RUN-TIME RECONFIGURATION FOR EFFICIENT TRACKING OF IMPLANTED MAGNETS WITH A MYOKINETIC CONTROL INTERFACE APPLIED TO ROBOTIC HANDS

SERGIO ANDRES PERTUZ MENDEZ

TESE DE DOUTORADO EM SISTEMAS MECATRÔNICOS

DEPARTAMENTO DE ENGENHARIA MECÁNICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA MECÁNICA

RUN-TIME RECONFIGURATION FOR EFFICIENT TRACKING OF IMPLANTED MAGNETS WITH A MYOKINETIC CONTROL INTERFACE APPLIED TO ROBOTIC HANDS

SERGIO ANDRES PERTUZ MENDEZ

ORIENTADOR: PROF. DR. DANIEL MAURICIO MUÑOZ ARBOLEDA
COORIENTADOR: PROF. DR. CARLOS HUMBERTO LLANOS QUINTERO

TESE DE DOUTORADO EM SISTEMAS
MECATRÔNICOS

PUBLICAÇÃO: PPGEA.TD-001/11
BRASÍLIA/DF: ABRIL - 2021

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA MECÁNICA

RUN-TIME RECONFIGURATION FOR EFFICIENT TRACKING OF IMPLANTED MAGNETS WITH A MYOKINETIC CONTROL INTERFACE APPLIED TO ROBOTIC HANDS

SERGIO ANDRES PERTUZ MENDEZ

TESE DE DOUTORADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA MECÁNICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR.

APROVADA POR:

____

Prof. Dr. Daniel Mauricio Muñoz Arboleda – FGA/Universidade de Brasília
Orientador

____

Prof. Dr. Díbio Leandro Borges – CIC/Universidade de Brasília
Membro Interno PPMEC

____

Prof. Dr. Cristiano Jacques Miosso Rodrigues Mendes – FGA/Universidade de Brasília
Membro Interno UnB

____

PhD. Raúl Marín Prades – Dep. Ciencia dels Computadors/Universitat Jaume I
Membro Externo

BRASÍLIA, 5 DE ABRIL DE 2021.

FICHA CATALOGRÁFICA

PERTUZ MENDEZ, SERGIO ANDRES	
Run-time Reconfiguration for Efficient Tracking of Implanted Magnets with a Myokinetic Control Interface Applied to Robotic Hands [Distrito Federal] 2021.	
xiv, 123p., 210 x 297 mm (ENM/FT/UnB, Doutor, Sistemas Mecatrônicos, 2021).	
Tese de doutorado – Universidade de Brasília, Faculdade de Tecnologia.	
Departamento de Engenharia Mecánica	
1. RBFNN	2. Reconfiguração parcial dinâmica
3. FPGA	4. Interface de controle myokinetica
I. ENM/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

PERTUZ, S. (2021). Run-time Reconfiguration for Efficient Tracking of Implanted Magnets with a Myokinetic Control Interface Applied to Robotic Hands . Tese de doutorado em Sistemas Mecatrônicos, Publicação PPGEA.TD-001/11, Departamento de Engenharia Mecánica, Universidade de Brasília, Brasília, DF, 123p.

CESSÃO DE DIREITOS

AUTOR: Sergio Andres Pertuz Mendez
TÍTULO: Run-time Reconfiguration for Efficient Tracking of Implanted Magnets with a Myokinetic Control Interface Applied to Robotic Hands .
GRAU: Doutor ANO: 2021

Sergio Andres Pertuz Mendez

Departamento de Engenharia Mecánica (ENM) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

A mis padres: Belmer y Lucila

ACKNOWLEDGMENTS

Quiero agradecer primero a mis padres Belmer y Lucila por brindarme siempre su apoyo y amor, que ha sido siempre incondicional a pesar de la distancia. También quiero agradecer a mis hermanos Belmer, Luciani y Samanta que han sido un importante apoyo moral en mi vida.
También quiero agradecer a Suamy Cepeda, mi compañera, que siempre estuvo ahí para decirme que podría alcanzar mis metas si así me lo proponía.
Quiero agradecer al Prof. Daniel Muñoz, mi orientador, amigo y prontamente colega a quien admiro mucho y cuyo apoyo y cariño ha sido muy especial para mi por todos estos años de Doctorado y Maestría.
De la misma manera quiero hacer una mención honorífica al Prof. Carlos Llanos que también ha sido una parte importante en este trayecto de mi vida.
Por ultimo pero no menos importante, le envió un agradecimiento especial a los Prof. Helon Ayala y Christian Cipriani por permitirme hacer parte de este proyecto MYKI.
Finalmente, agradezco a la CAPES y FAP/DF por financiar mi proyecto de doctorado por estos años.

RESUMO

Título: Run-time Reconfiguration for Efficient Tracking of Implanted Magnets with a Myokinetic Control Interface applied to robotic hands
Autor: Sergio Andres Pertuz Mendez
Orientador: Prof. Dr. Daniel Mauricio Muñoz Arboleda
Coorientador: Prof. Dr. Carlos Humberto Llanos Quintero
Programa de Pós-Graduação em Sistemas Mecatrônicos
Brasília, 5 de abril de 2021

Este trabalho introduz a aplicação de soluções de aprendizagem de máquinas visado ao problema do rastreamento de posição do antebraço baseado em sensores magnéticos. Especificamente, emprega-se uma estratégia baseada em dados para criar modelos matemáticos que possam traduzir as informações magnéticas medidas em entradas utilizáveis para dispositivos protéticos. Estes modelos são implementados em FPGAs usando operadores customizados de ponto flutuante para otimizar o consumo de hardware e energia, que são importantes em dispositivos embarcados. A arquitetura de hardware é proposta para ser implementada como um sistema com reconfiguração dinâmica parcial, reduzindo potencialmente a utilização de recursos e o consumo de energia da FPGA. A estratégia de dados proposta e sua implementação de hardware pode alcançar uma latência na ordem de microssegundos e baixo consumo de energia, o que encoraja mais pesquisas para melhorar os métodos aqui desenvolvidos para outras aplicações.

Palavras-chave: RBFNN, Reconfiguração parcial dinâmica, FPGA, Interface de controle myokinetica.

ABSTRACT

Title:
Author: Sergio Andres Pertuz Mendez
Supervisor: Prof. Dr. Daniel Mauricio Muñoz Arboleda
Co-Supervisor: Prof. Dr. Carlos Humberto Llanos Quintero
Graduate Program in Mechatronic Systems
Brasília, April 5th, 2021

This work introduces the application of embedded machine learning solutions for the problem of magnetic sensors-based limb tracking. Namely, we employ a data-driven strategy to create mathematical models that can translate the magnetic information measured to usable inputs for prosthetic devices. These models are implemented in FPGAs using customized floating-point operations to optimize hardware and energy consumption, which are important in wearable devices. The hardware architecture is proposed to be implemented as a dynamically partial reconfigured system, potentially reducing resource utilization and power consumption of the FPGA. The proposed data-driven strategy and its hardware implementation can achieve a latency in the order of microseconds and low energy consumption, which encourages further research on improving the methods herein devised for other applications.

Keywords: RBFNN, Dynamic partial reconfiguration, FPGA, Myokinetic control interface.

SUMMARY

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 CONTEXTUALIZATION

Upper limb amputation deprives individuals of their innate ability to manipulate objects. Prosthetic devices and control-strategies are among the primary goals in rehabilitation engineering, whose main goal is to restore dexterous motor functions after amputation. However, the quest for a human-machine interface (HMI) that allows for arbitrary and physiologically appropriate control over multiple degrees of freedom is still far from being completed. Besides commercial solutions that exploit external EMG signals to control the prosthesis [22], researchers are investigating alternative approaches that take advantage of different biological sources. For instance, solutions that exploit implanted electrodes have been proposed to record muscle activity, such as intramuscular [96] and epymisial electrodes [68], and neural electrodes to record peripheral information [62].

An alternative solution recently introduced by Tarantino et al. [88] exploits magnet tracking for controlling a prosthesis. The authors proposed a new HMI dubbed the myokinetic control interface. This interface derives information about muscle contractions from permanent magnets implanted into the amputee's forearm muscles. Indeed, localizing the magnets' position is equivalent to measuring the contraction/elongation of the muscle. This information can be used to interpret the voluntary movement of the subject. In [88], magnets were analytically modeled as point dipoles, and localization was obtained through the Levenberg–Marquardt (LM) optimization algorithm. The output of the optimization problem is a functioning solution to the inverse problem of magnetostatics. The LM algorithm does not provide a fixed time-execution, making the solution unreliable considering real-time constraints.

In this context, a vital research venue is to create efficient algorithms to estimate the magnets' position. The faster the estimation algorithm can provide an output, the more fine-grained control can be achieved for the prosthetic device. The design of a position transducer based on the magnetic sensor information should consider the compromise of precision, accuracy, and execution time. With more time-dense inputs, the control algorithm will have more fine-grained information to perform trajectory tracking, making it easier to represent human-like behavior. In this view, the latency of the algorithm (i.e., the time needed for localizing the magnets once the measurements are available) has to be short enough to ensure real-time tracking. In [91], the authors showed that the latency of algorithms that exploit an analytic representation of the magnets could be reduced by computing the analytic gradient of such representation. In [88], the tracking algorithm could not provide estimations as fast as the sensor could provide information. More specifically, while sensors could provide $\sim$75 samples per second (one sample every 13 ms), 45 ms were needed for localizing four magnets. In a more recent study [17], a fully embedded system was presented, which proved capable of tracking up to five magnets in less than $\sim$4 ms using 32 magnetic field sensors. In this case, the time needed for sampling the sensors readings (one sample every $\sim$21 ms) and sending them to the computation unit ($\sim$24 ms) represented the actual bottleneck of the system. Thus, the system architecture proposed in [17] could be optimized in terms of acquisition and data-transfer times to boost the system performance.

To this end, this work proposes the use of artificial neural networks to perform accurate estimations of individual muscle contractions based on offline collected measurements. According to their universal approximation property [70], and inherent parallel structure [14], it is hypothesized that artificial neural networks can lead to a better compromise between accuracy, computational implementation, and energy consumption for the tracking of several magnets.

Machine learning models have been fruitfully applied to various problems involving sensing, control, and estimation in biomedical applications, including interpreting biological signals to generate actionable commands. In this context, cases in which real-time response is required and solved by hardware implementations often involve such data-driven methods. In the scope of pneumatic muscles, ANNs have been implemented to dynamic modeling in [86] while in [92] a nonlinear proportional-integral-derivative ANN controller has been devised. ANNs have also been used for controlling wearable exoskeletons, in [98] to improve the torque estimation required by the apparatus and in [99] to perform adaptive control concerning parametric uncertainties and unknown disturbances, and in [8] for controlling grasp and lift of a cable-driven soft hand. In the case of prosthetic hands, in [87], the authors investigate the use of convolutional ANNs embedded in microcontrollers to classify hand gestures by interpreting EMG signals. In [20], the authors employ a dataset aggregation strategy for creating deep ANNs for processing EMG data to perform prosthetic limb control. As can be seen from this recent literature review, machine learning has shown great applicability for creating models that interpret complex biosignals. Moreover, such mathematical abstractions favor efficient hardware implementations due to their structure, composed of simple entities, such as neurons or support vectors.

The neural networks' inherent parallel architecture can be exploited with Field Programmable Gate Arrays (FPGAs). These devices are flexible logic structures that allow data processing with high-speed performance in parallel on a single device. A recent review on the topic of ANNs embedded solutions on FPGAs and their advantages to other heterogeneous computing platforms is given in [33]. Recently, FPGA solutions of deep artificial neural networks have also been studied for both training and inference stages. In [83], the authors report the most important features and advantages of hardware implementations for deep ANNs, namely lower power consumption and inherent reconfigurability. In [35], several activation functions are mapped and improve software, and GPU-based deployments, similar to [40], mainly due to its parallel dataflow characteristic. Hardware implementation of ANN for sensor-driven position estimation, such as the proposal of this work, has been previously used in [11, 69]. An interesting type of ANN is the Radial Basis Function Neural Network (RBFNN), which has an inherently parallel architecture and is usually implemented using Gaussian activation functions. Most recent works that focus on implementing RBFNNs on hardware can be found in [2, 64] and references therein [25, 15, 23, 50, 105]. In [2] new architectures for RBFNN were proposed with customized floating-point precision enabling hardware and energy consumption optimization. RBFNNs are relatively easy to design and training, besides dealing with linear and nonlinear problems relatively well. Furthermore, it has a solid tolerance for input noise, and even when the problem's complexity is significant [104].

On the other hand, FPGAs for robotic hand control have already been used for implementing algorithms with parallel motion control of fingers for piano playing [55]. Another example [37], asserts the importance of the usage of parallelism for tactile sensing in robotic hands application. Most recently [71], it was implemented a parallel System-on-Chip (SoC) approach that carries out, in parallel, the data acquisition and control of a robotic hand's multiple Degrees of Freedom (DoFs) developed at the University of Brasilia. The latter work compares with other software-based platforms to test its computational and energy performance with outstanding results.

Nonetheless, FPGAs have some drawbacks like hardware resource utilization and power consumption. The former is an issue considering that the reconfigurable hardware inside an FPGA is not limitless. Not every architecture will fit into a specific device. So, if the solution to that problem is to get a bigger FPGA, the latter drawback comes into play, and as a consequence, a larger device or architecture will draw more power. Several solutions might come into mind for these problems, like architecture optimization or better resource utilization; however, there are limitations regarding the available resources and the real advantages of optimizing the architecture. To meet these strict requirements of embedded applications in terms of performance, power consumption, and physical dimensions, one novel solution that gets beyond the previously mentioned is the implementation of dynamically partially reconfigurable (DPR) systems.

Most FPGAs devices are hardware devices that have to be reconfigured when powered-up; others have non-volatile Flash memory that can configure the FPGA on start-up. For either case, this is called static reconfiguration. However, some FPGAs allow run-time reconfiguration, often referred to as dynamic reconfiguration, which can be divided into two categories: full and partial. The first one considers dynamic reconfiguration of all the hardware, where previous features are erased and reconfigured. On the other hand, the second case is a technique that allows the device's partial reconfiguration, leaving the remainder of the circuit intact and even able to continue operation while the other part is reconfiguring itself [79].

DPR can potentially be used for SoCs-based solution architectures, like the one proposed in this work, due to the combination of HW/SW co-design flexibility and performance. This is because the partial reconfiguration (PR) technology currently allows the software to trigger reconfiguration directly.

1.2 PROBLEM DEFINITION AND MOTIVATIONS

Human hands have an estimate of 27 degrees of freedom (DoF). Robotic hands should ideally emulate all of that; however, this achievement has proven very difficult due to space, energy, and other physical restrictions. A solution to this problem have involved reducing the number of fingers [54] or reduce the number of DoF [18, 26, 89, 61, 45, 9]. Whatever the case, most robotic hands and prostheses have more than two DoF [90].

The LEIA robotic hand [71] has a total of 7 DoF. Once it is decided that each DoF should be controlled by a different muscle in the hand/wrist, the proposed solution's computational complexity will likely require more computational effort, making it harder to maintain determinism with a reasonable output rate.

In this context, there is a need for implementing computationally efficient inversion methods to obtain accurate estimations of individual muscle contractions based on the myokinetic proposed sensing devices [88]. This work proposes using artificial neural networks to estimate individual muscle contractions based on offline collected measurements accurately. Also, for developing a fast-enough embedded system for the localization of the muscle movements and the control of the robotic hand, using a single chip, the development of a DPR system might show some advantages over classical approaches.

Nevertheless, the development of DPR systems is currently not an easy task. Even with the advantages DPR systems have over those with static logic, their complexity is a significant drawback in their implementation. The recurrent issue designers must deal with when projecting DPR systems is the difficulty to assert its behavioral characteristics in the application domain, such as the reconfiguration time, performance, and its impact on the device as a whole.

On the other hand, even when the FPGA manufacturers fully support the feature, i.e., Xilinx Inc., synthesis and implementation tools such as Vivado heavily rely on graphical programming design script-based approach to build them. Few attempts have been carried out to consolidate a high-level tool that aides the design of SoC-based DPR systems. Usually, works aimed at dynamic reconfiguration applications have little to zero focus on the design tools and methodologies used during the design process.

Recently, a cooperation between the University of Brasilia (Brasilia, Brazil), the KTH Royal Institute of Technology (Stockholm, Sweden), and Saab ab (Linköping, Sweden) have boarded this issue. Initially, the project started as a design tool for fault-tolerant methodologies in autonomous aircraft applications, eventually evolving as a high-level modeling tool for DPR SoCs, named after RTRLib [43].

In this context, the present work aims at the promotion of three issues: (1) The development of computationally efficient AI models capable to accurately translate individual muscle contractions from the myokinetic interface developed by the MYKI project [88]; (2) To propose an SoC FPGA-based controller able to both collect the forearm's sensory data and control a robotic hand, using Xilinx's DPR features to reduce the area and power consumption of the controller; and (3) To improve the RTRLib tool used to perform DPR systems, break out from some of its current limitations, presented further ahead, to later be used for the last goal.

1.3 OBJECTIVES

This work's general objective is to develop an FPGA-based SoC for multi-magnet tracking of a myokinetic interface with adaptive capabilities using dynamic partial reconfiguration. The developed models will translate the myokinetic interface data to proper setpoints for driving a prosthetic robotic hand.

1.3.1 Specific Objectives

The following are the specific Objectives:

- To build data-driven soft-sensors, using black-box system-identification machine learning models, able to translate the myokinetic sensor information to identify the voluntary motor and force commands on a forearm muscle.
- To improve the functionalities of the RTRLib, aiming to obtain a high-level design tool that, under specific conditions, aids the design flow of dynamically partially reconfigurable systems.
- To design a novel DPR hardware implementation to pursue cheaper and more energy-efficient myokinetic transducers used in prosthetic hand control.

1.4 CONTRIBUTIONS

There are three types of contributions expected from this research work: (i) Technological, (ii) Scientific, and (iii) Academic contributions. They are listed below:

- Scientific:
  - Implementation of novel black-box system-identification machine learning models, such as neural networks, solve the inverse magnetism problem of the myokinetic interface. The benefits of this are its computational efficiency, accuracy and that the system's little knowledge to implement them allows for a more flexible transducer of the system. Thus facilitating its real implementation on different patients.
- Technological:
  - Improvement of the RTRLib's current limitations, such as floorplanning, Linux applications, decoupling, computation of the reconfiguration time, and integration with the RTRLib design flow.
  - To upgrade the robotic hand developed in the LEIA lab for compatibility with the MYKI data, improving its dexterity.
- Academic:
  - To begin a collaboration between the Biorobotics Institute at the Sant'Anna School of Advanced Studies about the MYKI Project.
  - Continuation of the collaboration between the University of Brasilia (UnB), Royal Institute of Technology (KTH), and the Saab ab company.
  - Collaboration through a signed agreement with the University of Pamplona, Colombia, promoting research internships at the University of Brasilia.
- Journal Articles
  - A Parallel System-on-Chip Approach for Impedance Controller for a 7-DoF Robotic Hand. Sergio A. Pertuz, Carlos Llanos, Cesar A. Peña, Daniel Muñoz. ANALOG INTEGRATED CIRCUITS AND SIGNAL PROCESSING. Qualis B1. Submitted: October 2019. Status: second-round review completed.
  - Development of a robotic hand using bioinspired optimization for mechanical and control design: UnB-Hand. Sergio A. Pertuz, Carlos Llanos, Daniel Muñoz. IEEE Access. Qualis A2. Accepted: March 2020. Status: Accepted.

   - Efficient Data-driven Real-time Magnetic Tracking for Myokinetic Control Interfaces. Sergio Pertuz, Marta Gherardini, Gabriel Vidigal de Paula Santos, Daniel Muñoz, Helon Vicente Hultmann Ayala, Christian Cipriani. MECHATRONICS (OXFORD). Qualis A1.

- Conference Papers
   - A Modular and Distributed Impedance Control Architecture on a Chip for a Robotic Hand. Sergio A. Pertuz, Carlos Llanos, Cesar A. Peña, Daniel Muñoz. 2018 31st Symposium on Integrated Circuits and Systems Design (SBCCI). Status: Published.
   - Simulation and Implementation of Impedance Control in a Robotic Hand. Sergio A. Pertuz, Carlos H. llanos, Daniel M. Muñoz. 24th ABCM International Congress of Mechanical Engineering (COBEM 2017). Status: Published.
   - Run-time Reconfiguration for EfficientTracking of Implanted Magnets with a Myokinetic Control Interface. Sergio A. Pertuz, Daniel M. Muñoz. International Conference on Field-Programmable Logic and Applications (FPL 2021). Status Submitted.

1.5 TEXT STRUCTURE

The rest of this document is constructed as follows: Chapter 2 comprises the theoretical foundation that starts from the definition of aspects of reconfigurable systems, presenting relevant concepts of partial reconfiguration, a review of robotic hands and robotic prostheses, and the definition of the model predictors that are going to be used. Chapter 3 does a quick literature review to substantiate the later choices and descriptions for more in-deep contextualization. Chapter 4 presents the methods of this work, including experimental setup at the Sant'Anna Institute and UnB, and a formal presentation of RTRlib is formally filed, describing it through its proposed methodology, limitations, and functionalities. Following is Chapter 5 introduces the results of the hardware implementation of the ML models implemented. Chapter 6 presents the training of the models using five magnets simultaneously, and finally, Chapter 7 presents the DPR system for the MYKI interface and its validation.

2 THEORETICAL FOUNDATION

2.1 REGRESSION USING BLACK-BOX DATA-DRIVEN MODELS

Model identification consists of discovering a good model architecture that may result in a fitting solution that accurately represents a system. Let that model architecture be $f[\cdot]$ as general nonlinear function mappings from the model inputs to the predicted output, denoted by $\hat{x} \in \mathbb{R}^n$ where n is the total number of input values. Fitting a model architecture to a system (parameter estimation) is usually a small difficulty problem, as long as it is a suitable model.

The more complicated issue is deciding what architecture to use. Determining a model architecture is the most critical part and depends on prior knowledge and physical insight into the system. Typically, according to the amount of knowledge previously amassed of a system, three types of model architectures exist and are color-coded [84].

- White-Box models: When the designer fully knows the system's physical behavior, it is possible to model it purely from this prior insight.
- Grey-Box models: When the designer has some physical insight into the model, but the parameters remain unknown, being necessary to determine them by observing the system.
- Black-Box models: When no physical insight is known, the system's measured data's behavior is nonlinear.

This work considers black-box models. Black-box model regression usually consists of obtaining a model architecture's parameters based on measured input and output data. It amounts thus to build flexible function surfaces to fit the acquired data [58]. For this, a system, given in Figure 2.1, can be defined with a set of measured inputs (x) and outputs (y). Where the different inputs measured at a given time instant are organized in columns, e.g. $x \in \mathbb{R}^{n \times m}$, where m is the ammount of measurements. For Black-Box Identification, it is assumed that both these quantities are possible to be measured and are available.

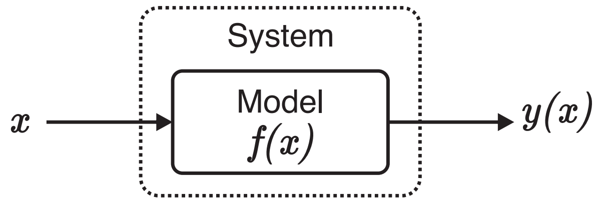


Figure 2.1 – Description of the system, detailing the notation for the input and output: x and $y(x)$, respectively.

The black-box model regression procedure is an iterative and experimental task, and its flowchart is depicted in Fig. 2.2 [67].

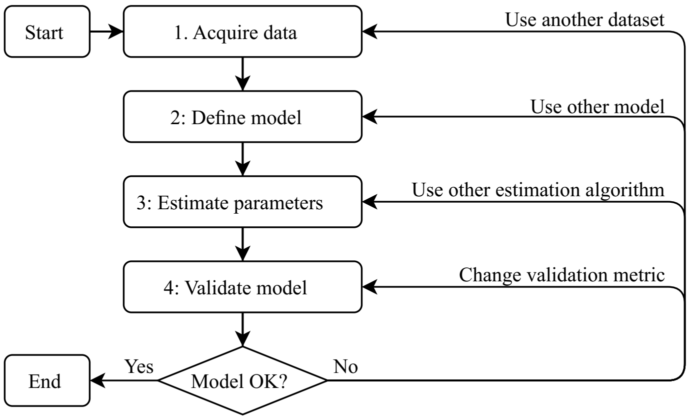


Figure 2.2 – System identification procedure. Adapted from [67]

According to Fig. 2.2:

1. Acquire data: The acquisition of the input/output data is critical given that if the system is excited in a way that non-informative data is generated, the built model will not represent the system adequately. Badly designed experiments lead to bad models. Hence, the designed input must be persistently exciting the system. For linear systems, it is said that the input should excite all the frequencies of interest, while in nonlinear the designed input ought to extend not only the bandwidth but also the full amplitude range of the system. [5].

2. Define model: The first definition is whether to go for linear and nonlinear model architectures. If the model is linear, well-known error prediction identification methods may be employed to estimate the parameters once the order is set. Linear models are not only straightforward to build but also easier to control. However, their accuracy may not be sufficient for a given application, and nonlinear models may be an option if the validation fails for linear models. The parsimony principle [59] states that there is no reason to use nonlinear models if the linear model can perform well. Moreover,

if the settled model is nonlinear, the following step is to set the structure. For this, frequently, the designer does this by trial-and-error.

3. Estimate the model parameters: Machine learning community often refers to this step as training or learning. Additionally, the estimation problem is termed as an optimization procedure to minimize metrics, such as the One-Step-Ahead (OSA) residuals prediction or evolutionary algorithms.

4. Validate the model: Validation is used to check if the model is suitable for its intended use. The most common quantitative way to do this is to analyze the resulting model's residuals and statistical properties. In this step, if the model is not valid, the designer should go backward in the procedure and overhaul one of the previous steps.

In the last step depicted in Figure 2.2, the designer faces whether the built model is going to be used or if it should be necessary to perform again one of the steps presented. In order to evaluate a model, some ways are more or less advisable, depending on the purpose of the model.

The most common measure to evaluate the models is the prediction error (or residuals). It is common to distinguish types of residuals according to the prediction horizon. According to such criterion, ξ_s is defined as the OSA prediction error. The Free-Run simulation (FR) simulation error is calculated according to

$$\xi_s = y(x) - \hat{y}_s(x) \tag{2.1}$$

, where $\hat{y}_s(t)$ is the estimated output of the system considering the autoregressive part of the model's inputs as the corresponding previously predicted values (e.g. $\hat{y}_s(x) = F(\cdot)$). The OSA prediction errors are used in the estimation procedure as an optimization task for convexity and computational complexity reasons. Most supervised estimation algorithms thus use the OSA prediction error to estimate the model's parameters.

Other standard metrics are also based on the residuals used in the present work scope to evaluate the models. The Mean Squared Error (MSE) is defined as

$$MSE = \frac{1}{N}\sum_{t=1}^{N}\left[\xi_s(x)\right]^2 \tag{2.2}$$

, where N is the total number of samples considered. Whereas the MSE is used to measure fit quality, other metrics may be used to evaluate it, not case-dependent. Specifically, metrics that are normalized in some sense provide results that are not dependent on the amplitude of the output. The normalized error, denoted by e, is given as

$$e = \left(\frac{\sum_{t=1}^{N} \xi_s(x)^2}{\sum_{t=1}^{N} y(x)^2} \right)^{1/2} \tag{2.3}$$

, and the multiple correlation coefficient (R^2) defined as

$$R^2 = 1 - \frac{\sum_{t=1}^{N} \left[\xi_s(x)\right]^2}{\sum_{t=1}^{N} \left[y(x) - \bar{y}\right]^2} \tag{2.4}$$

, The upper bar denotes the mean value of the sequence and can also measure the model quality based on the prediction residuals. As mentioned before, the advantage of these metrics is that they are not dependent on the amplitude of the quantity one is measuring. It is important to recall that $R^2 = 1$ means perfect data reconstruction and $R^2 > 0.9$ is considered sufficient for most applications [82].

Typical choices for the mapping function $F[\cdot]$ are higher degree polynomial functions, ANNs, fuzzy systems, splines, etc. As the purpose of the function mapping $F[\cdot]$ is to obtain an accurate approximation of $y(x)$. It is not easy to make a difference between black-box functions as they can approximate functions arbitrarily well. However, the model should represent the dynamics of the system under study and thus should make possible the required analysis. ANNs are an excellent choice in this subject as they are more transparent than other black-box nonlinear mappings, which is the focus of the following section.

2.2 ARTIFICIAL NEURAL NETWORKS - ANN

An important class of nonlinear models is Artificial Neural Networks (ANNs). ANNs are learning architectures interconnected by simpler processing units (neurons) inspired by the human brain's behavior and how it learns by correcting errors. The interconnected neurons are associated synapses with defined weights between its connection [67].

An important novel structure called perceptron was introduced in [78]. After that, in [63] the authors discussed the limitations of the present abstraction proposed for the ANN, following a period of inactivity in the area. It became active again with the results reported in [38], which created the now famously known Hopfield networks. Following that, the well-known backpropagation algorithm was devised, and people found out that the doctoral work of [97] was actually the first to introduce algorithms of this type, of which backpropagation was a particular case [36].

Important ANN examples are the MultiLayer Perceptron (MLP), support vector machines, and Radial Basis Neural Networks (RBFNNs) [36]. Despite their differences, all

ANNs have essentially the same functions and elements: they are formed by simpler elements, the neurons, which build a more complex structure that takes a set of inputs and produce a correspondent set of outputs (approximations) – all ANNs can thus be seen as vector mappings.

In [4] pros and cons of using ANNs for system identification are presented. As advantages, one may say that ANNs are simple structures, conceptually easy to understand and defined with a couple of equations. There are many tools available to perform the estimation of the model-related parameters and have good approximation properties. On this last topic, some ANNs, including the RBFNNs, which will be given in detail next, are proven to be global approximators. This means that they can approximate arbitrarily well any continuous function. In the context of system identification, this means that, after choosing an appropriate set of candidate lags for the input and output, one should define the ANN with a complex enough architecture in order to identify the system at hand. This is one reason why input selection is important. After the successful completion of this step, it is known that it is possible to capture the dynamics of the underlying system provided the input and output data are informative enough; however, if the lags chosen are not correct, then even if the MSE of the model is optimized throughout the learning task – as the ANNs are able to fit continuous functions – there might be still dynamics present in the residuals as the correlations may show. Another advantage is that the field of ANNs is very active, which means that many concepts, theories, and algorithms may be used in the context of system identification (as the universal function approximation property, which was just mentioned).

One important drawback that the user faces when using black-box structures such as ANNs is that the models are not easy to understand and sometimes so complex that it is impossible to write them down, and making them difficult to analyze. In the case of ANNs, one can say that they are good models in the sense that they are able to approximate well, but they are often very complicated to the point that it may be impossible to analyze each term if something is going wrong with the model.

In the following, MLP and RBFNN (of the most widely used ANNs) is formally stated. Details are given about its mathematical formulation, training algorithms, and how it fits in the context of system identification. The latter, is the scope of this work, motivates its implementation in reconfigurable hardware by its relatively simple structure described next.

2.2.1 Multilayer Perceptron - MLP

Figure 2.3 depicts a multilayer perceptron's architecture with two hidden layers and an output layer. The network shown here is fully connected to set the stage for a description of the multilayer perceptron in its general form. This means that a neuron in any network layer is connected to all the neurons (nodes) in the previous layer. Signal flow through the network

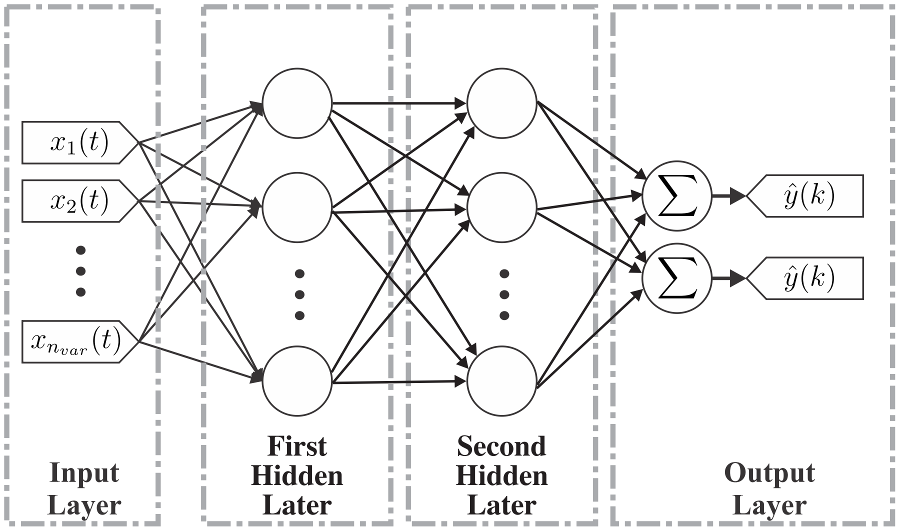


Figure 2.3 – MLP architecture.

progresses in a forward direction, from left to right, and on a layer-by-layer basis.

Such signals are called function signals and are defined as input signal (stimulus) that comes in at the input end of the network, propagates forward (neuron by neuron) through the network, and emerges at the network's output end an output signal. These signals are presumed to perform a useful function at the output of the network. Similarly, for each neuron of the network through which a function signal passes, the signal is calculated as a function of the inputs and associated weights applied to that neuron.

The output neurons constitute the output layer of the network. The remaining neurons constitute hidden layers of the network. The first hidden layer is fed from the input layer made up of sensory units (source nodes); the resulting outputs of the first hidden layer are in turn applied to the next hidden layer, and so on.

Each hidden or output neuron of a multilayer perceptron is designed to perform two computations:

1. the computation of the function signal appearing at the output of each neuron, which is expressed as a continuous nonlinear function of the input signal and synaptic weights associated with that neuron; and

2. the computation of an estimate of the gradient vector (i.e., the gradients of the error surface with respect to the weights connected to the inputs of a neuron), which is needed for the backward pass through the network.

The computation of the neuron's output (δ) for each neuron of the multilayer perceptron requires knowledge of the derivative of the activation function ($\phi(\cdot)$) associated with that neuron. For this derivative to exist, we require the function $\phi(\cdot)$ to be continuous. In basic terms, differentiability is the only requirement that an activation function has to satisfy. An example of a continuously differentiable nonlinear activation function commonly used in multilayer perceptrons is sigmoidal nonlinearity, described as

$$\phi(l) = \frac{1}{1 + e^{-al}}, \quad a > 0 \tag{2.5}$$

, where a is an adjustable positive parameter and l is an input.

A multilayer perceptron may be viewed as a practical vehicle for performing a nonlinear input-output mapping of the function $F[\cdot]$. To be specific, the network's input-output relationship defines a mapping from an N-dimensional Euclidean input space to an M-dimensional Euclidean output space, which is infinitely continuously differentiable when the activation function is likewise.

2.2.2 Radial Basis Artificial Neural Networks - RBFNN

Motivated by their relatively simple architecture and property of universal approximation, RBFNNs have been applied to a wide range of problems like the identification of nonlinear systems and control of dynamical systems. They were originally proposed by Broomhead [6] and have been proven to be global approximators. It means that the RBFNN is able to approximate arbitrarily well any continuous function, given that enough number of neurons are allowed in the architecture. One important feature of RBFNNs is that the weighting coefficients of the output layer are linear-in-the-parameters, what constitutes one advantage of the RBFNN architecture when compared to multilayer perceptrons. This results in faster learning algorithms for those of which make use of this mathematical property. The RBFNN architecture is also simpler when compared to the multilayer perceptrons [36].

RBFNNs comprise three layers: input, hidden, and output layers, see Fig. 2.4. The input layer receives data from the environment and delivers it to the hidden layer without weighing them, i.e., the inputs excite the neurons directly. The neurons are in the hidden layer and are activated by a radial basis function, such as thin-plate-spline, multi quadratic, inverse multi quadratic, and Gaussian functions. In the output layer, each neuron output is weighted and summed. This sum is the final RBFNN approximation (output). Each layer in the RBFNN was designed to have different and specific roles. The input layer, composed of source nodes, makes the connection of the NN with the environment. The inputs are passed directly to the second layer – the hidden layer with the processing units with radial basis functions activation functions. A nonlinear transformation is applied from the input to the hidden space, which is followed by a linear one from the hidden to the output space [36].

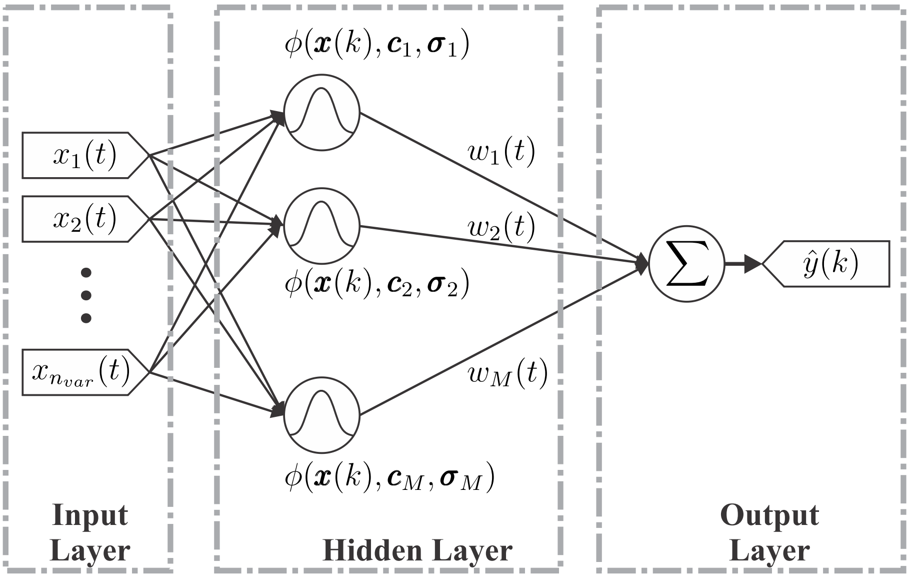


Figure 2.4 – RBFNN architecture.

Equation 2.6 defines an RBFNN mathematically:

$$\hat{y}(k) = F[\boldsymbol{x}(k)] = \sum_{m=1}^{M} w_m \phi(\boldsymbol{x}(k), \boldsymbol{c}_m, \sigma_m), \tag{2.6}$$

where $M \in \mathbb{N}^+$ is the number of neurons in the hidden layer, $\hat{y}(k) \in \mathbb{R}$ and $\boldsymbol{x}(k) \in \mathbb{R}^{n_{var}}$ are respectively the network predicted output and the input vector at a given instant $k \in \mathbb{N}^+$, $\boldsymbol{c}_m \in \mathbb{R}^{n_{var}}$ and $\sigma_m \in \mathbb{R}^+$ are respectively the center and the width (or variance) of the m-th hidden node of the ANN. Each of the output weights is given by $w_m \in \mathbb{R}$

The function $\phi(\cdot)$ is the radial basis activation function previously mentioned. The one that is the most frequently used is the Gaussian RBF, given by

$$\phi(\boldsymbol{x}(k), \boldsymbol{c}_m, \sigma_m) = \exp\left(-\frac{\|\boldsymbol{x}(k) - \boldsymbol{c}_m\|^2}{2\sigma_m^2}\right) = \exp\left(-\frac{1}{2\sigma_m^2}\sum_{i=1}^{n_{var}}(x_i(k) - c_{m,i})^2\right) \tag{2.7}$$

.

For the RBFNN model, the tunable parameters are the number of neurons in the hidden layer, the center's position and widths of the RBFs, and the output weights. The RBFNNs' complexity depends on the number of neurons and the system's number of inputs, which is fixed. However, setting the latter to solve an arbitrary approximation problem, though a fundamental question, is still an unsettled issue. After defining the number of neurons on the hidden layer, the training strategy most widely used is divided into a two-stage procedure: (i) locate the centers through unsupervised learning and (ii) supervised learning to define the output weights.

Other typical activation functions for RBFNN are:

1. Inverse multiquadratic:
$$\phi(l) = \frac{1}{\sqrt{l^2 + \sigma^2}}, \tag{2.8}$$

2. Linear
$$\phi(l) = l, \tag{2.9}$$

3. Cubic
$$\phi(l) = l^3, \tag{2.10}$$

4. Multiquadratic
$$\phi(l) = \sqrt{l^2 + \sigma^2}, \tag{2.11}$$

5. Thin-plate spline
$$\phi(l) = \frac{l^2}{\sigma} \log\left(\frac{l}{\sigma}\right) \tag{2.12}$$

2.3 RECONFIGURABLE HARDWARE AND FPGAS

Reconfigurable hardware mixes programmability after fabrication with the spatial (parallel) computing style of Application Specific Integrated Circuits (ASICs). ASICs are commonly more efficient than the temporal computing style (sequential) of instruction flow processors. Reconfigurable hardware offers an excellent balance between efficiency and flexibility for the implementation of application systems. Nowadays, complex systems often require high flexibility; these systems require devices that can adapt to ever-changing operating environments and conditions of the application they address. On the other hand, these solutions often demand power consumption and performance efficiency. ASICs are not suitable for the type of problem where certain flexibility is required, but reconfigurable hardware presents an exciting option for these cases [60].

Field Programmable Gate Arrays (or FPGA) are reconfigurable hardware devices. Its internal architecture is defined by its granular components, referring to the smallest block the device is built upon; these blocks are called Configurable Logic Blocks (CLB). In devices with fine-grained architectures, the CLBs are composed of memories based on LUTs (Lookup Tables) that implement combinatorial functions and some flip-flops, which can be programmed for simple sequential logic functions. FPGAs contain thousands, and sometimes millions, of these CLBs and are interconnected through a programmable network and can operate at frequencies up to 500 MHz.

Besides the CLBs, an FPGA is also composed of other components (see Fig. 2.5):

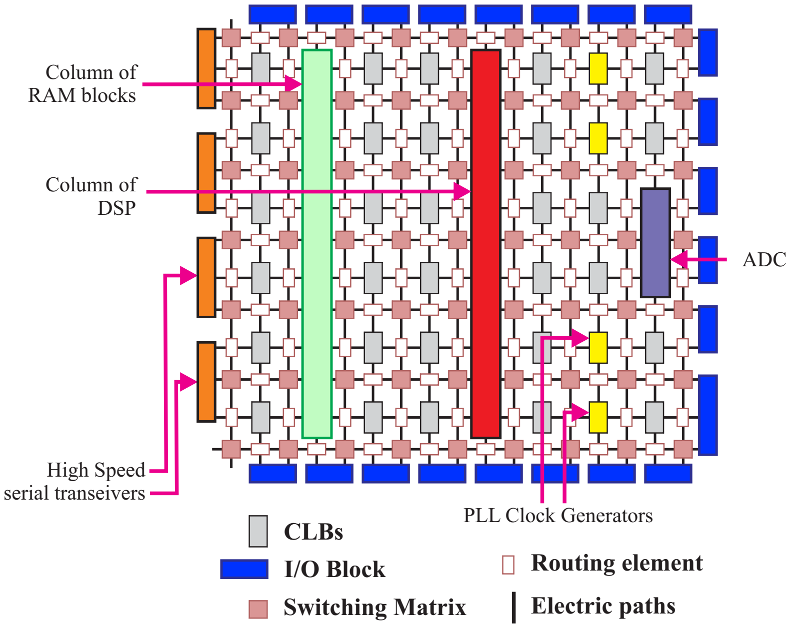


Figure 2.5 – Internal composition of an FPGA

a I/O blocks: Are Registers used to store information of the exterior logic pins.

b Routing element: Are part of the programmable network mentioned before. They are programmable blocks connected to different paths, allowing neighboring CLBs to be connected if desired.

c Switching matrix: They are also part of the programmable network. It provides programmable routing between the routing elements, allowing connections between CLB to CLB, CLB to other elements, and any connection between the programmable logic elements.

d Special circuits: FPGAs can include some ASICs for systems with high complexity, such as DSPs, RAM, PLLs, ADC, and others. Some even include Embedded processors such as the Xilinx 7-Series FPGAs; these devices are often referred to as System on Chip (SoC).

2.3.1 FPGA's Design Flow

When a designer describes a complex behavior or architecture of some sort, the logic synthesizer implements it by connecting these CLBs in a specific way. Fig. 2.6 describes the design-flow of logic systems on an FPGA [102]:

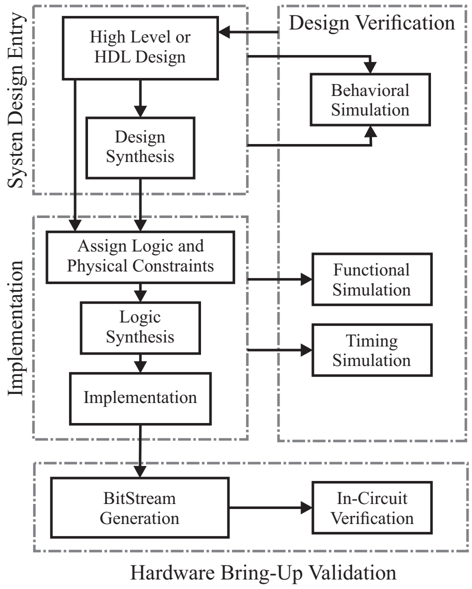


Figure 2.6 – Design Flow of an FPGA's hardware system development (adapted from [102]).

1. System Design Entry: It comprises the high-level (c++, system-c, and others) or low-level (VHDL, Verilog, and others) design description. In this step, the functional characterization of the system considering the area and performance restrictions is developed. After that, the code is synthesized as a netlist containing a Boolean register-based logic and an RTL logic diagram.

2. Implementation: In this phase, the logic and physical constraints, such as I/O ports localization and clock frequencies, are assigned. With that, the circuit netlist is refined, producing a list of logic elements and their connections. Later, the circuit mapping and placement are carried out where the logic functions are mapped and placed into the FPGA's available elements. Finally, the routing of the mapped elements is made, enabling to move to the next phase.

3. Design Verification: This part is shared between the previous ones. In the Behavioral simulation step, the circuit behavior can be analyzed from the hardware description before implementing it. The Timing simulation allows timing analysis to verify and detect timing failures of the circuit being implemented. Finally, the Functional simulation includes the behavioral and timing simulations considering physical constraints of the placed and routed circuit components in the FPGA, ensuring the circuit's operation.

4. Hardware Bring-Up Validation. In this phase, the bitstream generation for in-circuit validation is performed. A bitstream is a file that contains the information extracted from the implementation phase (placement and routing) coded as a bitstream that is understandable by the configuration logic of the FPGA. The bitstream is used to configure the FPGA for further physical validation of the system.

The first three phases are carried out in a synthesis tool, usually provided by the FPGA's manufacturer company. It ordinarily contains a text editor for the hardware description, the synthesis, implementation, bitstream generation tools, and a connection manager to send the bitstream to the FPGA.

2.3.2 Configuration Mechanism of an FPGA

In the last phase, the bitstream file can be delivered to the FPGA by an In-Circuit Verification interface (see Fig. 2.7). An FPGA can be partitioned into the configurable and non-configurable areas. The former is the part described in Fig. 2.5, the latter consist of the configuration interface and configuration logic (it may also include embedded processors in FPGA-based SoCs). In reconfiguration mode, the configuration logic gets the configuration bitstream from the interface circuit and writes it into proper configuration memory locations.

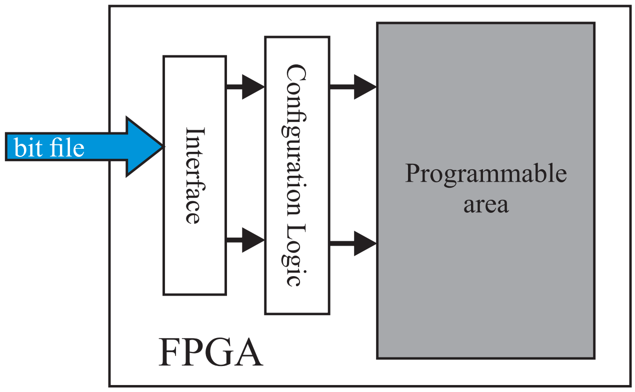


Figure 2.7 – FPGA's reconfiguration scheme.

The memory can be visualized as a rectangular array of bits, and it controls the FPGA's internal connections and the elements' settings.

Depending on the FPGA's manufacturer, different types of configuration interfaces are used for different purposes and characteristics. The following are Xilinx FPGAs available external interfaces (when the reconfiguration is controlled outside the chip) [101]:

- JTAG: Is initially designed for testing purposes. It provides a mechanism to shift testing vectors into the device I/O ports and shift circuit responses from the I/O ports. It also enables Boundary-Scan Chain.

- Select-MAP: or Select Memory Read Procedure, it loads the data at a rate of one or two-byte per clock cycle. It is desirable when configuration speed is a concern. In this interface, an external clock is needed.

- Master/Slave-Serial: Loads data at a rate of one-bit per clock cycle and uses an external clock. Slave mode, additionally, allows daisy-chain configuration.

In Xilinx FPGA-based SoCs, it is also possible to control the reconfiguration process internally. The interfaces to make this possible are:

- ICAP: Or internal configuration access port, is essentially an internal version of the SelectMAP interface.
- PCAP: Or processor configuration access port, is similar to the ICAP and is the primary port used for configuring a Zynq-7000 SoC devices.

2.4 DYNAMIC PARTIAL RECONFIGURATION

Dynamic Partial Reconfiguration (DPR) is a design process that allows a limited predefined logic portion of an FPGA to be reconfigured by downloading partial bit files. At the same time, the remainder of the device continues to operate without interruption. The use of PR can allow designers to:

- A more efficient use of available board space by only loading only the functionality that is needed at any point in time,
- By the reduced logic count, it is possible to move to fewer or smaller devices, potentially cheapening the implementation of a system,
- Similarly, the reduction of a system allows to reduce its power consumption, and
- Because the reconfiguration being "on-the-go" the system upgradability can be improved.

Additionally, DPR is used in applications where:

1. there is a necessity of in-the-field hardware upgrades and updates to remote sites,
2. the runtime reconfiguration is imperative,
3. there is a need for strategies used for fault recovering or fault-tolerant systems,
4. there are adaptive hardware algorithms, and
5. it is required to bring the end-user continuous service.

Fig. 2.8 depicts a DPR application. When implementing DPR systems, the chip layout is partitioned into static logic and reconfigurable partitions (RP). RPs may contain several reconfigurable modules (RM) and communicate with other parts, both static and reconfigurable. Another important characteristic of DPR systems is that the design's static logic can not rely on the state of the RM while reconfiguration is taking place.

