the Processing System (PS) of the ZedBoard).
7. **ZedBoard HW/SW Code Generation**: This step is not fully completed. A first HW/SW code generation can be obtained by feeding Vivado SDSoC with a PiSDF tested in PREESM [30,28] (for further details see the PREESM tutorial [3]). As a starting point, a profiling of the application is necessary for the identification of the most computationally intensive tasks that could be accelerated. When the design requirements imply flexibility of the application, the proposed design flow could be integrated with the multi-grain approach presented in [14], which offers to the developer the possibility to play

with the main reconfiguration strategies (Dynamic Partial Reconfiguration (DPR) [18] and Coarse Grain Reconfiguration (CGR) [31]). In the CERBERO toolchain, these methods are implemented respectively by ARTICo^3 [25] and MDC [4,16,15]. Furthermore, a self-adaptive behavior which allows the automatic reconfiguration of the system is often a desirable property. In this context, a proper monitoring of the hardware and software tasks is required. PREESM is already integrated with PAPIFY [5,20], a tool based on PAPI [6] that eases access to the Performance Monitor Counters (PMCs) already available on the CPUs for heterogeneous systems, and is able to provide a unified interface to access also custom monitors placed in the HW accelerators [19,29].

8. **Robotic Arm Actuators**: The generated C code is ready to be compiled (if computationally possible on the ARM cores) and executed directly on the ZedBoard which is in charge of driving the actuator for moving the WidowX in the Physical world.

Following the workflow proposed, it is clear that the big effort resides in describing the application using the Dataflow MoC (point 3 of Fig.5) and in the High-Level Synthesis (HLS) description of the hardware accelerators. All the other steps are fully-automated [28] and well documented by online available tutorials. Also, the workflow is a valid demonstrator of the different tools developed within the CERBERO H2020 project. The entire chain is applied to design and prototype a live demo where we successfully use the DLS IK algorithm to control a robotic arm directly connected to the MPSoC.

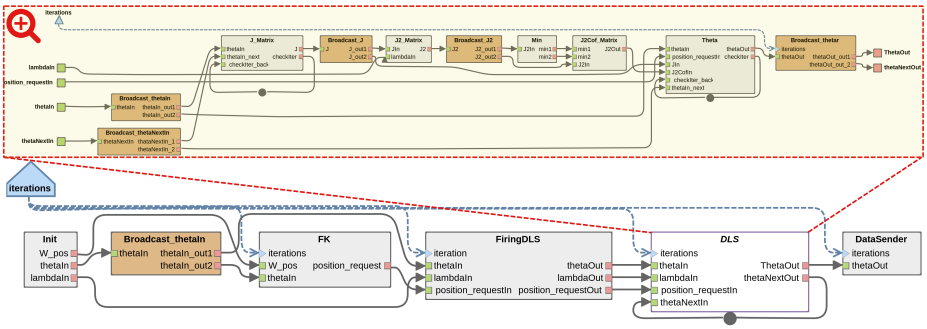


Fig. 6: Hierarchical PiSDF description the DLS in PREESM.

6 DLS algorithm assessment

The DLS algorithm has been implemented in MATLAB® environment for preliminary simulation tests prior to the porting onto the ZedBoard. The tests carried out have evaluated the percentage error in the end-effector position and the execution time while varying the number of iterations and λ, as reported in

