Configuration of the Actor and Critic Network of the Deep Reinforcement Learning controller for Multi-Energy Storage System

Paula Páramo-Balsa
Department of Electrical Engineering
Universidad de Sevilla
Seville, Spain
pparamo@us.es

F. Gonzalez-Longatt
Department of Electrical Engineering, Information Technology and Cybernetics
University of South-Eastern Norway
Porsgrunn, Norway
fglongatt@fglongatt.org

Martha N. Acosta
Department of Electrical Engineering, Information Technology and Cybernetics
University of South-Eastern Norway
Porsgrunn, Norway
Martha.Acosta@usn.no

Jose Luis Rueda Torres
Department of Electrical Sustainable Energy
Delft University of Technology
Delft, The Netherlands
J.L.RuedaTorres@tudelft.nl

Peter Palensky
Department of Electrical Sustainable Energy
Delft University of Technology
Delft, The Netherlands
P.Palensky@tudelft.nl

Francisco Sanchez
Wolfson School of Mechanical, Electrical and Manufacturing Engineering
Loughborough University
Loughborough, UK
f.sanchez@ieee.org

Juan Manuel Roldán-Fernández
Department of Electrical Engineering
Universidad de Sevilla
Seville, Spain
jmroldan@us.es

Manuel Burgos-Payán
Department of Electrical Engineering
Universidad de Sevilla
Seville, Spain
mburgos@us.es

***Abstract*—The computational burden and the time required to train a deep reinforcement learning (DRL) can be appreciable, especially for the particular case of a DRL control used for frequency control of multi-electrical energy storage (MEESS). This paper presents an assessment of four training configurations of the actor and critic network to determine the configuration training that produces the lower computational time, considering the specific case of frequency control of MEESS. The training configuration cases are defined considering two processing units: CPU and GPU and are evaluated considering serial and parallel computing using MATLAB® 2020b Parallel Computing Toolbox. The agent used for this assessment is the Deep Deterministic Policy Gradient (DDPG) agent. The environment represents the dynamic model to provide enhanced frequency response to the power system by controlling the state of charge of energy storage systems. Simulation results demonstrated that the best configuration to reduce the computational time is training both actor and critic network on CPU using parallel computing.**

Keywords—actor-network, critic network, deep reinforcement learning, energy storage systems, enhanced frequency response, parallel computing.

I. INTRODUCTION

The 2021 United Nations Climate Change Conference, COP26, produced the first multi-county reach net-zero emissions (over 140 countries). However, reaching net-zero emission is a complex task involving multiple actors in the energy sector. The electricity sector and, in particular electrical utilities, face a very complex path ahead. The very drastic and fast move in the sector to cope needed to reach zero-carbon implies the transition to high penetration of generation coming from environmentally friendly sources at the time. One necessary consequence is the sudden reduction of the total system rotational inertia that negatively affects the power system operation and security. However, seral technologies are taking a position as enablers of the zero-net future. The energy storage systems (ESS) are a desirable alternative to counteract the lack of inertia of the power system due to its controllability features and its ability to fast supply active power to the power system [1], [2].

Currently, there is a wide variety of ESS technologies, but among all the ESS, it is worth highlighting batteries (BESS), supercapacitors (UCESS) and flywheels (FESS) [3]. Authors in [4] proposed the use of a multi-electrical energy storage system (MEESS) to provide fast frequency response service, precisely the Enhanced Frequency Response (EFR) [5], [6]. The MEESS combines BESS, UCESS and FESS. In [4], a two-layer controller was presented where a fuzzy logic control was used to define the power reference of the MEESS considering the frequency deviation and the state of charge (SOC) of the storage assets. However, that publication made evident the need to carefully control the SOC of BESS, UCESS and FESS need be controlled to ensure it is within certain limits and guarantee an adequate amount of energy to operate when necessary [7].