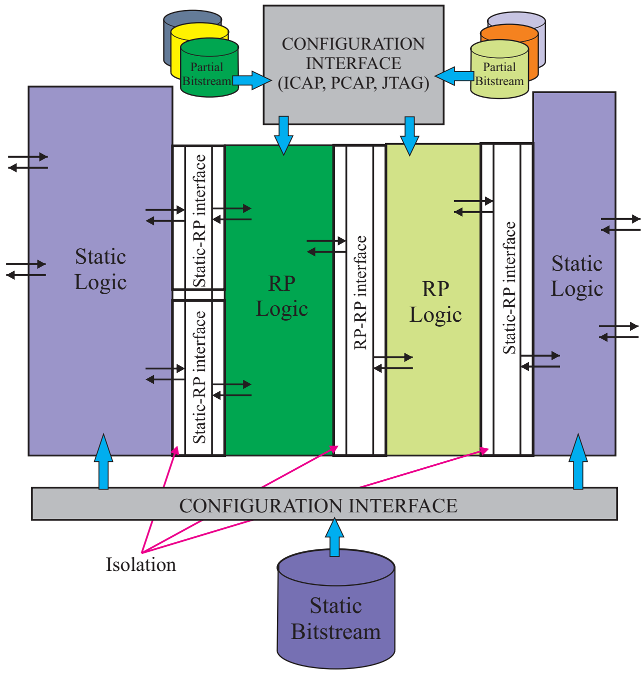


Figure 2.8 – Partial reconfiguration concept description

2.4.1 DPR Design Flow

Producing partial reconfiguration (PR) systems demands a slightly different design flow. Fig 2.9 depicts this new process where, even though it differs from the typical workflow, the phases are the same with extra steps.

1. System Design Entry: Here, the static logic, RPs, and RMs must be defined. Later on, a bottom-up synthesis must be performed, referring to synthesize the different modules separately. In Vivado Design Suite, this is named out-of-context (OOC) synthesis. The generate files ensure that the optimizations occur across the RP boundaries.

2. Implementation: Manages the floorplanning of the RPs, i.e., designate the resources to be reconfigured by selecting the regions that will host the RPs. Thenceforth the placing and routing of the design configurations are performed for every static-RM combination.

3. Hardware Bring-Up Validation: This step validates that the placement and routing results of all the systems, including the static implementation and interfaces to the

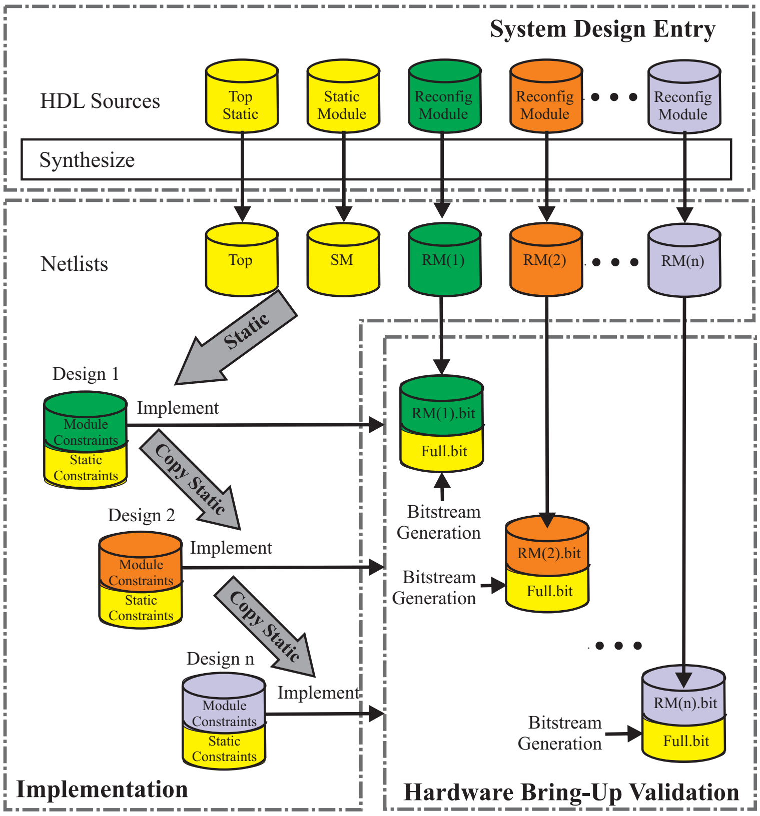


Figure 2.9 – Overview of Xilinx partial reconfiguration design flow for DPR. Adapted from [100]

RP, are consistent across all configurations. Finally, the bitstream files are generated. This phase generates a full bitstream file and partial bitstream files, unlike the standard design flow.

Even though there are many similarities in PR and non-PR system design, there are some premises this type of project requests to work correctly, such as the common logic between all the designs (static-RM combinations) must be identical. Other considerations are formulated in the next subsection.

2.4.2 Design Considerations

Before even evaluating if a PR system will behave correctly, firstly, the designer must meet some premises that constraints its synthesizing feasibility [100], some of them are:

1. Static-RP interfacing compatibility: This constraint outlines that the logical and physical interface of an RP must remain consistent for all the RMs implementation. It is carried out using different partition interfaces:

    - A bus-macro that creates a pair of LUTs and connects through fixed wires. One

LUT is positioned in the static region and the other one in the reconfigurable region. (NOT USED ANYMORE)

- A proxy logic inserts a single LUT element for each partition pin that has to be a fixed interface.
- Partition pins are the logical, physical connection between the RP and the static logic, automatically created for all reconfigurable partition ports. With this option, it is important to pay attention to the consistency of each part's active logic levels.

2. Decoupling: During the PR process, the outputs of the RP interface may be unstable, representing non-valid data and potentially triggering static logic improperly. Consequently, the static logic must ignore these unstable signals from the RP until Partial Reconfiguration is complete and the RM is reset. This action is called decoupling, and a way to implement this is by registering the output signals (on the static side of the interface) from the RP, using an enable signal to isolate the logic until it is completely reconfigured.

3. Routing isolation: Upon developing a fault-tolerant system, it is crucial to provide spatial detachment between functions to secure that a failure in one will not propagate to the other. Isolating tasks within a single chip is done by using a "fence" logic. It is a set of unused tiles with no present logic functions.

4. Dedicated initialization isolation: if this feature is used, the static logic must be kept out of the RP.

5. RP resource utilization: The logic and routing of an RM must be fully contained within the physical region that includes the RP, which is then translated into a partial bitstream. When selecting the RP space, the designer must consider if the chosen area has enough resources (CLBs, special circuits, and others) to receive the RMs.

3 STATE OF THE ART

Over the past four decades, there have been significant contributions in computer science, artificial intelligence, robotics, and other related fields. This progress has allowed the development of robotic systems that are more capable and sophisticated, such as biomimetic robotics [94].

There is also much effort towards building biomimetic robotic hands, which have contributed to a better understanding of implementing a human hand into dexterous prostheses, widening its applications through improved sensors and mechanisms and HMIs [74]. One of these improvements is tactile sensing, which uses physical signals to identify the phases of object manipulation, namely: (a) non-contact to contact, (b) rotation, and (c) sliding [61]. For the application of robot hands, these phases translate into object recognition, force control, and grasp. Tactile sensors can measure different magnitudes, such as force vectors, vibrations, and contact actions. On the other hand, signal-processing techniques and modeling improve these measurements or even estimate others when sensors cannot read them directly. Such measures are then used for control schemes that perform grasping tasks [48].

Some works have accomplished these aspects by clustering several data processing devices in parallel. Table 3.1 presents a summary of some of those works with their main characteristics and achievements. The table includes the following key points: (1) control scheme or strategy for grasping, (2) finger gear or type of transmission used for the fingers movement, (3) the type of actuators used, (4) the quantity and type of sensors used, (5) the CU or computational control unit, (6) the control loop refresh frequency in Hz, and (7) the number of DoF. It is worth noticing that only works that included the actuators either in the palm or the finger were considered; i.e., robotic hands with actuators in the forearm or outside the hand were not considered.

Table 3.1 indicates that some implementations use PID for force control [54, 9, 45]. Although this solution is simple, efficient, and easy to tune, it does not control the dynamics of the contact between the manipulator and object. The impedance controller cannot only do this but also performs well at exerting forces on the environment and achieve good robustness at handling flexible components with unknown stiffness.

On the other hand, Table 3.1 also shows that the computational unit (CU) of some approaches [54, 95, 53] use several components to achieve the required parallelism of the controllers by using well-known micro-controllers. However, it results in an ample physical space required by the CU and demands implementing communication strategies between the processing devices. Lastly, as expected, more components mean an increase in energy consumption.

Table 3.1 – Comparison with some other robotic hand implementations

Author	Control scheme	Finger gear	Actuators	Sensors	CU	Freq. (Hz)	DoF
Robotic Hand (This Work)	Impedance control	Four-bar linkage and worm gear	7 DC motors	7 angular position, 7 current sensors	SoC ARM + FPGA	$>$25000	7
KITECH-Hand [54]	PID current control	Spur gears	16 DC motors	16 angular position, 16 current sensors	μCU matrix (16)	333	16
Calderon *et al.* [9]	PID control	Four-bar linkage	5 DC motors	5 force sensors	PC	-	5
Jeong *et al.* [45]	Neural Networks for position estimation and PID current control	Four-bar linkage	6 DC motors	5 position sensors	PC	-	6
Wang *et al.* [95]	Impedance control with internal PID position control	Four-bar linkage	5 DC motors	5 Encoders; 5 hall position; 5 torque sensors	Multiple DSP (2)	10 - 40	5
LMS Hand [21],[32]	Force control with Neural Networks and PID position control	Wire-driven	16 DC motors	16 encoders (for motors), 16 absolute encoders (for joints)	PC	50	16
Lee *et al.* [55]	Hybrid PD position/force	Four-bar linkage	9 DC motors	9 incremental encoders, 4 resistive force sensors	PC	-	9
HIT/DLR Prosthetic Hand [107],[41]	Impedance control and PD position control	Four-bar linkage	3 DC motors	3 encoders, 3 hall position, 3 torque, 3 force sensors on fingertips	Multiple DSP (2)	1000	3
HIT/DLR Prosthetic Hand II [57],[53]	Impedance control and PD position control	Wire-driven	15 Brushless DC motors	15 position, 15 torque, 5 force on fingertips, 10 temperature sensors, tactile sensors.	Multi-processor DSP+FPGA (6 DSP & 6 FPGA)	-	20
KNU Hand [16],[44]	Position control with sliding detector	Worm gear and four-bar linkage	2 DC motors	2 position sensors	DSP	-	6

Many of the implementations depicted so far include simplifying the complexity of the human hand to achieve embedding. Field Programmable Gate Arrays (FPGAs) are a good match in pursuing this motivation. Additionally, these devices can be fundamental in the simultaneous control and processing of several actuators and sensors that must be handled synchronously. In conventional processors, the correct synchronization can be hampered by the serial nature of executing instructions in von Neumann-based architectures. In this way, FPGA-based platforms allow the designer to implement efficient digital architectures capable of reading signals, in parallel, from multiple sensors. They generate many output signals processed by complex models, sometimes driven by machine learning algorithms such as artificial neural networks.

In [2], the authors provided new hardware architectures for FPGA implementation of artificial neural networks with custom floating-point precision. The gains were in the order of hundreds compared to a serial solution at the same clock rate. Otherwise, the implementations of radial basis function artificial neural networks (RBFNNs) on hardware are not numerous in the literature, as summarized below. In [13] the authors implemented an offline trained RBFNN for nonlinear channel equalization mapped on a Xilinx Virtex-2 FPGA, where the exponential function was approximated by a 4th order Taylor expansion. A scheme for RBFNN evaluation and online training based on fuzzy c-means and recursive least mean square algorithms are implemented on an FPGA in [25]. The authors use benchmarks for classification problems to evaluate an Altera Cyclone III EP3C120 FPGA implementation and calculate the exponential function with the floating-point Altera mega function. An FPGA architecture for RBFNN is investigated in [15] when applied to uncertainty detection for online controller tuning of a permanent-magnet synchronous motor and tested through co-simulation using Simulink and ModelSim tools. The exponential function is calculated according to a 12th order Taylor expansion, and the data types are set in the Q15 format with 16 bits and two complements.

In [23], the authors propose the online training of RBFNNs using the least mean square algorithm, which was tested with the XOR problem and sine function approximation in a Xilinx Virtex-6 FPGA. The authors analyze fixed-point operations using hardware, circuit frequency, and output accuracy, varying the word length. The XOR problem is again used to test an RBFNN implementation with 32-bits floating-point operations in FPGAs with online backpropagation learning in [50], where the Taylor series approximation was used for implementing the Gaussian function. It is possible to see that the cited works lack to compare the impact of the word length on the precision and hardware consumption for floating-point operations according to a given task in order to establish an optimal compromise between hardware resources and accuracy. Moreover, none of them make use of the original modular architecture of the RBFNN presented in [2] to provide solutions with the same word length but with fewer hardware resources.

Among the most important factors when implementing ANNs on FPGA hardware, one may cite [39]: data representation, inner-product calculation, implementation of the activation function, storage, and update of the weights. Concerning data representation, precision should be considered as it will directly affect the accuracy. Note, however, that the resources needed in the hardware will increase together with the precision. On this matter, floating-point representations present advantages over fixed-point [85] given that a fixed-point representation would require a more extensive word or a truncate of the value range to obtain the same precision. Floating-point representation has an extended dynamic range with high-resolution [28]. Nonetheless, there is no extensive support for designers on floating-point arithmetics in FPGAs [80].

On the other hand, the limited resources of an FPGA and the necessity of embedding complex algorithms and systems with performance and power consumption requirements bring the scientific community to use partial dynamic reconfiguration for multitasking, power-efficient systems. Embedded systems implementation in FPGAs using DPR is an endeavor useful for several fields, such as security, fault-tolerant systems, machine learning, and Internet-of-Things (IoT), among other applications. A first example [46] exploits DPR's readiness to bring in-the-field hardware upgrades and updates to remote sites and end-user continuous service to implement a lightweight cryptographic security protocol with Physically Unclonable Function (PUF) circuits. Another example [49] takes advantage of one of DPR implementation's most significant claims, energy efficiency. The work introduces a reconfigurable architecture for Reduced Instruction Set Computing (RISC) processor to support IoT applications with different performance and energy trade-off requirements. On the other hand, [81] performs a similar approach to achieve energy-efficiency in IoT embedded applications by implementing a secure encryption protocol.

DPR systems can also be exploited for real-time applications such as those reported in [73]. In that work, hardware accelerators were used to implement computationally intensive tasks in real-time to capitalize on DPR's capabilities to replace accelerators that are no longer needed with new ones, leading to more efficient utilization of hardware resources. Similarly, [12] performs real-time image processing of earth hyperspectral using FPGA with DPR. The approach extracts the end members of a hyperspectral image using the Modified Vertex Component Analysis (MVCA) algorithm, implemented in FPGA with adjustable execution time performance thanks to DPR.

On the other hand, FPGAs were also used to be a viable alternative to Graphical Processing Units (GPUs) for machine learning algorithms, such as NN implementations, especially according to applications with strict power performance constraints. In [42] the authors built a dynamic multi-classifier architecture used to process bioinformatics data. The implemented classifiers were SVMs and KNN combined. The two classifiers were too big to be implemented, and DPR enabled even improving performance when compared with static solutions of either. Furthermore, [10] implemented an Ensemble Machine Learning in an FPGA using DPR showing improvements in resources utilization, reconfiguration speed, power consumption, and maximum clock frequency.

Deep Learning algorithms can also take advantage of DPR features, like in [103], where a Convolutional Neural Network (CNN) hardware accelerator is carried out. Given that the different network models, such as AlexNet, VGG, SSD, and YOLO, have different operators, DPR comes in handy when creating a coprocessor that can implement a set of different CNNs online, allowing to use any only when it is needed. Coprocessors in FPGAs are commonly used with a bigger processor and communicate through a bus, often referred to as Hardware/Software codesign. This was implemented in [52] where a CNN is also implemented on an FPGA-based SC with DPR.

The implementation of all these applications does not have a standard methodology, and most of them use custom DPR controller IPs that difficult the system implementation for non-expert designers and early-entry users of DPR. For that, this work aims to develop a tool that can be used, even by non-expert users, to develop DPR systems. However, this endeavor has already been pursued by previous researchers. Table 3.2 evidence this by depicting six other frameworks that closely relate to this work's objective. It also summarizes the tools' features compared to RTRLib's current capacities and the ones proposed by this work.

Table 3.2 – Comparison of the main PR design tools. * are features under development

	Interface	PR-PR Bus	GUI	RM HDL Generator	HLS	Fault Tolerace	App Generator	IDE	Platform
Recobus Builder [51] & macro	Various	×	✓	×	✓	×	× (XDL)	ISE Spartan 3	Virtex II,
GoAhead [3]	Direct wire	✓	✓	×	×	×	×	ISE (XDL)	Virtex II Spartan 6
RePaBit [77]	Bus macro	×	×	×	×	×	×	VIVADO (TCL)	Zynq SoC
IMPRESS [106]	Virtual interface	✓	×	×	×	×	×	VIVADO (TCL)	Zynq SoC
FASTER [75]	N/A	×	N/A	×	✓	×	✓	VIVADO (TCL)	Virtex 5, Zynq SoC
CAOS [76]	N/A	×	✓	HLS	✓	×	✓	VIVADO (TCL)	UltraScale VU9P
RTRLib [43]	Partition pins	✓*	✓	Structural and HLS *	✓	Hardware redundancy	Baremetal, FreeRTOS, Linux *	VIVADO (TCL)	Zynq SoC

Recobus-builder [51] introduces point-to-point PR-Static communication links to automate the design steps required to construct DPR systems. It permits a flexible integration of multiple modules using I/O bars, ideal for data streaming. Moreover, GoAhead [3] provides static/partial decoupling. (a goal of this work), hierarchical reconfiguration, and reconfigurable region crossing. Additionally, RepaBit [77] produces relocatable, one and two-dimensional, partial bitstreams for Xilinx Zynq devices. RepaBit uses the Xilinx Isolation Design Flow (IDF) to avoid feed-through routes. IMPRESS [106] implements a custom virtual interface based on fixed nodes shared between the static logic and the RPs. It also supports relocatable bitstreams avoiding feed-through routes and uses the AXI interface to communicate different parts. The FASTER tool [75] framework that allows the development of reconfigurable heterogeneous MPSoC systems. The user provides a C-based application, an XML file with information about the target architecture and the HDL implementations for the hardware cores. Also, FASTER implements high-level analysis, region optimizations, compile-time scheduling, and mapping into reconfigurable regions. Finally, CAOS [76] adopts HPRC (High-Performance Reconfigurable Computers) systems which comprehend the full process of application optimization; from the identification and optimization of the kernel functions to the generation of the runtime management for the target system, at the implementation of DPR systems.

RTRLib is a high-level modeling tool that automatically provides

- VIVADO-compatible TCL scrips for developing DPR systems on Xilinx Zynq devices,
- Early estimation of the resource utilization, latency, and reconfiguration time of each RM.

Which can guide the designer concerning technical issues such as the size of each RP, the communication interface between RPs, and the reconfiguration process. The features mentioned above are possible because the RTRLib is a platform-based design tool that uses the semi-formal refinement-by-replacement methodology, allowing users to interconnect functional blocks as a network process to express a specific behavior. Thus, each functional block is refined and replaced by previously characterized IP-cores that deliver the HDL code. In the case of hardware redundancy schemes for fault-tolerant systems, the tool also estimates the failure rate and reliability of each RM.

4 METHODS

4.1 MYKI PROJECT - SANT'ANNA

The experimental setup for this work is divided into two parts. The MYKI Project that aims to develop a new myokinetic HMI that detect the position of the multiple implanted magnets to control a robotic hand and the previously developed robotic hand already built in the LEIA group.

The present work has a chapter in collaboration with the researchers involved with the ERC-funded project titled "A BIDIRECTIONAL MYOKINETIC IMPLANTED INTERFACE FOR NATURAL CONTROL OF ARTIFICIAL LIMBS" (MYKI) together with the Biorobotics Institute at the Scuola Superiore Sant'Anna - SSSA and Prof. Helon Ayala from the Pontificia Universidade Catolica do Rio de Janeiro - PUC-Rio [93]. MYKI aims at developing and clinically evaluating a dexterous hand prosthesis with tactile sensing, which is naturally controlled and perceived by the amputee. This will be possible by overcoming the conventional approaches based on recording electrical signals from the peripheral nervous system (nerves or skeletal muscle) through the development of a radically new Human-Machine Interface (HMI) based on magnetic field principles, embracing the idea of sensing the magnetic field of implanted magnetic markers (MMs), able to both decode voluntary motor and force commands of an individual. During isotonic contractions, the deformation of the muscles is axial (they shrink). During isometric contractions (i.e., when the muscle generates force because the distal end is blocked), the muscle deforms in the radial direction (i.e., it bulges). By monitoring both these (radial and axial) deformations, in principle, it is possible to retrieve information on the force exerted by the muscle and on the joint position [93, 88].

To test the concept of the myokinetic interface, together with its design features, an anatomically equivalent forearm mockup is developed, see Fig. 4.1. The mockup is designed to replicate the anatomical placement and contraction movements of the extrinsic muscles of the hand. In particular, the mounted MMs emulates the Flexor Pollicis Longus (FPL - MM_0), the first two compartments of the Flexor Digitorum Superficialis (FDS1 and FDS2 - MM_2 and MM_3) and on the Abductor Pollicis Longus (APL - MM_1). These MMs are linked to the flexion control of the thumb, index, and middle fingers and the abduction of the thumb, respectively. The mockup was used to experimentally assess the accuracy, repeatability, and response time of the myokinetic HMI interface prototype.

The concept of said HMI to track the muscle contractions with implanted permanent magnets, using magnetic field sensors, was proposed in [88] in the scope of the MYKI

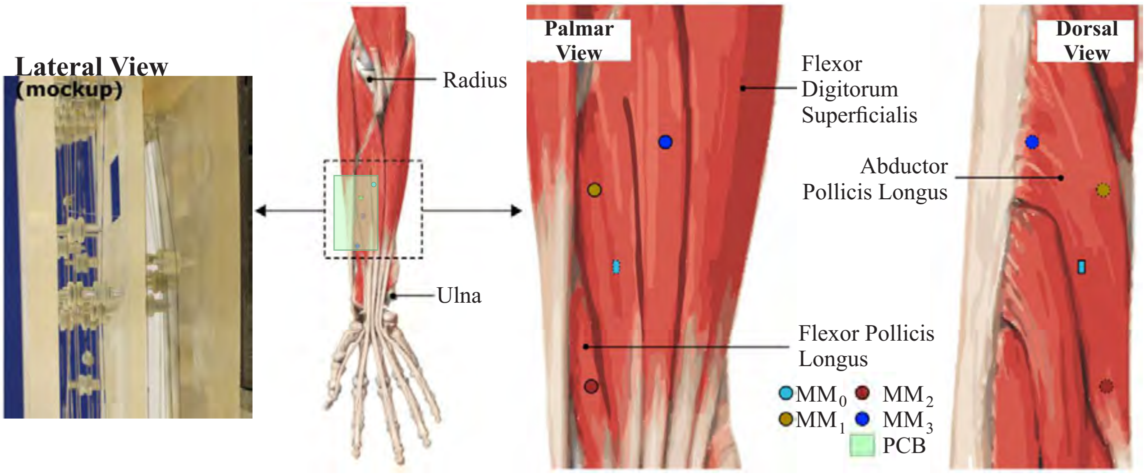


Figure 4.1 – Experimental Mockup setup. On the left is depicted forearm mockup with implanted magnets (MMs). The mockup reproduces natural position and orientation of forearm muscles, in addition to their deformation due to contraction. Adapted from [88]

project. In this work, the authors present the concept, features, and a demonstration of a prototype which exploits one of the four sensors' PCBs to localize a single magnet implanted in the mockup, for the control of a dexterous hand prosthesis [88]. Figure 4.2 presents the developed concept.

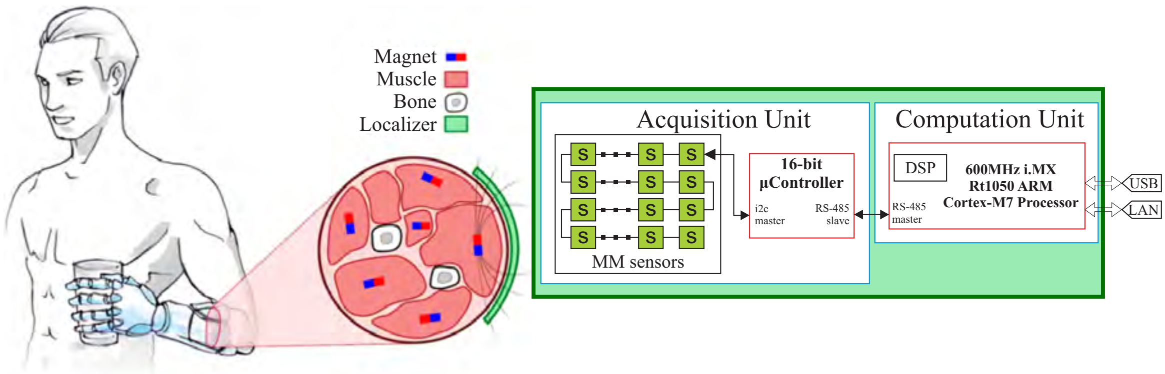


Figure 4.2 – Description of the current embedded architecture of the control interface for a prosthetic hand. The magnetic field is acquired by a matrix of sensors ("S" on the acquisition unit) to compute (computation unit) their position. For the target application (a prosthetic hand) this information could be used to control its movements with physiological accuracy. Notice that the red lines denote the microprocessors spatial limits. Adapted from [17]

4.1.1 Experiments Data Acquisition

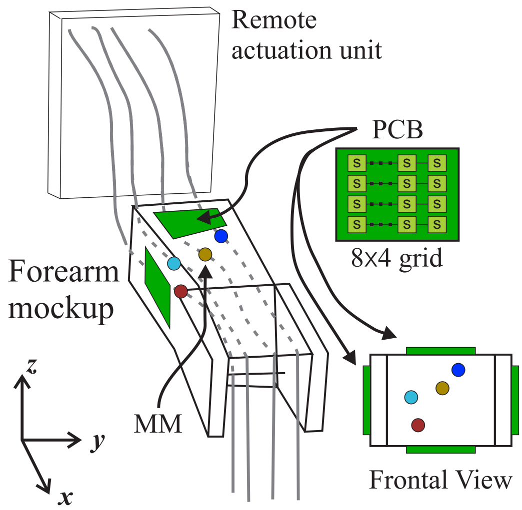


Figure 4.3 – Mockup Schematic. It simulates the movements of the hand's 17 degrees of freedom, for a total of 17 wires, although only one is currently being used. Muscles were modeled using a wire attached on one side to a servo motor (housed in a remote actuation unit) and on the other side to counterweight to maintain tension. Adapted from [88]

For the sake of these tests, two datasets are acquired, one with the displacement of one magnet and the other with five magnets (each representing one finger's movement of flexion-extension). The magnets are driven by servo motors, and the displacement of the magnets is adequately driven to achieve a maximum translation of the wire (i.e., of the muscle) of $\sim$10 mm. A total of eleven datasets are acquired (six using one magnet and five using five magnets), giving a new input to the servos at a $T_s = 50$ ms sample-rate. The input displacement (target) provided to the servos is used as a ground-truth for analyzing the accuracy of the retrieved magnet displacement.

The magnetic fields are sampled through four custom printed circuit boards (PCB), each equipped with 32 three-axis magnetic field sensors (Fig. 4.3). For every PCB, the sensors were laid out on orthogonal 8$\times$4 grids. A 9 mm gap separates each column and row. The PCBs are spatially centered on four opposite sides of a parallelepiped enclosing the workspace (100 mm $\times$ 54 mm $\times$ 100 mm) (two on the XZ plane and two on the XY plane, see Fig. 4.3). The PCBs send to a PC (Windows 7, Intel i7-6700 CPU running at 3.4 GHz, 32 GB of RAM) the readouts from the sensors at a rate of 20 Hz, through a serial bus (RS-485). The collected signals are stored to be used for offline processing.

4.1.1.1 Description of Data - 1 Magnet

In particular, for one magnet, the following datasets were used as target:

- One multisine data set, for creating the model. It is composed by a sum of sinusoid signals as

$$y_k = \frac{A}{2M}\left\{\sum_{i=1}^{M}\left[\sin\left(2\pi f_i t_s k + \phi_i\right)\right] + 1\right\}, \tag{4.1}$$

  where y_k denotes the servo command used as output for the model at discrete time $k = 0, 1, ..., N$, A is the desired amplitude for y_k in the range $[0, A]$, M is the total number of sinusoids in the equally spaced range for $f_1, f_2, \ldots, f_M$, and $\phi_1, \phi_2, \ldots, \phi_M$ are uniformly random phases in the range $[0, 90°]$. The multisine is able to excite the system in an specific range and also is able to reveal if the dynamic relationship between the input and the output of the system should be taken into account.

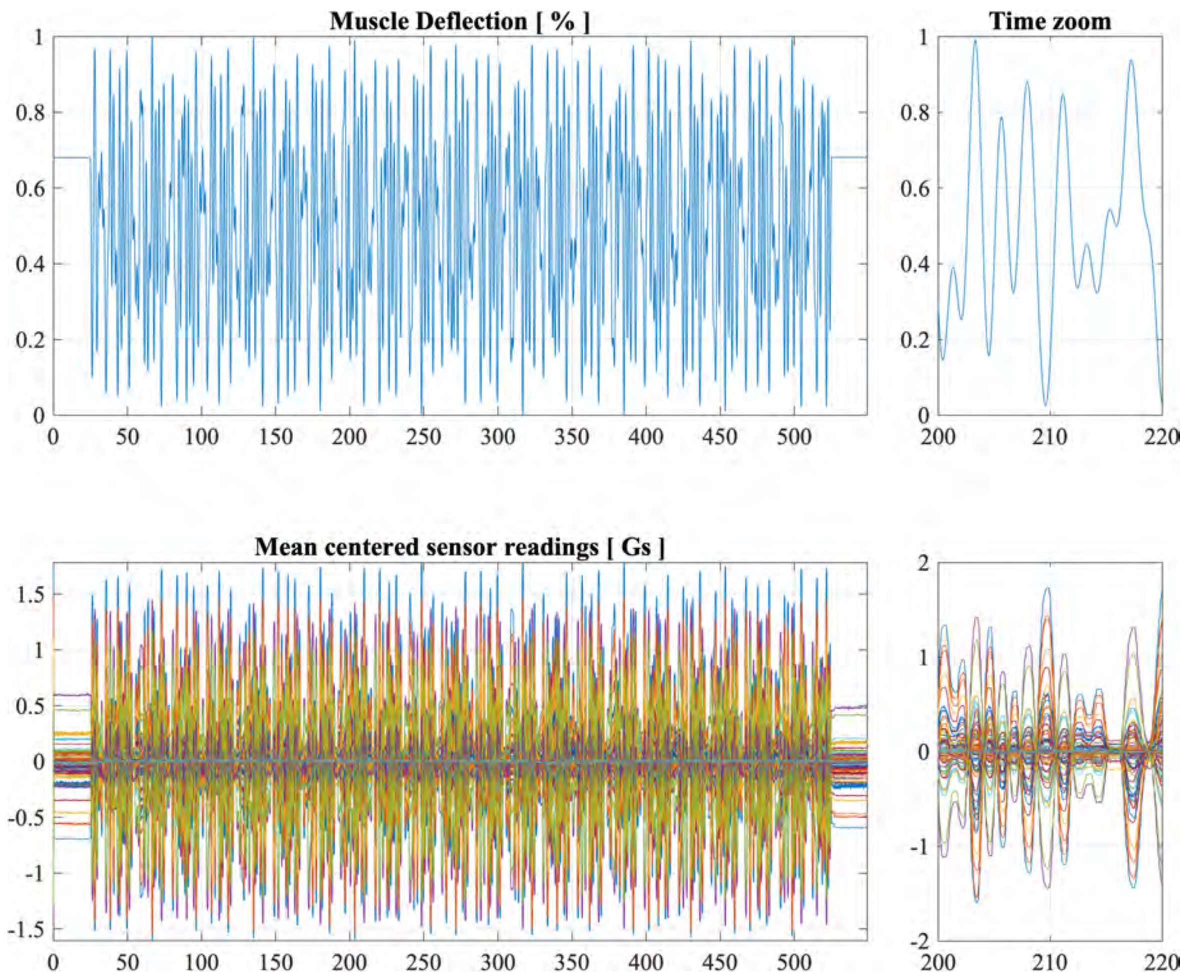


Figure 4.4 – Measured dataset with the multisine excitation signal (left) and zoomed around 20 seconds (right). The sensors readings are the magnetic fields in Gauss.

  For the acquisitions, it is used $A = 10$ mm, as it is the range of operation for the mockup, $M = 10$ sine waves, and frequencies $f_1, ..., f_M$ equally distributed in a range between 0.1 and 0.5 Hz (see Fig. 4.4).

- Four ramps and one sequence of steps, for testing the models. Ideally, a data-driven model should be tested with a different dataset than used to train it. To this end, it is used ramp and step signals datasets that are relevant for the application of trajectory tracking using the myokinetic interface. The ramps are configured with positive and negative slopes with different speeds during 20, 40, 60, and 80 seconds. The steps were equally divided into twenty levels, ranging from the minimum (0 mm) to the maximum (10 mm) servo displacement, and were applied sequentially. The sequence of ramps and steps is given in Fig. 4.5.

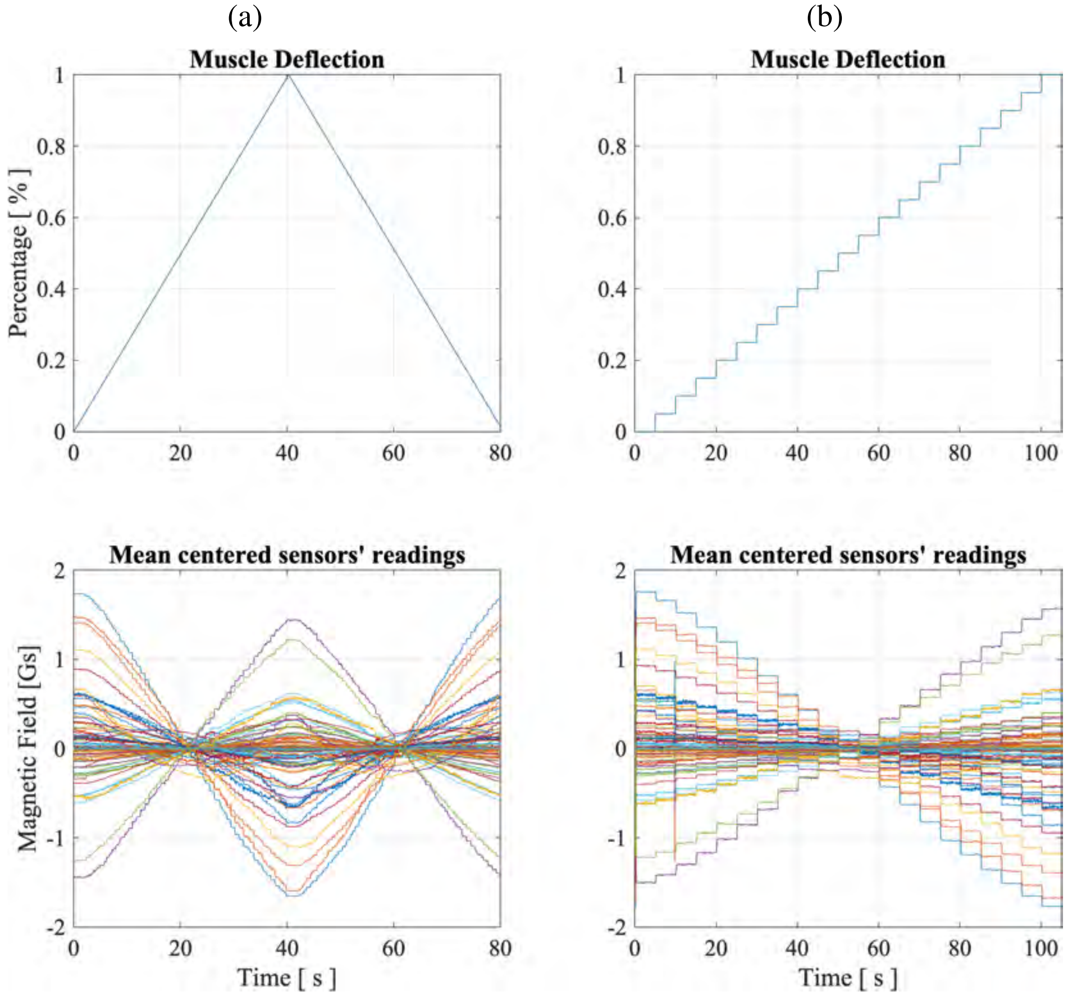


Figure 4.5 – Measured dataset for (a) the ramp excitation signal with 80 seconds total duration and (b) the step excitation signal.

From the plots in Fig. 4.4–4.5 it can see that there is a cause-effect relationship between the measured magnetic field and the output displacement. This observation is confirmed in Fig. 4.6, which plots the normalized histogram of the Pearson cross-correlation for the magnetic sensors data and the output displacement, ordered by groups of the amplitude of R in ascending order. It is possible to see that many sensors are linearly correlated with the output muscle deflection (positively and negatively), which indicates that machine learning models can represent the measured displacement based on the magnetic field data.

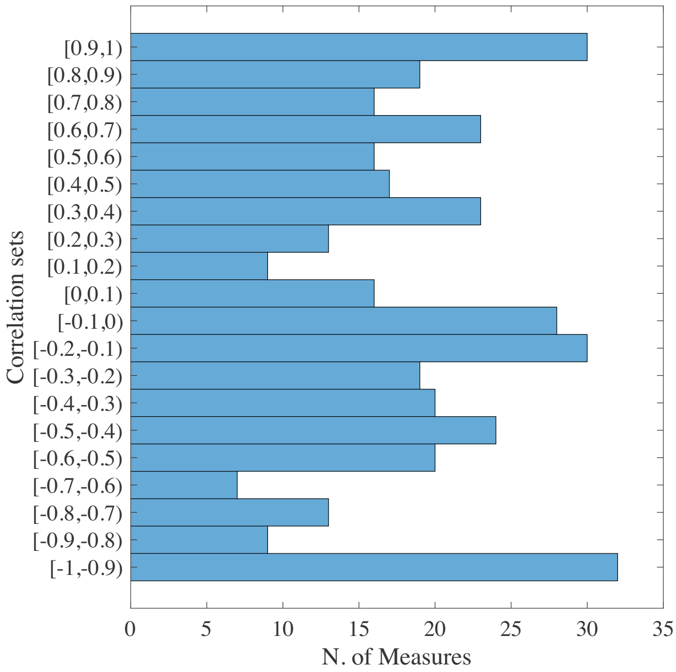


Figure 4.6 – Distribution of the cross-correlation among the magnetic sensors time series and the muscle's output displacement. Note that many measurements are directly or inversely linearly correlated with the output deflection.

4.1.1.2 Description of Data - 5 Magnets

For five magnets, the following datasets were used as a target:

- One multisine data set for creating the models. It is composed of a sum of sinusoid signals similar to Eq. 4.1. The multisine can excite the system in a specific range and reveal if the dynamic relationship between the input and the system's output should be taken into account. This can be more relevant when having many magnets since every magnet's magnetic field can overlap with the others and make the process of model estimation more complex.

  Similarly, for the five magnets dataset, a maximum distance of 10 mm was used, and the same range frequencies between 0.1 and 0.5 Hz (see Fig. 4.7).

- Four ramps, for testing the models. The ramps are also configured with positive and negative slopes with different speeds during 20, 40, 60, and 80 seconds for every magnet. In these datasets, every magnet was moved individually. Like this, the accuracy can be tested for the movements of every finger individually. The sequence of ramps are given in Fig. 4.8.

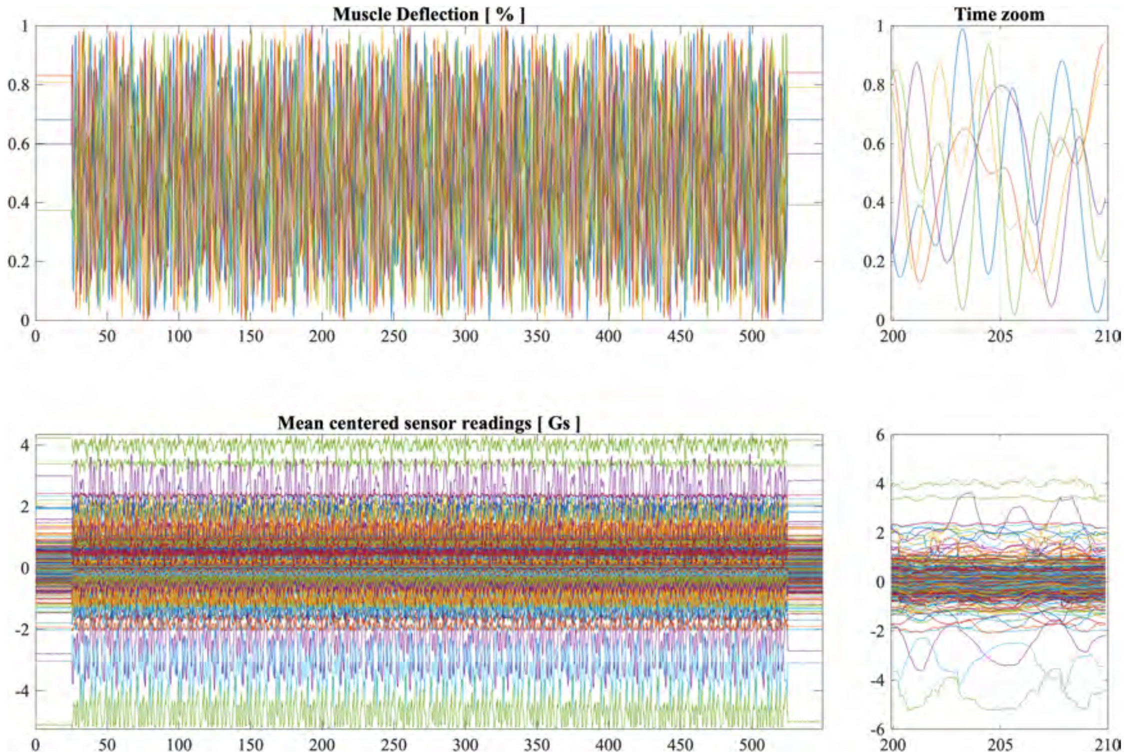


Figure 4.7 – Measured dataset for five magnets with the multisine excitation signal (left) and zoomed around 20 seconds (right). The sensors readings are the magnetic fields in Gauss.

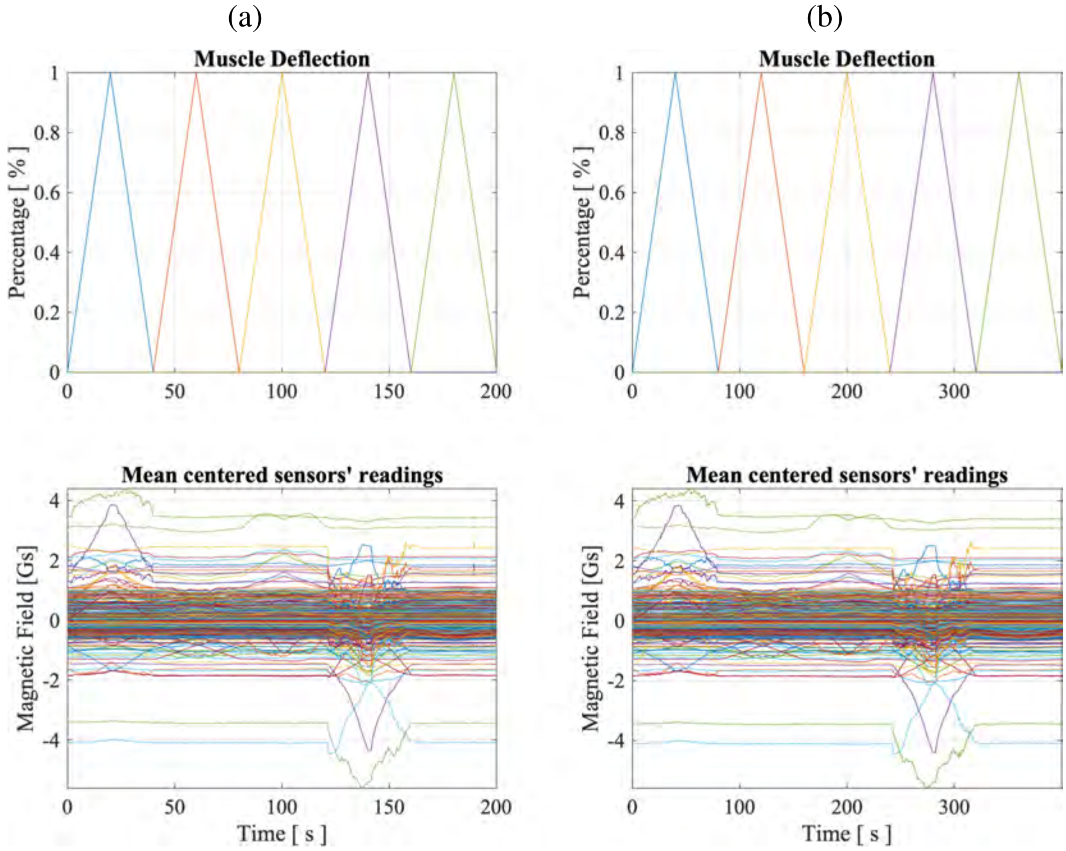


Figure 4.8 – Measured dataset for (a) the ramp excitation signal with 40 seconds total duration and (b) other with 80 seconds.

Somewhat different from the plots with one magnet in the previous section, the cause-effect relationship between the measured magnetic fields and the output displacement of every magnet is subtler now. This observation is confirmed in Fig. 4.9, which plots the normalized histogram of the Pearson cross-correlation for the magnetic sensors data and the displacement of the output, similar to Fig. 4.6. It is possible to see few sensors linearly correlate with the output of each magnet deflection, which indicates that simpler machine learning models may not represent the measured displacement correctly. It makes then that more adaptive models, such as ANN, may be more effective. Thus, they can then be implemented on hardware for fast and energy-efficient solutions, as devised in the next section. The challenge was to design offline a model that can be embedded as a general architecture that delivers better timing results than those available in the literature.

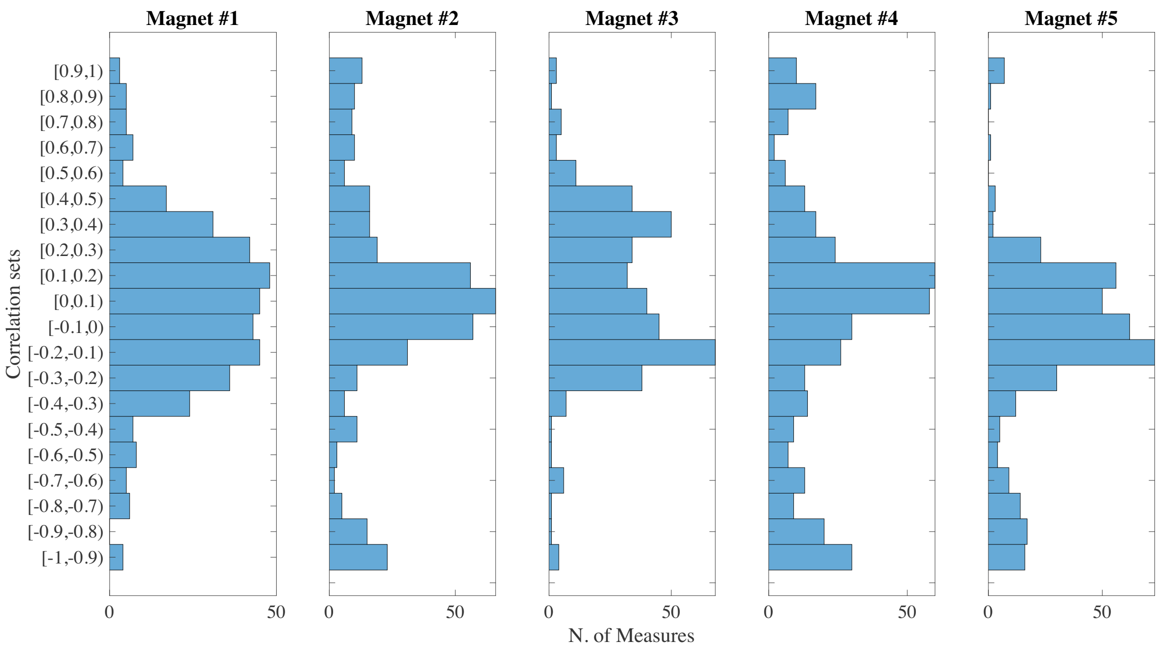


Figure 4.9 – Distribution of the cross-correlation among the magnetic sensors time series and the muscle's output displacement with five magnets. Note that here, few measurements are directly or inversely linearly correlated with the output deflection.

4.2 ROBOTIC HAND TESTBENCH - UNB

Figure 4.10 depicts the robotic hand used in this work to validate the proposed FPGA-based localizer-controller. The complete system is composed of 7 DoF, each actuated by a DC motor, seven analog rotational position sensors, and seven motor current sensors [71].

In total, there are five fingers in the robot. The thumb and index fingers' configurations have extra actuated joints for their *aa* movements. This project proposes three different finger configurations (see Fig. 4.10(a) for viewing the joints), as follows:

1. **Configuration 1** (Thumb): 2 DoF (*aa* and *fe*) and 3 joints,
2. **Configuration 2** (Index): 2 DoF (*aa* and *fe*) and 4 joints, and
3. **Configuration 3** (Remaining fingers): 1 DoF (*fe*) and 3 joints.

(a) (b) (c)

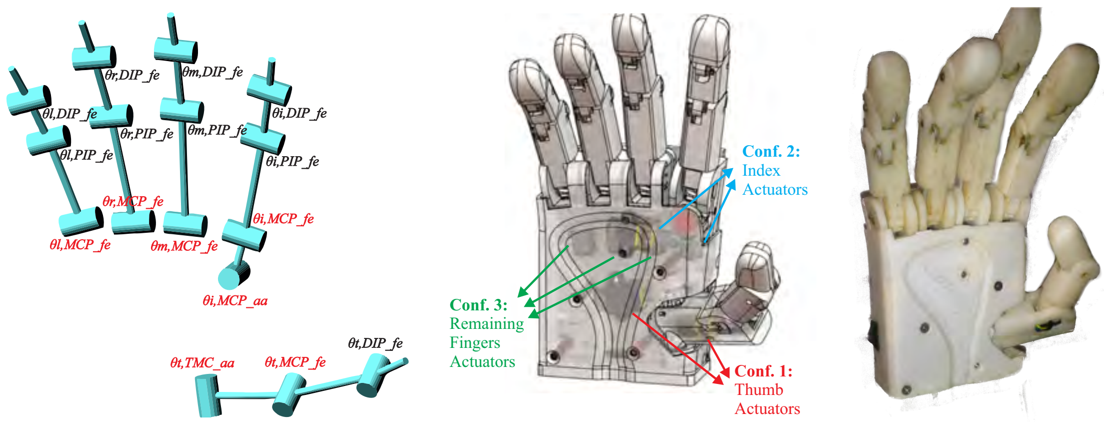


Figure 4.10 – Robotic Hand. (a) Kinematic structure of 7 DoF simplified robotic hand. Actuated joints are highlighted in red, the other ones are sub-actuated. (b) Render of assembled robotic hand with the locations of the actuators and extra DoF of the index and thumb. (c) Real picture of the assembled robotic hand.