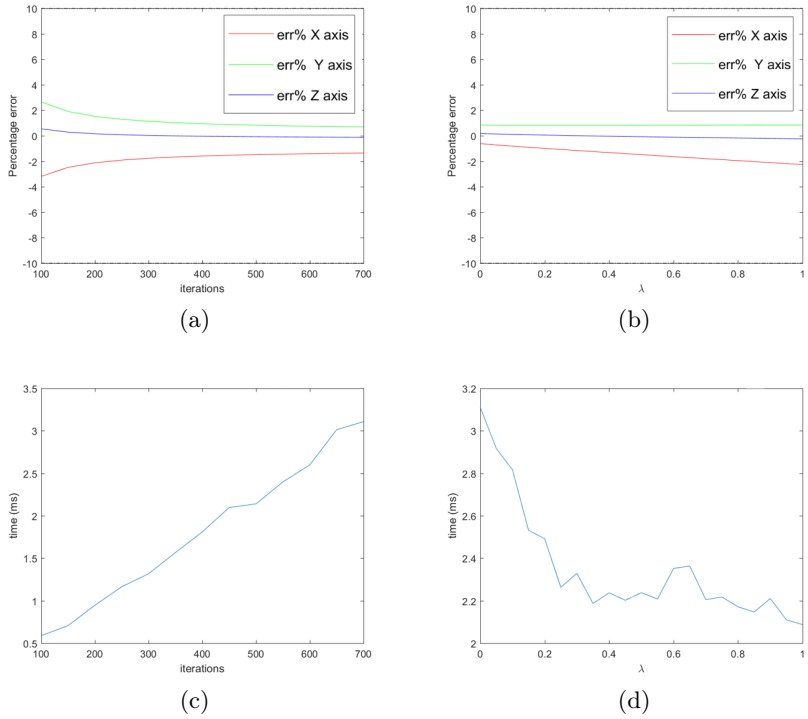
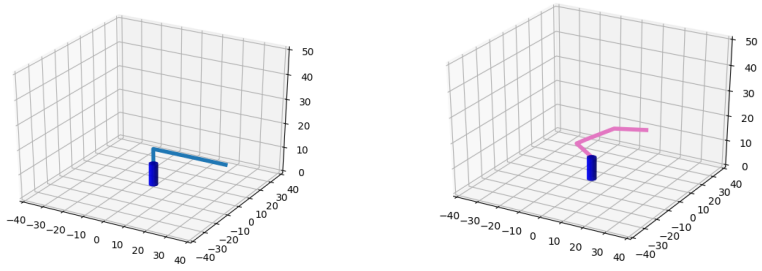


Fig. 7: (a) Percentage error vs. iterations (λ=0.5), (b) Percentage error vs. λ (iterations=500), (c) Execution time vs. iterations (λ=0.5), (d) Execution time vs. λ (iterations=500). Target = $[10\ 10\ 10]$ has been considered for all plots.

Figure 7. The percentage error is substantially constant for a number of iterations greater than 400 and for any λ in the interval $(0, 1]$, while the behaviour of the execution time is monotonic crescent for an increasing number of iterations or with a λ in proximity to zero. Increasing λ, the execution time is reduced on average. By the observations above, λ can be chosen as a static factor in the interval $[0.4, 1]$. A python simulator has been also used prior to the execution on the manipulator. Given the set of calculated angles per iteration, the simulator produces an animated 3-D model of the manipulator, as in Figure 8.

The execution times related to the actors of the PiSDF description have been also evaluated. In Table 2 are reported the values with respect to the graph firing. These take into account the repetitions of the DLS subgraph within an only one PiSDF execution. Indeed, the number of steps to obtain the trajectory (the input parameter *iterations*) leads to a significant timing increment of the DLS sub-blocks. For this reason, a hardware acceleration can be considered in order to reduce processing time and power consumption.



(a) (b)

Fig. 8: Simulated (a) initial and (b) final manipulator poses.

Table 2: Evaluation of the execution times on Zedboard ARM cores with iterations = 100. Values are reported in micro seconds, clock cycles (with respect to the 650-MHz chip frequency) and percentage with respect to the sum of the actor execution times.

