Learning and Nonlinear Models – Journal of the Brazilian Society on Computational Intelligence (SBIC), Vol. 21, Iss. 1, pp. 90-109, 2023
© Brazilian Society on Computational Intelligence

DEVELOPMENT OF A LOW-COST RELAY PROTOTYPE FOR REAL-TIME POWER PROTECTION FUNCTIONS

Pablo Rodrigues Lopes
Rui Bertho Junior
Federal University of Piauí
Department of Electrical Engineering
e-mails: prlopes4@yahoo.com.br and rui.bertho@ufpi.edu.br
Campus Petrônio Portella, Ininga, Center of Technology
CEP 64049-550, Teresina, PI, Brazil

Abstract- This paper presents a Raspberry Pi 3B+ based low-cost universal relay able to run power protection functions in real-time. The configurations necessary for this single-board computer to be able to provide real-time response are shown, as for latency tests to verify its response time. An experimental circuit was built to send three-phase fault signals from a Relay tester to the low-cost relay, in order to evaluate its response time to clear faults in comparison to a commercial relay. A neural networks algorithm was developed and executed in real-time by the proposed low-cost relay, which is able to differentiate three-phase faults from transient signals created from large load variations. The results show that the low-cost relay is capable of running simple and complex protection functions within a pre-defined runtime and acceptable precision, compared to a commercial protection relay. However, the sampling frequency the low-cost relay is able to handle is limited. The results have also shown that the low-cost relay meet the requirements for a soft real-time system, which is not ideal for practical power protection systems that require hard real-time systems.

Keywords- Neural network, protection systems, raspberry pi 3, real-time systems, relays.

1 Introduction

One of the main roles of power protection systems is to ensure that electrical power is delivered in a reliable and safe manner to its users. However, the continuous changes in power grids worldwide, like distributed generation and large usage of non-linear loads, pushes the power grid towards its stability and safety limits (Kuffel et al., 2016). Consequently, new power protection philosophies and devices have been developed over the decades to contemplate those changes and many others.

Digital protection relays have grown in computational power and complexity of its systems, which makes them able to integrate multiple protection functions and execute more tasks under fault conditions (Wang and Dinavahi, 2016). Nowadays, commercial numerical relays are largely configurable, from parameterization to evaluation of the response of those devices in simulated environments. However, the implementation of novel power protection algorithms is not always possible in those devices, due to the proprietary nature of its hardware and software. Hence, researchers resort to create their own hardware and software setups to test their algorithms in real-time, in order to achieve results closer to the reality of power systems. Literature review shows that the most common platform utilized by researchers for real-time tests of their power protection algorithms are: Field Programmable Gate Arrays (FPGA's), adapted desktop computers, and microcontrollers (Wang and Dinavahi, 2016; Monaro et al., 2012; Martins, 2017).

FPGA's have many desired characteristics when it comes to digital power protection, like its configurable logical blocks, natural parallel hardwired architecture, low computational latency while interfacing with peripherals, and high processing power and speed. These attributes are explored by and Dinavahi (2016), who proposed a multifunctional protective relay real-time system in a Virtex-7 Xilinx® device, and Mitra and Chattopadhyay (2019), who developed an adaptive overcurrent protection function for distributed generation systems using an Artificial Neural Network (ANN), embedded in a Cyclone IV Intel® device. Both works show the capabilities of this class of devices for power protection systems, by delivering a trip time close to the characteristic equation of those functions, with little delay. However, the purchase of those devices is usually onerous; and a deep understanding of a FPGA's architecture and programming skills is required to explore its capabilities is required from the user of these boards.

This paper considers adapted desktop computers as industrial computational boards that were adapted as digital protection relays. They have advantageous features in relation to FPGA's, since their fixed logical blocks allow users to use high level programming languages, while maintaining computational power. Monaro et al. (2012) and Pellini et al. (2013) are some examples of successful applications of those devices in power protection research. Monaro et al. (2012) developed an integrated hardware and software system using a PC/104 Consortium computer, able to execute traditional power protection functions in real-time, as for a group of power protection functions for synchronous generators, based on fuzzy logic (Monaro, 2013). As for Pellini et al. (2013), they used an IBM industrial computer as an automated distribution feeder recloser, which was adapted by Dantas et al. (2018) to execute their time-domain differential protection algorithm. In both works, the adapted desktop computers utilized delivered faster trip time at low sampling rates. Nevertheless, those platforms are still expensive to be acquired.

Lastly, microcontrollers are commonly used for researchers in power protection applications, due to the specificity those devices have while executing a required task. In addition, compared to FPGA's and adapted desktop computers, they are much more affordable. One example of their application in power protection is in Joy et al. (2022), who created a feeder protection solution based on an Arduino nano microcontroller, contemplating multiple protection functions, like under and overvoltage protection, earth fault protection, among others. Another application is from Martins (2017), who used a development board MSP432P401R from Texas Instruments to run an adaptive recloser protection algorithm for power transmission lines. Both works show that microcontrollers can extract relevant information from the input signals and adequately respond to electrical fault events. These platforms are the most accessible among the class of devices presented. Although the codification required to use microcontrollers is low level, which makes the implementation of algorithms more challenging, solutions like TensorFlow (2021) allows user to easily embed machine learning models into a variety of microcontrollers.

In short, an ideal platform for power protection applications would have the following features: affordability, the ability to support high-level programming, and the capability to run real-time power protection algorithms. Hence, it can be applied to validate power protection algorithms within real-time constraints. One type of platform that may have those features are single-board computers. Since little is known about their application in power protection systems, this paper investigates the possibility of a particular single-board computer, a Raspberry Pi 3B+, to be utilized as a dedicated real-time system for power protection applications, from simple power protection functions, like an overcurrent function to detect symmetrical three-phase faults, to more complex power protection algorithms, like the ones based on machine learning techniques (Mitra and Chattopadhyay, 2019; Monaro, 2013). The main goal of this application is to provide a low-cost solution for other researchers to quickly implement and test their own novel power protection algorithms in hardware.

The main concepts regarding real-time systems, as for evaluating the Raspberry Pi 3B+ performance as a real-time platform, are presented in Section 2. Then, Section 3 describes the designed setup (hardware and software) of the proposed low-cost relay. Section 4 presents the experimental steps taken to test the proposed platform; by modelling a distribution feeder to obtain steady-state, three-phase fault and transient signals; as for the neural network algorithm developed to differentiate those three events. Finally, Section 5 focuses on the experimental results of the proposed platform; testing its trip operating time against a commercial relay, while running a definite time overcurrent protection. That section also shows the performance of the low-cost relay while running the developed neural network algorithm.

2 Raspberry Pi and Real-time Systems

The Raspberry Pi's are a series of educational platforms focused on teaching computing and digital marketing skills (Raspberry Pi Foundation, 2021). They have similar physical dimensions and features of an Arduino microcontroller, but have higher computational resources and larger memory, which makes them able to run complex Operational Systems (OS). Along with its inexpensive cost, these platforms became popular among various academic fields, especially in research regarding Internet of Things (IoT) (Aqeel, 2018). In addition, there are numerous online communities dedicated to distributing applications for these series of single-board computers. Figure 1 illustrates the Raspberry Pi 3B+.

The main features this singles-board computer has that will be relevant to the proposed low-cost universal relay are (Raspberry Pi Foundation, 2021):

- **CPU** – Quad-core processor ARMv8; 64 bits; 1,4 Ghz (significant computational power);
- **GPIO** – 40-pin header able to interface other peripheral components;
- **Operational System** – Raspberry Pi OS (previously called Raspbian), which is a Linux distribution. The instructions to install this OS in a Raspberry Pi are provided in the Raspberry Pi Foundation website (Raspberry Pi Foundation, 2021).

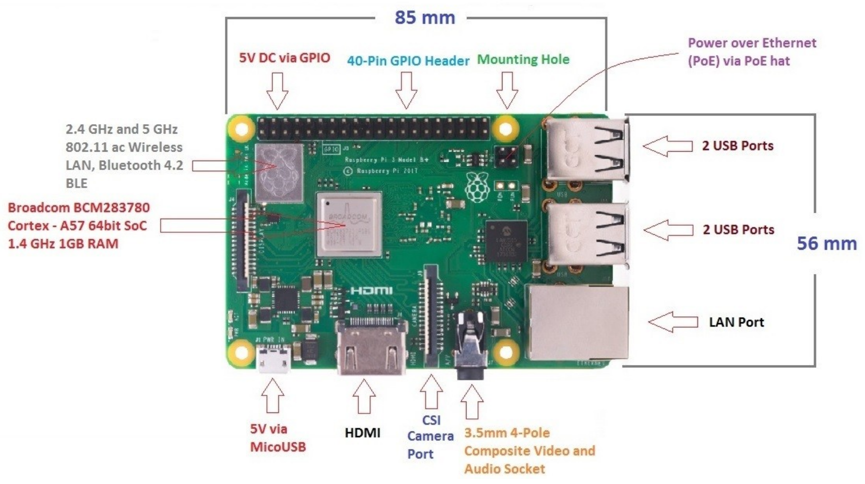

85 mm
5V DC via GPIO
40-Pin GPIO Header
Mounting Hole
Power over Ethernet (PoE) via PoE hat
2.4 GHz and 5 GHz 802.11 ac Wireless LAN, Bluetooth 4.2 BLE
Broadcom BCM283780 Cortex - A57 64bit SoC 1.4 GHz 1GB RAM
2 USB Ports
2 USB Ports
56 mm
LAN Port
5V via MicoUSB
HDMI
CSI Camera Port
3.5mm 4-Pole Composite Video and Audio Socket

Figure 1: Hardware details of the Raspberry Pi 3B+ module (Aqeel, 2018).

2.1 Definitions Regarding Real-time Systems

The meaning of the term "real-time" may vary according to the context it's used. Regarding this paper, Real-Time Operational Systems (RTOS) are computational systems that run tasks in a precise execution time. This means that they are able to receive information or data, process them, and return an answer or action within a determined time (Buttazzo, 2013).