Authors in [8] took advantage of reinforcement learning as a universal function approximator and proposed a deep reinforcement learning-based controller for the SOC management of MEESS. The authors compare the performance of the proposed Deep Reinforcement Learning (DRL) to classical proportional integral derivative (PID) control and a fuzzy logic controller (FLC), demonstrating the suitability of the proposed approach.

The DRL consists of an agent and an environment that interact through a sequence of actions, observations and rewards. The agent takes action on the environment to maximise the reward [8]. The training process is carried out over many episodes for the agent to learn the actions that it must take to get the maximum reward. Although the training process might require considerable time, the use of appropriate hardware reduces the total computational time.

The authors collaborate with the Digital Energy System Laboratory, DigEnSys-Lab [9], where specialised hardware has been implemented to develop research in the area of artificial intelligence. As a consequence, this paper is motivated by the need of the research team to experimentally identify the most favourable hardware arrange when training actor and critic networks to solve the delicate problem related to frequency control using MEESS.

This scientific paper presents an experimental assessment to determine the most favourable configuration to train the Deep Deterministic Policy Gradient (DDPG) agent for Multi-Energy Storage systems. It considers the central processing unit (CPU) and graphics processing unit (GPU) to train the DRL actor and critic network. Moreover, this assessment considers the powerful advantages provided by parallel computing. Finally, an experimental assessment of the training process is performed in terms of computational time-consuming (elapsed time) when the actor and critic network training is carried out on only CPU, only GPU or a combination of both.

The paper is structured as follows. Section II briefly describes the reinforcement learning framework and its components. Moreover, it describes the principle of frequency control using DRL. Section III presents the serial and parallel computing description and the main instructions used on MATLAB® 2020b Parallel Computing Toolbox MATLAB®. Section IV presents the experimental results and the principal findings of this paper. Finally, Section V provide the main conclusion of this paper.

II. Deep Reinforcement Learning in Frequency Response Control

The deep neuronal network (DNN) is an excellent option to be used as a universal function approximator. If the DNN is trained using reinforcement learning (RL) techniques, DRL formation [8] is an attractive technique for control purposes. The RL framework is composed of an agent and an environment. The agent contains two components: (i) a policy and (ii) a DRL algorithm. It receives a set of observations and a reward which are used by the DRL algorithm to determine the following action and provide a policy. The policy maximises the total upcoming rewards. Meanwhile, the environment represents the dynamic model of the problem to be solved. The environment uses the actions coming from the agent to achieve a task. It generates a set of observations that are the resulting behaviour of the dynamic model. Then, it calculates a reward that represents how well the actions contribute to accomplishing the task. The primary purpose of the RL is to train an agent to achieve a task inside an uncertain environment. The RL framework, which contains the iteration between the agent and the environment, is shown in Fig. 1.

There are several DRL algorithms in the literature, and they can be classified into three categories: (i) Value-based, (ii) policy-based and (iii) actor-critic. Depending on the DRL algorithms, the agent uses one or several DNN for training the policy and the DNN can be used in two different ways: (i) as an actor-network which, for a given observation, it returns the action that maximises the expected cumulative rewards and (ii) as a critic network which, for a given observation and action, it returns the expected value of the cumulative long-term reward for the task.

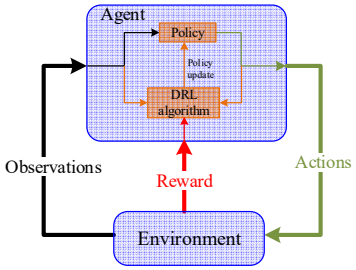

Fig. 1. Illustrative diagram showing the main components of the RL framework and their iteractions and signal flow.