4.2.1 Design Overview of Robotic Hand

The selection of the DoF used for this robotic hand was not trivial, and the reader is refered to [72] for a detailed description of the design and optimization of the robotic hand. The importance of these DoF in reference to others in a human hand can be quantified. For example, the thumb adaptation of the well-known Kapandji clinical test [47] can be used for this goal. This test assesses the thumb's opposition by checking its ability to touch a specific part of the hand. They are ten tests/positions in increasing difficulty, with the easiest consists of touching the proximal phalanx of the index finger and the hardest is to touch the distal palmar crease. Fig. 4.11 depicts two of the most representative Kapandji tests (six and ten) validating this work's thumb opposable ability by achieving the most challenging position.

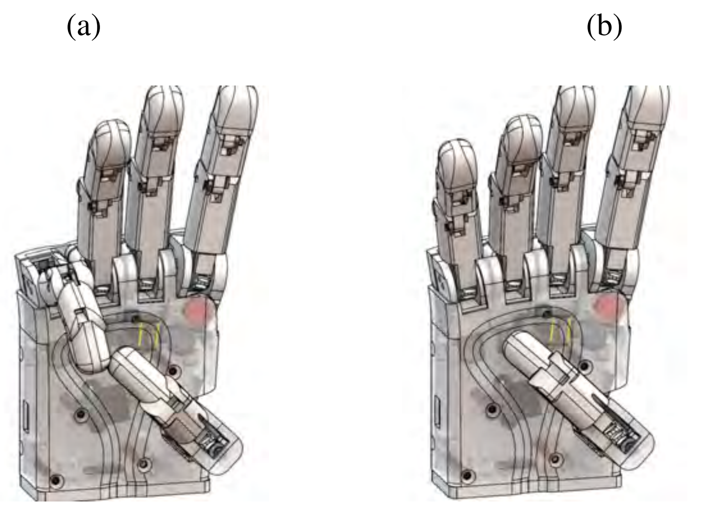


Figure 4.11 – Kapandji Test for robotic hand. (a) Position six: thumb touches the little finger. (b) Position ten: thumb touches distal palmar crease.

Additionally, another important aspect of this robotic hand is that electronics are embedded in the palm and located in a custom PCB, except for the position sensor. Figure 4.12 illustrates the schematic of the controller dividing it into two parts the Acquisition Unit (AU) and the Controller Part (CU). The signals are accessed from the CU through an IDC 34-pin header that connects to the controller device.

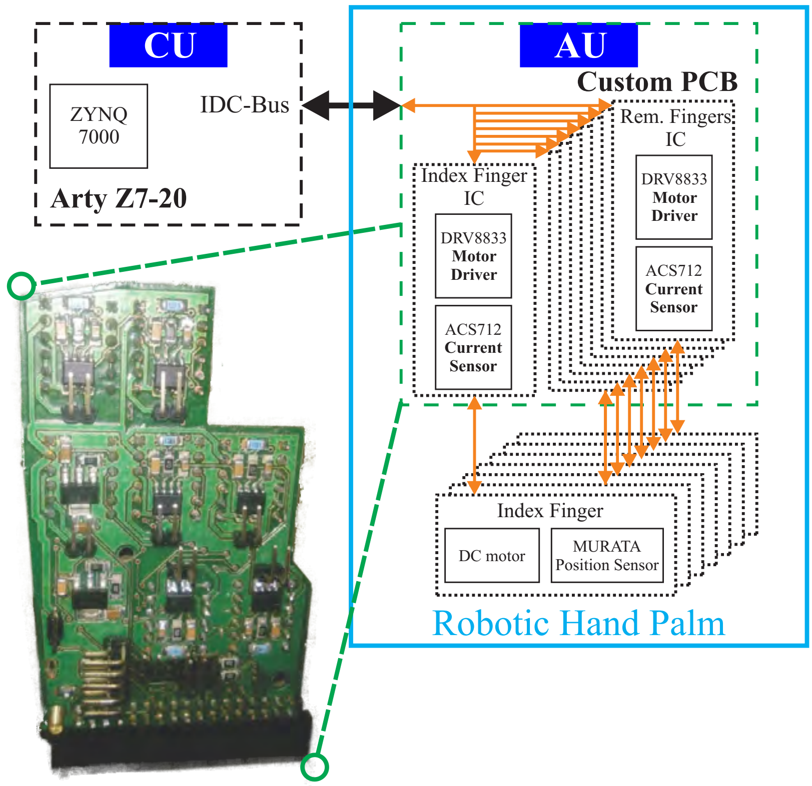


Figure 4.12 – Schematic diagram the Robotic Hand Acquisition Unit (AU) and Controller Unit (CU). The entire AU is embedded in the palm while the CU is the Arty board that is located outside. On the south-west corner is a photo of the custom PCB of robotic hand.

The AU includes the motor drivers (DRV8833), the current sensors (ACS712), and some other power regulation ICs (Integrated Circuits). It also has connection pins for the IDC breakthrough and sockets for the DC motors, and the analog angular position sensors. The PCB has dimensions of approximately $80 \times 52mm$ and has an irregular form designed to fit in the palm (See Fig. 4.12). The CU consists of the Arty Z7-20 development board. It contains an XC7Z010 SoC (System on a Chip) that includes a Zynq®-7000, an Artix™-7 FPGA (Field Programmable Gate Arrays), and a Dual-Core ARM®Cortex®-A9 at 667MHz. The chip also includes a XADC (Analog to Digital Converter) necessary for the current and position sensors.

4.2.2 Description of Controller Scheme

This work implementes an impedance controller that is executed in a decoupled form, meaning that the action of every DoF does not have effects on the others. Thus, an analysis of the system is more straightforward, being possible to implement the controllers separately.

The robotic hand aims to embed the low-level control algorithms and its novelty is that it does this as a HW/SW codesign. The filtering process and the impedance controller are implemented as reconfigurable architecture in an FPGA/SoC device. Doing so enables implementing computing-intensive functions in parallel, in a single chip, with little compromise to time-execution performance and energy consumption.

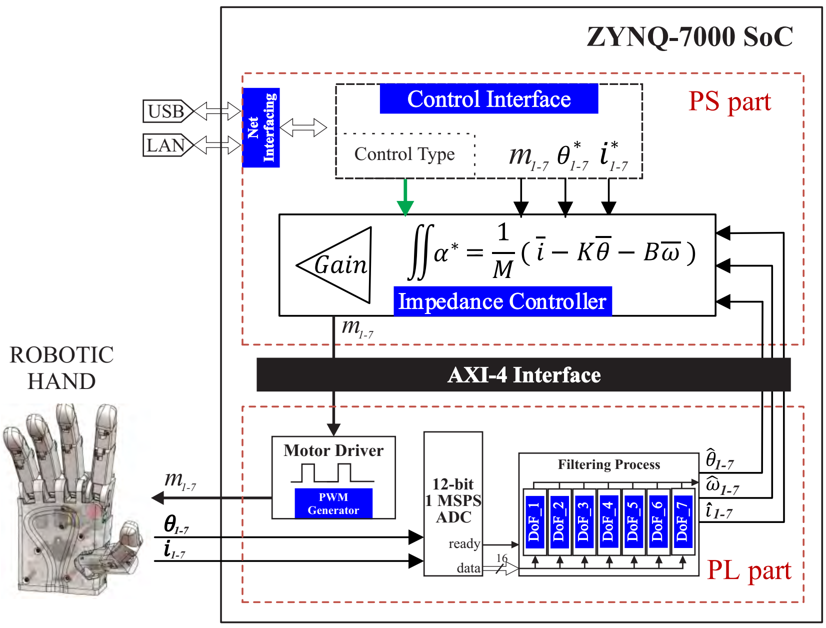


Figure 4.13 – Internal architecture of the CU. The Filtering Process block is replicated for the same amount of DoF (Degrees of Freedom) in the robotic hand. These blocks, including the Motor Driver, are controlled and synchronized via the AXI interface with the PS-part. The PS-part also executes the impedance controller algorithm and the control interface.

The Filtering Process block (presented in Fig. 4.13) is the most expensive part of the control scheme. So, this part is determined to be described in VHDL (VHSIC "Very High-Speed Integrated Circuit" Hardware Description Language) on the CU's FPGA (i.e. Programmable Logic Part / PL-part). It is essential to point out that hardware implementation has some advantages, like a logic architecture's intrinsic parallelism. On the other hand, a drawback is that the designer must be mindful not to surpass the number of logical resources on the chip.

Moreover, the Impedance Controller block in Fig. 4.15 is executed in the CU's Processor Unit (Programmable Software Part / PS-part). The composition of the CU's architecture and the dataflow is described in Fig. 4.13. Similarly, the robotic hand *Control Interface* block is implemented in the PS-part. It can switch the control technique between manual, position, and impedance control; this is described more thoroughly in the following subsection.

The *AXI interconnect* block, is a proprietary microcontroller bus that allows the connection between both the PS and the PL parts. This bus behaves as a two-way channel that allows the ARM to read and write the PL modules (supervisory task). The filtered data is forwarded to the PS-Part, which calculates the action of the impedance control. Finally, the Motor Driver module sends, in a parallel fashion, PWM (Pulse Width Modulation) signals to the motor drivers.

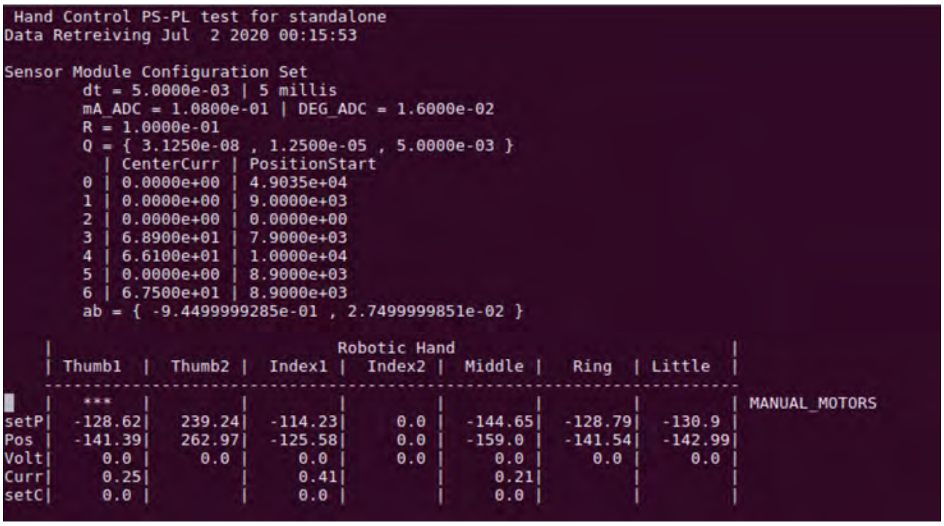
The PS-Part is also in charge of the connectivity of the system and as medium as Human-Machine Interface (HMI).It uses a text-based visual interface (see Fig. 4.14) that enables the user to control the robotic hand. The interface allows setting a series of predefined control values ($\hat{i}_{1,\cdot;\cdot 7}$) for different grasps and manually controlling each finger separately. The "***" below the fingers' names indicates the currently controlled one. The control variable of that finger can be increased or decreased. The user can also declare which control variable will be adjusted between three options by overriding it in the impedance controller dataflow. This circumvention of the dataflow is depicted in Fig. 4.15 with the red, green and blue lines for i^*, θ^* and m^* respectively. Resuming the control can be done with three strategies:

1. MANUAL_MOTORS mode overrides the m^* variable (Seen as "Volt" on Fig. 4.15), allowing to control the voltage of the motors of the robotic hand manually; and hence, the flexion-extension movements of every finger. It sends a constant pulse to the motors vetoing the output of the proportional gain.

2. P_ONLY: Bypasses the desired position of a finger (θ_k^* and shown as "setP"on Fig. 4.15). Setting this control variable makes it possible to set a finger in the desired position, nullifying the impedance controller's action. Since the output of the MIKY interface are percentil flexion/extension of the fingers, this mode is what will be used in this work.

3. FULL_IMPEDANCE: Is the full proposed impedance controller scheme. It updates

the control variable i^* (“setC” in Fig. 4.15) into the impedance controller. It feeds the desired current for grasping objects.



```
 Hand Control PS-PL test for standalone
Data Retreiving Jul  2 2020 00:15:53

Sensor Module Configuration Set
        dt = 5.0000e-03 | 5 millis
        mA_ADC = 1.0800e-01 | DEG_ADC = 1.6000e-02
        R = 1.0000e-01
        Q = { 3.1250e-08 , 1.2500e-05 , 5.0000e-03 }
            | CenterCurr | PositionStart
        0 | 0.0000e+00 | 4.9035e+04
        1 | 0.0000e+00 | 9.0000e+03
        2 | 0.0000e+00 | 0.0000e+00
        3 | 6.8900e+01 | 7.9000e+03
        4 | 6.6100e+01 | 1.0000e+04
        5 | 0.0000e+00 | 8.9000e+03
        6 | 6.7500e+01 | 8.9000e+03
        ab = { -9.4499999285e-01 , 2.7499999851e-02 }

    |                          Robotic Hand                              |
    | Thumb1   |  Thumb2 |  Index1 |  Index2 |  Middle |   Ring  | Little |
    --------------------------------------------------------------------
    |   ***    |         |         |         |         |         |        | MANUAL_MOTORS
setP|  -128.62|   239.24|  -114.23|     0.0 |  -144.65|  -128.79|  -130.9 |
Pos |  -141.39|   262.97|  -125.58|     0.0 |   -159.0|  -141.54|  -142.99|
Volt|      0.0|      0.0|      0.0|     0.0 |      0.0|      0.0|      0.0|
Curr|     0.25|         |     0.41|         |     0.21|         |        |
setC|      0.0|         |      0.0|         |      0.0|         |        |
```

Figure 4.14 – Robotic hand visual interface. It illustrates some relevant data for the sensors calibration and filters parameters. It also shows sensors and control data: *setP* and *Pos* are the desired and filtered angular position of a finger, *Volt* is the voltage of a finger’s motor, and *Curr* and *setC* are the filtered sensor and desired current respectively. Finally, it also specifies the current control strategy for testing (MANUAL_MOTORS, P_ONLY or FULL_IMPEDANCE).

The dataflow diagram for controlling each *DoF* is depicted in Fig. 4.15. Note that this control scheme also identifies which parts are in the PS and which ones are in the PL. It can be observed that the Filtering Process is fully implemented for each DoF in the PL part, and the same is replicated seven times (see Fig. 4.13). In this way, each *Filtering Process* circuit receives each pair θ_k and i_k from the finger sensors and calculates each estimated pair $\hat{\theta}_k/\hat{\omega}_k$ (using the KF) and also the i_k from a low-pass filter, as will be explained below.

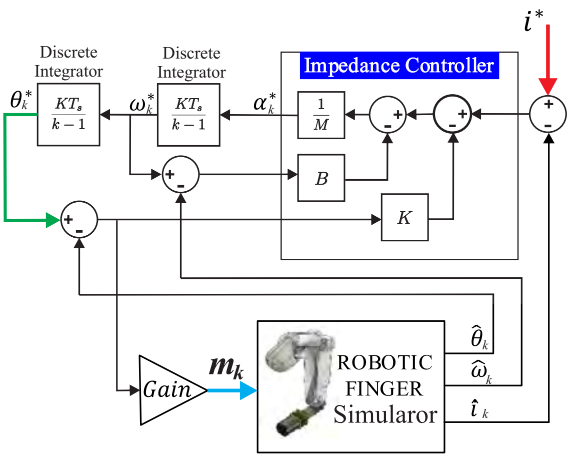


Figure 4.15 – Dataflow of finger simulator impedance controller (k=1 to 7). The colored lines highlight the data-path that the control interface override for testing (See 4.14).

4.2.3 Robotic Hand Design Validation

The present work can simulate human grasping motion despite DoF reduction (Degrees of Freedom). This is proved, doing experiments with the The Cutkosky taxonomy [19], which is a human grasping classification that has been extensively used for manipulation and machining tasks. The different types of grasping poses are classified into power and precision grasps. The experiments are executed by this work's robotic hand by maintaining an object's static grasping pose from the taxonomy.

Figure 4.16 shows the robotic hand performing some poses from said taxonomy. The robotic hand can perform five power-grasp and four precision-grasp successfully, demonstrating its ability to grasp and manipulate some objects, enabling a vast amount of applications. These results also show the relevance of the *TMC_aa* for most grasping poses.

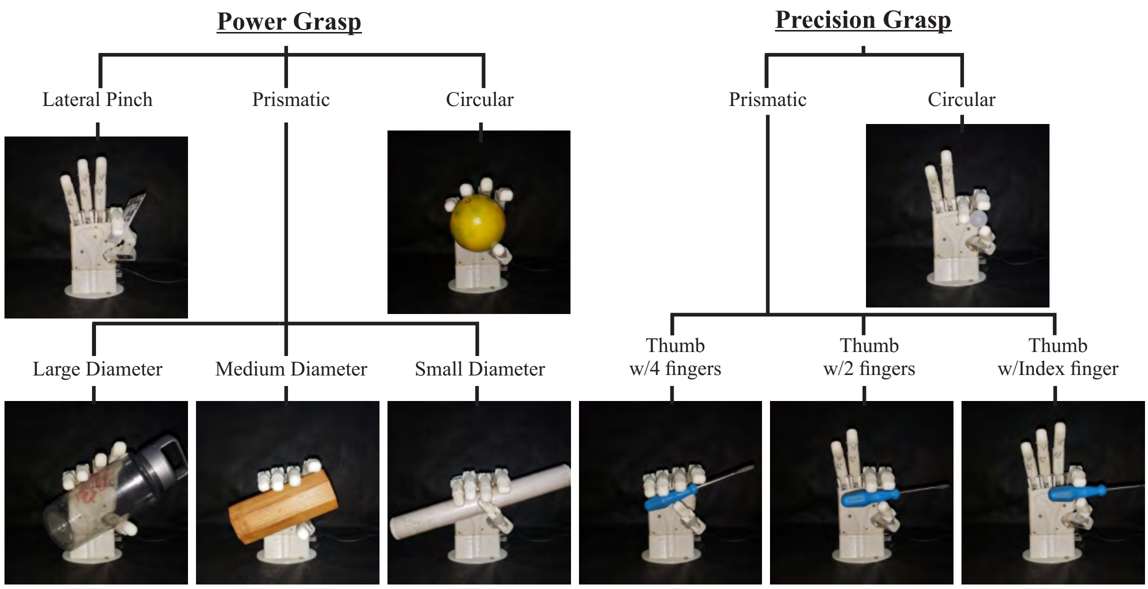


Figure 4.16 – Robotic hand performing grasping poses according to the Cutkosky taxonomy. See video

4.2.4 Integration of Robotic Hand and MYKI Interface

The goal of the integration of these experimental setups is to control the robotic hand developed in the LEIA lab with the MYKI interface data. A SoC-based solution to integrate the acquisition unit of the MYKY interface to the FPGA-based models to track the magnets was already developed and will be presented in Chapter 5.

A technical visit to the Biorobotics Institute at the Scuola Superiore Sant'Anna will allow for implementing the integration proposal, performing experiments and collecting new data. Initially, some data have been collected from the mockup to develop the initial models described in Chapter 5. They are six datasets that comprise the movements of a single magnet at a 50 ms sample-rate. Four of them are ramp movements; this is, the magnet is driven to its maximum length (approximately 10 mm) forward and back once for each dataset, during 20, 40, 60, and 80 seconds. The fifth dataset is a sequence of 10 steps across the traveling distance of the magnet. Finally, the last one is more complex; it is a movement that follows a composed sine wave; it was composed of 10 waves of different amplitudes and frequencies ranging from 0.1 to 0.5 Hz.

The previous datasets are artificial movements to tests the MYKI interface; however, to test the integration of the interface with the LEIA robotic hand, other datasets are meant to be taken. As an aditional (optional) contribution it is also planned to sample the movements of the magnets for when a hand is grasping various objects with different geometries using an electronic glove that tracks the actions of the fingers. The data obtained from the glove is translated to the forearm mockup and collect the reading of the magnetic field sensors. This data is meant to be collected offline. Finally, the MMs data is interpreted by the localizer with the implemented models and will be used to control the robotic hand. It must be considered that this task will be performed only if the time during the visit allows to do this.

4.3 RTRLIB

RTRLib is a model-based platform-oriented design tool for DPR systems. The tool is developed in Matlab/Simulink as a high-level development framework allowing a more flexible design of DPR systems. RTRLib exploits Simulink's "Variant Systems" functionality granting non-expert users the possibility to produce DPR solutions, see Fig 4.17, [43].

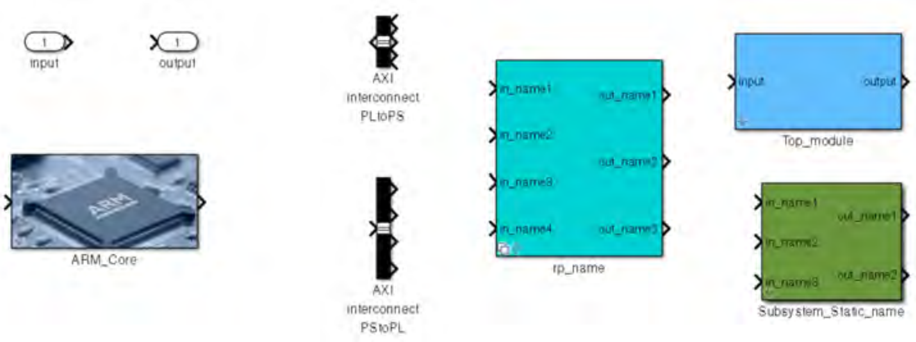


Figure 4.17 – Some Functional Blocks of RTRlib in Simulink using it “Variant Systems” feature. Adapted from [43]

The tool is based on semi-formal refinement-by-replacement methodology introduced in [1], enabling to simulate and analyze at design time the behavior of the system during the reconfiguration process. Furthermore, RTRLib provides a script generator that allows some

DPR solution to be mapped automatically on FPGA-based System-on-Chips (SoCs), specifically for the Xilinx Zynq 7010 and Zynq 7020 devices. The generator is based on several pre-characterized IP-Cores available on the FPLib [66, 66]. Therefore, to estimate the size of each Reconfigurable Part (RP), the designer must implement the high-level description of the Reconfigurable Modules (RMs) using the available IP-Cores. Thus, the tool translates the content of the RMs into its HDL description.

The tool has been tested with Matlab (2016a) and Vivado (2016.4/2017.4). Figure 4.18 shows a block diagram of the main flow phases; each one of them will be depicted within this section.

4.3.1 RTRLib Workflow

The design flow of RTRLib has four phases: *highlevel framework*, *hardware redundancy design methodology*, *PR hardware design flow*, and *PR software design flow*. Figure 4.19 illustrates the overall RTRLib design flow.

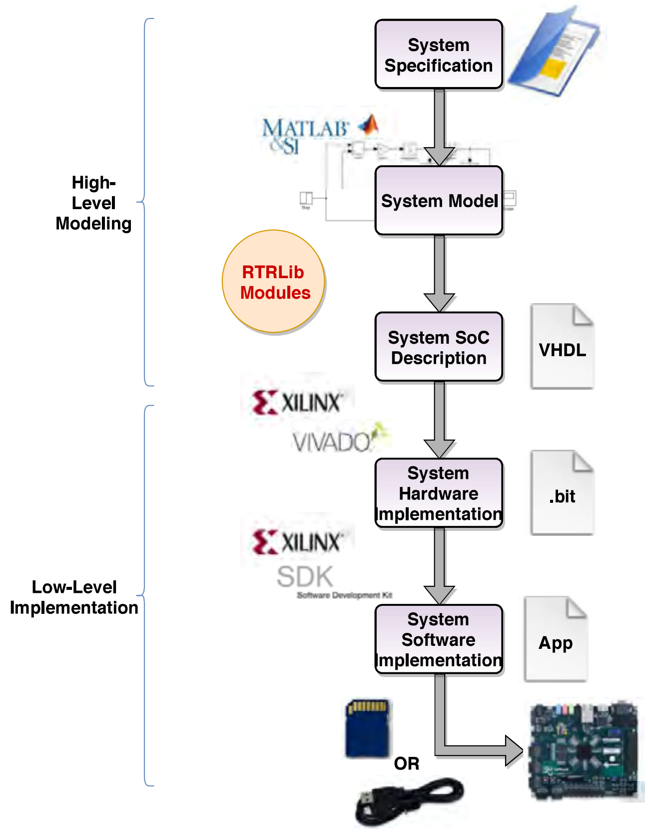


Figure 4.18 – DPR methodological work-flow diagram using RTRLib. “RTRLib modules” refers to the parts of the work-flow that are part of the RTRLib tool.

The first is where the system’s high-level description is developed. Here, the designer must provide all the details related to the desired system. It is possible to model hardware-only systems as well as SoC solutions using the ARM Cortex A9 processor embedded in the supported Zynq devices. In this phase, the user must instantiate and configure the Top Module Block, selecting the target board (Zedboard or Zybo) and the reconfiguration strategy between these three options:

- PRC (Partial Reconfigurable Controller) + ICAP: This strategy uses the PRC IP. It works with physical triggers to indicate an event, thus pulling the partial bitstreams from memory, delivering through ICAP.
- PCAP + ARM: This strategy implements software-only triggers events. In this case, the ARM controls the reconfiguration process through the PCAP.
- Manual: For this case, the reconfiguration is controlled using the Vivado Hardware Manager Tool through JTAG without intermediate controllers.

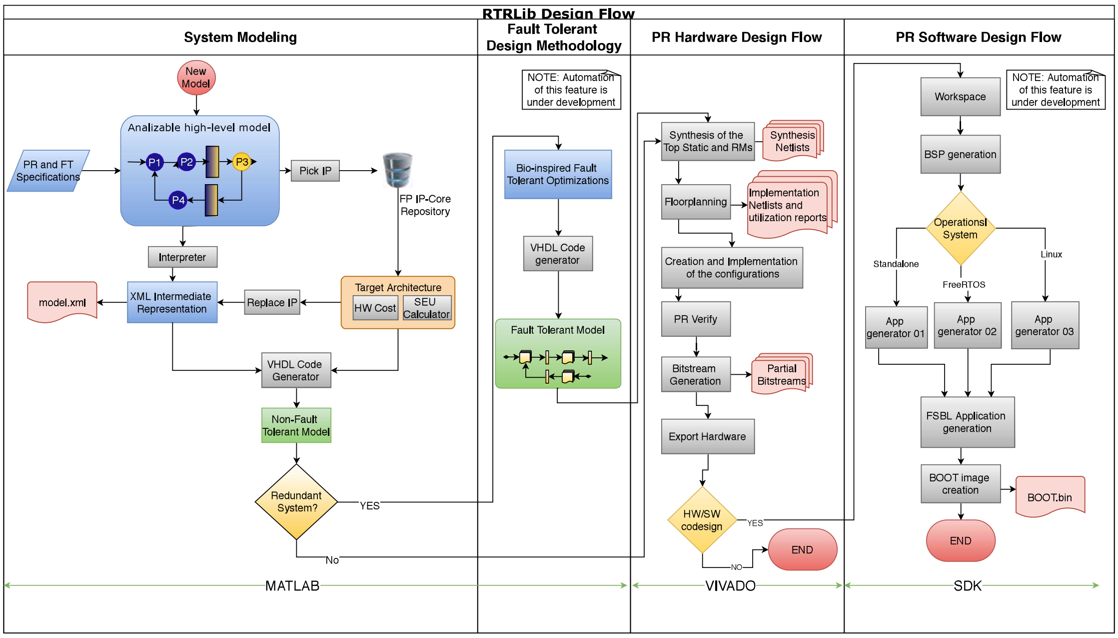


Figure 4.19 – Overview of the RTRlib design flow. Blue rectangles represent high-level models. Green squares represent fault-tolerant models. Tasks related to the design transformation and characterization, such as the automatic VHDL code generation, are depicted by gray boxes. Finally, the output files are highlighted in magenta.

For the hardware redundancy design phase, RTRLib implements a fault-tolerant design methodology based on the structural design of an NMR solution [24]. The designer provides the application details, such as the target FPGA platform and the mission time. With this data and the resource utilization estimation of each RP, the system reliability is projected and delivered by the tool. Two bio-inspired methodologies (mono and multi-objective) are in charge of finding the optimum redundancy level for each stage assessing the optimal number of replicas for each redundant module.

Consequently, and unconstrained by the existence of redundancy, the built model is dragged from the previous phases to the next one initiating the PR hardware design flow. The main output of this phase is a TCL-script that is later used in the Vivado TCL console environment. The script assembles the low-level system automatically according to the specifications provided by the designer.

Lastly, the PR software design flow phase is triggered if the solution requires a software part. For the exploitation of the software part of the Zynq SoC, RTRlib currently contemplates two choices when developing an application: C/C++ standalone and Xilinx-compatible FreeRTOS823. The user must select which software option will be used in the first phase, in the configuration of the RTRLib ARM block. Depending on the chosen alternative, the output of this phase may be a template that allows the partial reconfiguration through software triggers (C/C++ standalone). However; if the user decided to use FreeRTOS, early in the process, RTRLib provides a visual interface to assemble personalized applications.

4.3.2 Limitations of RTRLib

Table 4.1 – Limitations of the RTRLib. * Are limitations improved in the scope of this work

Hardware Features	Hardware script generalization	✓
	RPs AXI-Lite interface generator and IP packing	✓
	RPs AXI-Stream interface generator and IP packing	
	RMs IPs automatic synthesis	✓
	Floorplanning	✓
	IO mapping constraints	✓
	Power estimation	
	HDL generation for RMs	
	Resources Estimation	✓
Software Features	Software script generator	✓
	Application Type Selection	✓
	Standalone application generator	✓
	FreeRTOS application generator	✓
	Linux application generator	
PR Features	Decoupling *	✓
	Isolation	
	Relocation	
	Bitstream Compressing	
	Computation of the reconfiguration time *	✓
	Data dependency between RPs	
FT Methodology	Integration with the RTRLib Design Flow *	✓
	SEU Estimation	

RTRLib is still under development; currently, it works under particular conditions for some applications, and some functionalities are not yet enabled. Table 4.1 presents a set of some functionalities that are not yet implemented on the RTRLib, as well as the features that were already implemented. It is the aim of this work to improve some of RTRLib's limitations, such as: a) AXIStream and AXIDMA for IP packaging; b) Estimation of the reconfiguration time.

4.3.3 RTRLib on MYKI localization problem

Figure 4.2, depicts the myokinetic current data acquisition and localization scheme. As mentioned earlier, the PIC μcontroller takes 21.0 ms to collect 96 sensor readings sequentially. This is relatively slow given that the MAG3110 is rated to feed data at 12.5 ms [29]. Thus, if the AU could be rearranged in a way to have more i2c lines, changing, without compromising the number of devices involved, the full data rate of the sensors could be exploited, see Fig 4.20. Taking into account these timing restrictions, as well as the timing specifications of the prosthetic hand, the control algorithm executed by the control unit and the number of i2c lines must be selected to optimize the overall execution time and power consumption of the system.

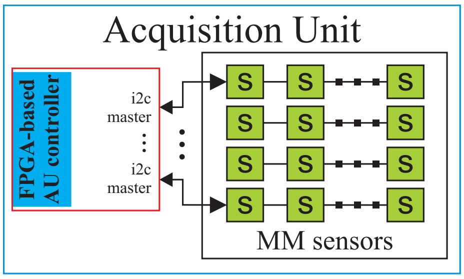


Figure 4.20 – Acquisition Unit with several i2c lines and a new.

In that context, a DPR solution can be used for different purposes:

a. Switching between different control strategies of the prosthetic hand. Currently, the LEIA robotic hand can be controlled using different control strategies, such as impedance controller, PID, and manual control (and other strategies that can be implemented in the future)—being the first, the more complex, with more prominent resource utilization than the others. Switching between these architectures, sacrificing some control efficiency could potentially reduce the power consumption of the FPGA-based control system when needed.

b. Swapping the myokinetic interface machine learning models. The goal of the myokinetic interface is to track a minimum of 5 magnets (5 DoF), every implanted magnet will need a model predictor, and these can be different. It is planned to train every magnet individually an together to extract a model to localize each. Because the five models may not fit in the FPGA, if implemented using static configuration, a solution for this problem could be the swapping between the models using DPR, potentially being able to move to a smaller, with less energy consumption, cheaper FPGA.

c. Adapting the myokinetic interface according to the timing constraints of the acquisition board and robotic hand controller unit. The hardware predictor models' execution performance depends on their complexity, and all of its computational power may not bee needed at some circumstances. It is proposed to switch to less complex architectures when the systems require a solution with less energy consumption. For example, reconfigure to a linear model with fewer operators leads to a smaller throughput, but fewer resources lead to the use of less energy.

5 MACHINE LEARNING MODELS IMPLEMENTATION ON HARDWARE

This section describes the results on the implementation of machine learning models for magnet localization in the MIKY project, namely, a Neural Network and Linear Regressors, which were implemented as hardware co-processors in an FPGA device. The following subsections define the mathematical formulation of the proposed models.

Both models are designed automatically with VHDL code generator tools (*pLinRgen* and *vRBFgen*) develop in MATLAB to easily producing different models. The tools receive configuration parameters that set the model's architecture composition, characterizing the number of arithmetical operators and the connections in between. The arithmetical operators are customized floating-point modules previously developed by Muñoz et al. [65, 66].

The models are trained with the composed sine wave movement dataset mentioned in Chapter 4. The reason this dataset is selected is that it contains the most important information about the system. The linear model is trained using the linear least-squares regression method [36] to determine its intersection and coefficients. On the other hand, the RBFNN is trained using the k-means optimization algorithm [36] to determine the RBF centers and the weights of the outputs neurons. The number of neurons was set to eight; after that, the model's accuracy did not appear to increase in the tests.

The description of the models and the configuration parameters are detailed in the following subsections.

5.1 LINEAR MODEL - PLINRGEN

The linear model predictor is implemented in FPGA as a combination of FSM (Finite State Machine) and pipeline architectures. It calculates the following equation.

$$\hat{y}_k = c_1 \cdot x_{1,k} + c_2 \cdot x_{2,k} + \cdots + c_{n_{var}} \cdot x_{n_{var},k} + b \tag{5.1}$$

where $c_1, c_2, \cdots, c_{n_{var}}$ and b are the coefficients of the model, $\hat{y}_k$ is the prediction, and $x_{1,k}, x_{2,k}, \cdots, x_{n_{var},k}$ are the input variables in the instant k.

The code generator configuration parameters are:

1. $nbits$: Is the bit-width of the floating point representation for the arithmetical operators

(adders and multipliers).

2. n_{var}: Indicates the number of variables the linear model has.

3. $c_1, \cdots, c_{n_{var}}$ and b: see Eq. 5.1

4. n_{op}: Is the number of operators executed in parallel. The number of stages of the pipeline architecture and the arrangement of the operators depends on this parameter (see Fig 5.1(b)).

The input/output ports of the linear model predictor are depicted in Fig. 5.1(a). As expected, the quantity of inputs depends on the n_{op} parameter. On the other hand, Fig. 5.1(b) illustrates a possible configuration for the linear model predictor with $n_{op} = 4$ and 4 levels of deepness (n_{stages}). The latter parameter can be determined by the following equation:

$$n_{stages} = \log_2(n_{op}) + 2 \tag{5.2}$$

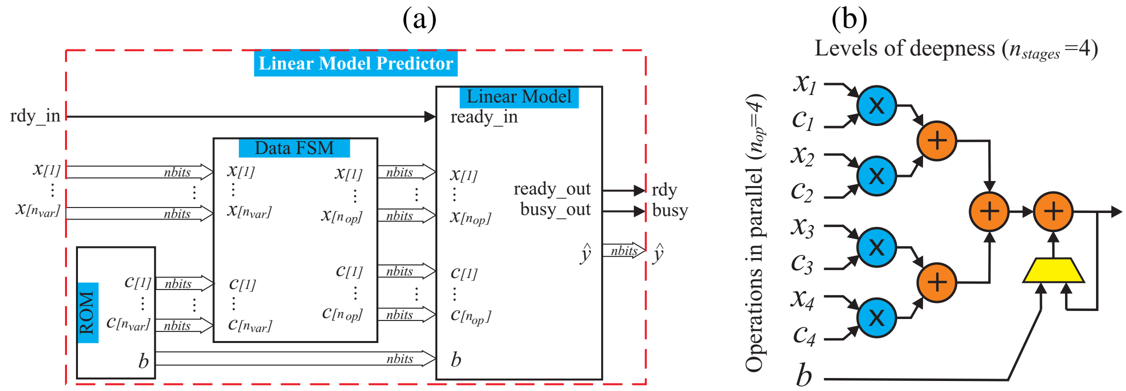


Figure 5.1 – Linear Model Predictor on FPGA. (a) Top level view of system. (b) Internal pipe-lined architecture of "Linear Model" block.

The "Data FSM" block is a finite state machine that synchronously feeds the variables and coefficients to the "Linear Model" block. When $n_{var} > n_{op}$ the FSM feeds the data in the correct sequence for $n_{cycles} = \overline{n_{var}/n_{op}}$, being the number of cycles necessary to carry out a prediction. This sequence is illustrated in Fig. 5.2 as a timing diagram with the states of the values for the inputs and outputs of the "Linear Model" block.

The latency in the diagram is depicted as the measured time between the first start pulse ("ready_in") and the first output ready pulse ("ready_out") of the system. It can be calculated as stated in equation 5.3, where T_{clock} is the global clock period, and the latency is measured in μseconds.

$$Latency = n_{cycles} \cdot n_{stages} \cdot t_{op} \tag{5.3}$$

where t_{op} is the time the operator modules require to carry out a result and is expressed as $2 \cdot T_{clock}$.

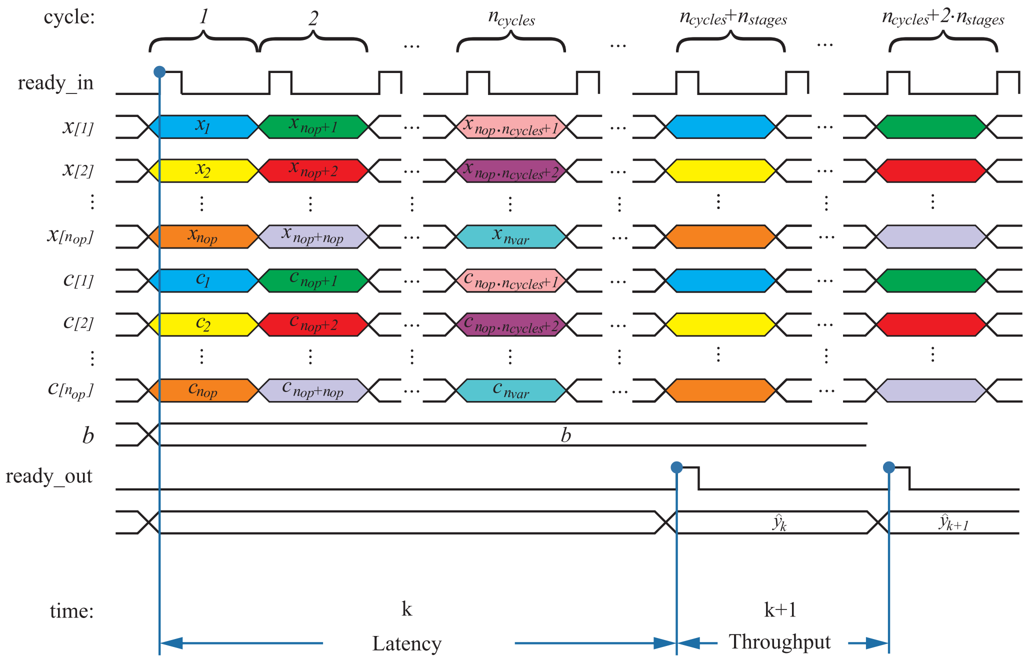


Figure 5.2 – Input/Output data time diagram of linear model in FPGA. The "ready_in" input signals the start of the process n_{cycles}-times. The "ready_out" output signal announce when a prediction is calculated.

Similarly, the throughput is the output data rate and can be measured in MFLOPS (Million of Floating Operations per Second). It is estimated as the inverse time between two consecutive output ready pulse ("ready_out"). However, because this hardware model has an FSM and a pipelined architecture working together, the traditional estimation of the throughput for a pipeline, equals to $\frac{1}{2 \cdot T_{clock}}$, does not reflect the real rate of this particular case. For this case, the throughput depends on the n_{var} and n_{op} variables and is estimated by the equation 5.4.

$$Throughput = \frac{1}{(n_{cycles} \cdot n_{stages} - n_{stages} + 1) \cdot t_{op}} \tag{5.4}$$

The latency and throughput are used to measure the computational performance in the following sections.

5.2 RADIAL BASIS FUNCTIONS ARTIFICIAL NEURAL NETWORK (RBFNN) MODEL - VRBFGEN

The RBFNN model is also implemented as a hardware architecture in FPGA using a FSM described in VHDL. The output of the RBFNN is mathematically defined in equation 2.6. The code generator implements the RBFNNs with the following configuration parameters:

1. $nbits$: Is the bit-width of the floating point representation for the arithmetical operators.
2. n_{var}: Is the number of inputs.
3. M: Is the number of neurons.
4. $\boldsymbol{c}$: Is an $M \times n_{var}$ matrix with the centers for every neuron.
5. δ: Is an $M \times 1$ array with the neurons' spreads, $1/(2\sigma^2)$ for this case, see Eq. 2.7.
6. w: Is an $M \times 1$ array with the output layer weights.

The construction of the RBFNN Model Predictor is described in Fig. 5.3(a) exhibiting: 1) The input layer composed by a "Register Array" block where the inputs are stacked, and a "ROM" block with the coefficients $c(M \times n_var)$; 2) The hidden layer with M-neurons working in parallel; and 3) the output layer that is integrated by an FSM, a ROM (containing the weight coefficients) and some operators in charge of calculating a prediction by weighing the neurons' outputs ϕ. Every "Neuron" block implements a Gaussian kernel, as seen in Eq. 2.7, using three operators: 1) an adder, 2) a subtractor, and 3) an exponential operator, differently from the Linear Model that only uses the former two. The latter is implemented as a CORDIC (Coordinate Rotation in a Digital Compute) algorithm [65]. The composition of this block is depicted in Fig. 5.3(b) and is also implemented using an FSM that controls the operations.

Since the RBFNN model and the NN blocks consist only of FSMs, its latency and throughput are equal. As said before, the NN blocks are implemented in parallel (one for every neuron of the model), and its total of inner operators are kept the same no matter the number of inputs; hence, the latency depends on the number of inputs and can be obtained using by

$$Latency = n_{var} \cdot t_{fsm} + t_{cordic} + (M + 1) \cdot t_{fsm} \tag{5.5}$$

, where t_{fsm} is the time the multiplier, adder and substractor require to execute an operation in an FSM and t_{cordic} is the time the CORDIC algorithm takes to perform an exponential operation.

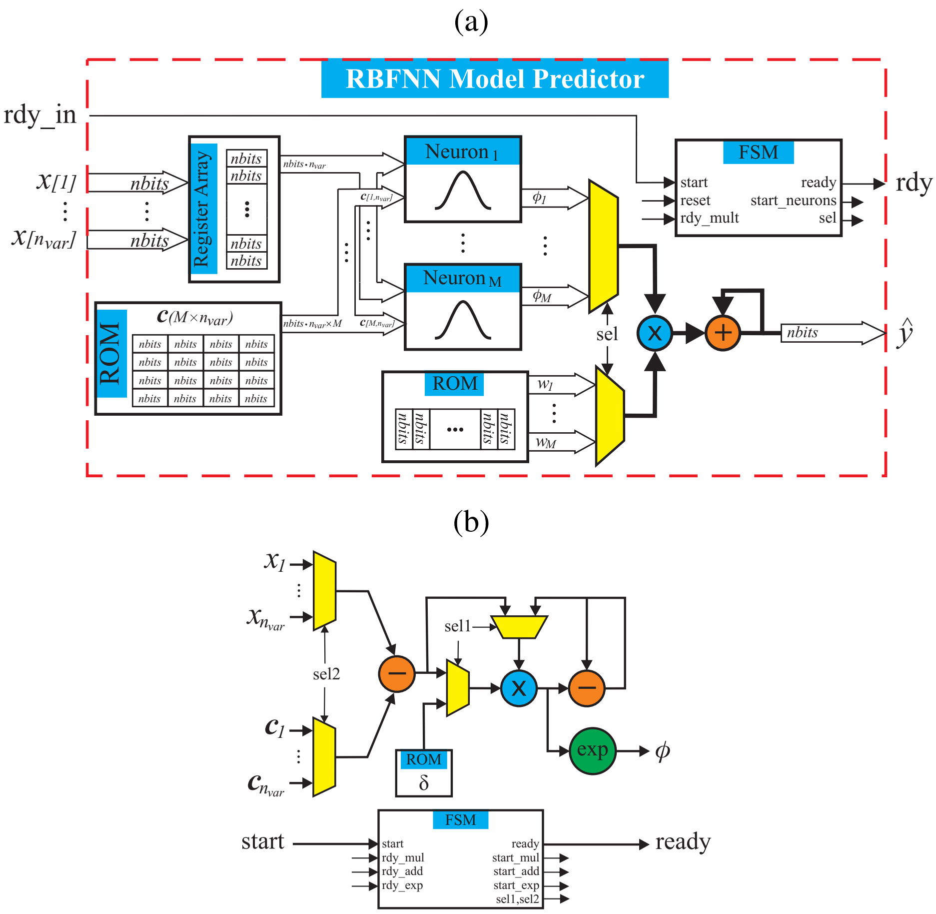


Figure 5.3 – RBFNN Model Predictor on FPGA. (a) Top level view of the system. (b) Internal FSM of radial based kernel of "Neuron" block.

The throughput is calculated using the latency with

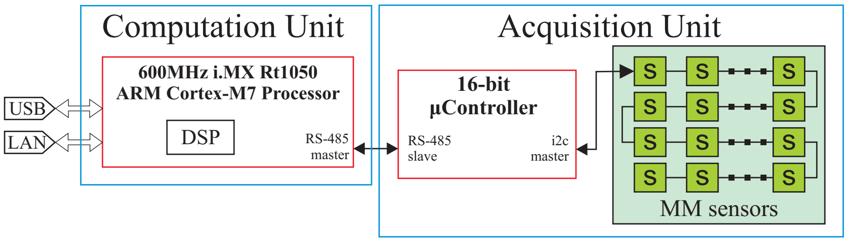
$$Throughput = \frac{1}{Latency} \tag{5.6}$$

5.3 INTEGRATION WITH THE ACQUISITION BOARD

In previous work [17], the acquisition board or "*Localizer*" involves two main parts; the Acquisition Unit (AU) and the Computation Unit (CU), see Fig. 5.4(a). The AU contains a 16-bit micro-controller that acquires data from an array of 32 three-axis magnetic field sensors through an I2C interface. The CU consists of an iMX RT1050 Processor that draws the data from the AU via UART RS-485 and makes the prediction by implementing a set of 96 equations (32 sensors $\times$ 3 axis) and solve it using a Levenberg–Marquardt algorithm (LMA) [56].

This work proposes to reduce the "*Localizer*" by joining the CU and the AU. Rather than using two Controller-Processor chip components, it is introduced a Zynq-7020 SoC that integrates a dual-core, 650 MHz, ARM Cortex-A9 processor (Programmable Software - PS) with a Xilinx 7-series FPGA (Programmable Logic - PL). Fig. 5.4(b) depicts the proposed architecture, which is implemented in the PL-part of the SoC using a custom μBlaze (32-bit RISC processor) as a master to manage the I2C interface and the model block (Linear or RBF) through AXI (AMBA eXtensible Interface - AXI is an on-chip bus architecture). Furthermore, the PS-part will use the predictions for prostheses control and AI algorithms, and, for this, the μBlaze is also connected to it over a non-exclusive FIFO mailbox block that allows bi-directional communication between the PS and PL parts.

(a)



(b)

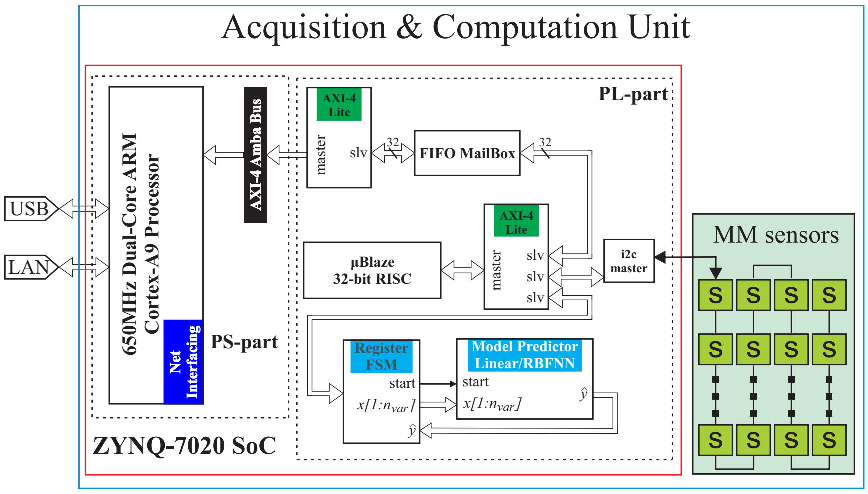


Figure 5.4 – *Localizer* for embedded MIKY sensoring system. (a) Is the overview of the architecture previously considered in [17]. The AU contains the matrix of the “S” magnetic sensors and a 16-bit μController, while the CU contains the ARM processor in charge of the prediction. (b) Depicts the proposed architecture of this article. The Acquisition and Computation are included in the same SoC chip. Note that the red lines denote a chip’s spatial limits.

5.4 FPGA UTILIZATION VS BIT-WIDTH

The FPGA’s hardware resources, such as DSP, look-up tables (LUT), and registers, are limited. In the case of this work, the FPGA’s utilization changes following the complexity of the architecture and bit-width of the models’ operators. Tables 5.2 to 5.1 describe the hardware consumption of the Linear and RBFNN model predictors implemented with different bit-width.

Table 5.1 – Hardware utilization of the extra blocks for the integration scheme.