The main concepts associated with real-time systems are (Buttazzo, 2013):

- **Determinism** – the system must always respond to a specific event (system input) in the same manner;
- **Predictability** – all tasks of a system must happen in a specific time frame;
- **Performance** – real-time systems must have low latency (time spent between sending and receiving an information), low jitter (deviation from the usual response time) and able to transfer data in a high rate.

Real-time systems are also classified according to their response case a time-determined task miss its deadline (Buttazzo, 2013):

- **Hard real-time** – tasks must be handled in a pre-determined time. Any delay in them will have catastrophic consequences for the system;
- **Firm real-time** – tasks must be handled in a pre-determined time. Any delay in them will not damage the system affected by hem, though its results are no longer useful;
- **Soft real-time** – tasks must ideally be handled in a pre-determined time, but its results are still useful even when their deadline is missed. They also don't harm the system affected by them.

Considering the concepts presented, a power protection system can be considered a hard real-time system, since they must detect electrical faults and operate peripherals with minimum delays. One of those tasks being executed too early or too late (like sending a trip signal at the wrong time) may result in the failure of the fault detection; and might cause physical damage to the electrical infrastructure of a power grid or even cause injury to people.

The previously mentioned characteristics for real-time systems are not met by General-Propose Operational Systems (GPOS), like Windows and Linux operating systems, since they are focused on overall performance, rather than run specific tasks in a time-constrained environment (Buttazzo, 2013). Hence, commercial IED's use operational systems developed specifically for power protection applications, in order to meet the critical response time needed to clear electrical faults.

As mentioned in the introduction of this paper, researchers that aim to test their power protection solutions in real-time develop their own hardware and software for their end. Due to Linux OS popularity among users, a few approaches were developed in order to make the Linux OS more suitable for real-time computing. They involve altering the *kernel* of Linux OS, responsible to manage all applications within the OS. According to Johansson (2018), those configurations are:

- **Single-kernel configuration** - it modifies the Linux kernel to make its performance more predictable;
- **Dual-kernel configuration** - it creates a virtual secondary kernel (also known as hypervisor), which handles processor time, memory space and more. This configuration prioritizes real-time tasks over common Linux processes.

Figure 2 visually describes those approaches.

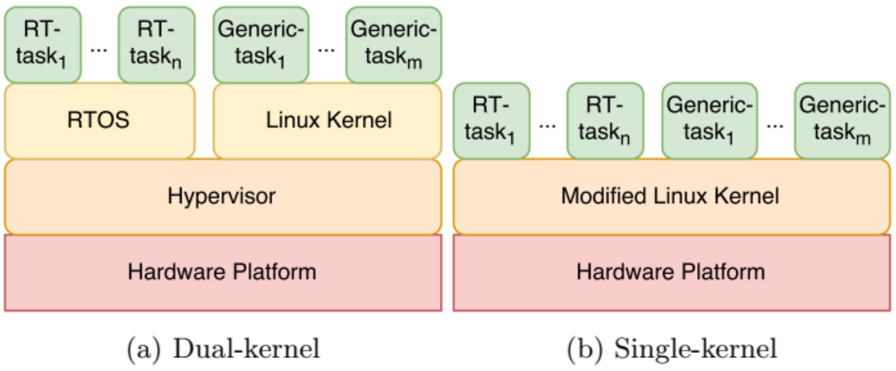


Figure 2: Single-kernel and dual-kernel approaches for real-time computing in Linux (Johansson, 2018).

2.2 Configuring the Raspberry Pi as a real-time system

According to Molloy (2016), the Raspberry Pi was not originally created for real-time computing. However, there are many free online patches that modify the *kernel* from the Raspberry Pi OS, in order to give it real-time properties. The most commonly used patches to that end are:

- **Preempt RT** – it gives the Raspberry Pi OS kernel preemptive features (ability to interrupt tasks for others with higher priority). It resembles single-kernel approach (Mckenney, 2015);
- **Xenomai** - Co-kernel that is installed in the Raspberry Pi OS, which handles the higher priority tasks, leaving the lower priority tasks for the non-preemptive kernel. It resembles dual-kernel approach (Gerum, 2021);

Literature review in this subject have shown that the *Xenomai* patch has a better real-time performance than the *Preempt RT* patch (Chalas, 2015; Johanson, 2018), while running in a Raspberry Pi. However, this paper will use the *Preempt RT* patch to develop the proposed low-cost relay, since this patch only requires from the user (once installed in the Raspberry Pi OS) knowledge regarding priority handling commands while coding an algorithm. The *Xenomai* patch has an extensive API, which would increase the development time of the proposed platform. For this paper, the Preempt RT patch utilized was the one provided by Tam (2018), which also provided instructions on how to install it in a Raspberry Pi.

Since the *Preempt RT* patch uses the single-kernel approach for real-time systems, the *kernel* from the Raspberry Pi OS will still handle real-time tasks at the same level than ordinary tasks, although with higher priority. Thus, an error in a real-time task may result in segmentation fault (when a program tries to read or write the wrong memory location). To diminish the frequency of this error to happen in the low-cost relay, this paper will also implement the solution provided by Roberts (2018): dedicating one of the Raspberry Pi cores for the real-time tasks. Thus, not only the real-time tasks will be better handle by a dedicated core, but the Raspberry Pi 3B+ can achieve latencies less than 3 µs with very low variance (Roberts, 2018). The solution provided by Roberts (2018) was the base for the real-time library that the developed relay uses, coded in the C/C++ programming language, allowing register access to the Raspberry Pi, and optimizing the priority handling of the real-time tasks.

2.3 Real-time Performance on Raspberry Pi 3B+

Chalas (2015) and Johanson (2018) have shown the real-time capabilities of a Raspberry Pi 2B and 3B, respectively. They concluded that, although the *kernel* modifications have given those single-board computers low latency and high performance while running real-time tasks, they did not achieve hard real-time characteristics (the tasks the authors put the Raspberry Pi through need it a precision time of 10 µs, which was not met). Since this paper combines the single-kernel approach (*Preempt*

RT patch) with the one core approach from Roberts (2018), it's necessary to evaluate the Raspberry Pi 3B+ performance as a real-time system, by measuring its latency in hardware and in software. The following tests were made in order to provide the required response.

- **Latency test in software**: the Raspberry Pi latency was measured after random delays of 1-1100 µs were inserted in the real-time library. For this test, the chosen task is empty, in order to access the inherent delays of the library itself. The latency value is the difference between the actual delay of the task and the requested delay (randomly generated). This test contemplates short delays (up to 50 µs) and long delays (up to 1100 µs);

- **Latency test in hardware**: two GPIO pins from the Raspberry Pi 3B+ were electrically connected. This task was run in a separate thread (another core from Raspberry Pi OS), creating rising edges at specific times in the output pin while the input pin continuously reads the digital values. The latency (difference between those two moments) is measured in the main thread of the real-time library. For this test, the GPIO pin 12 (configured as the output pin to create the rise edges) and pin 13 (configured as the input pin to read the digital values) were connected through a female-female jumper cable, as shown in Figure 3.

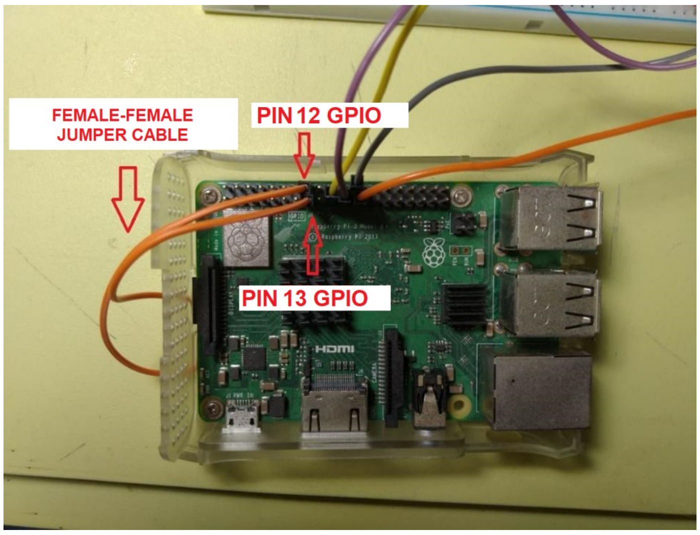


Figure 3: Electrical connection to run the latency test in hardware for the Raspberry Pi 3B+

Those two tests relate to events that can happen in the simplest power protection system, like random delays originated from communication between devices and poor electrical connection. Delays originating from those issues may cause a power protection system to fail, like a delay in a trip signal to clear a fault, or instability on a power system due to delays in disconnecting faulted lines (Wei and Chen, 2010).

Table 1 shows the results from the latency tests made in the Raspberry Pi 3B+, after a four-hour run of the real-time library, while taking numerous latency measurements each minute. The Raspberry Pi 3B+ has an internal counter, that increments at a 1 MHz frequency, which measured the latencies with a microsecond resolution.

Table 1: Latency test results for the Raspberry Pi 3 B+

Tests	**Latency (µs)**			
	Mean	*Minimum*	*Maximum*	*Variance*
Short delays (up to 50 µs)	1.03	1.0	41.0	1.03
Long delays (up to 1000 µs)	1.02	1.0	42.0	1.02
Hardware (GPIO)	0.8	0	32.0	0.8
Number of samples in four hours: 16,364,428.00				

The latency tests show that using the *Preempt RT* patch while dedicating one core from the Raspberry Pi 3B+ for real-time tasks, significantly improves the precision of this single-board computer. For all tests, the measured latency is around 1 µs, with a precision above 99%, since the average latency and variance for each test are very close. The minimum latency value for the hardware test is zero due to precision of the counter used, which means that latency values lower than 1 µs could not be measured in this test.

