

CENTRO UNIVERSITÁRIO DO INSTITUTO MAUÁ DE TECNOLOGIA

Escola de Engenharia Mauá

Engenharia Elétrica

DANIEL DALLA VECCHIA GUETER

JOÃO VICTOR MORAES DOS SANTOS

SULAMITA CAROLINA LOPES

THIAGO DA SILVA AMATE

VICTOR HENRIQUE PALMA LOIOLA

IoT Aplicada à Energia Renovável

São Caetano do Sul

2019

DANIEL DALLA VECCHIA GUETER

JOÃO VICTOR MORAES DOS SANTOS

SULAMITA CAROLINA LOPES

THIAGO DA SILVA AMATE

VICTOR HENRIQUE PALMA LOIOLA

IoT Aplicada à Energia Renovável

Trabalho de Conclusão de Curso apresentado
à Escola de Engenharia Mauá do Centro
Universitário do Instituto Mauá de Tecnologia
como requisito parcial para a obtenção do
título de Engenheiro Eletricista.

Orientador: Prof. Dr. Sergio Ribeiro Augusto

Área de concentração: Engenharia Elétrica

São Caetano do Sul

2019

IoT Aplicada à Energia Renovável. / Daniel Dalla Vecchia Gueter [et al.] — São Caetano do Sul: CEUN-IMT, 2019.
141 p.

Trabalho de Conclusão de Curso – Escola de Engenharia Mauá do Centro Universitário do Instituto Mauá de Tecnologia, São Caetano do Sul, SP, 2019.

Orientador(a): Prof. Dr. Sergio Ribeiro Augusto

1. Modulo fotovoltaico.
 2. Energia solar.
 3. Internet das coisas.
 4. Redes neurais artificiais.
 5. LoRa.
 6. Inteligência artificial.
 7. Previsão de produção de energia.
 8. Monitoramento em tempo real.
- I. Gueter, Daniel. II. Dos Santos, João Victor. III. Lopes, Sulamita. IV. Amate, Thiago. V. Loiola, Victor. VI. Instituto Mauá de Tecnologia. Escola de Engenharia. VII. Título.

DANIEL DALLA VECCHIA GUETER

JOÃO VICTOR MORAES DOS SANTOS

SULAMITA CAROLINA LOPES

THIAGO DA SILVA AMATE

VICTOR HENRIQUE PALMA LOIOLA

IoT Aplicada à Energia Renovável

Trabalho de Conclusão de Curso aprovado
pela Escola de Engenharia Mauá do Centro
Universitário do Instituto Mauá de Tecnologia
como requisito parcial para a obtenção do
título de Engenheiro Eletricista.

Banca avaliadora:

Prof. Dr. Sergio Ribeiro Augusto
Orientador

Prof. Dr. Edval Delbone
Avaliador

Prof. Dr. Wânderson de Oliveira Assis
Avaliador

São Caetano do Sul, 12 de Dezembro de 2019.

*Dedicamos este trabalho aos
nossos familiares que, com muito carinho e apoio,
não mediram esforços para que chegássemos até aqui.*

AGRADECIMENTOS

Após todo esse período dedicado ao trabalho, não podíamos deixar de agradecer a uma série de pessoas que nos momentos mais conturbados nos ajudaram e deram o apoio necessário. Também gostaríamos de agradecer aos pequenos momentos de risadas e descontração, que apesar de todos os problemas e estresses, o grupo proporcionou a ele mesmo.

Nosso grupo gostaria de agradecer em especial ao nosso querido, amado e aclamado professor e orientador Dr. Sergio Ribeiro Augusto, conhecido por nós como Sergião e considerado um pai; juntamente com a sua esposa que compreendeu os inúmeros atrasos dele no jantar. Paciência e sabedoria eram virtudes que buscávamos em um orientador, e o Sergião com certeza possui ambas.

Aos nossos familiares, amigos, parceiros e parceiras, que acompanharam de perto, aguentaram, e nos puseram para cima em todos os momentos tristes, difíceis, de desespero e de euforia. Sem eles esse trabalho, e nossas vidas, não seriam nada.

Somos gratos pela infinita graça e misericórdia do Senhor que nos permitiu concluir essa etapa das nossas vidas.

Em especial, ao engenheiro Rodrigo França por todo o tempo dedicado, suporte e a atenção dada nessa fase, que apesar de nem a pertencer foi tratada como própria.

Agradecemos também ao corpo docente do Instituto Mauá de Tecnologia, em especial aos professores, mestres, doutores e técnicos do tronco de elétrica e eletrônica, que ao longo da nossa graduação sempre nos proveram conhecimento e ajuda quando necessário, e até quando não. Também apresentamos nossos agradecimentos ao Centro de Pesquisas da Mauá, por ter dado o suporte necessário em segmentos extremamente complexos desse trabalho.

*“Quando uma criatura humana desperta um grande sonho
e sobre ele lança toda a força de sua alma, todo o universo conspira a seu favor.”*

Johann Goethe

RESUMO

A introdução de fontes de energias renováveis no sistema elétrico brasileiro vem crescendo constantemente, aumentando sua contribuição na geração de energia elétrica e abrindo caminho para um novo mercado, em especial às fazendas solares. Estas contêm centenas e até milhares de painéis fotovoltaicos, os quais são conectados ao sistema interligado nacional, sendo crucial realizar periodicamente a supervisão e a manutenção dos equipamentos, possibilitando-os trabalhar em condição nominal e evitando ao máximo falhas e problemas operacionais que limitem a potência e a energia gerada pelo complexo solar.

Aplicando conceitos elétricos, eletrônicos, de internet das coisas e de inteligência artificial, é possível medir e prever a geração de um módulo fotovoltaico, e ao comparar os valores medidos e previstos, pode-se obter conclusões de sua condição de operação.

Seguindo as orientações acima, desenvolveu-se neste trabalho uma solução capaz de monitorar e prever a geração de um módulo fotovoltaico. Utilizando um conjunto de sensores, assim como um microcontrolador e um módulo de radiofrequência do tipo LoRa, mede-se e transmite-se os parâmetros relevantes de uma placa solar (temperatura, radiação solar, tensão e corrente gerada) para um servidor que possibilita a visualização e análise desses dados em tempo real. Com os mesmos dados enviados, foi efetuada a previsão da tensão e corrente gerada por meio de um algoritmo proveniente de uma rede neural artificial, a qual foi previamente criada e treinada com valores obtidos pelo próprio *hardware* desenvolvido, utilizando como base um módulo fotovoltaico em condições de operação reais.

Palavras-chave: Módulo fotovoltaico. Energia solar. Internet das coisas. Redes neurais artificiais. LoRa. Inteligência artificial. Previsão de produção de energia. Monitoramento em tempo real.

ABSTRACT

The introduction of renewable energy sources into the brazilian electric system is in constantly growth, increasing its contribution in electric energy generation and opening the path for a new market, in special solar farms. These contain hundreds and even thousands of photovoltaic panels, which are connected to the national interconnected system, being crucial to execute periodically the supervision and maintenance of the equipment, allowing them to work on nominal condition and avoiding the number of failures as much as possible, as well as operational problems that limit the power and energy generated by the solar complex.

Applying electric, electronic, internet of things and artificial intelligence concepts, it's possible to measure and predict a photovoltaic module generation, and by comparing the measured and predicted values, one can conclude about its operation conditions.

Following the orientations above, it was developed in this work a solution able to monitor and predict the energy generation of a photovoltaic module. Using a mix of sensors, along with a microcontroller and a LoRa radiofrequency module, the relevant parameters of a solar panel (temperature, solar radiation, generated voltage and current) are measured and transmitted to a server that allows the real time visualization and analysis of these data. With the same sent data, the prediction of the generated voltage and current was made by an algorithm resulted of an artificial neural network, which was previously trained with values obtained by the developed hardware, using a photovoltaic module in real operation conditions as a base.

Keywords: Photovoltaic module. Solar energy. Internet of things. Artificial neural networks. LoRa, Artificial intelligence. Energy production prediction. Real time monitoring.

LISTA DE FIGURAS

Figura 1 – Matriz Elétrica Brasileira 2017	23
Figura 2 – Arquitetura LoRa.....	28
Figura 3 – Curvas da tensão medida pela tensão compactuada.....	29
Figura 4 – <i>One-Diode Model</i>	30
Figura 5 – Diagrama Geral do Sistema de Monitoramento do Painel Fotovoltaico.....	31
Figura 6 – Invólucro desenvolvido para abrigar o <i>hardware</i>	33
Figura 7 – Invólucro alojado atrás do painel solar.....	34
Figura 8 – Esquema elétrico elaborado na <i>protoboard</i>	35
Figura 9 – Esquema de ligação do Arduino com os sensores	35
Figura 10 – Arduino Mega	37
Figura 11 – Dragino LoRa Shield	38
Figura 12 – Módulo de Bluetooth.....	40
Figura 13 – Módulo <i>SD Card</i>	41
Figura 14 – Real-Time-Clock (RTC)	42
Figura 15 – Módulo Regulador de Tensão	43
Figura 16 – Módulo de Gerenciamento de Baterias.....	43
Figura 17 – Baterias 18650	45
Figura 18 – Sensor de Temperatura DS18B20.....	45
Figura 19 – Sensor de Luminosidade BH1750FVI	46
Figura 20 – Sensor de Tensão P25-GBK 25V DC	47
Figura 21 – Sensor de Corrente ACS712	47
Figura 22 – Sensor de Chuva YL-83.....	48
Figura 23 – Painel Fotovoltaico	49

Figura 24 – Esquema elétrico do circuito BMS	51
Figura 25 – Fluxograma do <i>software</i> implementado no Arduino Mega.....	54
Figura 26 – Telas de inicialização do aplicativo	56
Figura 27 – <i>Auth Token</i> e projetos desenvolvidos.....	57
Figura 28 – Tela principal do aplicativo.....	58
Figura 29 – Configuração dos <i>Widgets</i> : <i>Image Gallery</i> , <i>BLE</i> e <i>Notification</i>	59
Figura 30 – Rede de Nós do Sistema de Supervisão (Node-RED).....	61
Figura 31 – Bloco de aquisição dos dados do servidor	61
Figura 32 – Configuração do nó “EnergiaRenov”	62
Figura 33 – Bloco de decodificação dos dados.....	62
Figura 34 – Configuração do nó “Decode”	63
Figura 35 – Configuração do nó “Encode”	63
Figura 36 – Estrutura da mensagem hexadecimal proveniente do servidor	64
Figura 37 – Bloco de interpretação da mensagem e exibição dos dados.....	65
Figura 38 – Nós de transferência para variáveis globais.....	65
Figura 39 – Configuração do nó “set global.val_temperatura”	66
Figura 40 – Bloco de análise e exibição dos dados	66
Figura 41 – Modelo de um neurônio	68
Figura 42 – Fluxograma da aplicação de uma rede neural artificial.....	70
Figura 43 – Função de ativação <i>purelin</i> e <i>tansigmoidal</i> respectivamente	71
Figura 44 – Topologia da rede neural artificial criada	72
Figura 45 – Teste realizado na casa de um membro do grupo	75
Figura 46 – Dados coletados pelo SD Card com a placa instalada na casa de um membro do grupo	75
Figura 47 – Dados coletados pelo <i>Bluetooth</i>	76

Figura 48 – Dados coletados pelo SD Card com a placa instalada na casa de um membro do grupo na semana seguinte	76
Figura 49 – Placa instalada no quarto andar do bloco W	77
Figura 50 – <i>Dashboard</i> do LoRa	78
Figura 51 – Placa instalada no terraço do bloco W	79
Figura 52 – RNA da tensão elétrica.....	79
Figura 53 – RNA da corrente elétrica.....	80
Figura 54 – <i>Simulink</i>	81
Figura 55 – Coleta de dados com o holofote	82
Figura 56 – RNA da tensão elétrica com o holofote como fonte de luminosidade	82
Figura 57 – <i>Dashboard</i> em condição normal	83
Figura 58 – <i>Dashboard</i> com desvio de operação	84
Figura 59 – <i>Dashboard Blynk</i>	85

SUMÁRIO

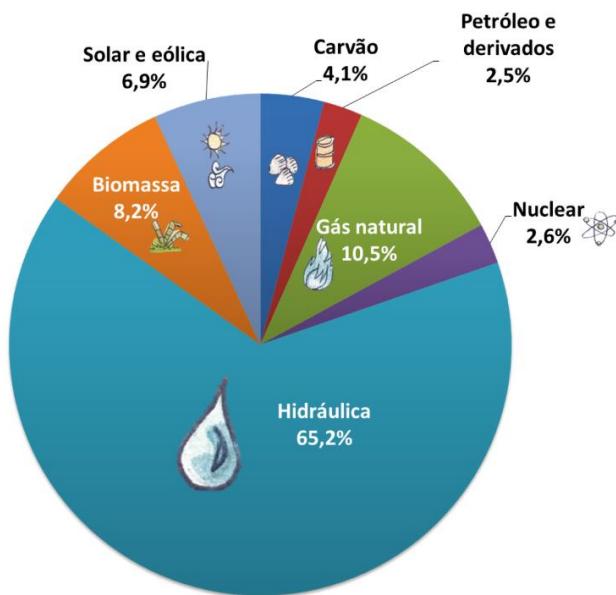
1	INTRODUÇÃO	23
1.1	OBJETIVOS PRINCIPAIS	24
2	REVISÃO DA LITERATURA.....	25
2.1	PAINEL SOLAR.....	26
2.2	TECNOLOGIA IOT E LORA	27
2.3	MODELAGEM E PREVISÃO	28
3	MATERIAIS E MÉTODOS	31
3.1	DESENVOLVIMENTO DO <i>HARDWARE</i>	32
3.1.1	MÓDULOS E SENsoRES UTILIZADOS.....	37
3.1.1.1	Arduino.....	37
3.1.1.2	Módulo de Comunicação LoRa	38
3.1.1.3	Módulo de Comunicação <i>Bluetooth</i>.....	39
3.1.1.4	Módulo <i>SD Card</i>.....	41
3.1.1.5	<i>Real-Time-Clock (RTC)</i>.....	42
3.1.1.6	Módulo Regulador de Tensão	42
3.1.1.7	Módulo de Gerenciamento de Baterias	43
3.1.1.8	Baterias	44
3.1.1.9	Sensor de Temperatura	45
3.1.1.10	Sensor de Luminosidade	46
3.1.1.11	Sensor de Tensão.....	47
3.1.1.12	Sensor de Corrente	47
3.1.1.13	Sensor de Chuva	48
3.1.1.14	Painel Fotovoltaico	49
3.1.2	<i>BATTERY MANAGEMENT SYSTEM (BMS)</i>	50
3.2	DESENVOLVIMENTO DO <i>SOFTWARE</i>	52
3.2.1	<i>FIRMWARE</i> DO DISPOSITIVO DE COLETA DE DADOS.....	52
3.2.2	<i>SOFTWARE</i> DO APlicativo de ACESSO REMOTO DE CURTA DISTÂNCIA.....	56
3.2.3	<i>SOFTWARE</i> DO APlicativo de ACESSO REMOTO DE LONGA DISTÂNCIA	60
3.3	REDE NEURAL ARTIFICIAL	67
3.3.1	INTRODUÇÃO E CONCEITOS BÁSICOS.....	67

3.3.2	APLICAÇÃO DA REDE NEURAL ARTIFICIAL	69
4	RESULTADOS E DISCUSSÃO.....	75
5	CONCLUSÕES	87
6	TRABALHOS FUTUROS.....	89
	REFERÊNCIAS	91
	APÊNDICE A – INSTALAÇÃO DAS BIBLIOTECAS DO ARDUINO.....	95
	APÊNDICE B – INSTALAÇÃO E CONFIGURAÇÃO DO NODE-RED ...	99
	APÊNDICE C – CÓDIGO FONTE DO DISPOSITIVO COLETOR.....	103
	APÊNDICE D – CÓDIGO FONTE DOS BLOCOS NO NODE-RED	117
	APÊNDICE E – CRIAÇÃO E SIMULAÇÃO DE UMA REDE NEURAL ARTIFICIAL NO <i>MATLAB</i>	123
	APÊNDICE F – IMPLEMENTAÇÃO MANUAL DE UMA REDE NEURAL DO TIPO PERCEPTRON MULTICAMADAS (MLP) GERADA EM <i>MATLAB</i>	131
	APÊNDICE G – TABELA DE CUSTOS	141

1 INTRODUÇÃO

Com o aumento da população mundial e da extrema dependência de energia elétrica no dia-a-dia das pessoas, novas alternativas de geração de energia sustentável estão sendo estudadas e desenvolvidas para suprir a necessidade do consumo populacional. Dentre essas alternativas têm-se mundialmente a energia eólica e a energia solar como relevantes, com tecnologias consolidadas, tendo inúmeros casos de sucesso e se tornando cada vez mais atrativas financeiramente (IRENA, 2019). Quando se trata da matriz elétrica brasileira, a grande parte da geração provém da energia hidráulica. Entretanto, pode-se notar pela Figura 1 que a energia solar e eólica em conjunto já ultrapassaram meios mais tradicionais de geração de energia, como as termoelétricas movidas a carvão ou a petróleo e derivados.

Figura 1 – Matriz Elétrica Brasileira 2017



Fonte: Empresa de Pesquisa Energética, 2018

Os grandes núcleos de geração de energia eólica e solar são denominados fazendas solares ou eólicas, contendo centenas ou até milhares de equipamentos, painéis solares ou turbinas eólicas, destinados à geração de energia renovável. Estes equipamentos, como todo existente, estão sujeitos à diversos tipos de problemas, ainda mais por estarem expostos continuamente ao ar livre e às suas condições climáticas. Na ocorrência de uma falha,

quanto antes se identificar a mesma e se fazer a devida manutenção mais rapidamente esse sistema gerador de energia voltará a trabalhar integralmente.

Uma solução válida para a prevenção dessas falhas, ou pelo menos para diminuir a quantidade de tempo que um equipamento desses sistemas se mantém inoperante, é o monitoramento remoto do mesmo, o qual se torna viável pelas tecnologias de comunicação e aquisição existentes e mais interessante ainda quando aplicados novos conceitos da Indústria 4.0 (Agenda brasileira para a Indústria 4.0, 2019), como a Internet das Coisas (MORGAN, 2014), conhecida também como IoT (*Internet of Things*).

Internet das coisas é o conceito de basicamente se conectar qualquer dispositivo com um botão de liga e desliga à Internet, podendo ser celulares, lâmpadas, sensores, ou até mesmo coisas inesperadas como a broca de uma furadeira ou um tênis de corrida. A conexão desses dispositivos com a Rede é possível por meio de *hardwares* e protocolos de comunicação, como por exemplo o LoRa (What is Lora?, 2019), o qual é a abreviatura de *Long Range* e representa um protocolo de comunicação sem fio por rádio frequência considerado de baixo consumo, baixo custo, e com capacidade de transmissão para longas distâncias.

1.1 OBJETIVOS PRINCIPAIS

O objetivo principal desse trabalho centra-se na aplicação de IoT no monitoramento de painéis solares, obtendo-se dados, e transmitindo-os por meio do protocolo LoRa, de forma a ser possível aplicar modelos de previsão de energia elétrica baseados em métodos teóricos, numéricos e estatísticos. Essa previsão é seguida de uma comparação de valores previstos com valores reais de energia gerada, possibilitando detectar equipamentos que não estão funcionando corretamente, ou seja, tendo uma tensão e corrente gerada diferente do previsto. Tais falhas, no caso de painéis solares, podem ser resultantes de um possível curto-círcuito interno, do rompimento de algum condutor do equipamento ou até mesmo da obstrução da luz, fazendo sombras na superfície do módulo solar.

2 REVISÃO DA LITERATURA

A utilização de sensores integrados com o IoT é comum na grande maioria dos estudos recentes (AL-DAHOUD, *et al.*, 2015), (CRACIUNESCU, *et al.*, 2016), (GAROUDJA, *et al.*, 2017), tendo como objetivo obter remotamente parâmetros relevantes relacionados à geração de energia em placas solares e turbinas eólicas. Dentre os parâmetros estudados em placas solares, têm-se como principais a temperatura ambiente, temperatura dos módulos, corrente gerada de saída, tensão gerada de saída e a radiação solar (em W/m²).

Quando se trata da utilização dos parâmetros medidos, encontraram-se diversos trabalhos focados na monitoração desses dados, (AL-DAHOUD, *et al.*, 2015). Coletar os dados, transmiti-los para um banco de dados e tratá-los é uma prática comum dentre os estudos analisados. Um exemplo é o trabalho de Al-Dahoud (2015) onde foi monitorada a temperatura ambiente em volta de painéis solares, a radiação solar presente, a quantidade de poeira no painel, umidade e a corrente gerada de saída. Tais dados são transmitidos por meio de uma comunicação WSN ZigBee (ZigBee Wireless Technology Architecture and Applications) com um módulo XBee (Digi XBee Ecosystem) acoplado a um microcontrolador Arduino Mega (ARDUINO - MEGA, 2019), visando a detecção de possíveis problemas nos módulos solares.

Outro trabalho similar (CRACIUNESCU, *et al.*, 2016) coletou novamente informações de temperatura ambiente, radiação solar, corrente gerada de saída e tensão gerada de saída de um painel solar com o objetivo de tratar, modelar e realizar previsões, assim como a integração dessas análises em um possível *Smart Grid* (What is the SmartGrid?, 2019). Uma vez obtidos esses dados por sensores e transmitidos por uma comunicação WSN (*Wireless Sensor Networks*), foi proposto um modelo de tensão gerada pelo painel solar para detectar possíveis problemas no equipamento.

Utilizando os mesmos parâmetros citados, o trabalho de Garoudja (GAROUDJA, *et al.*, 2017) referente a detecção de falhas em sistemas fotovoltaicos faz o uso de métodos estatísticos e um modelo de placas solares chamado de *One Diode Model* (ODM) para comparar as saídas medidas e previstas pelo modelo e detectar os principais tipos de falhas em placas solares,

tais como: possíveis curtos-circuitos no *hardware* interno, circuitos em aberto referentes a danos nos condutores e componentes, e por último sombras na superfície das placas solares que resultam de diversos fatores, como nuvens passando pela região ou sujeira depositada na estrutura.

Para aplicar o conceito de IoT nas placas fotovoltaicas é importante, primeiramente, apresentar os fundamentos básicos do elemento monitorado, painel solar; da transmissão dos dados, que neste trabalho será via LoRa; e da modelagem e previsão com base nos dados coletados.

2.1 PAINEL SOLAR

Os painéis solares fotovoltaicos possuem a função de converter a energia dos fótons contidos na radiação solar em energia elétrica. Os painéis são constituídos, principalmente, de materiais semicondutores dopados quimicamente, capazes de absorver a energia do fóton que quebra as ligações químicas entre as moléculas presentes na estrutura, fazendo com que cargas elétricas sejam liberadas e então utilizadas para realizar trabalho (ZILLES, MACÊDO, *et al.*, 2012).

Atualmente as células dos painéis são fabricadas, na sua grande maioria, com lâminas de silício em formato circular ou quadrado contendo uma área entre 50 cm^2 a 150 cm^2 , possuindo uma tonalidade entre azul escuro e o preto. De acordo com ZILLES *et al.* (2012) em condições de sol de 1000 W/m^2 e temperatura da célula de $25\text{ }^\circ\text{C}$, as células mais utilizadas são capazes de gerar em potência máxima uma corrente de 32 mA/cm^2 , considerando as placas de 50 cm^2 e 150 cm^2 . Portanto temos o resultado de corrente de $1,5\text{ A}$ e $4,5\text{ A}$ respectivamente, e uma tensão que varia entre $0,46\text{ V}$ e $0,48\text{ V}$. Com isso, as células são agrupadas em associações série e paralelo a fim de obter níveis de corrente e tensão adequadas para a aplicação desejada.

Com base nos estudos citados acima, conclui-se que para o desenvolvimento do projeto, os principais parâmetros a serem monitorados pelos sensores são: corrente de saída, tensão de saída, temperatura ambiente, temperatura do painel e radiação solar (em W/m^2).

2.2 TECNOLOGIA IOT E LORA

IoT é uma das novidades desse século com aplicações em diversas áreas, como indústria, residências, fazendas, medicina; esse conceito se tornou conhecido e é sinônimo de tecnologia. O objetivo é estabelecer conexão entre objetos ou “coisas”. No caso deste trabalho, o intuito é obter informações em tempo real de variáveis e condições de funcionamento de painéis solares, armazenando as mesmas em um banco de dados (SANTOS, et al., 2017).

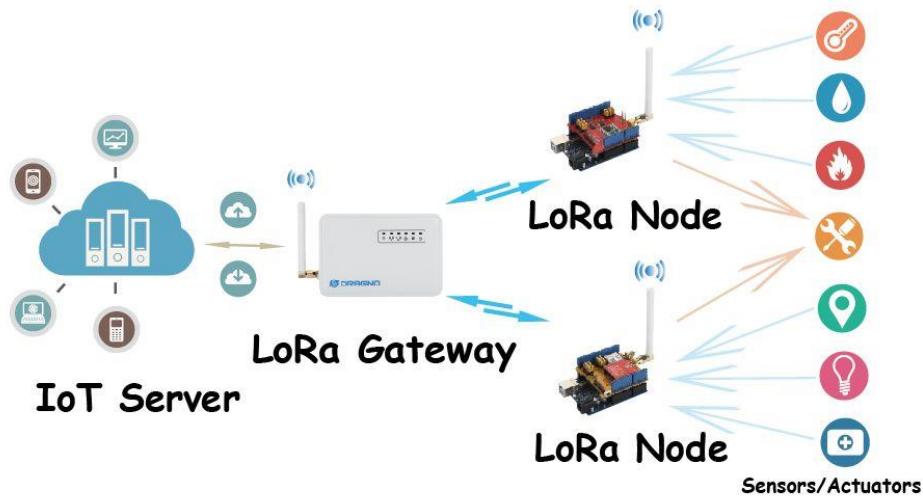
Dentro do conceito de IoT, existe a possibilidade de utilização de diversos protocolos de comunicação: *Mesh*, *Sigfox*, *Wi-Fi*, *Zigbee*, *3G*, *4G*, *5G*, *LoRa*, entre outros (NOVIDÁ, 2019). Cada um deles conta com vantagens e desvantagens, como distância de alcance de sinal, imunidade a obstáculos físicos, consumo de bateria, custo, velocidade, segurança, etc. Por consequência, cada aplicação tem o método de comunicação mais adequado a ser utilizado, de maneira que possa atender às necessidades específicas de cada projeto.

Neste trabalho, é utilizado o sistema LoRa cujo ponto forte é a capacidade de alcance de sinal: raio de 5 km em áreas urbanas e 15 km em áreas rurais (FILHO, SANTIN e YANG, 2017). Tal característica é importante para a aplicação estudada, uma vez que o monitoramento de fontes de energia renovável pode ser usado em pequenas fontes de geração, como em casas e locais comerciais, e também em grandes usinas solares e parques eólicos. Outra vantagem do LoRa é o baixo consumo energético, dispensando a conexão dos sensores a rede de energia, sendo viável o uso de baterias, e portanto, proporcionando a instalação em vários tipos de ambiente.

A comunicação usando LoRa permite velocidade entre 0,3 kbps até 50 kbps e garante segurança com criptografia. A baixa velocidade não é impeditiva para a aplicação neste trabalho, visto os longos intervalos de tempo entre transmissões e o baixo tráfego de dados.

O sistema LoRa é similar ao das redes de telefonia celular. Sensores conectados a um dispositivo de *hardware* (*LoRa Node*) fazem a leitura dos dados e transmitem a um *gateway*, que pode passar para outro, quando necessário, até que toda informação seja concentrada em um servidor central (*IoT Server*), como representado na Figura 2.

Figura 2 – Arquitetura LoRa



Fonte: (GUO, 2019)

2.3 MODELAGEM E PREVISÃO

Tratando-se de modelagem e previsão da energia gerada por painéis solares, encontra-se na literatura diferentes abordagens sobre o assunto. Alguns trabalhos optam por estimar a energia gerada por um equipamento a partir de redes neurais artificiais. Tem-se como exemplo o trabalho de Almonacid *et al.* (2009) e Dias (2015) onde é utilizada uma rede neural do tipo MLP (*Perceptron Multilayer*) (HAYKIN, 2003) para estimar a energia gerada em um painel fotovoltaico. A rede proposta neste trabalho possui uma camada escondida e utilizam-se como parâmetros de entrada a temperatura ($^{\circ}\text{C}$) do módulo fotovoltaico e a radiação solar (W/m^2), e como parâmetro de saída tem-se uma curva relacionando tensão e corrente gerada. Para treinar a rede foram usados dados reais medidos de temperatura e radiação solar e curvas reais de tensão por corrente, resultantes de radiações entre 200 e 1200 W/m^2 e temperaturas de 18 $^{\circ}\text{C}$ a 55 $^{\circ}\text{C}$. Fazendo uso de um algoritmo de retropropagação com Levenberg-Marquardt (HAYKIN, 2003) os pesos e *bias* da rede foram definidos por batelada, ou seja, somente depois de treinar completamente a rede.

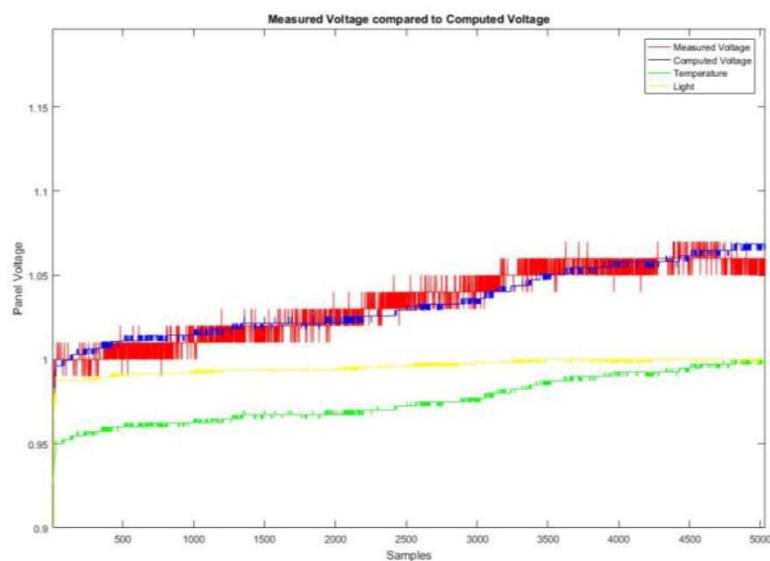
Para validação do treinamento utilizando dados previamente armazenados foram comparadas curvas reais de tensão por corrente com as curvas gerada pela rede neural, obtendo-se um baixo erro de 5,25 % no ano, considerado um resultado satisfatório de previsão do painel fotovoltaico.