Block	**LUT (53200)**	**DSP (220)**	**SR (106400)**	BRAM (144)
MicroBlaze	700 (1.31%)	0 ()	334 (0.3%)	8 (5.5%)
i2c Master	179 (0.3%)	0 ()	250 (0.2%)	0 ()
Fifo Mailbox	239 (0.4%)	0 ()	199 (0.2%)	0 ()
AXI-4 Lite	471 (0.9%)	0 ()	576 (0.5%)	0 ()

Table 5.2 – Hardware utilization and estimation error varying the floating-point bit-width of linear model using 8 and 128 operators (less is better in all columns)

	LUTs (53200)		**DSP (220)**		**SR (106400)**		**MSE [dB]**
bits	**8 ops**	**128 ops**	**8 ops**	**128 ops**	**8 ops**	**128 ops**	
20	2126 (3.9%)	31640 (59.5%)	8 (3.6%)	128 (58.2%)	1829 (1.7%)	27997 (26.3%)	-69.860
27	3340 (6.3%)	51420 (96.6%)	8 (3.6%)	128 (58.2%)	2419 (2.3%)	37227 (34.9%)	-99.845
32	3532 (6.6%)	82021 (154.2%)	16 (7.3%)	160 (72.7%)	2869 (2.7%)	44278 (41.6%)	-129.987
38	4971 (9.3%)	158651 (298.2%)	32 (14.5%)	160 (72.7%)	3415 (3.2%)	52743 (49.6%)	-166.450
42	5569 (10.5%)	206715 (388.6%)	32 (14.5%)	160 (72.7%)	3775 (3.5%)	58383 (54.9%)	-190.136
45	6382 (12.0%)	232451 (436.9%)	32 (14.5%)	160 (72.7%)	4045 (3.8%)	62613 (58.8%)	-208.556
64	9097 (17.1%)	504736 (948.7%)	72 (32.7%)	160 (72.7%)	5665 (5.3%)	87874 (82.6%)	-304.454

Additionally, the precision of the floating-point numeric representation depends on the used bit-width. Thus, since it is essential to preserve both a good precision error and a minimum hardware consumption, a compromise must be made between these two aspects. The numeric error of every case and model can be seen in the previous Tables and Fig. 5.5.

It was decided to proceed with the 27 bit-width of the floating-point operators, pointing out the following reasons:

1. Because, as seen in Fig. 5.5, after 27 bit-width, there is not much improvement in the MSE for the RBFNN model.

2. Similarly, the architecture with a bit-width greater than 27 does not fit in the selected Xilinx 7-series FPGA.

Table 5.3 – Hardware utilization and estimation error varying the floating-point bit-width of RBFNN model with 8 parallel neurons (less is better in every column)

bits	**LUTs (53200)**	**DSP (220)**	**SR (106400)**	**MSE [dB]**
20	50327 (94.6%)	9 (4.1%)	13443 (12.6%)	37.636
27	67065 (126.1%)	9 (4.1%)	16936 (15.9%)	-56.858
32	78877 (148.3%)	18 (8.2%)	19442 (18.3%)	-85.826
38	93481 (175.7%)	36 (16.4%)	22435 (21.1%)	-100.046
42	103385 (194.3%)	36 (16.4%)	24451 (23.0%)	-99.926
45	111770 (210.1%)	36 (16.4%)	25952 (24.4%)	-99.909
64	163417 (307.2%)	81 (36.8%)	35451 (33.3%)	-99.906

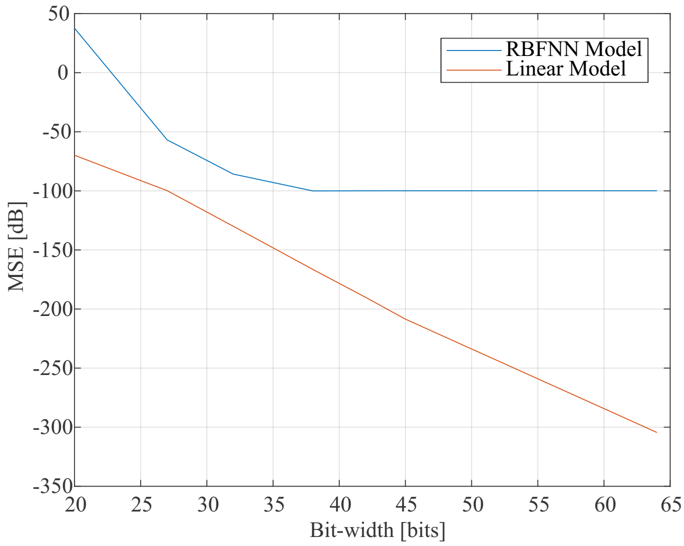


Figure 5.5 – MSE (Medium Square Error) in dB vs Bit-width for the RBFNN and Linear models

Lastly, Fig. 5.6 illustrates a prediction example of a 20 seconds ramp movement using both models.

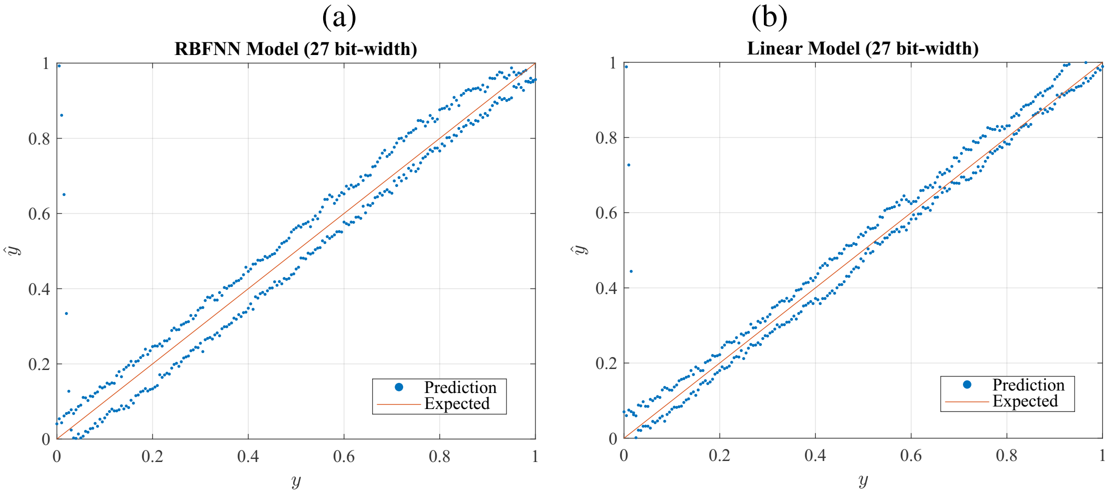


Figure 5.6 – Prediction error for RBFNN and Linear prediction models implemented on the FPGA using 27 bits.

5.5 MODEL TIME PERFORMANCE RESULTS

Table 5.4 compares the execution-time performance results of the models. The Linear Model compares three different layouts according to the n_{op} configuration parameter due to it being the only parameter that affects the timing. Differently, as said previously, the RBFNN model only has a single configuration timing considering that the only parameter that affects its timing is the number of inputs n_{var} (see Eq. 5.5).

Table 5.4 – Execution Time of Model Predictors

Model Type	N_{op}	Neurons	Execution Time Latency [μs]	Throughput [MFLOPS]
Linear	2	-	11.52	0.09
Linear	8	-	4.80	0.21
Linear	128	-	0.54	2.63
RBFNN	-	8	12.07	0.08

In Table 5.4 it can be seen that the $128N_{op}$-Linear Model dramatically outperforms the RBFNN Model. We can consider this a fair comparison since the hardware consumption of both models is similar according to Tables 5.2 and 5.3.

Nevertheless, another outstanding result is the $2N_{op}$-Linear Model due to its similar timing performance with the RBFNN Model using much fewer hardware resources. This result is remarkable because it is a much more suitable approach for this system, given that the sensory data comes from a serial RS-485 port; so, it makes more sense to deal with the data at the same pace it is being fed from the source.

The $8N_{op}$-Linear Model would not correspond to the previous reasoning; however, this solution could also be noteworthy if the sensory board could feed the data more than a single line. If, for example, it could be feed from eight data lines, the full potential of this model could be exploited.

Moreover, these results and models are implemented to localize a single magnet. The linear regressor's superior performance could be reversed when introducing several magnets and moving them at the same time. The interference of different magnetic fields could difficult for the linear model to pinpoint separate magnets. It is expected that the RBFNN model will prove its flexibility to detect several magnets in the following experiments accurately.

5.6 POWER CONSUMPTION AND IMPLEMENTATION FEASIBILITY

In order to test the feasibility of the models, Fig. 5.7 represents the chip utilization of the implemented models.

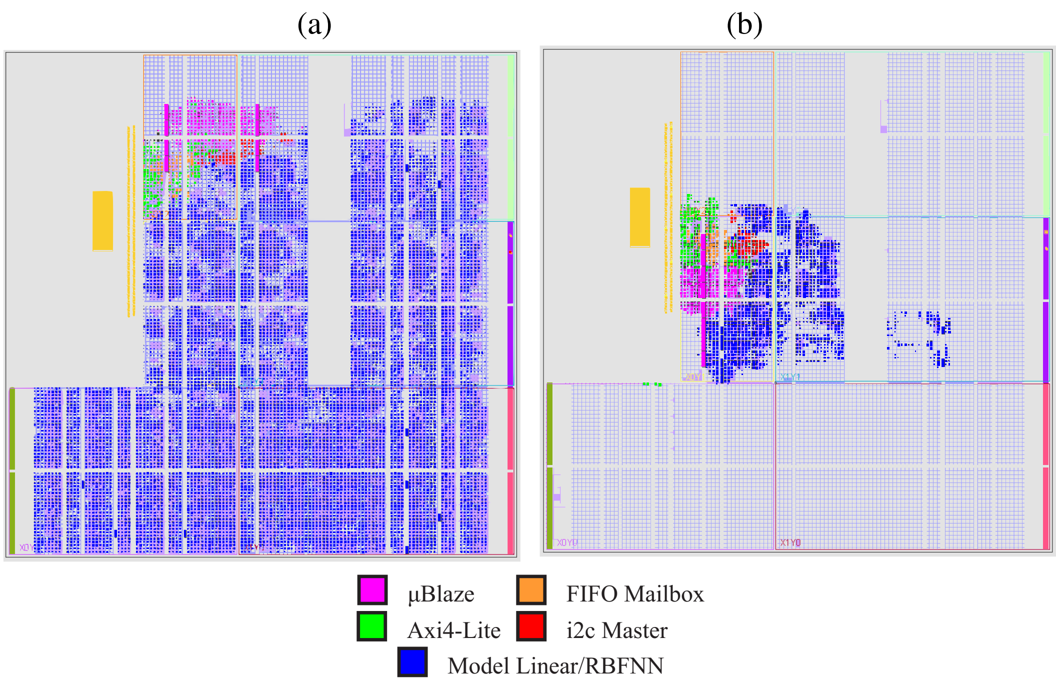


Figure 5.7 – Device overview of the FPGA utilization for both models of the complete system depicted in Fig. 5.1. Every module is highlighted with a different color. (a) RBFNN Model ($M = 8$). (b) Linear Model ($N_{op} = 8$).

Finally, the models' power consumption estimation is described in Table 5.5. The extra blocks are included, and they do not change as they are independent of the models. At the left of the table is the total consumption of all the systems.

Table 5.5 – Power consumption of prediction models scheme.

	μBlaze [mW]	Axi4 [mW]	Mbox [mW]	i2c [mW]	Crtx-A9 [mW]	Sensors [mW]	Model [mW]	Total [mW]
Linear	18	5	1	2	628	200	39	893
RBFNN							332	1186

5.7 CHAPTER CONCLUSIONS

In this chapter, it was validated the hypothesis that machine learning can be used for developing data-driven magnet localization approaches, as both models presented results with R^2 close to unity. The linear model presented overall better results in terms of accuracy and hardware consumption compared to the RBFNN model regarding model prediction accuracy. However, the RBFNN model showed a smaller static error, critical for such biosignal interpretation models.

The linear model demands significantly fewer hardware resources when we optimize the architecture by taking advantage of the model's inherently parallel structure. Hardware optimization was further applied for both models by tuning the bit-width numerical representation for the floating-point operations on hardware, highlighting the importance of ad-hoc floating-point precision hardware architectures. Similarly, more results are needed to test the adherence of machine learning models for more complex behavior observed when using several magnets for tracking different muscles at the same time.

6 TRACKING OF FIVE MAGNETS USING PROPOSED ARCHITECTURES

Indeed, the next natural step to test is the localization of many magnets concomitantly. The last chapter proved that it is possible to use machine learning to localize a single magnet. To do so, we need to perform experiments with more than one magnet and evaluate the data-driven models' accuracy in this more challenging scenario. This section describes the implementation of the previous chapter's machine learning models for several magnet localization in the MYKI project, specifically five magnets, each representing each finger's flexion-extension movements. The following subsections define how the experiments' data is pre-processed to improve the implemented ML models. This is done by performing a transformation of the data using Principal Component and Covariance Analysis. The models are then trained with the five-magnet composed sine wave movement dataset mentioned in Chapter 4.1. The training methods, as well as the model training parameters, are maintained in this chapter. However, another ML model, namely a Multilayer Perceptron (MLP), is also tested in this analysis for comparison purposes and further implementation. The dimensionality analysis and the models' training results and parameters are detailed in the following subsections.

6.1 GENERATED MODELS WITH UNPROCESSED RAW DATA

It is decided to make a different model for every magnet since it eases this system's implementation. The models are initially trained using the unprocessed input from the sensors. This is done to test how the trained models with reduced inputs perform against those who use all inputs. In this experiment, a new model is introduced: a Multilayer Perceptron (MLP) neural network. This model's architecture is two hidden layers where the first layer uses a tangential-sigmoid activation function and the second a linear activation function. The MLP also implements four neurons for this test, and it is trained with the Levenberg-Marquardt backpropagation algorithm that updates weight and bias values according to Levenberg-Marquardt optimization.

The MSE and R^2 values obtained after training the modes resulted in the following table.

Table 6.1 – MSE [dB] and R^2 rated error for models trained with raw input data. Highlighted cells represet the models that perform the best for each magnet localization prediction

Models	Linear		RBFNN		MLP	
Magnets	R^2	MSE [dB]	R^2	MSE [dB]	R^2	MSE [dB]
1	0.3006	-8.0078	0.8075	-13.6105	-0.9555	-3.5425
2	0.9863	-25.0856	0.8933	-16.1724	0.7492	-12.4624
3	0.9841	-24.4320	0.3095	-8.0632	0.9607	-20.5142
4	0.8214	-13.9363	0.8132	-13.7416	0.9013	-16.5111
5	0.9726	-22.0707	-3.1683	-0.2554	0.8146	-13.7748

Table 6.1 shows that some magnets localization is predicted better with a particular model while with others, the performance is terrible (i.e., models with negative values in R^2). This again reinsures the importance of using different types of models for this problem.

6.2 DIMENSIONALITY REDUCTION OF MAGNETIC SENSOR FEATURES USING PCA

Feature extraction techniques can reduce the number of inputs to the model, potentially reducing the hardware implementation's complexity. That could potentially reduce FPGA resource utilization and improve computational performance. The data exploration results show that a smaller amount of magnetic information can be used to create the models, which should also be tested in the multi-magnet scenario.

6.2.1 Principal Component Analysis

Principal components analysis (PCA) is based on Singular Value Decomposition (SVD), a crucial computational matrix factorization technique. SVD provides a systematic way to determine a low-dimensional approximation to high-dimensional data in terms of dominant patterns. This technique is data-driven because such patterns are discovered purely from data, without the addition of expert knowledge or intuition. It is numerically stable and provides a hierarchical representation of the data in terms of a new coordinate system defined by dominant correlations within the data and is guaranteed to exist for any matrix or dataset [7]. The SVD is a unique matrix decomposition that exists for every complex-valued matrix $X \in \mathbb{C}^{n \times m}$:

$$X = U\Sigma V^T \tag{6.1}$$

, where $U \in \mathbb{U}^{n\times n}$ and $V \in \mathbb{C}^{m\times m}$ are unitary matrices with orthonormal columns, and $\Sigma \in \mathbb{R}^{n\times m}$ is a matrix with real, non-negative entries on the diagonal and zeros off the diagonal.

PCA provides a data-driven, hierarchical coordinate system to represent high-dimensional correlated data. This coordinate system involves the matrices described in Eq. 6.1. Importantly, PCA pre-processes the data by mean subtraction and setting the variance to unity before performing the SVD. The resulting coordinate system's geometry is determined by principal components (PCs) that are uncorrelated (orthogonal) to each other but have a maximal correlation with the measurements. This theory was developed in [30].

As mentioned in Chapter 4.1, several magnetic field measurements were collected with different movements for five magnets. These measurements are as arranged into a row vector in a large matrix X. There are multiple challenges with this type of data, namely the high dimension of the data features. However, after implementing PCA to the five-magnet multisine dataset, we see from Fig 6.1 that there is significant variance captured in the first few PCA modes. Said another way, the magnetic field data is highly correlated, so that many sensors have significant overlap in their measurements from different magnets.

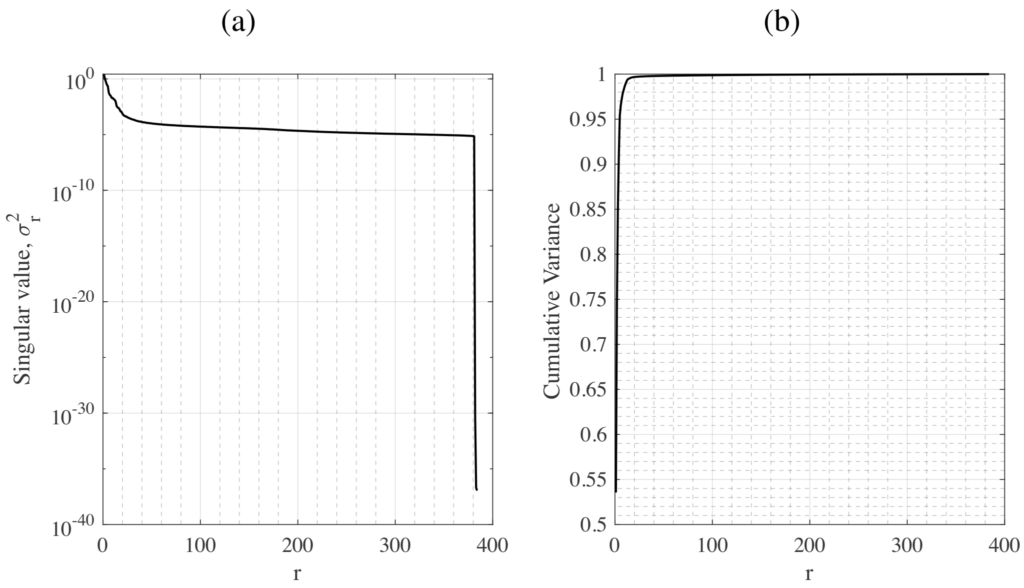


Figure 6.1 – Singular values for the five-magnet multisine dataset.

6.3 PRINCIPAL COMPONENTS TRUNCATION AND GENERATED MODELS

Deciding how many Principal Components (PCs) values to keep, i.e., where to truncate, is one of the most critical and contentious decisions when using PCA. There are many factors, including specifications on the system's desired rank, the magnitude of noise, and the distribution of the PCs. Often, one truncates the PCs at a rank r that captures a predetermined amount of the variance or energy in the original data, such as 90% or 99% truncation. For this case, the first 5 and 12 PCs represent the 90% and 99% variance, respectively. Although crude, this technique is commonly used.

Other techniques involve identifying “elbows” or “knees” in the PCs distribution, which may denote the transition from PCs representing important patterns from those that represent noise. Truncation may be viewed as a hard threshold on PCs, where values larger than a threshold are kept while remaining PCs are truncated.

Also, the alignment of data significantly impacts the rank of the PCA approximation. The PCA essentially relies on a separation of variables between the columns and rows of a data matrix. In many situations, such as when analyzing traveling waves or misaligned data, this assumption breaks down, resulting in artificial rank inflation.

In order to avoid this problem, given that we have several outputs for the same readings, a third approach is implemented for the PCs’ ranking by correlation with each input. After calculating each PC’s absolute correlation with every output (i.e., magnet displacement), the ranking is different and is depicted in Fig. 6.2.

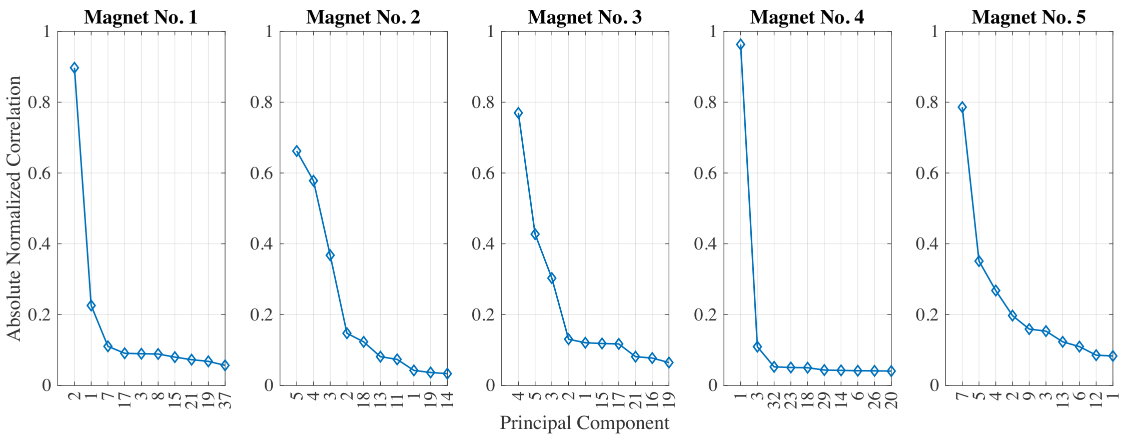


Figure 6.2 – Principal Component ranked by absolute correlation to each magnets displacement.

This individual ranking by correlation aids in selecting the most important PCs for every magnet. For example, if it is desired to train the models for *Magnet No. 1* using 5 PCs, The selected PCs will be PC_2, PC_1, PC_7, PC_{17} and PC_3. The experiments using both the PCs with and without sorting resulted in an improved result of the selected models.

After that ranking, several implementations with different truncate k values of the PCs are performed. Several tests using $k = [5\ 10\ 25\ 50\ 100\ 150\ 200]$ and their calculated the R^2 and MSE[dB] errors for all models are compared with the best result of the raw input models presented in Table 6.1. The results are in the following figures (the striped line indicates the model's error trained with all raw inputs without PCA).

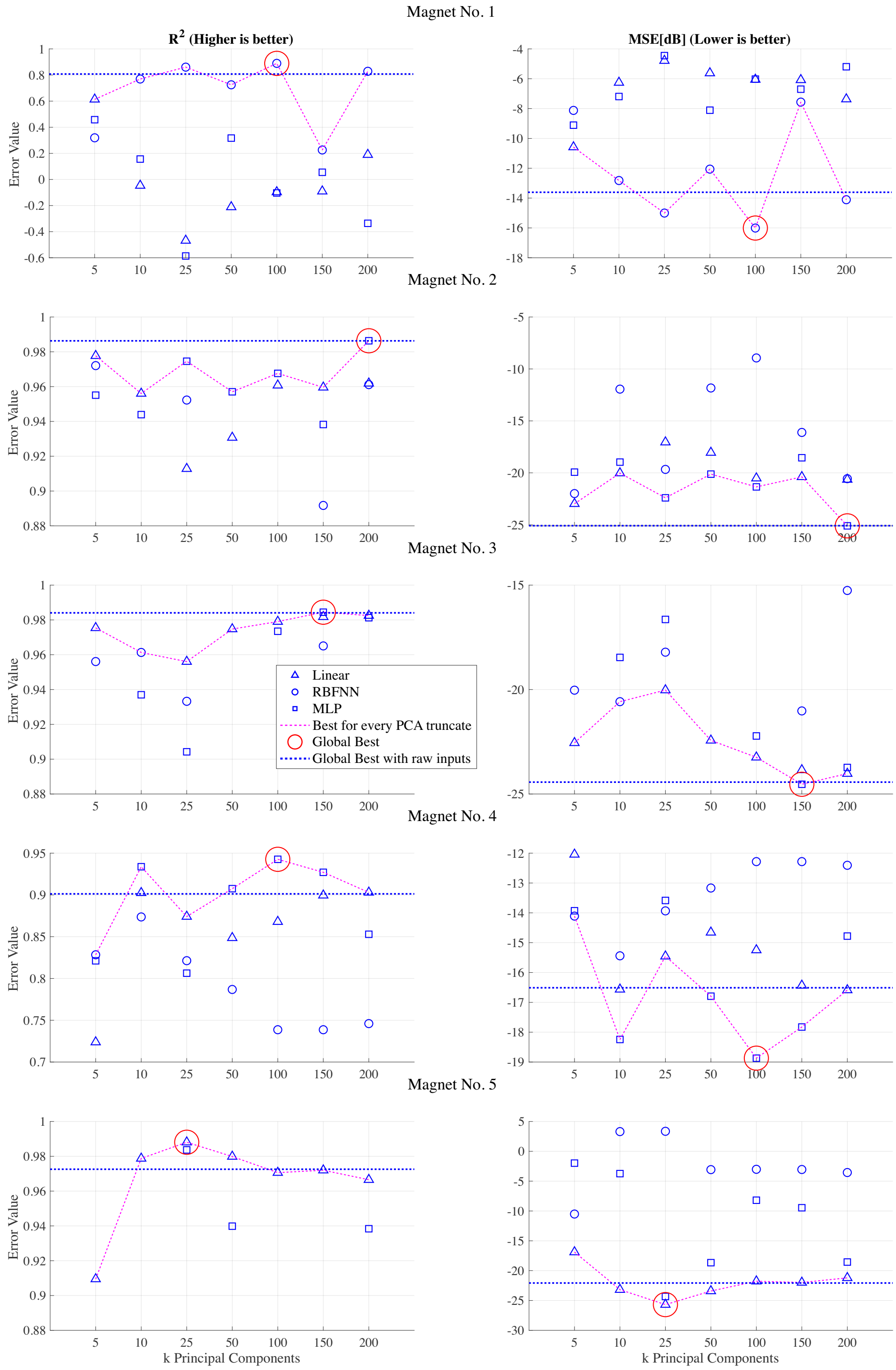


Figure 6.3 – Error-values of ML models using different truncated PC for every magnet. The x-axis represents the number of PCs used for said model (Linear, RBFNN, and MLP) represented by a symbol (△, ◯, and ☐ respectively). The dotted magenta line represents the model that performed the best for each number of used PCs. Finally, the big red circle indicates the best model using PCA.

By analyzing Fig. 6.3, it can be concluded that the best model architecturefor every magnet is as the following list:

- **Magnet No. 1**: RBFNN with 100 PCs.
- **Magnet No. 2**: MLP with 200 PCs.
- **Magnet No. 3**: MLP with 150 PCs.
- **Magnet No. 4**: MLP with 100 PCs.
- **Magnet No. 5**: Linear with 25 PCs.

6.4 CHAPTER FINAL REMARKS

In this chapter, the models subject to use for implementation are selected. The number of inputs is reduced for each magnet through a preprocessing technique using Principal Component Analysis and correlation matrixes.

It is important to point out that, in some cases, this preprocessing reduces the number of inputs and also improves the accuracy of the model. Namely, Magnets No. 1, 4, and 5 improve the model's accuracy using the raw inputs. The reason for this can be that PCA not only aids in the dimensionality reduction of a problem but also reduces unrepresentative information in signals such as white noise and others.

On the other hand, even though the selected models in the previous section's list effectively reduces the number of inputs of all models by at least half (The total number of inputs is 384). Models with much fewer PCs already perform with acceptable accuracy ($R^2 \geq 0.9$). With this in mind, these models become more interesting for implementation, given that the number of inputs significantly increases the complexity of the models.

With that in mind, because this work's final goal is to implement efficient hardware architectures that replicate these models, these solutions with lower PCs are preferred over the best performing models with bigger PC use. The final list of selected models for implementation is the following:

- **Magnet No. 1**: RBFNN with 25 PCs.
- **Magnet No. 2**: Linear with 5 PCs.
- **Magnet No. 3**: Linear with 5 PCs.
- **Magnet No. 4**: FFNN with 10 PCs.

- **Magnet No. 5**: Linear with 25 PCs.

The prediction performance of these models can be seen in the next Figure 6.4.

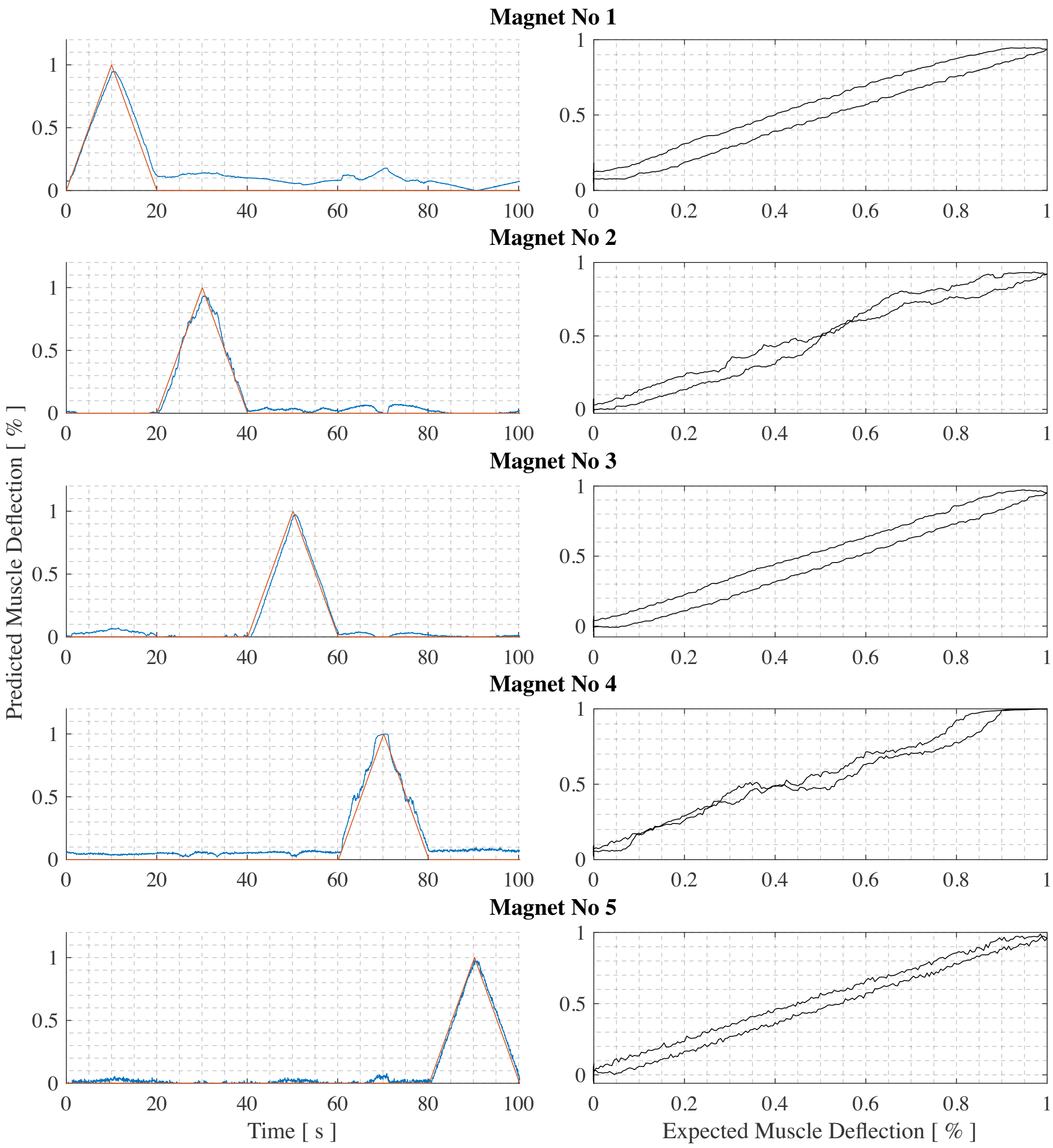


Figure 6.4 – Prediction of the selected models for hardware implementation.

7 DPR IMPLEMENTATIONS FOR MAGNET TRACKING

7.1 LINEAR REGRESSOR FOR DIFFERENT LATENCY MODELS

In this case, the scenario where the models must be adapted according to the acquisition board's timing constraints and the robotic hand controller unit is considered. The hardware predictor models' execution performance depends on their complexity, and all of its computational power may not be needed in some circumstances. Here, it is proposed to switch to less complex architectures when the systems require a solution with less energy consumption. For example, reconfiguring to a linear model with fewer operators leads to a smaller throughput, but fewer resources lead to less energy.

For this first approach in implementing the model predictor of the MYKI interface, different linear regressors are loaded into one RP, allowing the execution time of the position estimation process to be adjusted. The VHDL code of the two different linear models was generated using the pLinRgen tool, one with $N_{op} = 4$ and the other with $N_{op} = 8$. This approach selected the reconfiguration strategy using the PRC and RP connected to the ARM processor through the AXI-Lite Interconnect. The ARM processor sends the RP input data through AXI-Lite, and, in the opposite direction, the ARM reads the result of the operations through the AXI. The system is prepared to perform reconfiguration by hardware and software triggers through PRC, which controls the ICAP. Figure 7.1 RTRLib's the system's high-level model, with the connections between the RP and the ARM processor.

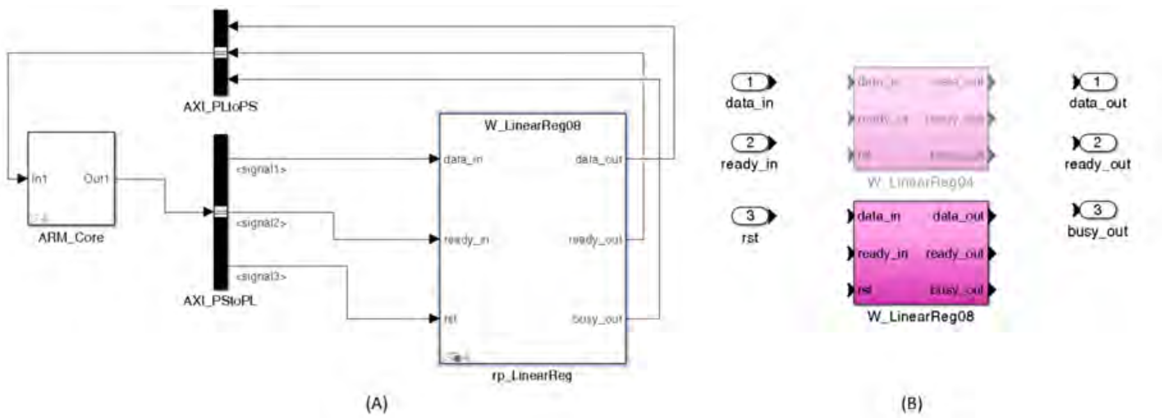


Figure 7.1 – High-level model of linear model predictor DPR implementation (A) Detail of the system-level model using the RTRLib, showing the RP, AXI, and the connections. The RP data input is sent to the ARM processor through the UART port (not depicted in the Figure). (B) Reconfigurable Modules of 4 and 8 operators.

Figure 7.2 shows the block design created through the script generated using RTRLib, which has the IPs related to the RPs, the ARM-Core processor IP, PRC, and the AXI Interconnect. Figure 7.3 shows the obtained circuit layout with four operators in the RP.

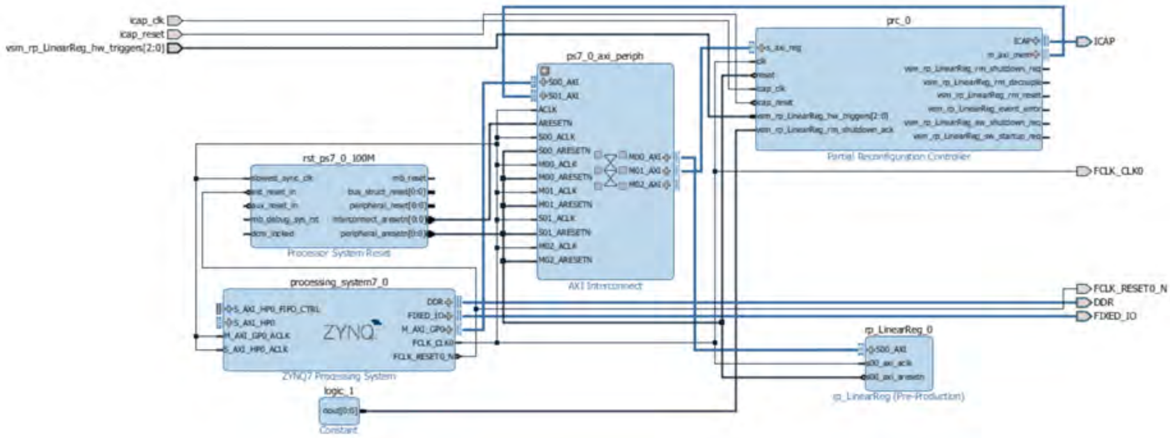


Figure 7.2 – Vivado block-based overview of the implemented system illustrating the RP, PRC, AXI interconnect and Zynq Processing System.

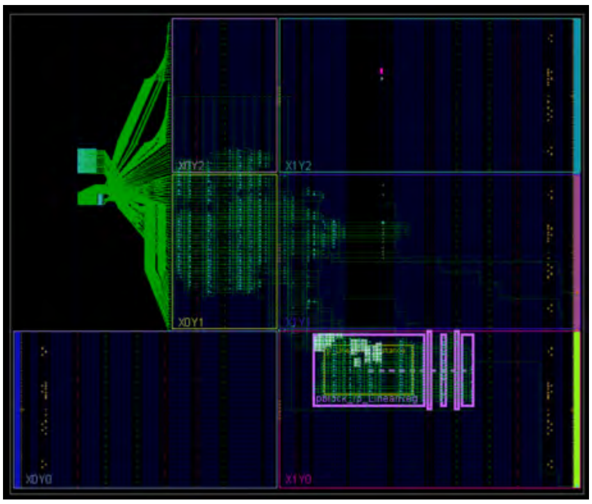


Figure 7.3 – Device implementation view of the DPR system including the static and reconfigurable partitions. The in this view the RP details the $N_{op} = 4$ version of the linear regressor.

7.2 TRUNCATED PCA MODELS FOR FIVE MAGNETS

In this case, the previously developed models with dimension reduction are implemented using DPR to swap the different ML models to track five magnets simultaneously on the target device PYNQ-Z1. As seen in the previous chapter, every implanted magnet will need a different model predictor. After training the models separately, they will be implemented separately, and the generated architectures will be used to localize each magnet in real-time. Because the five architectures may not fit in the FPGA if implemented using static configuration, a solution for this problem is swapping between them using DPR, potentially consuming hardware resource and energy.

Firstly, it must be mentioned that some modules have not been implemented up to this point. These modules are built using Vivado HLS with half-precision floating-point representation. They will briefly be described next. The first one is the architecture to accelerate the matrix multiplication used to calculate each measurement's PC components. This architecture subtracts the mean of each measurement and multiplies the result to the truncated $U\Sigma$ matrix described in Eq. 6.1. This is implemented as a pipeline in order to reuse operators in the FPGA. On the other hand, this architecture remains in the static area of the FPGA.

The other module is the MLP model. This module is implemented with a similar architecture to the RBF (See Fig. 2.4), where the neurons are implemented in parallel. Additionally, similar to the RBFNN custom architecture, it is also implemented using state machines. This module is implemented in HLS.

Table 7.1 – Hardware utilization of modules for the five-magnet tracking system.

	Block	**LUT (53200)**	**DSP (220)**	**SR (106400)**	BRAM (144)
	AXI-4 Lite	471 (0.9%)	0 ()	576 (0.5%)	0 ()
	i2c Master	179 (0.3%)	0 ()	250 (0.2%)	0 ()
	Decoupler	10 (0.0%)	0 ()	0 (0.0%)	0 ()
	DMA	1598 (3.0%)	0 ()	1975 (1.9%)	0 ()
	SMC	1598 (4.6%)	0 ()	1975 (1.9%)	0 ()
	Matrix Mult.	630 (1.2%)	4 (1.8%)	1086 (1.0%)	0 ()
Reconfigurable Modules	**Magnet No. 1** RBFNN 25 PCs	10616 (19.9%)	9 (4.1%)	2969 (2.8%)	0 ()
	Magnet No. 2 Linear 5 PCs	2736 (5.1%)	8 (3.6%)	1880 (1.8%)	0 ()
	Magnet No. 3 Linear 5 PCs	2721 (5.1%)	8 (3.6%)	1856 (1.7%)	0 ()
	Magnet No. 4 MLP 10 PCs	7188 (13.5%)	16 (7.3%)	6354 (6.0%)	0 ()
	Magnet No. 5 Linear 25 PCs	2779 (5.1%)	8 (3.6%)	1924 (1.8%)	0 ()

On the other hand, the custom RBFNN and Linear architectures were implemented using the code generator depicted in Chapter 5. This time, the used custom floating-point representation was 21 bits. After a precision analysis, this value is decided after it was proved that it provided the best accuracy/resource ratio that allowed the modules' to fit in a single clock region, a critical design parameter to DPR design. The implementation result of the modules is depicted in Table 7.1.

7.2.1 DPR Implementation in RTRLib

In RTRLib, the DPR system design looks like this:

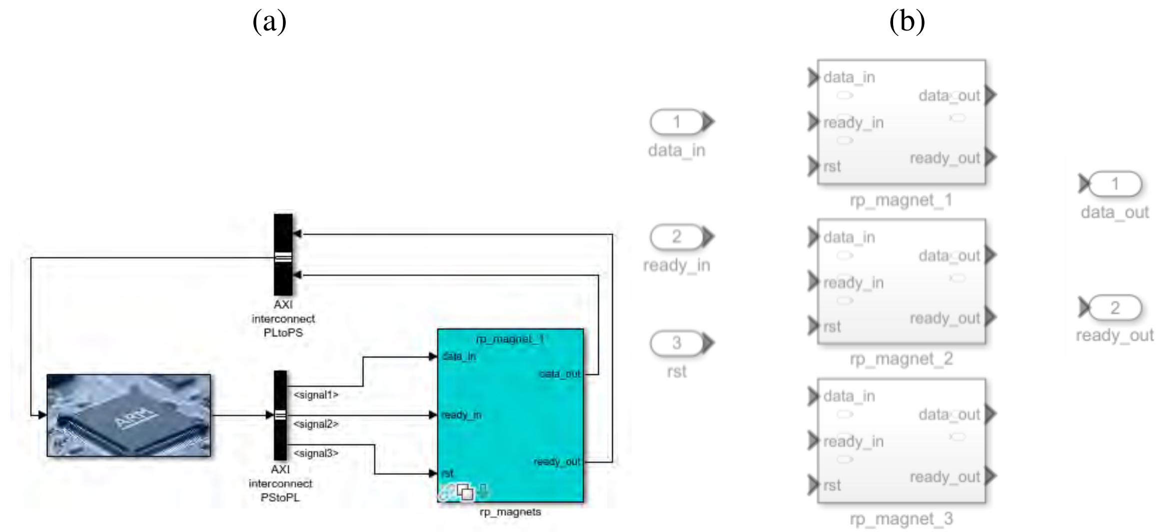


Figure 7.4 – High-level model of the models for the five magnets using DPR. (a) Detail of the system-level model using the RTRLib, showing the RP, AXI, and the connections. (b) Describes the reconfigurable modules for the five magnets.

After the implementation in Vivado, the results are depicted in the Figure 7.5.

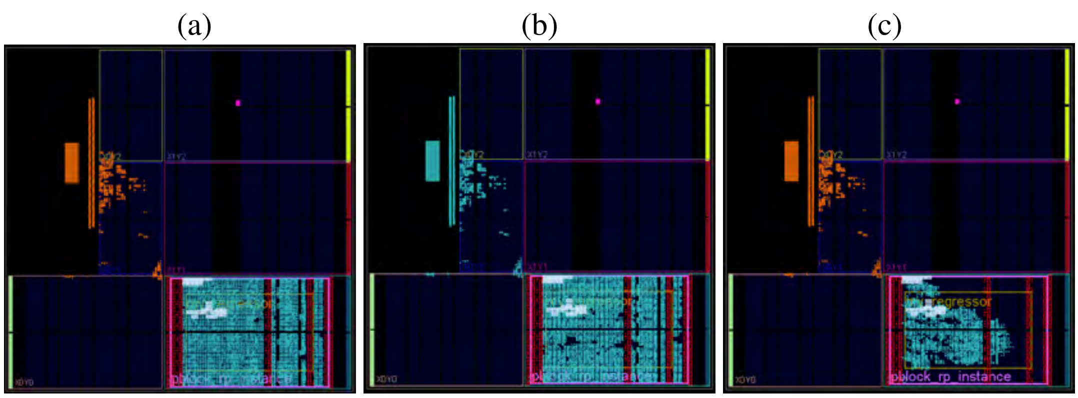


Figure 7.5 – Device implementation view of the DPR system for five magnets including the static and reconfigurable partitions. Each frame is different on every reconfigurable partition. (a) RBFNN Model (Magnet 1), (b) MLP model (Magnet 4), and (c) Linear model (Magnets 2,3, and 5).

7.2.2 Performance Results of DPR solution

The performance analysis will be considered three values: Reconfiguration time, time-execution, and energy consumption of the modules.

The first refers to the amount of time needed by the FPGA to perform the dynamic reconfiguration. It is the time needed to load a Reconfigurable Module (RM) into a Reconfigurable Partition (RP), moving its partial bitstream file from memory, particularly the DDR3 memory of the board platforms supported by RTRLib, to the reconfiguration memory of the FPGA.

The reconfiguration time depends on the amount of logic that will be written (size of the partial bitstream) and the reconfiguration method (discussed in Chapter 2). In particular, for this work, the Processor Configuration Access Port (PCAP) is used to reconfigure the Programmable Logic (PL) using a software application running on the Processing System (PS). On the other hand, it is essential to point out that the size of the partial bitstream includes the complete Reconfigurable Partition (RP) and not only the Reconfigurable Module (RM), as stated in [27]. The relationship between the reconfiguration time, using PCAP, and the bitstream file is linear and is modeled like

$$T_{reconf} = 7.7e - 3bit_{size} \tag{7.1}$$

, where bit_{size} is the partial bitstream size in *bytes* and T_{reconf} is the reconfiguration time in seconds.

Additionally, to verify the RTRLib's estimation of the reconfiguration time, it is also properly measured using one of the ARM Cortex CPU hardware timers. The same timer is used to measure the time-execution of each module. Considering that the data transaction between the PS and PL is also taken in this measurement, these values might differ from the estimated values depicted in Table 5.4. The timing results are depicted in Table 7.2

Table 7.2 – Timing result of five-magnet tracking DPR system.

	Magnet (Measured)					Reconf. time	
	1	2	3	4	5	Estimated	Measured
Time [μ]	12.07	5.1	5.1	7.5	6.1	4740.8	4742.3

Finally, the energy consumption measurement reported the results listed in Table 7.3.

Table 7.3 – Energy consumption of DPR system (measured).

Configuration	Static		Reconfigurable		
	ARM	PL Static	RBFNN	MLP	Linear
Energy [mW]	1514	162	12	23	8

7.3 CHAPTER FINAL REMARKS

In this section, the full DPR system for five-magnet tracking is implemented. The system is tested on the target device PYNQ-Z1. The results of the estimators produced by the development board were saved and compared against the software implementation. This allows us to compare the results against the *golden estimator* implemented in software using double floating-point precision. The results of the hardware models using less bit-width precision were acceptable, giving a median of $R^2 = 0.98101$ and $MSE[dB] = -22.603$ for the test datasets.

As seen in Table 7.2, the modules' execution-time combined with the reconfiguration time still meet the real-time requirement of the MYKI interface. The sensory data of the MYKI interface comes from an IIC serial line at a fixed rate of 12.5 ms (80 Hz), the NXP magnetic field sensor's maximum feed rate, which is the system's real-time requirement. An average person flex and extend his fingers no faster than 2 Hz [34] and highly skilled individuals, i.e., piano players, do it at 10.5 Hz at most [31]. Consequently, based on the Nyquist–Shannon sampling theorem, the sensors' feed rate can faithfully identify human finger gestures. If we consider that the RP must be reconfigured twice for every iteration, the total time-execution is

$$t_{total} = T_{M1} + T_{reconf} + T_{M2} + T_{M3} + T_{M5} + T_{reconf} + T_{M4} \tag{7.2}$$

, where $T_(Mi)$ is the reconfiguration time of every magnet module.

The last equation can assume that the total execution time is $\approx 9.5ms$. Keep in mind that Magnets 2, 3, and 5 uses the same RM; it is not necessary to reconfigure to calculate these results. This execution time complies with the real-time requirements of the MIKY interface.

Finally, results showed that the proposed system is more power-efficient in contrast to the previous work [17]. In that case, the AU's power consumption was 550 mW, and that of the CU was 430 mW, resulting in total consumption of 980 mW. Our solution effectively reduced power to less than 20% of that (if the ARM processor is not considered).

8 CONCLUSION AND FUTURE WORKS

This work efficiently built data-driven soft-sensors, using black-box system-identification machine learning models to translate the myokinetic sensor information to identify the voluntary motor and force commands on a forearm muscle. It dealt with it by applying machine learning algorithms to perform input selection and related model parameters for black-box nonlinear system identification. The selected models to be tested were implemented as ad-hoc hardware accelerators for magnet localization in the context of myokinetic prosthetic control. Experiments to gather relevant data of the system were carried out, and this massive data is processed for transforming and dimensionality reduction using PCA. The implementation of PCA resulted in that the first 20 principal components amassed more than 99% of the variance, which suggested a high overlapping and correlation of the data, which permitted the variable reduction on the trained models.

Since the interface's goal is to control at least five fingers, five predictors were implemented using machine learning techniques. By this, it was validated the hypothesis that machine learning can be used for developing data-driven magnet localization approaches. The models used in this work were artificial neural networks (RBFNN and MLP) and a simple linear regressor. The models presented results with R^2 close to unity. The linear model presented overall better results in accuracy and hardware consumption than the RBFNN and MLP. However, the RBFNN model showed a minor static error, critical for such biosignal interpretation models. The linear model does not perform as well for all magnets, i.e., magnets 1 and 4 perform poorly with a linear regressor, making the ANN architectures more necessary for this problem.

These models' architecture was implemented with custom floating-point operators, and the code generators implemented in Matlab generate the hardware description files automatically, complying with real-time and resource utilization constraints. After implementing the models, an efficient, partially reconfigurable system could fit in a single device, reusing the same logic to implement the different models. This DPR system is also more power-efficient than the previous work, effectively reducing power consumption to less than 20% of that (if the ARM processor is not considered).

The DPR system was implemented using RTRLib, a tool that aims to become a high-level design tool for DPR systems design under specific conditions and platforms. Some functionalities of the RTRLib were improved in this work, such as decoupling, computation of the reconfiguration time, the integration of Matlab with the RTRLib design flow, and the inclusion of AXIStream and AXIDMA for IP packaging.

Finally, as part of this work, a biomimetic robotic hand was improved using an FPGA/SoC-based (Field Programmable Gate Arrays / System on a Chip) approach using a Zynq chip Xilinx. The biomimetic hand's mechanical design contains 7 DoF (Degrees of Freedom) compared to the 24 an actual human hand has. The robotic hand's number and specific joints were selected according to the most significant DoF and how well they perform gripping objects. The robotic hand prototype could grasp objects with different geometries with power and precision grip, highlighting the robotic hand's dexterity and flexibility.

8.1 FUTURE WORKS

It was possible to highlight future research fronts to build more accurate models and computationally efficient algorithms to improve the myokinetic interface during the present work development. There is a need to develop methods and algorithms that could automate building models such that it is possible to make use of the current computational power available. The gains in time consumed for completing tasks are apparent and even more critical when the systems' complexity increases.