Although those tests show that the Raspberry Pi 3B+ is capable of real-time computing, the algorithm ran during testing was the real-time library itself. Thus, it's necessary to evaluate if this single-board computer is capable of maintaining real-time response while running complex tasks, like a power protection algorithm. Only after running a power protection function, the type of real-time system the Raspberry Pi 3B+ can handle will be defined (hard, firm or soft real-time).

3 Hardware and software design of the low-cost universal relay

The overall design of the proposed low-cost universal relay is presented as a block diagram in Figure 4. It consists in the following elements:

- **Signal Conditioning Module** – responsible to adjust the voltage and current AC signals from a power system to voltage levels adequate for the A/D Converter Module (0 – 5 V range);
- **A/D Converter Module** – converts the conditioned analog signals (voltage and current signals) to discrete signals, which feed information to the Raspberry Pi 3B+;
- **Power Protection Library** – collection of functions programmed within the Raspberry Pi 3B+, responsible to sample the input signals (sampling process), calculate its main electrical parameters (measuring process) and run a power protection algorithm programmed by the user. Once a fault is detected, a trip signal is sent by the Raspberry Pi.

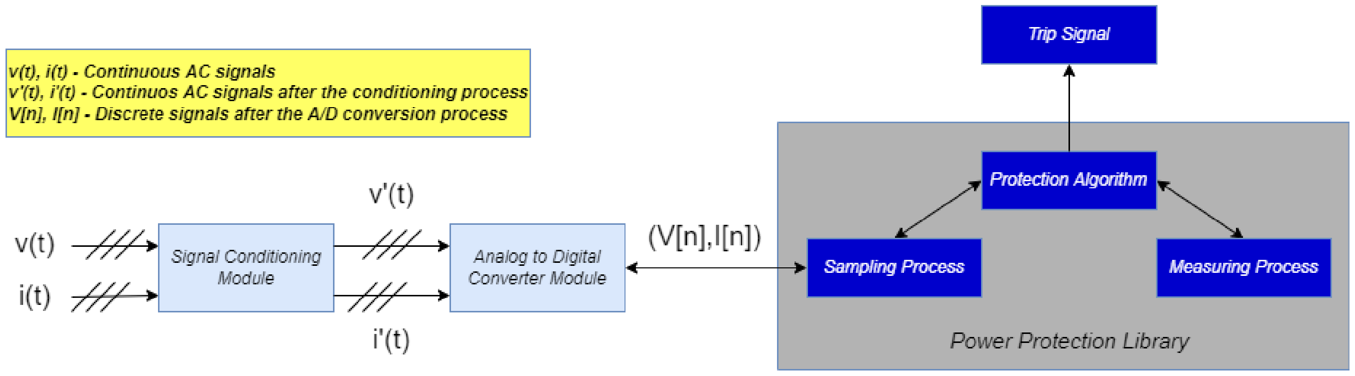

v(t), i(t) - Continuous AC signals
v'(t), i'(t) - Continuos AC signals after the conditioning process
V[n], I[n] - Discrete signals after the A/D conversion process
v(t)
i(t)
Signal Conditioning Module
v'(t)
i'(t)
Analog to Digital Converter Module
(V[n],I[n])
Trip Signal
Protection Algorithm
Sampling Process
Measuring Process
Power Protection Library

Figure 4: Block diagram of the proposed low-cost universal relay

The following subsections will detail the hardware components utilized for the Signal Conditioning and A/D Converter Modules. A detail diagram of the proposed low-cost relay is presented in Figure 13 from Appendix A. In the same section, the cost of all the hardware components utilized to build the low-cost relay are presented in Table 4. As for the Power Protection Library, instructions will be presented on how the user of this proposed low-cost relay can utilize it.

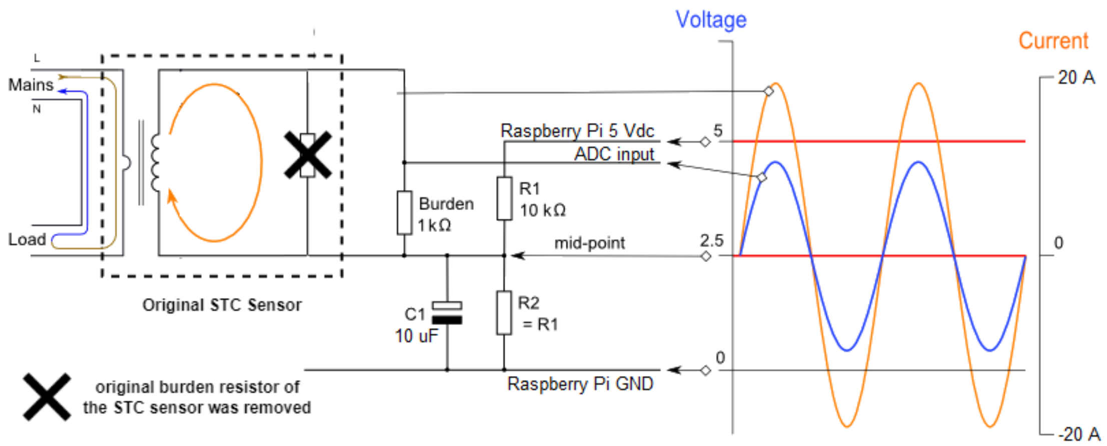
3.1 Signal Conditioning Module

This module consists of three current and three voltage sensors, which receive analog current and voltage signals, respectively, from a power source, and adjust them to lower voltage values (within the 0-5 V range), in order to meet the input requirements of the A/D Converter Module. The chosen sensors for this setup are as follows:

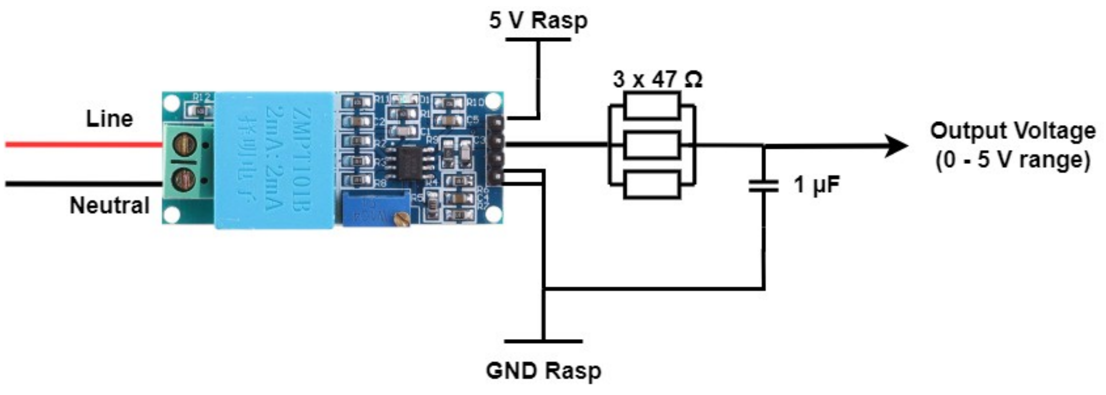
- **SCT – 013 – 20 non-invasive current sensor** – input AC current signal range (0 – 20 A); output AC voltage signal range (0 – 1 V); Sensor turns ratio (2000:1) (YHDC, n.d.);
- **ZMPT101B voltage sensor** – input AC voltage signal range (0 – 250 Vac); output AC voltage signal range (0 – 5 V) (ETC, n.d.).

The SCT current sensor originally gives a voltage output within the 0 – 1 V range, rather than the desired 0 – 5 V range. Thus, the circuitry of this current sensor was adapted to provide the desired voltage range, as shown in Figure 5-a. The internal burden resistor of the SCT sensor was removed, in order to directly obtain the current signal from the sensor. The conditioning circuit utilized for the current signals is based on the design provided by Learn Open Energy Monitor (n.d.).

As for the voltage sensors, a simple low-pass filter, with a cut frequency of around 10 kHz, was connected to their output as an anti-aliasing filter, while mitigating possible noises from the source signals, as shown in Figure 5-b.



(a)



(b)

Figure 5: Conditioning circuits for the (a) AC input currents and (b) AC input voltages

3.2 A/D Converter Module

To feed the Raspberry Pi discrete voltage and current signals, an A/D converter is necessary. For this purpose, the MCP3008 A/D converter, manufactured by Microchip, was selected, due to its high sampling rate (up to 200 ksps) and a built-in SPI serial interface, which makes the communication with the Raspberry Pi synchronized, and gives more control of the sampling process. This converter has eight analog input ports and uses a successive approximation approach to provide a 10-bit resolution for the sampled data (Microchip, 2008). From those eight input ports, six were connected to the output wires of the conditioning circuits, as shown in Figure 6. Since the Raspberry Pi 3B+ works at a 3.3 V logical level, while the A/D converter operates at 5.0 V logical level, a bi-directional logical converter was put between those devices to make their communication possible. Figure 6 also shows the ports from the Raspberry Pi 3B+ responsible for the SPI interface. Lastly, the GPIO pin 21 from the Raspberry Pi 3B+ was programmed to send the trip signal once an electrical fault is detected.

3.3 Power Protection Library

The Power Protection Library of functions developed for the low-cost relay is responsible for sampling the input voltage and current signals, run the programmed power protection functions, make the necessary calculations, like mean and RMS values of the input signals, send the trip signal when an electrical fault is detected, and save the measured signals into an oscillography file (*.csv* extension), contemplating data before and after fault conditions. Aside from the oscillography function, all of the mentioned processes are scheduled by the real-time library, in order to ensure real-time precision of the power protection functions.