Outra alternativa encontrada para previsão da geração de energia por placas solares foi obtida considerando-se uma relação linear da tensão em função da temperatura e da radiação solar sobre o painel (CRACIUNESCU, et al., 2016). Conforme Craciunescu (2016) a função descrita na equação (1) mostra que a saída em Volts (y) de uma placa fotovoltaica depende de uma constante de polarização (θ_0), e de outros dois parâmetros ajustáveis (θ_1 e θ_2) referentes à temperatura (x_1) e à radiação solar (x_2).

$$y = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 \quad (1)$$

Considerando-se que a equação (1) têm três incógnitas e três variáveis conhecidas, foi elaborada no estudo uma solução analítica utilizando-se de um sistema matricial, no qual são definidas as constantes desconhecidas da relação linear primordialmente proposta. No estudo de Craciunescu, a partir de um banco de dados de amostras de tensão, temperatura e radiação solar, resolveu-se o sistema matricial e encontrou-se valores para a constante de θ_0 e os outros dois parâmetros θ_1 e θ_2 . Para comparar os resultados, observa-se na Figura 3 as curvas plotadas de uma amostra de 5000 tensões medidas e das suas respectivas tensões computadas pela linearização a partir das temperaturas e radiações solares também medidas.

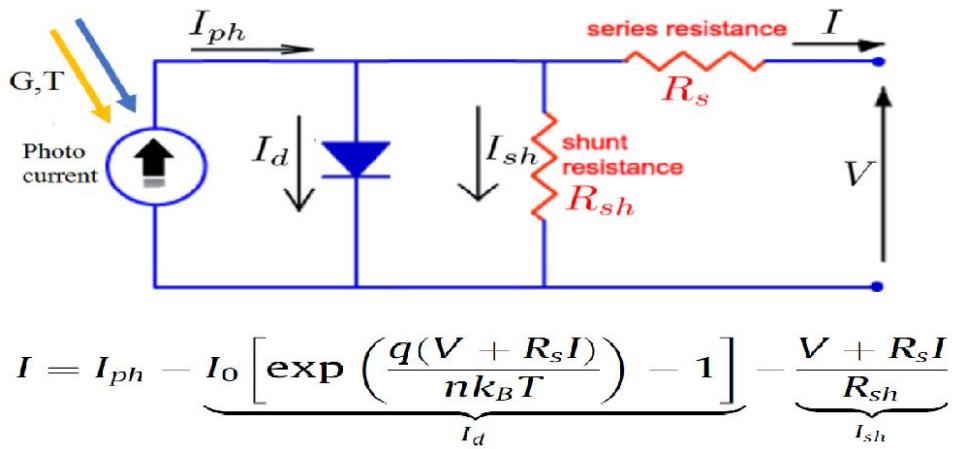
Figura 3 – Curvas da tensão medida pela tensão compactuada



Fonte: CRACIUNESCU, 2016.

Outro modelo da literatura para a previsão de energia produzida por uma célula fotovoltaica é o ODM, do inglês *One-Diode Model* (AUGUSTIN MCEVOY, 2012). O ODM (Figura 4), é um modelo muito utilizado em placas solares e é baseado em uma corrente gerada em paralelo com um diodo com resistências em série e paralelo, as quais representam perdas resistivas.

Figura 4 – *One-Diode Model*



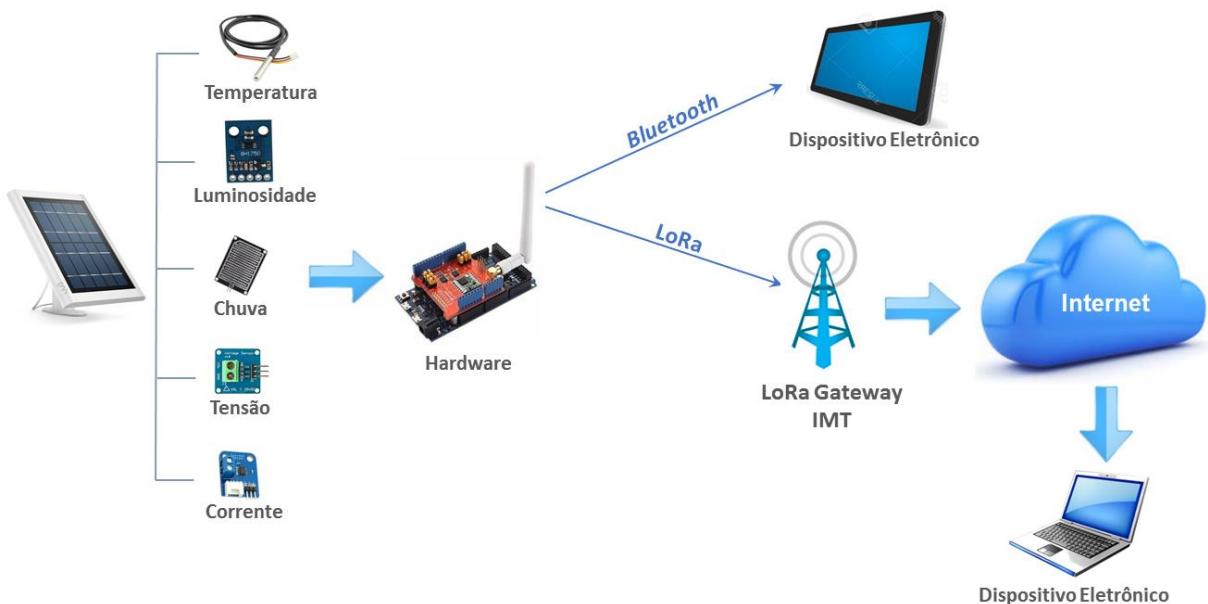
Fonte: GAROUDJA, 2017.

O ODM depende de cinco parâmetros indeterminados, os quais normalmente não são informados nas descrições técnicas apresentadas pelo fabricante: corrente fotogerada (I_{ph}), corrente de saturação (I_0), fator ideal do diodo (n) e resistências de série e *shunt* respectivamente (R_s e R_{sh}). Para determinar esses parâmetros tem-se na literatura a utilização de um algoritmo denominado *Artificial Bee Colony* (ABC), o qual extrai os valores dos parâmetros a partir de amostras medidas de tensão e corrente gerada por uma célula fotovoltaica. Com o modelo totalmente determinado, é feita uma comparação entre os valores de tensão e corrente medidos e os previstos pelo modelo, sendo utilizados métodos estatísticos para monitorar e controlar a diferença entre os parâmetros medidos e previstos, possibilitando a detecção e até classificação de uma possível falha no equipamento ou sistema (GAROUDJA, et al., 2017).

3 MATERIAIS E MÉTODOS

Com o intuito de monitorar um painel fotovoltaico foi desenvolvido um dispositivo que, através de sensores, coleta informações como temperatura ambiente, radiação solar, presença de chuva, tensão e corrente gerada pela placa solar. Os dados coletados são processados pelo *hardware* e enviados à um *gateway LoRa* do Instituto Mauá de Tecnologia e / ou a um dispositivo eletrônico via *Bluetooth*, conforme ilustrado na Figura 5.

Figura 5 – Diagrama Geral do Sistema de Monitoramento do Painel Fotovoltaico



Fonte: Os Autores, 2019.

No diagrama apresentado na Figura 5, observa-se que em um painel fotovoltaico de 1000 W/m² e 95 W foram instalados sensores de temperatura (Maxim Integrated DS18B20), chuva (Vaisala YL-83), tensão (P25-GBK 25V DC), corrente (Allegro MicroSystems ACS712) e luminosidade (Rohm Semiconductor BH1750FVI), sendo esse último convertido em radiação, conforme descrito no item 3.2.1.

Os sensores foram conectados a um dispositivo de coleta de dados denominado *hardware*. Tal dispositivo é constituído de um Arduino Mega associado a uma placa de comunicação LoRa (*Dragino LoRa Shield*) e comunicação *Bluetooth* (HM-10 4.0 BLE); a um dispositivo de gravação de dados em cartão de memória (SD Card Arduino); ao *Real Time Clock*; e a um sistema de gerenciamento de baterias (BMS), recarregável pelo próprio painel, desenvolvido

para suprir o dispositivo de coleta. Dessa forma, os dados coletados e processados podem ser acessados de três formas distintas:

- através de um cartão de memória interno ao equipamento (armazenamento local);
- via interface *Bluetooth*, utilizando um aplicativo instalado em um *smartphone* ou *tablet* (acesso remoto de curta distância);
- ou por meio do sistema de comunicação LoRa, onde os dados são enviados a um *gateway LoRa* (acesso remoto de longa distância).

Uma vez que o *gateway LoRa* esteja conectado à Internet, torna-se possível enviar os dados para um servidor, ou central de supervisão, onde as informações coletadas são apresentadas em um *dashboard* e comparadas com os valores esperados.

Para se obter os valores esperados, foram coletados dados de temperatura ambiente, radiação, tensão e corrente de um painel fotovoltaico durante seu funcionamento, treinando uma Rede Neural Artificial (RNA) que aprendeu o comportamento da placa solar e toda sua dinâmica, prevendo os valores de tensão e corrente com base na temperatura e radiação medidas. Dessa forma, a central de supervisão compara os valores de tensão e corrente medidos com os previstos, e caso a diferença seja superior a 10 %, emite-se um alerta de verificação do sistema de geração.

3.1 DESENVOLVIMENTO DO HARDWARE

Para monitorar os painéis solares, com sucesso, utilizando a filosofia da tecnologia IoT garantindo sua robustez, confiança, e a precisão dos dados coletados, interpretados e mostrados pelo sistema, foi desenvolvido um *hardware* capaz de suportar as intempéries climáticas da natureza, uma vez que ele estará exposto ao sol, chuva, vento e variações de temperatura.

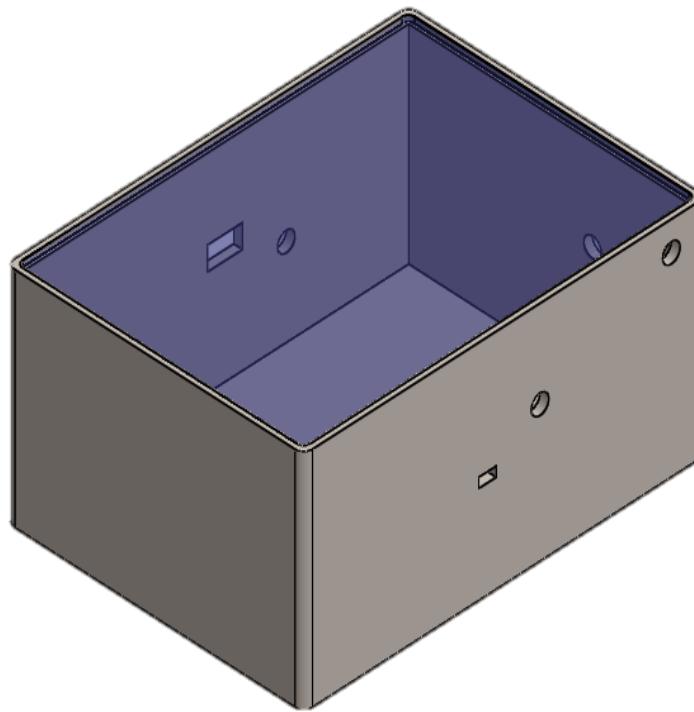
Antes de se dar início ao desenvolvimento do *hardware* foi preciso apurar se os sensores estavam de acordo com a aplicação. Para comprovar esta hipótese, todos os sensores e módulos, antes de serem montados na placa de fenolite, foram testados e calibrados.

As análises preliminares de funcionamento do sistema iniciaram-se ao testar cada sensor, individualmente, utilizando uma *protoboard*, verificando se as leituras estavam condizentes. Os sensores testados foram: sensor de chuva, temperatura, luminosidade, corrente e tensão, assim como os módulos: *Real Time Clock*, *Bluetooth* e *SD Card*. Para isso, foi realizada a programação em Arduino de cada componente.

Com os sensores calibrados e testados, sem erros de leitura e sem problemas encontrados nos componentes foi dado sequência a montagem do *hardware*, fixando e conectando os componentes na placa de fenolite.

Em uma impressora 3D foi desenvolvido um invólucro, conforme ilustrado na Figura 6, para proteger todos os módulos e sensores que não ficariam expostos. O material utilizado foi o ABS, por não ser biodegradável, ter rigidez considerável e ser a prova d'água.

Figura 6 – Invólucro desenvolvido para abrigar o *hardware*



Fonte: Os Autores, 2019.

O invólucro fica alocado atrás do painel, conforme ilustrado na Figura 7, e preso por velcros, pois são de fácil instalação e não requerem quaisquer intervenções físicas na placa, tornando-se mais fácil a retirada do módulo para eventuais manutenções.

Figura 7 – Invólucro alojado atrás do painel solar



Fonte: Os Autores, 2019.

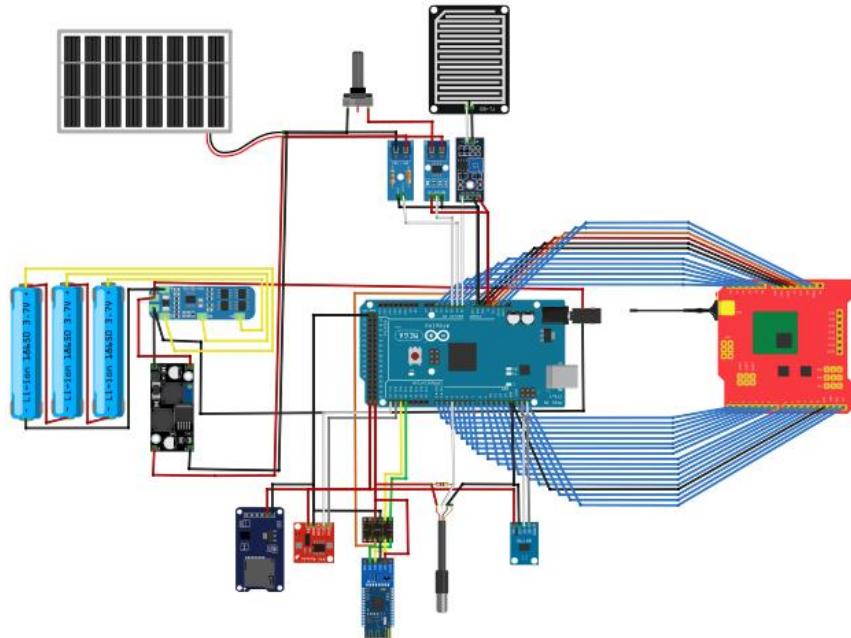
Os sensores que ficam expostos são os de luminosidade (BH1750FVI) e chuva (YL-83), localizados na parte frontal do painel de forma que não prejudique o recebimento da energia solar, e o sensor de temperatura (DS18B20) que fica encostado no lado de trás do painel. Já os módulos *Bluetooth*, *Dragino*, *SD card*, *Real-Time-Clock* (RTC), Arduino Mega e os sensores de tensão e corrente estão localizados dentro do invólucro.

Foram feitas adaptações nos conectores do circuito junto ao cabo do painel, para que pudessem ser ligados aos sensores de tensão e corrente. Os cabos foram conectados a uma extensão adaptada, criada para que eles se encaixassem corretamente. Os conectores do painel são do tipo MC4 (STÄUBLI GROUP, 2013) enquanto os dos sensores são bornes para fios de até 1 mm², onde foi feito um fio ramificado para estes serem atendidos.

A partir do conector MC4 do painel conecta-se o ramo enquanto na outra ponta dispõem-se de conectores tipo banana que são plugados na placa e destes, estendidos fios até os bornes dos sensores.

Desta forma, foi desenvolvido o esquema ilustrado na Figura 8 com os componentes a serem melhor descritos no item 3.1.1.

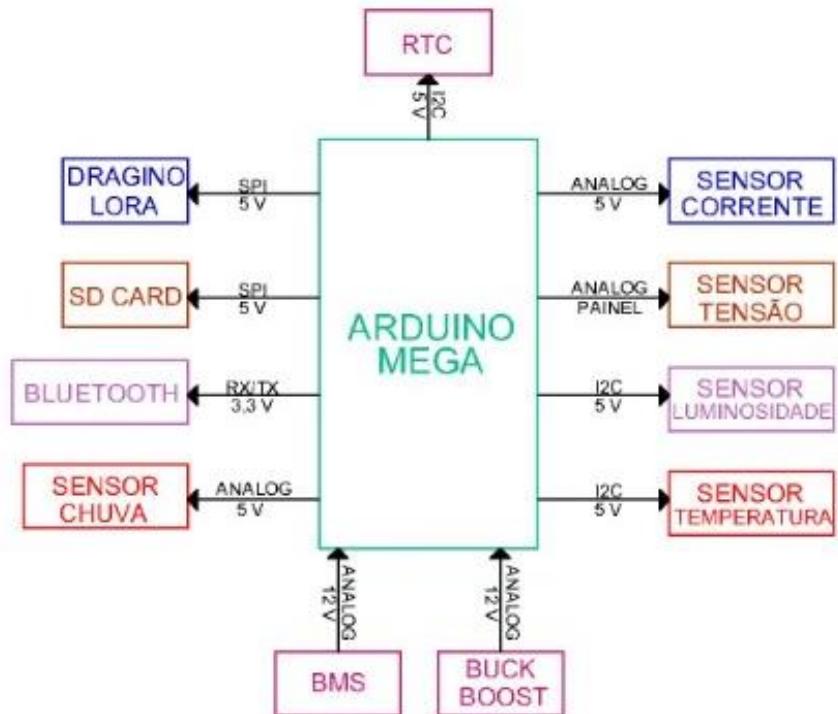
Figura 8 – Esquema elétrico elaborado na *protoboard*



Fonte: Os Autores, 2019.

O módulo Arduino Mega é responsável pela lógica de programação, coleta e administração dos sensores do sistema, e está conectado com os mesmos conforme ilustrado na Figura 9. O código implementado no módulo está explicado no item 3.2.1.

Figura 9 – Esquema de ligação do Arduino com os sensores



Fonte: Os Autores, 2019.

Os dados coletados pelos sensores são armazenados por um cartão SD de 8 GB, podendo ser trocado de acordo com a necessidade. O cartão pode ser inserido em um computador para os dados armazenados serem lidos em uma planilha eletrônica. Também são transmitidos via *Bluetooth* para o aplicativo do *smartphone*, onde pode-se monitorar individualmente cada painel em tempo real, para efeitos de manutenção ou alguma checagem rotineira. Também são transmitidos via rádio pelo módulo *Dragino*, utilizando o protocolo LoRa de comunicação, ao sinótico desenvolvido em Node-RED, melhor descrito no item 3.2.3, onde é gerada a mensagem de atenção às falhas previstas pelo sistema.

Para a energização do sistema, a própria placa solar fornece energia para o mesmo. Para isso foram utilizados dois componentes: BMS e *Buck-Boost*, pois a potência gerada pelo painel depende da radiação e da temperatura do ambiente, variáveis intermitentes, e como não se pode controlá-las, foram adotados os referidos componentes para manter constante o carregamento de baterias locais. O Sistema Gerenciador de Baterias (*BMS - Battery Management System*) é o responsável por gerenciar a carga de tais baterias.

Pensando-se na autonomia do sistema, caso o sistema de transmissão de dados via rádio seja configurado para enviar os dados em uma taxa de amostragem de um segundo, sem ter a incidência da luz solar na placa, para que seja feita a recarga da bateria, ou seja, com o sistema ligado somente à bateria, sem qualquer tipo de recarga, o sistema permanece por volta de 20 horas em funcionamento.

A autonomia da bateria pode ser aumentada à medida que se configure a taxa de envio de dados pelo *Dragino*. Quando o sistema envia dados é consumido 300 mA, em média, já em condição de descanso, apenas realizando a coleta de dados, 100 mA. Para melhor desempenho energético o ideal é configurar a amostragem dos avisos de manutenção, sendo enviados apenas quando houver uma previsão de falha, pois assim a potência consumida se torna menor, economizando energia do sistema.

O sistema contém um período de operação configurável, das 06h00 às 18h00. No período noturno, o sistema permanece em repouso, para poupar energia para o dia seguinte.

Obviamente o horário de operação muda em período de outono-inverno e primavera-verão, este podendo ser configurado via *software* do sistema.

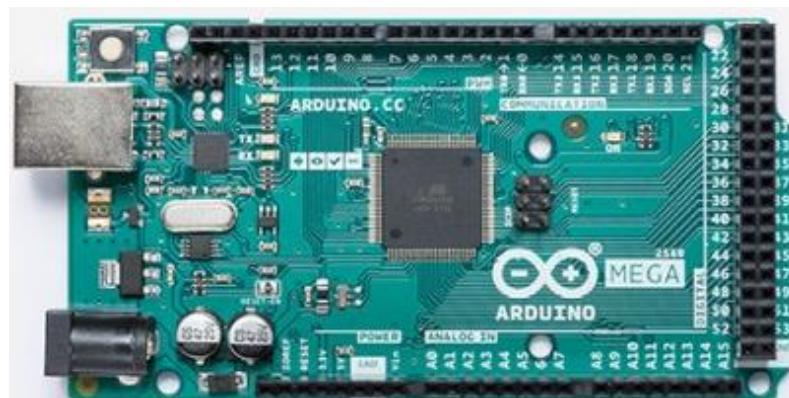
3.1.1 MÓDULOS E SENSORES UTILIZADOS

A seguir serão apresentados os módulos e os sensores utilizados para o desenvolvimento do protótipo. Os componentes utilizados contêm precisão o suficiente para apurar os dados requeridos e avaliar seus atributos em relação ao módulo fotovoltaico.

3.1.1.1 Arduino

O Arduino Mega 2560, ilustrado na Figura 10, foi escolhido para ser o controlador dos periféricos do projeto por ser uma plataforma de prototipagem eletrônica *open-source* que se baseia em *hardware* e *software* flexíveis e fáceis de usar (ARDUINO - MEGA, 2019).

Figura 10 – Arduino Mega



Fonte: (ARDUINO - MEGA, 2019)

O Arduino pode sentir o estado do ambiente que o cerca por meio da recepção de sinais de sensores e pode interagir com os seus arredores, controlando luzes, motores e outros atuadores. Os projetos desenvolvidos com o Arduino podem ser autônomos ou podem comunicar-se com um computador para a realização da tarefa.

Especificações:

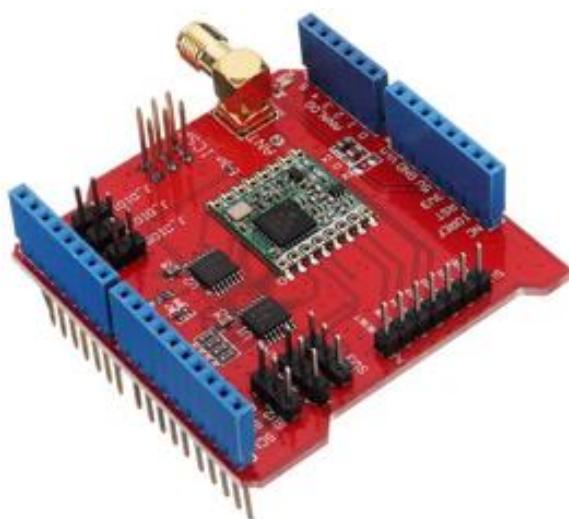
- Microcontrolador: ATmega2560;
- Tensão de operação: 5 V;

- Tensão de Entrada (nominal): 7 a 12 V;
- Tensão de Entrada (limite): 6 a 20 V;
- Pinos Digitais I/O: 54 (dos quais 15 podem ser sinais PWM);
- Pinos de Entradas Analógicas: 16;
- Corrente (CC) por pino I/O: 20 mA;
- Corrente (CC) para o pino de 3,3V: 50 mA;
- Memória *Flash*: 256 KB das quais 8 KB são usadas como *bootloader*;
- SRAM: 8 KB;
- EEPROM: 4 KB;
- *Clock Speed*: 16 MHz;

3.1.1.2 Módulo de Comunicação LoRa

Para comunicação de longas distâncias foi utilizado o Dragino LoRa Shield 915 MHz, ilustrado na Figura 11, por ter toda a tecnologia LoRa *wireless* de longo alcance, disponível para o Arduino enviando e recebendo dados de forma confiável e segura. O baixo consumo permite que o módulo LoRa seja alimentado por baterias e funcione por longos períodos sem necessidade de recarga, usando funções como hibernação para ativar o dispositivo apenas em horários específicos (WIKI.DRAGINO, 2019).

Figura 11 – Dragino LoRa Shield



Fonte: (WIKI.DRAGINO, 2019)

Especificações:

- Shield LoRa 915 MHz para Arduino v1.4;
- Compatível com placas Arduino com nível de sinal de 3,3 e 5 Vcc;
- Frequência: 915 MHZ;
- Baixo consumo de energia;
- Compatível com Arduino Uno, Mega, Leonardo, Due;
- Conector SMA para antena externa.

Especificações RFM95W:

- Constante de saída RF: +20 dBm – 100 mW;
- Potência: +14 dBm;
- Bit rate programável até 300 kbps;
- Alta sensibilidade: -148 dBm;
- Excelente imunidade contra interferências;
- Baixa corrente de recepção: 10,3 mA;
- Modulações FSK, GFSK, MSK, GMSK, LoRa TM and OOK;
- Sensor de temperatura e de bateria baixa embutidos.

3.1.1.3 Módulo de Comunicação *Bluetooth*

O módulo Bluetooth HM-10 4.0 BLE, ilustrado na Figura 12, foi escolhido por ser compatível com iOS e Android, foi desenvolvido para enviar e receber informações via *Bluetooth*, sendo compatível com plataformas de prototipagem Arduino, Raspberry PI, ARM, AVR e PIC. O módulo permite que o dispositivo envie ou receba dados através de tecnologia *Bluetooth* sem que seja necessário conectar um cabo serial no computador. Capaz de trabalhar em dois modos diferentes: *Master* (mestre) e *Slave* (escravo). Cada um dos dois modos possui diferentes finalidades, o modo mestre é utilizado para se conectar a outros dispositivos, já o modo escravo serve apenas para receber conexões (ARDUINO - HM-10, 2019).

Figura 12 – Módulo de Bluetooth



Fonte: (ARDUINO - HM-10, 2019)

Ao utilizar-se o *Bluetooth* junto de equipamentos com sistema iOS e Android, o mesmo necessita de um aplicativo para o reconhecimento e emparelhamento entre os mesmos, tornando necessário, neste caso, a utilização de um aplicativo para leitura e/ou inserção de dados, para se usufruir de todos os seus benefícios.

Especificações:

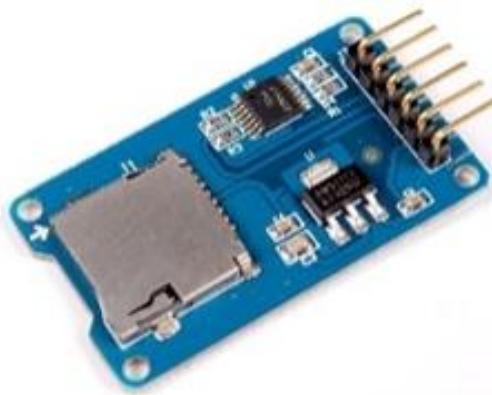
- CI: CC2541;
- Alimentação: 3,3 Vcc;
- Versão: Bluetooth 4.0 BLE;
- Frequência de trabalho: Banda ISM de 2,4 GHz;
- Alcance: 10 metros em área aberta;
- Método de modulação: GFSK (*Gaussian Frequency tecla Shift Keying*);
- Segurança: Autenticação e criptografia;
- Fonte de alimentação: 3,3 Vcc;
- TX potência: 0 dBm;
- Sensibilidade de RX: -93 dBm;
- Transmissão: dBm, 23-6 dBm, 0 dBm e 6 dBm;
- Protocolo de comunicação: Uart (TTL);
- Taxa de transmissão padrão: 115.200 bps;
- Corrente de transmissão: 15 mA;
- Receber corrente: 8,5 mA;
- Corrente de sono profundo: 600 µA;

- Banda de frequência: 2.402 GHz a 2.480 GHz;
- Taxa Máxima de Dados: 1 Mbps;
- Impedância de entrada de RF: 50 Ω;
- *Crystal OSC* de banda base: 16 MHz;
- Interface: UART;
- Sensibilidade: -90 dBm;
- Temperatura de funcionamento: -40 a +65 °C
- Senha padrão (PIM): 000000 ou 123456.

3.1.1.4 Módulo SD Card

O módulo *SD Card*, ilustrado na Figura 13, é utilizado para armazenar os dados coletados e, posteriormente, visualizá-los em uma planilha eletrônica. Possui interface de comunicação SPI (FILIPEFLOP - MÓDULO SD CARD, 2019). Por ser pequeno e de fácil instalação, foi escolhido para o projeto.

Figura 13 – Módulo *SD Card*



Fonte: (FILIPEFLOP - MÓDULO SD CARD, 2019)

Especificações:

- Nível lógico: 3,3 V (divisor de tensão já embutido no módulo);
- Tensão de operação: 4,5 a 5,5 Vcc;
- Interface de comunicação: SPI;
- Cartões compatíveis: micro *SD Card* / *SDHC*.

3.1.1.5 Real-Time-Clock (RTC)

Para se registrar as datas e horários dos dados coletados, utilizou-se o *Real-Time-Clock* DS1307 da Maxim Integrated, conforme ilustrado na Figura 14. Por ter interface de comunicação I2C (FILIPEFLOP - RTC, 2019) permite que seja facilmente instalado no protótipo.

