Failure Detection in PLC Controlled Industrial Machines Using Machine Learning

Guilherme A. P. Guim * **Maurício A. Dias** *

* *Intelligent Systems and Autonomous Vehicles (FHO-SIVA), Department of Engineering, Fundação Hermínio Ometto, SP, (e-mail: guilhermeguim@alunos.fho.edu.br ; macdias@fho.edu.br).*

Abstract: PLCs are the backbone of modern industrial automation, providing precise and reliable control over manufacturing processes, ensuring efficiency, safety, and cost-effectiveness. The use of PLCs has revolutionized the industrial sector, enabling companies to optimize their production processes, reduce downtime, increase productivity, and ultimately, stay ahead of the competition. A range of factors such as electrical and mechanical issues, environmental factors, software errors, and human error can cause failure in machines controlled by PLCs, underlining the critical importance of regular maintenance, proper testing, and thorough validation of PLC programming to prevent costly downtime and ensure the smooth operation of industrial processes. Identifying errors in these machines can be challenging due to the complex nature of industrial processes, the potential for multiple causes of failure, and the intricate programming involved in PLCs, underscoring the importance of having experienced professionals with specialized knowledge in PLCs and industrial automation to diagnose and resolve issues quickly and effectively. This work presents the development of a machine learning algorithm to identify operation sequence failures in machines controlled by PLCs. The system achieved an accuracy of 90% in detecting abnormal actuator activation or non-activation, registering an alarm condition and providing the exact input or output address of the error. The results indicate that the system has the potential to reduce downtime and the number of stops in industrial operations, showcasing the potential of machine learning techniques in industrial applications.

Keywords: Machine Learning, Operation sequence failures, PLC, Python, Failure Detection, Downtime reduction

1. INTRODUCTION

In the current scenario of the fourth industrial revolution, the use of devices capable of obtain generated data in various ways within manufacturing factories has brought a great challenge to companies: finding ways to use this data for their own benefit (Azizi and Barenji, 2022). With the emergence of new technologies, the automation of the production line has become essential so that companies can be more productive, increase the quality of products, offer better conditions to workers in workstations, and reduce the final production cost. The implementation of automation offers significant potential for companies seeking strategic growth in a challenging market (Lamb, 2013).

The PLC is responsible for controlling industrial processes and is therefore a major source of data related to the drives and states of the actuators, as well as the machine drive sequences (Petruzella, 2016). However, this information is usually difficult to use. The data can have many variations, there are collection problems, and there are limitations of the equipment's own hardware and architecture. In order to maintain productivity and efficiency rates in a production line, it is essential that reliability and availability information be highly controlled by planning teams. However, unexpected failures in machines controlled by PLCs are a recurring problem, impacting various indicators, such as long production stops, production loss, and increased maintenance costs.

One interesting way to solve this problem is the use of machine learning algorithms, that can allow companies to gain more control over their processes and operations, analyzing patterns that would not be easily perceived by humans. The use of machine learning with data obtained from PLCs, combined with traditional fault diagnostic methods, can reduce downtime due to failure or even prevent it.

The main goal of this work is to develop a machine learning algorithm capable of capturing data related to drive sequences in the inputs of a Siemens PLC, processing them, and learning the equipment operating patterns. Subsequently, the algorithm must analyze the triggering patterns in real time, issuing alert messages when a movement is different from what was learned. The system must also make a prediction of what is the expected state at the time it finds the failure and accurately point out the failure address.

During the results analysis of applying this model, it was possible to calculate an accuracy of over 90% in real-time classification and indication of sequence faults in PLC-controlled machines. This results showed the usefulness of machine learning systems in the industrial environment

and encourage their use by maintenance sectors, improving industrial processes and reducing the time and amount of unplanned production line stops.

The remainder of this paper is presented as follows: some studies related to the proposed topic will be presented, addressing the current situation of the use of machine learning in the diagnosis of faults in industrial equipment. Next section presents materials and methods and the description of the environment where the project was applied, including data collection and the methodology for training, testing, and analysis of the model used. In the results and analysis section, the results obtained with the application of the system will be presented, as well as the impact it had on the production line. Finally, the conclusion section will analyze the reach of the established objectives, possible contributions to the industrial context, and final considerations.