In future works, the following topics will be pursued concerning the procedures for improving the models of the MYKI interface:

- Compare the implemented HLS MLP neural networks with the one implemented at Master level in PPMEC.
- Optimize models using different construction and feature selection tools such as interactive stepwise regression, parameter estimation using the elastic net regularization, or Ridge regression for the linear model.
- For the ANN algorithms, weight optimization techniques may also bring better results.
- It is also of interest using other models including state-of-the-art solutions [35, 40, 11, 69].
- Create a training framework for new patients. It will be necessary to retrain the architectures for every new user or patient. It is then of importance for a framework that allows this for actual patients of the MIKY interface.
- Perform the technical visit to Sant'Anna to integrate the forearm mockup to the developed robotic hand.

Concerning RTRlib

- Include relocation on RTRlib in collaboration with TU-Dresden ADS group. This group has a custom relocation and partial reconfiguration controller added to RTRLib in collaboration with the University of Brasilia.

- Generate compressed bitstreams: RTRLib does not compress the partial bitstream; an eventual redundancy in the architecture of a given RM is not exploited to reduce the size of the reconfiguration packet sent to the reconfiguration memory of the PL. Thus, reducing reconfiguration time.

[1] ATTARZADEH NIAKI, S. H., AND SANDER, I. Co-simulation of embedded systems in a heterogeneous MoC-based modeling framework. In *2011 6th IEEE International Symposium on Industrial and Embedded Systems* (jun 2011), IEEE, pp. 238–247.

[2] AYALA, H. V. H., MUÑOZ, D. M., LLANOS, C. H., AND DOS SANTOS COELHO, L. Efficient hardware implementation of radial basis function neural network with customized-precision floating-point operations. *Control Engineering Practice 60* (2017), 124–132.

[3] BECKHOFF, C., KOCH, D., AND TORRESEN, J. Go Ahead: A Partial Reconfiguration Framework. In *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines* (apr 2012), IEEE, pp. 37–44.

[4] BILLINGS, S. *Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio-Temporal Domains*. Wiley, 2013.

[5] BILLINGS, S., AND VOON, W. S. F. Structure detection and model validity tests in the identification of nonlinear systems. *IEE Proceedings Pt. D: Control Theory and Applications 130*, 4 (1983), 193–199.

[6] BROOMHEAD, D., AND LOWE, D. Multivariable functional interpolation and adaptive networks. *Complex Systems 2* (1988), 321–355.

[7] BRUNTON, S., AND KUTZ, J. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019.

[8] BURNS, M. K., PEI, D., AND VINJAMURI, R. Myoelectric control of a soft hand exoskeleton using kinematic synergies. *IEEE Transactions on Biomedical Circuits and Systems 13*, 6 (2019).

[9] CALDERON, C. A., RAMIREZ, C., BARROS, V., AND PUNIN, G. Design and Deployment of Grasp Control System applied to robotic hand prosthesis. *IEEE Latin America Transactions* (2017).

[10] CARDARILLI, G. C., DI NUNZIO, L., FAZZOLARI, R., GIARDINO, D., MATTA, M., RE, M., SILVESTRI, F., AND SPANÒ, S. Efficient Ensemble Machine Learning Implementation on FPGA Using Partial Reconfiguration. In *Applications in Electronics Pervading Industry, Environment and Society*. Springer Nature Switzerland AG, 2019, pp. 253–259.

[11] CELIKEL, R. ANN based angle tracking technique for shaft resolver. *Measurement 148* (2019).

[12] CERVERO, T. G., CABA, J., LÓPEZ, S., DONDO, J. D., SARMIENTO, R., RINCÓN, F., AND LÓPEZ, J. A Scalable and Dynamically Reconfigurable FPGA-Based Embedded System for Real-Time Hyperspectral Unmixing. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 8*, 6 (2015), 2894–2911.

[13] CHEN, P., TSAI, H., LIN, C., AND LEE, C. FPGA realization of a radial basis function based nonlinear channel equalizer. In *Advances in Neural Networks - ISNN 2005*, J. Wang, X.-F. Liao, and Z. Yi, Eds., vol. 3498 of *Lecture Notes in Computer Science*. Springer, Berlin, 2005, pp. 320–325.

[14] CHEN, S., COWAN, C. F., AND GRANT, P. M. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks 2*, 2 (1991), 302–309.

[15] CHOU, H.-H., KUNG, Y.-S., QUYNH, N. V., AND CHENG, S. Optimized FPGA design, verification and implementation of a neuro-fuzzy controller for PMSM drives. *Mathematics and Computers in Simulation 90* (2013), 28 – 44.

[16] CHU, J.-U., JUNG, D.-H., AND LEE, Y.-J. Design and control of a multifunction myoelectric hand with new adaptive grasping and self-locking mechanisms. In *2008 IEEE International Conference on Robotics and Automation* (may 2008), IEEE, pp. 743–748.

[17] CLEMENTE, F., IANNICIELLO, V., GHERARDINI, M., AND CIPRIANI, C. Development of an embedded myokinetic prosthetic hand controller. *Sensors 19*, 14 (2019), 3137.

[18] COMPANY, S. R. Shadow Dexterous Hand Technical Specification. Tech. Rep. August, Shadow Robot Company, 2015.

[19] CUTKOSKY, M. On grasp choice, grasp models, and the design of hands for manufacturing tasks. *IEEE Transactions on Robotics and Automation 5*, 3 (jun 1989), 269–279.

[20] DANTAS, H., WARREN, D. J., WENDELKEN, S. M., DAVIS, T. S., CLARK, G. A., AND MATHEWS, V. J. Deep learning movement intent decoders trained with dataset aggregation for prosthetic limb control. *IEEE Transactions on Biomedical Engineering 66*, 11 (2019).

[21] DAOUD, N., GAZEAU, J., ZEGHLOUL, S., AND ARSICAULT, M. A real-time strategy for dexterous manipulation: Fingertips motion planning, force sensing and grasp stability. *Robotics and Autonomous Systems 60* (mar 2012).

[22] DE LUCA, C. J. The use of surface electromyography in biomechanics. *Journal of applied biomechanics 13*, 2 (1997), 135–163.

[23] DE SOUZA, A. C. D., AND FERNANDES, M. A. C. Parallel fixed point implementation of a radial basis function network in an FPGA. *Sensors 14*, 10 (2014), 18223.

[24] DUBROVA, E. *Fault-Tolerant Design*. Springer New York, New York, NY, 2013.

[25] FAN, Z.-C., AND HWANG, W.-J. Efficient VLSI architecture for training radial basis function networks. *Sensors 13*, 3 (2013), 3848–3877.

[26] FAUDZI, A., OOGA, J., GOTO, T., TAKEICHI, M., AND SUZUMORI, K. Index Finger of a Human-like Robotic Hand using Thin Soft Muscles. *IEEE Robotics and Automation Letters 3*, 1 (2017), 1–1.

[27] FAZZOLETTO, E. *Characterization of Partial and Run- Time Reconfigurable FPGAs*. PhD thesis, KTH ROYAL INSTITUTE OF TECHNOLOGY, 2016.

[28] FERREIRA, P., RIBEIRO, P., ANTUNES, A., AND DIAS, F. M. A high bit resolution FPGA implementation of a FNN with a new algorithm for the activation function. *Neurocomputing 71*, 1-3 (2007), 71–77.

[29] FREESCALE SEMICONDUCTOR. Freescale Semiconductor Xtrinsic MAG3110 Three-Axis, Digital Magnetometer. Tech. Rep. 9.2, Freescale Semiconductor, 2013.

[30] F.R.S., K. P. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 2* (1901).

[31] FURUYA, S., AND SOECHTING, J. F. Speed invariance of independent control of finger movements in pianists. *Journal of Neurophysiology 108*, 7 (oct 2012).

[32] GAZEAU, J., ZEHLOUL, S., ARSICAULT, M., AND LALLEMAND, J. The LMS hand: force and position controls in the aim of the fine manipulation of objects. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)* (2001), vol. 3, IEEE, pp. 2642–2648.

[33] GUO, K., ZENG, S., YU, J., WANG, Y., AND YANG, H. A survey of fpga-based neural network inference accelerators. *ACM Transactions on Reconfigurable Technology and Systems 12*, 1 (2019), 2:1–2:26.

[34] Häger-Ross, C., and Schieber, M. H. Quantifying the Independence of Human Finger Movements: Comparisons of Digits, Hands, and Movement Frequencies. *The Journal of Neuroscience 20*, 22 (nov 2000).

[35] Hajduk, Z. High accuracy FPGA activation function implementation for neural networks. *Neurocomputing 247* (2017).

[36] Haykin, S. S. *Neural networks and learning machines*, 3rd ed. Prentice Hall, Upper Saddle River, 2009.

[37] Hidalgo-l, A., Navas-gonz, R., Herr, J., Vidal-verd, F., and Antonio, S. FPGA-Based Tactile Sensor Suite Electronics for Real-Time Embedded Processing. *IEEE Transactions on Industrial Electronics 64*, 12 (2017), 9657–9665.

[38] Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences 79* (1982).

[39] Hu, H., Huang, J., Xing, J., and Wang, W. Key issues of FPGA implementation of neural networks. In *Second International Symposium on Intelligent Information Technology Application* (Shanghai, China, Dec 2008), vol. 3, pp. 259–263.

[40] Hu, Q., Tang, X., and Tang, W. A smart chair sitting posture recognition system using flex sensors and fpga implemented artificial neural network. *IEEE Sensors Journal 20*, 14 (2020).

[41] Huang, H., Li Jiang, Liu, Y., Hou, L., Cai, H., and Liu, H. The Mechanical Design and Experiments of HIT/DLR Prosthetic Hand. In *2006 IEEE International Conference on Robotics and Biomimetics* (2006), IEEE.

[42] Hussain, H. M., Benkrid, K., and Seker, H. Dynamic partial reconfiguration implementation of the SVM/KNN multi-classifier on FPGA for bioinformatics application. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)* (aug 2015), IEEE, pp. 7667–7670.

[43] Ivo, R. M., and Munoz, D. M. RTRLib: A High-Level Modeling Tool for the Implementation of Dynamically Partial Reconfigurable System-on-Chips. In *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)* (dec 2019), IEEE, pp. 1–5.

[44] Jeong, D. H., Chu, J. U., and Lee, Y. J. Development of myoelectric hand with infrared led-based tactile sensor. *Journal of Institute of Control, Robotics and Systems 15*, 8 (aug 2009), 831–838.

[45] JEONG, S. H., KIM, K.-S., AND KIM, S. Designing Anthropomorphic Robot Hand with Active Dual-Mode Twisted String Actuation Mechanism and Tiny Tension Sensors. *IEEE Robotics and Automation Letters 2*, 3 (2017), 1–1.

[46] JOHNSON, A. P., CHAKRABORTY, R. S., AND MUKHOPADHYAY, D. A puf-enabled secure architecture for fpga-based iot applications. *IEEE Transactions on Multi-Scale Computing Systems 1*, 2 (2015), 110–122.

[47] KAPANDJI, A., AND TORRES, M. *Fisiología articular: esquemas comentados de mecánica articular. Hombro, codo, pronosupinacion, muñeca, mano. 1.* Fisiología articular. Tomo 1. Hombro, codo, pronosupinación, muñeca, mano. Editorial Médica Panamericana, 2006.

[48] KAPPASSOV, Z., CORRALES, J.-A., AND PERDEREAU, V. Tactile sensing in dexterous robot hands — Review. *Robotics and Autonomous Systems 74* (dec 2015), 195–220.

[49] KIAT, W. P., MOK, K. M., LEE, W. K., GOH, H. G., AND ACHAR, R. An energy efficient FPGA partial reconfiguration based micro-architectural technique for IoT applications. *Microprocessors and Microsystems 73* (2020), 102966.

[50] KIM, J., AND JUNG, S. Implementation of the RBF neural chip with the back-propagation algorithm for on-line learning. *Applied Soft Computing 29* (2015), 233 – 244.

[51] KOCH, D., BECKHOFF, C., AND TEICH, J. ReCoBus-Builder: A novel tool and technique to build statically and dynamically reconfigurable systems for FPGAS. In *2008 International Conference on Field Programmable Logic and Applications* (2008), IEEE, pp. 119–124.

[52] KÄSTNER, F., JANSSEN, B., KAUTZ, F., HÜBNER, M., AND CORRADI, G. Hardware/software codesign for convolutional neural networks exploiting dynamic partial reconfiguration on pynq. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (May 2018), pp. 154–161.

[53] LAN, T., LIU, Y., JIN, M., FAN, S., FANG, H., XIA, J., AND LIU, H. DSP&FPGA-based joint impedance controller for DLR/HIT II dexterous robot hand. In *2009 IEEE/ASME International Conference on Advanced Intelligent Mechatronics* (jul 2009), IEEE.

[54] LEE, D.-H., PARK, J.-H., PARK, S.-W., BAEG, M.-H., AND BAE, J. KITECH-Hand: A Highly Dexterous and Modularized Robotic Hand. *IEEE/ASME Transactions on Mechatronics 22*, 2 (apr 2017), 876–887.

[55] Lee, S., Noh, S., Lee, Y., and Park, J. H. Development of bio-mimetic robot hand using parallel mechanisms. In *2009 IEEE International Conference on Robotics and Biomimetics (ROBIO)* (dec 2009), IEEE, pp. 550–555.

[56] Levenberg, K. A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics 2*, 2 (jul 1944), 164–168.

[57] Liu, H., Wu, K., Meusel, P., Seitz, N., Hirzinger, G., Jin, M., Liu, Y., Fan, S., Lan, T., and Chen, Z. Multisensory five-finger dexterous hand: The DLR/HIT Hand II. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems* (sep 2008), IEEE, pp. 3692–3697.

[58] Ljung, L. Perspectives on system identification. *Annual Reviews in Control 34*, 1 (2010), 1 – 12.

[59] Martin, E., and Hine, R. *A Dictionary of Biology*. Oxford Paperback Reference. OUP Oxford, 2008.

[60] Masselos, K., and Voros, N. S. Introduction to Reconfigurable Hardware. In *System Level Design of Reconfigurable Systems-on-Chip*. Springer-Verlag, Berlin/Heidelberg, 2005, ch. 1, pp. 15–26.

[61] Mattar, E. A survey of bio-inspired robotics hands implementation: New directions in dexterous manipulation. *Robotics and Autonomous Systems 61*, 5 (2013), 517–544.

[62] Micera, S., Carpaneto, J., and Raspopovic, S. Control of hand prostheses using peripheral information. *IEEE reviews in biomedical engineering 3* (2010), 48–68.

[63] Minsky, M., and Papert, S. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1972.

[64] Mohammadi, M., Krishna, A., Nalesh, S., and Nandy, S. K. A Hardware Architecture for Radial Basis Function Neural Network Classifier. *IEEE Transactions on Parallel and Distributed Systems 29*, 3 (2018).

[65] Muñoz, D. M., Sánchez, D., Llanos, C., and Ayala-Rincón, M. FPGA-based floating-point library for CORDIC algorithms. In *Proc. IEEE Int. Conf. Southern Programmable Logic* (Porto de Galinhas, Brazil, 2010), pp. 55–60.

[66] Muñoz, D. M., Sánchez, D., Llanos, C., and Ayala-Rincón, M. Tradeoff of FPGA design of a floating-point library for arithmetic operators. *International Journal of Integrated Circuits and Systems 5*, 1 (2010), 42–52.

[67] NØRGÅRD, P. M., RAVN, O., POULSEN, N. K., AND HANSEN, L. K. *Neural networks for modelling and control of dynamic systems: a practitioner's handbook.* Springer-Verlag, London, 2000.

[68] ORTIZ-CATALAN, M., HÅKANSSON, B., AND BRÅNEMARK, R. An osseointegrated human-machine gateway for long-term sensory feedback and motor control of artificial limbs. *Science translational medicine 6*, 257 (2014), 257re6–257re6.

[69] PANO-AZUCENA, A. D., TLELO-CUAUTLE, E., DE LA FRAGA, L. G., SANCHEZ-LOPEZ, C., RANGEL-MAGDALENO, J. J., AND TAN, S. X. Prediction of chaotic time-series with different MLE values using FPGA-based ANNs. *SMACD 2017 - 14th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design* (2017).

[70] PARK, J., AND SANDBERG, I. W. Universal approximation using radial-basis-function networks. *Neural computation 3*, 2 (1991), 246–257.

[71] PERTUZ, S. A., LLANOS, C., PEÑA, C. A., AND MUÑOZ, D. M. A modular and distributed impedance control architecture on a chip for a robotic hand. In *2018 31st Symposium on Integrated Circuits and Systems Design (SBCCI)* (Aug 2018), pp. 1–6.

[72] PERTUZ, S. A., NOZ, D. M., AND LLANOS, C. Development of a robotic hand using bioinspired optimization for mechanical and control design: Unb-hand. *IEEE Access* (2021).

[73] PEZZAROSSA, L., KRISTENSEN, A. T., SCHOEBERL, M., AND SPARSØ, J. Using dynamic partial reconfiguration of FPGAs in real-Time systems. *Microprocessors and Microsystems 61*, February (2018), 198–206.

[74] PIAZZA, C., GRIOLI, G., CATALANO, M. G., AND BICCHI, A. A Century of Robotic Hands. *Annual Review of Control, Robotics, and Autonomous Systems 2* (2019), 1–32.

[75] PNEVMATIKATOS, D., PAPADIMITRIOU, K., BECKER, T., BÖHM, P., BROKALAKIS, A., BRUNEEL, K., CIOBANU, C., DAVIDSON, T., GAYDADJIEV, G., HEYSE, K., LUK, W., NIU, X., PAPAEFSTATHIOU, I., PAU, D., PELL, O., PILATO, C., SANTAMBROGIO, M., SCIUTO, D., STROOBANDT, D., TODMAN, T., AND VANSTEENKISTE, E. FASTER: Facilitating Analysis and Synthesis Technologies for Effective Reconfiguration. *Microprocessors and Microsystems 39*, 4-5 (jun 2015), 321–338.

[76] RABOZZI, M., BRONDOLIN, R., NATALE, G., DEL SOZZO, E., HUEBNER, M., BROKALAKIS, A., CIOBANU, C., STROOBANDT, D., AND SANTAMBROGIO, M. D.

A CAD Open Platform for High Performance Reconfigurable Systems in the EXTRA Project. In *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)* (jul 2017), IEEE, pp. 368–373.

[77] RETTKOWSKI, J., FRIESEN, K., AND GOHRINGER, D. RePaBit: Automated generation of relocatable partial bitstreams for Xilinx Zynq FPGAs. In *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)* (nov 2016), IEEE, pp. 1–8.

[78] ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review 65*, 6 (1958).

[79] ROUSSEAU, B., MANET, P., GALERIN, D., MERKENBREACK, D., LEGAT, J. D., DEDEKEN, F., AND GABRIEL, Y. Enabling certification for dynamic partial reconfiguration using a minimal flow. *Proceedings -Design, Automation and Test in Europe, DATE* (2007), 983–988.

[80] SAHIN, S., BECERIKLI, Y., AND YAZICI, S. Neural network implementation in hardware using fpgas. In *Neural Information Processing*, I. King, J. Wang, L.-W. Chan, and D. Wang, Eds., vol. 4234 of *Lecture Notes in Computer Science*. Springer, Berlin, 2006, pp. 1105–1112.

[81] SAMIR, N., GAMAL, Y., EL-ZEINY, A. N., MAHMOUD, O., SHAWKY, A., SAEED, A. R., AND MOSTAFA, H. Energy-adaptive lightweight hardware security module using partial dynamic reconfiguration for energy limited internet of things applications. *Proceedings - IEEE International Symposium on Circuits and Systems 2019-May* (2019), 2–5.

[82] SCHOUKENS, J., SUYKENS, J., AND LJUNG, L. Wiener-Hammerstein benchmark. In *15th IFAC Symposium on System Identification* (Saint-Malo, France, 2009).

[83] SHAWAHNA, A., SAIT, S. M., AND EL-MALEH, A. Fpga-based accelerators of deep learning networks for learning and classification: A review. *IEEE Access 7* (2019), 7823–7859.

[84] SJÖBERG, J., ZHANG, Q., LJUNG, L., BENVENISTE, A., DELYON, B., GLORENNEC, P.-Y., HJALMARSSON, H., AND JUDITSKY, A. Nonlinear black-box modeling in system identification: a unified overview. *Automatica 31*, 12 (dec 1995), 1691–1724.

[85] SMITH, S. W. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, San Diego, CA, USA, 1997.

[86] SONG, C., XIE, S., ZHOU, Z., AND HU, Y. Modeling of pneumatic artificial muscle using a hybrid artificial neural network approach. *Mechatronics 31* (2015), 124 – 131.

[87] TAM, S., BOUKADOUM, M., CAMPEAU-LECOURS, A., AND GOSSELIN, B. A fully embedded adaptive real-time hand gesture classifier leveraging hd-semg and deep learning. *IEEE Transactions on Biomedical Circuits and Systems 14*, 2 (2020).

[88] TARANTINO, S., CLEMENTE, F., BARONE, D., CONTROZZI, M., AND CIPRIANI, C. The myokinetic control interface: tracking implanted magnets as a means for prosthetic control. *Scientific reports 7*, 1 (2017), 17149.

[89] TAVAKOLI, M., AND DE ALMEIDA, A. T. Adaptive under-actuated anthropomorphic hand: ISR-SoftHand. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems* (sep 2014), IEEE.

[90] TAVAKOLI, M., ENES, B., SANTOS, J., MARQUES, L., AND DE ALMEIDA, A. T. Underactuated anthropomorphic hands: Actuation strategies for a better functionality. *Robotics and Autonomous Systems* (2015).

[91] TAYLOR, C. R., ABRAMSON, H. G., AND HERR, H. M. Low-latency tracking of multiple permanent magnets. *IEEE Sensors Journal 19*, 23 (2019).

[92] THANH, T. D. C., AND AHN, K. K. Nonlinear pid control to improve the control performance of 2 axes pneumatic artificial muscle manipulator using neural network. *Mechatronics 16*, 9 (2006), 577 – 587.

[93] THE BIOROBOTICS INSTITUTE, S. S. S. Myki : A bidirectional myokinetic implanted interface for natural control of artificial limbs, 2020. `http://www.mykierc.eu/summary-and-figures/`, Access on Aug 2020.

[94] VEPA, R. Biomimetic Robotics. In *Engineered Biomimicry*, Newnes, Ed. Elsevier, 2013, pp. 81–105.

[95] WANG, X., LIU, Y., YANG, D., LI, N., JIANG, L., AND LIU, H. Progress in the biomechatronic design and control of a hand prosthesis. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems* (oct 2010), IEEE.

[96] WEIR, R. F., TROYK, P. R., DEMICHELE, G. A., KERNS, D. A., SCHORSCH, J. F., AND MAAS, H. Implantable myoelectric sensors (imess) for intramuscular electromyogram recording. *IEEE Transactions on Biomedical Engineering 56*, 1 (2008), 159–171.

[97] WERBOS, P. J. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Ph. D. Harvard University, 1975.

[98] WU, Q., CHEN, B., AND WU, H. Neural-network-enhanced torque estimation control of a soft wearable exoskeleton for elbow assistance. *Mechatronics 63* (2019), 102279.

[99] WU, Q., WANG, X., CHEN, B., AND WU, H. Development of an rbfn-based neural-fuzzy adaptive control strategy for an upper limb rehabilitation exoskeleton. *Mechatronics 53* (2018), 85 – 94.

[100] XILINX. Partial Reconfiguration User Guide. Tech. rep., Xilinx, 2013.

[101] XILINX. 7 Series FPGAs Configuration. Tech. rep., Xilinx, 2015.

[102] XILINX. Vivado Design Suite User Guide. Tech. rep., Xilinx, 2015.

[103] YIN, J., YANG, J., YANG, H., AND DU, Z. A CNN accelerator on embedded FPGA using dynamic reconfigurable coprocessor. In *Proceedings of the International Conference on Artificial Intelligence, Information Processing and Cloud Computing - AIIPCC '19* (New York, New York, USA, 2019), ACM Press, pp. 1–5.

[104] YU, H., XIE, T., PASZCZYÑSKI, S., AND WILAMOWSKI, B. M. Advantages of radial basis function networks for dynamic system design. *IEEE Transactions on Industrial Electronics 58* (2011).

[105] YUAN, L., ZHAO, H., CHEN, H., AND REN, B. Nonlinear mpc-based slip control for electric vehicles with vehicle safety constraints. *Mechatronics 38* (2016), 1 – 15.

[106] ZAMACOLA, R., GARCIA MARTINEZ, A., MORA, J., OTERO, A., AND DE LA TORRE, E. IMPRESS: Automated Tool for the Implementation of Highly Flexible Partial Reconfigurable Systems with Xilinx Vivado. In *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)* (dec 2018), IEEE, pp. 1–8.

[107] ZHAO, D., JIANG, L., HUANG, H., JIN, M., CAI, H., AND LIU, H. Development of a Multi-DOF Anthropomorphic Prosthetic Hand. In *2006 IEEE International Conference on Robotics and Biomimetics* (2006), IEEE, pp. 878–883.

APPENDIX

A RELATED ARTICLES

Analog Integrated Circuits and Signal Processing manuscript No.
(will be inserted by the editor)

A Parallel System-on-Chip Approach for Impedance Controller for a 7-DoF robotic hand

Sergio A. Pertuz [1] · Carlos Llanos [1] · Cesar Peña [2] · Daniel Muñoz [1,3]

Received: date / Accepted: date

Abstract Robotic hands tend to have a high number of sensors and actuators in a small space, whose supervising and control must be performed at the same time with precision and, in some cases, at high speed. Usually, a grid of Micro-Processors-Units (MPUs) or Micro-Controller-Units (MCUs) is employed to solve this problem, given the ease of programming. However, this solution can carry some drawbacks like the necessity of more space for computer resources. This work introduces a System-on-Chip (SoC) based approach that carries out the control of a robotic hand with multiple Degree of Freedom (DoF). In order to justify its advantages, the proposed solution was tested in a robotic hand and compared against other implementations on (a) an ATMEL microcontroller, (b) an ARM processor, and (c) a full-dedicated hardware architecture on FPGA. Comparisons were made in terms of performance, computational resources, and power consumption. Additionally, our results showed an improvement over a previous MCU-Grid-based hand controller that achieved a control loop frequency of 1 KHz. In contrast, the proposed SoC approach achieved a 47.26 kHz frequency, showing the advantages of using our parameterizable floating-point arithmetic cores, which allow the designer to adjust the word-width to optimize both the hardware resources and energy consumption.

Keywords FPGA · Impedance Controller · Kalman Filter · Modular Architecture · Robotic Hand · SoC-System on a Chip

✉ S. Pertuz
E-mail: sergio.pertuz@unb.br

[1] Laboratory of Embedded Systems and Integrated Circuits (LEIA), Dept. of Mechanical Engineering, University of Brasilia, Brasilia, Brazil.

[2] Automation and Control research group, Faculty of Engineering and Architecture, University of Pamplona, Pamplona, Colombia.

[3] Electronics Engineering Program, Faculty of Gama, University of Brasilia, Brasilia, Brazil.

1 INTRODUCTION

Over the past four decades, there have been significant contributions in the field of computer science, artificial intelligence, robotics, and other related fields. This progress has allowed the development of robotic systems that are more capable and sophisticated, such as biomimetic robotics. Such robots are regularly more capable, robust, and efficient than other types of conventional robots when used in unstructured workplaces [25]. In this context, the field of *robotic hand control* is a sub-topic of biomimetic robotics approached by many researchers achieving significant advances in the last decade.

Robotic hands can be used as prosthetics hands, like the *i-limb* [24], the *Taka-Hand* [23], the *Michelangelo hand* [16], and the *bebiobic* [15]. Normally, these systems need to be sturdy for durability and, for that, they have few functions and components.

Otherwise, robotic hands used as end-effectors, e.g., *KITECH-Hand* [12], *Prensilia-Hand* [20], the *Jeong-hand* [9]*Shadow-hand* [22], *DLR-hand* [11] and the *Schunk-hand* [21], are more sophisticated and have more functions due to the high quantity of components, sensors, and actuators. Given the complexity of the overal system, sometimes a Micro Processor Unit (MPU) or Micro-Controller-Unit (MPU) is not enough to meet the required processing speed to reproduce a desired action or control. Because of this, in order to embed the low-level control, some designers choose to develop a grid of multiple MCUs in a multi-DoF (Degree of Freedom) robotic hand [12].

In contrast, other approaches exploit the advantages of FPGAs (Field-Programmable Gate Arrays) to perform parallel processing to implement parallel control loops at high-frequency rates [11, 13, 27]. The benefits of this second approach are that, unlike having a large number of MPUs, an FPGA can implement various architectures simultaneously, consuming less energy and achieving similar or even better control frequencies. Besides, FPGAs allow overcoming the synchronization problems between different controllers implemented over a set of independent processors, e.g., each one generating a specific control law.

For a robotic hand, being able to perform grasps or manipulation of objects, some force/torque control is necessary. This type of control usually is more effective when combined with some feedback information from different sensors [4,10]. This work is dedicated to implement an impedance control, which is a type of control that aims to control the dynamics of a robot when it interacts with its environment. This is done by modeling the contact of the robot with an object, for example, as a mass-spring-damper system. This control has previously been used in robotic hands [27] and is typically represented using the equation:

$$Mx\ddot{\theta}(t) + Bx\dot{\theta}(t) + Kx\theta(t) = T(t), \tag{1}$$

where T is the resulting torque of the system, M, B and K are mass, damping, and spring coefficients, respectively, and x represents the displacement of the robot [8].

The impedance controller is relatively simple on its own; however, the complexity can increase when added to the necessary sensors to deal with a robot with several DoF. Besides the control, another issue that must be addressed is the sensory management and filtering. Similar to the control system, the management and filtering process must be performed at a required minimum frequency. This issue has been previously approached using FPGAs [26].

In this work, parallel Kalman filters were implemented to estimate the position and velocity of the fingers of a 7 DoF robotic hand. A linear Kalman Filter (KF) is an algorithm that works in a two-step process. In the *prediction step*, the KF estimates the current state variables (from a process model), along with their uncertainties. In the *estimation step*, the measurement system provides, from the sensors, corrupted signals with some amount of error (Gaussian noise), and the estimates are updated using a weighted average of the state variables, with more weight being given to estimates with higher certainty. The algorithm is recursive, and it can run in real-time, using only the present input measurements and the previously calculated state and its uncertainty matrix [7]. The estimation of state variables using the KF tends to be more accurate than an estimation using a single measurements.

Algorithm 1 describes a canonical KF. In there, the estimated state variable vector is $\hat{\mathbf{x}}$, $\mathbf{A}$ and $\mathbf{B}$ are the discrete space state linear model, $\mathbf{u}$ is the control signal, $\hat{\mathbf{z}}$ is the measurements vector, $\mathbf{P}$ is the estimate of the covariance (with Gaussian probability distribution), and $\mathbf{K}$ is the Kalman gain (ẽstimated state variable vector is $\hat{\mathbf{x}}$, $\mathbf{A}$ and $\mathbf{B}$ are the discrete space state linear model, $\mathbf{u}$ is the control signal, $\hat{\mathbf{z}}$ is the measurements vector, $\mathbf{P}$ is the estimate of the covariance (with Gaussian probability distribution), and $\mathbf{K}$ is the Kalman gain ($\mathbf{S}$ is used to calculate $\mathbf{K}$). The outputs of the system are calculated from $\hat{\mathbf{y}} = \mathbf{H}\hat{\mathbf{x}}$. In our case the state variable vector is $\hat{\mathbf{x}_k} = \left[\hat{\theta}\ \hat{\omega}\right]^T$, where $\hat{\theta}$ and $\hat{\omega}$ are the position and angular velocity, respectively. The $\hat{\mathbf{x}}_k$ vector is calculated *a priori* in line 2 (prediction step) and *a posteriori* in line 7 (estimation step). P_k denotes the covariance of the estimation error (for $\hat{\theta}$ and $\hat{\omega}$) which also is calculated *a priori* and *a posteriori* in lines 3 and 8, respectively. Note that ϵ (line 4) represents the measurement residual, that is the difference between the noisy measurements ($\mathbf{z}$) and the vector $\mathbf{H}\hat{\mathbf{x}}$.

Algorithm 1 Linear Kalman Filter (KF) Algorithm

1: **function** BASIC-KALMAN-FILTER(z_k)

$$\left.\begin{array}{ll} 2: & \hat{\mathbf{x}}_{k|k-1} = \mathbf{A}_k\hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k\mathbf{u}_k \\ 3: & \mathbf{P}_{k|k-1} = \mathbf{A}_k\mathbf{P}_{k-1|k-1}\mathbf{A}_k^{\mathrm{T}} + \mathbf{Q}_k \end{array}\right\} \quad \text{Predict}$$

$$\left.\begin{array}{ll} 4: & \tilde{\epsilon}_k = \mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k|k-1} \\ 5: & \mathbf{S}_k = \mathbf{R}_k + \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^{\mathrm{T}} \\ 6: & \mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^{\mathrm{T}}\mathbf{S}_k^{-1} \\ 7: & \hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k \\ 8: & \mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1} \end{array}\right\} \quad \text{Estimate}$$

This work aims to implement an impedance control loop [19] for a previously developed 7 DoF robotic hand [17] using an SoC (System on Chip), specifically a Zynq-7020c chip (from Xilinx), to perform grasping tasks. The prototype can perform different types of grasp; such as power and precision grasp, with a single finger and the full hand for objects of different shapes. The proposed system combines the use of an MPU to synchronize the functions of several filtering modules.

It is important to point out that the main novelty of this work is that no similar work exists combining hardware architectures and MPU to perform an impedance control of a robotic hand. In [18] it was proposed a HW-SW approach for the impedance control of a robotic finger. In this work the proposed approach was extended for a 7-DoF robotic hand performing some power and precision grasps with all its fingers. Similarly, a numerical comparison with MCU-based solutions reported in previous work is addressed in terms of performance (loop control frequency), energy consumption, scalability, and area (onboard) consumption. Additionally, this work includes a solution with a Petalinux OS connected with the programmable hardware, improving the flexibility and enabling the development of user applications with suitable connection capabilities.

The proposed solution outperforms previous related works in the following aspects: (a) the filtering processes from all sensors and the loop control were implemented on a single chip in contrast to the solutions of a grid of FPGAs/MCU used in [11, 12]; (b) the loop control frequency of 47.26 KHz was improved if compared with [11, 12], allowing the development of high-dynamic robotic hands [14]; (c) the use of parameterizable floating-point arithmetic cores enabling the designer to adjust the word-width in order to optimize the precision and hardware and energy consumption.

2 ROBOTIC HAND

Fig. 1 depicts the robotic hand used in this work to validate the proposed FPGA-based grasping controllers. The complete system is composed of 7 DoF each actuated by a DC motor, 7 analog rotational position sensors, and 7 motor current sensors [17].

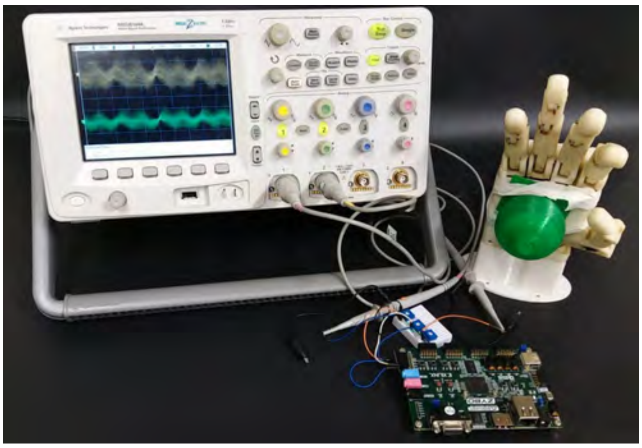


Fig. 1 Robotic hand performing a grasp action

In total there are five fingers in the robot. Some actuated by two Dof and others by one:

1. The thumb and index have 2 DoF (*aa* and *fe*); and
2. The remaining fingers have 1 DoF.

The controller will be executd in a decoupled form, meaning that it asumes the action of every DoF does not have actions on the others. Thus, an analysis of the system is more straightforward, and it will be possible to implement the controllers separately.

This project explores the implementation of a hybrid controller using a hardware-software codesign approach. For the full hand, depicted in the previous figure, the high-level controller is described in Fig.2. In the PL-part (*Programmable Logic*), position and current sensors of the robotic hand are read through the on-chip 12-bits ADC. The data from sensors are synchronously fed to the *Filtering Process* block (θ_1-θ_7 for position and i_1-i_7 for current), in which there are seven parallel submodules, one for each DoF, that filter the sensors data.

The filtered data is forward to the MPU, in the PS-part (*Programmable-Software*), that calculates the action of the impedance control. Finally, the Motor Driver module send, in a parallel fashion, PWM (Pulse Width Modulation) signals to the motor drivers.

The MPU is also in charge of the conectivity of the system and behaves as a supervisor. The *AXI interconnect* block, is a proprietary microcontroller bus that allows the connection between both the PS and the PL (*Programmable-Logic*) parts. This bus behaves as a two-way channel that allow the MPU to read and write the PL modules (supervisory task).

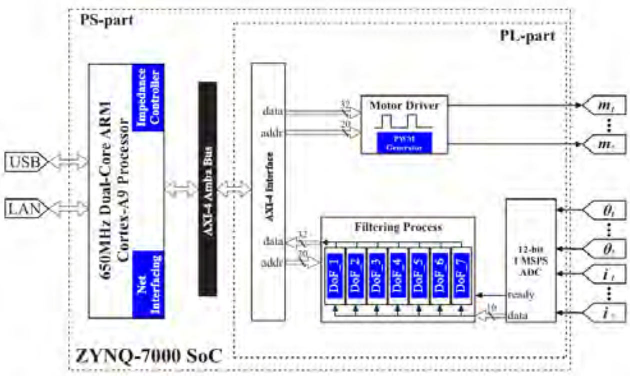


Fig. 2 Control System of the full hand DoF.

The dataflow diagram for controlling each *DoF* is depicted in Fig. 3. Note that this control scheme also identifies which parts are in the PS and which ones are in the PL. It can be observed that the Filtering Process is fully implemented for each *DoF* in the PL part, and the same is replicated seven times (see Fig. 2). In this way, each *Filtering Process* circuit receives each pair θ_k and i_k from the finger sensors and calculates each estimated pair $\hat{\theta}_k/\hat{\omega}_k$ (using the KF) and also the i_k from a low-pass filter, as will be explained below.

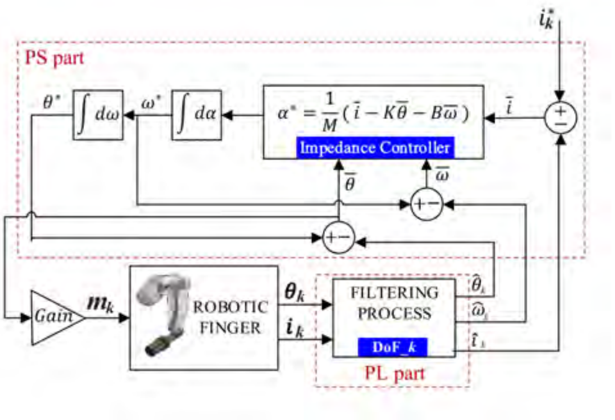


Fig. 3 Control scheme for a finger. It should be notice that the motor shaft torque in Eq.1 is proportional to the motor current (k=1 to 7)

The contents of the *Filtering Process* (DoF_1 to DoF_7 in Fig. 5 and *impedance controller* blocks are described in the following subsections.

2.1 PS part: Impedance Controller Scheme

In this work, a decoupled and linear behavior is considered, where every finger has an independent controller with torque and position feedback for the robotic fingers. Fig. 4 depicts the impedance model for each finger.

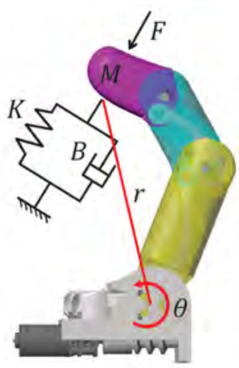


Fig. 4 Impedance Model for Robotic Finger

The implementation of the impedance controller is depicted in Fig. 3 (see PS part). This implementation is based on the use of a discrete integration method [3], as described in the following three steps:

1. Discretize Eq. 1 as follows:

$$\alpha^*(k+1) = (M)^{-1}(\bar{i}(k) - K\ \bar{\theta}(k) - B\ \bar{\omega}(k)), \quad (2)$$

where the torque T on Eq. 1 is replaced by the tracking motor current error $(\bar{i}(k))$ at a moment k, $\bar{\theta}(k)$ is the tracking position error, and $\bar{\omega}$ is the tracking angular speed error.

2. The result of the previous step is the target angular acceleration at the next step $(\alpha^*(k+1))$, which must be integrated to obtain the tracking angular velocity,

$$\omega^*(k+1) = \int_{k-1}^{k} \alpha^*(k+1)dk \quad (3)$$

3. Finally, the tracking angular position, $\theta^*(k+1)$, is obtained integrating the tracking angular velocity,

$$\theta^*(k+1) = \int_{k-1}^{k} \omega^*(k+1)dk \quad (4)$$

A 2nd order Euler's integration method was adopted to estimate Eqs. 3 and 4. It is important to highlight that the impedance controller scheme is required for each DoF and will be executed on the MPU of the SoC device.

2.2 PL part: Filtering Process

This block is composed of a Kalman filter module used for two tasks: (a) estimation of the velocity and position of the finger's first joint; (b) implementation of a low-pass filter for the current sensor. This block is depicted in Fig. 5, being required for each DoF, as depicted in figures 2 and 3 (see PL part). In this way, this circuit is replicated seven times, each one for each DoF, in order to exploit the intrinsic parallelism of the solution in the FPGA.

Fig. 5 presents the hardware architecture for the *Predict*, *update* $\hat{x}$ and *update* P stages of the Kalman filter algorithm (see Alg. 1). It additionally depicts the architecture of the 1st order low-pass discrete filter for the current sensor, given by Eq. 5. In addition, State 1-5 perform the prediction step; State 6-10 update $\hat{\theta}$, $\hat{\omega}$ and $\hat{i}$; and State 11-14 update P. Two adders, one multiplier, and one divider are shared between the states. Dotted lines represent internal registers.

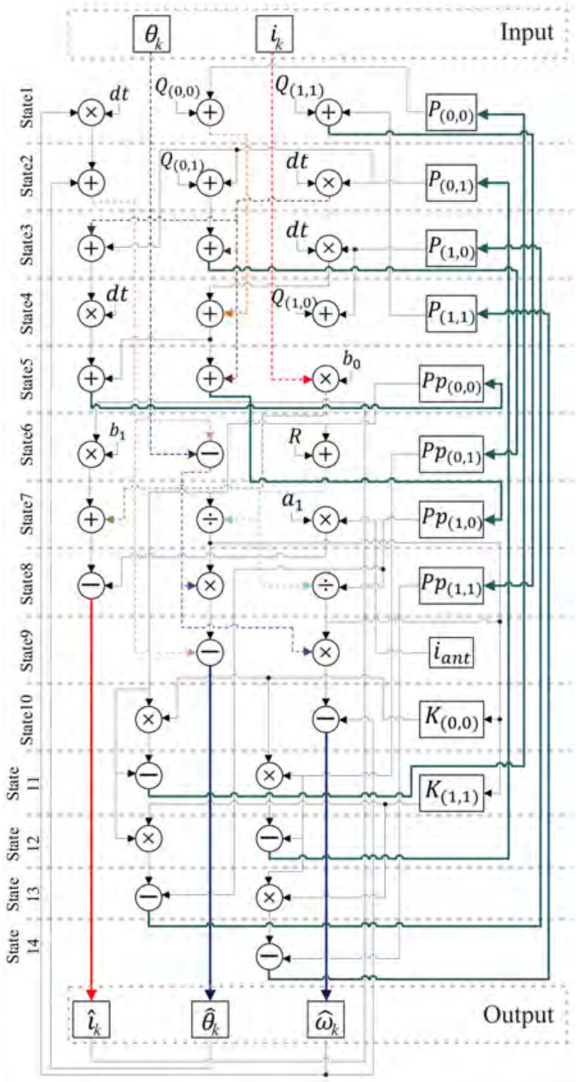


Fig. 5 *FILTERING PROCESS*: Hardware architecture for estimating current using a low-pass filter and, position and velocity of the finger's first joint using Kalman Filter. Two adders, one multiplier, and one divider are shared between the states.

From Algorithm 1: $\hat{\mathbf{x}} = \begin{bmatrix} \hat{\theta} \\ \hat{\omega} \end{bmatrix}$, $\mathbf{A} = \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix}$ and $\mathbf{B} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $\mathbf{u} = 0$, $\hat{\mathbf{z}} = \begin{bmatrix} \theta \\ \omega \end{bmatrix}$ where dt is the filter estimation period.

$$\hat{i}(k+1) = b_0 i(k) + b_1 i(k-1) - a_1 \hat{i}(k) \quad (5)$$

where b_0, b_1, a_1 are the filters coefficient.

The proposed architecture is based on custom 27 bits floating-point operators [6]. This resolution was chosen because it preserves the best resources/precision ratio for the selected FPGA device. The architectures were implemented using Finite State Machines (FSMs) for controlling the arithmetic operators. The outputs of the architecture are the registers $\hat{i}_k$, $\hat{\theta}_k$ and $\hat{\omega}_k$ for estimated position, velocity and current, respectively.

3 PARALLEL ARCHITECTURE OF CONTROL SCHEME

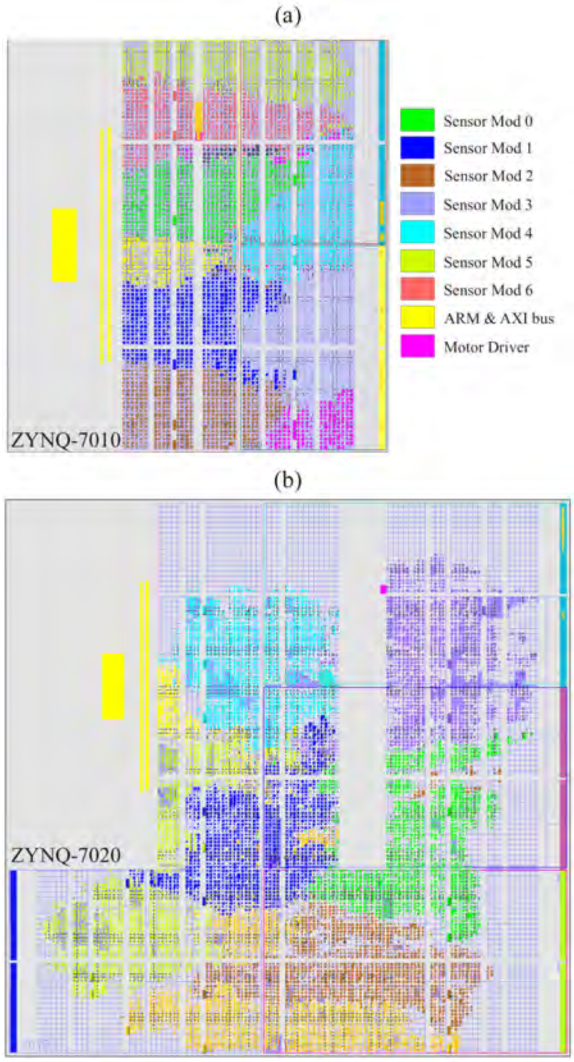


Fig. 6 Device utilization of proposed SoC-based architecture for the impedance control scheme for 7 DoF (5 fingers): (a) Is the hybrid solution using the Zynq-7010c and, (b) is the full-architecture implementation using Zynq-7020c device. The same color caption is used for both figures.

The Arty/Zynq-7020c SoC platform was used to implement the proposed control scheme. This platform allows the PS and PL parts of the *Zynq* to be connected through an *AXI* bus protocol (Advanced eXtensible Interface). In addition, the AXI permits to easily add a custom *IP* (Intellectual Property core) in a system together with the *ARM-CortexA9* (MPU) of the PS and the *XADC* (Xilinx Analog to Digital Converter) available on-chip.

This is done the *Xilinx Vivado* interface, where it is possible to connect the PS (Programmable Software) and PL (Programmable Logic) via proprietary *AXI* bus protocol (Advanced eXtensible Interface). This protocol permits to easily add a custom *IP*s (semiconductor Intellectual Property core) in a system together with the *ARM-Cortex9* and the *XADC* available in the chip.

Fig. 6(a) presents the chip utilization allowing to analyze the hardware resources used by the proposed SoC-based architecture. It can be seen that the chip's logic slices are almost fully used for the *Zynq-7010*.

For comparison purposes, Fig. 6(b) depicts the *full-logic* architecture architecture mapped on a Zynq-7020. It can be seen that it would not fit on the *Zynq-7010* device. Table 1 describes the resourses utilization for both chips, and demonstrates that the proposed SoC-based architecture is optimized in terms of hardware resources on the Zynq-7010. At this end, it is important to analyse the power consumption and the performance in terms of loop control frequency of the proposed solution.

Table 1 Utilization Report in terms of Slice LUTs (LUT), Slice Registers (SR) and DSPs (DSP) consumption

Z7010 Resource	Hybrid	Sol. [%]	Z7020 Resource	Full	Logic [%]
LUT (17,600)	13,295	76	(53,200)	22,521	42
SR (35,200)	4,632	13	(106,400)	25,702	24
DSP (80)	16	20	(220)	21	10

The on-chip power consumption for the designed architecture is presented in Fig. 7. It can be seen that, as expected, most of the system dynamic power (92%) is consumed by the MPU (presented as *PS7* on Figure). The rest of the logic components mentioned in the report compose the *sensor filtering* modules and consume 4% of the dynamic power. In total, the system has a power consumption of $1.582 Watts$ at standard voltage levels ($Vccint = 1V$, $Vccaux = 1.8V$ and $Vcco33 = 3.3V$).

In comparison, the ATMega microcontroller chip has a rated power consumption of approximately 410 mW [1]. If desired to implement the robot with 7 DoF, a grid of 7 MCUs would be required, similar to the Kitech-hand [12], summing up to $2.87W$; almost double the power needed for the implemented hybrid system.

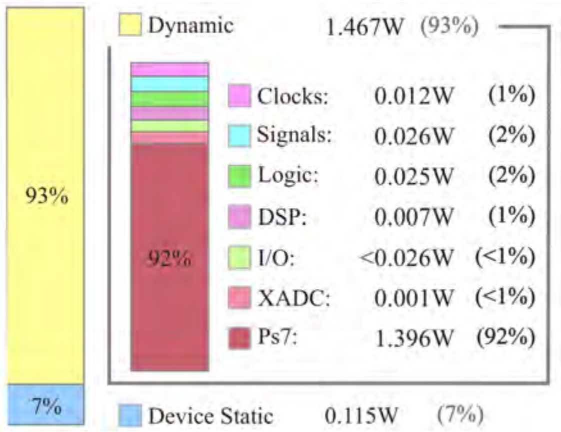


Fig. 7 On-chip power consumption of the proposed SoC-based solution for 7 DoF (5 fingers).