A. SOC control of the ESS in the RL framework

The structure of the RL framework is taken from [8]. The main objective is to provide EFR services to the power system while the SOC of the ESS is controlled. Below a brief description is presented: (i) *Agent*: The DDPG, which is an actor-critic algorithm type, is used. Every time step, the agent generates a set of actions and receives a set of observations and a reward. (ii) *Environment*: The environment consists of the dynamic model of three ESS technologies (BESS, UCESS and FESS) equipped with an EFR controller. Each time step (t), the environment produces a set of outputs (observations and reward) and receives a set of inputs which are the actions generated by the agent. (iii) *Actions*: The actions are the control reference of the ESS technologies and are related to the injection/absorption of active power to charge or discharge the ESS. (iv) *Observations*: The observation vector contains the active power output, the previous control action, the SOC signal and the SOC error signal of each ESS. (v) Rewards: The total reward is the sum of the discrete and continuous rewards obtained from the environment. The formulation of the rewards is in such a way that penalised if the SOC is not within predefined values by using the SOC error.

III. Parallel Computing in DRL

Since the emergence of the first computers, they have evolved and become a fundamental tool in the research area in recent years. Consequently, advances in various disciplines have been greatly accelerated by advances in computing and the reduction in computer prices [10], [11]. Traditionally, computer programs were developed so that, given a set of instructions, they were implemented in series. In this way, only one instruction could be executed at a time, and the subsequent instructions could not be executed and must wait until the current instruction was finished. This can be seen in Fig. 2. The problem to be solved is written as a set of instructions executed one by one following the order in which they were written. Recently, the significant technological

advances in software and hardware have allowed models and solve several complex problems closer to reality. This entails defining more complex mathematical problems with a high number of operations. For this reason, this type of serial computation is not as efficient as the time it would take to solve these problems would be extremely high, so the process would not be computationally efficient [12]. Due to the drawbacks mentioned above of serial computing, a new concept to perform multiple processes simultaneously is raised, and it is called parallel computing. The main objectives of parallel computing are related to the possibility of adjusting the model and software to reality. In this way, if several processors are working simultaneously, it is possible to reduce the simulation time and improve their efficiency significantly.

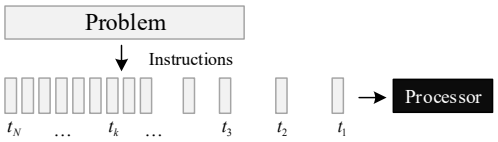

Fig. 2. Block diagram depicting a generic scheme of serial computing.

Parallel computing enables the possibility of splitting the main problem into several independent tasks, each containing a set of instructions, where all tasks are executed simultaneously. Moreover, each task executes its instruction set using a serial computing process (see Fig. 3). Thus, parallel computing produces a considerable reduction in total execution time.

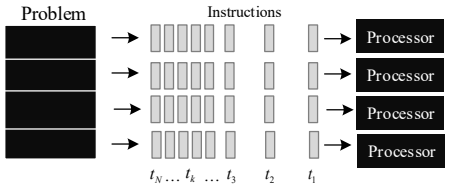

Fig. 3. Block diagram depicting a generic scheme of parallel computing.

Parallel computing usually deals with the *Master* / *Worker* paradigm. Thus, a process acts as a coordinator, Master, directing the rest, which acts as *Workers*. The *Master* is responsible for dividing the problem into smaller tasks, distributing them among the different *Workers* and receiving the partial results to compose the final solution of the problem. On the other hand, the *Workers* must receive the task to perform, then process it and send the results to the *Master*. This procedure is iterative until the *Master* has no more tasks to solve. Also, the partial results received by the *Master* can generate new tasks which must be distributed to the *Workers* [13]. According to the user's choice, the *Master* can be distributing the task to the *Workers* either in a synchronised or non-synchronised way. In the synchronised distribution, the Master waits until it receives all the *Workers*' results and then generates a new set of data to distribute. Meanwhile, in the non-synchronised distribution, the *Master* immediately after receiving the result of any *Worker* it assigns a new set of data to that *Worker*. Therefore, *Workers* do not require synchronisation between them.