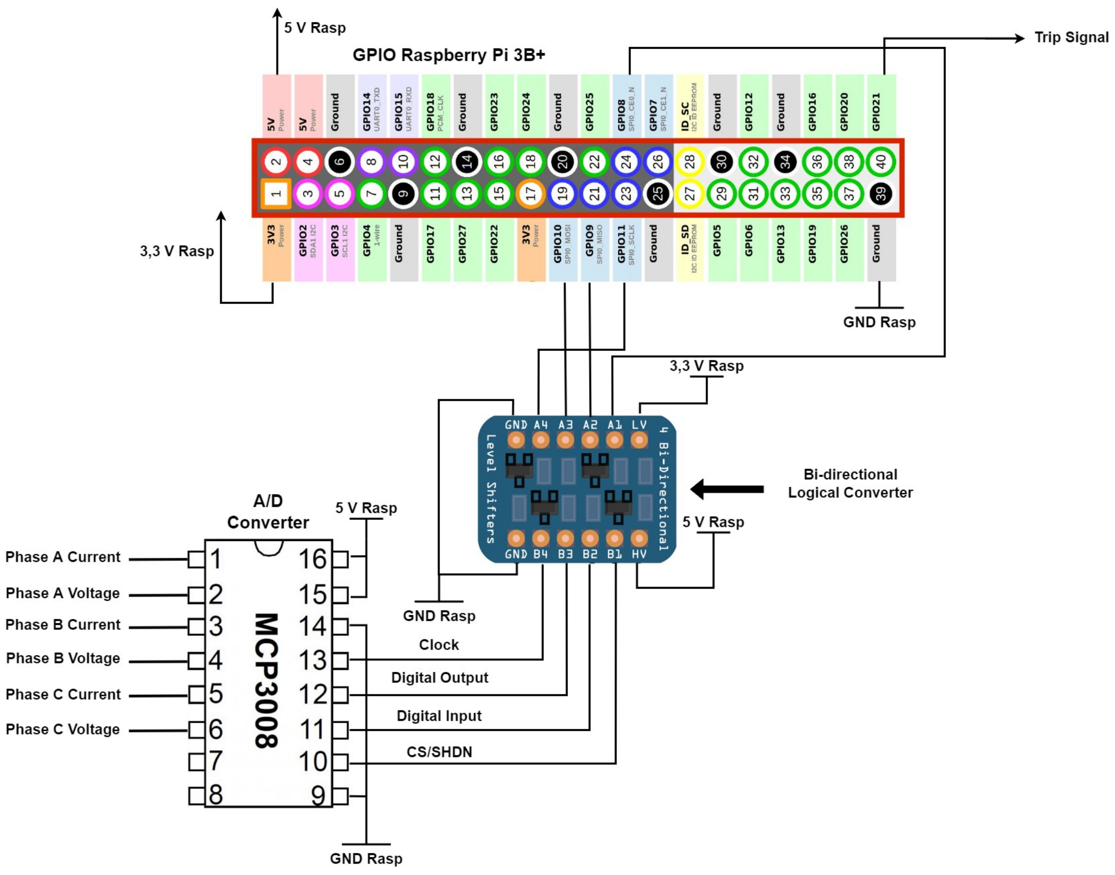


Figure 6: Connection between the A/D converter and the Raspberry Pi 3B+

As mentioned in Subsection 2.2 of this paper, the real-time library handles real-time tasks and ordinary tasks at the same level. Thus, the proposed low-cost relay cannot run power protection functions and save the data signals in a file simultaneously, since it may cause segmentation fault error. The user of this platform must choose whether the low-cost relay will save an oscillography file or run a power protection function.

This Power Protection Library was based on the solution provided by Ryan (2019). Like the real-time library, the power protection library was written in C/C++ programming language. Four input parameters are required to initialize the low-cost relay. They must be declared within the Raspberry Pi's command prompt environment, as show in Figure 7.

```
$ cd ~/projects {or where you keep the power protection library files}
$ cd VoltageCatcher/src
$ sudo vc -s 1 -f 2000 -c 0,1,2,3,4,5 -o /home/pi/VoltageCatcher/Samples.csv
```

Figure 7: Command prompt commands to initialize the low-cost relay

The input parameters must be declared by the user of the low-cost relay utilizing the following codification:

- sudo vc – administrative access to the Raspberry Pi, in order to run the Voltage Catcher program (this name represents the power protection library);

- (-s) is an integer number, representing the amount of AC signal cycles, to generate a data window. Case the number equals to 1, the low-cost relay will run the power protection algorithm, updating the data window until the trip signal is sent. For values above 1, the low-cost relay will save the sampled data into an oscillography file;

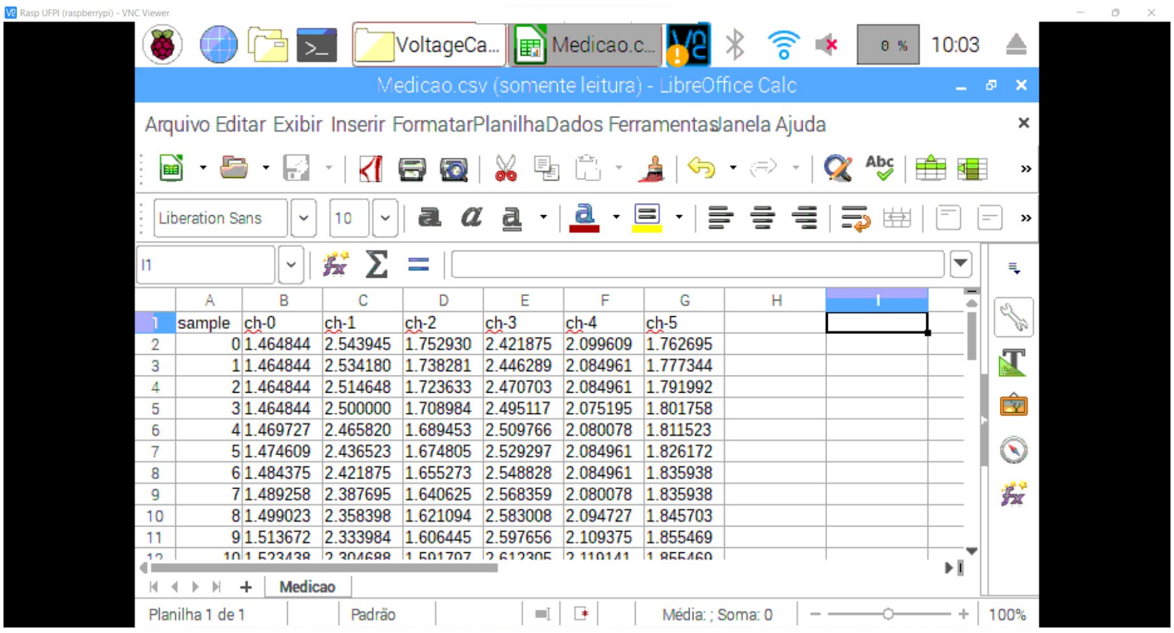
- (-f) is the sampling frequency, represented by an integer number in Hz. Once the value is inserted by the user, an equivalent SPI clock speed is calculated by the power protection library and sent to the A/D converter in order to start the sampling process;

- (-c) is number of input signals, up to eight. The user can choose which channels they want to use;

- (-o) indicates the path directory (within the Raspberry Pi environment) where the oscillography file will be saved, case this functionality is chosen by the user.

In the example shown in Figure 7, the low-cost relay will create a data window at the sampling frequency of 2 kHz (for a 60 Hz input signal, each data window for each voltage and current signal will have around 33 samples). After the channels 0 to 5 from the A/D converter (ports 1 to 6 from the MCP3008 chip) are sampled, the data window is then sent to the power protection algorithm, which will call the measurement process to calculate the required electrical parameters from the data window. The low-cost relay will evaluate if the data sampled represents a steady-state signal or an electrical fault, according to the power protection function designed by the user.

After an electrical fault is detected, the low-cost relay will send a trip signal and cease activity, until the user start-up the power protection library again. Case the user opts to save the sampled data in an oscillography file, they must use a value for *-s* higher than 1. Then, the *Samples.csv* file will be saved under the *VoltageCatcher* folder within the Raspberry Pi. The saved file will contain all samples for each channel selected by the user, as shown in Figure 8. The functionalities of the Power Protection Library are presented as a flowchart in Figure 9.


Rasp UFPI (raspberrypi) - VNC Viewer
VoltageCa...
Medicao.c...
0 %
10:03
Medicao.csv (somente leitura) - LibreOffice Calc
Arquivo Editar Exibir Inserir Formatar Planilha Dados Ferramentas Janela Ajuda
Liberation Sans
10
I1
Medicao
Planilha 1 de 1
Padrão
Média: ; Soma: 0
100%

	A	B	C	D	E	F	G
1	sample	ch-0	ch-1	ch-2	ch-3	ch-4	ch-5
2	0	1.464844	2.543945	1.752930	2.421875	2.099609	1.762695
3	1	1.464844	2.534180	1.738281	2.446289	2.084961	1.777344
4	2	1.464844	2.514648	1.723633	2.470703	2.084961	1.791992
5	3	1.464844	2.500000	1.708984	2.495117	2.075195	1.801758
6	4	1.469727	2.465820	1.689453	2.509766	2.080078	1.811523
7	5	1.474609	2.436523	1.674805	2.529297	2.084961	1.826172
8	6	1.484375	2.421875	1.655273	2.548828	2.084961	1.835938
9	7	1.489258	2.387695	1.640625	2.568359	2.080078	1.835938
10	8	1.499023	2.358398	1.621094	2.583008	2.094727	1.845703
11	9	1.513672	2.333984	1.606445	2.597656	2.109375	1.855469
12	10	1.523438	2.304688	1.591797	2.612305	2.119141	1.855469