Figura 14 – Real-Time-Clock (RTC)



Fonte: (FILIPEFLOP - RTC, 2019)

Especificações:

- Módulo: DS1307;
- Computa segundos, minutos, horas, dias da semana, dias do mês, meses e anos (de 2000 a 2099);
- Interface I2C;
- Circuito de detecção de falha de energia;
- Consome menos de 500 nA no modo bateria com oscilador em funcionamento;
- Faixa de temperatura: -40 °C a +85 °C.

3.1.1.6 Módulo Regulador de Tensão

O módulo regulador de tensão, *Buck-Boost*, ilustrado na Figura 15, foi utilizado para elevar a tensão de entrada, de 3 V a 12 V, para uma tensão de saída na faixa de 12 V a 32 V (ELETROGATE, 2019), para que em dias nublados, ele consiga dar continuidade no carregamento das baterias.

Figura 15 – Módulo Regulador de Tensão



Fonte: (ELETROGATE, 2019)

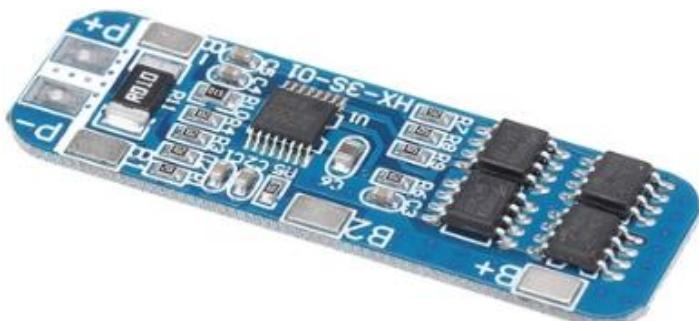
Especificações:

- Faixa de tensão de entrada: 3 V a 32 V;
- Faixa de tensão de saída: 5 V a 35 V;
- Suporta corrente de 4 A (utilizar dissipador, caso haja a necessidade) e a eficiência pode chegar a até 94 %;
- Ultra alta frequência de comutação até 400 KHz;
- A Frequência do LM2577 é apenas 50 KHz.

3.1.1.7 Módulo de Gerenciamento de Baterias

Para gerenciar o carregamento das baterias, foi utilizado o módulo 3S-12V-10A-18650 da OEM, conforme ilustrado na Figura 16. O módulo tem uma descarga máxima de 10 A e é capaz de prevenir as baterias contra sobre e sub-tensão, sobre e sub-corrente, além de eventuais explosões (GROBOTRONICS, 2019).

Figura 16 – Módulo de Gerenciamento de Baterias



Fonte: (GROBOTRONICS, 2019)

Especificações:

- Descarga de corrente máxima: 10 A;
- Sobrecarga de tensão de detecção: 3,9 a 4,35 V \pm 0,05 V;
- Sobre-descarga de tensão de detecção: 2,3 a 3,0 V \pm 0,05 V;
- Máxima de trabalho atual: 5 A a 8 A;
- Transiente atual: 9 A a 10 A;
- Temperatura de operação:-40 a 50 °C;
- Temperatura de armazenamento:-40 a 80 °C;
- Corrente de repouso: inferior a 6 μ A;
- Resistência interna: inferior a 60 m Ω ;
- Vida útil: superior a 30.000 horas;
- Proteção contra curto circuito;
- Proteção da sobrecarga;
- Proteção sobre-descarga;
- Proteção de sobrecorrente;
- Indicado para:
 - 10,8 V (tensão Nominal da bateria de polímero);
 - 11,1 V (18650 ou 3,7 V tensão nominal da bateria de lítio);
 - 12,6 V (bateria de Lítio tensão de carga completa).

3.1.1.8 Baterias

Para manter o sistema operando mesmo durante a ausência de radiação solar são utilizadas três baterias 18650 da Samsung. A bateria 18650, conforme ilustrada na Figura 17, é uma célula de 18 mm por 65 mm que oferece o desempenho de uma célula de íons de lítio, tendo uma capacidade na faixa de 1800 mAh a 3500 mAh e uma saída de 3,7 V (SAMSUNG, 2019).

Figura 17 – Baterias 18650



Fonte: (TACBATTERY, 2019)

As baterias 18650 foram utilizadas no projeto como fonte de energia armazenada, pela sua capacidade de suportar a demanda de energia utilizada pelo *hardware*.

3.1.1.9 Sensor de Temperatura

Um sensor DS18B20, da Maxim Integrated, foi utilizado para medir a temperatura atingida pelo painel, pois permite faixa de medição de -55 °C a +125 °C (ELEKTRONIKLAVPRIS, 2019), oferecendo boa precisão e fácil instalação no módulo solar. O sensor de temperatura DS18B20 é ilustrado na Figura 18.

Figura 18 – Sensor de Temperatura DS18B20



Fonte: (ELEKTRONIKLAVPRIS, 2019)

Especificações:

- À prova d'água, permite fazer medições em ambientes úmidos e molhados;
- Precisão: $\pm 0,5$ °C entre -10 °C e +85 °C;
- Leituras de temperatura de até 12-bits (configurável);

- Tensão de operação: 3 a 5,5 V;
- Faixa de medição: -55 °C a +125 °C;
- Ponta de aço inoxidável;
- Interface I2C.

3.1.1.10 Sensor de Luminosidade

Com o intuito de se obter a radiação solar incidente sobre o painel, foi utilizado um sensor de luminosidade cujos valores obtidos são convertidos em radiação, conforme descrito no item 3.2.1.

O sensor utilizado foi o BH1750FVI da Rohm Semiconductor, vide Figura 19, uma vez que este permite leituras de luminosidade de 1 a 65.535 lux, estando dentro da variação desejada. A interface de comunicação com o microcontrolador é a I2C (FILIPEFLOP - BH1750, 2019), o que facilita o processo de conexão e configuração.

Figura 19 – Sensor de Luminosidade BH1750FVI



Fonte: (FILIPEFLOP - BH1750, 2019)

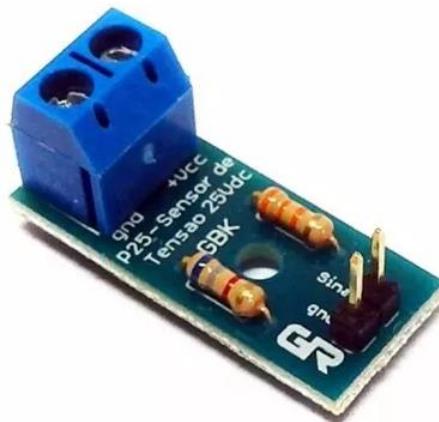
Especificações:

- Módulo sensor de luminosidade GY-302;
- Tensão de operação: 3 a 5 Vcc;
- Faixa de medição: 1 a 65.535 lux;
- Interface: I2C;
- Conversor AD de 16 bits.

3.1.1.11 Sensor de Tensão

O sensor P25-GBK 25V DC é do tipo analógico, sendo uma associação de resistores de $33\text{ k}\Omega$ e $6,8\text{ k}\Omega$ (USAINFO, 2019), conforme ilustrado na Figura 20. Ele é usado para medir a tensão gerada pelo painel, pois permite leituras de tensão de 0 a 25 V, estando dentro da variação de tensão da placa fotovoltaica.

Figura 20 – Sensor de Tensão P25-GBK 25V DC

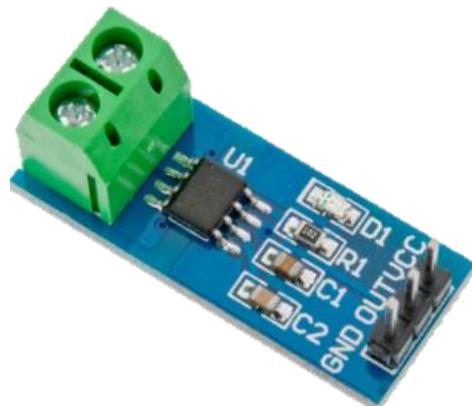


Fonte: (USAINFO, 2019)

3.1.1.12 Sensor de Corrente

Para medir a corrente gerada pelo painel, foi utilizado o sensor ACS712 da Allegro MicroSystems, conforme ilustrado na Figura 21. O sensor permite leituras de corrente de 0 a 5 A (ELECTROKIT, 2019), estando dentro da variação de corrente da placa fotovoltaica.

Figura 21 – Sensor de Corrente ACS712



Fonte: (ELECTROKIT, 2019)

Especificações:

- Sensor analógico;
- Leitura: de 0 a 5 A (CA e CC);
- Saída: tensão linear entre 0 - VCC, centralizada em VCC / 2 em 0 A;
- Tensão de alimentação: 5 Vcc;
- Sensibilidade: 185 mV / A;
- Sensor: ACS712ELC-05B.

3.1.1.13 Sensor de Chuva

O sensor YL-83 da Vaisala, ilustrado na Figura 22, é utilizado para medir se está chovendo ou não sobre o painel.

Figura 22 – Sensor de Chuva YL-83



Fonte: (FILIPEFLOP - SENSOR DE CHUVA, 2019)

Este sensor pode ser usado para monitorar condições climáticas como gotas de chuva ou neve. Quando o clima está seco a saída do sensor permanece em estado alto e quando há uma gota de chuva sobre ele, a saída permanece em estado baixo (FILIPEFLOP - SENSOR DE CHUVA, 2019).

O limite entre tempo seco e chuva pode ser ajustado através do potenciômetro presente no sensor que regula a saída digital D0, contudo para se ter uma resolução melhor foi utilizado a saída analógica A0 e conectado a um conversor AD.

Especificações:

- Sensor puramente resistivo;
- Tensão de Operação: 3,3 a 5 V;
- Corrente de Saída: 100 mA;
- Sensibilidade ajustável via potenciômetro;
- Saída Digital e Analógica;
- Comparador LM393.

3.1.1.14 Painel Fotovoltaico

Durante os testes explicitados no item 4 - Resultados e discussão, utilizou-se a placa solar YL095P-17b de 95W da Yingli, conforme ilustrado na Figura 23. O painel fotovoltaico, disponível no Instituto Mauá de Tecnologia, possui células do tipo Silício Policristalino (YINGLI, 2019) e sua principal aplicação é em sistemas OFF-Grid com o uso de baterias, em locais que não tem energia para alimentação de lâmpadas, torres de internet e iluminação.

Figura 23 – Painel Fotovoltaico



Fonte: (YINGLI, 2019)

Especificações:

- Potência máxima (Pmax): 95 W;
- Tolerâncias de saída de potência: +/- 5 %;

- Eficiência do módulo: 14,20 %;
- Tensão de máxima potência (V_m): 18,20 V;
- Corrente da máxima potência (I_m): 5,23 A;
- Tensão de circuito aberto (V_{oc}): 22,5 V;
- Corrente do curto-círcuito (I_{sc}): 5,59 A;
- Tipo de células: Silício Policristalino;
- Condições de teste padrão: 1000 W/m² radiação, 25 °C temperatura do módulo;
- Coeficiente de temperatura ($P_{máx}$) -0,45 % / °C;
- Coeficiente de temperatura (V_{oc}) -0,37 % / °C;
- Coeficiente de temperatura (I_{sc}) 0,06 % / °C;
- Temperatura operacional nominal da célula 46 ± 2 °C;
- Peso bruto: 8,45 Kg;
- Dimensão: 2,5 cm x 66 cm x 101 cm (Altura x Largura x Comprimento);
- NCM: 85414032.

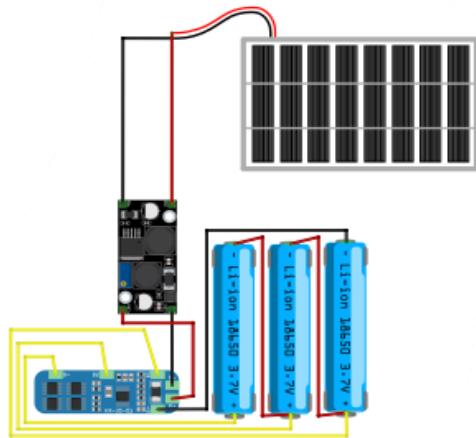
3.1.2 BATTERY MANAGEMENT SYSTEM (BMS)

Com o intuito de tornar o sistema autossustentável, ou seja, sem depender de outras fontes de alimentação para suprir a energia necessária do projeto, além do próprio painel solar em que foi aplicado, e também a fim de preservar a vida útil das baterias responsáveis pela alimentação do circuito eletrônico, utilizou-se um circuito conversor DC-DC, suprido pelo painel solar, em conjunto com um BMS (*Battery Management System* – Sistema de Gerenciamento de Baterias).

Durante a operação do sistema, as baterias são responsáveis por fornecer a energia elétrica necessária ao circuito, ou seja, elas estão em constante uso e então passam a se descarregar conforme são requisitadas, sendo necessário recarregá-las. Sabe-se que elas não são descarregadas de forma idênticas, uma célula pode se descarregar mais que outra, além disso, as baterias podem enfrentar alguns problemas durante o processo de carregamento, tais como sobre-carregamento, carregamento desbalanceado das células que as compõem e sobreaquecimento, podendo ser danificadas ou até mesmo sofrer uma explosão. Portanto, é necessário um processo de carregamento muito bem gerenciado.

O circuito conversor DC-DC é responsável por abaixar a tensão fornecida pelo painel solar em condições de alta radiação para 12 V, ou então aumentar a tensão para o mesmo valor quando temos uma condição de geração de tensão menor que 12 V, esta tensão é a necessária para o processo de carregamento do conjunto de células da bateria. Já o circuito BMS é a interface entre o conversor e as baterias, é responsável por fazer o gerenciamento do processo de carregamento de forma individual sobre cada célula, controlando a tensão e corrente de carregamento, o momento em que se deve parar de carregar e também evita que o nível de carga fique abaixo do mínimo permitido. No instante em que uma célula se carrega primeiro que as outras, o gerenciador simplesmente para de carregar esta e permanece carregando as demais, até que todas estejam devidamente carregadas. Dessa forma, o BMS prolonga a vida útil das baterias, que seria consideravelmente menor caso não houvesse esse processo de carregamento gerenciado (ENGINEERING.COM, 2019). A Figura 24 ilustra este circuito.

Figura 24 – Esquema elétrico do circuito BMS



Fonte: Os Autores, 2019.

Após a finalização da montagem do circuito BMS, foi feita sua conexão com o restante do circuito eletrônico a fim de analisar o comportamento do gerenciador e das condições de geração de energia durante o carregamento das células da bateria. Os primeiros testes apontaram que o BMS estava cumprindo sua tarefa de carregar as células. Entretanto, ao se aprofundar nesta constatação, verificou-se que no decorrer do carregamento, as condições de geração do painel eram consideravelmente alteradas, isto é, os valores de tensão gerada caiam devido à baixa potência fornecida pelo painel utilizado para fins acadêmicos. Para solucionar tal problema seria necessário um aperfeiçoamento do sistema BMS, ou então,

utilizar outro painel fotovoltaico com capacidade de geração de potência maior e certificarse de que ele seria capaz de suprir a energia necessária para o carregamento das baterias e para alimentação das cargas do sistema.

3.2 DESENVOLVIMENTO DO SOFTWARE

3.2.1 FIRMWARE DO DISPOSITIVO DE COLETA DE DADOS

O *software* elaborado inicialmente, visava o funcionamento primitivo do sistema. Portanto, antes de se iniciar a programação propriamente dita, faz-se necessário instalar e configurar as bibliotecas do Arduino, conforme explicitado no Apêndice A – Instalação das bibliotecas do Arduino.

Posteriormente, partiu-se da criação das variáveis, organização das configurações de cada sensor e módulo, organização do código e dos valores lidos, assegurando que o *software* e o *hardware* estariam funcionando de acordo com o planejado, mas sem medidas de segurança redundantes para garantir a integridade dos valores lidos, apenas se estaria tudo funcionando e posteriormente estas providências seriam adicionadas na programação.

Nesta etapa foi verificado se as configurações estavam certas e os dados coerentes, com ajustes de alguns parâmetros. O *firmware* desenvolvido está disponível no Apêndice C – Código fonte do dispositivo coletor.

Conforme mencionado anteriormente, um sensor de luminosidade (Rohm Semiconductor BH1750FVI) foi utilizado. Dessa forma, faz-se necessário converter os valores lidos de iluminância (lux) para radiação (W/m^2). De acordo com Pires (2017), para se realizar essa conversão, utiliza-se a equação (2).

$$\text{iluminância (lux)} = \frac{\text{radiação} (W/m^2)}{0,0079} \quad (2)$$

Da equação (2), para transformar os valores de iluminância em radiação, basta dividir por 126,7, conforme indicado na equação (3).

$$\text{radiação} \left(W/m^2 \right) = \frac{\text{iluminância (lux)}}{126,7} \quad (3)$$

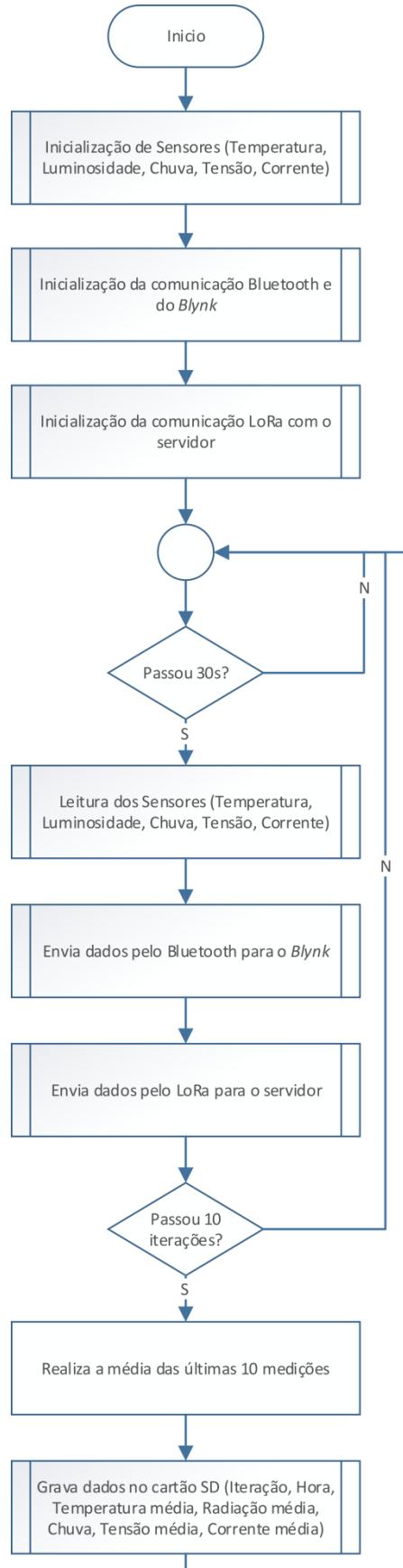
Os valores captados para avaliação de falha prevista pela rede neural e pelo *software* são tensão, corrente, radiação e temperatura do painel. O sensor de chuva foi utilizado para a finalidade de se conhecer a frequência de limpeza do painel pela chuva, mas não é um parâmetro decisivo que afeta as previsões de falha.

Após os testes iniciais foram introduzidos os seguintes aprimoramentos:

- Tempo de amostragem para coleta de dados, se dando no período matutino, das 06h00 às 18h00, sendo contabilizados 12 horas de monitoramento, desde o alvorecer do Sol ao seu poente, economizando bateria para o dia seguinte, sendo que o painel não gera energia durante a noite;
- Amostragem de coletas feita pela média e não instantaneamente. Uma vez que o sistema é atualizado a cada 15 segundos, a amostragem na tela das variáveis a serem monitoradas pode ser errônea, por exemplo, dentro dos 15 segundos a serem monitorados se uma folha cai rapidamente sobre o painel solar, bem no instante em que os dados são coletados e enviados ao sinótico, isso acarretará em uma medida lida errada. Devido a tal fato, faz-se a coleta de dados serem pela média e não de forma instantânea. O método utilizado foi ajustar o período de leitura dos sensores, por exemplo em 15 segundos, e dentro deste período divide-se o período de amostragem em cinco. Assim, os valores são contabilizados e guardados a cada 3 segundos e quando precisam ser enviados, no final dos 15 segundos é feita a média desses valores para que não ocorra este tipo de erro durante o monitoramento das placas;
- Criação de histórico de monitoramento: uma vez o sistema funcionando, todos os valores avaliados são registrados no cartão de memória, onde se pode ter fácil acesso de visualização em uma planilha eletrônica, caso se deseje realizar um estudo do comportamento da placa.

A Figura 25 demonstra a lógica implementada no *software* do dispositivo coletor de dados.

Figura 25 – Fluxograma do software implementado no Arduino Mega



Fonte: Os Autores, 2019.

Da Figura 25, observa-se que os dados são enviados para o servidor LoRa do Instituto Mauá de Tecnologia. Nesse processo, é importante ressaltar que depois de lidos os valores de temperatura, radiação solar, presença de chuva, tensão e corrente da placa fotovoltaica, os dados são manipulados e concatenados em uma única mensagem de texto antes de ser enviado ao *gateway* LoRa.

Como na mensagem enviada ao servidor não é possível identificar as casas decimais, faz-se necessário multiplicar o valor lido, de acordo com o grau de precisão desejado, para depois dividi-lo no lado receptor, conforme explicitado no item 3.2.3.

Neste projeto, os valores referentes a temperatura e tensão possuem o grau de precisão de uma casa decimal. Dessa maneira, o valor mensurado pelos sensores foi multiplicado por dez. Já para radiação e corrente, o grau de precisão é de duas casas decimais, fazendo-se necessário multiplicar o valor lido por cem.

O sensor de chuva retorna o valor "1" ao detectar a presença de chuva. Caso contrário, ele retorna "0". Portanto, nesse caso, não é necessário multiplicar o valor adquirido pelo sensor.

Posteriormente, os dados referentes à temperatura, radiação solar, presença de chuva, tensão e corrente são convertidos da base decimal para hexadecimal.

Uma vez manipulados os dados, a mensagem que será enviada ao servidor LoRa é construída obedecendo uma sequência pré-definida para que o receptor consiga interpretá-la de forma correta. Neste projeto a mensagem obedece a seguinte construção:

- *Byte 0 a 2: Temperatura;*
- *Byte 3 a 6: Radiação solar;*
- *Byte 7 a 9: Tensão;*
- *Byte 10 a 12: Corrente;*
- *Byte 13: Presença de chuva;*

3.2.2 SOFTWARE DO APLICATIVO DE ACESSO REMOTO DE CURTA DISTÂNCIA

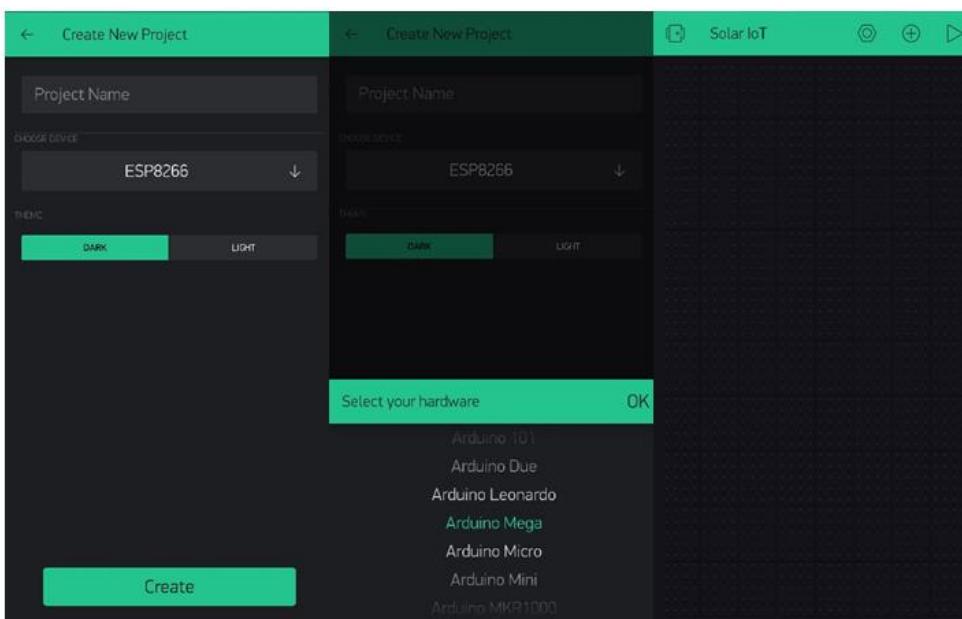
O aplicativo *Blynk* é uma plataforma desenvolvida para criar outros aplicativos de forma simples e com uma interface amigável ao usuário. O *Blynk* é parcialmente *open source*: servidor e bibliotecas possuem código aberto; aplicativo *mobile* é de código fechado (BLYNK, 2019).

O *Blynk* foi escolhido para ser utilizado neste projeto por permitir a comunicação com uma plataforma embarcada possibilitando que as mesmas sejam controladas remotamente, de forma que os dados dos sensores e dos módulos possam ser obtidos e exibidos em aplicativo instalado no dispositivo móvel.

O aplicativo móvel foi desenvolvido parte via *smartphone*, no próprio aplicativo e parte via *software* no Arduino.

Após realizar o *download* do *Blynk* no *smartphone*, disponível para sistemas Android e iOS, ao abrir o aplicativo, cria-se o projeto desejado escolhendo o nome e qual dispositivo deseja ser utilizado. O aplicativo dispõe de diversas plataformas para serem utilizadas, neste caso, a plataforma escolhida foi o Arduino Mega, conforme ilustrado na Figura 26.

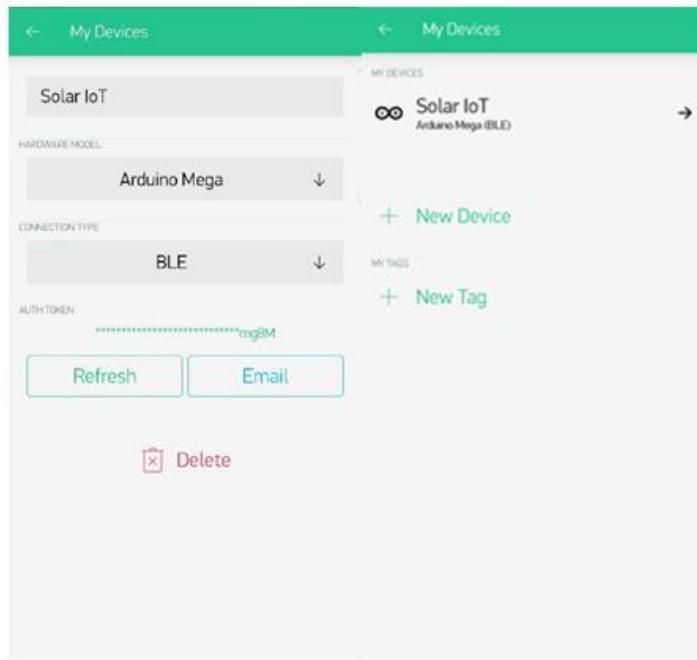
Figura 26 – Telas de inicialização do aplicativo



Fonte: Os Autores, 2019.

Logo após esta etapa, fica disponível o *Auth Token* do aplicativo, conforme ilustrado na Figura 27. O *Auth Token* é a senha de autenticação para que o código, via Arduino, consiga ter acesso ao aplicativo, ou seja, apenas o usuário poderá utilizar o aplicativo, tornando-o mais seguro, uma vez que não fica permitido qualquer dispositivo parear com o aplicativo.

Figura 27 – *Auth Token* e projetos desenvolvidos

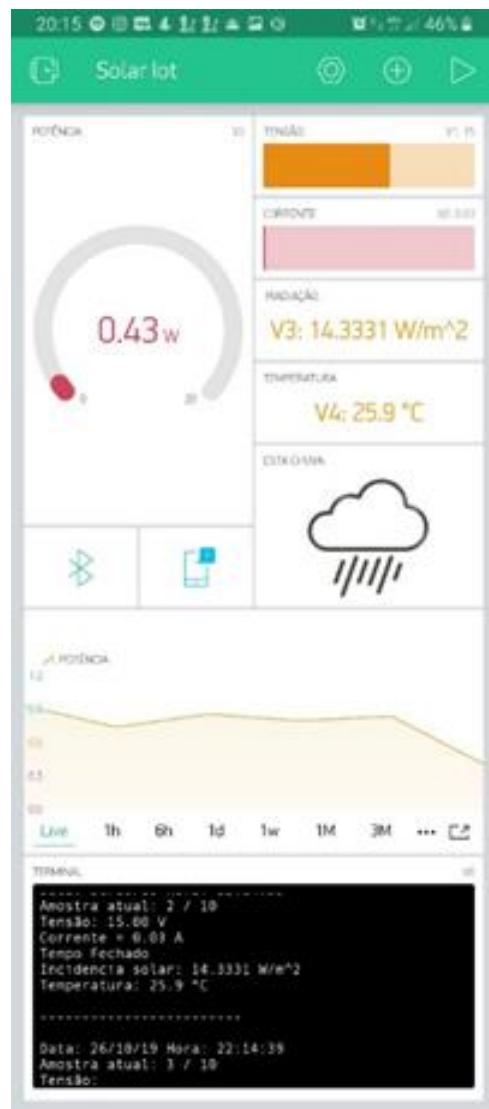


Fonte: Os Autores, 2019.

Uma vez realizada as configurações iniciais para o projeto desenvolvido, inicia-se a programação no dispositivo móvel, escolhendo cada elemento gráfico desejado e alocando-os da forma que se achar mais conveniente.

Sendo assim, no próprio celular configurou-se o layout do aplicativo adicionando os elementos gráficos (*Widgets*): “*value display*, *labeled value display*, *gauge display*, *chart graph*, *notification pop-up*, *Bluetooth BLE (Bluetooth Low-Energy)*, *level H display*, *image gallery display* e *terminal serial BLE display*”, conforme ilustrado na Figura 28.

Figura 28 – Tela principal do aplicativo



Fonte: Os Autores, 2019.