Nowadays, several computational programs use parallel computing as the primary mechanism to speed up their processes. One of those computational programs is MATLAB® which is equipped with a powerful MATLAB® Parallel Computing Toolbox that solves problems with a significant computational load using multicore processors, graphics processing units and clusters. This toolbox can execute a task in parallel using several *Workers* (MATLAB® calculation engines) that work locally. Therefore, if the computer has multiple processors, the *Workers* will activate them to perform operations more quickly. Fig. 4 depicts the general architecture of the parallel computing toolbox.

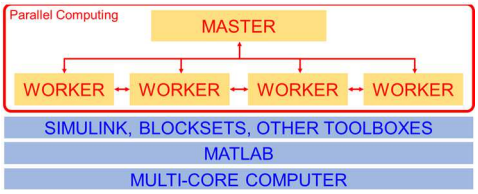

Fig. 4. Illustrative architecture used by parallel computing toolbox [13].

The parallel computing toolbox is widely used in medical image processing, control systems or Deep Learning (DL), among others [14]. Focusing on the DL and neural networks field, new forms of training have been developed to allow the programmer to use several options such as the CPU, multiple GPUs, or parallel computing to solve problems [15]. The selection of any of these options is easily made by using the following instruction:

```
trainingOptions.ExecutionEnvironment = 'multi-gpu'
```

However, this instruction is not valid for DRL applications. The parallel computing toolbox only allows the user to select GPU or CPU as an option to training the agents, and it is done by using the following instruction:

```
rlRepresentationOptions.UseDevice = 'gpu' or 'cpu'
```

As a consequence, the multi-GPU option for DRL training purposes has not to be developed and included in the parallel computing tool. On the other hand, in order to speed up the DRL training calculations and make it efficient, MATLAB® allows user to select the Parallel Computing option within the training options, and it is enabled by the following instruction:

```
rlTrainingOptions.UseParallel = 'true'
```

IV. Experiments and Results

This section is dedicated to presenting the numerical results of the actor and critic configuration assessment for a DRL frequency controller for MEESS. In this paper, the focus on the configuration of actor and critic based on the hardware, the implementation of the DRL frequency controller for MEESS was provided by the authors of [8]. Fig. 5 shows the bloc diagram of the EFR controller of an EESS, which determines the power reference, P^*_{EESS} from the grid frequency, $f(t)$. Then, the EESS model takes as an output the actual power response from the EESS, $P_{EESS}(t)$. This paper considers the implementation of BESS, FESS and UCESS as presented in [4], mode trails of modelling and parameters are shown in [4].

The DRL agent-environment model of the frequency control was modelled in Simulink®, Fig 6. and Fig 7. The full details of the model and its parameters can be found in [8]. For assessment purposes, the total simulation time is chosen as 1800 seconds, the time step is 1 second, and the maximum number of episodes is 100. The experiments are divided into two main tests: Test 1: *serial computing* and Test 2*: parallel computing*. For each test, four possible training configurations

of the actor and critic network training are considered, and it is shown in Table I.

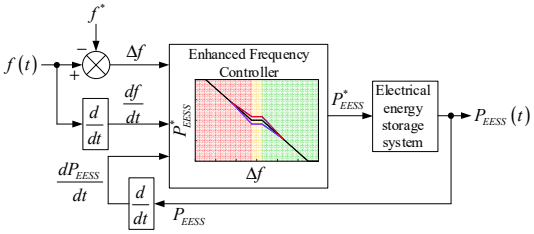

Fig. 5. EFR controller integrated with the EESS to control output power (P_{EESS}) and ramp rate (dP_{EESS}/dt). f^* is the nominal system frequency (50 Hz) [4].

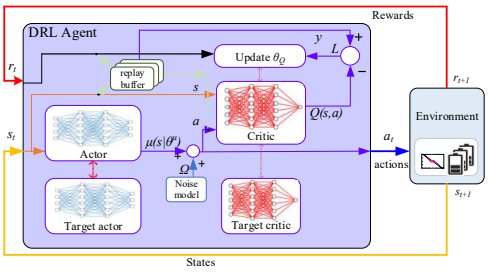

Fig. 6. Overview of an RL agent trained with DDPG [8].