2. RELATED WORKS

This work's proposal is relatively different compared to other failure detection found in literature, but some previous works present interesting discussion about the topic. Previous automation scenario indicates that the main issue was to test the PLC in order to investigate it's reliability. One example is Mitchell and Williams (1993) work that presented an analysis of PLC failure percentage comparing the real industry numbers to manufacturer's information. The result was a $0.025/year$ failure rate in 1993. Also to address PLC failures researchers, as presented by Bang and Bien (1997), designed algorithms and test procedures but the computational power available was not sufficient for accurate results. Recent work in this area turned to output signal analysis for conformance verification (Guignard and Faure, 2014).

One of the possible failure detection using PLC is regarding power transmission. Lallbeeharry et al. (2018) work proposed a sequence for PLCs that can be used to verify the power line communication quality. The correlation and euclidean distance between collected data and the expected values was used to verify if there was a fault. Also, sensors fault can be detected by comparing input signal values to pre-defined measures as proposed in Mohod and Raut (2019) work. Howerever, some of these works are only applied in simulation processes that are interesting but different for real plant scenarios. Machine learning is used in different scenarios considering PLC controlled industries. Artificial Neural Networks (ANN) are a recurrent choice because of the numerical nature of PLC signal data. Jung (2008) work presents an ANN to identify if the PLC signals are similar to what is expected or are causing failures to the processes. Considering that manufacturing processes are modeled by state machines, Ghosh et al. (2020) work presents a automata model together with an Artificial Neural Network (ANN) for failure detection in PLC controlled industrial processes. The main contributions pointed by the authors were: the focus on real applications, the capacity of handling sensors and actuators and the capacity to handle analog signals.

Presented related work showed that this works main goal is important for industry in order to detect and correct faults in PLC-based industrial processes. The only metric used for each work is the fact that the system should be able to correctly identify the fault, so this work is also going to use this metric to evaluate the obtained results.

3. MATERIALS AND METHODS

This section presents the equipment used in this research work together with the description of the designed ETL (Extract, Transform, Load) process.

3.1 Hardware and Software

In this study, the object of analysis was the Programmable Logic Controller (PLC), specifically the Siemens S7-300 model. PLCs are devices responsible for controlling inputs, outputs, and memory blocks in industrial equipment, with each data type receiving a specific address. Sensors connected to the PLC inputs trigger actuators connected to the PLC outputs. In the case of Siemens PLCs, addresses are composed of a set of characters that indicate the type of address and its logical location (Stewart, 2022). For example, in the address 'I1.7', the letter 'I' indicates that the address belongs to an input located in the 7th bit of byte number 1. For this project, all received information was treated as individual binary bits.

Only digital inputs were monitored, since analyzing outputs would be redundant. This is because, in case of a failure in triggering an output, the corresponding sensor connected to the input would not be read, allowing the system to identify the failure. The analyzed data was generated directly by the PLC. The inclusion of analog information may be the subject of a new study.

For data collection, the Python (Müller and Guido, 2016) language was used in conjunction with the open-source *snap7* [1] library for interaction with Siemens PLCs. To receive and execute the data models, was used a Raspberry Pi 4 Model B board with 4 GB of RAM. This choice was based on price, offered configurations, and compatibility with the Python language used in the project. In addition, the mobility of the Raspberry Pi and the possibility of installing a screen for operation and result verification were crucial for the choice. The connection was established using the Ethernet protocol via cable, with the Raspberry Pi being configured to belong to the same network as the PLC control system (Figure 1). Thus, communication between devices was possible, and the connection could be made from the Ethernet ports of the PLC and the Raspberry Pi, without the need for intermediate devices.

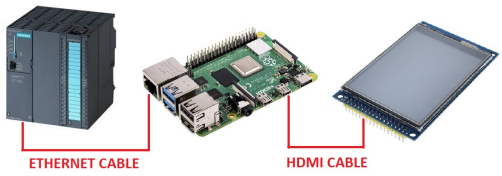

Figure 1. System Assembly Illustration

[1] https://python-snap7.readthedocs.io/en/latest/

3.2 Analyzed equipment

The source of the collected data was an industrial machine controlled by the PLC described above, responsible for a cyclic operation with defined start and end of the cycle. Data from 27 bytes (216 digital inputs) of the machine were considered for the algorithm application.