Figure 8: Example of the samples file created the power protection library

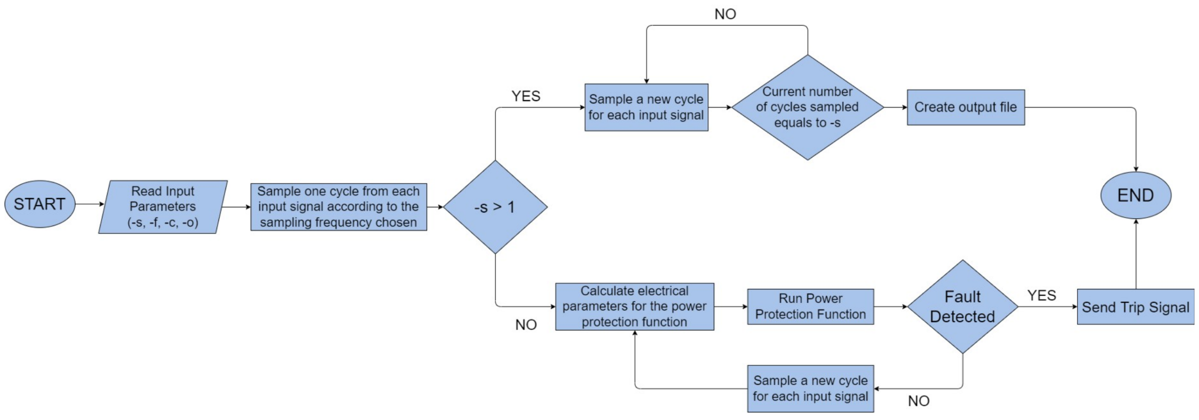

START
Read Input Parameters (-s, -f, -c, -o)
Sample one cycle from each input signal according to the sampling frequency chosen
-s > 1
YES
Sample a new cycle for each input signal
Current number of cycles sampled equals to -s
NO
Create output file
END
NO
Calculate electrical parameters for the power protection function
Run Power Protection Function
Fault Detected
YES
Send Trip Signal
Sample a new cycle for each input signal
NO

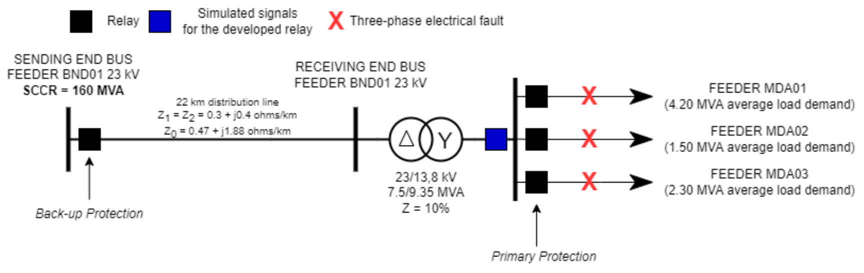
Figure 9: Flowchart for the power protection library

4 Experimental Setup

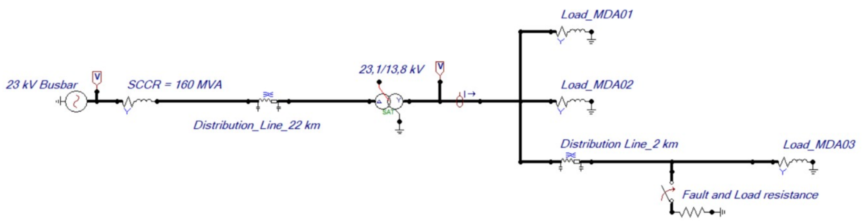
This section presents the simulated power system, modelled in an ATPDraw environment (ATPDraw, 2012), which provided the three-phase voltage and current signals that feed the proposed low-cost relay. It also shows the definite time overcurrent protection function and the neural network algorithm created to test the proposed low-cost relay regarding its real-time response facing electrical faults and load power demand variations.

4.1 Simulated scenario

Figure 10-a illustrates a single line diagram of a 22 km, 23 kV three-phase distribution feeder from the Blumenau II Substation, located in Blumenau – SC, Brazil. This feeder, whose codename is BND01, sends power to a step-down 23/13.8 kV transformer, from which three distribution feeders (MDA01, MDA02 and MDA03), provide electrical power to the city of Massaranduba - SC, Brazil, whose average maximum daily power demand is estimated in 8 MVA. This power system, owned by the utility company *Celesc Distribuição*, operates at a fundamental frequency of 60 Hz and 0.92 power factor. Currently, the back-up protection for the 13.8 kV relay feeders comes from the 23 kV relay from the Blumenau II Substation, which could take longer to operate due to the long distance, Thus, the proposed low-cost relay will emulate a protection relay at the 13.8 kV busbar, also shown in Figure 10-a. Figure 10-b shows the mentioned power system modelled within the ATPDraw environment.


Relay
Simulated signals for the developed relay
Three-phase electrical fault
SENDING END BUS
FEEDER BND01 23 kV
SCCR = 160 MVA
22 km distribution line
Z1 = Z2 = 0.3 + j0.4 ohms/km
Z0 = 0.47 + j1.88 ohms/km
RECEIVING END BUS
FEEDER BND01 23 kV
23/13,8 kV
7.5/9.35 MVA
Z = 10%
FEEDER MDA01
(4.20 MVA average load demand)
FEEDER MDA02
(1.50 MVA average load demand)
FEEDER MDA03
(2.30 MVA average load demand)
Back-up Protection
Primary Protection

(a)



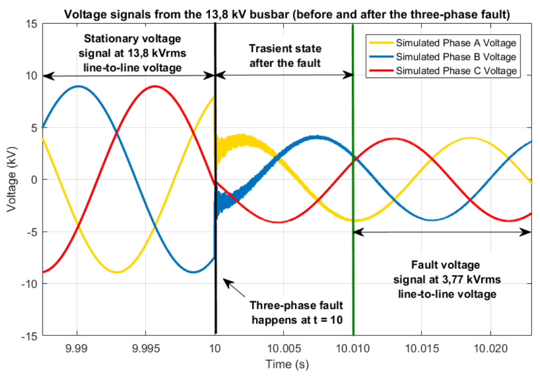
(b)

Figure 10: BND01 23 kV distribution feeder from Blumenau II Substation (a) single-line diagram (b) modelled feeder on ATP.

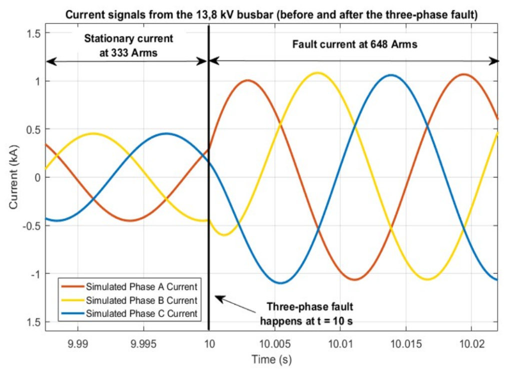
Three-phase symmetrical overcurrent faults of various magnitudes and positions along the 13.8 kV distribution feeders were simulated, as for a constant load surges and sags of 2.5 MW. This load represents the maximum power demand that power utility companies in Brazil have to provide power using their medium voltage distribution system (usually at 13,8 kV or 23 kV voltage levels) (Aneel, 2021). This load magnitude can create relevant transient signals in the low voltage busbar of the presented 23/13.8 kV transformer. The components employed to model the 23 kV distribution feeder, as for its main configurations, are shown in Table 4 from Appendix B.

The simulation time for the three-phase symmetrical faults and large load variations (load surge and load sag) was set in 20 seconds, where either event happens at the 10 seconds mark. An example of simulated fault and load variances are shown in Figure 11, representing signals from the MDA03 feeder, 2 km from the 13.8 kV busbar.

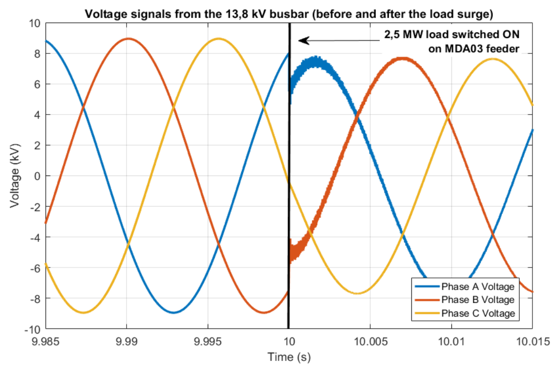
In order to feed the proposed low-cost relay with the simulated signals, a Relay tester from *Conprove Engenharia* (model CE – 6006) was utilized. This equipment is a universal six-phase Relay Tester, able to provide high precision voltage and current signals, emulating, for example, steady-state signals, overcurrent, over and under voltage, among others. It also provides test reports, from which the datasheet characteristics of an IED can be compared to its performance upon testing, and indicate whether the results match the predicted outcome or not. Through its Digital Signal Processor (DSP), this Relay tester can create various waveforms, reaching frequencies up to 3 kHz. Lastly, this device can reproduce signals generated by ATPDraw simulations (*.lis* output file) (Conprove Engenharia, 2021). Appendix A shows how the Relay tester was connected to the voltage and current sensors of the Signal Conditioning Module. The Relay tester scales down the original simulation values of voltages and currents, in order to meet the sensors' input parameters. The maximum voltage value provided by the Relay tester was set at 220 Vrms, while the maximum current value was set at 5 Arms. After the Signal Conditioning Module, the maximum voltage values that reach the Raspberry Pi 3B+ is 3.3 V for the voltage signals and 1.8 V for the current signals.