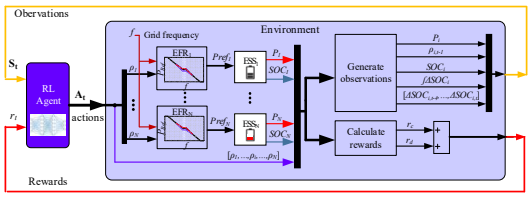

Fig. 7. Overview of the control environment used to train the DRL agent [8].

TABLE I. ACTOR AND CRITIC NETWORK TRAINING CONFIGURATION

Network	Training Configuration			
	I	*II*	*III*	*IV*
Actor	CPU	CPU	GPU	GPU
Critic	CPU	GPU	CPU	GPU

The computational platform used for the training is a specifically designed computer at the DigEnSys-Lab. The computer uses an operating system, Windows 10 Pro (64 bit), with 64 GB of RAM. The processor is an Intel® Xeon® W-3235 CPU (3.30 GHz), and the graphic processor is an NVIDIA GeForce RTX 2080 Ti GPU; for more details, visit [9].

A. Test 1: Serial computing

The serial computing tests consists of evaluating the four training configurations presented in Table I. The resulting computational time after executing the training over 100 episodes, considering the four training configurations is shown in Table II.

TABLE II. COMPUTATIONAL TIME RESULTS USING SERIAL COMPUTING TO TRAIN THE ACTOR AND CRITIC NETWORK

Training configuration	Serial Computing		
	Training	*Time (s)*	*Time (h)*
A	✓	13,126	3.65
B	×	-	-
C	✓	13,983	3.88
D	✓	15,610	4.34

From Table II, it can be observed that training configurations A, C and D have been implemented successfully. On the other hand, training configuration B, which refers to training the actor-network on CPU and critic network on GPU, failed. The DRL toolbox of MATLAB® does not support such training configuration. The training configuration A (CPU-CPU) shows the best performance, i.e., for the same 100 episodes, it takes 3.65 hours of training, which compared with the worst performance produced by training configurations D (GPU-GPU), it reduces the computational time-consuming 15.9 %, saving 0.65 hours (41.4 minutes). Even though using training configuration I to train the actor and critic network reduces the computation time, it must be noticed that the training just considered 100 episodes, and usually, the DRL uses at least 2000-3000 episodes to be correctly trained. Therefore, the computational time using serial computing is still considered to be very high. For instance, if 2000 episodes (20 times the current value) are used, the computation time would be excessive, approximately 73 hours.

B. Test 2: Parallel computing

Due to the high computation time required by the DRL to carry out 100 training episodes using serial computing, the MATLAB® Parallel Computing Toolbox is used to train the actor and critic network. The training considers the same parameter of serial computing: the total simulation time is chosen as 1800 seconds, the time step is 1 second, and the maximum number of episodes is 100. In this scenario, the four training configurations defined in Table I are considered. Besides, it is contemplated the effect of using the maximum number of *Workers* available or if it is preferable to leave a *Worker* available to perform the rest of the operations required. The platform used for the training has 24 logical processors, which is a maximum of 12 *Workers*. Table III present the resulting computational time after executing the training over 100 episodes, considering the four training configurations defined in Table I.

TABLE III. COMPUTATIONAL TIME RESULTS USING PARALLEL COMPUTING TO TRAIN THE ACTOR AND CRITIC NETWORK

Training configuration	Parallel Computing			
	Maximum workers	*Training*	*Time (s)*	*Time (h)*
A	11	✓	3,606	1.00
	12	✓	3,910	1.09
B	4	×	-	-
C	4	✓	4,094	1.14
	3	✓	4,655	1.29
D	4	✓	5,773	1.60

From Table III, it can be observed that the training configuration B (CPU-GPU) the training failed as occurred using serial computing. In general, the rest of the training configurations have successful training and reduce around one-third of the computational time concerning serial computing. Another interesting result is that maintaining one *Worker* free to perform other operation outcomes in a computational time reduction of 10 %, in contrast, to use the maximum number of *Workers* when using training

configuration A (CPU-CPU). Meanwhile, in training configuration C, which involves GPU and CPU, there is a computational time reduction of around 20 % when the maximum number of *Workers* are used compared with the case when one Worker is set to be available.