Some additional information bits were necessary for the system to function, namely:

- Work Complete bit: When activated, indicates that the machine is performing the operation cycle. When deactivated, indicates that the machine is performing the setup cycle.
- Manual bit: When activated, indicates that the operator is manually controlling the machine.
- Alarm bit: When activated, indicates that the machine has stopped working for any reason.

The machine chosen for the study had some programmed failure alarm logic, which was crucial to validate the proposed model and generate data for the machine learning analysis metrics. In case of machine stoppage, it was possible to compare the algorithm results with the PLC failure displayed on the HMI (Human-Machine Interface).

3.3 Data Collection and Processing

To collect the data, the Python Snap7 library was used, which has the `'read_area'` function to read the current states in the outputs of the PLC. In this function, it is possible to specify the type of area to be read (input, output, memory, etc.), the starting address and the size of the word to be read. For example, when it is necessary to read input addresses from 120 to 122, use the "Areas.PE" parameter, specifying that it is an input address. So, you must inform the DB value to be read, this resource was not used, therefore the value 0 was adopted. Next, the value 120 is configured, representing the byte from which the reading will start, and then the value 3, indicating the number of bytes to be read (120, 121 and 122).

```
data = plc.read_area(Areas.PE, 0, 120, 3)
```

This function returns a value in the form of a bytearray structure with the requested addresses. When converted to binary, it is possible to see more clearly how this represents the current activation or deactivation states of each machine bit.

If the following supposed bytearray is used:

```
(b'\x93\xe3\x7a')
```

Value divided in bytes (hex):
Byte 120: 93 | Byte 121: e3 | Byte 122: 7a

Value in binary:
Byte 120: 10010011 | Byte 121: 11100011 | Byte 122: 01111010

Going a little deeper, if byte 120 is decomposed, for example, it is possible to verify the state of each bit's activation of the address.

Bit 0 - I120.0 - Value: 1 (on)
Bit 1 - I120.1 - Value: 1 (on)
Bit 2 - I120.2 - Value: 0 (off)
Bit 3 - I120.3 - Value: 0 (off)
Bit 4 - I120.4 - Value: 1 (on)
Bit 5 - I120.5 - Value: 0 (off)
Bit 6 - I120.6 - Value: 0 (off)
Bit 7 - I120.7 - Value: 1 (on)

After the conversion to binary, the complete value is obtained: 10010011111000011011111010

A function was developed that, upon receiving the bytearray value, would convert it to binary representation and then to hexadecimal. For the supposed bytearray, the following complete value in hexadecimal is:

Hex Value: 93e37a

The data collection algorithm called this function in a loop for a defined area and observed the returned hexadecimal (current state). If the new read state was different from the previous one, it was concluded that there was a movement in the machine (activation or deactivation of some bit). Thus, it was possible to obtain a record of all machine state changes, generating a list of states, that was called a sequence. Every time the bit defined as Work Complete changed from 0 to 1, it was understood that the system started a new working sequence, and when it went from 1 to 0, the system finished the working sequence and started the setup cycle. All the process is described in Figure 2.