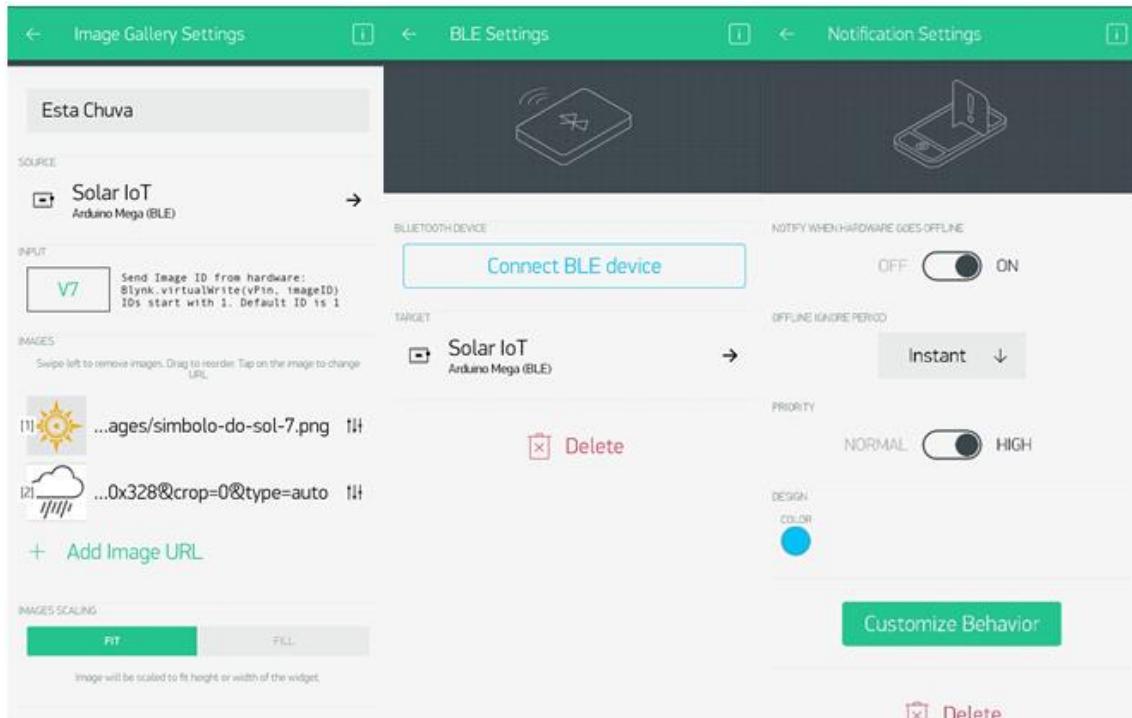
Os dois “*level H displays*” mostrados na parte superior direita foram usados para exibir os valores de tensão, em laranja, e corrente, em vermelho, com esses valores inserem-se o “*Gauge display*”, para saber a potência gerada pela placa instantaneamente e o “*Chart graph*” possibilitando a visualização nos modos ao vivo “*Live*”, em 1 hora, em 6 horas, em 1 dia, em 1 semana, em 1 mês ou em 3 meses de uso do painel, gerando uma visualização do histórico da geração de energia.

Os valores de radiação e temperatura são mostrados nos “*labeled value displays*” em laranja, juntamente com as respectivas unidades de medida.

O “image gallery display” serve para monitorar o clima e saber se a placa está sendo limpa pela chuva. Em dias chuvosos aparece o desenho de uma nuvem carregada chovendo, em dias ensolarados a figura muda e se torna um Sol.

Os *widgets* do “Bluetooth BLE (Bluetooth Low-Energy)” e do “notification pop-up” são inseridos, vide Figura 29, para que o aplicativo tenha disponibilizada a conexão via *Bluetooth* com o *smartphone* e possa ser configurada para quando houver alguma falha prevista enviar uma mensagem notificando o usuário de que algo está errado, com nível de prioridade máxima, pois assim a notificação permanece no topo da lista de notificações do aparelho.

Figura 29 – Configuração dos *Widgets*: *Image Gallery*, *BLE* e *Notification*



Fonte: Os Autores, 2019.

O “terminal serial BLE display” foi utilizado para observar as mensagens exibidas na saída serial do *Bluetooth*, de forma que possa ser comparada com os valores mostrados no aplicativo para se ter certeza de que está tudo funcionando corretamente.

A outra parte, via *software*, programada em C, basicamente foi alinhar os componentes gráficos às variáveis criadas no *software* do Arduino que são interpretadas pelo aplicativo.

Tendo configurado o aplicativo, também foi preciso assegurar seu funcionamento em conjunto com o sistema desenvolvido, assim, com o *hardware* pronto, efetuaram-se alguns testes que puderam tornar o uso do aplicativo possível ao projeto. As condições de simulação de falhas basearam-se no teste de conexão do *Bluetooth*. Foi verificado que o modo de conexão é simples e rápido de ser executado, testou-se se a perda de conexão afetaria a estabilidade do sistema o que não foi observado. Mesmo desconectado o sistema continua operante.

Para que seja possível o pareamento do módulo junto ao *smartphone*, este já conectado ao sistema, basta selecionar no aplicativo o que ele chama de “*Target*”, e a escolha do projeto que se deseja monitorar. Feito isso o aplicativo começa a funcionar automaticamente.

3.2.3 SOFTWARE DO APLICATIVO DE ACESSO REMOTO DE LONGA DISTÂNCIA

As informações adquiridas pelo dispositivo de coleta de dados são codificadas em hexadecimal e concatenadas em uma única mensagem de texto, conforme descrito no item 3.2.1. Essa mensagem é enviada a um *gateway LoRa*, conectado à Internet, que a disponibiliza em um determinado endereço do servidor.

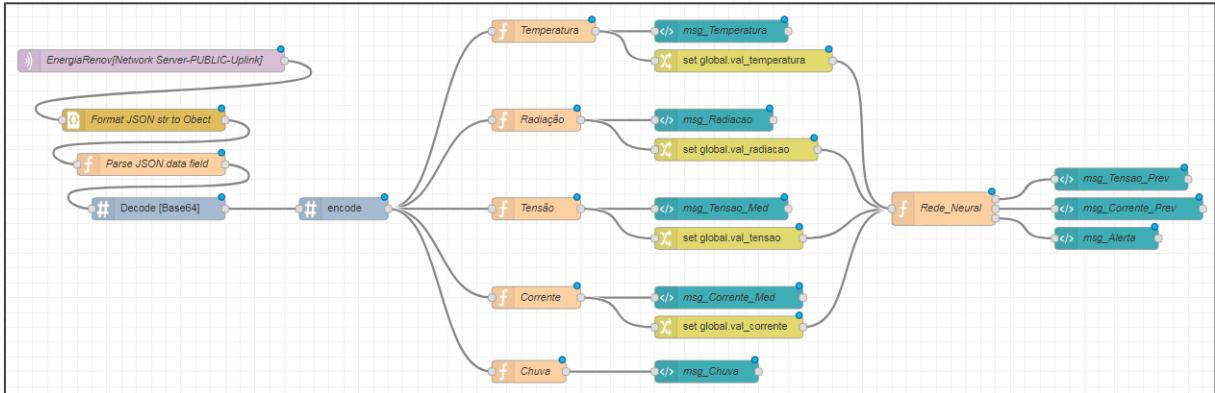
Para se ter acesso à mensagem e posteriormente decodificá-la e interpretá-la, é necessário instalar e configurar o *software Node-RED* em um *notebook*, conforme explicitado no Apêndice B – Instalação e configuração do Node-RED.

Node-RED é uma ferramenta de programação de código aberto criado por uma equipe da IBM e recentemente integrado a JS Foundation. Possui um ambiente de desenvolvimento onde permite que o usuário arraste e solte os componentes, também denominados de nós (SHENG, *et al.*, 2017).

Por ser uma ferramenta de programação baseada em fluxos, o Node-RED descreve a funcionalidade de uma aplicação através de nós, onde cada um tem seu propósito definido. Recebendo dados, processando-os e retornando seu resultado. Em suma, é uma rede de nós compondo uma solução (NODE-RED, 2019).

Portanto, no presente trabalho, para acessar, decodificar e processar as informações contidas na mensagem proveniente do *gateway* LoRa foi desenvolvida a rede de nós ilustrada na Figura 30.

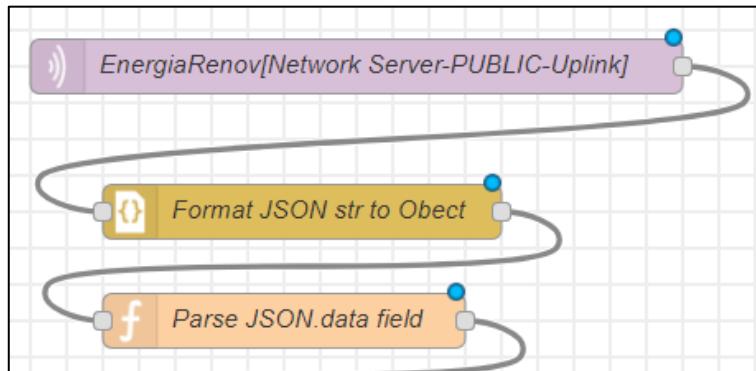
Figura 30 – Rede de Nós do Sistema de Supervisão (Node-RED)



Fonte: Os Autores, 2019.

De uma forma geral, o sistema requisita os dados do servidor, os decodifica, os interpreta e analisa os mesmos. Dividindo a rede para compreendê-la melhor, os nós responsáveis por acessar os dados no servidor e transformá-los em uma mensagem texto estão detalhados na Figura 31.

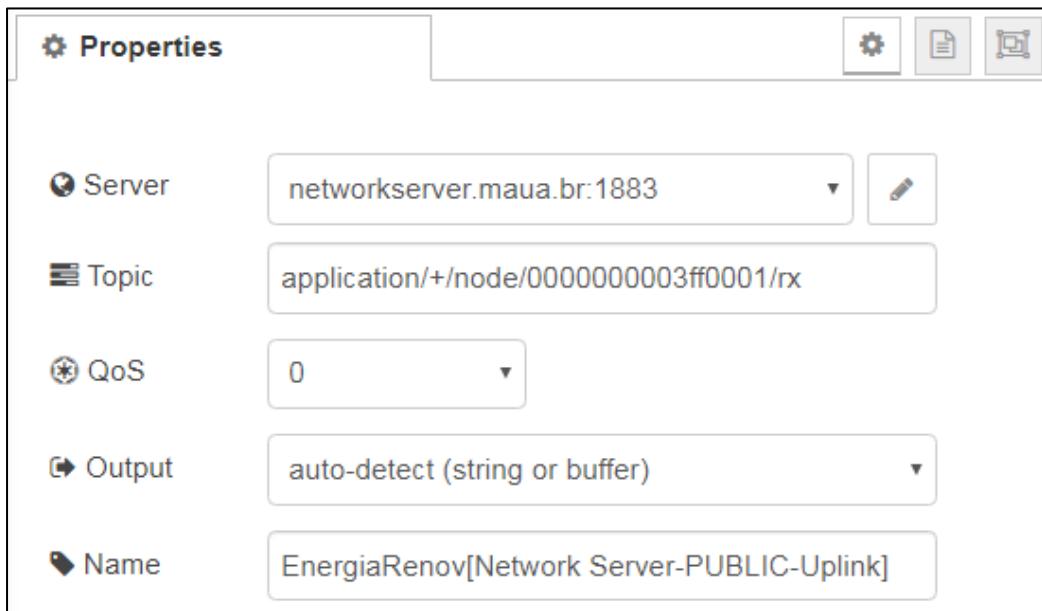
Figura 31 – Bloco de aquisição dos dados do servidor



Fonte: Os Autores, 2019.

O nó “EnergiaRenov” têm por finalidade requisitar os dados em um determinado endereço do servidor que no caso do presente trabalho é 000000003ff0001, conforme ilustra a Figura 32.

Figura 32 – Configuração do nó “EnergiaRenov”



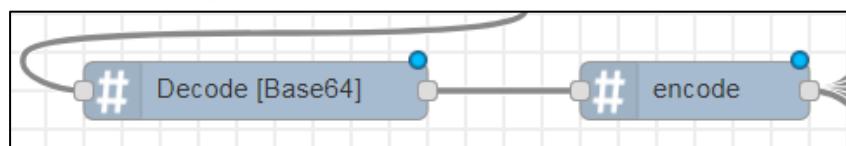
Fonte: Os Autores, 2019.

Os dois nós seguintes, “Format JSON str to Object” e “Parse JSON.data field” são responsáveis por, respectivamente, transformar a mensagem Json tipo texto em um objeto Json e separar somente o valor referente ao objeto campo *.data* (INSTITUTO MAUÁ DE TECNOLOGIA, 2018). Para o nó “Format JSON str to Object” não é necessária nenhuma configuração adicional, ao passo que para o segundo é necessário adicionar as seguintes linhas de código.

```
msg.payload = msg.payload.data;
return msg;
```

Os dados requisitados encontram-se codificados na Base64, portanto, eles necessitam ser decodificados e posteriormente codificados em hexadecimal, para que finalmente a mensagem de texto fique idêntica à enviada pelo dispositivo coletor. Os nós responsáveis por essas tarefas estão identificados de uma forma mais detalhada na Figura 33.

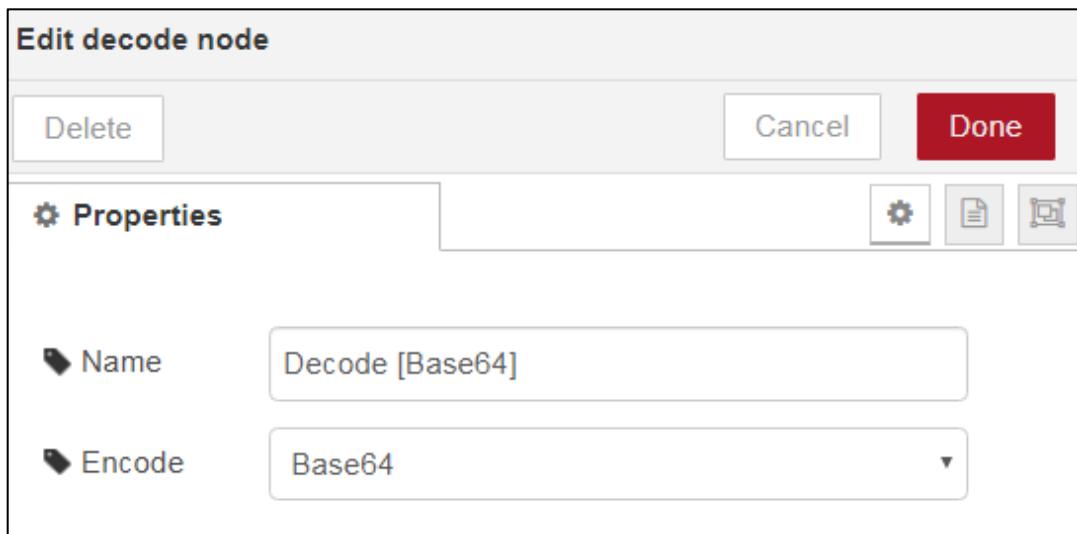
Figura 33 – Bloco de decodificação dos dados



Fonte: Os Autores, 2019.

O nó “Decode” tem a finalidade de decodificar a mensagem proveniente do servidor que está na Base64. A configuração desse nó deve possuir no campo “Encode” o parâmetro “Base64” (INSTITUTO MAUÁ DE TECNOLOGIA, 2018), conforme ilustrado na Figura 34.

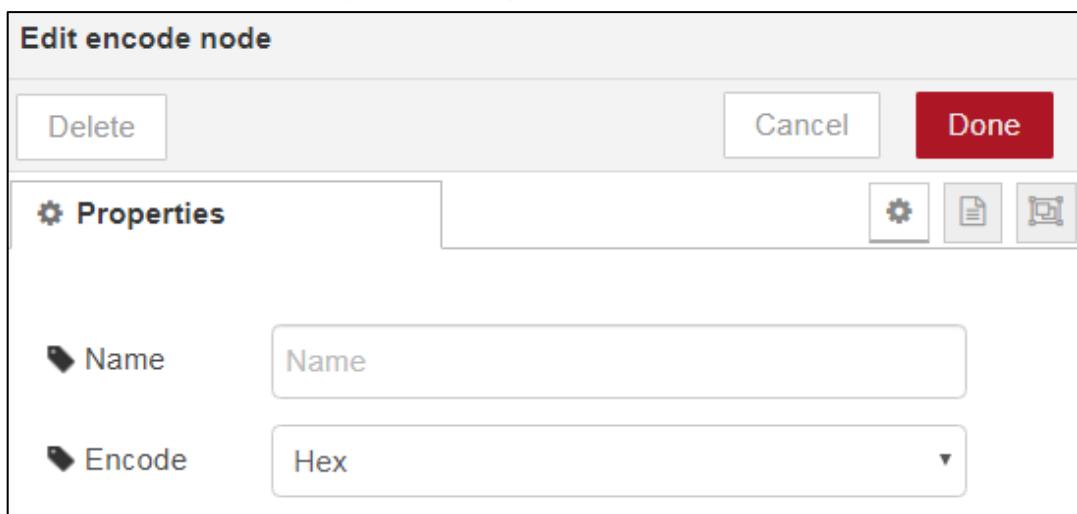
Figura 34 – Configuração do nó “Decode”



Fonte: Os Autores, 2019.

Já o nó “Encode” tem a função de passar a mensagem para a base hexadecimal, sendo assim, é necessário que em sua configuração, o campo “Encode” esteja como “Hex”, conforme ilustrado na Figura 35.

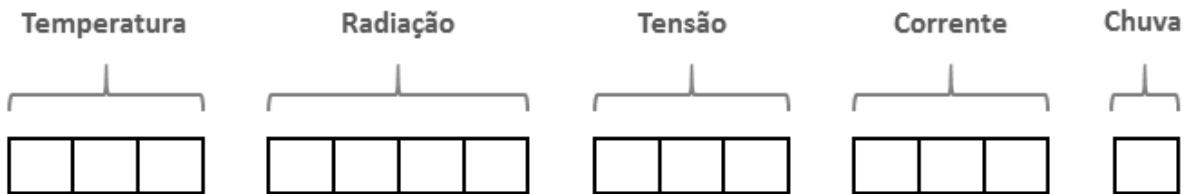
Figura 35 – Configuração do nó “Encode”



Fonte: Os Autores, 2019.

Após o processo de requisição e decodificação, a mensagem possui 14 *bytes* hexadecimais contendo as informações de temperatura, radiação, presença de chuva, tensão e corrente da placa fotovoltaica, como ilustra a Figura 36.

Figura 36 – Estrutura da mensagem hexadecimal proveniente do servidor



Fonte: Os Autores, 2019.

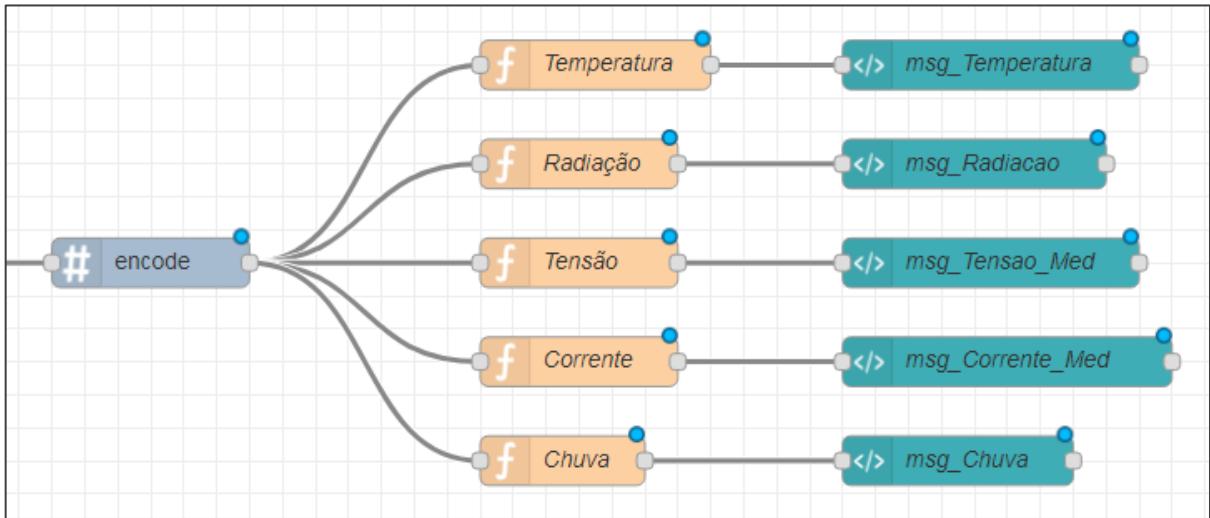
Isto posto, a mensagem necessita ser interpretada, ou seja, que os *bytes* referentes a cada grandeza sejam separados, convertidos para a base decimal e divididos por dez ou cem, de acordo com o grau de precisão definidos no código do *firmware* do dispositivo coletor, descrito no item 3.2.1.

Os valores referentes a temperatura e tensão possuem o grau de precisão de uma casa decimal. Sendo assim, no código do *firmware* do dispositivo coletor, o valor lido pelos sensores foi multiplicado por dez e transmitido dessa maneira. Portanto, ao interpretar a mensagem requisitada, é necessário dividir o valor já convertido em decimal por dez. Essa técnica é utilizada porque na mensagem codificada e enviada não é possível identificar as casas decimais.

O mesmo ocorre para a radiação e corrente, porém o seu grau de precisão é de duas casas decimais, o que implica em dividir o valor já convertido em decimal por cem. No caso do *byte* referente à presença de chuva não é necessário dividir o valor, uma vez que o valor “1”, representa que está chovendo e “0”, que não está chovendo.

Os nós responsáveis por interpretar a mensagem hexadecimal, traduzindo-a em valores de temperatura, radiação, tensão, corrente, presença de chuva e exibi-los no *dashboard* estão identificados na Figura 37.

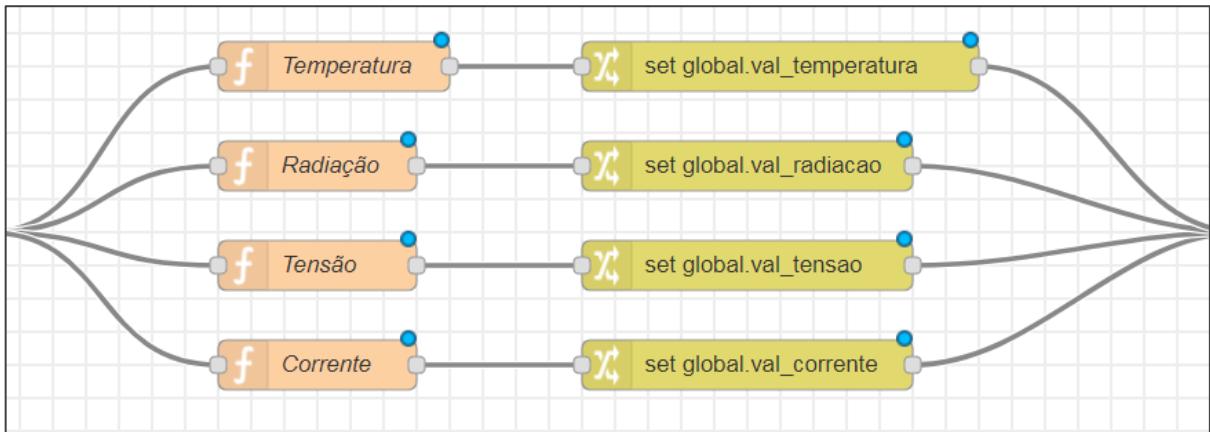
Figura 37 – Bloco de interpretação da mensagem e exibição dos dados



Fonte: Os Autores, 2019.

A fim de utilizar os valores lidos pelos nós “Temperatura”, “Radiação”, “Tensão” e “Corrente” em outra função, é necessário transferí-los para uma variável global, conforme mostra a Figura 38.

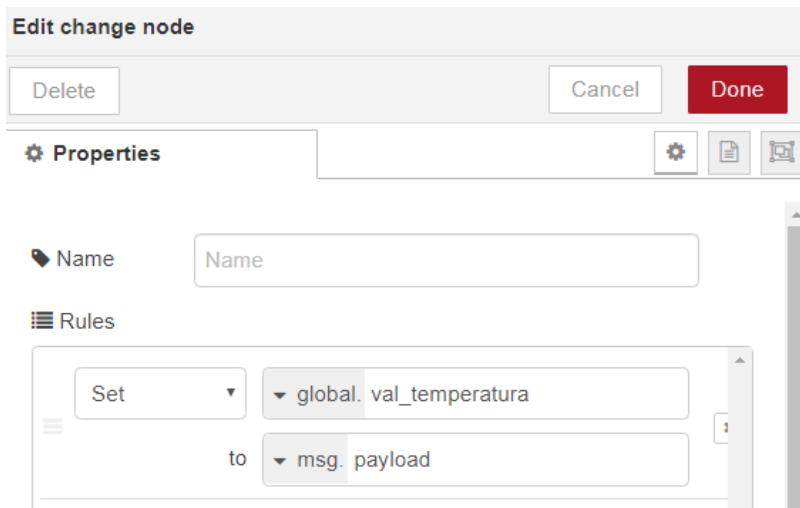
Figura 38 – Nós de transferência para variáveis globais



Fonte: Os Autores, 2019.

A configuração do nó “set global.val_temperatura” é demonstrada na Figura 39. De forma análoga, os demais nós são configurados.

Figura 39 – Configuração do nó “set global.val_temperatura”

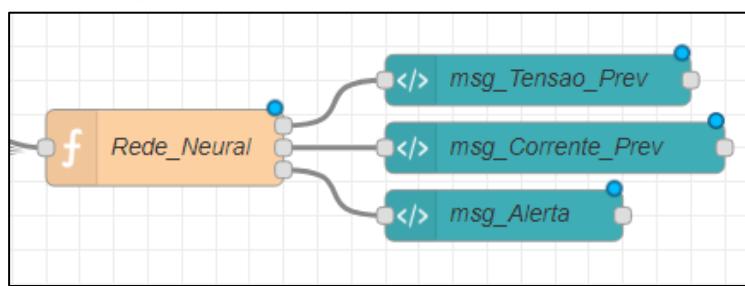


Fonte: Os Autores, 2019.

Posto que os dados foram requisitados do servidor, decodificados, interpretados e as grandezas de temperatura, radiação, tensão e corrente exibidas em um *dashboard*, é crucial que estes sejam analisados de modo a verificar se os valores medidos estão dentro do esperado.

Para tal, implementou-se uma Rede Neural Artificial com duas entradas e duas saídas em um nó denominado “Rede_Neural”, conforme ilustrado na Figura 40. A criação da RNA é explanada detalhadamente no item 3.3

Figura 40 – Bloco de análise e exibição dos dados



Fonte: Os Autores, 2019.

Como visto na Figura 40 o nó “Rede_Neural” possui três saídas que posteriormente são exibidas no *dashboard*: duas correspondem às saídas da RNA, valores de tensão e correntes previstos; e a outra é o endereço de uma imagem que representa um sinal de alerta caso os valores de tensão ou corrente medidos extrapolarem em 10 % os valores previstos.

O código fonte referente aos blocos de interpretação da mensagem, análise e exibição dos dados estão demonstrados no Apêndice D – Código fonte dos blocos no Node-RED.

3.3 REDE NEURAL ARTIFICIAL

3.3.1 INTRODUÇÃO E CONCEITOS BÁSICOS

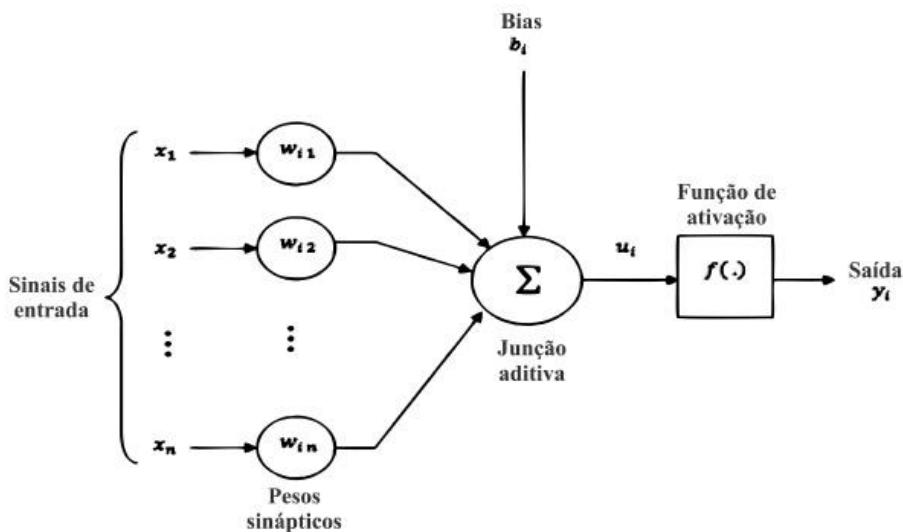
As redes neurais artificiais, cuja origem se baseia no estudo do cérebro humano e na possibilidade de replicar e adaptar o seu sistema de processamento de informações, são métodos computacionais matemáticos que tem como vantagens a não-linearidade, adaptabilidade e a efetiva capacidade de generalização. De acordo com Haykin (2003), “*O cérebro é um computador (sistema de processamento de informações) altamente complexo, não-linear e paralelo.*”, sendo as unidades de processamento desse computador denominadas neurônios, que são responsáveis por receber informações de entrada (*inputs*), processá-las, e logo em seguida gerar informações de saída (*outputs*). Os *outputs* de um neurônio viram respectivamente os *inputs* de outro neurônio, e essa sequência juntamente com o aprendizado desses neurônios é resumidamente o que constitui a complexa unidade de processamento do cérebro humano, e consequentemente o conceito padrão de uma Rede Neural Artificial (MAGO e BHATIA, 2012).

Para entender melhor o funcionamento de uma rede neural, é necessário analisar primeiro o modelo de um neurônio, exposto na Figura 41, e como definido por Haykin (2003), são identificados três elementos básicos e o *bias* em um modelo neuronal:

- *Elos de conexão*, cada um caracterizado por um peso próprio. Os pesos têm como função multiplicar os sinais de entrada, e no processo aprendizagem de uma rede neural artificial eles sofrem alterações até atingir o resultado desejado da rede.
- Um *somador*, que soma todos os sinais de entrada após serem multiplicados pelos elos de conexão.
- Uma *função de ativação*, a qual limita a amplitude do sinal de saída de um neurônio a um valor finito.