Actor	Repeats [unit]	Execution Time per Graph Firing [μs]	[x10^9 cc]	[%]
Init	1	2	1.30	0.10
FK	1	22	14.30	1.11
FiringDLS	1	9	5.85	0.45
DLS: J_Matrix	iterations	700	455.00	35.29
DLS: J2_Matrix	iterations	400	260.00	20.16
DLS: Min	iterations	200	130.00	10.08
DLS: J2Cof_Matrix	iterations	300	195.00	15.12
DLS: Theta	iterations	400	260.00	20.16
DataSender	1	149	96.90	7.51

7 Conclusions and Next Steps

The work presented in this paper is part of the CERBERO H2020 project assessment. This implementation is intended to be used to prospectively demonstrate the benefits of using heterogeneous MPSoC technologies in the field of space applications. The current baseline demonstrator solves computational intensive and iterative Inverse Kinematics algorithms to determine the trajectory of a robotic arm. Prospectively this baseline setup will be extended to show how robustness to faults and different trade-off executions can be successfully addressed using reconfigurable technologies and heterogeneity. As already mentioned, as a natural evolution of the work, specific hardware accelerators are going to be created with a special focus on performance improvements and fault tolerance. The challenge will be coped by making use of the new technologies developed within CERBERO as discussed in Section 5.

Acknowledgement

This work has received funding from the EU Commissions H2020 Program under grant agreement No 732105. The authors would also like to thank the Universidad Politécnica de Madrid for the Leonardo Suriano's predoctoral contract under RR01/2015 (Programa Propio).

References

1. https://www.mathworks.com/products/matlab-coder.html
2. https://preesm.github.io/docs/workflowtasksref/
3. https://preesm.github.io/tutos/sdsoc/
4. http://sites.unica.it/rpct/
5. https://github.com/Papify
6. https://icl.utk.edu/papi/
7. *Cross-layer modEl-based fRamework for multi-oBjective dEsign of Reconfigurable systems in unceRtain hybRid envirOnments*, http://www.cerbero-h2020.eu/
8. Aristidou, A., Lasenby, J., Chrysanthou, Y., Shamir, A.: Inverse Kinematics Techniques in Computer Graphics: A Survey. Computer Graphics Forum (2018)
9. Buss, S.R.: Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods (2009)
10. Buss, S.R., Kim, J.S.: Selectively damped least squares for inverse kinematics. Journal of Graphics Tools **10**, 37–49 (2004)
11. Desnos, K., Pelcat, M., Nezan, J.F., Aridhi, S.: Memory analysis and optimized allocation of dataflow applications on shared-memory mpsocs. Journal of Signal Processing Systems **80**(1), 19–37 (2015)
12. Desnos, K., Pelcat, M., Nezan, J.F., Bhattacharyya, S.S., Aridhi, S.: Pimm: Parameterized and interfaced dataflow meta-model for mpsocs runtime reconfiguration. In: Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII), 2013 International Conference on. pp. 41–48. IEEE (2013)
13. Di Vito, D., Natale, C., Antonelli, G.: A comparison of damped least squares algorithms for inverse kinematics of robot manipulators. IFAC **50**(1), 6869–6874 (2017)
14. Fanni, T., Rodríguez, A., Sau, C., Suriano, L., Palumbo, F., Raffo, L., de la Torre, E.: Multi-grain reconfiguration for advanced adaptivity in cyber-physical systems. In: 2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig). pp. 1–8. IEEE (2018)
15. Fanni, T., Sau, C., Meloni, P., Raffo, L., Palumbo, F.: Power and clock gating modelling in coarse grained reconfigurable systems. In: Proceedings of the ACM International Conference on Computing Frontiers, CF'16. pp. 384–391 (2016)
16. Fanni, T., Sau, C., Raffo, L., Palumbo, F.: Automated Power Gating Methodology for Dataflow-based Reconfigurable Systems. In: Proceedings of the 12th ACM International Conference on Computing Frontiers, CF'15. pp. 61:1–61:6 (2015)
17. Lewis, F., Dawson, D., Abdallah, C.: Robot Manipulator Control: Theory and Practice. 2 edn. (2003)
18. Lombardo, M., et al.: Power management techniques in an FPGA-based WSN node for high performance applications. In: Int. Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC). pp. 1–8 (July 2012)