3.1 Control System Implementation Approaches.

For comparison purposes, the controller is implemented as three different types of structures, exploiting the modules proposed in the previous subsection as follows: (1) the architectures named as *software-only*, performs the overall control system using only software, without exploiting the parallel logic modules for sensing and control (inside an MPU or MCU); (2) similarly, the architectures named as *logic-only* performs the overall control system in hardware by mapping the overall structure in the FPGA; (3) finally, there is also a hybrid architecture that uses programmable software and logic to implement the system, exploiting both software and some logic modules in an SoC (a hardware/software co-design approach, such as proposed in Fig. 2).

Table 2 lists the different types of proposed implementations. The first column shows the type of implementation. For instance, *PS-Arduino* means that all control system has been implemented only in software using the Arduino platform. On the other hand, *PS-ArtyBM* describes a system implementation in software, using the platform ARTY/Zynq-7020c (in the MPU), without operating system (bare-metal), while *PS-ArtyLnx* represents a software implementation with Petalinux operating system [5]. In addition, it is also evaluated an implementation where all the control was implemented in the FPGA (see entry *PL-Arty*). Finally, the hardware/software codesign approaches are in the last two lines of Table 2, for bare-metal and for Petalinux.

Table 2 Control system implementation on different platforms.

ID	Type of Arch	Platform	OS	Clock Frequency
PS-Arduino	Software -Only	ATmega2560 Cortex-A9	Bare metal	16 MHz
PS-ArtyBM	Software -Only	ARM Cortex-A9	Bare metal	650 MHz
PS-ArtyLnx	Software -Only	ARM Cortex-A9	PLinux	650 MHz
PL-Arty	Logic -Only	XC7Z020	N/A	100 MHz
PS/PL-ArtyBM	Hybrid	ARM Cortex-A9 / 1CLG400C	Bare metal	650 MHz / 100 MHz
PS/PL-ArtyLnx	Hybrid	ARM Cortex-A9 / 1CLG400C	PLinux	650 MHz / 100 MHz

In total, six approaches were performed in this work, including the implementation of the controller in an Arduino (ATmega2560) microcontroller for a perspective on how a cheaper solution delivers. The comparison consists of executing the control loop and measuring the execution time for one DoF and then for the seven DoF. Table 3 lists the results for every architecture.

Table 3 Control system implementation on different platforms. t indicates the control execution time in microseconds. $Freq$ indicates the calculated control loop frequency in Hertz.

	1 DoF		7 DoF	
ID	t[us]	Freq[kH]	t[us]	Freq[kH]
PS-Arduino	1017.00	0.98	7219.00	0.14
PS-ArtyBM	5.24	190.84	32.92	30.38
PS-ArtyLnx	13.72	72.88	87.80	11.39
PL-Arty	1.20	830.00	1.44	692.52
PSPL-ArtyBM	3.38	303.03	21.16	47.26
PSPL-ArtyLnx	5.85	170.94	39.00	25.64

4 RESULTS AND DISCUSSIONS

The proposed SoC-based solution was compared with two previous approaches where the control scheme was implemented in a *ATmega2560* microcontroller of an *Arduino Mega* and in a fully dedicated architecture using a *Arty/Zynq-7020c* platform. In the latter all the functional blocks in Fig. 3 were implemented using the programmable logic of the FPGA without using the on-chip MPU. Additionally, in this work, different hybrid implementations are carried out to compare the execution time performance of the control algorithm for the complete robotic hand.

The performance of the approaches using software-only are better on bare-metal than with Petalinux-OS. Nevertheless, the hybrid system of Petalinux with PL (*PSPL-ArtyLnx*) stretches this difference in the execution time.

For a single finger, the proposed SoC-based solution achieved a maximum control loop frequency of 47.26 KHz using a clock of 667 MHz on the PS and 100 MHz on the PL. In contrast, the fully dedicated architecture (using the Zedboard) achieved a maximum control loop frequency of 830 KHz with a clock of 100 MHz, whereas the *ATMEL-2560* achieves a maximum control loop Frequency of 0.98 KHz operating with a clock of 16 MHz.

The maximum control loop time accepted to an implementation be considered feasible for the robotic hand is aproximately 1 KHz, which depends on: (a) the system's parameters, (b) the response time of the motors and, (c) sensors and *XADC* latency. It should be noted that the execution in the *ATmega2560* is sequential; that is, in order to control seven DoFs (five fingers) the expected control loop frequency using Arduino is around 0.14 KHz. Therefore, the implementation on the *Arduino* board does not meet this design parameter to control the hands 7 DoF of the robotic hand.

On the other hand, the SoC-based and fully dedicated solutions explore the parallelism achieving high control loop frequencies, enabling its application on high-speed dynamic robotic hands.

The control response of the finger using the proposed SoC-based architecture is similar to those implemented previously in the Arduino Mega and in the Zedboard (using the fully dedicated architecture). Fig. 8 illustrates the robotic hand performing a grasping task with a finger.

Fig. 8(a)-(c) shows the behavior of the controller and the filters grasping a semi-spherical object with 8.0 cm diameter. It can be seen that the filters reduce a lot of the sensor's noise, and although the position sensor does not have much noise in contrast with the current sensor, the velocity estimation is smoothed accurately which would be a result hard to achieve if not for the Kalman filter. Furthermore, the tracking current signal, shown as a yellow line, is corrected in a limited way. This indicates sub-damping of the controller which is a problem intended to be corrected in future works.

Initially, the controller was being implemented on a *Zybo-board* with a *Z7010* chip. However, the implementation for the full hand could not be possible because this board does not have the necessary ADC breakout pins for the sensors. The used Arty-based board provides several ADC channels that allows controlling the five fingers of the robotic hand simultaneously, enabling to perform experiments grasping several objects. Fig. 9 illustrates the grasping of three objects.

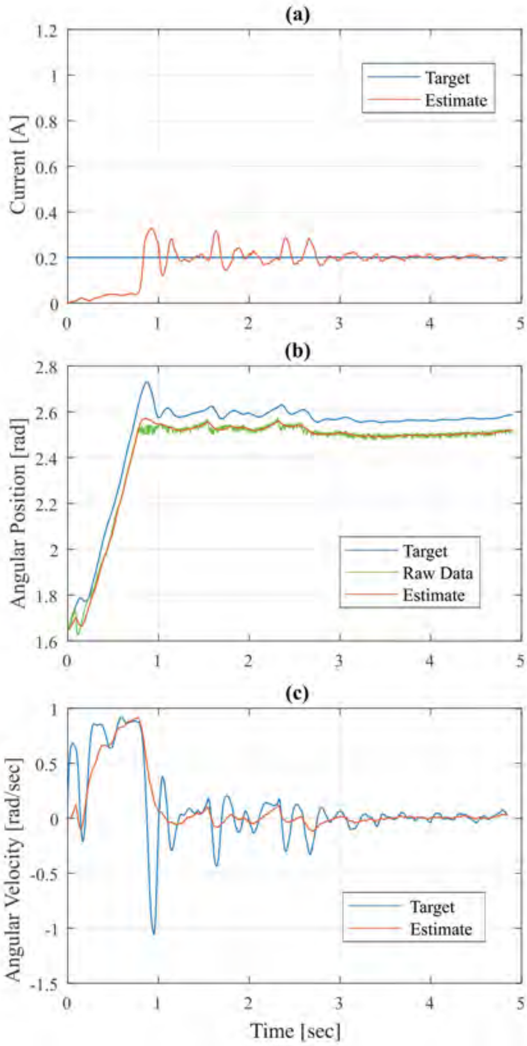


Fig. 8 Control behavior of impedance controller of one finger. (a) Current vs Time, (b) Position vs Time, (c) Velocity vs Time. Readers are addressed to https://youtu.be/kGw9jN-9ns8 to see a short video demonstrating the robotic hand in action.

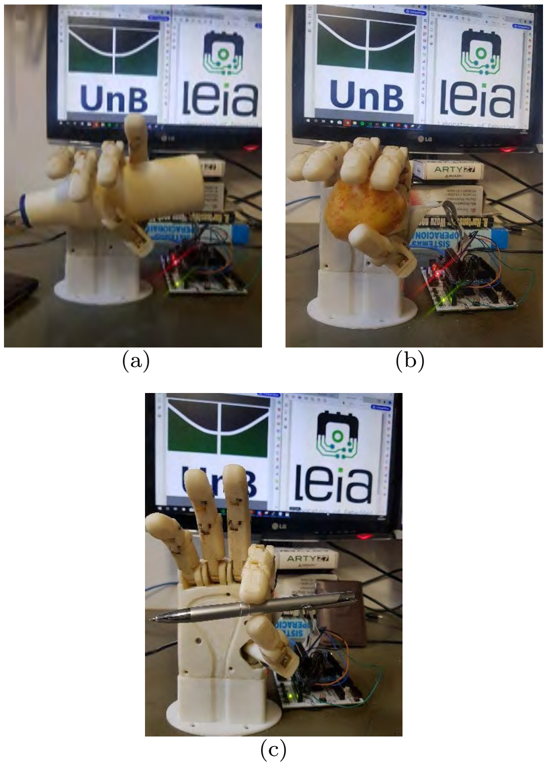


Fig. 9 (a)-(c) Robotic hand performing grasp on various objects.

Fig. 9(a)-(b) shows the robotic hand performing power and precision grasps. The former is a circular grasp for a spherical object, whereas the latter is a prismatic grasp of a cylindrical object. Fig. 9(c) illustrates a precision grasp using the thumb and index finger.

These experiments demonstrate the ability of the robotic hand control algorithms to perform some power and precision grasps.

5 CONCLUSIONS AND FUTURE WORKS

In this work, a novel parallel SoC-based impedance controller solution was proposed and implemented on Xilinx Zynq devices for performing grasping tasks using a robotic hand. The suitability of the proposed SoC solution to implement force/torque control, where the problems of performance, parallelism, and synchronization can be solved, was validated using a real robotic hand with 7 DoF. Performance comparisons in terms of execution time and control loop frequency were conducted for different approaches: *PS-Arduino*, *PS-Arty-BM*, *PS-ArtyLnx*, *PL-Arty*, *PSPL-ArtyBM*.

Those different approaches have each its advantages. On one hand, the applications in Bare-Metal (*PS-Arty-BM* and *PSPL-ArtyBM*) guarantee the correct execution of the controller at a fixed frequency. On the other hand, although the solutions using the Petalinux (*PS-ArtyLnx* and *PSPL-ArtyLnx*) do not guarantee real-time, they provide more flexibility for interfacing and networking. Connectivity for this project is essential given that this work is intended as part of a much bigger application, where several elements need to communicate with each other. Finally, the HW/SW approaches (*PSPL-ArtyBM* and *PSPL-ArtyLnx*) improved the *PS-only* maximum control loop frequency, and although the *PL-only* max frequency was not improved, the *PL-only* requires a bigger chip (*Zynq-7020*) and consumes more energy, whereas the HW/SW design can be implemented on a cheaper device (*Zynq-7010*). It is important to highlight that even though the achieved frequencies are not required for the current system, this architecture opens a path for other platforms with faster dynamics.

A relevant achievement of this implementation is that the power consumption of this solution improved a proposed MCU grid alternative using ATMega2560 ($1.5W$ vs. $2.8W$ for 7 DoF). Another advantageous characteristic of this work is that it is scalable without compromising too much energy consumption. If the robot uses more DoFs, for the MCU grid, then more MCU are needed increasing the power linearly, whereas for this work's hybrid implementation only a few more mW would be added for every new module on the FPGA.

As future works, it is proposed to improve the subdamping of the controller. Also, conduct more experiments on grasping tasks and manipulation to better test the dexterous of the robotic hand in combination with the control algorithm. As the covariance matrix of the KF was set empirically, another solution that will be further sought is to dynamically set these values as shown in [2]. Moreover, the authors will focus on the development of a GUI interface connected to Ethernet and WIFI, taking advantage of the PetaLinux flexibility, to compose a better interaction of the robotic hand with the final user.

Acknowledgements This work was supported by the Coordination for the Improvement of Higher Education Personnel of Brazil (CAPES) and the Foundation of Support to Research of the Federal District (FAPDF). The authors thank the anonymous reviewers for their contributions, which greatly improved the quality of the present work.

Conflict of interest

The authors declare that they have no conflict of interest.

References

1. Atmel, C.: 8-bit Atmel Microcontroller with 64K/128K/256K Bytes In-System Programmable Flash (2012). URL `www.atmel.com`
2. Basso, G.F., Guilherme Silva De Amorim, T., Brito, A.V., Nascimento, T.P.: Kalman filter with dynamical setting of optimal process noise covariance. IEEE Access **5**, 8385–8393 (2017)
3. Caccavale, F., Natale, C., Siciliano, B., Villani, L.: Integration for the next generation: embedding force control into industrial robots. IEEE Robotics & Automation Magazine **12**(3), 53–64 (2005)
4. Calderon, C.A., Ramirez, C., Barros, V., Punin, G.: Design and Deployment of Grasp Control System applied to robotic hand prosthesis. IEEE Latin America Transactions **15**(2), 181–188 (2017)
5. Co Xilinx: PetaLinux Reference Guide (2018). URL `https://www.xilinx.com/support/documentation/sw{_}manuals/xilinx2018{_}2/ug1144-petalinux-tools-reference-guide.pdf`
6. Daniel M. Munoz Diego F. Sanchez, C.H.L., Ayala-Rincon, M.: Tradeoff of FPGA design of a floating-point library for arithmetic operators. Journal of Integrated Circuits and Systems **5**(1), 42–52 (2010)
7. Einicke, G.: Smoothing, Filtering and Prediction - Estimating The Past, Present and Future. InTech (2012). DOI 10.5772/2706
8. Hogan, N.: Impedance Control: An Approach to Manipulation: Part IIImplementation. Journal of Dynamic Systems, Measurement, and Control **107**(1), 8 (1985)
9. Jeong, S.H., Kim, K.S., Kim, S.: Designing Anthropomorphic Robot Hand with Active Dual-Mode Twisted String Actuation Mechanism and Tiny Tension Sensors. IEEE Robotics and Automation Letters **2**(3), 1–1 (2017)
10. Kappassov, Z., Corrales, J.A., Perdereau, V.: Tactile sensing in dexterous robot hands Review. Robotics and Autonomous Systems **74**, 195–220 (2015)
11. Lan, T., Liu, Y., Jin, M., Fan, S., Fang, H., Xia, J., Liu, H.: DSP&FPGA-based joint impedance controller for DLR/HIT II dexterous robot hand. In: 2009 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, pp. 1594–1599. IEEE (2009)
12. Lee, D.H., Park, J.H., Park, S.W., Baeg, M.H., Bae, J.: KITECH-Hand: A Highly Dexterous and Modularized Robotic Hand. IEEE/ASME Transactions on Mechatronics **22**(2), 876–887 (2017)
13. Li, Y., Huang, C.: Force control for the fingers of the piano playing robot a gain switched approach. In: 2015 IEEE 11th International Conference on Power Electronics and Drive Systems, pp. 265–270 (2015)
14. Namiki, A., Komuro, T., Ishikawa, M.: High-speed sensorymotor fusion for robotic grasping. Measurement Science and Technology **13**(11), 1767–1778 (2002)
15. Otto Bock HealthCare: bebionic (2017). URL `http://bebionic.com/`
16. Ottobock: Michelangelo Hand (2017). URL `https://www.ottobockus.com/prosthetics/upper-limb-prosthetics/solution-overview/michelangelo-prosthetic-hand/`
17. Pertuz, S.A.: Dissertação de mestrado sistema embarcado baseado em arquiteturas reconfiguráveis do controle dinâmico de uma mão robótica sintonizado com algoritmos bioinspirados. Master disertation, University of Brasilia (2017). URL `http://repositorio.unb.br/handle/10482/24540`
18. Pertuz, S.A., Llanos, C., Peña, C.A., Muñoz, D.M.: A modular and distributed impedance control architecture on a chip for a robotic hand. In: 2018 31st Symposium on Integrated Circuits and Systems Design (SBCCI), pp. 1–6 (2018)
19. Pertuz, S.A., Llanos, C.H., Muñoz, D.: Simulation and implementation of impedance control in robotic hand. In: 24th ABCM International Congress of Mechanical Engineering, pp. 1–10. ABCM, Curitiba (2017)
20. Prensilia S R L: IH2 Azzurra Series. Self-Contained Robotic Hand Getting Started Guide (2014). URL `http://www.prensilia.com/files/support/doc/PRENSILIA{_}IH2{_}basic{_}16.pdf`
21. Schunk: Technical data Schunk SVH (2014). URL `https://schunk.com/br{_}en/gripping-systems/highlights/svh/`
22. Shadow Robot Company: Shadow Dexterous Hand Technical Specification. Tech. Rep. August, Shadow Robot

Company (2015). URL http://www.shadowrobot.com/products/dexterous-hand/
23. Taska Prosthetics: Taska Hand (2017). URL http://www.taskaprosthetics.com/
24. Touch Bionics: i-limb (2018). URL http://www.touchbionics.com/products/active-prostheses/i-limb-ultra
25. Vepa, R.: Biomimetic Robotics. In: Newnes (ed.) Engineered Biomimicry, pp. 81–105. Elsevier (2013)
26. Yang, S., Wong-Lin, K., Rano, I., Lindsay, A.: A single chip system for sensor data fusion based on a drift-diffusion model. In: 2017 Intelligent Systems Conference (IntelliSys), pp. 198–201 (2017)
27. Zhang, T., Jiang, L., Fan, S., Wu, X., Feng, W.: Development and experimental evaluation of multi-fingered robot hand with adaptive impedance control for unknown environment grasping. Robotica **34**(5), 1168–1185 (2016)

Sergio Pertuz Mendez has MSc. in Mechatronics Systems at the University of Brasilia (UnB), also has a B.Sc. in Mechatronics Engineering at the University of Pamplona (UPA). Has experience in the Systems on Chip (SoC) area, microchip programming and FPGA, mechanical and electrical designs and projects and process automation.

Prof.Dr. Carlos H. Llanos received his B.s. degree in Electrical Engineering from the University of Valle, Cali, Colombia, in 1983; M.Sc. degree in Computer Science from the University of Minas Gerais - UFMG, Minas Gerais, Brazil, in 1990; and a PhD degree in Electrical Engineering from the University of Sao Paulo, Brazil, in 1998. He lectures in the graduate program in Mechatronics Systems in the Department of Mechanical Engineering at University of Brasilia - UnB, Brasilia, Brazil, since 2003. He is a member of the IEEE. His current research interests include reconfigurable system for automation, intelligent systems, instrumentation, and embedded systems design.

César Augusto Peña Cortés is currently a full professor in the Department of Mechanical, Mechatronics and Industrial Engineering at the University of Pamplona (since 2004). He is part of the Automation and Control research group. He holds a PhD in Automation and Robotics from the Universidad Politcnica de Madrid, Spain (2006). His research topics revolve around service robots, artificial vision and neurosignals, in which he has several publications in journals and congresses lectures

Daniel M. Muñoz received a B.S degree in Physics Engineering from Universidad del Cauca, Colombia, and the MS.c and PhD degrees in Mechatronics Systems from University of Braslia, Brazil, in 2003, 2006 and 2012, respectively. Since 2012, he has been at the University of Brasilia, where he is currently an assistant professor at the Electronics undergraduate program, Faculty of Gama, and at the Mechatronics graduate program, Department of Mechanical Engineering. His current research includes digital circuit design using FPGAs, artificial intelligence and bio-inspired optimization algorithms.

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Development of a robotic hand using bioinspired optimization for mechanical and control design: UnB-Hand

SERGIO A. PERTUZ[1], CARLOS H. LLANOS[1], AND DANIEL M. MUÑOZ.[1,2]

[1]University of Brasilia, Faculty of Technology, Postgraduate in Mechatronic Systems

[2]University of Brasilia, Faculty of Gama, Electronics Undergraduate

Corresponding author: Sergio A. Pertuz (e-mail: sergio.pertuz@unb.br).

The authors would like to thank the Coordination for the Improvement of Higher Education Personnel of Brazil - CAPES, the Foundation for the Support for Research - FAP-DF, and the Decan of Pos-Graduate Courses of the University of Brasilia - DPG-UnB for its financial support.

ABSTRACT For the last four decades, the development of robotic hands has been the focus of several works. However, a small part of those approaches consider the exploitation of parallelism of FPGA-based (Field Programmable Gate Arrays) systems or discuss how using bioinspired optimization algorithms could improve the mechanical and controller components. This work considers developing a bioinspired robotic hand that achieves motion and force control with a logic hardware architecture implemented in FPGA intended to be replicated and executed with suitable parallelism, fitting a single device. The developed robotic hand prototype has five fingers and seven DoF (Degrees of Freedom). Using bioinspired optimization, such as PSO (Particle Swarm Optimization), both the rigid finger mechanism and the impedance controller were optimized and incorporated the results in several practical grasping experiments. The validation of this work is done with the Cutkosky grasping taxonomy and some grasping experiments with interference. The tests proved the proficiency of this works for a wide range of power and some precision grasp. The reader can see the experiments in the attached videos.

INDEX TERMS Bioinspired optimization, FPGA, Grasping taxonomy, Impedance controller, Robotic hand, SoC

INTRODUCTION

OVER the past four decades, there have been significant contributions in the field of computer science, artificial intelligence, robotics, and other related fields. This progress has allowed the development of robotic systems that are more capable and sophisticated, such as biomimetic robotics. Such robots are more skilled, robust, and efficient than other types of conventional robots when used in unstructured workplaces [1].

There is also much effort towards building biomimetic robotic hands, which have contributed to a better understanding of implementing a human hand into a dexterous gripper/manipulation robot, widening its applications through improved sensors and mechanisms [15]. One of these improvements is tactile sensing, which uses physical signals to identify the phases of object manipulation, namely: (a) non-contact to contact, (b) rotation, and (c) sliding [16]. For the application of robot hands, these phases translate into object recognition, force control, and grasp. Tactile sensors can measure different magnitudes, such as force vectors, vibrations, and contact actions. On the other hand, signal-processing techniques and modeling improve these measurements or even estimate others when sensors cannot read them directly. Such measures are then used for control schemes that perform grasping tasks [17]. Efficient grasping techniques have proven to be complicated on both implementation and computational issues, considering that the diverse robotic hands' components must be controlled and supervised in parallel and real-time. Some works have accomplished these aspects by clustering several data processing devices in parallel. Table 1 presents a summary of some of those works with their main characteristics and achievements. The table includes the following key points: (1) control scheme or strategy for grasping, (2) finger gear or type of transmission used for the fingers movement, (3) the type of actuators used, (4) the quantity and type of sensors

TABLE 1. Comparison with some other robotic hand implementations

Author	Control scheme	Finger gear	Actuators	Sensors	CU	Freq. (Hz)	DoF
Robotic Hand (This Work)	Impedance control	Four-bar linkage and worm gear	7 DC motors	7 angular position, 7 current sensors	SoC ARM+FPGA	>25000	7
KITECH-Hand [2]	PID current control	Spur gears	16 DC motors	16 angular position, 16 current sensors	μCU matrix (16)	333	16
Calderon *et al.* [3]	PID control	Four-bar linkage	5 DC motors	5 force sensors	PC	-	5
Jeong *et al.* [4]	Neural Networks for position estimation and PID current control	Four-bar linkage	6 DC motors	5 position sensors	PC	-	6
Wang *et al.* [5]	Impedance control with internal PID position control	Four-bar linkage	5 DC motors	5 Encoders; 5 hall position; 5 torque sensors	Multiple DSP (2)	10 - 40	5
LMS Hand [6], [7]	Force control with Neural Networks and PID position control	Wire-driven	16 DC motors	16 encoders (for motors), 16 absolute encoders (for joints)	PC	50	16
Lee *et al.* [8]	Hybrid PD position/force	Four-bar linkage	9 DC motors	9 incremental encoders, 4 resistive force sensors	PC	-	9
HIT/DLR Prosthetic Hand [9], [10]	Impedance control and PD position control	Four-bar linkage	3 DC motors	3 encoders, 3 hall position, 3 torque, 3 force sensors on fingertips	Multiple DSP (2)	1000	3
HIT/DLR Prosthetic Hand II [11], [12]	Impedance control and PD position control	Wire-driven	15 Brushless DC motors	15 position, 15 torque, 5 force on fingertips, 10 temperature sensors, tactile sensors.	Multi-processor DSP+FPGA (6 DSP & 6 FPGA)	-	20
KNU Hand [13], [14]	Position control with sliding detector	Worm gear and four-bar linkage	2 DC motors	2 position sensors	DSP	-	6

used, (5) the CU or computational control unit, (6) the control loop refresh frequency in Hz, and (7) the number of DoF. It is worth noticing that only works that included the actuators either in the palm or the finger were considered; i.e., robotic hands with actuators in the forearm or outside the hand were not considered.

Table 1 indicates that some implementations use PID for force control [2]–[4]. Although this solution is simple, efficient, and easy to tune, it does not control the dynamics of the contact between the manipulator and object. The impedance controller cannot only do this but also performs well at exerting forces on the environment and achieve good robustness at handling flexible components with unknown stiffness.

On the other hand, Table 1 also shows that the computational unit (CU) of some approaches [2], [5], [12] use several components to achieve the required parallelism of the controllers by using well-known micro-controllers. However, it results in a large physical space required by the CU and demands implementing communication strategies between the processing devices. Lastly, as expected, more components mean an increase in energy consumption.

Many of the implementations depicted so far include simplifying the complexity of the human hand to achieve embedding. This work's motivation follows that path by scaling down the robotic system's dimensions and complexity, enclosing the CU in a single device or chip. This device should execute complex algorithms fast enough to attend to the control loop and real-time grasping and manipulation requirements. As mentioned before, integrating and centralizing the CU also avoids communication lag between different devices. Finally, reducing the number of components can also aim for a more cheap and energy-efficient robotic hand solution.

Field Programmable Gate Arrays (FPGAs) are a good match in pursuing this motivation. Additionally, these devices can be fundamental in the simultaneous control of several actuators that must be handled synchronously. In conventional processors, the correct synchronization can be hampered by the serial nature of executing instructions in von Neumann-based architectures. In this way, FPGA-based platforms allow the designer to implement efficient digital architectures capable of reading signals in parallel from multiple sensors and generating many output signals through parallel processing. They have been used to implement algorithms for parallel motion control of fingers for piano playing [8]. Another example [18] asserts the importance of the usage of parallelism for tactile sensing in robotic hand applications. Other potential advantages of FPGAs, although not guaranteed, are the possibility of achieving more energy-efficient architectures that can be scalable to more complex systems. Nevertheless, the main drawback of using FPGAs to embed control algorithms is the necessity of proficiency in hardware development from the designer and longer design time, which is greater than developing software.

This work's primary goal is developing a bioinspired robotic hand that achieves motion and force control based on the previously developed logic hardware architecture validated for a single finger [19]. In this sense, this work explores a suitable manner to parallelize the impedance controllers for a complete 7-DoF robotic hand, fitting a single FPGA device. The developed robotic hand prototype has five fingers and seven DoF (Degrees of Freedom). The prototype consists of seven DC motors, seven position-sensors, and six current sensors processed in parallel.

This work's contributions are the following: (a) A novel experimental setup process for implementing a bioinspired robotic hand with an embedded controller using reconfigurable hardware FPGA/SoC (System on a Chip). This approach's main advantage is its capacity to control several DoF in parallel using a single chip that's relatively cheap and energy-efficient, different from other works that need to use a grid of micro-controllers or big processors. (b) The robotic hand fingers are based on the novel bioinspired optimized mechanism in [20], which was vital in reducing the number of needed actuators and allow their inclusion in the palm. Additionally, it also allowed incorporating the sensors and electronics in the palm, avoiding external elements. (c) A new approach for tuning an impedance controller's parameters using a bioinspired Opposition-based learning Particle Swarm Optimization algorithm (OPSO). We portray the OPSO tuning methodology and believe that it be extended for other applications where impedance control can be useful. There is no literature about tuning impedance controllers using bioinspired algorithms to the best of our knowledge. In this work, this controller is used to achieve the robotic hand's dynamic and force control to perform grasp without having tactile sensors. (d) Finally, this work presents real experiments of the proposed FPGA/SoC-based bioinspired robotic hand for the first time. The conveyed experiments consisted of testing several grasping positions using the Cutkosky taxonomy and executing some of those grasps with external interference. With these experiments, we successfully demonstrate the robotic hand capabilities analytically and critically.

This paper is organized as follows: Section I describes the robotic hand mechanical and electronic design, describing the fingers' optimization and listing the used electronic devices. Section II depicts the impedance controller that implements the force control and its tuning using bioinspired algorithms. Section III develops the embedded FPGA/SoC-based system that executes this control, using hardware/software co-design. Section IV unfolds a performance analysis of the previously mentioned embedded system, comparing it to other solutions. Finally, Section V describes the experiments that were carried out to validate this work results.

I. HAND DESIGN

Human hands have 31 muscles and 19 articulations that actuate at least 25 DoF (Degrees of Freedom) [21]: four in every upper finger, of which three are responsible for flexion-extension and one for adduction-abduction; four in the thumb, where two are for flexion-extension and two for opposability. The rests lay on the palm and wrist's rotation and translation.

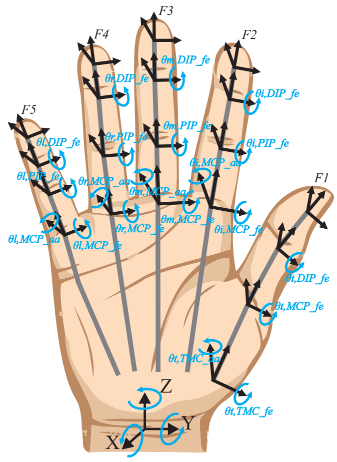


FIGURE 1. DoF of a human hand. Where DIP is Distal-Inter-Phalangeal joint, PIP is Proximal-Inter-Phalangeal joint, MCP is MetaCarpal-Phalangeal joint, TMC is CarpoMetaCarpal Joint, *fe* is flexion-extension, and *aa* is abduction-adduction.

Fig. 1 illustrates the large number of joints or DoF of a human hand. Ideally, a biomimetic hand should emulate all DoF. However, this achievement has proven to be very difficult due to space, energy, and other physical restrictions. Recent solutions to this problem have involved reducing the number of fingers [2], locating the actuators outside the hand to achieve a similar amount of DoF in the robot [22], [23], transforming and reducing the number of DoF [3], [4], [16], [24].

For the final solution, this work takes advantage of the fact that some joints in the human hand restrict the state of others [21]. Therefore, their movement set is constrained and defined mathematically regarding other joints. With this in mind, different human hands' DoF can be hierarchized according to the relevance of its actions when performing particular tasks, such as grasping [25]. Table 2 lists the most relevant joints of the human hand, where *aa* and *fe* states the movements of the MCP and TMC joints for the front and thumb fingers, respectively. The joints with a "no" are sub-actuated according to the others' state, and *yes* indicates the actuated joints. Table 3 defines the mathematical relation of the constrained ones.

TABLE 3. Interphalangeal constraints for the actuated joints. Adapted from Cobos [25]

Thumb	Index	Middle	Ring	Little
θ_{t,TMC_aa}				
	θ_{i,MCP_aa}			
$\theta_{t,MCP\_fe} \approx \frac{4}{5}\theta_{t,DIP\_fe}$	$\theta_{i,MCP\_fe} \approx \frac{4}{3}\theta_{i,PIP}$	$\theta_{m,MCP\_fe} \approx \frac{4}{3}\theta_{m,PIP}$	$\theta_{r,MCP\_fe} \approx \frac{4}{3}\theta_{r,PIP}$	$\theta_{l,MCP\_fe} \approx \frac{4}{3}\theta_{l,PIP}$
$\theta_{t,DIP}$	$\theta_{i,PIP} \approx \frac{3}{2}\theta_{i,DIP}$	$\theta_{m,PIP} \approx \frac{3}{2}\theta_{m,DIP}$	$\theta_{r,PIP} \approx \frac{3}{2}\theta_{r,DIP}$	$\theta_{l,PIP} \approx \frac{3}{2}\theta_{l,DIP}$
	$\theta_{i,DIP}$	$\theta_{m,DIP}$	$\theta_{r,DIP}$	$\theta_{l,DIP}$

TABLE 2. Ten more relevant human hand joints when performing grasping tasks. Refer to Fig. 1 for the positions of the joints. Adapted from Cobos [25]

No.	Finger	Articulation	Is used in this work
1	Thumb	θ_{t,TMC_aa}	yes
2	Thumb	θ_{t,TMC_fe}	yes
3	Index	θ_{i,MCP_aa}	yes
4	Index	θ_{i,PIP_fe}	yes
5	Middle	θ_{m,MCP_fe}	yes
6	Middle	θ_{m,PIP_aa}	no
7	Ring	θ_{r,MCP_fe}	yes
8	Ring	θ_{r,PIP_aa}	no
9	Little	θ_{l,MCP_fe}	yes
10	Little	θ_{l,PIP_aa}	no

Figure 2 presents the proposed kinematic structure. It depicts the five fingers and their joints as sixteen rotatory joints, seven actuated while the rest behaves according to the constraints seen in Table 3.

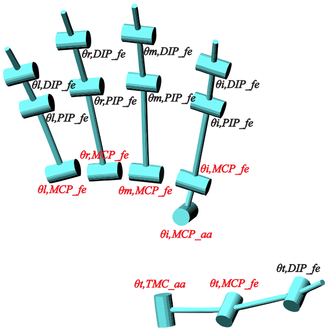


FIGURE 2. Kinematic structure of 7 DoF simplified robotic hand. Actuated joints are highlighted in red, the other ones are sub-actuated according to Table 3

Table 4 shows the ranges of the joints and phalangeal length of the fingers, which were established based on real human hand for Latin-American individuals [26].

TABLE 4. Kinematic Parameters of Robotic Hand.

Joint	Phalangeal lengths of the finger [mm]					range [^{o}]
	Thumb	Index	Middle	Ring	Little	
MCP_aa	-	17	-	-	-	$0-15$
TMC_aa	38	-	-	-	-	$0-90$
MCP_fe	37.67	45.77	51.36	47.60	37.67	$0-90$
PIP_fe	-	25.75	30.30	29.51	20.84	$0-67.5$
DIP_fe	20.84	18.20	20.02	19.90	18.36	$0-45$

The parameters on Table 4 establish the robot's workspace, which is the total volume the fingertips can reach when combining every possible joint configuration [27]. Fig. 3 exhibits the proposed robotic hand workspace using a cloud of fingertips' points. It illustrates the intersections between the upper fingers and the thumb, validating its opposability. This figure also denotes that the middle, ring, and little fingers' workspace is a 2-dimensional spline due to the single DoF they possess. The following subsection describes the resulted design for the fingers using a four-bar linkage mechanism and how it was optimized to perform the constraints depicted in Table 3.

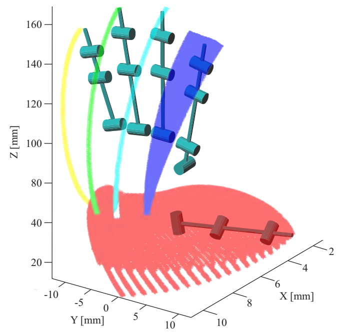


FIGURE 3. Robotic hand workspace.

A. MODELLING AND OPTIMIZATION OF ROBOTIC FINGER MECHANISM

Pertuz et al. [20] proposed a robotic finger mechanism that located the actuators in the palm of the robot hand, which is useful when the robotic hand needs to be adapted to larger and more complex systems [15]. The drawback of this approach is that the space in the palm and finger of a human-sized robot hand is minimal, thus the necessity of reducing the number of DoF.

The *fe* movements of the fingers are sub-actuated with 1 DoF, i.e. they have a single motion input and the other joints move at a certain proportion, through a four-bar linkage mechanism. Fig. 4 presents the mechanism and its motion path, the input of the mechanism is θ_1 (from here onwards the *MCP*, *PIP*, and *DIP* joints will be referred as θ_1, θ_2, and θ_3, see Fig. 1).

Fig. 4(c) also describes two fingertip trajectories: 1) the one developed by the mechanism using trivial link lengths named as *mechanism's fingertip path*, and 2) the one generated by the constraints on Table 3 referred as *desired fingertip path*. For both trajectories, the proximal, medial, and distal phalanges have the values listed in Table 4. The first path depends on the mechanism's link lengths and is expected to fit the second. The solution to this problem is not straightforward; however, it can be seen as an optimization problem where the variables are the link's positions and the objective is to fit the fingertip path.

TABLE 5. Optimized decision variables for the coupled 4-bar mechanisms of the robotic hand for all fingers. Values are in mm.

	Proximal-Medial Link				Medial-Distal Link			
	$P_{a2\hat{i}}$	$P_{a2\hat{j}}$	L_{a1}	L_{b1}	$P_{a2\hat{i}}$	$P_{a2\hat{j}}$	L_{a1}	L_{b1}
Ring	3.26	3.44	46.09	5.98	3.12	3.44	27.64	6.59
Index	3.14	3	44.32	5.75	3	3	24.12	5.75
Middle	2	2	50.1	5.56	3.75	3.29	28.52	5.75
Little	2.56	2.93	35.95	6.33	4.1	2.82	17.64	6.14
Thumb	2.56	2.93	35.95	6.33				

In order to optimize the four-bar mechanism, the bioinspired optimization algorithms PSO (Particle Swarm Optimization) [28], DE (Differential Evolution) [29] and GA (Genetic Algorithm) [30] were used. The results of each algorithm were analyzed, compared, and selected for a final

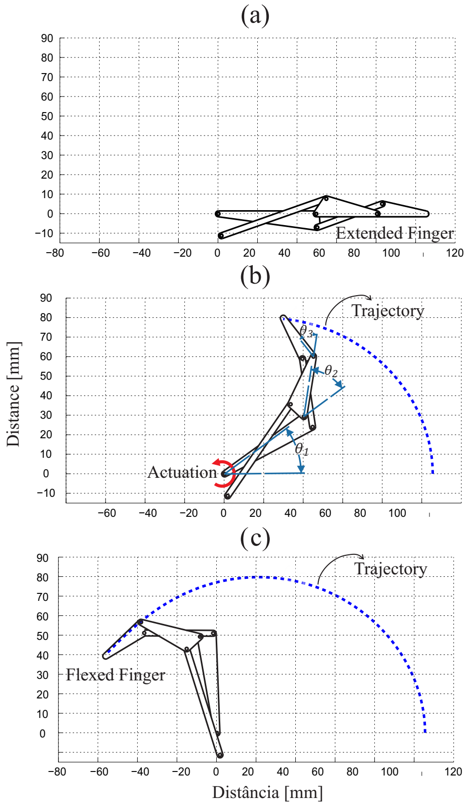


FIGURE 4. Coupled 4-bar mechanism. (a) Finger fully extended; due to the mechanisms self-lock feature, in this position, it only allows counter-clock wise movement input, (b) Finger upon input movement in θ_1, and (c) Fully flexed finger.

prototype. The optimization was performed for one finger and adapted in proportion to the others. The decision variables are listed in Table 5 and Fig.5 illustrates the optimized mechanism's fingertip path for the index finger. The readers are referred to [20] for more details regarding the mechanical design optimization procedure.

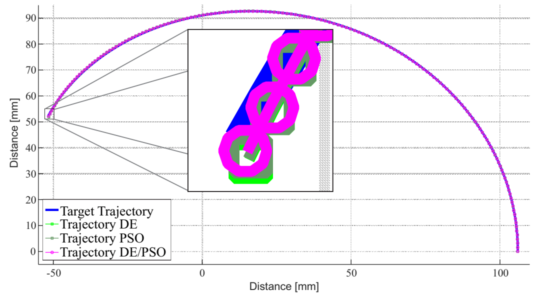


FIGURE 5. Optimized fingertip paths using PSO and DE

B. DESIGN AND ASSEMBLING

This project enclosed the novel four-bar finger mechanism optimized in previous work [20]. It allowed to reduce the DoF and thus the mechanical components of the flexion-extension and adduction abduction movements, permitting them to be enclosed in the palm. The robotic hand was built using mainly 3D printed parts; however, we used other manufacturing processes for the metallic pieces. The finger mechanism depicted in the previous subsection is anthropomorphized for a better appearance, as shown in Fig. 6.

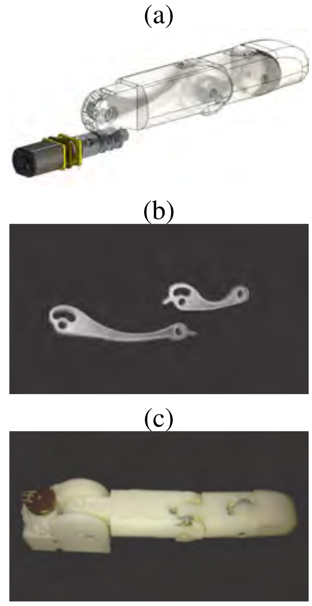


FIGURE 6. Antropomorphed index finger mechanism. (a) CAD with transparent finger depicting the phalanges and the links of the mechanism, and photos of (b) the proximal-medial and medial-distal links, and (c) the assembled finger. Most of the robotic hand was 3D printed with ABS plastic except for the worm gear and links that are fabricated in aluminum

The motors used in this project are low power brushed DC motors (200 RPM and 0.28 Nm) and actuate the mechanism through a worm gear to bring more grasping torque. The worm mechanism also allows to rotate of the movement axis 90^o, easing the location of the motors (see Fig. 6(c)).

The thumb and index fingers' configurations have extra actuated joints for their *aa* movements. This project proposes three different finger configurations (see Fig. 2 for viewing the joints), as follows. (1) Configuration 1 (Thumb): 2 DoF (*aa* and *fe*) and 3 joints. (2) Configuration 2 (Index): 2 DoF (*aa* and *fe*) and 4 joints. (3) Configuration 3 (Remaining fingers): 1 DoF (*fe*) and 3 joints.

The extra DoF of the thumb and index fingers were added as seen in Fig. 7(a). Fig. 7(b) also illustrates the prototype robotic hand's final result.

The thumb opposability of this work can be quantified using an adaptation of the well-known Kapandji clinical test [31]. This test assesses the thumb's opposition by checking its ability to touch a specific part of the hand. They are ten tests/positions in increasing difficulty, with the easiest consists of touching the proximal phalanx of the index finger and the hardest is to touch the distal palmar crease. Fig. 8 depicts two of the most representative Kapandji tests (six and ten) validating this work's thumb opposable ability by achieving the most challenging position.

C. ELECTRONICS

The robotic hand electronics are embedded in the palm and, excepting the position sensor, are located in a custom PCB.

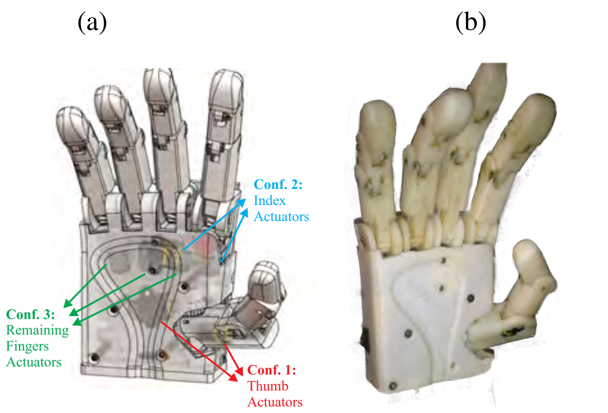


FIGURE 7. Assembled Robotic Hand. (a) Render of assembled robotic hand with the locations of the actuators and extra DoF of the index and thumb. (b) Real picture of the assembled robotic hand.

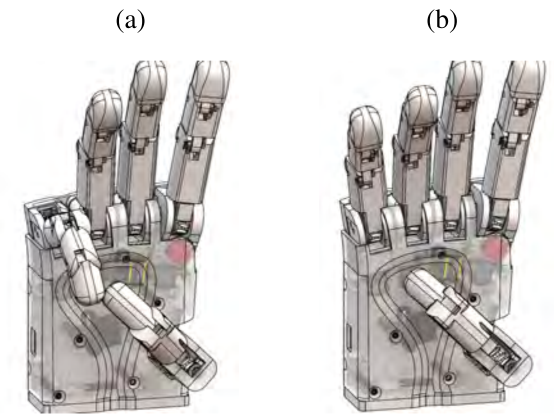


FIGURE 8. Kapandji Test for robotic hand. (a) Position six: thumb touches the little finger. (b) Position ten: thumb touches distal palmar crease.

The signals are accessed through an IDC 34-pin header that connects to the controller device. Figure 9 illustrates the schematic of the controller dividing it into two parts the Acquisition Unit (AU) and the Controller Part (CU).

The AU includes the motor drivers (DRV8833), the current sensors (ACS712), and some other power regulation ICs (Integrated Circuits). It also has connection pins for the IDC breakthrough and sockets for the DC motors, and the analog angular position sensors. The PCB has dimensions of approximately $80 \times 52mm$ and has an irregular form designed to fit in the palm (See Fig. 9). The CU consists of the Arty Z7-20 development board. It contains an XC7Z010 SoC (System on a Chip) that includes a Zynq®-7000, an Artix™-7 FPGA (Field Programmable Gate Arrays), and a Dual-Core ARM®Cortex®-A9 at 667MHz. The chip also includes a XADC (Analog to Digital Converter) necessary for the current and position sensors.

II. DESIGN OF THE ROBOTIC FINGER FORCE CONTROLLER

Several approaches can be implemented when attempting to grasp with a robotic hand [32]. This work considers a scheme that plans to grasp an object whose physical characteristics

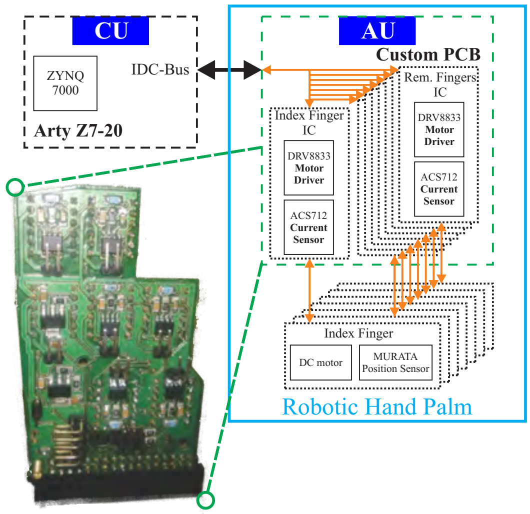


FIGURE 9. Schematic diagram the Robotic Hand Acquisition Unit (AU) and Controller Unit (CU). The entire AU is embedded in the palm while the CU is the Arty board that is located outside. On the south-west corner is a photo of the custom PCB of robotic hand.

are unknown using only position and motor current to estimate the torque generated by the fingers, avoiding tactile sensors. The amount of torque/force is controlled by the algorithms in order to sustain a stable grasp. Previous works have already considered this approach [33], [34].

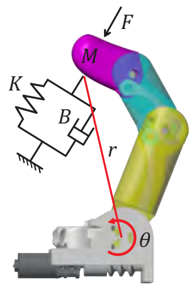


FIGURE 10. Schematic diagram of the impedance model in the robotic finger.

Impedance control is adopted to control the torque produced by DC motors of the fingers. An impedance controller aims to control the dynamics a robot has when interact with its environment [35], [36]. Fundamentally, the impedance controller is defined as a second-order dynamic system, such as a mass-damper-spring system, with adjustable parameters. A decoupled and linear behavior is considered in this work, i.e., every finger has an independent controller with torque and position feedback, see Fig. 10. The controller is represented by Eq. 1

$$Mr\ddot{\theta}(t) + Br\dot{\theta}(t) + Kr\theta(t) = F(t) \quad (1)$$

, where F is the resulting force of the system, M, B and K are mass, damping, and spring coefficients, respectively, and r is the distance between the first joint and the fingertip.

The control parameters, M, B, and K, can have any value; however, their tuning can be complicated for stable system response. The tuning of control parameters of several types (PID, Neural Networks, and others) using bioinspired optimization algorithms have been implemented in [37], [38]. This work proposes implementing an OPSO (Opposition-based learning Particle Swarm Optimization) algorithm to tune the impedance controller's parameters with torque feedback and a closed-loop to achieve the desired response for a step input.

A. IMPEDANCE CONTROLLER SCHEME AND SIMULATION

The implementation of Eq. 1 as the controller of the system is done using a discrete integration method [39]. This is described through three steps:

1) Discretize Eq. 1 as follows:

$$\alpha^*(t+1) = (Mr)^{-1}(\overline{F}(t) - Kr\,\overline{\theta}(t) - Br\,\overline{\omega}(t)) \quad (2)$$

where the Force F on Eq. 1 is replaced with the Force Error at a moment t, $\overline{F}(t)$, $\overline{\theta}(t)$ is the tracking position error and $\bar{\dot{\theta}}$ is the tracking angular speed error.

2) The product of the previous step is the target angular acceleration at the next step $t+1$; $\ddot{\theta}^*(t+1)$, which is integrated to obtain the tracking angular velocity:

$$\omega^*(t+1) = \int_{t-1}^{t} \alpha^*(t+1)d\theta \quad (3)$$

3) Finally, the tracking angular position, $\theta^*(t+1)$, is obtained integrating the tracking velocity.

$$\theta^*(t+1) = \int_{t-1}^{t} \omega^*(t+1)d\theta \quad (4)$$

Fig. 11 presents the block diagram for the control scheme for a k DoF and its data-flow. This model aims to detect objects that obstruct the movement upon closing the finger and maintain a desired gripping force i^* (measured with the motor current $\hat{i}_k$ directly considering it is directly proportional to the force F in Eq. 1).

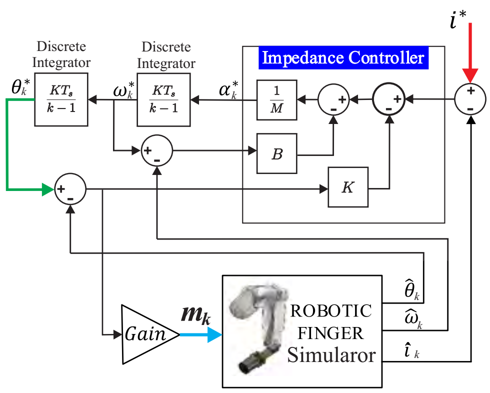


FIGURE 11. Dataflow of finger simulator impedance controller (k=1 to 7). The colored lines highlight the data-path that the control interface override for testing (See Chapter III.C).

The *Robotic Finger Simulator Block* includes the friction forces of the mechanism and the action of an object that hampers its movement. The simulator was designed in Matlab/Simulink and includes current, position, and speed transducers. It performs a flexion movement with an object that hampers its path. The simulator is better described in [40].

B. OPTIMIZATION OF IMPEDANCE CONTROLLER

The tuning of the impedance controller parameters is a complicated and expensive procedure. One of the main reasons is that the designer must consider coupling effects between multiple-joints. Tuning can become even more complex when the robot dynamic has to behave with high-accuracy and high speed. The reader is referred to [41]–[43] to find several approaches for auto-tuning impedance controllers. This problem can be addressed with bioinspired optimization algorithms such as PSO (Particle Swarm Optimization), as explored in this work's mechanical design. Some related applications of PSO tuning robots controllers can be found in [44]–[46]. With these examples as a basis, it is possible to extend PSO to tune impedance controllers with little effort.