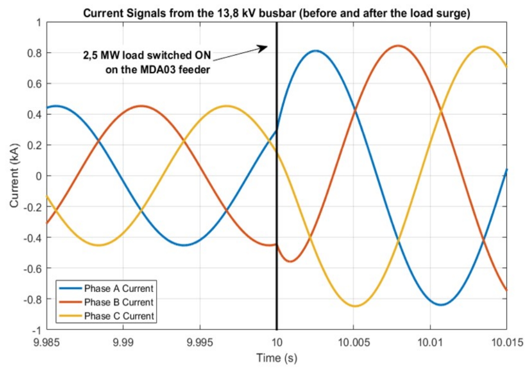
(a)



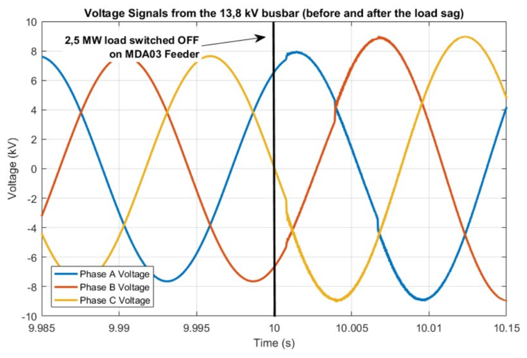
(b)

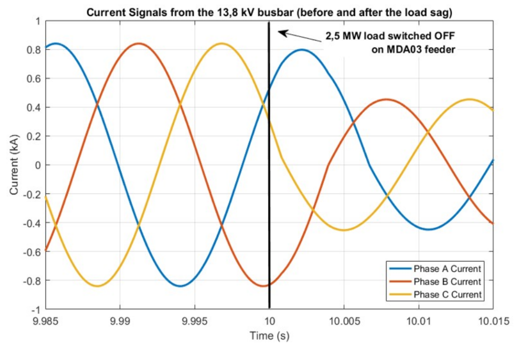


(c)



(d)





(e) (f)

Figure 11: Simulated voltage and current signals representing a (a), (b) three-phase fault, (c), (d), load surge, and (e), (f) load sag.

4.2 Power protection functions developed for the low-cost relay

Due to the open-source nature of the proposed low-cost relay, users of this platform can program their own novel protection functions in order to test them in a real-time setup. To evaluate the capability of the low-cost relay in maintain real-time response facing electrical faults, two functions were developed:

- **Definite time overcurrent protection** – the RMS value of each input signals, calculated over a period of half cycle, is compared to the pick-up value set by the user. A change in the state of one of the inputs must be maintained over a specified period of time (also set by the user) before a fault is registered and the trip signal sent;

- **Neural network algorithm** – it identifies three-phase overcurrent symmetrical faults against transient signals created by load surges and load sags in a distribution feeder, based on the RMS value of each input signal. Due to the similarities of the transient signals from those events (as show in Figures 11-a and 11-c), a neural network like a multilayer perceptron, can differentiate those events. This algorithm will have a specific amount of time (set by the user) to make the decision of whether the input signals represent a steady-state signal, an electrical fault, or a transient state. The ANN created only sends the trip signal after an electrical fault is detected and the time set by the user ends.

The definite time overcurrent protection was chosen to test the proposed low-cost relay due to its simplicity in be coded, since the only electrical parameter required to run this protection function is the RMS value of each input signal. As for the ANN created for this paper, this algorithm is not a novel power protection function. Its main goal is to show that the low-cost relay is capable to run more complex algorithms with real-time precision. The steps taken to create and train the ANN created are shown in Appendix C.

Unlike the input parameters presented in Subsection 3.4 of this paper, the user must set the pick-up value for the definite time overcurrent protection within the code, rather than declaring it in the Raspberry Pi's command prompt environment. The same is true for the trip time the user requires for both functions. It's recommended that the user choose pick-up values at the same scale provided by the Relay Tester (220 Vrms/5 Arms), in order to use the sensor's ratio with better precision.

5. Tests of the hardware setup and results

This section presents the tests and results made in the proposed low-cost relay, in order to evaluate its real-time precision while running a definite time overcurrent protection and the developed neural network algorithm. It's also verified the maximum sampling frequency the low-cost relay can handle without missing the time defined for each function presented. The results from the low-cost relay are compared to a MiCOM P142 relay from Schneider Electrics, a relay for feeder protection of AC systems (Schneider Electrics, 2020), in other to verify the efficacy of the real-time performance of the low-cost relay in relation to a commercial digital protection relay.

5.1. Tests' description

The following tests were made in the low-cost relay to evaluate its real-time performance:

- **Overcurrent protection test** – the low-cost relay and the P142 relay were configured to detect an overcurrent three-phase symmetrical fault (Figure 11) occurs, using a definite time function. The steady-state current of the input signals was set at 3 Arms, while the pick-up current was set at 5 Arms (Relay tester scale). The Relay tester from *Conprove Engenharia* measured the trip time from each relay, in order to verify if the low-cost has a response time closer to the commercial relay. The trip time for each relay was set at 250 ms, a pre-set configuration of the commercial relay. The P142 relay has a fixed sampling rate of 24 samples per cycle, which is equivalent to a sampling frequency of 1.44 kHz for a 60 Hz signal (Schneider Electrics, 2020). The low-cost also utilizes that sampling frequency as one of its input parameters. In addition, the low-cost relay had its the sampling frequency increased until it could no longer deliver real-time response, starting from 1 kHz and increased at 0.5 kHz steps;

- **Neural network test** – the low-cost relay ran the ANN created while three-phase symmetrical faults and transient signals from load variations (Figure 11) were sent to it by the Relay tester. The RMS value of the steady-state voltage and current of the input signals were set at 110 Vrms and 3 Arms, respectively, while the three-phase fault and the load surge were set at 220 Vrms and 5 Arms. The load sag was represented by the RMS values 50 Vrms/1 Arms (Relay tester scale). The ANN created will have a defined time of 250 ms to decide whether the input signals represent a three-phase fault or a transient state once one of those events happen, in contrast to the steady-state of the input signals. Like the Overcurrent protection test, the initial input sampling frequency of the low-cost relay was 1 kHz, and increased at 0.5 kHz steps until it could no longer deliver real-time response for this algorithm.

The wiring setup of the P142 relay with the Relay tester and the low-cost is presented in Figure 13 from Appendix A. The Relay tester uses its binary inputs to receive the output signals from both relays.

5.2 Results and discussions

The results from the Overcurrent protection test are shown in Figure 12.

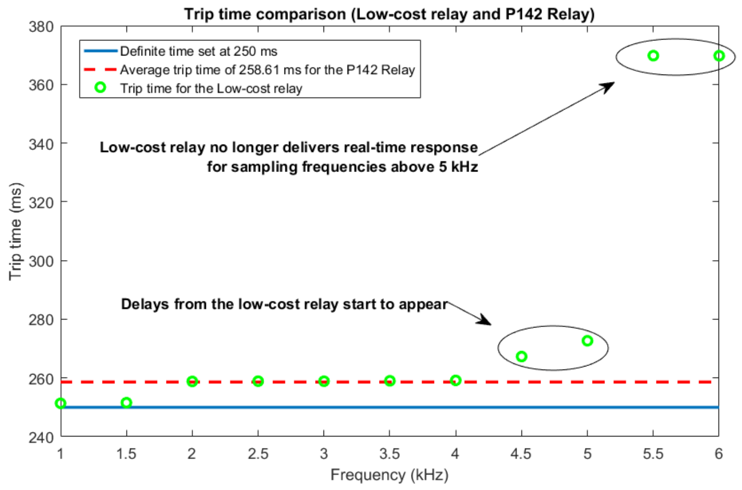


Figure 12: Comparison of response time of the low-cost relay and the P142 under overcurrent faults

From Figure 12, it's observed that the low-cost relay has a trip time very close to the P142 relay, at the initial sampling frequency. As the sampling frequency increased, the trip time of the low-cost relay had little delays, until it could no longer maintain real-time response for sampling frequencies above 5 kHz. Table 2 shows the same results, but numerically.

Table 2: Trip time delays from the Low-cost Relay and P142 relay above the 250 ms definite time

Relays	Total delay (ms)			
	Mean	*Minimum*	*Maximum*	*Variance*
P142 Relay (Schneider Electrics)	8.62	8.05	9.12	0.13
Low-cost relay (at the fixed sampling frequency of 1.44 kHz)	1.08	0.63	1.57	0.11
Low-cost relay (all sampling frequencies tested)	13.04	1.37	32.67	126.45
Number of measurements taken for each relay: 10				

From Table 2, it´s observed that, at the sampling frequency of 1.44 kHz, the low-cost relay has a response time to the three-phase symmetrical fault lower than the P142 relay, while having a variance in the trip time close to it. This is not an indication that the proposed relay of this paper is better than an all-functional commercial relay; it only shows that the real-time library developed for the low-cost relay is able to maintain precision to its response time similar to a commercial relay. By including the measurements from all tested sampling frequencies, the data variance drastically increases, showing that higher sampling frequencies have a significant impact in the real-time response of the proposed platform. This frequency limitation indicates that this low-cost relay is not recommendable to run DC (Direct Current) power protection functions, since they require non-conventional instruments, able to provide high sampling frequencies, from tens of kHz to few MHz (Wang et al., 2017).

As mentioned in Subsection 2.2 of this paper, the real-time library of the low-cost relay handles real-time task linearly, although with higher priority than ordinary tasks. Thus, the sampling process of the power protection library comes first before sending the samples to the definite time overcurrent protection, as for the consequential measurement process. The low-cost relay runs the definite time overcurrent protection numerous times before the set trip time is reached. The usual runtime for this power protection function is 150 µs. At each iteration, errors are accumulated while the samples and RMS values are sent from one process to another, hindering the final response time of the low-cost relay. Using one of the cores from the Raspberry Pi 3B+ minimizes those errors, but don't make them disappear. In sum, the low-cost relay can be considered a **soft real-time system**, able to validate the response time of a power protection algorithm created by a user of this platform, since its response time is closer to a commercial relay, although not recommended for actual protection of power system, since this platform is not a hard real-time system.

As for the Neural network test, the response time of the low-cost relay for the ANN created is presented in Table 3.