19. Madroñal, D., Fanni, T.: Run-time performance monitoring of hardware accelerators: Poster. In: Proceedings of the 16th ACM International Conference on Computing Frontiers. pp. 289–291. ACM (2019)
20. Madroñal, D., Morvan, A., Lazcano, R., Salvador, R., Desnos, K., Juárez, E., Sanz, C.: Automatic instrumentation of dataflow applications using papi. In: Proceedings of the 15th ACM International Conference on Computing Frontiers. pp. 232–235. ACM (2018)
21. Masin, M., Palumbo, F., Myrhaug, H., de Oliveira Filho, J.A., Pastena, M., Pelcat, M., Raffo, L., Regazzoni, F., Sanchez, A.A., Toffetti, A., de la Torre, E., Zedda, K.: Cross-layer design of reconfigurable cyber-physical systems. In: Design, Automation & Test in Europe Conference & Exhibition. pp. 740–745 (2017)
22. Palumbo, F., Fanni, T., Sau, C., Pulina, L., Raffo, L., Masin, M., Shindin, E., de Rojas, P.S., Desnos, K., Pelcat, M., Rodriguez, A., Juárez, E., Regazzoni, F., Meloni, G., Zedda, K., Myrhaug, H., Kaliciak, L., Andriaanse, J., de Olivieria Filho, J., Muñoz, P., Toffetti, A.: CERBERO: cross-layer model-based framework for multi-objective design of reconfigurable systems in uncertain hybrid environments. In: Proceedings of the 16th ACM International Conference on Computing Frontiers. pp. 320–325 (2019)
23. Pelcat, M., Desnos, K., Heulot, J., Guy, C., Nezan, J.F., Aridhi, S.: Preesm: A dataflow-based rapid prototyping framework for simplifying multicore dsp programming. In: Education and Research Conference (EDERC), 2014 6th European Embedded Design in. pp. 36–40 (Sept 2014)
24. Pelcat, M., Nezan, J.F., Piat, J., Croizer, J., Aridhi, S.: A system-level architecture model for rapid prototyping of heterogeneous multicore embedded systems (2009)
25. Rodríguez, A., Valverde, J., Portilla, J., Otero, A., Riesgo, T., De la Torre, E.: Fpga-based high-performance embedded systems for adaptive edge computing in cyber-physical systems: The artico3 framework. Sensors **18**(6) (2018)
26. Spong, M., Hutchinson, S., Vidyasagar, M.: Robot Dynamics and Control. Wiley, 2 edn. (2004)
27. Sugihara, T.: Solvability-unconcerned inverse kinematics by the levenberg–marquardt method. IEEE Transactions on Robotics **27**(5), 984–991 (2011)
28. Suriano, L., Arrestier, F., Rodriguez, A., Heulot, J., Desnos, K., Pelcat, M., de la Torre, E.: Damhse: Programming heterogeneous mpsocs with hardware acceleration using dataflow-based design space exploration and automated rapid prototyping. Microprocessors and Microsystems (2019)
29. Suriano, L., Madroñal, D., Rodríguez, A., Juárez, E., Sanz, C., de la Torre, E.: A unified hardware/software monitoring method for reconfigurable computing architectures using papi. In: 2018 13th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC). pp. 1–8. IEEE (2018)
30. Suriano, L., Rodriguez, A., Desnos, K., Pelcat, M., de la Torre, E.: Analysis of a heterogeneous multi-core, multi-hw-accelerator-based system designed using preesm and sdsoc. In: 2017 12th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC). pp. 1–7 (2017)
31. Wijtvliet, M., Waeijen, L., Corporaal, H.: Coarse grained reconfigurable architectures in the past 25 years: Overview and classification. In: 2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS). pp. 235–244 (July 2016)
32. Wright, M.H., et al.: Nelder, mead, and the other simplex method. Documenta Mathematica **7**, 271–276 (2010)