Algorithm 1 Pseudo-code for the OPSO algorithm

function OPSO(S,N,c_1,c_2,Max_{iter},$OBLMax_{iter}$,$threshold$)
  Start swarm;
  $iter = 1$
  $OBL_{iter} = 1$
  repeat
    for i **do** 1 S
      if $f(x_k) \leq f(y_{ik})$ **then**
        $y_{ik} \leftarrow x_k$
        $OBL_{iter} = 1$
    calculate y_s using the S fitness values $f(y_{ik})$
    for i **do** 1 S
      for j **do** 1 N
        $v_{ij}^{(t+1)} \leftarrow wv_{ij}^{(t)} + c_1U_{1j}(y_{ij}^{(t)} - x_{ij}^{(t)}) + c_2U_{2j}(y_{sj}^{(t)} - x_{ij}^{(t)})$
        $x_{ij}^{(t+1)} \leftarrow x_{ij}^{(t)} + v_{ij}^{(t+1)}$
  $OBL_{iter} = OBL_{iter} + 1$
  if $OBL_{iter} > OBLMax_{iter}$ **then**
    $x^{(t+1)} \leftarrow a + b - x^{(t)}$
    $OBL_{iter} \leftarrow 1$
  $iter = iter + 1$
until $(f(y_{y_s}) < threshold)||(Iter <= Max_{iter})$
return y_{best}

The central part of an optimization problem is the design of the cost function. For this case, it is to track the desired force i_k^* with the decision variables being the controller's coefficients (M, B, and K). It is calculated by observing the estimated motor current and extracting some characteristics from the response, such as (a) the over-impulse percentage (O_i), (b) the necessary time for the finger to reach collision with the object (t_o), (c) the settling time after the collision is reached (t_e) and, (d) the force tracking Mean Squared Error (MSE). The selection of these criteria was made empirically and are weighted according to the following equation

$$f_{cost} = 0.05(O_i) + 0.15(t_o) + 0.15(t_e) + 0.65(MSE) \quad (5)$$

, where the weights were decided empirically.

Due to the lack of knowledge about the coefficients, a random initial position is fed to the algorithm. OPSO is a modification to the original PSO algorithm that attempts to avoid falling on local minimums. This diversification uses the opposition based learning (OBL) method [47] when an individual best is not improved after a certain quantity of iterations. The following algorithm describes how the OPSO works.

The minimum error achieved by the PSO after 500 iterations was 0.0725 with the decision variables of $K = 5.0774$, $B = 0.3450$ and $M = 0.0052$. Finally, the resulting coefficients were tested for the simulator's controller with a collision time of $3.565 seconds$, a settling time of $0.335 seconds$, and a steady-state error of 0.012.

III. IMPLEMENTATION OF THE EMBEDDED ROBOTIC FINGER FORCE CONTROLLER

The robotic hand aims to embed the low-level control algorithms. Some authors approach this issue by distributing the computation between several devices to comply with real-time constraints [2]. This work proposes, in novelty, implementing the filtering process and the impedance controller as a reconfigurable architecture in an FPGA/SoC device. Doing so enables implementing computing-intensive functions in parallel, in a single chip, with little compromise to time-execution performance and energy consumption.

The Filtering Process block (presented in Fig. 12) is the most expensive part of the control scheme. So, this part is determined to be described in VHDL (VHSIC "Very High-Speed Integrated Circuit" Hardware Description Language) on the CU's FPGA (Field Programmable Gate Arrays), from now on referred to as the Programmable Logic Part / PL-part). It is essential to point out that hardware implementation has some advantages, like a logic architecture's intrinsic parallelism. On the other hand, a drawback is that the designer must be mindful not to surpass the number of logical resources on the chip.

Moreover, the Impedance Controller block in Fig. 11 is executed in the CU's Processor Unit (from now referred to as Programmable Software Part / PS-part). The composition of the CU's architecture and the dataflow is described in Fig. 12. Similarly, the robotic hand *Control Interface* block is implemented in the PS-part. It can switch the control technique between manual, position, and impedance control; this is described more thoroughly in the following subsection.

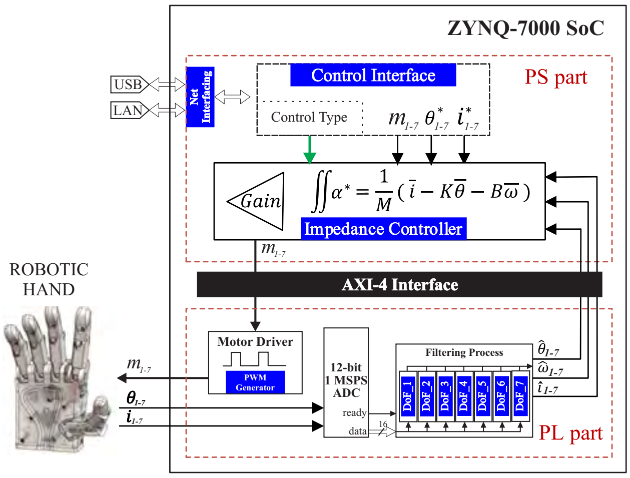


FIGURE 12. Internal architecture of the CU. The Filtering Process block is replicated for the same amount of DoF (Degrees of Freedom) in the robotic hand. These blocks, including the Motor Driver, are controlled and synchronized via the AXI interface with the PS-part. The PS-part also executes the impedance controller algorithm and the control interface.

The platform-chip used in the CU allows communication between the PS- and PL-part, enabling the creation of hybrid systems combining the advantages of software and hardware, i.e., the possibility of implementing hardware accelerators in the PL-part and the flexibility of software.

A. PL-PART: SENSORS FILTERING PROCESS

The PL-part includes an ADC converter, the motor PWM signals generator (Motor Drivers), communication protocol (AXI Interface), and the sensors Filtering Process. The latter implements two filters for the analog current and position sensors: (1) a Kalman filter [48] that estimate position and velocity for the analog position sensor; and (2) a second-order low-pass filter for the current sensor.

The architecture uses FSM to control dataflow and operations. There are four arithmetical operators (two adders, one multiplier, and one divider) that can be used in parallel and shared between states and two filters. Additionally, the operators are custom-made [49] and use a 27-bit floating-point numeric representation. This bit resolution is used because it preserves the best *resource utilization/precision* ratio for the selected FPGA device. A more thorough description of the Filtering process is depicted in [19], [50].

B. PS-PART: IMPEDANCE CONTROLLER

The controller block involves the impedance controller scheme introduced in Section II. It performs all the scheme steps using the sensor filter outputs and the motor's current reference. It is important to note that it cannot be executed in parallel given the step's equations' data dependency, which is another reason not to implement it on hardware. The controller's output is fed to the motor driver block as seen in Fig. 12. The PS-part is also in charge of the data synchronization between modules through the AXI Interface.

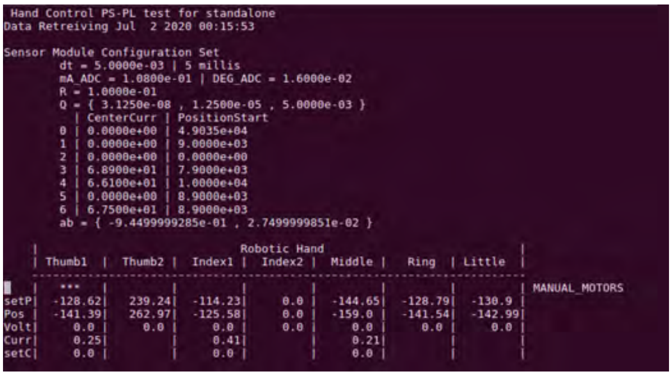


FIGURE 13. Robotic hand visual interface. It illustrates some relevant data for the sensors calibration and filters parameters. It also shows sensors and control data: *setP* and *Pos* are the desired and filtered angular position of a finger, *Volt* is the voltage of a finger's motor, and *Curr* and *setC* are the filtered sensor and desired current respectively. Finally, it also specifies the current control strategy for testing (MANUAL_MOTORS, P_ONLY or FULL_IMPEDANCE).

C. PS-PART: CONTROL INTERFACE

This work uses a text-based visual interface (see Fig. 13) that enables the user to control the robotic hand. The interface allows setting a series of predefined control values ($i*_{1,\cdot;\cdot7}$) for different grasps and manually controlling each finger separately. The "***" below the fingers' names indicates the currently controlled one. The control variable of that finger can be increased or decreased. The user can also declare which control variable will be adjusted between three options by overriding it in the impedance controller dataflow. This circumvention of the dataflow is depicted in Fig. 11 with the red, green and blue lines for i^*, θ^* and m^* respectively. Resuming the control can be done with three strategies:

1) MANUAL_MOTORS mode overrides the m^* variable (Seen as "Volt" on Fig. 11), allowing to control the voltage of the motors of the robotic hand manually; and hence, the flexion-extension movements of every finger. It sends a constant pulse to the motors vetoing the output of the proportional gain.
2) P_ONLY: Bypasses the desired position of a finger (θ_k^* and shown as "setP"on Fig. 11). Setting this control variable makes it possible to set a finger in the desired position, nullifying the impedance controller's action.
3) FULL_IMPEDANCE: Is the full proposed impedance controller scheme. It updates the control variable i^* ("setC" in Fig. 11) into the impedance controller. It feeds the desired current for grasping objects.

D. RESULTS AND SYNTHESIS ANALYSIS

An FPGA has limited resources, such as LUTs (Look-Up Tables), slice registers, DSPs (Digital Signal Processors), and BRAM (Block RAM). Table 6 presents the consumption of those resources for the implementation of the system described in Fig. 12 in the column under the name *PSPL-Arty*. Additionally, for contrast, the table also presents the resource consumption of the full-hardware implementation (without using the PS-part); these results are described in the column under the name *PL-Arty*.

TABLE 6. Synthesis Utilization of Full Controller System for One Finger and the Full Hand

Resource Utilization	PSPL-Arty 1 Finger (%)	PSPL-Arty Full Hand (%)	PL-Arty 1 Finger (%)	PL-Arty Full Hand (%)	ArtyZ720 Available
LUT	1976 (3,71)	12252 (23,03)	2888 (5,43)	22521 (42,33)	53200
LUTRAM	0 (0,00)	0 (0,00)	0 (0,00)	62 (0,36)	17400
FF	1917 (1,80)	10714 (10,07)	3522 (3,31)	25702 (24,16)	106400
DSP	2 (0,91)	14 (6,36)	3 (1,36)	21 (9,55)	220

For full-logic implementation, almost half of the FPGA resources were used, allowing little space for further development and other characteristics the work might need in the future. In contrast, the hardware-software implementation results (PSPL-Arty) drastically reduced the resource utilization of the FPGA.

IV. PERFORMANCE ANALYSIS OF THE CONTROLLER

This section compares the implementation of this control system using various architectures and platforms. The controller is implemented as three different structures, exploiting the modules created in the previous subsection: 1) The architectures named as "software-only" perform the systems using only software. 2) Similarly, the architectures named "Full-HW" perform the system using only the PL-part. 3) Finally, the architecture proposed in Fig. 12 is named Hybrid.

The software-only implementations are replicated in different platforms, comparing cost and performance. Table 7 lists the different architectures, platforms, and Operating Systems (OS) implemented in this work.

TABLE 7. Control system implementation on different platforms.

ID	Type of Arch	Platform	OS	Clock Frequency
PS-Arduino	Software -Only	ATmega2560	Bare metal	16MHz
PS-ArtyBM	Software -Only	ARM Cortex-A9	Bare metal	650MHz
PS-ArtyLnx	Software -Only	ARM Cortex-A9	Linux	650MHz
PL-Arty	Full -HW	XC7Z020	-	100MHz
PSPL-ArtyBM	Hybrid	ARM Cortex-A9 / 1CLG400C	Bare metal	650MHz / 100MHz
PSPL-ArtyLnx	Hybrid	ARM Cortex-A9 / 1CLG400C	Linux	650MHz / 100MHz

In total, six approaches were performed in this work. The comparison consists of executing the control loop and measuring the execution time of each iteration for one DoF (Degree of Freedom) and then for the seven. Table 8 lists the execution time consumed for every architecture.

TABLE 8. Control system implementation on different platforms.

ID	1 DoF t[us]	Freq[kH]	7 DoF t[us]	Freq[kH]
PS-Arduino	1017.00	0.98	7219.00	0.14
PS-ArtyBM	5.24	190.84	32.92	30.38
PS-ArtyLnx	13.72	72.88	87,80	11.39
PL-Arty	1.20	830.00	1.44	692.52
PSPL-ArtyBM	3.38	303.03	21.16	47.26
PSPL-ArtyLnx	5.85	170.94	39.00	25.64

All the PS (Programmable Software) approaches used the same code with small adaptations for some functions that were not compatible between platforms. Table 8 shows a big timing gap between the PS-Arduino approach and the rest, which is an unfair comparison in many cases. However, it allows the execution time to be compared with a much cheaper solution than the *Arty-Z720* used on the other approaches. The PS-Arduino strategy's execution demonstrates that it can only achieve a maximum control frequency of $140Hz$.

The different approaches in the *Arty-Z720* have each their advantages. For instance, the applications in Bare-Metal (*PS-ArtyBM* and *PSPL-ArtyBM*) can be executed in real-time; this guarantees the correct execution of the controller at a fixed frequency. Real-time implementation is not as simple for the cases with the Linux kernel used in the OS (*PS-ArtyLnx* and *PSPL-ArtyLnx*) because it is not real-time. However, using Linux in the platform provides more flexibility for interfacing and networking for future project applications. Connectivity for this project is essential given that this work is intended as part of a much bigger application where several elements need to communicate with each other.

The performance of the approaches using software-only is better on bare-metal than with Linux-OS. Nevertheless, the hybrid system of Linux with PL (*PSPL-ArtyLnx*) stretches this difference in the execution time. Additionally, the issue regarding real-time in Linux is minimized in this approach, given that the PL (Programmable Logic) part executes the control in real-time.

On the other hand, the dynamic energy is related to the user design's power consumption on the FPGA (Field Programmable Gate Arrays). In contrast, the static power represents the steady-state intrinsic leakage of the transistors on the device. For the hybrid system *PSPL*, the used static power is $0.115W$, and the dynamic is $1.467W$. Most of the latter is related to the *PS-Part* ($1.39W$), given that the Cortex A9 included in the chip is a high-performance multi-core processor. Nonetheless, the energy consumption of the proposed architecture is only $0.077W$.

A. COMPARISON TO RELATED WORKS

According to table 1, previous works [9], [10] have achieved the most considerable control loop frequency (around 1 kHz) for a robotic hand with 3 DoF, using two DSP devices, directly sensing position, torque, and force on fingertips. In contrast, this work accomplishes a control loop frequency of approximately 25 kHz of a robotic hand with 7 DoF, using a single chip device and sensing position and current to estimate torque.

In terms of DoF and control technique, works in [13], [14] developed a robotic hand with 6 DoF using only two DC motors and only one DSP device for controlling position with sliding detection. However, control loop frequency is not reported, and the dexterous is limited since only two DC motors are used. A robotic hand with 16 DoF with force and position control was developed in [6], [7]; however, it is limited in terms of space given that it uses a PC, achieving a control loop frequency of 50 Hz. An embedded solution to force control of a robotic hand with 16 DoF was developed in [2]. The authors used a PID current control and embedded the solution in a matrix of 16 uCs and achieving a control loop frequency of 333 MHz. Finally, works in [11] and [12] implemented an embedded solution for impedance and position control of a robotic hand with 20 DoF using 6 DSPs and 6 FPGAs devices. This massive parallel solution does not report the control loop frequency but probably increases the energy consumption and the controller board complexity.

Using a single chip for all DoF is a novel contribution of this work that saves the embedded controller board's real-estate and can be more efficient in energy consumption and performance. Additionally, the current system can be scaled; i.e., if the number of DoF increases, the hardware architecture can also be adapted. As reported in Table 6, there are enough hardware resources for implementing more filters and processing more DoF, still achieving a high control loop frequency.

V. EXPERIMENTS

The finger's control response using the proposed PSPL-ArtyLnx architecture is smooth and is tested to grasp various objects. Fig. 14 illustrates the control response of the sensors for one finger.

It can be seen it stabilizes for a set current that is proportional to the grip torque. The impedance controller sends the position signal to the proportional controller that calculates the motor's input voltage.

A. IMPEDANCE CONTROLLER INTERFERENCE TESTING

The *Arty-Z720*'s available ADC channels allows controlling the five fingers of the robotic hand simultaneously, enabling experiments that involve grasping objects with specific interference.

The test consists of setting a current setpoint for the impedance controller and then see how it behaves upon contact with a cylindrical object and with some type of interference afterward. It is important to highlight that the setpoint values are selected empirically. Figure 15 illustrates these experiments.

Figure 15(a)-(e) illustrates the first test of grasping a rigid cylindrical object, where immediately after contact (around the 20th second), a user carries out interference to evaluate if the robotic hand can maintain the grasp. First, in Fig. 15(b)

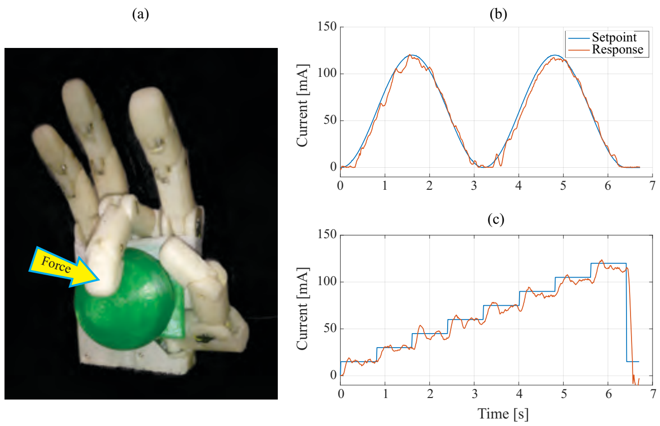


FIGURE 14. Control behavior of impedance controller of one finger. (a) Robotic hand experimental setup, (b)-(c) Current response to a sine wave and step input.

the object is pushed outwards and then sideways Fig. 15(c)-(d). The opposable thumb is the most affected finger in this type of interference, and it can be seen in its curve. However, the thumb can maintain the grasp and is stabilized after the interference is stopped, around the second 40.

The second experiment (Fig. 15(f)-(h)) details the curves for when the finger enters in contact with a non-rigid object on its surface. It is then performed a series of increased steps to see if the fingers interject more on the object upon increasing the grasping force. It can be seen that this is not the case because the initial setpoint is already high enough to interject the object the maximum amount. This value can not be less because the fingers will not start moving because of the work gear inertia.

B. CUTKOSKY TAXONOMY TESTING

Additional experiments are performed to test if the present work can simulate human grasping motion despite DoF reduction (Degrees of Freedom). The Cutkosky taxonomy [51] is a human grasping classification that has been extensively used for manipulation and machining tasks. The different types of grasping poses are classified into power and precision grasps. The experiments are executed by this work's robotic hand by maintaining an object's static grasping pose from the taxonomy.

Figure 16 shows the robotic hand performing some poses from said taxonomy. The robotic hand can perform five power-grasp and four precision-grasp successfully, demonstrating its ability to grasp and manipulate some objects, enabling a vast amount of applications. These results also show the relevance of the *TMC_aa* for most grasping poses.

CONCLUSIONS

In this project, a biomimetic robotic hand was implemented using an FPGA/SoC-based (Field Programmable Gate Arrays / System on a Chip) approach using a Zynq chip from Xilinx. The biomimetic hand's mechanical design contains 7 DoF (Degrees of Freedom) compared to the 24 an actual human hand has. The number and specific joints of the project were selected according to the most significant DoF and how well they perform gripping objects. Given that some movements are constrained by other DoF in this project, the fingers execute the flexion-extension movements with 4-link mechanisms. A bioinspired optimization was applied to solve a mechanism that generates the necessary trajectories for emulating a human finger's action. The optimization process reached an average quadratic error of $0,00266$. This result was extended for the five fingers' mechanical design according to their size, allowing the full robotic hand's implementation similar to a real hand. Reaching dimensions of $210 \times 84 \times 38mm$ with $413.13gr$ (for the robotic hand design) versus $175 \times 89 \times 43mm$ with $409,5gr$ (for a regular male hand). Additionally, the thumb's ability of this project was demonstrated with the Kapandji test. The prototype was able to reach all ten levels of finger positioning.

The impedance controller developed in this work was first evaluated via numerical simulation. For this action, a robotic finger simulator was implemented for tuning the controller parameters without endangering the prototype. The impedance controller's tuning was performed with bioinspired optimization algorithms, precisely the OPSO (Opposition-based learning Particle Swarm Optimization). The optimization process resulted in an underdamped motor current response, efficiently achieving a rising time of $355ms$ and a steady-state error of $1,2\%$. This bioinspired tuning method for an impedance controller of a robotic system is a new approach and one of this work's main contributions.

As reported in Section V, the embedded controller was developed through several approaches: PS-Arduino, PS-ArtyBM, PS-ArtyLnx, PL-Arty, PSPL-ArtyBM, PSPL-ArtyLnx. The experiments conducted with the proposed PSPL architectures of the impedance controller proved that the tuning performs correctly outside of the simulator. However, complex physical phenomena such as dynamic friction, gaps, and tolerances in the physical prototype (specifically the worm gear), introduced several oscillations in the system's response. This is also derived from the oscillations of the position set point. Despite that, the response of the controller remains stable for most of the experiments.

The execution time had to achieve a maximum of one millisecond for every DoF, and for this reason, the PS-Arduino approach was ruled out to implement the full hand. The PSPL-ArtyLnx method applies the hybrid system using software for interfacing and communication. Programmable logic enables the robotic hand's control in real-time while achieving an execution time much higher than the required (25.64 kHz). It is essential to highlight that even though the achieved frequencies are not required for the current system, this architecture opens a path for other platforms with faster dynamics.

Finally, this project's prototype could grasp objects with different geometries with power and precision grip, highlighting the robotic hand's dexterity and flexibility. Also, The results of the experiment in Fig. 15(f)-(h) showed that for low

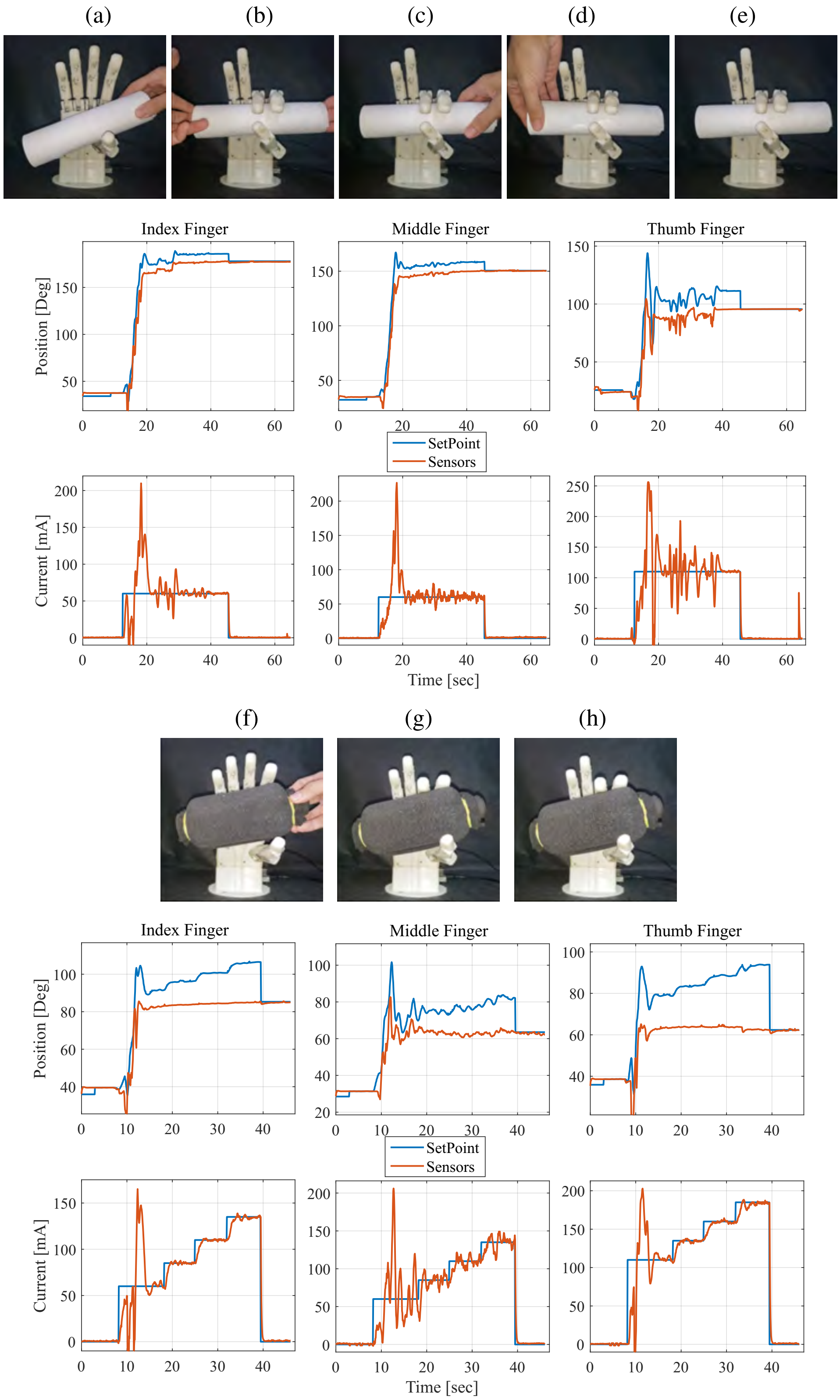


FIGURE 15. Grasping tests with interference. (a)-(e) is the first experiment and (f)-(h) is the second. Bellow the experiments are the curves for current and position for the index, middle and thumb fingers. The pictures are in chronological order. See video

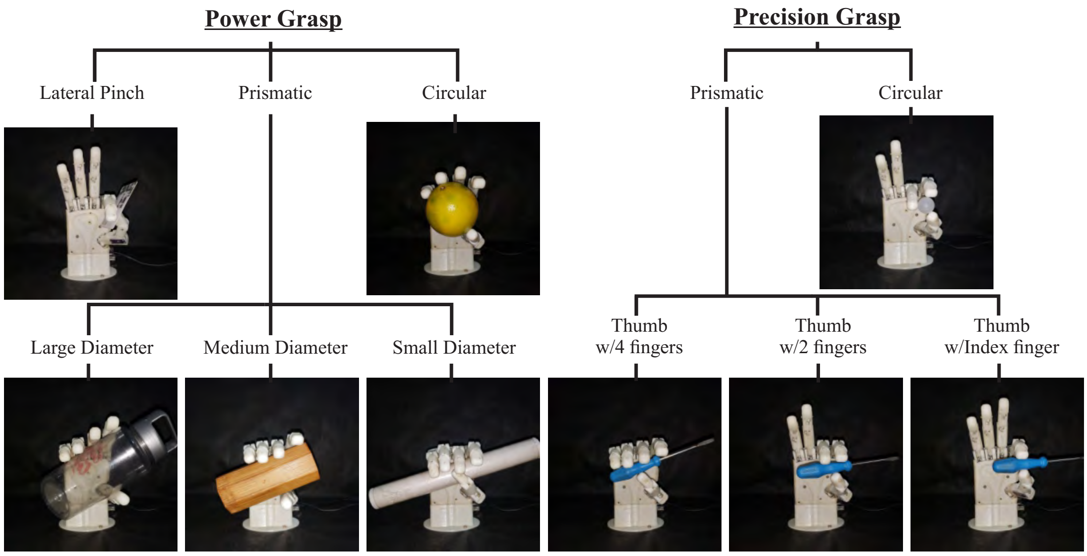


FIGURE 16. Robotic hand performing grasping poses according to the Cutkosky taxonomy. See video

force grasping it is still necessary a complementary strategy or data to the impedance controller, such as tactile sensing. One could be a combined impedance/position control that performs a position control until contact with an object is reached, switching to the impedance control. These are objectives for future works.

REFERENCES

[1] R. Vepa, "Biomimetic Robotics," in Engineered Biomimicry, Newnes, Ed. Elsevier, 2013, pp. 81–105.

[2] D.-H. Lee, J.-H. Park, S.-W. Park, M.-H. Baeg, and J. Bae, "KITECH-Hand: A Highly Dexterous and Modularized Robotic Hand," IEEE/ASME Transactions on Mechatronics, vol. 22, no. 2, pp. 876–887, apr 2017.

[3] C. A. Calderon, C. Ramirez, V. Barros, and G. Punin, "Design and Deployment of Grasp Control System applied to robotic hand prosthesis," IEEE Latin America Transactions, vol. 15, no. 2, pp. 181–188, feb 2017.

[4] S. H. Jeong, K.-S. Kim, and S. Kim, "Designing Anthropomorphic Robot Hand with Active Dual-Mode Twisted String Actuation Mechanism and Tiny Tension Sensors," IEEE Robotics and Automation Letters, vol. 2, no. 3, pp. 1–1, 2017.

[5] X. Wang, Y. Liu, D. Yang, N. Li, L. Jiang, and H. Liu, "Progress in the biomechatronic design and control of a hand prosthesis," in 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, oct 2010, pp. 5880–5885.

[6] N. Daoud, J. Gazeau, S. Zeghloul, and M. Arsicault, "A real-time strategy for dexterous manipulation: Fingertips motion planning, force sensing and grasp stability," Robotics and Autonomous Systems, vol. 60, no. 3, pp. 377–386, mar 2012.

[7] J. Gazeau, S. Zehloul, M. Arsicault, and J. Lallemand, "The LMS hand: force and position controls in the aim of the fine manipulation of objects," in Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164), vol. 3. IEEE, 2001, pp. 2642–2648.

[8] S. Lee, S. Noh, Y. Lee, and J. H. Park, "Development of bio-mimetic robot hand using parallel mechanisms," in 2009 IEEE International Conference on Robotics and Biomimetics (ROBIO). IEEE, dec 2009, pp. 550–555.

[9] D. Zhao, L. Jiang, H. Huang, M. Jin, H. Cai, and H. Liu, "Development of a Multi-DOF Anthropomorphic Prosthetic Hand," in 2006 IEEE International Conference on Robotics and Biomimetics. IEEE, 2006, pp. 878–883.

[10] H. Huang, Li Jiang, Y. Liu, L. Hou, H. Cai, and H. Liu, "The Mechanical Design and Experiments of HIT/DLR Prosthetic Hand," in 2006 IEEE International Conference on Robotics and Biomimetics, no. 50435040. IEEE, 2006, pp. 896–901.

[11] H. Liu, K. Wu, P. Meusel, N. Seitz, G. Hirzinger, M. Jin, Y. Liu, S. Fan, T. Lan, and Z. Chen, "Multisensory five-finger dexterous hand: The DLR/HIT Hand II," in 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, sep 2008, pp. 3692–3697.

[12] T. Lan, Y. Liu, M. Jin, S. Fan, H. Fang, J. Xia, and H. Liu, "DSP&FPGA-based joint impedance controller for DLR/HIT II dexterous robot hand," in 2009 IEEE/ASME International Conference on Advanced Intelligent Mechatronics. IEEE, jul 2009.

[13] J.-U. Chu, D.-H. Jung, and Y.-J. Lee, "Design and control of a multi-function myoelectric hand with new adaptive grasping and self-locking mechanisms," in 2008 IEEE International Conference on Robotics and Automation. IEEE, may 2008, pp. 743–748.

[14] D. H. Jeong, J. U. Chu, and Y. J. Lee, "Development of myoelectric hand with infrared led-based tactile sensor," Journal of Institute of Control, Robotics and Systems, vol. 15, no. 8, pp. 831–838, aug 2009.

[15] C. Piazza, G. Grioli, M. G. Catalano, and A. Bicchi, "A Century of Robotic Hands," Annual Review of Control, Robotics, and Autonomous Systems, vol. 2, pp. 1–32, 2019.

[16] E. Mattar, "A survey of bio-inspired robotics hands implementation: New directions in dexterous manipulation," Robotics and Autonomous Systems, vol. 61, no. 5, pp. 517–544, 2013.

[17] Z. Kappassov, J.-A. Corrales, and V. Perdereau, "Tactile sensing in dexterous robot hands âĂŤ Review," Robotics and Autonomous Systems, vol. 74, pp. 195–220, dec 2015.

[18] A. Hidalgo-l, R. Navas-gonz, J. Herr, F. Vidal-verd, and S. Antonio, "FPGA-Based Tactile Sensor Suite Electronics for Real-Time Embedded Processing," IEEE Transactions on Industrial Electronics, vol. 64, no. 12, pp. 9657–9665, 2017.

[19] S. A. Pertuz, C. Llanos, C. Peña, and D. Muñoz, "A parallel system-on-chip approach for impedance controller for a 7-DoF robotic hand," Analog Integrated Circuits and Signal Processing, apr 2020.

[20] S. A. Pertuz, C. H. Llanos, and D. M. Munoz, "Bioinspired Optimization of a Robotic Finger Mechanism," in 2016 XIII Latin American Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR). IEEE, oct 2016, pp. 199–204.

[21] H. van Duinen and S. C. Gandevia, "Constraints for control of the human hand," The Journal of Physiology, vol. 589, no. 23, pp. 5583–5593, dec 2011.

[22] S. R. Company, "Shadow Dexterous Hand Technical Specification," Shadow Robot Company, Tech. Rep. August, 2015. [Online]. Available: http://www.shadowrobot.com/products/dexterous-hand/

[23] A. Faudzi, J. Ooga, T. Goto, M. Takeichi, and K. Suzumori, "Index Finger

of a Human-like Robotic Hand using Thin Soft Muscles," IEEE Robotics and Automation Letters, vol. 3, no. 1, pp. 1–1, 2017.

[24] M. Tavakoli and A. T. de Almeida, "Adaptive under-actuated anthropomorphic hand: ISR-SoftHand," in 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, no. Iros. IEEE, sep 2014, pp. 1629–1634.

[25] S. Cobos, M. Ferre, and R. Aracil, "Simplified human hand models based on grasping analysis," in 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, Oct 2010, pp. 610–615.

[26] O. Binvignat, A. Almagià, P. Lizana, and E. Olave, "Aspectos Biométricos de la Mano de Individuos Chilenos," International Journal of Morphology, vol. 30, no. 2, pp. 599–606, jun 2012.

[27] K. Waldron and J. Schimiedeler, "Kinematics," in Springer Handbook of Robotics, B. Siciliano and O. Khatib, Eds. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2008, ch. 1, pp. 9–33.

[28] H. Tizhoosh, "Opposition-Based Learning: A New Scheme for Machine Intelligence," in CIMCA-IAWTIC'06, vol. 1. IEEE, 2006, pp. 695–701.

[29] R. Storn and K. Price, "Differential evolution âĂŞ a simple and efficient heuristic for global optimization over continuous spaces," Journal of Global Optimization, vol. 11, no. 4, pp. 341–359, 1997.

[30] D. Whitley, "A genetic algorithm tutorial," Statistics and Computing, vol. 4, no. 2, jun 1994.

[31] A. Kapandji and M. Torres, Fisiología articular: esquemas comentados de mecánica articular. Hombro, codo, pronosupinacion, muñeca, mano. 1, 6th ed., ser. Fisiología articular. Tomo 1. Hombro, codo, pronosupinación, muñeca, mano, M. Torres, Ed. Editorial Médica Panamericana, 2006.

[32] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-Driven Grasp Synthesis - A Survey," IEEE Transactions on Robotics, vol. 30, no. 2, pp. 289–309, apr 2014.

[33] M. Regoli, U. Pattacini, G. Metta, and L. Natale, "Hierarchical grasp controller using tactile feedback," in 2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids), Nov 2016, pp. 387–394.

[34] S. B. Godfrey, M. Bianchi, A. Bicchi, and M. Santello, "Influence of force feedback on grasp force modulation in prosthetic applications: A preliminary study," in 2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Aug 2016, pp. 5439–5442.

[35] N. Hogan, "Impedance Control: An Approach to Manipulation: Part IIâĂŤImplementation," Journal of Dynamic Systems, Measurement, and Control, vol. 107, no. 1, p. 8, 1985.

[36] ——, "Impedance Control: An Approach to Manipulation: Part IâĂŤTheory," Journal of Dynamic Systems, Measurement, and Control, vol. 107, no. 1, p. 1, 1985.

[37] W. He, W. Ge, Y. Li, Y. Liu, C. Yang, and C. Sun, "Model identification and control design for a humanoid robot," IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 47, no. 1, pp. 45–57, 2017.

[38] J. Sergey, S. Sergei, and Y. Andrey, "Comparative analysis of global optimization-based controller tuning methods for an exoskeleton performing push recovery," in 2016 20th International Conference on System Theory, Control and Computing (ICSTCC), 2016, pp. 107–112.

[39] F. Caccavale, C. Natale, B. Siciliano, and L. Villani, "Integration for the next generation: embedding force control into industrial robots," IEEE Robotics & Automation Magazine, vol. 12, no. 3, pp. 53–64, sep 2005.

[40] C. H. Llanos, D. Munoz, and S. A. Pertuz Mendez, "Simulation and Implementation of Impedance Control in Robotic Hand," in Procceedings of the 24th ABCM International Congress of Mechanical Engineering. ABCM, 2017.

[41] S. Dehghani, H. D. Taghirad, and M. Darainy, "Self-tuning dynamic impedance control for human arm motion," in 2010 17th Iranian Conference of Biomedical Engineering (ICBME), 2010, pp. 1–5.

[42] B. Maldonado, M. Mendoza, I. Bonilla, and I. Reyna-GutiÃľrrez, "Stiffness-based tuning of an adaptive impedance controller for robot-assisted rehabilitation of upper limbs," in 2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), 2015, pp. 3578–3581.

[43] P. Balatti, D. Kanoulas, G. F. Rigano, L. Muratore, N. G. Tsagarakis, and A. Ajoudani, "A self-tuning impedance controller for autonomous robotic manipulation," in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 5885–5891.

[44] Y. Raza, S. F. Ahmed, A. Ali, M. K. Joyo, and K. A. Kadir, "Optimization of pid using pso for upper limb rehabilitation robot," in 2018 IEEE 5th International Conference on Engineering Technologies and Applied Sciences (ICETAS), 2018, pp. 1–4.

[45] J. Campos, S. Jaramillo, L. Morales, O. Camacho, D. ChÃąvez, and D. Pozo, "Pso tuning for fuzzy pd + i controller applied to a mobile robot trajectory control," in 2018 International Conference on Information Systems and Computer Science (INCISCOS), 2018, pp. 62–68.

[46] E. Mendez-Flores, E. M. Martinez-Galicia, J. d. J. Lozoya-Santos, R. Ramirez-Mendoza, R. Morales-Menendez, I. Macias-Hidalgo, A. Vargas-Martinez, and A. Molina-Gutierrez, "Self-balancing robot control optimization using pso," in 2020 5th International Conference on Control and Robotics Engineering (ICCRE), 2020, pp. 7–10.

[47] H. Wang, Z. Wu, S. Rahnamayan, Y. Liu, and M. Ventresca, "Enhancing particle swarm optimization using generalized opposition-based learning," Information Sciences, vol. 181, no. 20, pp. 4699–4714, oct 2011.

[48] G. Einicke, Smoothing, Filtering and Prediction - Estimating The Past, Present and Future, G. A., Ed. InTech, feb 2012.

[49] C. H. L. Daniel M. Muñoz, Diego F. Sanchez and M. AyalaRincon, "Trade-off of FPGA design of a floating-point library for arithmetic operators," Journal of Integrated Circuits and Systems, vol. 5, no. 1, pp. 42–52, 2010.

[50] S. A. Pertuz, C. Llanos, C. A. Pena, and D. Munoz, "A Modular and Distributed Impedance Control Architecture on a Chip for a Robotic Hand," in 2018 31st Symposium on Integrated Circuits and Systems Design (SBCCI). IEEE, aug 2018, pp. 1–6.

[51] M. Cutkosky, "On grasp choice, grasp models, and the design of hands for manufacturing tasks," IEEE Transactions on Robotics and Automation, vol. 5, no. 3, pp. 269–279, jun 1989.

SERGIO PERTUZ MENDEZ has MSc. in Mechatronics Systems at the University of Brasilia (UnB), also has a B.Sc. in Mechatronics Engineering at the University of Pamplona (UPA). He has experience in Systems on Chip (SoC), microchip programming and FPGA, mechanical and electrical designs, and process automation.

PROF.DR. CARLOS H. LLANOS received his B.s. degree in Electrical Engineering from the University of Valle, Cali, Colombia, in 1983; M.Sc. degree in Computer Science from the University of Minas Gerais - UFMG, Minas Gerais, Brazil, in 1990; and a Ph.D. degree in Electrical Engineering from the University of Sao Paulo, Brazil, in 1998. He lectures in the graduate program in Mechatronics Systems in the Department of Mechanical Engineering at the University of Brasilia - UnB, Brasilia, Brazil, since 2003. He is a member of the IEEE. His current research interests include reconfigurable systems for automation, intelligent systems, instrumentation, and embedded systems design.

DANIEL M. MUÑOZ received a BS degree in Physics Engineering from Universidad del Cauca, Colombia, and the MS.c and Ph.D. degrees in Mechatronics Systems from University of BrasÃ■lia, Brazil, in 2003, 2006 and 2012, respectively. Since 2012, he has been at the University of Brasilia. He is currently an assistant professor at the Electronics undergraduate program, Faculty of Gama, and at the Mechatronics graduate program, Department of Mechanical Engineering. His current research includes digital circuit design using FPGAs, artificial intelligence, and bio-inspired optimization algorithms.

Efficient Data-driven Real-time Magnetic Tracking for Myokinetic Control Interfaces

Sergio A. Pertuz, Marta Gherardini, Gabriel Vidigal de Paula Santos, Daniel Muñoz, Helon Vicente Hultmann Ayala, Christian Cipriani *Senior Member, IEEE*

***Abstract*— This work presents an embedded machine learning solutions for the control of prosthetic limbs based on magnetic tracking. Namely, we employ a data-driven strategy to create mathematical models, which can translate measured magnetic information from implanted magnets to desired commands for active prosthetic devices. Initially, the prediction uses an optimization algorithm to solve an extensive equation system. Instead, we employed machine learning models such as linear and Radial Basis Functions Artificial Neural Networks (RBFNN) due to their inherently parallel architecture. They were developed offline and then implemented on field-programmable gate arrays using customized floating-point operators, optimizing computational precision, hardware, and energy consumption, which are essential features in the context of wearables devices. When used to track a single magnet in an anatomical mockup of the human forearm, the proposed data-driven strategy implemented in hardware could achieve a tracking accuracy of 720 μm 95% of the time and latency of a 12.07 μs. Also, the proposed system architecture requires a lower energy consumption than previous solutions reported in the literature. This work's outcomes encourage further research on improving the devised methods to deal with a more significant number of magnets simultaneously.**

***Index Terms*— myokinetic control interface; prosthetic control; artificial neural networks; machine learning; FPGA**

I. Introduction

UPPER limb amputation deprives individuals of their innate ability to manipulate objects. In order to restore these dexterous motor functions after amputation, prosthetic devices and control strategies have been one of the primary goals in rehabilitation engineering. However, the quest for a human-machine interface (HMI) that allows for arbitrary and physiologically appropriate control over multiple degrees of freedom is still far from being completed. Besides commercial solutions that exploit superficial electromyography (EMG) signals to control the prosthesis [1], researchers are investigating alternative approaches that take advantage of different biological sources. For instance, solutions have been proposed which exploit implanted electrodes (such as intramuscular [2] and epymisial electrodes [3]) to record muscle activity, as well as neural electrodes to record peripheral information [4]. An alternative solution recently introduced in [5] exploiting magnet tracking for controlling a prosthesis. The authors proposed a new HMI dubbed the myokinetic control interface. This interface derives information about muscle contractions from permanent magnets implanted into the amputee's forearm muscles. Indeed, localizing the magnets' position is equivalent to measuring the contraction/elongation of the muscle. This information can be used to interpret the voluntary movement of the subject. In [5], magnets were analytically modeled as point dipoles, and localization was obtained through the Levenberg–Marquardt optimization algorithm. It finds an optimal solution to the inverse problem of magnetostatics.

A vital research venue is to create efficient algorithms to estimate the magnets' position in this context. The faster the estimation algorithm can provide an output, the more fine-grained control can be achieved for the prosthetic device. The design of a position transducer based on the magnetic sensor information should consider the compromise of precision, accuracy, and execution time. With more time-dense inputs, the control algorithm will have more fine-grained information to perform trajectory tracking, making it easier to represent human-like behavior. In this view, the latency of the algorithm (i.e., the time needed for localizing the magnets once the measurements are available) has to be short enough to ensure real-time tracking. In [6], the authors showed that the latency of algorithms that exploit an analytic representation of the magnets could be reduced by computing the analytic gradient of such representation. In [5], the tracking algorithm could not provide estimations as fast as the sensor could provide information. More specifically, while sensors could provide $\sim$75 samples per second (one sample every 13 ms), 45 ms were needed for localizing four magnets. In a more recent study [7], a fully embedded system was presented, which proved capable of tracking up to five magnets in less than $\sim$4 ms using 32 magnetic field sensors. In this case, the time

This research was supported by the European Research Council under the MYKI project (ERC-2015-StG, Grant no. 679820). I was also partially supported by CNPq (430395/2018-3-Univ, 400119/2019-6-ERC), the Coordination for the Improvement of Higher Education Personnel of Brazil - CAPES (88882.383336/2019-01) and the Foundation for the Support for Research - FAP-DF (28552.100.43376.14112019) (Corresponding author: Sergio A. Pertuz).

Sergio A. Pertuz and Daniel Muñoz are with the Department of Mechanical Engineering, University of Brasilia, Brasilia 70910-900, Brazil (e-mails: sergio.pertuz, damuz@unb.br).

Marta Gherardini and Christian Cipriani are with the BioRobotics Institute, Scuola Superiore Sant'Anna, Pontedera 56025, Italy (e-mails: m.gherardini, christian.cipriani@santannapisa.it).

Gabriel V. P. Santos and Helon V. H. Ayala are with the Department of Mechanical Engineering, Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro 22453-900, Brazil. (e-mails: gabriel@spsantos.com.br, helon@puc-rio.br).

needed for sampling the sensors readings (one sample every ∼21 ms) and sending them to the computation unit (∼24 ms) represented the actual bottleneck of the system. Thus, the system architecture proposed in [7] could be optimized in terms of acquisition and data-transfer times to boost the system performance.

This work aims to evaluate data-driven solutions based on embedded machine learning algorithms, an alternative to that in [5], [7]. Linear models and artificial neural networks (ANNs) were used to approximate mathematical abstractions from input-output tuples [8]. As a result of its universal approximation property, ANNs can reproduce the behavior of any continuous function [9]. The inherent parallelism and modular structure of ANNs [10], [11] can be tuned in order to achieve a better compromise between accuracy, precision, computational complexity, and energy consumption. Similarly, their computational implementation on embedded hardware, such as Field-programmable gate arrays (FPGAs), is simplified for the nature of its structure, favoring simpler and more efficient solutions adjusting the execution time according to the task [12].

Machine learning models have been fruitfully applied to various problems involving sensing, control, and estimation in biomedical applications, including interpreting biological signals to generate actionable commands. In this context, cases in which real-time response is required and solved by hardware implementations often involve such data-driven methods. In the scope of pneumatic muscles, ANNs have been implemented to dynamic modeling in [13] while in [14] a nonlinear proportional-integral-derivative ANN controller has been devised. ANNs have also been used for controlling wearable exoskeletons, in [15] to improve the torque estimation required by the apparatus and in [16] to perform adaptive control concerning parametric uncertainties and unknown disturbances, and in [17] for controlling grasp and lift of a cable-driven soft hand. In the case of prosthetic hands, in [18], the authors investigate the use of convolutional ANNs embedded in microcontrollers to classify hand gestures by interpreting EMG signals. In [19], the authors employ a dataset aggregation strategy for creating deep ANNs for processing EMG data to perform prosthetic limb control. As can be seen from this recent literature review, machine learning has shown great applicability for creating models that interpret complex biosignals. Moreover, such mathematical abstractions favor efficient hardware implementations due to their structure, composed of simple entities, such as neurons or support vectors.

A recent review on the topic of ANNs embedded solutions on FPGAs and their advantages to other heterogeneous computing platforms is given in [20]. Recently, FPGA solutions of deep artificial neural networks have also been studied for both training and inference stages. In [21], the authors report the most important features and advantages of hardware implementations for deep ANNs, namely lower power consumption and inherent reconfigurability. In [22], several activation functions are mapped and improve software, and GPU-based deployments, similar to [23], mainly due to its parallel dataflow characteristic. Hardware implementation of ANN for sensor-driven position estimation, such as the proposal of this work, has been previously used in [24], [25]. An interesting type of ANN is the Radial Basis Function Neural Network (RBFNN), which has an inherently parallel architecture and is usually implemented using Gaussian activation functions. Most recent works that focus on implementing RBFNNs on hardware can be found in [12], [26] and references therein [27]–[31]. In [12] new architectures for RBFNN were proposed with customized floating-point precision enabling hardware and energy consumption optimization. RBFNNs are relatively easy to design and training, besides dealing with linear and nonlinear problems relatively well. Furthermore, it has a strong tolerance to input noise, and even when the problem's complexity is significant [32].

In this context, the present work aims to evaluate new hardware architectures for machine learning models for faster and energy-efficient solutions in the myokinetic control interface. Thus, this work's contributions are the following: (i) To empirically prove that it is possible to build data-driven machine learning models that can translate the magnetic information to useful commands for driving prosthetic devices. We confirmed with real-world measured data made in a test bench that it is possible to map the magnetic information to muscle deflection using machine learning models. (ii) Create tools that automatically map the proposed machine learning models on reconfigurable hardware, allowing other applications (*pLinRgen* and *vRBFgen*). They enable customization of the models' architecture according to the number of parallel operators and the bit-width representation. In this way, the latency, throughput, and computational power can be precisely adjusted according to the application requirements, which is important in the present application as the deployment of each prosthesis is ideally personalized and thus involves reconfiguration that can be optimized using the tools mentioned above. They permit the myokinetic interface to explore the full sampling rate of the magnetic field sensors. Current techniques are based on complex models such as Levenberg–Marquardt optimization algorithm, which provides a solution to the inverse problem of magnetostatics [5]. Although this approach provides accurate information, real-time implementation is a challenge using embedded solutions with limited resources. The method and tools gave herein allow efficiently real-time magnet tracking considering energy consumption and performance. The RBFNN model achieved a localization error below 720 μm 95% of the time and a 12.07 μs latency. In contrast, the linear model achieved 520 μm with 11.52 μs. However, even though the linear model has better accuracy than RBFNN, the latter has better precision (a more relevant characteristic for this case), as proved in the Discussions section. Additionally, energy consumption was reduced to less than 57% compared to the previous implementation, a major accomplishment. (iii) The I2C communication of the magnetic sensors is a slow protocol given the number of sensors and their sample rate. The use of several data lines with the board allows to fully exploit these models' computational performance, which is plausible in an FPGA.