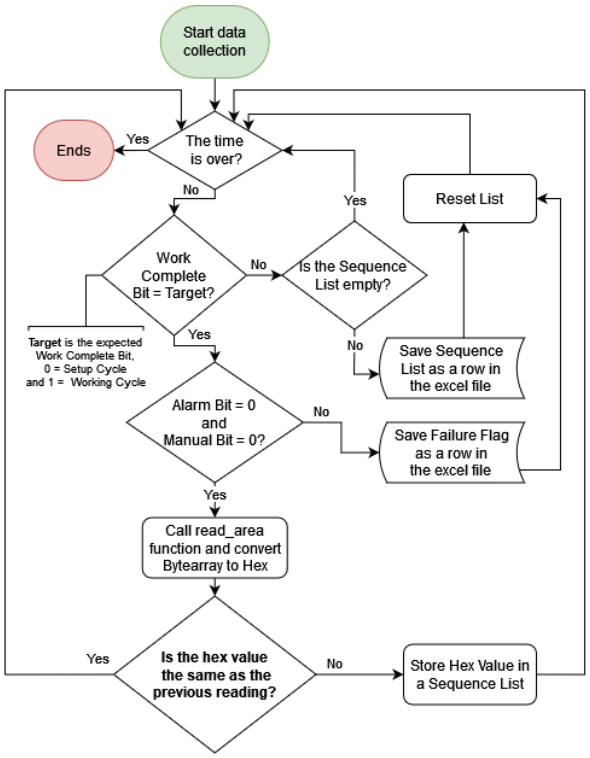

Figure 2. Data Collection steps.

In this way, it was possible to store the sequence executed by the machine with each new cycle. The working and setup cycles were analyzed but saved in different training files. Learning was programmed to pause data collection when changes in the manual and alarm bits were identified, preventing the system from learning as normal the sequences that were not good. In the occurrence of these events, the algorithm recorded in the training file a flag that there was a pause, this information was used in the data processing. Finally, data collection was completed after reaching the maximum time stipulated.

After cleaning the data registered in the XLSX files, an attempt was made to eliminate the sequences that were not

registered from the beginning (they started to be collected in the middle of the sequence) and those that were not collected until the end. For this, the lines where there were records of the pause flag were excluded, in addition to the lines positioned just above and just below this one, since there was an interruption in the reading and return in an unexpected position. The first and last lines of the dataset were also removed. Figure 3 presents the final format of the training file.

	STATE 1	STATE 2	STATE 3	STATE 4	STATE 5	STATE 6	STATE 7	STATE 8	STATE 9
1° Sequence	55	54	50	42	4a	6a			
2° Sequence	aa	a8	a0	a1	81	85	95	55	
3° Sequence	55								
4° Sequence	55	54	50	42	4a	6a			
5° Sequence	aa	a8	a0	a1	81	85	95	15	55
6° Sequence	55								
7° Sequence	55	50	52	42	4a	6a			
8° Sequence	aa	a8	a1	81	85	95	15	55	
9° Sequence	55								
10° Sequence	55	54	50	42	4a	6a			
11° Sequence	aa	a8	a0	a1	81	85	95	15	55

Figure 3. Part of a byte training data in setup cycle

To ensure that the collected data represented cyclic sequences, the entire machine was divided into different analysis regions. These regions are parts of the equipment that perform distinct tasks and cycle at different times than the other regions, for example:

Region 1: Bytes from 0 to 7 -> Operation check bits (Motor Power On, Fan Power On, Door Closed, Oil Level Alarm, etc.)
Region 2: Bytes from 8 to 20 -> Actuator Trigger Sensors (In this region, each byte was analyzed separately to improve the stability of the collected sequences.)
Region 3: Byte 21 -> Inverter check bits
Region 4: Bytes from 22 to 25 -> Bits related to the piece transporters (insertion, removal, and piece transport)
Region 5: Byte 26 -> Interlock check bits.

During this study, information from approximately 7000 cycles of work and preparation was collected, which composed the datasets for each region. The system was able to execute the `read_area` function call and perform all the logic approximately every 0.15s.

3.4 Model Training

In order for the model to be able to analyze operation error in operation sequences, it was necessary to create a way to store information about state changes made during training for a defined region. Therefore, a Finite State Machine (FSM) structure was created that could store more than one sequence start option, as well as more than one ending condition, and also the process state changes.

This structure reads the training dataset line by line and iterates through the columns. Initially, all unique states, initial and final states are identified. With this information, the FSM is initialized. The next step is to iterate through the dataset again, and the algorithm registers each source-destination state pair as a possible transition. Each transition, when added for the first time, is assigned a transition weight value of 1, and whenever the system tries to add the same transition pair again, the weight value is incremented by 1. This way, more common transitions will have a higher weight value in the end, as they are more likely to occur. Additionally, this also allows a single source state to have multiple destination states with different scores. At the end of the process, a structure is obtained as shown in Figure 4:

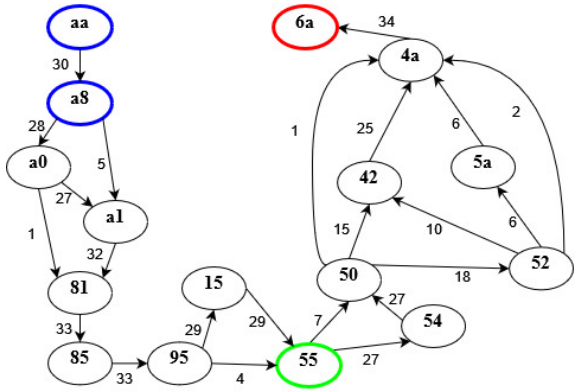

Figure 4. Finite State Machine of 100 lines of training from a byte in setup cycle

The initial states are identified by blue ellipses, the final states are in red, and the states that can be both initial and final are in green. The number displayed next to the arrows represents the transition weight value. Once finished, it is possible to traverse the entire state machine. Using the created methods, it is necessary to start the movement with a possible initial state, and then sequentially indicate the destination states. If the transition is known, the FSM will move and point to a new current state, until a final state is reached.

3.5 Failure Detecting

The fault detection system was developed using an analysis cycle (Figure 5). Upon starting, the algorithm creates an instance of the Finite State Machine and calls the function to read the PLC areas, which return hexadecimal values. The first step is to verify if the machine is performing the desired type of operation (work or setup, according to the Work Complete Bit) and is not in manual operation mode. If both results are positive, the FSM is initialized from the first hexadecimal value read. If the value is within the possible initial states learned during training, the collection continues. If the state is not recognized as a possible start, the fault is identified. In case of successful initialization of the first FSM state, the system calls the read and conversion functions again, making the necessary validations.

Just like during the training period, if the hexadecimal value returned is different from the previous read, a state change is identified. When a new state is found, the algorithm checks if the FSM allows the transition from the previous state to the new state. If so, the analysis proceeds normally, transitioning between states until the end of the operation (change of the Work Complete Bit state). If the FSM does not allow the transition, a fault in the operation sequence is also identified.

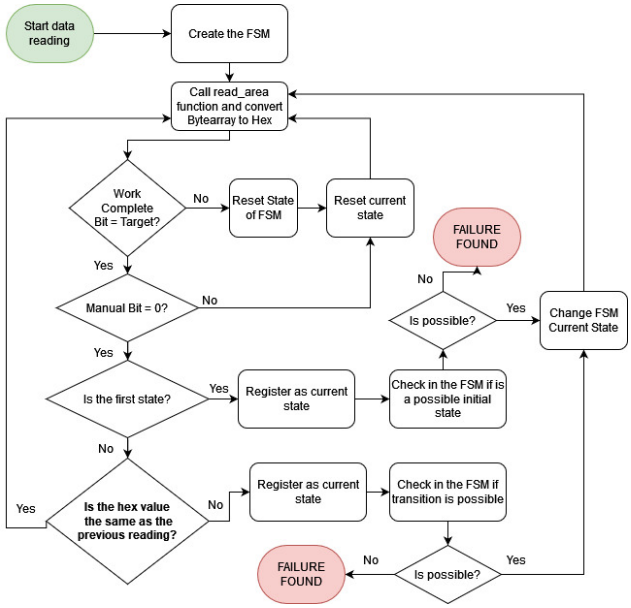

Figure 5. Scanning Fluxogram

3.6 Failure Prediction

Upon detecting a fault in the prior phase, the system starts a process to pinpoint the faulty actuator (Figure 6).

The first step of processing is to obtain, from the last state, the possible transition options and their respective weights. After that, all possible transitions and the state found are converted to binary. At this stage, a score is calculated for each possibility (Figure 6). The state with the highest score is returned as a prediction. The predicted state is the one the system should have found instead of the failure state. With this return, the system compares bit by bit the failure and predicted states, and the bits that show variation between them are identified as actuators in failure.