Table 3: Delays in the response time of the low-cost relay while running the ANN created

	Mean	*Minimum*	*Maximum*	*Variance*
ANN created (ms) (all sampling frequencies tested)	5.66	4.65	8.88	1.47
Number of runtimes: 10				

Similar to the previous test, at the sampling frequency of 5 kHz, the low-cost relay could no longer deliver real-time response (response time of the ANN created were well above 250 ms). In contrast, the low-cost relay had a higher variance while running the ANN created, compared to the definite time overcurrent protection. This reflects the more complex nature of the algorithm executed. However, the delay in the response time for this test is lower than the trip time of the P142 relay, and has a better result once all sampling frequencies tested are compiled together. Thus, the low-cost relay is able to maintain real-time precision while running complex algorithms, regardless of the sampling frequency utilized (up to the 5 kHz limit). Like the previous test, the low-cost relay is a soft real-time system, since it runs the ANN created at 250 µs at each iteration, while accumulating errors until the defined response time ends.

During the neural network tests, the accuracy of the ANN created was estimated in around 90%, since only one of the ten tests shown in Table 3 gave a different outcome than expected (false negative result by detecting a load surge instead of a three-phase fault).

5 Conclusion

This paper presented a new platform that can be utilized by researchers as a low-cost digital relay, able to test their novel power protection functions in real-time; which is affordable, doesn't require low level programming or deep knowledge of its architecture. Although the main component of this universal relay, a Raspberry Pi 3B+, was not originally created to work with real-time systems, like power protection systems, this single-board computer can achieve real-time response by having the kernel of its OS properly configured for this purpose.

Combining the Raspberry 3B+ with AC voltage and current sensors and an A/D converter, it was shown that the proposed platform has a response time to electrical faults similar to a commercial relay. The low-cost relay was able to give real-time precision at sampling frequencies up to 5 kHz, whether running a simple overcurrent protection function or a more complex algorithm, like a neural network. The delays measured for the low-cost relay show that this platform is able to validate novel power protection algorithms, though not recommended to be utilized as an actual power protection relay in real-time systems, since its real-time response is classified as soft.

Further testing is required to evaluate if more functionalities can be added to this platform without compromising its real-time response, like Fourier or Wavelet Transformations; other AC power protection functions, like distance protection, directional protection, differential protection, or even novel power protection algorithms in the literature. The real-time library presented in this paper can also be improved by adding parallelism computing to its structure, thus making the sampling process more efficient and giving more time to the low-cost relay while analyzing a fault signal; giving it faster and precise response time.

6 Acknowledgment

The authors would like to thank Federal University of Piauí (UFPI) for supporting this research.

7 References

Aneel. (2021). Resolução normativa Aneel nº 1.000, de 7 de dezembro de 2021. Aneel.

Aqeel, A. (2018, July 25). Introduction to Raspberry Pi 3 B+. The Engineering Projects. Retrieved December 4, 2021, from https://www.theengineeringprojects.com/2018/07/introduction-to-raspberry-pi-3-b-plus.html

ATPDraw. (2012). Welcome to the web page of ATPDraw. ATP Draw. Retrieved December 4, 2021, from https://www.atpdraw.net/index.php

Buttazzo, G. C. (2013). Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications (Real-Time Systems Series, 24) (Softcover reprint of hardcover 3rd ed. 2011 ed.). Springer.

Chalas, K. (2015, September). Evaluation of Real-time operating systems for FGC controls. CERN.

Conprove Engenharia. (2021, June 21). CE − 6006. Conprove. Retrieved December 4, 2021, from https://conprove.com/produto/02-ce-6006-testador-universal-hexafasico-e-analisador-de-energia-microprocessado-com-protocolo-iec-61850-mala-de-testes-de-reles-hexafasica/

Dantas, D. T., Pellini, E. L., & Manassero Junior, G. (2018). Energy and reactive power differential protection hardware-in-the-loop validation for transformer application. The Journal of Engineering, 2018(15), 1160−1164. https://doi.org/10.1049/joe.2018.0223

ETC. (2021). ZMPT101B(ZMPT107) voltage transformer operating guide [Datasheet]. ETC.

Gerum, P. (2021, July 3). Home · Wiki · xenomai / xenomai. GitLab. https://source.denx.de/Xenomai/xenomai/-/wikis/home

Johansson, G. (2018). Real-Time Linux Testbench on Raspberry Pi 3 using Xenomai (Monography). School of electrical engineering and computer science.

Joy, U. B., Chakraborty, S., Murad, T. K., Tasnim, A., Barua, B., & Das, J. (2022). Microcontroller Based Feeder Protection System from Various Fault Conditions in Distribution Line. 2022 International Conference on Innovations in Science, Engineering and Technology (ICISET), 89-94. https://doi.org/10.1109/ICISET54810.2022.9775914.

Kuffel, R., Forsyth, P., & Peters, C. (2016). The Role and Importance of Real Time Digital Simulation in the Development and Testing of Power System Control and Protection Equipment. IFAC-PapersOnLine, 49(27), 178−182. https://doi.org/10.1016/j.ifacol.2016.10.739

Learn Open Energy Monitor. (n.d.). CT Sensors - Interfacing with an Arduino [Illustration]. Learn Open Energy Monitor. https://learn.openenergymonitor.org/electricity-monitoring/ct-sensors/interface-with-arduino

Martins, C. A. P. (2017). Desenvolvimento de hardware para aquisição de sinais e processamento de algoritmos de proteção e controle (Master's dissertation). Universidade Estadual de Campinas.

Mckenney, P. (2005, August 10). A realtime preemption overview. LWN.Net. Retrieved December 4, 2021, from https://lwn.net/Articles/146861/

Microchip. (2008). MCP3004/3008 [Datasheet]. Microchip.

Mitra, S., & Chattopadhyay, P. (2019). Design and implementation of flexible Numerical Overcurrent Relay on FPGA. International Journal of Electrical Power & Energy Systems, 104, 797–806. https://doi.org/10.1016/j.ijepes.2018.07.022

Molloy, D. (2016). Exploring Raspberry Pi: Interfacing to the Real World with Embedded Linux (1st ed.). Wiley.

Monaro, R. M., Silva, R. P. M. D., Vieira Júnior, J. C. D. M., & Coury, D. V. (2012). Sistema integrado para desenvolvimento e execução em tempo real de algoritmos de proteção de sistemas elétricos. Sba: Controle & Automação Sociedade Brasileira de Automatica, 23(2), 202–215. https://doi.org/10.1590/s0103-17592012000200007

Monaro, R. M. (2013). Lógica fuzzy aplicada na melhoria da proteção digital de geradores síncronos (PH.D thesis). Universidade de São Paulo.

Pellini, E. L., Senger, E. C., Manassero Junior, G., Reis Filho, F., & Rufato Junior, E. (2013). Custom distribution feeder recloser IED with high impedance protection function. International Conference and Exhibition on Electricity Distribution, 22, 1–4.

Raspberry Pi Foundation. (2021, July 5). Raspberry Pi Foundation - About Us. Raspberry Pi. Retrieved December 4, 2021, from https://www.raspberrypi.org/about/

Roberts, T. [tjrob]. (2013, February 19). Solution: Dedicating one core to a real-time process - Raspberry Pi Forums [Raspberry Pi Forum post]. Raspberry Pi Forum. https://forums.raspberrypi.com/viewtopic.php?t=228727

Ryan, W. [wryan67]. (2020, April 23). GitHub - wryan67/VoltageCatcher: Capture voltage using Raspberry Pi and mcp3008 [GitHub post]. GitHub. https://github.com/wryan67/VoltageCatcher

Schneider Electric. (2020, October). Easergy MiCOM P14x [Feeder Management Relay]. Schneider Electric.

Tam, H. [Tam Ho]. (2018, September 16). GitHub - thanhtam-h/rpi23-rt: Preempt-rt patched kernel 4.9.80 for raspberry pi 2, 3 (include 3b+) [Github post]. GitHub. https://github.com/thanhtam-h/rpi23-rt

TensorFlow (2021, September 1). TensorFlow Lite for microcontrollers. TensorFlow. https://www.tensorflow.org/lite/microcontrollers

Wang, M., Abedrabbo, M., Leterme, W., van Hertem, D., Spallarossa, C., Oukaili, S., Grammatikos, I., & Kuroda, K. (2017). A Review on AC and DC Protection Equipment and Technologies: Towards Multivendor Solution. Cigrè Winnipeg 2017 Colloquium, Winnipeg, Canada.

Wang, Y., & Dinavahi, V. (2016). Real-time digital multi-function protection system on reconfigurable hardware. IET Generation, Transmission & Distribution, 10(10), 2295–2305. https://doi.org/10.1049/iet-gtd.2015.0718

Wei, M., & Chen, Z. (2010). Distribution system protection with communication technologies. IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society. Published. https://doi.org/10.1109/iecon.2010.5675293

Winkle, L. V. [codeplea]. (2020, December 17). GitHub - codeplea/genann: simple neural network library in ANSI C [GitHub post]. GitHub. https://github.com/codeplea/genann

YHDC. (2021). Split core current transformer [Datasheet]. YHDC.