The manuscript is organized as follows. Section II explores the test bench and its features, the soft sensing strategy based on machine learning for solving the transduction problem, and the protocols for data acquisition. Section III depicts the machine learning hardware implementation architectures, while the results for its real-time implementation are given in Section IV. Lastly, discussions and future research activities are given respectively in Section V and VI.

II. Experimental Setup and Data Acquisition

A physical mockup resembling the human forearm was used to acquire the data for training and testing the machine learning models. In such a system, muscles were modeled with wires attached to servo motors, pulleys, and weights. In this setup, the servo motor rotation corresponds to a translation (contraction/elongation) of the muscle (wire), reproducing their axial deformation. The implants are commercially available NdFeB cylindrical magnets (D = 4 mm, L = 2 mm, residual magnetic flux density Br = 1.27 T) and are attached to the wire.

The magnets' magnetic field is sampled through 128 three-axis magnetic field sensors disposed on four custom PCBs (32 sensors per board), as depicted in Fig. 1. Each PCB is laid on one of the four orthogonal planes along the arm, and the sensors are aligned in grids of eight columns and four rows separated by a 9 mm gap. The four PCBs were spatially centered on four opposite sides of a parallelepiped structure enclosing the workspace (100 mm × 54 mm × 100 mm) (two on the XZ plane, two on the XY plane – Fig. 1). In the mockup, the data is streamed to a PC (Windows 7, Intel i7-6700 CPU running at 3.4 GHz, 32 GB of RAM) the readouts from the sensors at a rate of 20 Hz, through a serial bus (RS-232). The collected signals are stored, so they can later be used for offline processing. In total, there are 384 time-series recorded that carry information about the magnetic field (three-axis times 32 sensors times four boards).

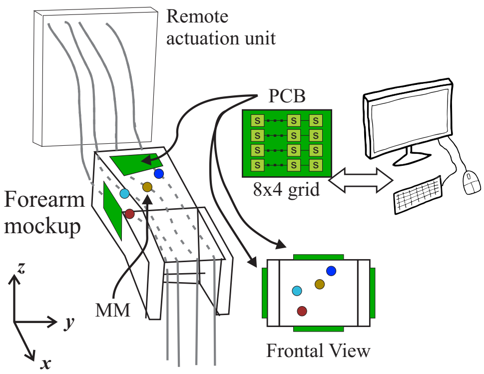


Fig. 1. Mockup Schematic. It emulates the movements of the hand's 17 degrees of freedom, for a total of 17 wires, although only one is currently being used. Muscles were modeled using a wire attached on one side to a servo motor (housed in a remote actuation unit) and on the other side to counterweight to maintain tension. Adapted from [5].

For this work, the displacement of a single magnet is adequately driven to achieve a maximum translation of the wire (i.e., of the muscle) of ∼10 mm. A total of six datasets are acquired, giving a new input to the servo at a $t_s = 50$ ms sample-rate. The input displacement (target) provided to the servo was used as groundtruth for analyzing the accuracy of the retrieved magnet displacement. In particular, the following datasets were used as target:

- Multisine, for creating the model. It is composed by a sum of sinusoid signals as

$$y_k = \frac{A}{2M}\left\{\sum_{i=1}^{M}\left[\sin\left(2\pi f_i t_s k + \phi_i\right)\right] + 1\right\}, \quad (1)$$

  where y_k denotes the servo command used as output for the model at discrete time $k = 0, 1, ..., N$, A is the desired amplitude for y_k in the range $[0, A]$, M is the total number of sinusoids in the equally spaced range for $f_1, f_2, \ldots, f_M$, and $\phi_1, \phi_2, \ldots, \phi_M$ are uniformly random phases in the range $[0, 90°]$. The multisine is able to excite the system in an specific range and also is able to reveal if the dynamic relationship between the input and the output of the system should be taken into account. For the acquisitions, we used $A = 10$ mm, as it is the range of operation for the mockup, $M = 10$ sine waves, and frequencies $f_1, ..., f_M$ equally distributed in a range between 0.1 and 0.5 Hz (see Fig. 2).
- Ramp and sequence of steps, for testing the models. Ideally, a data-driven model should be tested with a different dataset than used to train it. To this end, we used ramp and step signals datasets that are relevant for the application of trajectory tracking using the myokinetic interface. The ramps are configured with positive and negative slopes with different speeds during 20, 40, 60, and 80 seconds. The steps were equally divided into twenty levels, ranging from the minimum (0 mm) to the maximum (10 mm) servo displacement, and were applied sequentially. The sequence of ramps and steps is given in Fig. 3.

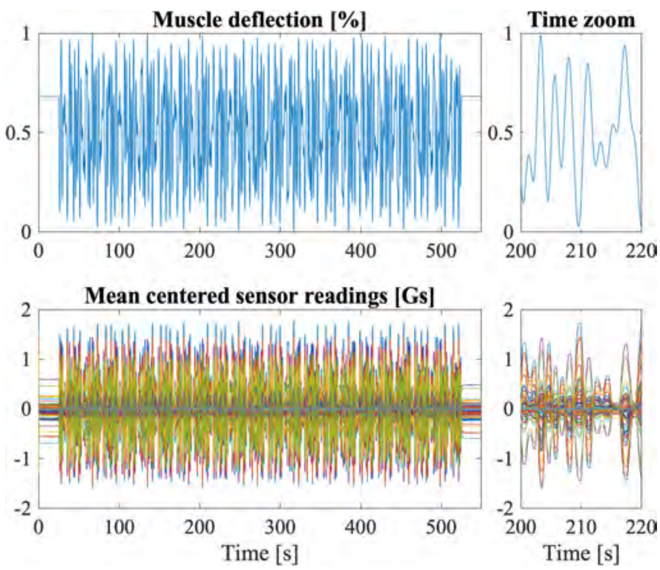


Fig. 2. Measured dataset with the multisine excitation signal (left) and zoomed around 20 seconds (right). The sensors readings are the magnetic fields in Gauss.

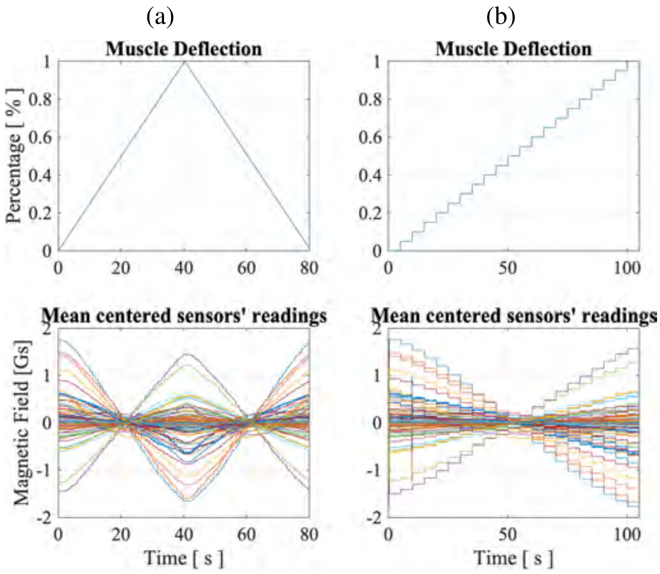


Fig. 3. Measured dataset for (a) the ramp excitation signal with 80 seconds total duration and (b) the step excitation signal.

From the plots in Fig. 2–3 we can see that there is a cause-effect relationship between the measured magnetic field and the output displacement. This observation is confirmed in Fig. 4, which plots the normalized histogram of the Pearson cross-correlation for the magnetic sensors data and the output displacement, ordered by groups of the amplitude of R in ascending order. It is possible to see that many sensors are linearly correlated with the output muscle deflection (positively and negatively), which indicates that machine learning models can represent the measured displacement based on the magnetic field data. Thus, they can then be implemented on hardware for fast and energy-efficient solutions, as we devise in the next section. The challenge was to design offline a model that can be embedded as a general architecture that delivers better timing results than those available in the literature.

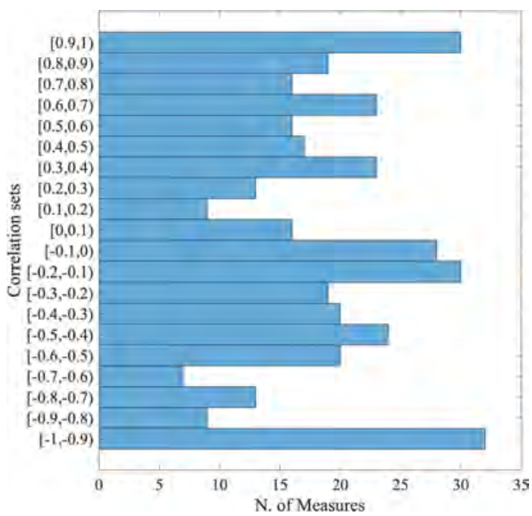


Fig. 4. Distribution of the cross-correlation among the magnetic sensors time series and the muscle's output displacement. Note that many measurements are directly or inversely linearly correlated with the output deflection.

III. Machine Learning Models Implementation on Hardware

A. Mathematical Formulation

In the present paper we employ both the linear and RBFNN models. The linear model is simply described by a linear combination of the inputs of the model, that is,

$$\hat{y}_k^L = c_1 \cdot x_{1,k} + c_2 \cdot x_{2,k} + \cdots + c_{n_{var}} \cdot x_{n_{var},k} + b \qquad (2)$$

where $c_i \in \mathbb{R}, i = 1, 2, \ldots, n_{var}$ and $b \in \mathbb{R}$ are the free coefficients of the model with n_{var} representing the number of inputs, $\hat{y}_k^L$ and $x_{i,k} \in \mathbb{R}, i = 1, 2, \ldots, n_{var}$ are the output and the input variables of the linear model in the instant k. The linear model is the weighted combination of the model inputs. On the n the other hand, the RBFNN is given by

$$\hat{y}_k^R = F[\boldsymbol{x}_k] = \sum_{m=1}^{M} w_m \phi(\boldsymbol{x}_k, \boldsymbol{c}_m, \sigma_m), \qquad (3)$$

where $\hat{y}_k^R \in \mathbb{R}$ and $\boldsymbol{x}_k \in \mathbb{R}^{n_{var}}$ are respectively the network predicted output and the input vector at a given instant k, $M \in \mathbb{N}^+$ is the number of neurons in the hidden layer, the output weights are denoted by $w_m \in \mathbb{R}$, $\boldsymbol{c}_m \in \mathbb{R}^{n_{var}}$ and $\sigma_m \in \mathbb{R}^+$ are respectively the center and the width of the m-th hidden node of the RBFNN. The function $\phi : \mathbb{R}^{n_{var}} \times \mathbb{R}^{n_{var}} \times \mathbb{R} \mapsto \mathbb{R}$ is the radial basis activation function, which is frequently set as the Gaussian with

$$\phi(\boldsymbol{x}(k), \boldsymbol{c}_m, \sigma_m) = \exp\left[-\frac{1}{2\sigma_m^2}\sum_{i=1}^{n_{var}}(r_i(k) - c_{m,i})^2\right]. \qquad (4)$$

The linear model was trained using the linear least-squares regression method to determine its intersection and coefficients [33]. On the other hand, the RBFNN was trained using a simple yet effective 2-stage procedure [11], meaning that the centers are defined according to unsupervised learning, and the weights are obtained by any least-squares method as they are linear in the parameters. We employed k-means clustering algorithm for the centers, and the output weights were defined by QR factorization. We defined the number of neurons with trial and error, aiming at better validation metrics.

B. Validation Metrics

In this work, the validation metrics used to provide a quantitative measure of agreement between the predictive models and the expected observations are the Mean Squared Error in decibels (MSE [db]) and the coefficient of determination (R^2). The former is calculated by

$$\text{MSE [db]} = 10\log_{10}\left(\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2\right) \qquad (5)$$

and the latter with

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \overline{y})^2} \qquad (6)$$

where y is the reference dataset, $\hat{y}$ is the models' estimation result, $\overline{y}$ is the mean of y, and N is the size of the datasets.

The MSE [db] is used as a prediction fidelity measure whose goal is to provide a quantitative score describing the degree of similarity/fidelity or, conversely, between the real and the estimated magnet displacement. The MSE is selected because it is the natural way to define the error signal's energy and is an excellent metric in the context of optimization [34]. On the other hand, the R^2 is a widely used metric for curve fitting that provides a measure of how well the expected outcomes are replicated by the models, based on the proportion of total variation of outcomes explained by the model [35].

The mean absolute error deviation is also used to compare the final prediction of the models. It is calculated by contrasting the prediction to the expected value with

$$s = \frac{1}{N}\sqrt{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{7}$$

C. Architectures for the Linear and RBFNN Models

Both hardware architectures were automatically generated using our VHDL code generator tools (*pLinRgen* and *vRBFgen*) developed in MATLAB to easily produce different models. The tools receive configuration parameters that set the model's architecture composition, characterizing the number of arithmetical operators and the connections in between. The arithmetical operators are customized floating-point modules previously developed in [36], [37]. The hardware implementation models' description and the tools' configuration parameters are detailed in the following subsections.

1) Linear Model - pLinRgen: The linear model predictor architecture generator was implemented in FPGA as a combination of FSM (Finite State Machine) and pipeline architectures. It calculates equation 2, and the architecture can be customized using the following configuration parameters:

1) $nbits$: Is the bit-width of the floating-point representation.
2) n_{var}: is the number of variables of the linear model.
3) $c_1, \cdots, c_{n_{var}}$ and b: see Eq. 2
4) n_{op}: Is the number of operators executed in parallel. The number of stages of the pipeline architecture and the arrangement of the operators depends on this parameter (see Fig 5(b)).

The input/output ports of the linear model predictor are depicted in Fig. 5(a). As expected, the quantity of inputs depends on the n_{op} parameter. Figure 5(b) illustrates a possible configuration for the linear model predictor with $n_{op} = 4$ and 4 levels of deepness (n_{stages}), which is determined by

$$n_{stages} = \log_2(n_{op}) + 2. \tag{8}$$

The "Data FSM" block is an FSM that synchronously feeds the variables and coefficients to the "Linear Model" block. When $n_{var} > n_{op}$ the FSM feeds the data in the correct sequence for $n_{cycles} = \lceil n_{var}/n_{op} \rceil$, being the number of cycles necessary to carry out a prediction. This sequence is illustrated in Fig. 6 as a timing diagram with the states of the values for the inputs and outputs of the "Linear Model" block.

The latency in the diagram is depicted as the measured time between the first start pulse ("ready_in") and the first output ready pulse ("ready_out") of the system. It is calculated by

$$L = n_{cycles} \cdot n_{stages} \cdot t_{op}, \tag{9}$$

where T_{clock} is the global clock period, t_{op} is the time the operator modules require to carry out a result and is expressed as $2 \cdot T_{clock}$, and the latency is measured in μseconds.

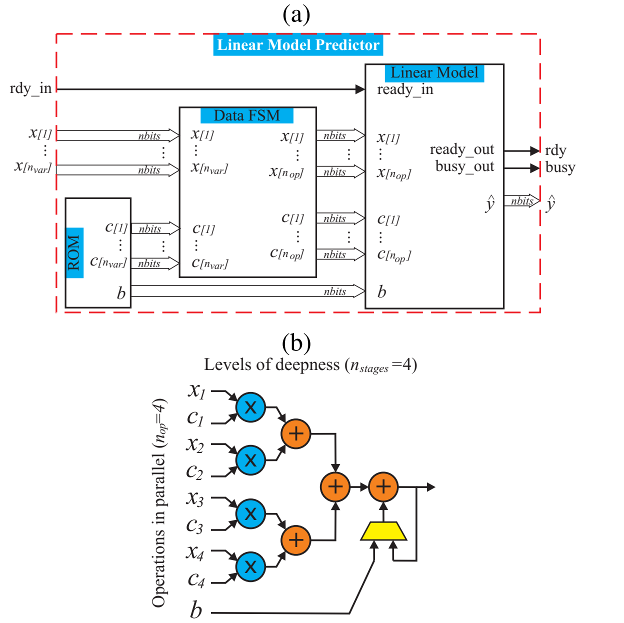


Fig. 5. Linear Model Predictor on FPGA. (a) Top level view of system. (b) Internal pipe-lined architecture of "Linear Model" block.

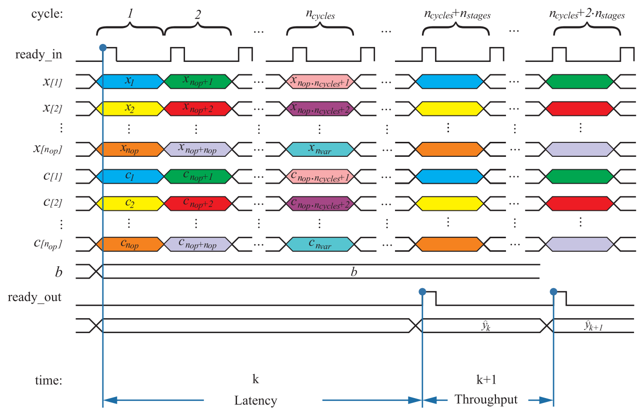


Fig. 6. Input/Output data time diagram of linear model in FPGA. The "ready_in" input indicates the start of the process n_{cycles}-times. The "ready_out" output announces when a prediction is calculated.

Similarly, the throughput is the output data rate and can be measured in MFLOPS (Million of Floating Operations per Second). It is estimated as the inverse time between two consecutive output ready pulses ("ready_out"). However, because it combines FSM and a pipelined architecture, the traditional estimation of the throughput for a pipeline ($\frac{1}{2 \cdot T_{clock}}$)

does not work. For this case, the throughput (T) also depends on the n_{cycles} and n_{stages} variables and is estimated by

$$T = \frac{1}{(n_{cycles} \cdot n_{stages} - n_{stages} + 1) \cdot t_{op}}. \quad (10)$$

As seen, these timing characteristics of the architecture depend on the input parameters of the code generators. Thus, they can be adjusted according to the application requirements.

2) RBFNN Model - vRBFgen: The RBFNN VHDL code generator is builded using parallel neurons controlled by an FSM. It implements the equations 3 and 4 with the following configuration parameters:

1) $nbits$: Is the bit-width of the floating-point representation.
2) n_{var}: Is the number of inputs.
3) M: Is the number of neurons.
4) $\boldsymbol{c}$: Is an $M \times n_{var}$ matrix with the centers for every neuron.
5) δ: Is an $M \times 1$ array with the neurons' spreads. $1/(2\sigma^2)$ for this case, see Eq. 4. And
6) w: Is an $M \times 1$ array with the output layer weights.

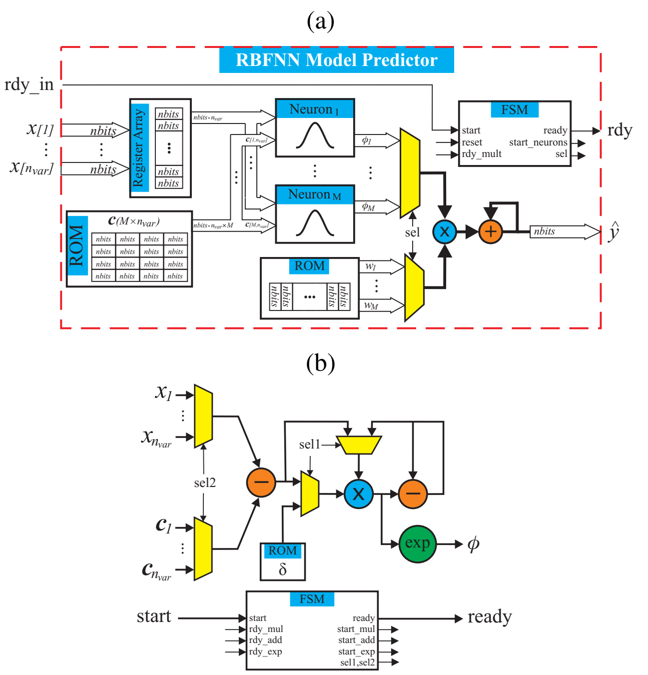


Fig. 7. RBFNN Model Predictor on FPGA. (a) Top level view of the system. (b) Internal FSM of radial based kernel of "Neuron" block.

The RBFNN Model Predictor architecture shown in Fig. 7(a) is composed of: 1) the input layer which contains a "Register Array" block where the inputs are stacked, and a "ROM" block with the coefficients $c(M \times n_var)$, 2), the hidden layer with M parallel neurons and 3) the output layer that is integrated by an "FSM", a "ROM" (containing the weight coefficients) and some operators weighing the neurons' outputs ϕ. Every "Neuron" block implements a Gaussian kernel, as seen in Eq. 4, using an adder/subtractor, a multiplier, and an exponential operator, differently from the Linear Model that only uses the former two. The latter is implemented as a CORDIC (Coordinate Rotation in a Digital Compute) algorithm [36]. The composition of this block is depicted in Fig. 7(b) and is also implemented using an FSM that controls the operations.

As seen in Fig. 7, the RBFNN model in (a) and the NN blocks in (b) consists only of FSM; thus, its latency and throughput are equal. As said before, the NN blocks are implemented in parallel (one for every neuron of the model), and its total of inner operators are kept the same no matter the number of inputs; hence, the latency (L) depends on the number of inputs and can be obtained by

$$L = n_{var} \cdot t_{fsm} + t_{cordic} + (M + 1) \cdot t_{fsm}, \quad (11)$$

where t_{fsm} is the execution time of the multiplier, adder and substractor in an FSM and t_{cordic} is the time the CORDIC algorithm takes to perform an exponential operation. The throughput (T) is calculated using the latency as $T = \frac{1}{L}$.

D. Integration with the acquisition board

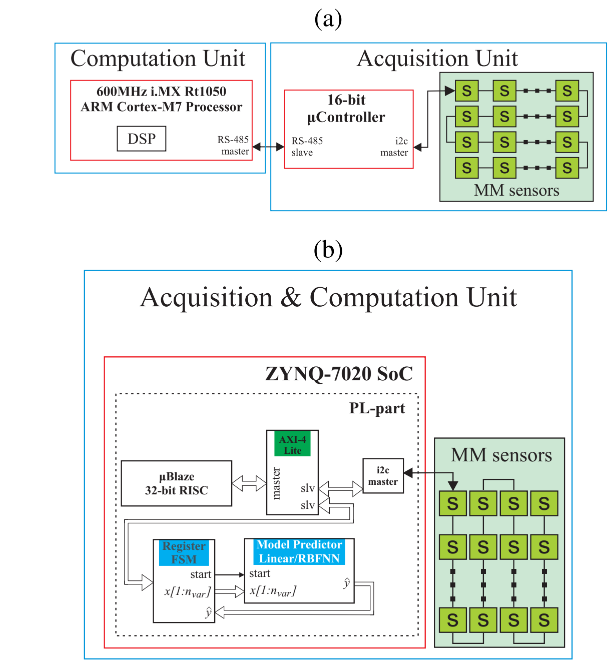


Fig. 8. Localizer for embedded myokinetic sensoring system. (a) Describe the overview of the architecture previously considered in [7]. The AU contains the matrix of the "S" magnetic sensors and a 16-bits μController, while the CU contains the ARM processor in charge of the prediction. (b) Depicts the proposed architecture of this article. The Acquisition and Computation Units are included in the same SoC. Note that the red lines denote the chip's spatial boundaries.

In previous work [7], the localizer comprised two parts: the acquisition unit (AU) and the Computation Unit (CU), see Fig. 8(a). The AU contains a 16-bits micro-controller that acquires data from an array of 32 three-axis magnetic field sensors through an I2C interface. The CU consists of an iMX RT1050 Processor that draws the data from the AU via UART RS-485 and makes the prediction by implementing a set of 96 equations (32 sensors $\times$ three-axis) and solve it using a Levenberg–Marquardt algorithm (LMA) [38].

This work proposes simplifying the localizer implementation by integrating the CU and the AU on a single System on Chip (SoC). To this end, it was used a Zynq-7020 SoC that integrates a dual-core, 650 MHz, ARM Cortex-A9 processor (Programmable Software - PS) with a Xilinx 7-series FPGA (Programmable Logic - PL). Figure 8(b) depicts the proposed architecture (please note that the ARM-core is not used). It is implemented in the PL-part of the SoC using a custom μBlaze (32-bits RISC processor) as a master to manage the I2C interface and the model block accelerators (Linear or RBFNN) through AXI (AMBA eXtensible Interface), which is an On-Chip bus architecture.

IV. Results

In the present section, constructing the linear and RBFNN models is stated together with their hardware implementation results, focusing on the analysis of hardware consumption and accuracy. In the results hereafter given, both models use as input all the measurements made by the magnets, where the Multisine signal in Fig. 2 is used for estimation of the model, while the ramp and sequence of steps in Fig. 3 are used for validation. Firstly, we analyze the results in terms of modeling accuracy and interpret its efficiency for hardware implementation in the following subsections.

TABLE I

Results in terms o MSE [dB] and R² (lower, between parenthesis) for the models tested. It is possible to see that the simple linear model achieves better results with R^2 close to unity and that there is no significant improvement of the RBFNN model by increasing the model complexity. It can be seen that for all models, the error is similar for every dataset. For the RBFNN, it can be seen that increasing the number of neurons does not improve the error dramatically.

Signal / Model	Step	Ramp (20 s)	Ramp (40 s)	Ramp (60 s)	Ramp (80 s)
Linear	-29.33 (0.9854)	-32.50 (0.9929)	-29.76 (0.9870)	-32.55 (0.9931)	-33.28 (0.9948)
RBFNN (6 neurons)	-24.59 (0.9567)	-28.35 (0.9816)	-25.55 (0.9659)	-27.28 (0.9770)	-31.44 (0.9921)
RBFNN (7 neurons)	-24.95 (0.9601)	-28.37 (0.9817)	-25.77 (0.9677)	-27.37 (0.9775)	-30.87 (0.9910)
RBFNN (8 neurons)	-24.68 (0.9576)	-28.59 (0.9826)	-25.47 (0.9653)	-27.17 (0.9764)	-30.94 (0.9911)
RBFNN (9 neurons)	-24.74 (0.9581)	-28.75 (0.9833)	-25.52 (0.9657)	-27.20 (0.9765)	-31.12 (0.9915)
RBFNN (10 neurons)	-25.38 (0.9639)	-29.66 (0.9864)	-26.00 (0.9693)	-27.73 (0.9792)	-31.09 (0.9914)

Using the procedures for determining the parameters of the linear and RBFNN models given in Section III, we obtained the results provided in Table I, which were evaluated offline for the validation metrics R^2 and MSE. The results are similar for every dataset, indicating that the models perform almost equally in different situations, including cases where the magnet moves at different speeds. Furthermore, the results for the RBFNN using different complexities are similar, meaning that more neurons, in this case, do not improve the prediction capability of the model. As the number of neurons for the RBFNN model dramatically increases the hardware architecture consumption, as will be shown later on, it was decided to use eight neurons for this work, utilizing the maximum resources available of the FPGA. Moreover, the linear model achieves better results for all types of muscle deflection signals in prediction accuracy. In the following, we discuss the impact of the hardware implementation of both models on hardware.

A. Hardware utilization and precision analysis

Since it is essential to preserve both good precision error and minimum hardware consumption, a trade-off analysis between the hardware occupation, computational performance, and precision must be addressed to allow the designer to guide the physical implementation. As previously mentioned, the architectures can be customized, and the design criteria depend on the model, the number of operators, and the bit-width representation. The automatically generated model architectures were implemented using the Vivado 2018.2 Design Suite.

Figure 9 presents the MSE metric for both models. It is calculated using MATLAB (64-bits floating-point representation) as reference or golden result (y_i) and all different bit-widths implementations in hardware (See Eq. 5). This is done using all testing data, i.e., ramps and sequence of steps. The ad-hoc architectures' precision is not improved for more than 32 and 38 bit-width representations for the linear and RBFNN models, respectively. The previous illustrates the importance of manipulating the architectures' floating-point precision in design, as the bit-width directly impacts hardware resources and energy consumption. In the figure, the MSE being that small means that the hardware implementation gives a very accurate response for such lower bit-width than the golden reference, which justifies using the tools/methods discussed in the contribution section. Moreover, optimizing hardware consumption for such applications is also essential to enumerate at which bit-width representation the error stalls. The hardware implementation naturally has its limits and does not improve anymore concerning the golden model in software, which is also evident in Table II.

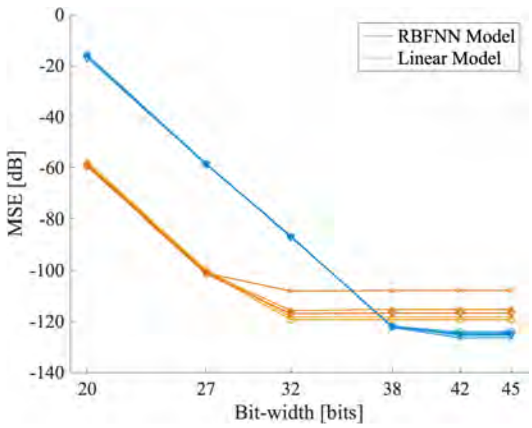


Fig. 9. MSE [dB] vs bit-width for the RBFNN and Linear models using the five tests datasets ('+' Ramp (20 s), 'o' ramp (40 s), '*' ramp (60 s), 'x' ramp (80 s), and 'v' step datasets). It is possible to see that the RBFNN model does not improve significantly for architectures with more than 38 bits, while the same can be affirmed for the linear model with 32 bits.

Table II reports the estimates of the hardware consumption (in terms of DSP blocks, look-up tables (LUTs), and registers)

TABLE II

HARDWARE UTILIZATION AND ERROR ESTIMATION VARYING THE FLOATING-POINT BIT-WIDTH OF THE LINEAR MODEL USING 8 AND 128 OPERATORS. IT CAN BE SEEN THAT INCREASING THE BIT-WIDTH OPERATIONS INCREASES THE HARDWARE CONSUMPTION AND DECREASES THE PREDICTION ERROR SIGNIFICANTLY UP TO 32-BITS.

	Linear Model							**RBFNN Model**			
	LUTs (53200)		**DSP (220)**		**SR (106400)**		MSE	**LUTs**	**DSP**	**SR**	**MSE**
bits	**8 ops**	**128 ops**	**8 ops**	**128 ops**	**8 ops**	**128 ops**	[dB]	**(53200)**	**(220)**	**(106400)**	[dB]
20	2126 (3.9%)	31640 (59.5%)	8 (3.6%)	128 (58.2%)	1829 (1.7%)	27997 (26.3%)	-58.61	50327 (94.6%)	9 (4.1%)	13443 (12.6%)	-16.11
27	3340 (6.3%)	51420 (96.6%)	8 (3.6%)	128 (58.2%)	2419 (2.3%)	37227 (34.9%)	-100.61	67065 (126.1%)	9 (4.1%)	16936 (15.9%)	-58.50
32	3532 (6.6%)	82021 (154.2%)	16 (7.3%)	160 (72.7%)	2869 (2.7%)	44278 (41.6%)	-115.68	78877 (148.3%)	18 (8.2%)	19442 (18.3%)	-86.75
38	4971 (9.3%)	158651 (298.2%)	32 (14.5%)	160 (72.7%)	3415 (3.2%)	52743 (49.6%)	-115.52	93481 (175.7%)	36 (16.4%)	22435 (21.1%)	-122.11
42	5569 (10.5%)	206715 (388.6%)	32 (14.5%)	160 (72.7%)	3775 (3.5%)	58383 (54.9%)	-115.52	103385 (194.3%)	36 (16.4%)	24451 (23.0%)	-125.20
45	6382 (12.0%)	232451 (436.9%)	32 (14.5%)	160 (72.7%)	4045 (3.8%)	62613 (58.8%)	-115.53	111770 (210.1%)	36 (16.4%)	25952 (24.4%)	-125.22

after logical synthesis and the numerical error of the linear and RBFNN models predictors implemented with different bit-width floating-point representations. Given that the different datasets' MSE values are very similar within the same model, the Tables depict the mean of the datasets' MSE for every bit-width. On the other hand, Table III shows the hardware consumption of the extra modules of the integrated architecture depicted in Fig. 8b.

As shown in Table II, the logical synthesis results indicate that the linear model of 128 operators with a bit-width greater than 27 and the RBFNN model with a bit-width greater than 20 do not fit in the selected SoC device. In particular, for both models, the LUTs consumption exceeds the maximum available. However, it is essential to remark that the synthesis and implementation tool can optimize the circuit, with, e.g., area multiplication optimization, enabling it to be effectively mapped on the selected device. Moreover, solutions with less than 20 bits did not achieve acceptable results due to error mitigation; hence, they were not included in the table. On the other hand, architectures with bit-width greater than 32 and 38 bits do not significantly improve the MSE for the linear and the RBFNN model, respectively. Both models achieve satisfactory results with 27 bit-width representation in all validation datasets. The linear model showed an absolute error smaller than 520μm and than 720μm for the RBFNN for 95% of the estimations and mean absolute error deviation 220μm and 340μm, respectively.

TABLE III

HARDWARE UTILIZATION OF THE EXTRA BLOCKS FOR THE INTEGRATION SCHEME.

Block	**LUT (53200)**	**DSP (220)**	**SR (106400)**	BRAM (144)
MicroBlaze	700 (1.31%)	0 ()	334 (0.3%)	8 (5.5%)
I2C Master	179 (0.3%)	0 ()	250 (0.2%)	0 ()
Fifo Mailbox	239 (0.4%)	0 ()	199 (0.2%)	0 ()
AXI-4 Lite	471 (0.9%)	0 ()	576 (0.5%)	0 ()

B. Execution Time Results

Table IV compares the execution time of the architectures. For the linear model, three different values of the N_{op} configuration parameter were studied since they directly affected the execution time. Differently, as explained before, for the RBFNN model, only the number of inputs n_{var} affects the execution time (see Eq. 11).

The sensory data comes from an IIC serial line at a fixed rate of 12.5 ms (80 Hz), the NXP magnetic field sensor's maximum feed rate, which is the system's real-time requirement. An average person flex and extend his fingers no faster than 2 Hz [39] and highly skilled individuals, i.e., piano players, do it at 10.5 Hz at most [40]. Consequently, based on the Nyquist–Shannon sampling theorem, the sensors' feed rate can faithfully identify human finger gestures.

TABLE IV

EXECUTION TIME OF MODEL PREDICTORS

Model Type	N_{op}	Neurons	Execution Time Latency [μs]	Throughput [MFLOPS]
Linear	2	-	11.52	0.09
Linear	8	-	4.80	0.21
Linear	128	-	0.54	2.63
RBFNN	-	8	12.07	0.08

In Table IV it can be seen that the $128N_{op}$ linear model dramatically outperforms the RBFNN model. We can consider this a fair comparison since both models' hardware consumption is similar according to Table II.

Nevertheless, another outstanding result is the $2N_{op}$ linear model due to its similar timing performance with the RBFNN model using much fewer hardware resources. It is a much more suitable approach for this system, given that it makes more sense to compute the data faster than the source. In contrast, the $8N_{op}$ linear model would not correspond to the previous reasoning. However, this solution could also be noteworthy if the sensory board could feed the data using more IIC lines. If, for instance, it could be feed from eight data lines, the full potential of this model could be exploited. The good results achieved with faster and least complex models are essential for the myokinetic interfaces since they will use more magnets.

C. Implementation Feasibility and Power Consumption Estimation

Figure 10 represents the system layout of the 27 bit-width models mapped on a Zynq 7020 SoC device to test the feasibility of implementing the system on real hardware. Both models effectively operate at a clock frequency of 100 MHz. Finally, the models' power consumption estimation is presented in Table V. The energy consumption was not directly measured from the architecture, as it was estimated using the development tool's design-flow. The power consumption related to the linear model is expected to be 56% lower than that of the RBFNN model. The extra blocks are included, and as they are independent of the models, they do not change significantly. The last column shows the total power consumption of the integrated scheme.

TABLE V
POWER CONSUMPTION ESTIMATION OF THE PREDICTION MODELS.

	μBlaze [mW]	Axi4 [mW]	Mbox [mW]	i2c [mW]	Sensors [mW]	Model [mW]	Total [mW]
Linear RBFNN	18	5	1	2	200	39 332	265 558

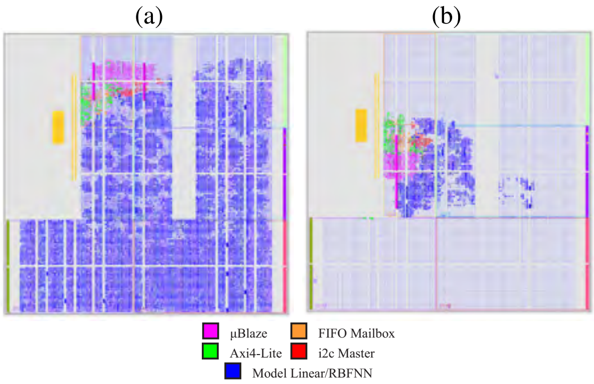


Fig. 10. Device overview of the FPGA utilization for both models of the complete system depicted in Fig. III. Every module is highlighted with a different color. (a) RBFNN Model ($M = 8$). (b) Linear Model ($N_{op} = 8$).

V. DISCUSSIONS

This work presented an embedded solution for solving the magnetic tracking problem for a myokinetic interface. Two machine learning techniques were used as system model predictors to solve the magnets' localization using an anatomically relevant forearm mockup. Both models perform with acceptable accuracy. However, considering only the MSE, the linear model outperforms the RBFNN model with a mean absolute error deviation of 220μm over 340μm, (see Sec. III-B). Both values are comparable to the previous works [5], [7]; however, the functionality of the MIKY interface does not depend much on its capacity to fetch the exact position of the magnet. Repeatability of the measurement is more relevant for this case, i.e., the precision is more important than the accuracy [5]. Table VI reports the variances of the estimated magnet position for nine steps singled sampled from the step dataset. This table was calculated by isolating the magnetic field data corresponding to the moments the magnet is static. In particular, nine sub-datasets were derived: the first in correspondence of the first step (10% the muscle deflection), then the second (20% the muscle deflection), and so on. Table VI states that the RBFNN model has less dispersed error values when considering a static position, given that eight out of the nine values are smaller than those of the linear model. In other words, RBFNN has a better precision because most of its static variances are lower, despite its more significant hardware resource usage.

TABLE VI
VARIANCE OF PREDICTOR MODELS IN μm^2.

Model \ y[mm]	1	2	3	4	5	6	7	8	9
RBFNN	6.48	7.33	6.72	7.28	8.87	9.94	4.98	8.42	9.22
Linear	8.16	8.39	6.97	6.56	11.22	15.10	7.31	8.78	10.05

However, to prove that this direct comparison is valid, it was proved that all nine sub-datasets came from different distributions. Thus, the one-sample Kolmogorov-Smirnov test [41] was performed to test if all the group samples (one for every position of y and model) were normally distributed or not. The hypothesis to test was H_0, stating that the data set has a normal distribution and H_1 otherwise. Since each sub-dataset came out as a non-normal distribution, the next procedure was to use a non-parametric test to see if each sub-dataset originates from an identical distribution or not. The rank-sum test [42] was used where the null hypothesis H_0 states that two independent data sets come from distributions with equal medians and H_1 otherwise. The test confirmed that all the samples were drawn from different distributions, confirming that the RFBNN model is more precise than the linear model.

The computational performance of these architectures proved to be superior to previous embedded solutions [7], achieving execution times of 4.8 and 12.07 μs for the $8N_{op}$ linear and RBFNN models, respectively, compared to 250 μs reported in [7]. This will allow localizing five or more magnets in the same chip taking advantage of the full sampling rate of the magnetic field sensor array. Likewise, this implementation also eases the system's real-time requirements since its execution time is deterministic for both models. It is also remarkable that the previous work [7], being a pure software solution and implementing an optimization algorithm, did not have deterministic execution time, even when using a real-time processing core.

That the system proposed in [7] used only one PCB with 32 three-axis sensors. In contrast, this approach used four PCBs (128 sensors); therefore, if the same scheme were to be used in the previous solution, its execution time would increase because of the superposition effect of solving the inverse problem of magnetostatics [5]. Besides, considering the communication overhead that took most of the system's execution time in [5], the full exploitation of the sensors' sample rate would be even more unattainable. The solution proposed in this work targets that issue by minimizing the communication overhead and reducing the embedded system to a single chip.

Finally, another critical aspect of an embedded portable solution is its energy consumption. Indeed, results showed that the proposed system is more power-efficient in contrast to the previous work [7]. In that case, the AU's power consumption was 550 mW, and that of the CU was 430 mW, resulting in total consumption of 980 mW. Our solution effectively reduced power to less than 57% of that.

VI. Conclusion

This work presented the application of two machine learning models and their ad-hoc implementation on hardware for magnet localization in the context of myokinetic prosthetic control. First, we validated the hypothesis that machine learning can be used for developing data-driven magnet localization approaches, as both models presented results with R^2 close to unity. The linear model presented overall better results in terms of accuracy and hardware consumption compared to the RBFNN model regarding model prediction accuracy. However, the RBFNN model showed a smaller static error, critical for such biosignal interpretation models.

The linear model demands significantly fewer hardware resources when we optimize the architecture by taking advantage of the model's inherently parallel structure. Hardware optimization was further applied for both models by tuning the bit-width numerical representation for the floating-point operations on hardware, highlighting the importance of ad-hoc floating-point precision hardware architectures. Even though the simple linear model presented better results than the RBFNN in terms of accuracy, we suggest testing in future research as more models, including state-of-the-art solutions [22]–[25]. Similarly, more results are needed to test the adherence of machine learning models for more complex behavior observed when using several magnets for tracking different muscles at the same time.

Indeed, the next natural step to test is the localization of many magnets concomitantly. We have proved that it is possible to use machine learning to localize a single magnet. To do so, we need to perform experiments with more than one magnet and evaluate the data-driven models' accuracy in this more challenging scenario. Moreover, other possibilities, such as feature extraction techniques, should be tested to reduce the number of inputs to the model, potentially reducing the hardware resources needed. The latter could also include machine learning adaptation mechanisms [43] for improving the models' accuracy. Any method for dimensionality reduction or feature extraction would reduce the number of input variables, reducing the hardware implementation's complexity. That could potentially reduce FPGA resource utilization and improve computational performance. The data exploration results show that a smaller amount of magnetic information can be used to create the models, which should also be tested in the multi-magnet scenario. Additionally, it will prove useful to test as many models as it is feasible as we are not sure which will be best for a more challenging task [44], and also evaluate stacking ensembles of many different models towards more precise data-driven localization [45], [46].

References

[1] C. J. De Luca, "The use of surface electromyography in biomechanics," *Journal of Applied Biomechanics*, vol. 13, no. 2, 1997.

[2] R. F. Weir, P. R. Troyk, G. A. DeMichele, D. A. Kerns, J. F. Schorsch, and H. Maas, "Implantable myoelectric sensors (IMESs) for intramuscular electromyogram recording," *IEEE Transactions on Biomedical Engineering*, vol. 56, 2008.

[3] M. Ortiz-Catalan, B. Håkansson, and R. Brånemark, "An osseointegrated human-machine gateway for long-term sensory feedback and motor control of artificial limbs," *Science Translational Medicine*, vol. 6, 2014.

[4] S. Micera, J. Carpaneto, and S. Raspopovic, "Control of hand prostheses using peripheral information," *IEEE Reviews in Biomedical Engineering*, vol. 3, 2010.

[5] S. Tarantino, F. Clemente, D. Barone, M. Controzzi, and C. Cipriani, "The myokinetic control interface: tracking implanted magnets as a means for prosthetic control," *Scientific Reports*, vol. 7, no. 1, 2017.

[6] C. R. Taylor, H. G. Abramson, and H. M. Herr, "Low-latency tracking of multiple permanent magnets," *IEEE Sensors Journal*, vol. 19, no. 23, 2019.

[7] F. Clemente, V. Ianniciello, M. Gherardini, and C. Cipriani, "Development of an embedded myokinetic prosthetic hand controller," *Sensors*, vol. 19, no. 14, 2019.

[8] C. M. Bishop, *Pattern Recognition and Machine Learning*, 1st ed. New York, USA: Springer, 2006.

[9] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Computation*, vol. 3, no. 2, 1991.

[10] S. Chen, C. F. Cowan, and P. M. Grant, "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Transactions on Neural Networks*, vol. 2, no. 2, 1991.

[11] S. S. Haykin, *Neural networks and learning machines*, 3rd ed. Upper Saddle River: Prentice Hall, 2009, p. 239 - 245.

[12] H. V. H. Ayala, D. M. Muñoz, C. H. Llanos, and L. dos Santos Coelho, "Efficient hardware implementation of radial basis function neural network with customized-precision floating-point operations," *Control Engineering Practice*, vol. 60, 2017.

[13] C. Song, S. Xie, Z. Zhou, and Y. Hu, "Modeling of pneumatic artificial muscle using a hybrid artificial neural network approach," *Mechatronics*, vol. 31, 2015.

[14] T. D. C. Thanh and K. K. Ahn, "Nonlinear pid control to improve the control performance of 2 axes pneumatic artificial muscle manipulator using neural network," *Mechatronics*, vol. 16, no. 9, 2006.

[15] Q. Wu, B. Chen, and H. Wu, "Neural-network-enhanced torque estimation control of a soft wearable exoskeleton for elbow assistance," *Mechatronics*, vol. 63, 2019.

[16] Q. Wu, X. Wang, B. Chen, and H. Wu, "Development of an rbfn-based neural-fuzzy adaptive control strategy for an upper limb rehabilitation exoskeleton," *Mechatronics*, vol. 53, 2018.

[17] M. K. Burns, D. Pei, and R. Vinjamuri, "Myoelectric control of a soft hand exoskeleton using kinematic synergies," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 6, 2019.

[18] S. Tam, M. Boukadoum, A. Campeau-Lecours, and B. Gosselin, "A fully embedded adaptive real-time hand gesture classifier leveraging hd-semg and deep learning," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 14, no. 2, 2020.

[19] H. Dantas, D. J. Warren, S. M. Wendelken, T. S. Davis, G. A. Clark, and V. J. Mathews, "Deep learning movement intent decoders trained with dataset aggregation for prosthetic limb control," *IEEE Transactions on Biomedical Engineering*, vol. 66, no. 11, 2019.

[20] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A survey of fpga-based neural network inference accelerators," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 12, no. 1, 2019.

[21] A. Shawahna, S. M. Sait, and A. El-Maleh, "Fpga-based accelerators of deep learning networks for learning and classification: A review," *IEEE Access*, vol. 7, 2019.

[22] Z. Hajduk, "High accuracy FPGA activation function implementation for neural networks," *Neurocomputing*, vol. 247, 2017.

[23] Q. Hu, X. Tang, and W. Tang, "A smart chair sitting posture recognition system using flex sensors and fpga implemented artificial neural network," *IEEE Sensors Journal*, vol. 20, no. 14, 2020.

[24] R. Celikel, "ANN based angle tracking technique for shaft resolver," *Measurement*, vol. 148, 2019.

[25] A. D. Pano-Azucena, E. Tlelo-Cuautle, L. G. De La Fraga, C. Sanchez-Lopez, J. J. Rangel-Magdaleno, and S. X. Tan, "Prediction of chaotic time-series with different MLE values using FPGA-based ANNs," *SMACD 2017 - 14th International Conference on Synthesis, Modeling,*

Analysis and Simulation Methods and Applications to Circuit Design, 2017.

[26] M. Mohammadi, A. Krishna, S. Nalesh, and S. K. Nandy, "A Hardware Architecture for Radial Basis Function Neural Network Classifier," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 3, 2018.

[27] Z.-C. Fan and W.-J. Hwang, "Efficient VLSI architecture for training radial basis function networks," *Sensors*, vol. 13, no. 3, 2013.

[28] H.-H. Chou, Y.-S. Kung, N. V. Quynh, and S. Cheng, "Optimized FPGA design, verification and implementation of a neuro-fuzzy controller for PMSM drives," *Mathematics and Computers in Simulation*, vol. 90, 2013.

[29] A. C. D. de Souza and M. A. C. Fernandes, "Parallel fixed point implementation of a radial basis function network in an FPGA," *Sensors*, vol. 14, no. 10, 2014.

[30] J. Kim and S. Jung, "Implementation of the RBF neural chip with the back-propagation algorithm for on-line learning," *Applied Soft Computing*, vol. 29, 2015.

[31] L. Yuan, H. Zhao, H. Chen, and B. Ren, "Nonlinear mpc-based slip control for electric vehicles with vehicle safety constraints," *Mechatronics*, vol. 38, 2016.

[32] H. Yu, T. Xie, S. Paszczyñski, and B. M. Wilamowski, "Advantages of radial basis function networks for dynamic system design," *IEEE Transactions on Industrial Electronics*, vol. 58, 2011.

[33] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: With Applications in R*, 1st ed. New York: Springer, 2014.

[34] Zhou Wang and A. Bovik, "Mean squared error: Love it or leave it? A new look at Signal Fidelity Measures," *IEEE Signal Processing Magazine*, vol. 26, no. 1, 2009.

[35] R. G. D. Steel, J. H. Torrie, and D. A. Dickey, *Principles and procedures of statistics: a biometrical approach*, 2nd ed. McGraw-Hill, 1980.

[36] D. M. Muñoz, D. Sánchez, C. Llanos, and M. Ayala-Rincón, "FPGA-based floating-point library for CORDIC algorithms," in *Proc. IEEE Int. Conf. Southern Programmable Logic*, Porto de Galinhas, Brazil, 2010.

[37] ——, "Tradeoff of FPGA design of a floating-point library for arithmetic operators," *International Journal of Integrated Circuits and Systems*, vol. 5, no. 1, 2010.

[38] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly of Applied Mathematics*, vol. 2, 1944.

[39] C. Häger-Ross and M. H. Schieber, "Quantifying the Independence of Human Finger Movements: Comparisons of Digits, Hands, and Movement Frequencies," *The Journal of Neuroscience*, vol. 20, no. 22, nov 2000.

[40] S. Furuya and J. F. Soechting, "Speed invariance of independent control of finger movements in pianists," *Journal of Neurophysiology*, vol. 108, no. 7, oct 2012.

[41] F. J. Massey, "The Kolmogorov-Smirnov Test for Goodness of Fit," *Journal of the American Statistical Association.*, vol. Vol. 46, no. 253, 1951.

[42] J. D. Gibbons and S. Chakraborti, *Nonparametric Statistical Inference*, 5th ed. Boca Raton, FL: Chapman & Hall/CRC Press, Taylor & Francis Group, 2011.

[43] P. Kadlec, R. Grbić, and B. Gabrys, "Review of adaptation mechanisms for data-driven soft sensors," *Computers & Chemical Engineering*, vol. 35, 2011.

[44] M. Kuhn and K. Johnson, *Applied predictive modeling*. Springer, 2013, vol. 26.

[45] K. Gu, Z. Xia, and J. Qiao, "Stacked selective ensemble for pm2.5 forecast," *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 3, 2020.

[46] K. Yu and X. Xie, "Predicting hospital readmission: A joint ensemble-learning model," *IEEE Journal of Biomedical and Health Informatics*, vol. 24, no. 2, 2020.