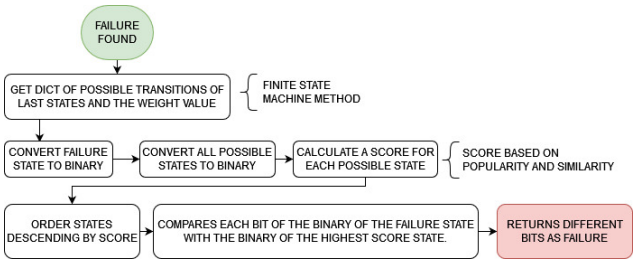

Figure 6. Failure Processing

For each state in the possible transition options, a score is calculated. Prediction computation relies on states popularity and similarity. A popularity score is obtained based on how much more often the transition to this state happens compared to other possibilities, and a similarity score is obtained based on how similar the state is to the failure state found. Finally, a final score is calculated based on these two results. The calculations are shown below.

$$Sum_{popularity} = \sum_{i=0}^{len(\text{next states dict})} \text{Next State's Transition Weight}_{[i]} \quad (1)$$

$$Popularity_{Score} = \frac{\text{Next State's Transition Weight}}{Sum_{popularity}} \quad (2)$$

Popularity score is the percentage value that the transition weight represents within the sum of the weights of all possible transitions. For example, taking state '52', shown in figure 4, the following transition options are obtained:

42 –> Weight = 10 | 4a –> Weight = 2 | 5a –> Weight = 6

Sum Popularity would be equal to the sum of all weights:

$\text{Sum}_{\text{popularity}} = 10 + 2 + 6 = 18.$

While the popularity score for the transition to state '42', for example, would be:

$\text{Popularity}_{\text{Score}} = 10 \div 18 = 0.56.$

$$Sum_{Similarity} = \frac{\text{Number of equal bits in the comparison between Next State and Failure State}}{} \quad (3)$$

$$Similarity_{Score} = \frac{Similarity_{Sum}}{\text{Binary Word Size}} \quad (4)$$

Similarity score considers the binaries of transition options, using the '52' state transitions, binary options are:

42 –> 01000010 | 4a –> 01001010 | 5a –> 01011010

Assuming that the fault state found was '4e' (01001110), the Sum Similarity value for state '42' would be the number of bits that differ between '42' and '4e', being:

42 –> 0 1 0 0 0 0 1 0
4e –> 0 1 0 0 1 1 1 0

The bits in positions 4 and 5 (considering the sequence from 0 to 7) are different between states, so:

$\text{Sum}_{\text{Similarity}} = 2$

After that, to maintain the stability of the calculation in case of larger binary words, the sum value is divided by the binary word length, obtaining that:

$\text{Similarity}_{\text{Score}} = 2/8 = 0.25$

To find the Final Score, the following expression is used, which gives greater weight to the Similarity Score

$$Final_{Score} = (2 * Similarity_{Score}) + Popularity_{Score} \quad (5)$$

Applying for state '42' –> $\text{Final}_{\text{Score}} = (2 * 0.25) + 0.56 = 1.06$

As a way to increase the chances of success and improve the performance of the maintenance team, a classification strategy was created for the predictions. If the failure with classification 1 is not correct in the field verification, it is possible to act on failures with classification 2, 3, or 4, respectively in order of importance. The classification was made according to the truth table below:

Table 1. Failure Classification Truth Table

Alarm Bit	Highest Score	Class
0	0	Class 4
0	1	Class 3
1	0	Class 2
1	1	Class 1

4. RESULTS AND ANALISYS

The results obtained from the application of the machine learning algorithm were collected over approximately 15 days of continuous execution with the analyzed machine in constant operation. The model was able to correctly classify (accuracy) over 90% of the sequence of operation failures of machines controlled by PLCs, pointing out the address that did not execute the operation. An example of the system's output can be seen below:

Table 2. Failure Result Example

Address	Class	Name	Datetime
I5.0	1	Oil Pump Sensor Low	2023-04-14 12:26:12
I12.2	3	Clamp 2 Advance	2023-04-13 18:42:27
I17.3	2	Table Rotate Return	2023-04-12 08:34:14
I13.7	4	Clamp 4 Return	2023-04-06 14:24:47

The confusion matrix below represents the distribution of results found in the model. The following were considered:

- True Positive: System Failure equal Real Failure
- False Positive: System Failure different from Real Failure or Non-existent
- False Negative: System did not detect failure, but the machine stopped.
- True Negative: System did not detect class 1 failure and there was no machine stoppage.