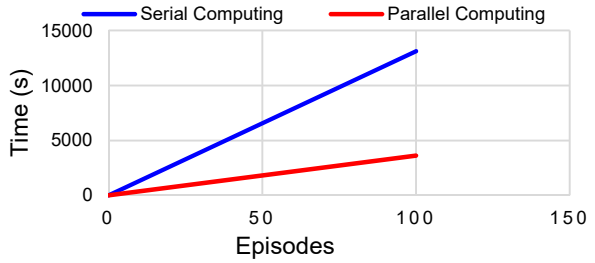

Fig. 8. Computational time comparison for serial and parallel computing using training configuration A(CPU – CPU), Test 2: Parallel computing.

The training configuration A (CPU-CPU) leaving one Worker available is the best configuration to train the actor and critic network since its configuration produces less computational time in both serial and parallel computing. Moreover, using parallel computing with training configuration A and leaving one Worker available is the best option since it reduces 73 % of the computational time (see Fig. 8). The principal reason the computation time is slow down when using GPU is the DRL toolbox of MATLAB® 2020b has no multiple GPU training support. It produces all calculations to be performed on the same GPU even though there are more GPUs available.

V. Conclusions

This scientific paper presents an experimental assessment to determine the most favourable configuration to train the Deep Deterministic Policy Gradient (DDPG) agent for frequency controller of a MEESS. The assessment considers two hardware schemes for the training: the central processing unit (CPU) and graphics processing unit (GPU); they are used to train the DRL actor and critic network. The simulations were divided into two groups: serial computing and parallel computing. The MEESS has been provided in the form of a Simulink model, and MATLAB® Parallel Computing Toolbox has been used for the DDPG. The results of assessing four training configuration cases, considering serial and parallel computing, have demonstrated that the best suitable configuration to perform the actor and critic network training is using the parallel computing for both actor and critic network without using the maximum number *Workers*. This configuration reduces around 73% of the computation time compared to serial computing, representing a remarkable improvement to training DRL algorithms that usually involve thousands of episodes to be trained. The computation time showed better results when training the actor and critic network on CPU than using GPU or CPU-GPU configuration. The increase in computation time is due to the DRL toolbox of MATLAB® 2020b having no support for training on multiple GPU and producing a time slowdown because the computation is carried out only on one GPU. This result shows that although the latest released computers offer excellent tools for speeding up computational time to solve complex problems, computer programs still need to be improved to take advantage of all the benefits that hardware offers. In particular, the implementation of DRL algorithms still has a gap concerning parallel computing that must be covered to take advantage of all benefit that using multiple GPUs provide to speed up the training time. As future work, the proposed configuration strategies allow to simulate more complex systems and to train more advanced DRL in a lower period of time. In fact, the authors are working in adding to the actual system economic criteria. In this way, the enhanced frequency response service will be provided while maximizing the economic benefit. For these reasons, adding economic reward to the discrete and continuous reward of the power system used in [8] will be necessary.

References

[1] M. N. Acosta, F. Gonzalez-Longatt, S. Denysiuk, and H. Strelkova, "Optimal Settings of Fast Active Power Controller: Nordic Case," in *2020 IEEE 7th International Conference on Energy Smart Systems (ESS)*, May 2020, pp. 63–67, doi: 10.1109/ESS50319.2020.9160281.

[2] H. R. Chamorro, F. R. S. Sevilla, F. Gonzalez-Longatt, K. Rouzbehi, H. Chavez, and V. K. Sood, "Innovative primary frequency control in low-inertia power systems based on wide-area RoCoF sharing," *IET Energy Syst. Integr.*, Feb. 2020, doi: 10.1049/iet-esi.2020.0001.