- O *bias*, que aumenta ou diminui fixamente o valor de saída do somador dependendo do seu sinal. Se o *bias* de um neurônio valer -1 (menos um), e todos os pesos desse mesmo neurônio forem nulos, temos o valor do sinal de saída do somador o próprio *bias*, ou seja, -1 (menos um). É importante ressaltar que o *bias* também sofre alterações no processo de aprendizagem.

Figura 41 – Modelo de um neurônio



Fonte: Redes Neurais: Princípios e Prática, 2003.

Outro ponto crucial de uma rede neural artificial é a sua capacidade de aprender. Do mesmo modo que o cérebro humano aprende, incorporando diversos novos conceitos e aprimorando tarefas e conhecimentos, uma rede neural também realiza esse processo, o qual é conhecido como *processo de aprendizagem* e acontece por meio de um *algoritmo de aprendizagem* (HAYKIN, 2003). Existem diversos tipos de aprendizagem (por correção de erro, competitiva e baseada em memória, por exemplo), entretanto como os mesmos não são o enfoque do trabalho, é importante saber que o *processo de aprendizagem* se baseia na modificação contínua dos valores de parâmetros de uma rede neural até ela atingir os resultados desejados. Vale apresentar uma definição adaptada e citada por Haykin (2003), a qual define a aprendizagem de uma rede neural como: “*Aprendizagem é um processo pelo qual os parâmetros livres de uma rede neural são adaptados através de um processo de estimulação pelo ambiente no qual a rede está inserida. O tipo de aprendizagem é determinado pela maneira pela qual a modificação dos parâmetros ocorre.*”.

As redes neurais tem a capacidade de resolver problemas complexos, e uma vez aprendido os conceitos básicos e o funcionamento delas, a sua aplicação se resume à definição das melhores características para cada caso, como o número de neurônios e a *função de ativação*, à utilização do melhor método de aprendizagem e à capacidade de saber analisar a qualidade e veracidade dos dados de entrada e saída dessa rede para a aplicação desejada. O item 3.3.2 demonstra como foi realizada a aplicação de uma rede neural para a previsão de parâmetros de um painel solar, assim como descrever os parâmetros escolhidos e pontos de atenção.

3.3.2 APLICAÇÃO DA REDE NEURAL ARTIFICIAL

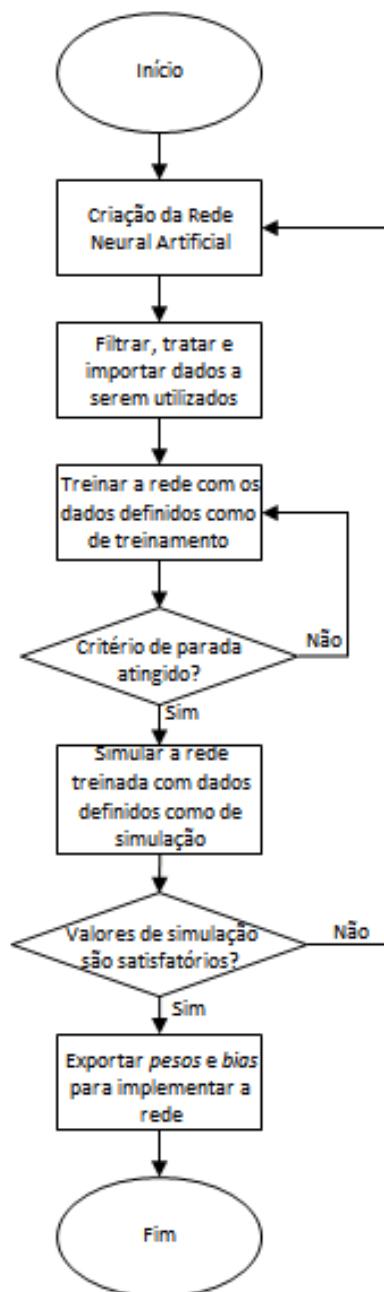
Como observado na literatura, pode-se relacionar a temperatura do painel ($^{\circ}\text{C}$) e a radiação solar incidida sobre ele (W/m^2), com a tensão (V) e a corrente (A) gerada (GAROUDJA, HARROU, *et al.*, 2017). Entretanto pelo fato dessa relação também depender de parâmetros eletrônicos construtivos do painel solar, os quais não são comumente informados nas informações técnicas padrão desse equipamento e necessitam testes para serem descobertos, optou-se por utilizar redes neurais artificiais para estimar as grandezas desejadas a partir das medidas. Essa abordagem se torna interessante pelo fato do potencial de generalização e resolução de problemas não-lineares das redes neurais artificiais se encaixarem corretamente no caso em estudo, apresentando bons resultados em trabalhos previstos (ALMONACID, *et al.*, 2009). A aplicação de uma rede neural artificial pode ser resumida nos seguintes passos:

- I. Criação da rede neural artificial, definindo a configuração da rede e o método de aprendizagem baseado no problema em questão. Optar por um *software* onde haja familiaridade é recomendável para garantir a correta implementação dessa rede neural na interface computacional.
- II. Filtrar e importar os dados que serão utilizados para o treinamento e a simulação da rede neural artificial. Esses dados devem ser corretamente filtrados, tratados e logo depois divididos aleatoriamente entre dados de treinamento e dados de simulação.
- III. Treinar a rede com os dados de treinamento a partir do método de aprendizagem definido, estipulando critérios de parada considerados adequados para a aplicação.

- IV. Simular a rede criada e treinada com os dados de simulação a fim de avaliar a qualidade da rede neural artificial, ou seja, ver se os valores previstos por ela estão coerentes com os reais medidos.
- V. Exportar os *pesos* e *bias* da rede, os quais são necessários para implementar a rede neural artificial em outro *software* ou outra aplicação.

A Figura 42 mostra o fluxograma da aplicação de uma rede neural artificial.

Figura 42 – Fluxograma da aplicação de uma rede neural artificial



Fonte: Os Autores, 2019.

Neste trabalho foram criadas duas redes neurais artificiais, sendo um referente à tensão (V) e outra à corrente (A) gerada.

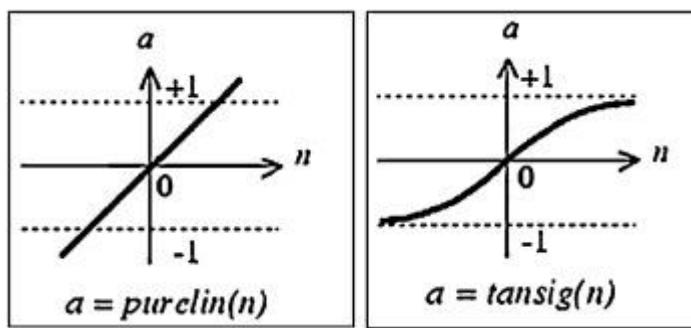
O passo-a-passo do processo de criação e aplicação dessa rede neural, assim como definições do *software* e pontos de atenção no seu uso, estão descritos no Apêndice E – Criação e simulação de uma Rede Neural Artificial no *Matlab*.

Começando pela tensão (V), optou-se pela criação e utilização de uma rede do tipo *Perceptron* multicamadas, com uma camada escondida composta por 7 (sete) neurônios e uma camada de saída composta por um neurônio. A escolha dessa configuração se baseou no trabalho de (ALMONACID, *et al.*, 2009), que apresentou bons resultados.

Para o treinamento da rede neural, utilizou-se uma aprendizagem em batelada com um algoritmo de retro-propagação do tipo Levenberg-Marquardt, o qual minimiza a magnitude do erro quadrático médio (*MSE*) entre os valores previstos e os medidos reais (ALMONACID, *et al.*, 2009).

Em relação às funções de ativação, para os 7 (sete) neurônios da camada escondida optou-se pela função de ativação *tansigmoidal*, enquanto para o neurônio da camada de saída a função de ativação optada foi a *purelin*. A Figura 43 apresenta em forma gráfica ambas as funções citadas acima.

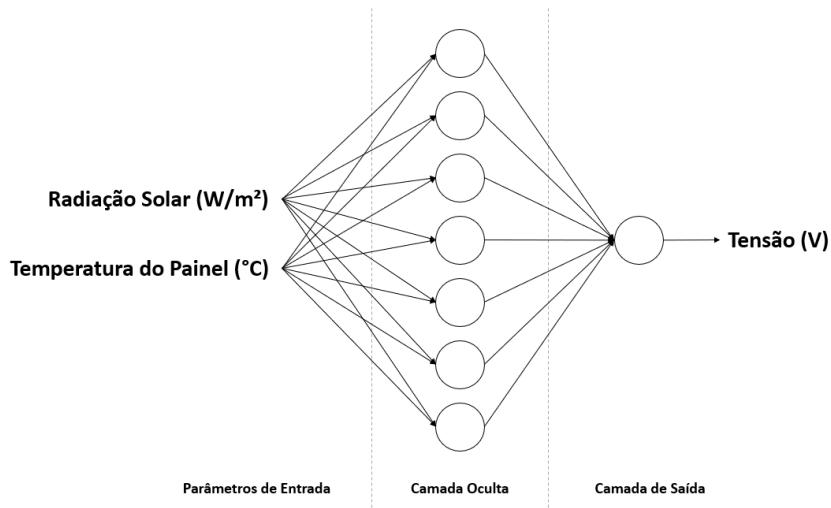
Figura 43 – Função de ativação *purelin* e *tansigmoidal* respectivamente



Fonte: (HMAMOUCHI, LARIF, *et al.*, 2015)

Uma vez criada a rede neural artificial de previsão de tensão (V), pode-se observar na Figura 44 a sua topologia.

Figura 44 – Topologia da rede neural artificial criada



Fonte: Os Autores, 2019.

O procedimento de filtragem de dados pode ser automatizado, contendo uma lógica de programação por trás com limites e condições, ou meramente manual. No caso desse trabalho os dados de temperatura ($^{\circ}\text{C}$), radiação solar (W/m^2) e a suas respectivas tensões (V) e correntes (A) geradas foram filtrados manualmente, deletando-se os valores que se apresentavam totalmente incorretos em relação aos demais e às condições climáticas. Como exemplo temos o caso de obtermos um valor próximo a zero de radiação solar (W/m^2) em pleno meio-dia, um instante do dia que com certeza esse valor não seria quase nulo. Outro exemplo de medição incorreta dos parâmetros seria uma linha de dados cuja temperatura ($^{\circ}\text{C}$) estivesse negativa, sendo que no local geográfico onde a medição foi efetuada sabe-se que temperaturas negativas são extremamente improváveis.

Antes de importar os dados referentes aos parâmetros de entrada da rede, os mesmos foram normalizados. Essa manipulação dos valores é necessária primeiramente para prevenir que as *funções de ativação* atuem predominantemente em região de saturação, e em segundo lugar para evitar que valores com grandezas diferentes e número de ordem muito discrepante não influenciem de maneira inadequada os *pesos* e o *processo de aprendizagem* (GAMBOGI, 2013).

Os valores de temperatura ($^{\circ}\text{C}$) e radiação solar (W/m^2) normalizados foram obtidos através da equação (4).

$$x = \left[(y - \text{MaxReal}) \times \frac{(\text{MaxN} - \text{MinN})}{(\text{MaxReal} - \text{MinReal})} \right] + \text{MaxN} \quad (4)$$

Em que:

x é o valor normalizado;

y é o valor a ser normalizado;

MaxReal é o valor máximo considerado para os valores reais;

MaxN é o valor máximo considerado para os valores normalizados;

MinReal é o valor mínimo considerado para os valores reais;

MinN é o valor mínimo considerado para os valores normalizado.

A Tabela 1 contém os valores definidos como máximos e mínimos reais e normalizados, assim como um exemplo de cada valor.

Tabela 1 — Valores utilizados para normalização dos parâmetros de entrada

Grandeza	Máx. definido	Min. definido	Ex. Valor Real	Ex. Valor Normalizado
Temperatura (°C)	60	0	38,40	0,2800
Radiação solar (W/m ²)	1200	0	88,77	-0,9202

Fonte: Os Autores, 2019.

Após todo o processo de filtragem dos dados e a normalização dos valores de entrada, foram importados mais de 7000 (sete mil) dados para o Matlab, sendo que metade deles foram rotulados aleatoriamente como dados de treinamento, e a outra metade como dados de teste/simulação.

Utilizando-se os dados destinados à aprendizagem, treinou-se a rede neural artificial na própria ferramenta *nntool* do Matlab (MATHWORKS, 2019), definindo-se os seguintes critérios de parada:

- Máximo de 1000 (mil) iterações, sem um tempo máximo definido de treinamento;
- Critério de performance definido: Erro quadrático médio (MSE) de 0,0001 ou 0,01 %;
- Máximo de 6 iterações sem melhora do critério de performance.

Uma vez completo o *processo de aprendizagem* com algum dos critérios de parada atingidos passa-se para a fase de simulação da rede neural, ou seja, testa-se a rede com os dados de simulação. Considerando como entradas da rede as temperaturas ($^{\circ}\text{C}$) e as radiações solares (W/m^2), obtemos como saída a de tensão (V) previstos pela rede neural artificial. Para checar se esses dados obtidos são satisfatórios ou não, foram plotadas e comparadas as curvas de ambas as tensões (V), previstas e medidas, referentes aos parâmetros de *input*. Os resultados obtidos e as análises da rede neural artificial criada podem ser consultados na parte de *Resultados e Discussão* deste trabalho, capítulo 4.

O último passo da aplicação da rede neural é exportar os *pesos* e os *bias* dos seus neurônios para implantar em outros *softwares*, interfaces ou processos automatizados. Esses valores são resultados do *processo de aprendizagem* de uma rede neural artificial e são o que as diferenciam entre si. Duas redes neurais podem conter o mesmo número de camadas, com o mesmo número de neurônios e as mesmas *funções de ativação*, entretanto os seus *pesos* e *bias* variam de acordo com o caso a ser estudado.

O procedimento para a criação da rede neural que irá prever a corrente (A) gerada é o mesmo que a da tensão (V), entretanto na hora de realizar o treinamento e simulação, utilizamos como saída real as correntes (A) referentes às temperaturas ($^{\circ}\text{C}$) e radiações solares (W/m^2). É importante ressaltar que apesar do processo ser praticamente igual, os *pesos* e *bias* modificados e exportados serão totalmente diferentes que os relativos à rede neural de tensão (V), pois apesar de serem grandezas relacionadas, cada processo de aprendizagem é único, gerando sempre diferentes valores.

4 RESULTADOS E DISCUSSÃO

Com o *hardware* pronto, foi iniciado o processo de testes e coletas de dados. A placa foi instalada em três lugares distintos, primeiro na residência de um dos membros do grupo, depois no quarto andar do bloco W, localizado no Instituto Mauá de Tecnologia, por fim, posicionado no terraço do mesmo bloco.

Inicialmente, a placa foi colocada na casa de um dos membros, conforme ilustrado na Figura 45, para supervisionar o desempenho do sistema. Na ocasião, o conjunto foi testado utilizando alimentação por baterias e sem o sistema de comunicação LoRa. Porém as baterias apresentaram problemas, não atenderam a necessidade energética por um tempo satisfatório e o BMS não estava funcionando como esperado, pois não estava carregando as baterias.

Figura 45 – Teste realizado na casa de um membro do grupo



Fonte: Os Autores, 2019.

Figura 46 – Dados coletados pelo SD Card com a placa instalada na casa de um membro do grupo

Data: 07/09/19 - Hora: 14:27:42	18.11 V	1.28 A	Nao esta chovendo	33.0205 W/m²	37.3 °C
Data: 07/09/19 - Hora: 14:28:12	18.00 V	1.26 A	Nao esta chovendo	31.9312 W/m²	37.3 °C
Data: 07/09/19 - Hora: 14:28:42	18.02 V	1.30 A	Nao esta chovendo	32.4085 W/m²	37.4 °C
Data: 07/09/19 - Hora: 14:29:12	18.00 V	1.29 A	Nao esta chovendo	32.0747 W/m²	37.4 °C
Data: 07/09/19 - Hora: 14:29:42	17.97 V	1.27 A	Nao esta chovendo	31.7731 W/m²	37.4 °C
Data: 07/09/19 - Hora: 14:30:12	18.08 V	1.28 A	Nao esta chovendo	33.1552 W/m²	37.4 °C
Data: 07/09/19 - Hora: 14:30:42	18.02 V	1.27 A	Nao esta chovendo	33.0293 W/m²	37.4 °C
Data: 07/09/19 - Hora: 14:31:12	17.97 V	1.31 A	Nao esta chovendo	32.8624 W/m²	37.5 °C
Data: 07/09/19 - Hora: 14:31:42	18.08 V	1.28 A	Nao esta chovendo	34.3543 W/m²	37.5 °C
Data: 07/09/19 - Hora: 14:32:12	18.00 V	1.31 A	Nao esta chovendo	33.4129 W/m²	37.6 °C
Data: 07/09/19 - Hora: 14:32:42	18.00 V	1.31 A	Nao esta chovendo	33.5915 W/m²	37.6 °C
Data: 07/09/19 - Hora: 14:33:12	18.00 V	1.27 A	Nao esta chovendo	34.0776 W/m²	37.6 °C
Data: 07/09/19 - Hora: 14:33:42	18.02 V	1.29 A	Nao esta chovendo	34.6120 W/m²	37.5 °C
Data: 07/09/19 - Hora: 14:34:12	18.05 V	1.28 A	Nao esta chovendo	35.2782 W/m²	37.6 °C
Data: 07/09/19 - Hora: 14:34:42	18.05 V	1.32 A	Nao esta chovendo	35.6691 W/m²	37.7 °C
Data: 07/09/19 - Hora: 14:35:12	18.13 V	1.30 A	Nao esta chovendo	36.8272 W/m²	37.6 °C
Data: 07/09/19 - Hora: 14:35:42	18.11 V	1.33 A	Nao esta chovendo	37.4715 W/m²	37.8 °C
Data: 07/09/19 - Hora: 14:36:12	18.11 V	1.30 A	Nao esta chovendo	37.7408 W/m²	37.8 °C
Data: 07/09/19 - Hora: 14:36:42	18.00 V	1.31 A	Nao esta chovendo	37.8492 W/m²	37.7 °C
Data: 07/09/19 - Hora: 14:37:12	18.11 V	1.32 A	Nao esta chovendo	38.8375 W/m²	37.8 °C
Data: 07/09/19 - Hora: 14:37:42	18.11 V	1.29 A	Nao esta chovendo	39.7042 W/m²	37.6 °C
Data: 07/09/19 - Hora: 14:38:12	17.89 V	1.31 A	Nao esta chovendo	38.8799 W/m²	37.7 °C
Data: 07/09/19 - Hora: 14:38:42	18.05 V	1.30 A	Nao esta chovendo	40.0556 W/m²	37.6 °C
Data: 07/09/19 - Hora: 14:39:12	18.00 V	1.31 A	Nao esta chovendo	40.6852 W/m²	37.6 °C
Data: 07/09/19 - Hora: 14:39:42	18.00 V	1.26 A	Nao esta chovendo	40.2255 W/m²	37.5 °C
Data: 07/09/19 - Hora: 14:40:12	18.11 V	1.29 A	Nao esta chovendo	41.7218 W/m²	37.4 °C
Data: 07/09/19 - Hora: 14:40:42	18.13 V	1.29 A	Nao esta chovendo	42.3895 W/m²	37.4 °C
Data: 07/09/19 - Hora: 14:41:12	18.08 V	1.31 A	Nao esta chovendo	42.4348 W/m²	37.5 °C

Fonte: Os Autores, 2019.

No primeiro teste, os dados eram armazenados no cartão SD a cada 30 segundos e foram colhidas 525 amostras. O *Bluetooth* também funcionou como o previsto, vide Figura 47, validando os dois sistemas de comunicação.

Figura 47 – Dados coletados pelo *Bluetooth*

```

Terminal
14:33:44.053
14:33:44.071
14:34:13.428
14:34:13.428 —Data: 07/09/19 Hora: 14:34:12
14:34:13.448 Tensão: 18.05 V
14:34:13.482 Corrente = 1.28 A
14:34:13.968 Nao esta chovendo
14:34:14.009 Incidencia solar: 35.2782 W/m^2
14:34:14.044 Temperatura: 37.6 oC
14:34:14.061
14:34:14.061
14:34:43.424
14:34:43.439 —Data: 07/09/19 Hora: 14:34:42
14:34:43.439 Tensão: 18.05 V
14:34:43.470 Corrente = 1.32 A
14:34:43.969 Nao esta chovendo
14:34:44.013 Incidencia solar: 35.6691 W/m^2
14:34:44.071 Temperatura: 37.7 oC
14:34:44.071
14:34:44.085
14:35:13.424
14:35:13.554 —Data: 07/09/19 Hora: 14:35:12
14:35:13.580 Tensão: 18.13 V
14:35:13.580 Corrente = 1.30 A
14:35:13.982 Nao esta chovendo
14:35:14.002 Incidencia solar: 36.8272 W/m^2
14:35:14.045 Temperatura: 37.6 oC
14:35:14.058
14:35:14.071
14:35:43.424
14:35:43.439 —Data: 07/09/19 Hora: 14:35:42
14:35:43.450 Tensão: 18.11 V
14:35:43.482 Corrente = 1.33 A
14:35:43.993 Nao esta chovendo
14:35:44.002 Incidencia solar: 37.4715 W/m^2
14:35:44.039 Temperatura: 37.7 oC
14:35:44.077
14:35:44.087
M1 M2 M3 M4 M5 M6 M7 >

```

Fonte: Os Autores, 2019.

A troca de baterias foi realizada, e então, após uma semana de ajustes no sistema BMS, foram iniciados novos testes, novamente na residência de um dos membros. Os dados coletados nessa ocasião estão ilustrados na Figura 48.

Figura 48 – Dados coletados pelo SD Card com a placa instalada na casa de um membro do grupo na semana seguinte

Data: 07/09/19 - Hora: 14:27:42	18.11	V	1.28	A	Nao esta chovendo	33.0205 W/m^2	37.3 oC
Data: 07/09/19 - Hora: 14:28:12	18.00	V	1.26	A	Nao esta chovendo	31.9312 W/m^2	37.3 oC
Data: 07/09/19 - Hora: 14:28:42	18.02	V	1.30	A	Nao esta chovendo	32.4085 W/m^2	37.4 oC
Data: 07/09/19 - Hora: 14:29:12	18.00	V	1.29	A	Nao esta chovendo	32.0747 W/m^2	37.4 oC
Data: 07/09/19 - Hora: 14:29:42	17.97	V	1.27	A	Nao esta chovendo	31.7731 W/m^2	37.4 oC
Data: 07/09/19 - Hora: 14:30:12	18.08	V	1.28	A	Nao esta chovendo	33.1552 W/m^2	37.4 oC
Data: 07/09/19 - Hora: 14:30:42	18.02	V	1.27	A	Nao esta chovendo	33.0293 W/m^2	37.4 oC
Data: 07/09/19 - Hora: 14:31:12	17.97	V	1.31	A	Nao esta chovendo	32.8624 W/m^2	37.5 oC
Data: 07/09/19 - Hora: 14:31:42	18.08	V	1.28	A	Nao esta chovendo	34.3543 W/m^2	37.5 oC
Data: 07/09/19 - Hora: 14:32:12	18.00	V	1.31	A	Nao esta chovendo	33.4129 W/m^2	37.6 oC
Data: 07/09/19 - Hora: 14:32:42	18.00	V	1.31	A	Nao esta chovendo	33.5915 W/m^2	37.6 oC
Data: 07/09/19 - Hora: 14:33:12	18.00	V	1.27	A	Nao esta chovendo	34.0776 W/m^2	37.6 oC
Data: 07/09/19 - Hora: 14:33:42	18.02	V	1.29	A	Nao esta chovendo	34.6120 W/m^2	37.5 oC
Data: 07/09/19 - Hora: 14:34:12	18.05	V	1.28	A	Nao esta chovendo	35.2782 W/m^2	37.6 oC
Data: 07/09/19 - Hora: 14:34:42	18.05	V	1.32	A	Nao esta chovendo	35.6691 W/m^2	37.7 oC
Data: 07/09/19 - Hora: 14:35:12	18.13	V	1.30	A	Nao esta chovendo	36.8272 W/m^2	37.6 oC
Data: 07/09/19 - Hora: 14:35:42	18.11	V	1.33	A	Nao esta chovendo	37.4715 W/m^2	37.8 oC
Data: 07/09/19 - Hora: 14:36:12	18.11	V	1.30	A	Nao esta chovendo	37.7408 W/m^2	37.8 oC
Data: 07/09/19 - Hora: 14:36:42	18.08	V	1.31	A	Nao esta chovendo	37.8492 W/m^2	37.7 oC
Data: 07/09/19 - Hora: 14:37:12	18.11	V	1.32	A	Nao esta chovendo	38.8375 W/m^2	37.8 oC
Data: 07/09/19 - Hora: 14:37:42	18.11	V	1.29	A	Nao esta chovendo	39.7042 W/m^2	37.6 oC
Data: 07/09/19 - Hora: 14:38:12	17.89	V	1.31	A	Nao esta chovendo	38.8799 W/m^2	37.7 oC
Data: 07/09/19 - Hora: 14:38:42	18.05	V	1.30	A	Nao esta chovendo	40.0556 W/m^2	37.6 oC
Data: 07/09/19 - Hora: 14:39:12	18.08	V	1.31	A	Nao esta chovendo	40.6852 W/m^2	37.6 oC
Data: 07/09/19 - Hora: 14:39:42	18.00	V	1.26	A	Nao esta chovendo	40.2255 W/m^2	37.5 oC
Data: 07/09/19 - Hora: 14:40:12	18.11	V	1.29	A	Nao esta chovendo	41.7218 W/m^2	37.4 oC
Data: 07/09/19 - Hora: 14:40:42	18.13	V	1.29	A	Nao esta chovendo	42.3895 W/m^2	37.4 oC
Data: 07/09/19 - Hora: 14:41:12	18.08	V	1.31	A	Nao esta chovendo	42.4348 W/m^2	37.5 oC

Fonte: Os Autores, 2019.

Foram colhidas 2214 amostras, registradas a cada 1 minuto. Com essa nova coleta foi verificada a autonomia da bateria que é de aproximadamente 20 horas e foram identificados problemas nos valores de corrente lidos: quando a bateria estava sendo carregada, a corrente era consumida pelo sistema, com essa variação de carga, a corrente sofria variações não desejadas para a aplicação.

De modo a evitar as coletas de dados equivocados passou-se a alimentar o sistema através da rede elétrica ao invés de utilizar o sistema de baterias. A solução desse problema está sugerida no capítulo 6 como trabalhos futuros.

Devido à necessidade de dados suficientes para gerar a RNA, instalou-se a placa no bloco W do Instituto Mauá de Tecnologia, conforme ilustrado na Figura 49, de forma que o grupo pudesse visitar a instalação e acompanhar os registros. Os primeiros testes contaram com a rede elétrica como sistema de alimentação, acessada pela tomada de uso geral presente no local, com o sistema de comunicação LoRa funcionando. Ao utilizar a rede para alimentar o dispositivo, a corrente elétrica não sofre mais influência das baterias, assim, os dados colhidos não possuem interferência inapropriada para montar a rede neural.

Figura 49 – Placa instalada no quarto andar do bloco W



Fonte: Os Autores, 2019.

O *dashboard* do LoRa pode ser acessado de qualquer lugar, através do sistema desenvolvido. Os sensores lêem os dados e são processados pelo Arduino que monta uma *string* com 14 *bytes* hexadecimais e que, através do *Dragino* acoplado no sistema, envia para o *gateway* LoRa presente no campus da Mauá. O *gateway* é conectado à internet e permite o acesso dessa mensagem. No notebook, acessa-se a mensagem e a decodifica. Após tratar os dados e desenvolver um *dashboard* para exibi-los, a leitura dos sensores pode ser interpretada por qualquer pessoa e acessada de qualquer lugar.

Através da Figura 50, verifica-se que o sistema coletou e enviou com sucesso os dados de maneira remota para a nuvem. Da mesma forma que nos testes anteriores, foi realizado o armazenamento local através do *SD Card* e verificada comunicação via *Bluetooth* que funcionou corretamente, não apresentando falha em nenhum instante.

Figura 50 – *Dashboard* do LoRa



Fonte: Os Autores, 2019.

Dando sequência na coleta de dados, a placa foi instalada no terraço do bloco W, localizado no campus São Caetano do Sul do Instituto Mauá de Tecnologia, com objetivo de colher dados suficientes para a construção da rede neural artificial. A instalação é ilustrada na Figura 51.

Figura 51 – Placa instalada no terraço do bloco W

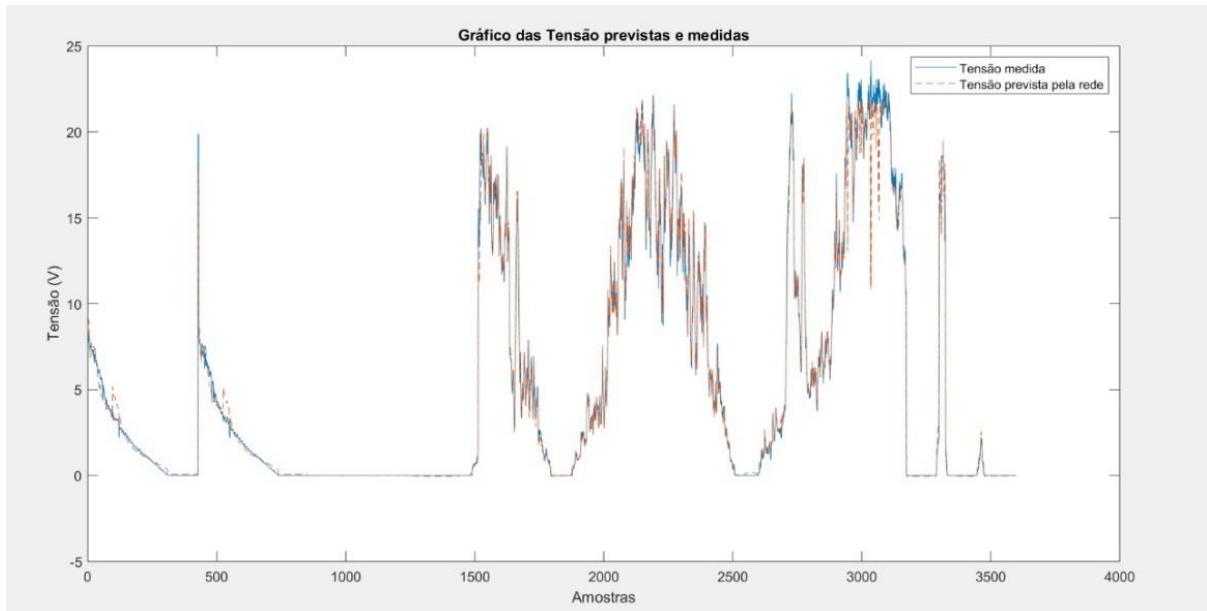


Fonte: Os Autores, 2019.