Table 3. Confusion Matrix

		Predicted	
		Predicted Failure	No Failure
Real	Real Failure	21	3
	No Failure	3	51

Upon analyzing the results, it is possible to verify that the algorithm obtained a significant precision in real time classification and identification of failures, reaching 87% considering this metric. The False Negative rate, which would leave the maintenance team without support in the intervention, was low, reaching about 8%. However, the system had difficulties in functioning when returning from manual operation of the equipment, often getting lost in the FSM and generating False Positive results. The data collection speed of the algorithm was also an impacting factor. With 24/7 execution, the hardware began to overheat and reduce the execution speed, generating instability in the sequences read, resulting in incorrect classifications.

Conversely, the tool proved highly sensitive, as slight variations in actuator activation time generated sequence variations insufficient to stop the machine by alarm. The model identified the variations and classified them as class 3 and 4 faults. It was observed that excessive increase in the number of faults in these classes in the same equipment and in a short period of time resulted in a type 1 failure in the future at the same address. Therefore, it was determined that the system can also have characteristics that contribute to predictive maintenance.

Finally, the work also showed to be able to deal with two of the main contributions pointed out by the authors of related works, being able to be effective in a real application and working with equipment at the level of sensors and actuators. For industrial applications, the results obtained indicate that the developed machine learning model can be a tool to help reduce maintenance time and machine downtime in production lines. The system can be easily applied to other machines controlled by PLCs, being a scalable and effective option for error detection.

5. CONCLUSION

This study produced a machine learning algorithm to detect faults in the operation sequence of machines controlled by PLCs, pinpointing faulty actuators. Using a training set of around 7000 sequences, the system attained over 90% accuracy in fault classification, with just 8% False Negatives. This suggests the model performed well in real-world scenarios, offering valuable predictive insights for maintenance planning and minimizing downtime in PLC-controlled machines.

Yet, it is evident that the model shows limitations on hardware overheating, failures in the return from manual state, and false negative occurrences. These aspects can be analyzed and addressed in future research to enhance classification accuracy. Potential areas for improvement include treating only the element that caused the alteration in the state machine instead of the state as a whole, refining classification calculations, or adopting a neural network approach.

REFERENCES

Azizi, A. and Barenji, R. (2022). *Industry 4.0: Tec., App., and Challenges*. Emerging Trends in Mechatronics. Springer Nature Singapore.

Bang, W.C. and Bien, Z. (1997). Design of an algorithm for plc fault diagnosis system. *IFAC Proceedings Volumes*, 30(13), 141–146.

Ghosh, A., Wang, G.N., and Lee, J. (2020). A novel automata and neural network based fault diagnosis system for plc controlled manufacturing systems. *Computers and Industrial Engineering*, 139, 106188.

Guignard, A. and Faure, J.M. (2014). A conformance relation for model-based testing of plc. *IFAC Proceedings Volumes*, 47(2), 412–419. 12th IFAC International Workshop on Discrete Event Systems (2014).

Jung, In-Sung, e.A. (2008). Plc control logic error monitoring and prediction using neural network. In *2008 ICNC*, volume 2, 484–488.

Lallbeeharry, N., Mazari, R., Dégardin, V., and Trebosc, C. (2018). Plc applied to fault detection on in-vehicle power line. In *2018 IEEE (ISPLC)*, 1–5. doi:10.1109/ISPLC.2018.8360233.

Lamb, F. (2013). *Industrial Automation: Hands On.* McGraw Hill LLC.

Mitchell, C.M. and Williams, K. (1993). Failure experience of programmable logic controllers used in emergency shutdown systems. *Reliability Engineering and System Safety*, 39(3), 329–331.

Mohod, S. and Raut, A. (2019). Plc scada based fault detection system for steam boiler in remote plant. In *2019 (ICICICT)*, volume 1, 1007–1010.

Müller, A. and Guido, S. (2016). *Introduction to Machine Learning with Python.* O'Reilly Media.

Petruzella, F.D. (2016). *Programmable Logic Controllers.* McGraw-Hill Education, 5th edition.

Stewart, R.G. (2022). *Siemens PLC programming for beginners.* Independently Published, 1th edition.