[3] A. M. Ersdal, F. Gonzalez-Longatt, M. N. Acosta, J. L. Rueda, and P. Palensky, "Frequency Support of Fast-Multi-Energy Storage Systems in Low Rotational Inertia Scenarios," in *2020 IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe)*, Oct. 2020, pp. 879–883, doi: 10.1109/ISGT-Europe47291.2020.9248843.

[4] F. Sanchez, J. Cayenne, F. Gonzalez-Longatt, and J. L. Rueda, "Controller to Enable the Enhanced Frequency Response Services from a Multi-Electrical Energy Storage System," *IET Gener. Transm. Distrib.*, Nov. 2018, doi: 10.1049/iet-gtd.2018.5931.

[5] "National Grid ESO, 'Mandatory response services.'" https://www.nationalgrideso.com/balancing-services/frequency-response-services/mandatory-response-services?overview (accessed Aug. 15, 2020).

[6] F. Sanchez, F. Gonzalez-Longatt, and D. Bogdanov, "Probabilistic assessment of enhanced frequency response services using real frequency time series," *2018 20th Int. Symp. Electr. Appar. Technol. SIELA 2018 - Proc.*, pp. 1–4, 2018, doi: 10.1109/SIELA.2018.8447080.

[7] F. Gonzalez-Longatt, M. N. Acosta, H. R. Chamorro, and D. Topic, "Short-term Kinetic Energy Forecast using a Structural Time Series Model: Study Case of Nordic Power System," 2020.

[8] F. S. Gorostiza and F. Gonzalez-Longatt, "Deep Reinforcement Learning-Based Controller for SOC Management of Multi-Electrical Energy Storage System," *IEEE Trans. Smart Grid*, vol. 3053, no. c, pp. 1–1, 2020, doi: 10.1109/tsg.2020.2996274.

[9] "Website of fglongatt-Lab: DigEnSys-Lab," 2021. https://fglongattlab.fglongatt.org/index.html (accessed Jan. 15, 2022).

[10] W. Gao, Q. Kemao, H. Wang, F. Lin, and H. S. Seah, "Parallel computing for fringe pattern processing: A multicore CPU approach in MATLAB® environment," *Opt. Lasers Eng.*, vol. 47, no. 11, pp. 1286–1292, Nov. 2009, doi: 10.1016/j.optlaseng.2009.04.018.

[11] S. Rastogi and H. Zaheer, "Significance of parallel computation over serial computation," in *2016 International Conference on Electrical, Electronics, and Optimisation Techniques (ICEEOT)*, Mar. 2016, pp. 2307–2310, doi: 10.1109/ICEEOT.2016.7755106.

[12] Q. Wu, M. Spiryagin, C. Cole, and T. McSweeney, "Parallel computing in railway research," *Int. J. Rail Transp.*, vol. 8, no. 2, pp. 111–134, Apr. 2020, doi: 10.1080/23248378.2018.1553115.

[13] "Mathworks. 'Parallel Computing Support in MATLAB and Simulink Products.'" https://es.mathworks.com/products/parallel-computing/parallel-support.html (accessed Nov. 01, 2020).

[14] S. Neralkar and J. Katti, "An Efficient Technique of Parallel Share Generation and Reconstruction for Medical Images," in *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, Aug. 2018, pp. 1–6, doi: 10.1109/ICCUBEA.2018.8697617.

[15] M. Antonio Cruz, R. Silva Ortigoza, C. Marquez Sanchez, V. M. Hernandez Guzman, J. Sandoval Gutierrez, and J. C. Herrera Lozada, "Parallel Computing as a Tool for Tuning the Gains of Automatic Control Laws," *IEEE Lat. Am. Trans.*, vol. 15, no. 6, pp. 1189–1196, Jun. 2017, doi: 10.1109/TLA.2017.7932708.