É importante citar que as leituras foram feitas em clima ensolarado e chuvoso, de dia e a noite. Dessa maneira, o conjunto foi exposto a situação distintas, que é de extrema importância para a construção de uma boa RNA.

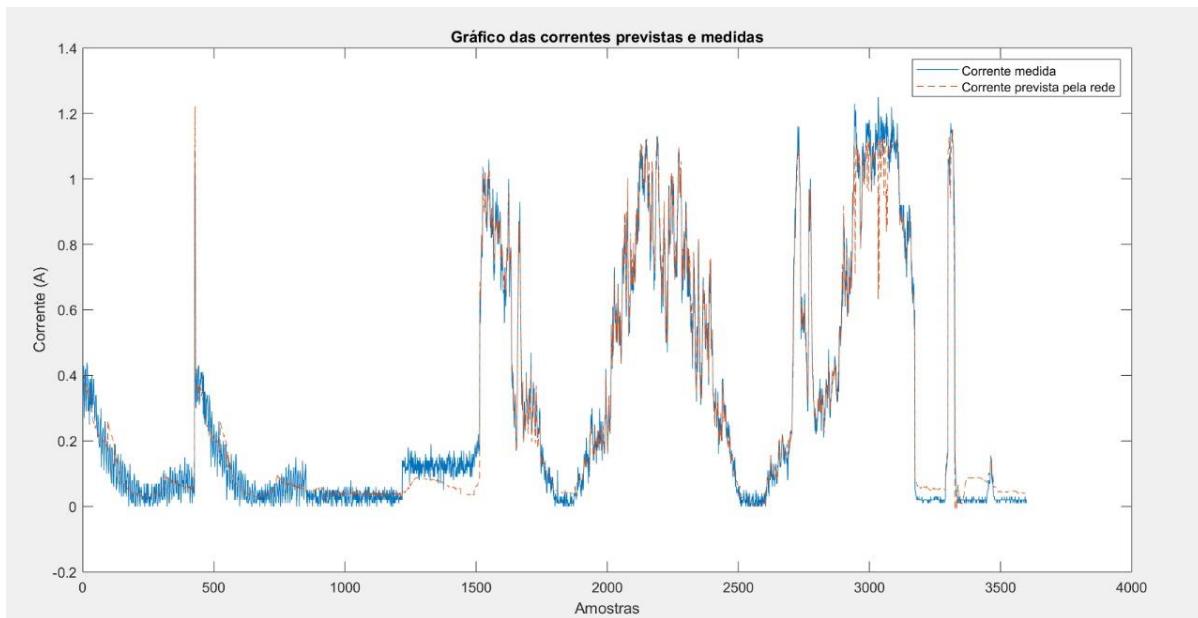
Após 7 dias de coleta, com 7197 amostras extraídas, o passo seguinte foi analisar os valores lidos e descartar os dados duvidosos, dessa maneira a aprendizagem da rede neural não foi prejudicada. A RNA foi montada para prever tensão e corrente elétrica usando a radiação e a temperatura como valores de entrada, conforme ilustram a Figura 52 e Figura 53.

Figura 52 – RNA da tensão elétrica



Fonte: Os Autores, 2019.

Figura 53 – RNA da corrente elétrica



Fonte: Os Autores, 2019.

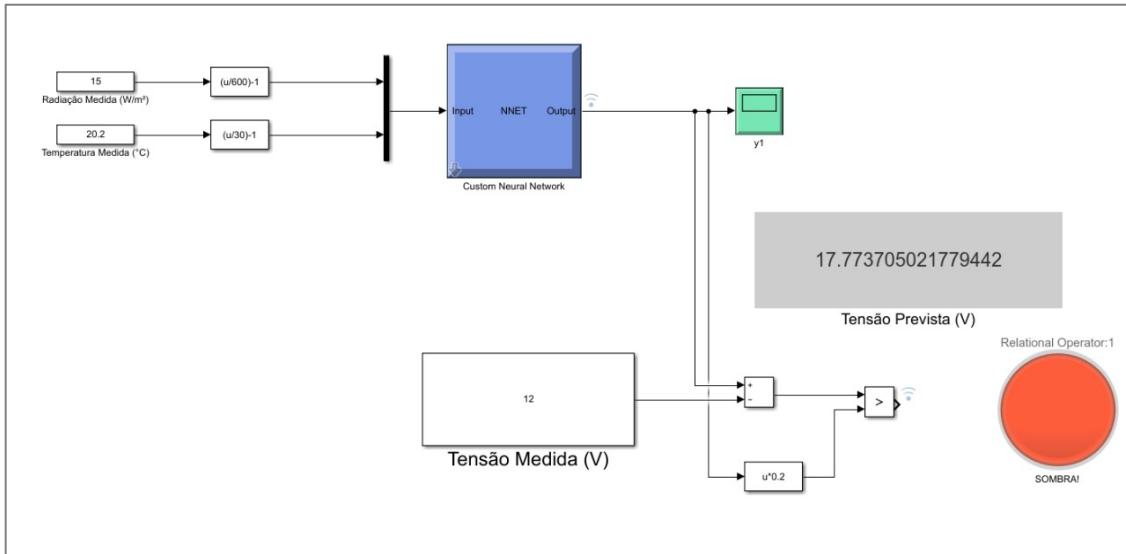
Nota-se que a RNA funciona bem para a aplicação. A Figura 52 ilustra os dados previstos de tensão pela rede neural (curva laranja) e comparando com os dados reais (curva azul), para as mesmas condições de radiação e temperatura, e em condição normal de geração de energia. E a Figura 53 ilustra a resposta da corrente elétrica. A RNA contou com uma única camada escondida, com 7 neurônios. Ao analisar, se percebe que os dados previstos são próximos dos reais, mostrando a eficiência da rede neural.

Vale observar que como utilizou-se um reostato, que é uma carga linear, os dados de tensão e corrente são proporcionais, apresentando curvas visualmente similares, todavia com ordem de grandeza diferente.

A partir disso, o sistema criado conta com inteligência artificial. No Node-RED foi inserida na programação a rede neural. Dessa forma, ao chegarem dados de tensão, corrente, radiação e temperatura, o algoritmo recebe os valores de entrada, radiação e temperatura, e prevê os dados de saída, tensão e corrente. Os dados previstos são comparados com os dados reais de tensão e corrente, e caso apresentar uma discrepância acima da margem de erro, que neste projeto foi adotado o valor de 10 %, representa um desvio de operação. A partir disso, o monitoramento das placas solares é executado com sucesso e atende as expectativas do

projeto. O algoritmo de comparação também foi realizado no *Simulink* (MATHWORKS, 2019), como está exposto na Figura 54 a seguir.

Figura 54 – *Simulink*



Fonte: Os Autores, 2019.

A partir dos valores de radiação e temperatura medidos, a rede neural realizou a predição da tensão de saída. Ao comparar a tensão prevista com a tensão medida, problemas como sombra podem ser detectados, como mostrado acima pelo alerta em vermelho.

Porém para expor na feira de exposição dos trabalhos de graduação (Eureka), realizada no ginásio de esportes do campus São Caetano do Sul do Instituto Mauá de Tecnologia, a fonte de luz primária utilizada foi de um holofote de led de 100 W. A intenção era usar a rede neural extraída do teste anterior, ou seja, com o sol como fonte. Entretanto, a potência gerada pela placa a partir da lâmpada não conseguia suprir a carga, a tensão gerada tendia a zero quando o reostato de carga era conectado.

Mediante esse cenário, houve a necessidade de se treinar uma nova rede neural. Para isso, se iniciou coletas de dados com o circuito em aberto, ou seja, sem carga e usando o holofote como fonte luminosa, vide Figura 55. Para ter dados suficientes para montar a RNA, as coletas de dados foram feitas em diferentes salas, com tipos diferentes de iluminação, incluindo o ginásio, local da exposição do trabalho.

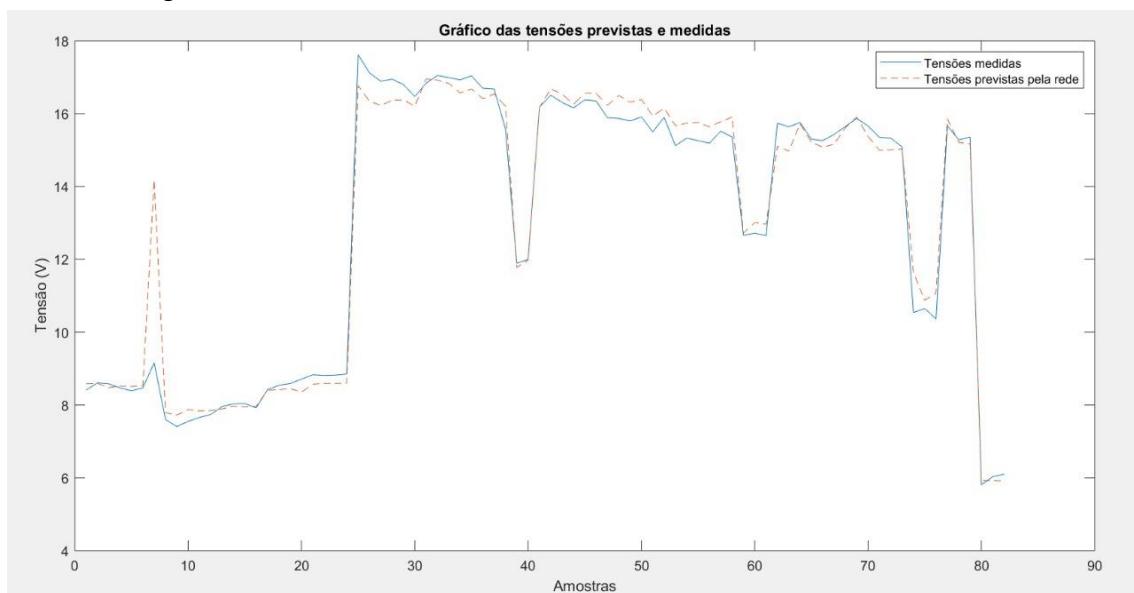
Figura 55 – Coleta de dados com o holofote



Fonte: Os Autores, 2019.

Foram coletados 125 dados nessas condições, 62 amostras foram usadas como valores de treinamento e 63 amostras foram usadas como valores de teste. Para essa nova RNA foram utilizados 4 neurônios em uma única camada escondida para previsão da tensão elétrica. O gráfico das tensões medidas e previstas dessa nova RNA é ilustrado na Figura 56.

Figura 56 – RNA da tensão elétrica com o holofote como fonte de luminosidade



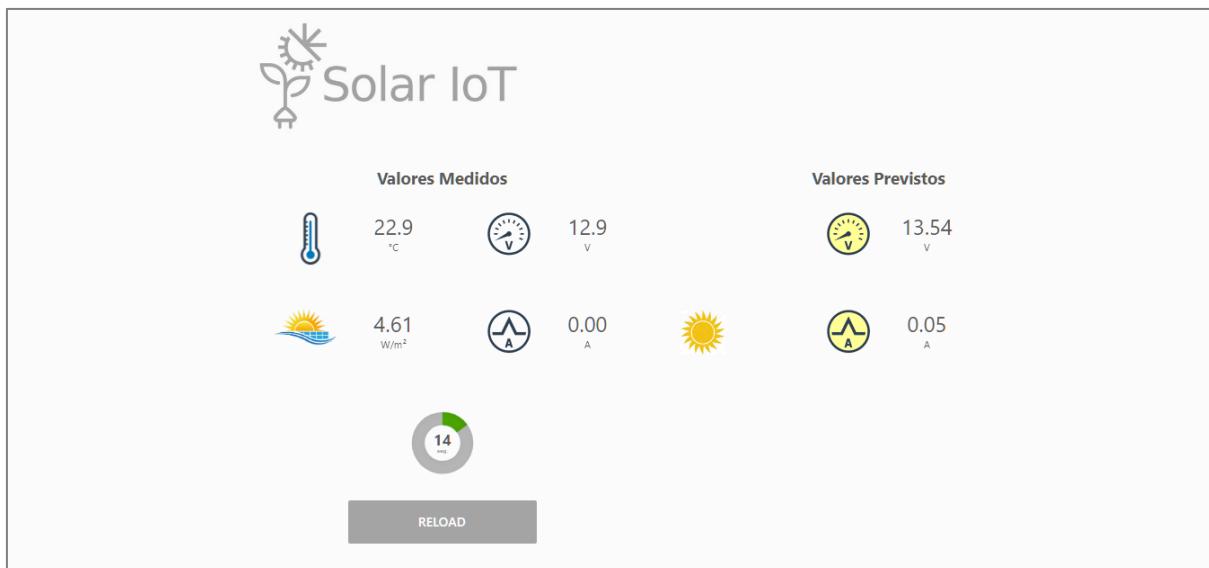
Fonte: Os Autores, 2019.

Nota-se que a RNA atende a necessidade do projeto, pois a curva de valores previstos acompanha a de valores medidos. Em comparação com a primeira rede neural gerada, Figura 52, é possível perceber a necessidade da rede em ter bastante dados para aprendizado, pois assim a RNA fica mais robusta.

Ao extrair os pesos da RNA e programar no Node-RED, o objetivo do projeto foi atingido, pois com a RNA implantada no sistema supervisório, falhas de operação podem ser identificadas.

A Figura 57 mostra o *dashboard* apresentado na feira Eureka. Nele é possível observar os valores lidos de tensão, radiação e temperatura, além de indicar se está chovendo ou não através de um desenho de sol ou de nuvem chovendo. No caso representado, não estava chovendo. Vale lembrar que o valor de corrente não estava sendo medido, pois o circuito estava sem carga.

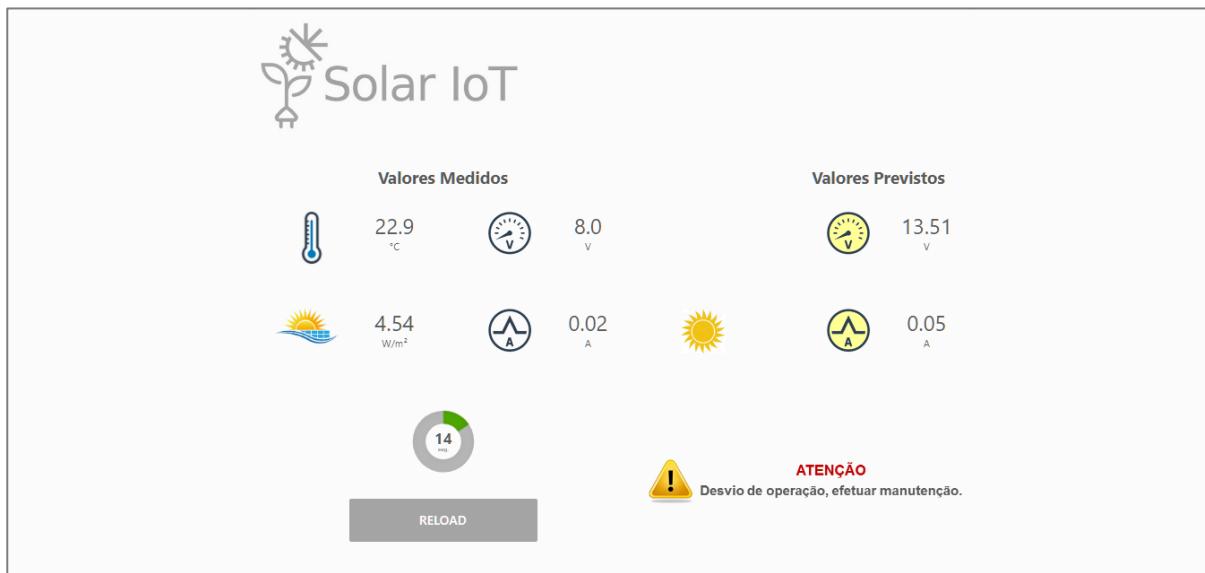
Figura 57 – *Dashboard* em condição normal



Fonte: Os Autores, 2019.

Para testar o sistema de identificação de falhas, foi simulado sombra na placa. Com um papelão, uma parte da placa foi tampada impedindo a entrada da luz. Assim, o sistema reconhecia o desvio na operação, conforme ilustrado na Figura 58, pois a radiação e temperatura que os sensores estavam lendo deveriam gerar uma tensão superior da medida.

Figura 58 – Dashboard com desvio de operação



Fonte: Os Autores, 2019.

Ao se observar a Figura 58, nota-se que para os valores de entrada: 22,9 °C e 4,54 W/m², deveria estar saindo 13,51 V, porém o sensor está lendo 8,0 V, o que configura um desvio de operação e um sinal de alerta é emitido.

Por fim, o aplicativo *Blynk* foi desenvolvido para ser mais um *dashboard*, mas dessa vez, acessado via dispositivo móvel. Através do *Bluetooth*, os dados chegavam ao *smartphone* e eram exibidos, como ilustrado na Figura 59.

No aplicativo são exibidos os valores de temperatura, radiação, corrente e tensão elétrica, além de informar se está chovendo ou não; há ainda uma curva temporal de potência elétrica sendo projetada, assim como seu valor instantâneo. E no final da tela há as medições em formato texto.

Figura 59 – Dashboard Blynk



Fonte: Os Autores, 2019.

5 CONCLUSÕES

O aumento da população mundial, juntamente com os avanços das tecnologias, as grandes demandas de energia da indústria e a grande dependência do ser humano em relação a energia elétrica, tem tornado a busca por novas fontes de energia cada vez mais necessária. Qualquer tipo de fonte de energia não é suficiente para suprir esta demanda, pois além da necessidade energética, há um grande anseio por algo sustentável, já que no cenário atual do mundo há grandes preocupações em relação ao aquecimento global, efeito estufa e outros desastres naturais que ameaçam o futuro da humanidade.

Em meio a esta situação tem-se destacado algumas tecnologias, entre elas estão a geração solar através de painéis fotovoltaicos e a energia eólica por meio de turbinas. Este trabalho concentrou-se principalmente na geração solar e verificou-se que apesar do grande crescimento de geração, há uma grande necessidade por métodos mais eficientes para a determinação da manutenção e identificação de falhas de operação dos painéis. Os grandes avanços da indústria 4.0 possibilita que esta necessidade seja sanada com a aplicação dos conceitos de IoT (Internet das coisas – “*Internet Of Things*”), mais especificamente, utilizando o protocolo de comunicação LoRa que atende as necessidades de comunicação em longas distâncias e confiabilidade suficiente para tal aplicação, já que possivelmente este circuito seria aplicado nas chamadas fazendas de painéis solares, que podem possuir milhares de painéis, onde é necessário uma comunicação em longas distância entre os painéis e uma central de comando.

Para contemplar o objetivo deste trabalho, foram utilizados alguns sensores responsáveis por adquirir dados relevantes para o bom rendimento da geração de energia dos painéis, entre eles o sensor de temperatura, sensor de luminosidade, sensor de corrente elétrica e tensão gerada pelo painel. Para efeitos de confiabilidade do sistema, foram propostos três formas distintas de armazenamento destes dados: salvando os dados em um *SD Card* local; através do compartilhamento e salvamento na nuvem pela comunicação LoRa – Node-RED; e conectando-se via *Bluetooth* com o circuito, utilizando o App *Blynk* que coleta todos os dados instantaneamente do período em que se manteve conectado. O custo total do projeto foi de R\$ 606,16, conforme detalhado no Apêndice G – Tabela de Custos.

Com estes dados coletados sobre diferentes circunstâncias possíveis de operação e muito bem filtrados, foi utilizado o *software* MatLab para treinar uma rede neural MLP capaz de correlacionar os dados e então traçar uma curva de previsão de quanto que o sistema deveria estar gerando de energia. Com esta previsão e com os dados reais de geração instantânea dos painéis, é chegado ao ponto mais relevante do trabalho, onde é feito a comparação de quanto a inteligência artificial está prevendo de geração com o que de fato está sendo gerado. Com esta comparação feita, foi apontado falhas de operação que envolvem a obstrução do painel e então, gerado um alerta para uma central de comando sobre a ineficiência e necessidade de manutenção neste painel em específico.

6 TRABALHOS FUTUROS

Após análises do funcionamento do sistema apresentado neste trabalho, foram levantados pontos de melhoria com o intuito de aperfeiçoar o desempenho sistema. Entre eles destaca-se a observação de que para reduzir os custos ao cliente, uma forma de não colocar um circuito desse em cada painel do proprietário, seria de agrupar painéis em conjuntos e configurar o circuito para operar e detectar falhas nos níveis de tensão, corrente e potência gerada por cada conjunto desses. Outro ponto relevante, seria o aperfeiçoamento do sistema BMS que gerencia o carregamento das baterias do circuito eletrônico, para que consuma menos potência dos painéis solares, já que ao conectar o BMS no circuito a fim de carregar as baterias, ele acaba drenando uma potência considerável do painel. Assim, uma potência que poderia ser aproveitada pela carga, está sendo utilizado para carregamento das baterias provocando uma queda nos valores de tensão gerada do painel. Uma alternativa para solucionar este problema é acrescentar na programação o modo *sleep*, que consiste em deixar o sistema “dormindo” e só “acordar” para realizar a leitura dos sensores e mandar a informação através dos sistemas de comunicação. Tal estratégia, juntamente com o aumento do tempo de envio de amostras, proporciona economia energética ao sistema. Para reduzir os custos e o tamanho físico do sistema, é necessário utilizar uma eletrônica especialmente preparada para o projeto, adotando soluções disponíveis no mercado como placa de circuito impresso e microcontroladores mais compactos, assim como um LoRa *device* de tamanho reduzido. Pensando ainda na eficiência do método proposto, uma melhoria para o sistema seria inserir uma RNA de aprendizagem constante na programação, isso significa que a cada nova amostra recebida pelo sistema, ela seria utilizada para treinamento da rede neural, a aperfeiçoando. Um último ponto a ser melhorado refere-se aos métodos de detecção de falhas, não contemplando apenas a detecção de sombras/obstrução no painel, mas também inserir modelos capazes de detectar curtos circuitos e circuitos abertos internos ao painel que podem gerar falhas adicionais, tais como sobrecorrente ou sub tensão de geração.

REFERÊNCIAS

AGENDA brasileira para a Indústria 4.0. **Indústria 4.0**, 2019. Disponível em: <<http://www.industria40.gov.br/>>. Acesso em: 2019.

AL-DAHOUD, A. et al. **Improving Monitoring and Fault Detection of Solar Panels Using Arduino Mega in WSN**. Londres: [s.n.]. 2015.

ALMONACID, F. et al. Estimation of the energy of a PV generator using artificial neural network. **Elsevier**, 21 Junho 2009.

ARDUINO - HM-10. HM-10, 2019. Disponível em: <<http://arduino.ua/prod1291-modyn-bluetooth-4-0>>. Acesso em: 23 Novembro 2019.

ARDUINO - MEGA. Arduino, 2019. Disponível em: <<https://store.arduino.cc/usa/mega-2560-r3>>. Acesso em: 23 Novembro 2019.

AUGUSTIN MCEVOY, L. C. T. M. **Solar Cells: Materials, Manufacture and Operation**. 2^a. ed. [S.I.]: Academic Press, 2012.

BLYNK. Getting-Started Now. **Documentation**, 2019. Disponível em: <<docs.blyncc>>. Acesso em: 30 November 2019.

CRACIUNESCU, M. et al. **Integration of a solar panel in power microgrid via Internet of Things**. University of Politehnica of Bucharest. Bucharest. 2016.

DIAS, A. J. F. D. C. **Análise de Dados para Previsão de Micro Produção de Energia Solar e Eólica**. Instituto Politécnico de Bragança. Bragança. 2015.

DIGI XBee Ecosystem. **Digi**. Disponível em: <<https://www.digi.com/xbee>>.

ELECTROKIT. Electrokit, 2019. Disponível em: <<https://www.electrokit.com/produkt/stromsensor-ac5712-5a/>>. Acesso em: 23 Novembro 2019.

ELEKTRONIKLAVPRIS. Elektroniklavpris, 2019. Disponível em: <https://elektroniklavpris.dk/images/user_uploaded/2017/140343_1493374818.jpg>. Acesso em: 23 Novembro 2019.

ELETROGATE. eletrogate, 2019. Disponível em: <https://www.eletrogate.com/modulo-regulador-de-tensao-step-up-xl6009e1?utm_source=Site&utm_medium=GoogleMerchant&utm_campaign=GoogleMerchant&gclid=EA1aIQobChMIsZ37kJqQ5AIVCA6RCh1fowidEAQYASABEgL2oPD_BwE>. Acesso em: 23 Novembro 2019.

EMPRESA DE PESQUISA ENERGÉTICA. Matriz Energética e Elétrica, 2018. Disponível em: <<http://epc.gov.br/pt/abcdenergia/matriz-energetica-e-eletrica>>. Acesso em: 15 Maio 2019.

ENGINEERING.COM. Battery Management Systems. **Battery Management Systems**, 2019. Disponível em: <<https://www.engineering.com/productshowcase/batterymanagementsystems.aspx>>. Acesso em: 30 Novembro 2019.

FILHO, A.; SANTIN, R. V.; YANG, R. Computação Móvel 2017. **Seminários**, 2017. Disponível em: <https://www.ime.usp.br/~diogojp/computacao-movel-2017/seminar/rene_santin_LPWAN.pdf>. Acesso em: 2 Junho 2019.

FILIPEFLOP - BH1750. Filipeflop, 2019. Disponível em: <<https://www.filipeflop.com/produto/sensor-de-luz-bh1750fvi-lux/>>. Acesso em: 23 Novembro 2019.

FILIPEFLOP - MÓDULO SD CARD. filipeflop, 2019. Disponível em: <<https://www.filipeflop.com/produto/modulo-cartao-sd-card/>>. Acesso em: Novembro 2019.

FILIPEFLOP - RTC. filipeflop, 2019. Disponível em: <<https://www.filipeflop.com/produto/real-time-clock-rtc-ds1307/>>. Acesso em: Novembro 2019.

FILIPEFLOP - SENSOR DE CHUVA. Filipeflop, 2019. Disponível em: <<https://www.filipeflop.com/produto/sensor-de-chuva/>>. Acesso em: 23 Novembro 2019.

GAMBOGI, J. A. **Aplicação de Redes Neurais na tomada de decisão no mercado de ações**. Escola Politécnica da Universidade de São Paulo. São Paulo. 2013.

GAROUDJA, E. et al. Statistical fault detection in photovoltaic systems. **Elsevier**, 8 Maio 2017.

GROBOTRONICS. grobotronics, 2019. Disponível em: <<https://grobotronics.com/li-ion-battery-charger-protection-module-3s-10a.html>>. Acesso em: 23 Novembro 2019.

GUO, D. Instructables Circuits. **Instructables Circuits**, 2019. Disponível em: <<https://www.instructables.com/id/Use-the-LoRa-Kit-to-Build-Your-Own-IoT-Network/>>.

HAYKIN, S. **Redes Neurais: Princípios e Prática**. 2^a. ed. [S.I.]: Bookman, 2003.

HMAMOUCHI, R. et al. redictive modeling of the activity of coumarin derivatives DL50 using neural statistical approaches: Electronic descriptors - based DFT, Journal of Taibah University for Science, 10 Junho 2015.

INSTITUTO MAUÁ DE TECNOLOGIA. Exemplo de Aplicação MQTT Broker (Node-RED). **Smart Campus Mauá**, 2018. Disponível em: <https://smartcampus.maua.br/?page_id=252>. Acesso em: 23 Novembro 2019.

IRENA. International Renewable Energy Agency, 2019. Disponível em: <<https://www.irena.org/newsroom/pressreleases/2019/Apr/Renewable-Energy-Now-Accounts-for-a-Third-of-Global-Power-Capacity>>. Acesso em: 27 Maio 2019.

MAGO, V. K.; BHATIA, N. **Cross-Disciplinary Applications of Artificial Intelligence and Pattern Recognition: Advancing Technologies.** [S.I.]: [s.n.], 2012.

MATHWORKS. nntool. **Documentation**, 2019. Disponível em: <https://www.mathworks.com/help/deeplearning/ref/nntool.html?searchHighlight=nntool&s_tid=doc_srcTitle>. Acesso em: 05 Dezembro 2019.

MATHWORKS. Simulink. **Documentation**, 2019. Disponível em: <<https://www.mathworks.com/help/simulink/>>. Acesso em: 05 Dezembro 2019.

MORGAN, J. A Simple Explanation of 'The Internet of Things'. **Forbes**, 2014. Disponível em: <<https://www.forbes.com/sites/jacobmorgan/2014/05/13/simple-explanation-internet-things-that-anyone-can-understand/#6382ec901d09>>.

NODE-RED. Documentação. **Node-RED**, 2019. Disponível em: <<https://nodered.org/about/>>. Acesso em: 20 Novembro 2019.

NOVIDÁ. Conectividade e a principal barreira à massificação de estratégias IoT no Brasil. **Conectividade**, 2019. Disponível em: <<https://novidá.com.br/blog/conectividade/>>. Acesso em: 2 Junho 2019.