APPENDIX A – Low-cost relay circuit diagram

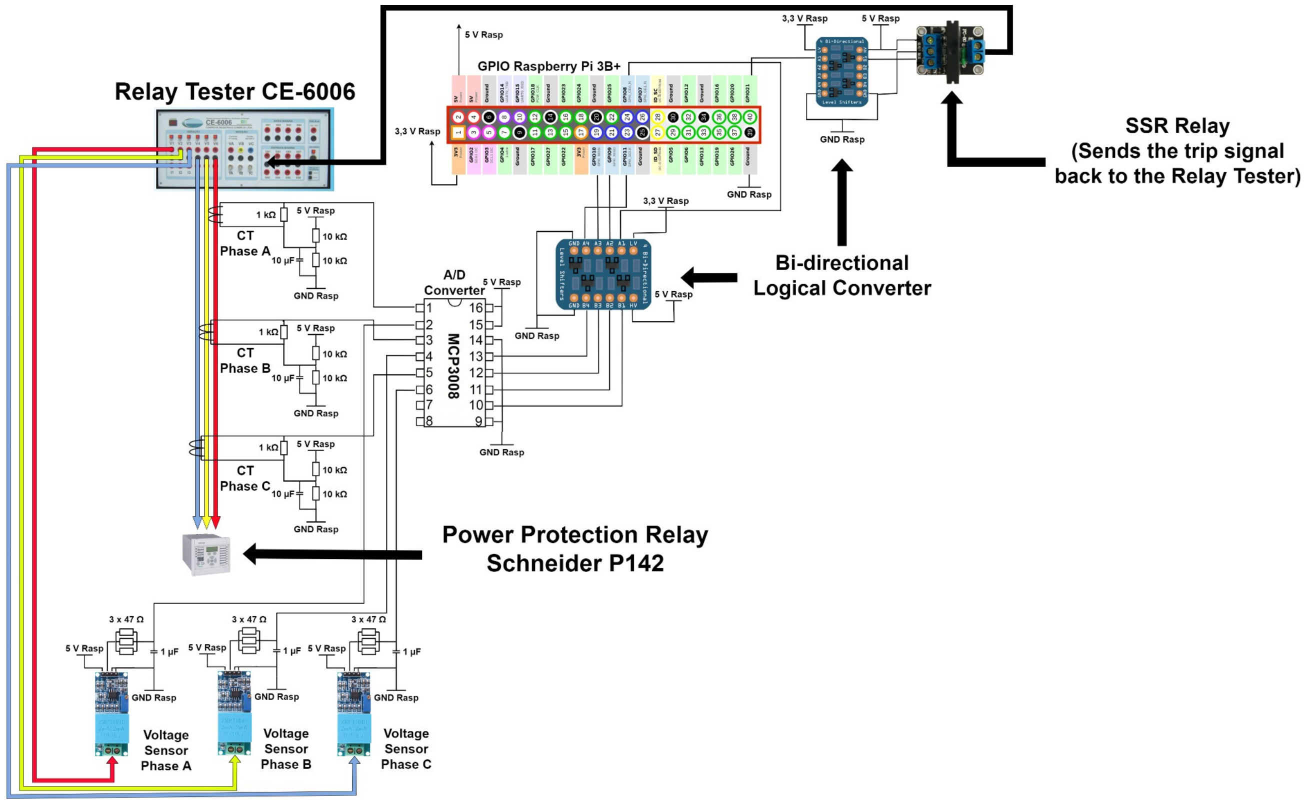


Figure 13: Low-cost relay circuit diagram

The total cost spent of each hardware component of the low-cost relay is presented in Table 4. The price reference is dated from June/2022. The prices presented are an average value from various online electronics shops.

Table 4: Hardware components of the low-cost relay and their cost

Component	Quantity	Unitary Cost (US$)	Total Cost (US$)
AC voltage sensor ZMPT101B (0 a 250V)	3	6.12	18.36
Non-invasive current sensor SCT – 013 - 20 A	3	15.06	45.17
400 point breadboard	1	3.33	3.33
140 Jumpers Kit for breadboard	1	4.63	4.63
Resistor 47R 5% (1/4W)	9	0.01	0.11
Resistor 1K 5% (1/2W)	3	0.03	0.08
Resistor 10K 5% (1/2W)	6	0.07	0.41
Electrolytic Capacitor 1μF / 50V	3	0.05	0.14
Electrolytic Capacitor 10μF / 100V	3	0.05	0.14
MCP3008 Analog/Digital Converter	1	6.83	6.83
Bi-directional Level Shifter 3.3V-5V - 4 Chanels	2	1.28	2.57
SSR Relay 5V	1	4.00	4.00
Raspberry Pi 3 Model B+	1	84.41	84.41
Total Investment: US$ 170.30			

APPENDIX B – ATPDraw components configuration

Table 5: ATPDraw components and its configurations to emulate the BND01 distribution feeder

ATP Component	**Configuration Parameters**
AC source (ACSOURCE)	Three-phase grounded voltage source, 23 kV RMS line-to-line, phase angles 0°, 120° and 240°, frequency 60 Hz
Short-circuit rating (RLCY3)	Resistance of all phases equal to 3.04 ohms Inductance of all phases equal to 3.43 mH Capacitance of all phases equal to zero
Distribution line (LINEPI3S)	Resistance in the positive sequence system = 7.79 e-4 ohms/m Inductance in the positive sequence system = 50.85 e-3 mH/m Resistance in the zero sequence system = 6.13 e-4 ohms/m Inductance in the zero sequence system = 11.37 e-3 mH/m All Capacitances equal to zero Length = 22000 meters
Transformer (SATTRAFO)	Delta to Star grounded coupling, phase shift of 30° 23 kV primary voltage/7,967 kV secondary voltage Primary resistance of 9.58 ohms/Secondary resistance of 1.67 ohms Primary inductance of 28.93 mH/Secondary inductance of 5.06 mH
Load from MDA01 feeder (RLCY3)	Resistance of all phases equal to 29.34 ohms Inductance of all phases equal to 33.16 mH Capacitance of all phases equal do zero
Load from MDA02 feeder (RLCY3)	Resistance of all phases equal to 104.91 ohms Inductance of all phases equal to 118.55 mH Capacitance of all phases equal do zero
Load from MDA03 feeder (RLCY3)	Resistance of all phases equal to 81.03 ohms Inductance of all phases equal to 84.24 mH Capacitance of all phases equal do zero
Switch to emulate the fault (TSWITCH)	Close time = 0.05 seconds; Open time = 1 second (for three-phase faults) Close time = -1 second; Open time = 0.05 seconds (for load variance) Number of phases = 3
Resistance connected between the switch and ground to emulate the fault (RESISTOR)	Resistance = 1 ohm (for three-phase faults) or 8.103 ohms (for 2.5 MW load variance)
Simulation configuration	Time domain simulation Δt = 1e-6, maximum simulation time = 20 seconds

APPENDIX C – Neural Network creation and training

The ANN created is a basic feed-forward neural network, particularly a multilayer perceptron type (MLP). It was created based on the C++ library provided by Winkle (2018), using a backpropagation algorithm, able to train and save the weights' values of each neuron. The dataset used to train this ANN consisted of 50,000 samples, acquired by the low-cost relay at the sampling frequency of 3 kHz (due to the Relay tester limitation, a higher sampling frequency was not necessary), which implicate 50 samples/period of each AC input signal. For each group of 50 samples, the RMS value of each AC voltage and current input signals was calculated, contemplating data before and after the three-phase symmetrical faults, as for the load surges and sags of a constant load of 2.5 MW, simulated at various positions along the 13.8 kV distribution feeders presented in Section 4 of this paper. From a total of 1,000 RMS values for each input signal, 100 of them represent the steady-state of each input signal, while 300 of them represent the three-phase faults. The remaining 300 correspond to the load variations. The proportion chosen to train and test the ANN created was divided in 80%/20%, respectively.

The ANN created consists of six input parameters (voltage and current RMS value of each phase); one layer of six hidden neurons; and three neurons in the output layer. The combination of the binary results in each output represents the state of the input signals. Figure 14 graphically shows the ANN created. Due to its simplicity, no cross-validation was required. The creation and training of this neural network was all done within the low-cost relay.

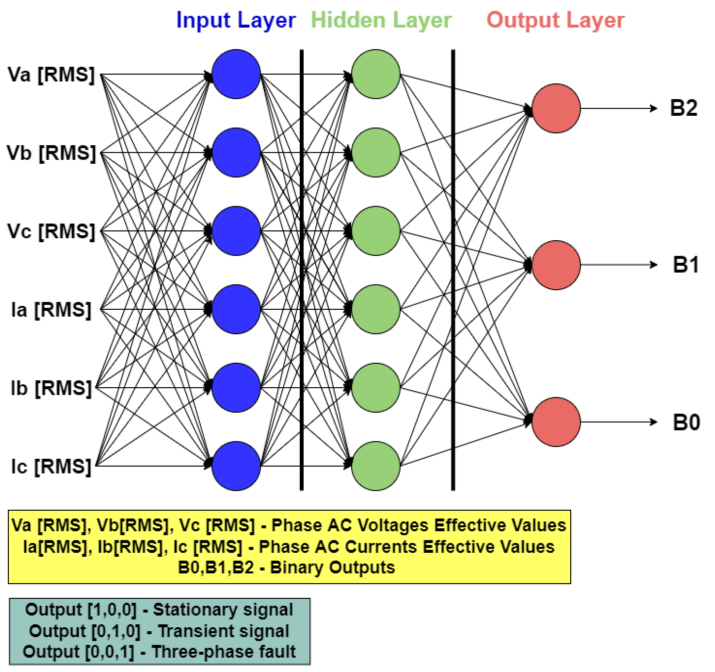

Input Layer
Hidden Layer
Output Layer
Va [RMS]
Vb [RMS]
Vc [RMS]
Ia [RMS]
Ib [RMS]
Ic [RMS]
B2
B1
B0
Va [RMS], Vb[RMS], Vc [RMS] - Phase AC Voltages Effective Values
Ia[RMS], Ib[RMS], Ic [RMS] - Phase AC Currents Effective Values
B0,B1,B2 - Binary Outputs
Output [1,0,0] - Stationary signal
Output [0,1,0] - Transient signal
Output [0,0,1] - Three-phase fault

Figure 14: Power protection algorithm based on Neural Network

Case the user already has their own ANN multilayer perceptron, they can create it within the low-cost relay by declaring the following variables in a separate file (e.g., .txt file):

- Number of neurons in the input layer;
- Number of hidden layers;
- Number of neurons in each hidden layer;
- Number of neurons in the output layer;
- Weight of each neuron declared, starting from the first neuron from the input layer, to the last neuron of the output layer.

All variables must be declared sequentially in the separate file, with no delimiters.