PIRES, C. P. O. **Identification of residential electricity consumption profiles for building dynamic simulation through smart-meter data**. Técnico Lisboa. Lisboa, p. 114. 2017.

RENEWABLE Energy Now Accounts for a Third of Global Power Capacity. **International Renewable Energy Agency**, 2019. Disponível em: <<https://www.irena.org/newsroom/pressreleases/2019/Apr/Renewable-Energy-Now-Accounts-for-a-Third-of-Global-Power-Capacity>>. Acesso em: 27 Maio 2019.

SAMSUNG. fulgurbattman, 2019. Disponível em: <<https://eshop.fulgurbattman.cz/priloha.php?ak=13210>>. Acesso em: 24 Novembro 2019.

SANTOS, B. P. et al. **Internet das Coisas: da Teoria à Prática**. Universidade Federal de Minas Gerais (UFMG). Belo Horizonte, MG, Brasil, p. 50. 2017.

SHENG, Q. Z. et al. **Managing the Web of Things: Linking the Real World to the**. [S.I.]: Morgan Kaufmann, 2017.

STÄUBLI GROUP. **Catalogue for Installers: MC3 and MC4**. Allschwil, p. 16. 2013.

TACBATTERY. tacbattery., 2019. Disponível em: <<http://portuguese.tacbattery.com/sale-2034630-cell-3-7v-lithium-ion-cylindrical-battery-18650-2200mah-notebook-li-ion-batteries.html>>. Acesso em: 23 Novembro 2019.

USAINFO. Usainfo, 2019. Disponível em: <<https://www.usinainfo.com.br/sensores/sensor-de-tensao-dc-0-25v-para-arduino-p25-4364.html>>. Acesso em: 23 Novembro 2019.

WHAT is Lora? **Semtech**, 2019. Disponível em: <<https://www.semtech.com/lora/what-is-lora>>. Acesso em: 31 Maio 2019.

WHAT is the SmartGrid? **SmartGrid.gov**, 2019. Disponível em: <https://www.smartgrid.gov/the_smart_grid/smart_grid.html>. Acesso em: 30 Maio 2019.

WIKI.DRAGINO. wiki.dragino, 2019. Disponível em: <http://wiki.dragino.com/index.php?title=Lora_Shield>. Acesso em: 24 Novembro 2019.

YINGLI. energiatotal, 2019. Disponível em: <<https://www.energatotal.com.br/painel-solar-95w-yingli-yl095p-17b>>. Acesso em: 24 Novembro 2019.

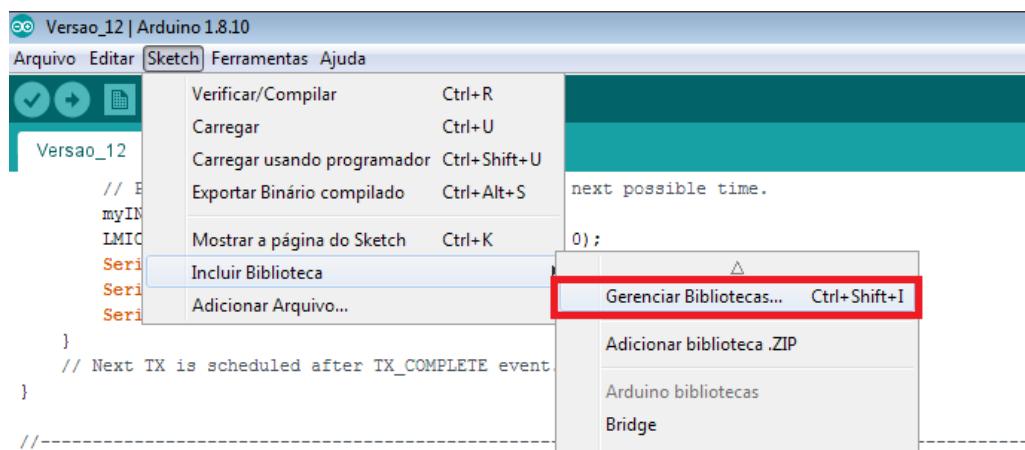
ZIGBEE Wireless Technology Architecture and Applications. **El PRO CUS**. Disponível em: <<https://www.elprocus.com/what-is-zigbee-technology-architecture-and-its-applications/>>.

ZILLES, R. et al. **Sistemas Fotovoltaicos Conectados à Rede Elétrica**. 1st. ed. [S.I.]: Oficina de Textos, 2012.

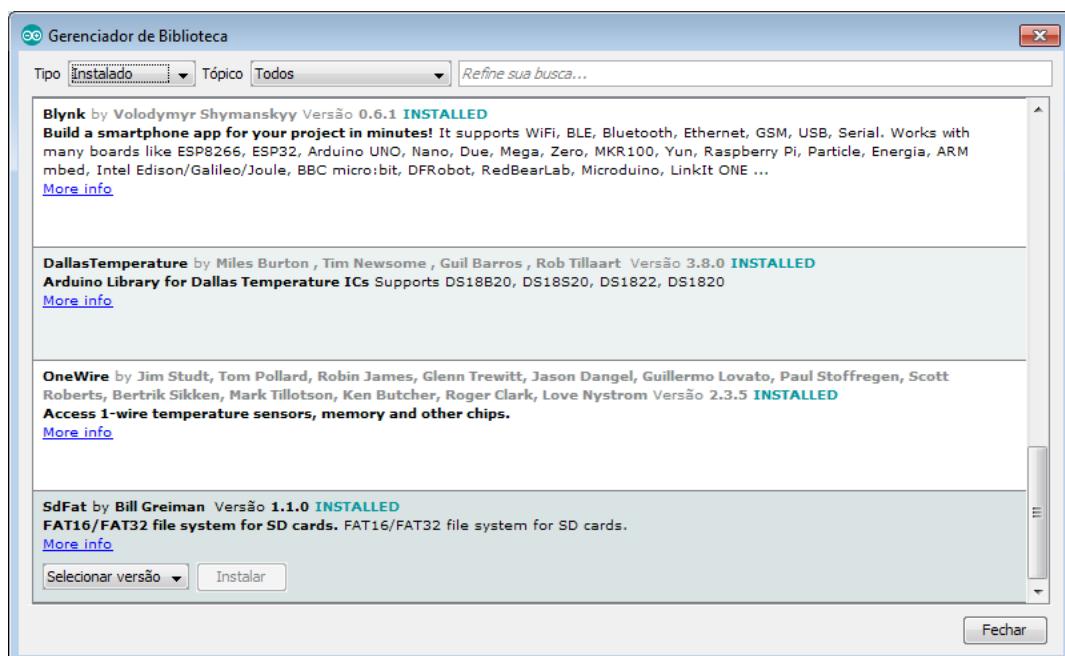
APÊNDICE A – INSTALAÇÃO DAS BIBLIOTECAS DO ARDUINO

Este apêndice tem por finalidade apresentar um guia para a configuração das bibliotecas do Arduino utilizadas neste trabalho. Para a correta compilação do *firmware* apresentado no Apêndice C – Código fonte do dispositivo coletor, recomenda-se que os seguintes itens sejam realizados.

1. Na IDE do Arduino, clicar no menu **Sketch > Incluir Biblioteca > Gerenciar Bibliotecas...**;

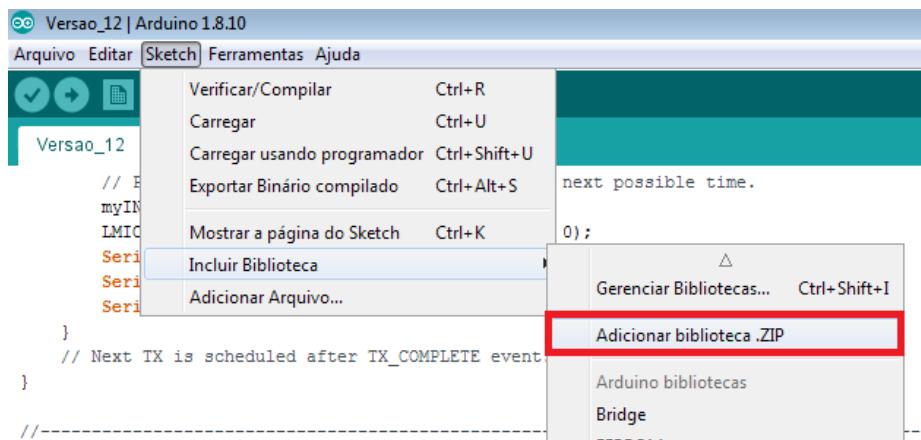


2. Instalar as bibliotecas **Blynk**, **Dallas Temperature**, **OneWire** e **SdFat**;

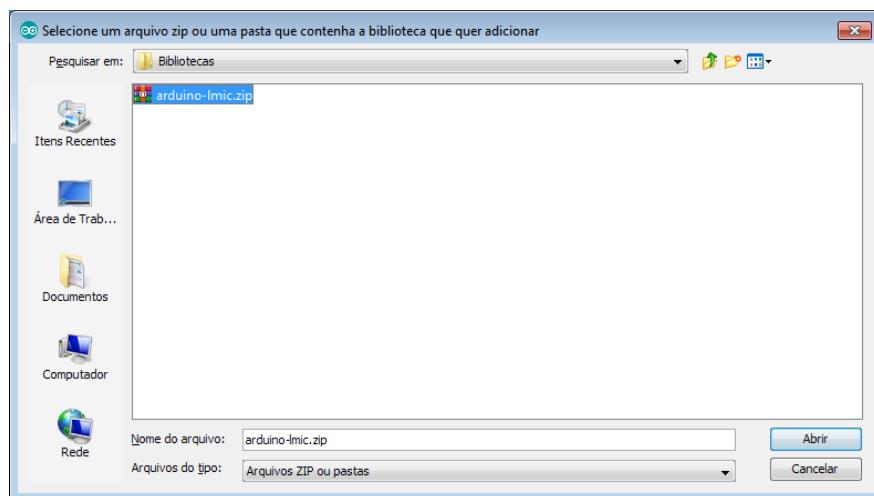


3. Acessar o endereço <https://github.com/matthijskooijman/arduino-lmic> e fazer o *download* do arquivo **arduino-LMIC**.

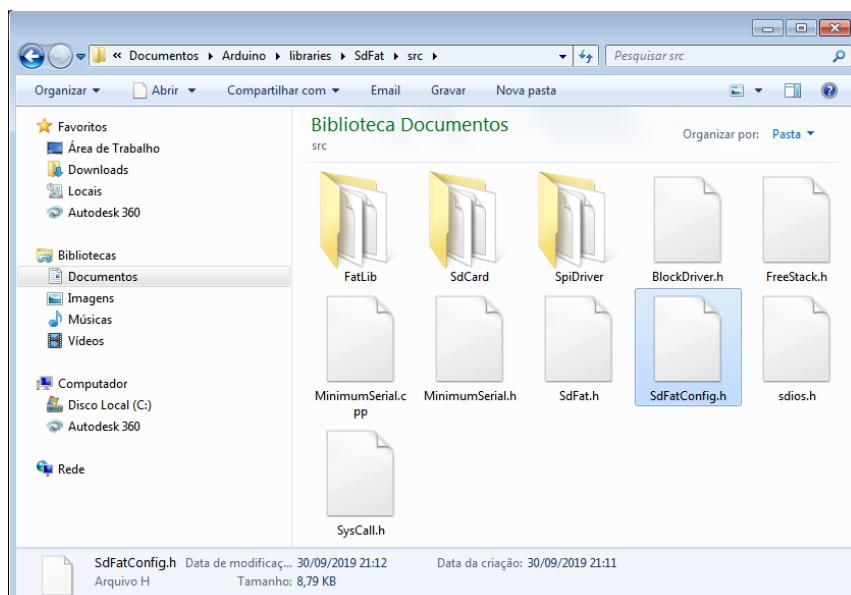
4. Clicar no menu **Sketch > Incluir Biblioteca > Adicionar biblioteca .ZIP** ;



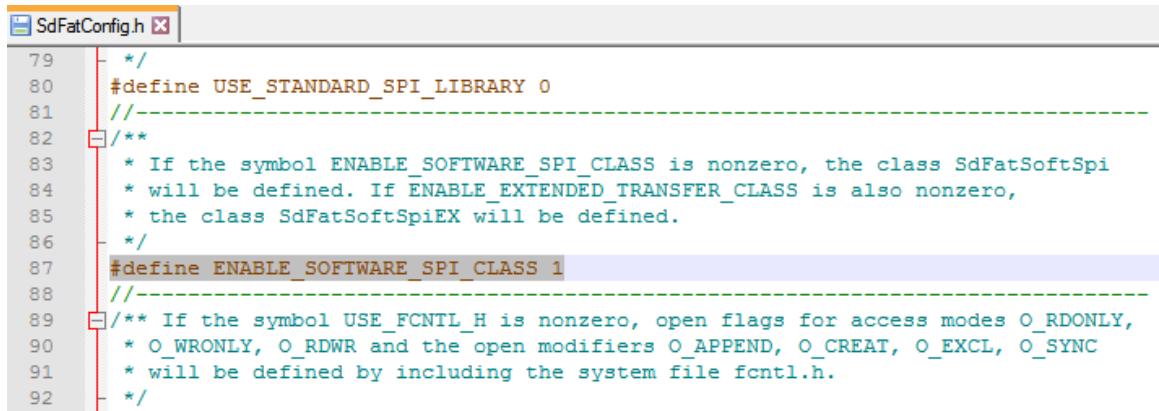
5. Selecionar o arquivo **arduino-lmic.zip** e clicar em **Abrir**;



6. Abrir o arquivo **C:\Users\....\Documents\Arduino\libraries\SdFat\src\SdFatConfig.h**;



7. Alterar o parâmetro da linha 87 para 1;



```
79  */
80 #define USE_STANDARD_SPI_LIBRARY 0
81 /**
82  * If the symbol ENABLE_SOFTWARE_SPI_CLASS is nonzero, the class SdFatSoftSpi
83  * will be defined. If ENABLE_EXTENDED_TRANSFER_CLASS is also nonzero,
84  * the class SdFatSoftSpiEX will be defined.
85  */
86 #define ENABLE_SOFTWARE_SPI_CLASS 1
87 /**
88  * If the symbol USE_FCNTL_H is nonzero, open flags for access modes O_RDONLY,
89  * O_WRONLY, O_RDWR and the open modifiers O_APPEND, O_CREAT, O_EXCL, O_SYNC
90  * will be defined by including the system file fcntl.h.
91  */
92 */
```


APÊNDICE B – INSTALAÇÃO E CONFIGURAÇÃO DO NODE-RED

Este apêndice tem por finalidade apresentar um guia para a instalação e configuração do Node-RED. Para a correta interpretação da rede de nós apresentada no item 3.2.3 pela ferramenta, é imprescindível que os seguintes itens sejam realizados.

1. Acessar o site do Node.js no link <https://nodejs.org/en/>;
2. Clicar na opção **10.16.3 LTS - Recommended for Most Users**;

Download for Windows (x64)



3. Instalar o Node.js;

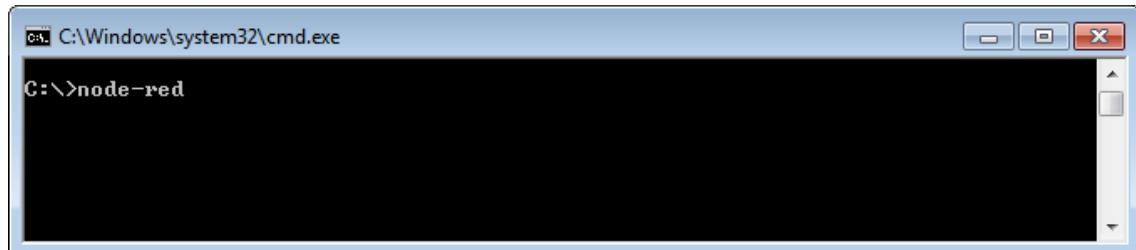


4. Após a instalação do Node.js, abrir o prompt de comando do Windows e digitar o seguinte comando: **npm install -g --unsafe-perm node-red**;

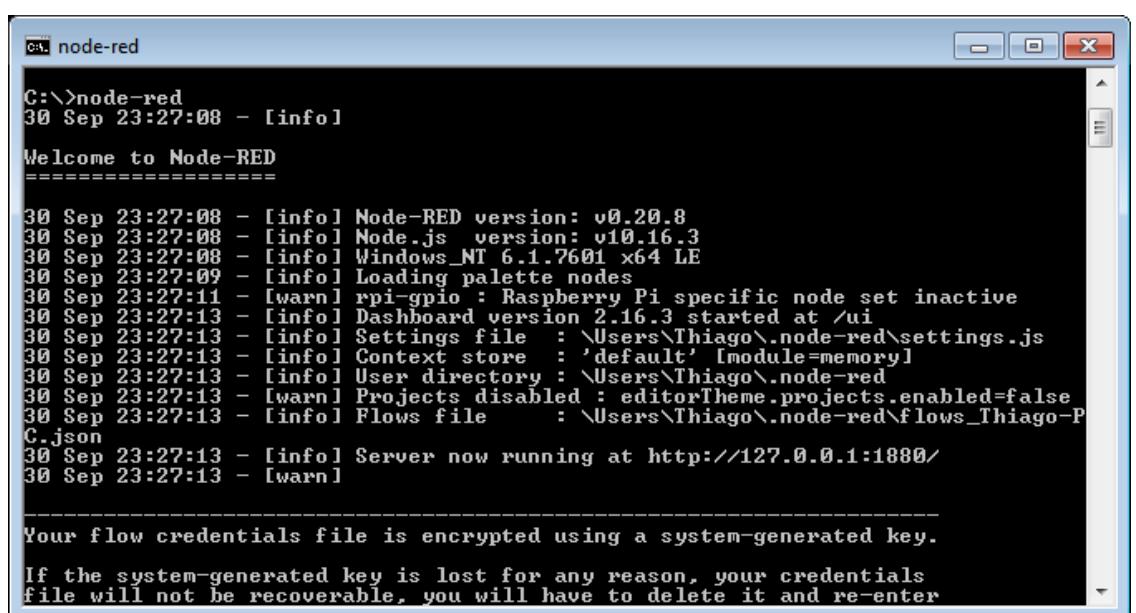
The image shows a screenshot of a Windows Command Prompt window with the title 'C:\Windows\system32\cmd.exe'. The command 'C:\>npm install -g --unsafe-perm node-red' is visible in the terminal window.

```
C:\>npm install -g --unsafe-perm node-red
```

5. Finalizada a instalação, fechar e abrir novamente o prompt de comando do Windows e digitar **node-red**;



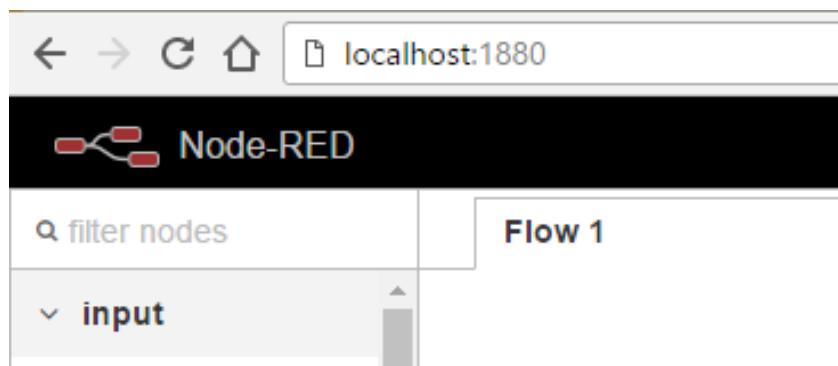
```
C:\>node-red
```

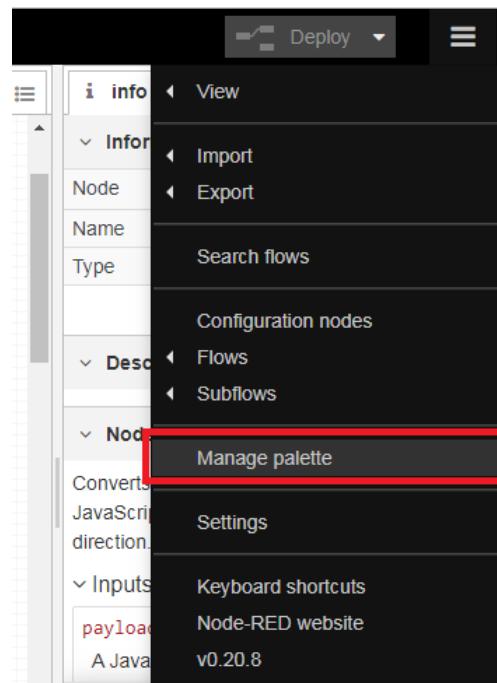
```
C:\>node-red
30 Sep 23:27:08 - [info]
30 Sep 23:27:08 - [info] Welcome to Node-RED
=====
30 Sep 23:27:08 - [info] Node-RED version: v0.20.8
30 Sep 23:27:08 - [info] Node.js version: v10.16.3
30 Sep 23:27:08 - [info] Windows_NT 6.1.7601 x64 LE
30 Sep 23:27:09 - [info] Loading palette nodes
30 Sep 23:27:11 - [warn] rpi-gpio : Raspberry Pi specific node set inactive
30 Sep 23:27:13 - [info] Dashboard version 2.16.3 started at /ui
30 Sep 23:27:13 - [info] Settings file : \Users\Thiago\.node-red\settings.json
30 Sep 23:27:13 - [info] Context store : 'default' [module=memory]
30 Sep 23:27:13 - [info] User directory : \Users\Thiago\.node-red
30 Sep 23:27:13 - [warn] Projects disabled : editorTheme.projects.enabled=false
30 Sep 23:27:13 - [info] Flows file : \Users\Thiago\.node-red\flows_Thiago-P.json
30 Sep 23:27:13 - [info] Server now running at http://127.0.0.1:1880/
30 Sep 23:27:13 - [warn]

-----
Your flow credentials file is encrypted using a system-generated key.
If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
```

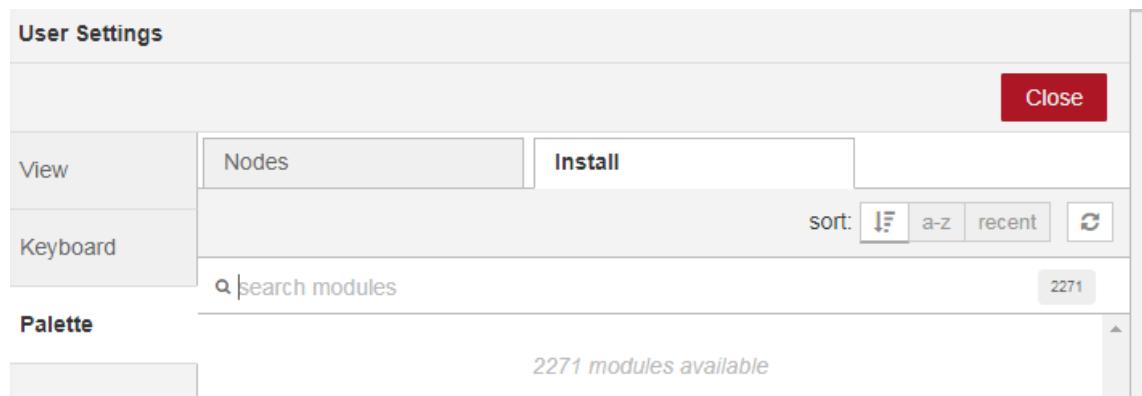
6. Em seguida é necessário abrir uma aba do navegador e digitar o seguinte endereço: **localhost:1880**;



7. Clicar na aba **Menu > Manage palette**;



8. Clicar na aba **Install**;



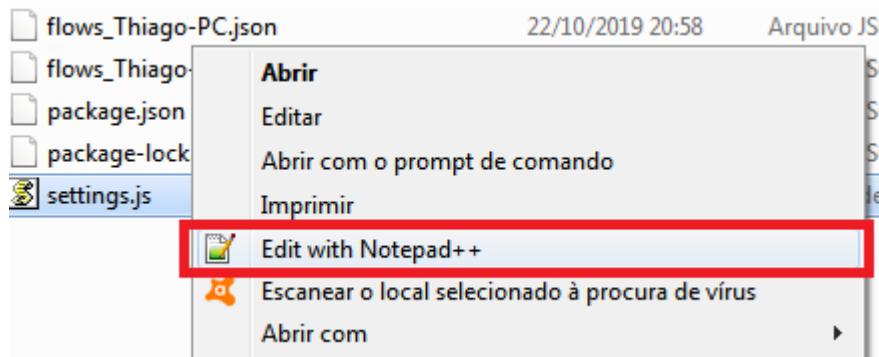
9. Procurar e instalar as bibliotecas: ***node-red-contrib-crypto-js-mod (v0.1.1)*** e ***node-red-dashboard (v2.16.3)***;

10. Copiar a pasta Solar_IoT para o diretório "**C:\Users\ Usuario \Documents**"

11. Acessar a pasta "**C:\Users\ Usuario\.node-red**"



12. Clicar com o botão direito sobre o arquivo **settings.js** e, em seguida, editar o arquivo;



13. Adicionar a linha de comando: **httpStatic: 'C:\\\\Users\\\\ Usuário\\\\Documents\\\\Solar_IoT'**,

```
// When httpAdminRoot is used to move the UI to a different root path, the
// following property can be used to identify a directory of static content
// that should be served at http://localhost:1880/.
//httpStatic: '/home/nol/node-red-static/',
httpStatic: 'C:\\\\Users\\\\Thiago\\\\Documents\\\\Solar_IoT',
```

APÊNDICE C – CÓDIGO FONTE DO DISPOSITIVO COLETOR

Este apêndice tem por finalidade apresentar o código fonte inserido no Arduino Mega, integrante do *hardware* do dispositivo coletor de dados.

```
//---: TCC - Solar IoT - MONITORAMENTO DE PLACAS SOLARES POR IoT ::---\\
//----- Blynk -----
/* Comment this out to disable prints and save space */
#define BLYNK_PRINT Serial

#include <BlynkSimpleSerialBLE.h>

// You should get Auth Token in the Blynk App.
// Go to the Project Settings (nut icon).
char auth[] = "NKXjpKIP4fgA4Q8_P6CFZnkcg981mg8M";

/* Serial with Bluetooth to communicate with Blynk */
#define BLYNK_BLE Serial1

/* SolarIot Blynk App Virtual Pins */
#define POWER_GAUGE_VPIN      V0
#define VOLTAGE_HLEVEL_VPIN   V1
#define CURRENT_HLEVEL_VPIN  V2
#define IRRADIATION_LVALUE_VPIN V3
#define TEMPERATURE_LVALUE_VPIN V4
#define POWER_SUPERCHART_VPIN V5
#define TERMINAL_VPIN         V6
#define RAIN_IMGG_VPIN        V7

// Attach virtual serial terminal to Virtual Pin TERMINAL_VPIN
WidgetTerminal BLE_Terminal(TERMINAL_VPIN);

BlynkTimer ExecutionTimerMs; /* Announcing the Execution timer */

//----- LoRa -----
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>

// LoRaWAN NwkSKey, network session key. This is the default Semtech key,
// which is used by the prototype TTN network initially.
static const PROGMEM u1_t NWKSKEY[16] = { 0x2B, 0x7E, 0x15, 0x16, 0x28,
0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C };

// LoRaWAN AppSKey, application session key. This is the default Semtech
// key, which is used by the prototype TTN network initially.
static const u1_t PROGMEM APPSKEY[16] = { 0x2B, 0x7E, 0x15, 0x16, 0x28,
0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C };

// LoRaWAN end-device address (DevAddr)
static const u4_t DEVADDR = 0x03ff0001 ;

// This EUI must be in little-endian format, so least-significant-byte
// first. When copying an EUI from ttncctl output, this means to reverse
```

```

// the bytes. For TTN issued EUIs the last bytes should be 0xD5, 0xB3, 0x70.
static const u1_t PROGMEM APPEUI[8] = { 0x11, 0x11, 0x11, 0x11, 0x11,
0x11, 0x11, 0x11 };
void os_getArtEui (u1_t* buf) {
    memcpy_P(buf, APPEUI, 8);
}

// This should also be in little endian format, see above.
static const u1_t PROGMEM DEVEUI[8] = { 0x00, 0x00, 0x00, 0x00, 0x03,
0xff, 0x00, 0x01 };
void os_getDevEui (u1_t* buf) {
    memcpy_P(buf, DEVEUI, 8);
}

// This key should be in big endian format (or, since it is not really a
// number but a block of memory, endianness does not really apply). In
// practice, a key taken from the TTN console can be copied as-is.
static const u1_t PROGMEM APPKEY[16] = { 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
void os_getDevKey (u1_t* buf) {
    memcpy_P(buf, APPKEY, 16);
}

//Inicializando mydata - Variavel tipo array enviado ao SmartCampus do IMT
static uint8_t mydata[] = "XXXXXXXXXXXXXX";

//Inicializando myINF - Variavel tipo String inserido do array mydata[]
String myINF = "";

static osjob_t sendjob;
const unsigned TX_INTERVAL = 1; // Tempo TX - Envio para o SmartCampus.

// Pin mapping for Dragino Lorashield
const lmic_pinmap lmic_pins = {
    .nss = 10,
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 9,
    .dio = {2, 6, 7},
};

void onEvent (ev_t ev) {
    Serial.print(os_getTime());
    Serial.print(": ");
    switch (ev) {
        case EV_SCAN_TIMEOUT:
            Serial.println(F("EV_SCAN_TIMEOUT"));
            break;
        case EV_BEACON_FOUND:
            Serial.println(F("EV_BEACON_FOUND"));
            break;
        case EV_BEACON_MISSED:
            Serial.println(F("EV_BEACON_MISSED"));
            break;
        case EV_BEACON_TRACKED:
            Serial.println(F("EV_BEACON_TRACKED"));
            break;
        case EV_JOINING:
            Serial.println(F("EV_JOINING"));
            break;
        case EV_JOINED:
            Serial.println(F("EV_JOINED"));
    }
}

```

```

        break;
    case EV_JOIN_FAILED:
        Serial.println(F("EV_JOIN_FAILED"));
        break;
    case EV_REJOIN_FAILED:
        Serial.println(F("EV_REJOIN_FAILED"));
        break;
        break;
    case EV_TXCOMPLETE:
        Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
        if (LMIC.txrxFlags & TXRX_ACK)
            Serial.println(F("Received ack"));
        if (LMIC.dataLen) {
            Serial.print(F("Received "));
            Serial.print(LMIC.dataLen);
            Serial.println(F(" bytes of payload"));
        }
        // Schedule next transmission
        os_setTimedCallback(&sendjob, os_getTime() +
sec2osticks(TX_INTERVAL), do_send);
        break;
    case EV_LOST_TSYNC:
        Serial.println(F("EV_LOST_TSYNC"));
        break;
    case EV_RESET:
        Serial.println(F("EV_RESET"));
        break;
    case EV_RXCOMPLETE:
        // data received in ping slot
        Serial.println(F("EV_RXCOMPLETE"));
        break;
    case EV_LINK_DEAD:
        Serial.println(F("EV_LINK_DEAD"));
        break;
    case EV_LINK_ALIVE:
        Serial.println(F("EV_LINK_ALIVE"));
        break;
    case EV_TXSTART:
        Serial.println(F("EV_TXSTART"));
        break;
    default:
        Serial.print(F("Unknown event: "));
        Serial.println((unsigned) ev);
        break;
    }
}

void do_send(osjob_t* j) {
// Check if there is not a current TX/RX job running
if (LMIC.opmode & OP_TXRXPEND) {
    Serial.println(F("OP_TXRXPEND, not sending"));
} else {
    // Prepare upstream data transmission at the next possible time.
    myINF.toCharArray(mydata, sizeof(mydata));
    LMIC_setTxData2(1, mydata, sizeof(mydata) - 1, 0);
    Serial.println(F("Packet queued"));
    Serial.print(F("Sending packet on frequency: "));
    Serial.println(LMIC.freq);
}
// Next TX is scheduled after TX_COMPLETE event.
}

```

```

}

//----- LoRa -----
float PLACA_TENSAO = 0;
double PLACA_CORRENTE = 0;
float PLACA_IRRAD = 0;
float PLACA_TEMP = 0;
int PLACA_CHUVA = 0;

//----- SD Card -----
#include <SdFat.h>

// Pin numbers in templates must be constants.
const uint8_t SOFT_MISO_PIN = 30;
const uint8_t SOFT_MOSI_PIN = 31;
const uint8_t SOFT_SCK_PIN = 32;
// Chip select may be constant or RAM variable.
const uint8_t SD_CHIP_SELECT_PIN = 33;

SdFatSoftSpi<SOFT_MISO_PIN, SOFT_MOSI_PIN, SOFT_SCK_PIN> SdFatDev;
File myFile;
String strIndex = "Indice";
String strRTC = "RTC";
String strTensao = "Tensao";
String strCorrente = "Corrente";
String strChuva = "Chuva";
String strIrradiacao = "Irradiacao";
String strTemperatura = "Temperatura";

//----- RTC -----
#include <Wire.h>
#define DS1307_ADDRESS 0x68
byte zero = 0x00;

//----- SENSOR DE TENSÃO -----
float tensaoEntrada = 0;
float tensaoMedida = 0;
float offset = 0.4124;
float valorR1 = 31000;
float valorR2 = 6700;
int leituraSensor = 0;

//----- SENSOR DE CORRENTE -----
#define qtde 100
double vetCorrente[qtde];

//----- CÁLCULO DA POTENCIA -----
float potenciaInstantanea = 0.0;

//----- SENSOR DE CHUVA -----
int thresholdValue = 500;

//----- SENSOR DE LUMINOSIDADE (RADIAÇÃO) -----
#include <Wire.h>
int BH1750_Device = 0x23;
unsigned int Scaled_FtCd;
float Wattsm2;
int Lux;
int i;

```

```

//----- SENSOR DE TEMPERATURA-----
#include <OneWire.h>
#include <DallasTemperature.h>
OneWire pino(3); // pino digital 3
DallasTemperature barramento(&pino);
DeviceAddress sensor;

//----- VARIAVEIS PARA MEDIA DE 5 MINUTOS -----
float AcumuladorTensao = 0.0;
double AcumuladorCorrente = 0.0;
float AcumuladorIrradiacao = 0.0;
unsigned int ContadorGravacoes = 0;
int HoraAtual = 0;

//----- TEMPORIZACAO -----
const unsigned long int TempoGravacaoMs = 1UL * 60UL * 1000UL; // Intervalo
de tempo entre gravacoes, em ms (1 min)
unsigned int ContadorAmostras = 0;
const unsigned int QuantidadeAmostras = 10; // Quantidade de amostras entre
gravações
const unsigned long int IntervaloTempoMs = TempoGravacaoMs /
QuantidadeAmostras; // Intervalo de tempo entre iterações, em ms

//::::::::::::::::::: INICIALIZAÇÃO DO PROGRAMA :::::::::::::::\

void GravarSD() {
    BLE_Terminal.println ();
    BLE_Terminal.println ("-----");
    BLE_Terminal.println ();
    BLE_Terminal.println("Gravando dados no cartao:");
    myFile = SdFatDev.open("dados.txt", FILE_WRITE);
    // if the file opened okay, write to it:
    if (myFile) {
        myFile.print(strIndex);
        myFile.print("\t");
        myFile.print(strRTC);
        myFile.print("\t");
        myFile.print(strTensao);
        myFile.print("\t");
        myFile.print(strCorrente);
        myFile.print("\t");
        myFile.print(strChuva);
        myFile.print("\t");
        myFile.print(strIrradiacao);
        myFile.print("\t");
        myFile.println(strTemperatura);
        // close the file:
        myFile.close();
        // print
        BLE_Terminal.print(strIndex);
        BLE_Terminal.print("\t");
        BLE_Terminal.print(strRTC);
        BLE_Terminal.print("\t");
        BLE_Terminal.print(strTensao);
        BLE_Terminal.print("\t");
        BLE_Terminal.print(strCorrente);
        BLE_Terminal.print("\t");
        BLE_Terminal.print(strChuva);
        BLE_Terminal.print("\t");
        BLE_Terminal.print(strIrradiacao);
        BLE_Terminal.print("\t");
    }
}

```

```

    BLE_Terminal.println(strTemperatura);
    ContadorGravacoess++;
    strIndex = String(ContadorGravacoess);
} else {
    // if the file didn't open, print an error:
    BLE_Terminal.println("Erro abrindo DadosSolar.txt");
}
}

void ExecSolarIot();
void InitSdCard(void);
void InitRtc(void);
void InitTensao(void);
void InitCorrente(void);
void InitChuva(void);
void InitIrradiacao(void);
void InitTemperatura(void);
void ExecRtc(void);
float ExecTensao(void);
double ExecCorrente(void);
int ExecChuva(void);
float ExecIrradiacao(void);
float ExecTemperatura(void);
void ExecSdCard(void);
//::::::DEFINIÇÕES DE FUNÇÕES DO RTC E DO SENSOR DE LUMINOSIDADE::::::\\
//-----RTC (Real Time Clock)-----\\

void setDate() {

    byte segundo =      00; //0-59
    byte minuto =       59; //0-59
    byte hora =         16; //0-23
    byte diasemana =   3; //1-7
    byte dia =          25; //1-31
    byte mes =          9; //1-12
    byte ano =          19; //0-99

    Wire.beginTransmission(DS1307_ADDRESS);
    Wire.write(zero);
    Wire.write(decToBcd(segundo));
    Wire.write(decToBcd(minuto));
    Wire.write(decToBcd(hora));
    Wire.write(decToBcd(diasemana));
    Wire.write(decToBcd(dia));
    Wire.write(decToBcd(mes));
    Wire.write(decToBcd(ano));
    Wire.write(zero);
    Wire.endTransmission();
}

byte decToBcd(byte val) {
    // Conversão de decimal para binário
    return ( (val / 10 * 16) + (val % 10) );
}

byte bcdToDec(byte val)  {

    // Conversão de binário para decimal
    return ( (val / 16 * 10) + (val % 16) );
}

void printDate() {
    Wire.beginTransmission(DS1307_ADDRESS);
    Wire.write(zero);
}

```

```

Wire.endTransmission();
Wire.requestFrom(DS1307_ADDRESS, 7);
int segundo = bcdToDec(Wire.read());
int minuto = bcdToDec(Wire.read());
int hora = bcdToDec(Wire.read() & 0b111111); //Formato 24 horas
int diasemana = bcdToDec(Wire.read()); //0-6 -> Domingo - Sábado
int dia = bcdToDec(Wire.read());
int mes = bcdToDec(Wire.read());
int ano = bcdToDec(Wire.read());
//Exibe a data e hora. Ex.: Data: 3/12/13 - Hora: 19:00:00
// BLE_Terminal.println (" ");
BLE_Terminal.print("Data: ");
BLE_Terminal.print(strPadInt(dia, 2));
BLE_Terminal.print("/");
BLE_Terminal.print(strPadInt(mes, 2));
BLE_Terminal.print("/");
BLE_Terminal.print(strPadInt(ano, 2));
BLE_Terminal.print(" ");
BLE_Terminal.print("Hora:");
BLE_Terminal.print(" ");
BLE_Terminal.print(strPadInt(hora, 2));
BLE_Terminal.print(":");
BLE_Terminal.print(strPadInt(minuto, 2));
BLE_Terminal.print(":");
BLE_Terminal.println(strPadInt(segundo, 2));
strRTC = "Data: " + strPadInt(dia, 2) + "/" + strPadInt(mes, 2) + "/" +
strPadInt(ano, 2) + " - " + "Hora: " + strPadInt(hora, 2) + ":" +
strPadInt(minuto, 2) + ":" + strPadInt(segundo, 2);
/* Atualiza a hora atual */
HoraAtual = hora;
}

//----- SENSOR DE LUMINOSIDADE (RADIAÇÃO) -----
unsigned int BH1750_Read() {
    unsigned int i = 0;
    Wire.beginTransmission (BH1750_Device);
    Wire.requestFrom (BH1750_Device, 2);
    while (Wire.available())
    {
        i <= 8;
        i |= Wire.read();
    }
    Wire.endTransmission();
    return i / 1.2; // Convert to Lux
}

//-----FUNÇÃO QUE COLOCA ZEROS NA DATA E HORA-----
String strPadInt(int iNumber, int iPadding) {
    String strPaddedNumber;
    char strCommand[] = "%09d";
    char strBuffer[10];
    if (iPadding <= 9) {
        strCommand[2] = (char)(48 + iPadding);
    }
    sprintf(strBuffer, strCommand, iNumber);
    strPaddedNumber = (String)strBuffer;
    return strPaddedNumber;
}

```

```

void setup()

//----- LoRa -----
pinMode(13, OUTPUT);
Serial.begin(115200);
#ifndef VCC_ENABLE
pinMode(VCC_ENABLE, OUTPUT);
digitalWrite(VCC_ENABLE, HIGH);
delay(1000);
#endif

// LMIC init
os_init();
// Reset the MAC state. Session and pending data transfers will be
// discarded.
LMIC_reset();

// Set static session parameters. Instead of dynamically establishing a
// session by joining the network, precomputed session parameters are be
// provided.
#ifndef PROGMEM
// On AVR, these values are stored in flash and only copied to RAM once.
// Copy them to a temporary buffer here, LMIC_setSession will
// copy them into a buffer of its own again.
uint8_t appskey[sizeof(APPSKEY)];
uint8_t nwkskey[sizeof(NWKSKEY)];
memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
LMIC_setSession (0x13, DEVADDR, nwkskey, appskey);
#else
// If not running an AVR with PROGMEM, just use the arrays directly
LMIC_setSession (0x13, DEVADDR, NWKSKEY, APPSKEY);
#endif

Serial.println(F("Loading AU915/AU921 Configuration..."));
LMIC_selectSubBand(0);
LMIC_setLinkCheckMode(0);
LMIC.dn2Dr = DR_SF9;
LMIC_setDrTxpow(DR_SF7, 14);

//----- Blynk -----
BLYNK_BLE.begin(9600); /* Blynk BLE Baud Rate */
Blynk.begin(BLYNK_BLE, auth);

Serial.println("Waiting for connections...");

BLE_Terminal.clear(); /* Clear termninal contents */
BLE_Terminal.println(F("Blynk v" BLYNK_VERSION ": Device started")); /* 
Pinrt Blynk Software version */
BLE_Terminal.println(" ----- Solar Iot ----- ");
BLE_Terminal.println(" Loiola ainda é gostozaum!! S2 S2 S2 ");
BLE_Terminal.flush(); /* Ensure that all text is written in the terminal
*/
ExecutionTimerMs.setInterval(IntervaloTempoMs, ExecSolarIot); /* Set
Execution interval, in ms */

//----- SD CARD -----
InitSdCard();

```

```

//----- RTC -----
InitRtc();
//-----CONFIGURAÇÕES DOS SENsoRES-----
//----- SENSOR DE TENSÃO -----
InitTensao();
//----- SENSOR DE CORRENTE -----
InitCorrente();
//----- SENSOR DE CHUVA -----
InitChuva();
//----- SENSOR DE LUMINOSIDADE (RADIAÇÃO) -----
InitIrradiacao();
//----- SENSOR DE TEMPERATURA -----
InitTemperatura();
}

//----- -VOID LOOP() -----
void loop() {

    ExecutionTimerMs.run(); /* Start Execution timer */

} //FIM DA COMPILAÇÃO

void ExecSolarIot() {
    /* Code to be executed at each Execution timer interval */
    BLE_Terminal.println();
//-----RTC (Real Time Clock)-----
ExecRtc();
    BLE_Terminal.print("Amostra atual: ");
    BLE_Terminal.print(ContadorAmostras + 1);
    BLE_Terminal.print(" / ");
    BLE_Terminal.println(QuantidadeAmostras);

//-----CONFIGURAÇÕES DOS MÓDULOS E SENsoRES-----
//-----SENSOR DE TENSÃO-----
PLACA_TENSAO = ExecTensao();
AcumuladorTensao += PLACA_TENSAO;
//-----SENSOR DE CORRENTE-----
PLACA_CORRENTE = ExecCorrente();
AcumuladorCorrente += PLACA_CORRENTE;
//----- CÁLCULO DA POTÊNCIA-----
potenciaInstantanea = PLACA_TENSAO * PLACA_CORRENTE;
Blynk.virtualWrite(POWER_GAUGE_VPIN, String(potenciaInstantanea, 2));
/* Write text value to pin POWER_GAUGE_VPIN */
    Blynk.virtualWrite(POWER_SUPERCHART_VPIN, String(potenciaInstantanea,
2)); /* Write numeric value to SuperChart at pin POWER_SUPERCHART_VPIN */
//-----SENSOR DE CHUVA-----
PLACA_CHUVA = ExecChuva();
//----- SENSOR DE LUMINOSIDADE (RADIAÇÃO) -----
PLACA_IRRAD = ExecIrradiacao();
AcumuladorIrradiacao += PLACA_IRRAD;
//-----SENSOR DE TEMPERATURA-----
PLACA_TEMP = ExecTemperatura();
//-----SD CARD-----
ContadorAmostras++;
if (QuantidadeAmostras <= ContadorAmostras) {
    strTensao = String(AcumuladorTensao / (float)QuantidadeAmostras, 2) + "
V";
}
}

```

```

    strCorrente = String(AcumuladorCorrente / (double)QuantidadeAmostras,
2) + " A";
    strIrradiacao = String(AcumuladorIrradiacao /
(float)QuantidadeAmostras, 4) + " W/m^2";
    ExecSdCard();
    ContadorAmostras = 0;
    AcumuladorTensao = 0.0;
    AcumuladorCorrente = 0.0;
    AcumuladorIrradiacao = 0.0;
}
BLE_Terminal.println ();
BLE_Terminal.println ("-----");
//----- LoRa -----
//Variáveis multiplicadas para atender a quantidade de casas depois da
//virgula
int PLACA_TEMP_INT = PLACA_TEMP * 10;
int PLACA_IRRAD_INT = PLACA_IRRAD * 100;
int PLACA_TENSAO_INT = PLACA_TENSAO * 10;
int PLACA_CORRENTE_INT = PLACA_CORRENTE * 100;

//Conversão das variáveis para hexadecimal
String PLACA_TEMP_HEX = String(PLACA_TEMP_INT, HEX);
String PLACA_IRRAD_HEX = String(PLACA_IRRAD_INT, HEX);
String PLACA_TENSAO_HEX = String(PLACA_TENSAO_INT, HEX);
String PLACA_CORRENTE_HEX = String(PLACA_CORRENTE_INT, HEX);
String PLACA_CHUVA_HEX = String(PLACA_CHUVA, HEX);

//Verificação do tamanho da String de Temperatura para manter 3 bytes
int LENGTH_PLACA_TEMP_HEX = PLACA_TEMP_HEX.length();
switch (LENGTH_PLACA_TEMP_HEX) {
    case 1:
        PLACA_TEMP_HEX = "00" + PLACA_TEMP_HEX;
        break;
    case 2:
        PLACA_TEMP_HEX = "0" + PLACA_TEMP_HEX;
        break;
    default:
        PLACA_TEMP_HEX = PLACA_TEMP_HEX;
        break;
}

//Verificação do tamanho da String de Radiação para manter 4 bytes
int LENGTH_PLACA_IRRAD_HEX = PLACA_IRRAD_HEX.length();
switch (LENGTH_PLACA_IRRAD_HEX) {
    case 1:
        PLACA_IRRAD_HEX = "000" + PLACA_IRRAD_HEX;
        break;
    case 2:
        PLACA_IRRAD_HEX = "00" + PLACA_IRRAD_HEX;
        break;
    case 3:
        PLACA_IRRAD_HEX = "0" + PLACA_IRRAD_HEX;
        break;
    default:
        PLACA_IRRAD_HEX = PLACA_IRRAD_HEX;
        break;
}

//Verificação do tamanho da String de Tensão para manter 3 bytes

```

```

int LENGTH_PLACA_TENSAO_HEX = PLACA_TENSAO_HEX.length();
switch (LENGTH_PLACA_TENSAO_HEX) {
    case 1:
        PLACA_TENSAO_HEX = "00" + PLACA_TENSAO_HEX;
        break;
    case 2:
        PLACA_TENSAO_HEX = "0" + PLACA_TENSAO_HEX;
        break;
    default:
        PLACA_TENSAO_HEX = PLACA_TENSAO_HEX;
        break;
}

//Verificação do tamanho da String de Corrente para manter 3 bytes
int LENGTH_PLACA_CORRENTE_HEX = PLACA_CORRENTE_HEX.length();
switch (LENGTH_PLACA_CORRENTE_HEX) {
    case 1:
        PLACA_CORRENTE_HEX = "00" + PLACA_CORRENTE_HEX;
        break;
    case 2:
        PLACA_CORRENTE_HEX = "0" + PLACA_CORRENTE_HEX;
        break;
    default:
        PLACA_CORRENTE_HEX = PLACA_CORRENTE_HEX;
        break;
}
//Montagem da variável de toda a informação a ser enviada
myINF = PLACA_TEMP_HEX + PLACA_IRRAD_HEX + PLACA_TENSAO_HEX +
PLACA_CORRENTE_HEX + PLACA_CHUVA_HEX;
// Start job
do_send(&sendjob);
unsigned long now;
now = millis();
if ((now & 512) != 0) {
    digitalWrite(13, HIGH);
}
else {
    digitalWrite(13, LOW);
}
os_runloop_once();
BLE_Terminal.flush(); /*Ensure that all text is written in the terminal*/
}

void InitSdCard(void) {
    BLE_Terminal.print("Initializing SD card...");
    if (!SdFatDev.begin(SD_CHIP_SELECT_PIN)) {
        BLE_Terminal.println("initialization failed!");
        return;
    }
    BLE_Terminal.println("initialization done.");
    GravarSD();
    // open the file. note that only one file can be open at a time,
    // so you have to close this one before opening another.
    // myFile = SdFatDev.open("test.txt", FILE_WRITE);
    // if the file opened okay, write to it:
    // if (myFile) {
    //     BLE_Terminal.print("Writing to test.txt...");
    //     myFile.println("testing 1, 2, 3.");
    //     // close the file:
    //     myFile.close();
    //     BLE_Terminal.println("done.");
}

```

```

// } else {
//   // if the file didn't open, print an error:
//   BLE_Terminal.println("error opening test.txt");
// }
// re-open the file for reading:
myFile = SdFatDev.open("test.txt");
if (myFile) {
  BLE_Terminal.println("test.txt:");
  // read from the file until there's nothing else in it:
  while (myFile.available()) {
    Serial.write(myFile.read());
  }
  // close the file:
  myFile.close();
} else {
  // if the file didn't open, print an error:
  BLE_Terminal.println("error opening test.txt");
}
}

void InitRtc(void) {
  Wire.begin();
  // setDateTime();           //Necessário configurar na função
"setDateTime()" CONFIGURADA AO FIM DO PROGRAMA
}
void InitTensao(void) {
  pinMode(A1, INPUT);
}
void InitCorrente(void) {
  pinMode(A2, INPUT);
}
void InitChuva(void) {
  pinMode(A0, INPUT);
}
void InitIrradiacao(void) {
  Wire.begin();
  Wire.beginTransmission (BH1750_Device);
  Wire.write(0x10); // Set resolution to 1 Lux
  Wire.endTransmission ();
  delay(1000); }
void InitTemperatura(void) {
  Serial.begin(9600);
  barramento.begin();
  barramento.getAddress(sensor, 0);
}

void ExecSdCard(void) {
  GravarSD();
}
void ExecRtc(void) {
  printDate();
}
float ExecTensao(void) {
  leituraSensor = analogRead(A1);
  tensaoEntrada = (leituraSensor * 5.0) / 1024.0;
  tensaoMedida = tensaoEntrada / (valorR2 / (valorR1 + valorR2)) - offset;
//VARIÁVEL RECEBE O VALOR DE TENSÃO DC MEDIDA PELO SENSOR
  if (tensaoMedida < 0) {
    tensaoMedida = 0;
  }
  BLE_Terminal.print("Tensão: ");
  BLE_Terminal.print(tensaoMedida, 2);
}

```

```

BLE_Terminal.println(" V");
strTensao = String(tensaoMedida, 2) + " V";
Blynk.virtualWrite(VOLTAGE_HLEVEL_VPIN, String(tensaoMedida, 2)); /*
Write text value to pin VOLTAGE_HLEVEL_VPIN, limited to 2 decimal places */
    return tensaoMedida;
}
double ExecCorrente(void) {
    for (int i = 0; i < qtde; i++) {
        {
            vetCorrente[i] = analogRead(A2);
            delayMicroseconds(1000);
        }
    double valor_Corrente = 0;
    for (int i = 0; i < qtde; i++) {
        valor_Corrente = valor_Corrente + vetCorrente[i];
    }
    valor_Corrente = valor_Corrente / qtde;
    valor_Corrente = valor_Corrente * 0.0048828125;
    valor_Corrente = valor_Corrente - 2.51; //Offset
    valor_Corrente = valor_Corrente * 1000;
    valor_Corrente = valor_Corrente / 134;           //sensibilidade ajustada
com o multímetro
    valor_Corrente = abs(valor_Corrente);
    BLE_Terminal.print("Corrente = ");
    BLE_Terminal.print(valor_Corrente, 2);
    BLE_Terminal.println(" A");
    strCorrente = String(valor_Corrente, 2) + " A";
    Blynk.virtualWrite(CURRENT_HLEVEL_VPIN, String(valor_Corrente, 2)); /*
Write text value to pin CURRENT_HLEVEL_VPIN, limited to 2 decimal places */
    return valor_Corrente;
}
int ExecChuva(void) {
    int sensorValue = analogRead(A0);
    // BLE_Terminal.print(sensorValue);
    if (sensorValue < thresholdValue) {
        BLE_Terminal.println("Tempo Fechado");
        strChuva = "Tempo Fechado";
        Blynk.virtualWrite(RAIN_IMGG_VPIN, 2); /* Send image ID to pin
RAIN_IMGG_VPIN, image indexing starts from 1 */
        Blynk.setProperty(RAIN_IMGG_VPIN, "label", "Tempo Fechado"); /* Set
image label */
        return 1;
    }
    else {
        BLE_Terminal.println("Tempo Aberto");
        strChuva = "Tempo Aberto";
        Blynk.virtualWrite(RAIN_IMGG_VPIN, 1); /* Send image ID to pin
RAIN_IMGG_VPIN, image indexing starts from 1 */
        Blynk.setProperty(RAIN_IMGG_VPIN, "label", "Tempo Aberto"); /* Set
image label */
        return 0;
    }
}
float ExecIrradiacao(void) {
    Lux = BH1750_Read();
    Wattsm2 = Lux / 126.7;
    Wattsm2 = abs(Wattsm2);
    BLE_Terminal.print("Incidencia solar: ");
    BLE_Terminal.print (Wattsm2, 4);
    BLE_Terminal.println(" W/m^2");
    strIrradiacao = String(Wattsm2, 4) + " W/m^2";
}

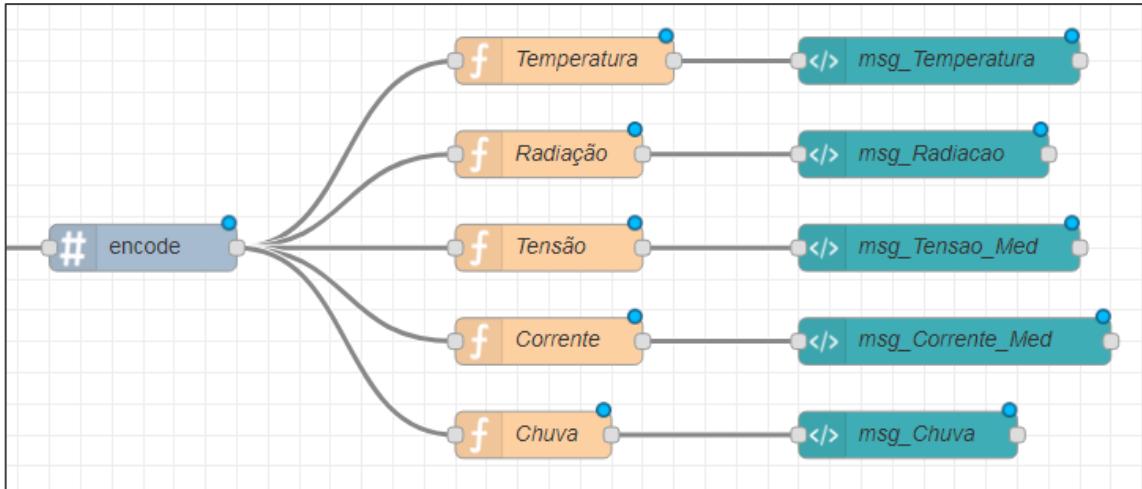
```

```
Blynk.virtualWrite(IRRADIATION_LVALUE_VPIN, String(Wattsm2, 4)); /* Write
text value to pin IRRADIATION_LVALUE_VPIN */
    return Wattsm2;
}
float ExecTemperatura(void) {
    barramento.requestTemperatures();
    float temperatura = barramento.getTempC(sensor);
    BLE_Terminal.print("Temperatura: ");
    BLE_Terminal.print(temperatura, 1);
    BLE_Terminal.println(" °C");
    strTemperatura = String(temperatura, 1) + " °C";
    Blynk.virtualWrite(TEMPERATURE_LVALUE_VPIN, String(temperatura, 1)); /* Write
text value to pin TEMPERATURE_LVALUE_VPIN */
    return temperatura;
}
```

APÊNDICE D – CÓDIGO FONTE DOS BLOCOS NO NODE-RED

Este apêndice tem por finalidade apresentar o código fonte inserido nos blocos de interpretação da mensagem, análise e exibição dos dados.

Figura D.1 – Bloco de interpretação da mensagem e exibição dos dados



Código do nó “Temperatura”

```

var buf = new Buffer(msg.payload.toString('hex'), 'hex');
msg.payload = buf.toString();
msg.payload = ((parseInt(msg.payload.substr(0,3),16)/10)).toFixed(1);
return msg;
  
```

Código do nó “Radiação”

```

var buf = new Buffer(msg.payload.toString('hex'), 'hex');
msg.payload = buf.toString();
msg.payload = ((parseInt(msg.payload.substr(3,4),16)/100)).toFixed(2);
return msg;
  
```

Código do nó “Tensão”

```

var buf = new Buffer(msg.payload.toString('hex'), 'hex');
msg.payload = buf.toString();
msg.payload = ((parseInt(msg.payload.substr(7,3),16)/10)).toFixed(1);
return msg;
  
```

Código do nó “Corrente”

```

var buf = new Buffer(msg.payload.toString('hex'), 'hex');
msg.payload = buf.toString();
msg.payload = ((parseInt(msg.payload.substr(10,3),16)/100)).toFixed(2);
return msg;
  
```

Código do nó “Chuva”

```
// Lê o último byte da mensagem
//
//Se for "1" envia o endereço de uma imagem tipo PNG que representa chuva.
//
//Caso contrário, se o valor for "0", é enviado o endereço de uma imagem
//que representa um dia ensolarado.
//
var buf = new Buffer(msg.payload.toString('hex'), 'hex');
msg.payload = buf.toString();
var str_chuva = ((parseInt(msg.payload.substr(13,1),16))).toFixed(0);
msg.payload = "http://localhost:1880/medidas_icon/s_chuva.png";
if (str_chuva === "1"){
    msg.payload = "http://localhost:1880/medidas_icon/c_chuva.png";
}
return msg;
```

Código do nó “msg_Temperatura”

```
<!DOCTYPE html>
<html>
<body>
<center ng-bind-html="msg.payload" style="font-size:150%"></center>
<center style="font-size:70%">°C</center>
</body>
</html>
```

Código do nó “msg_Radiacao”

```
<!DOCTYPE html>
<html>
<body>
<center ng-bind-html="msg.payload" style="font-size:150%"></center>
<center style="font-size:70%">W/m2</center>
</body>
</html>
```

Código do nó “msg_Tensao”

```
<html>
<body>
<center ng-bind-html="msg.payload" style="font-size:150%"></center>
<center style="font-size:70%">V</center>
</body>
</html>
```

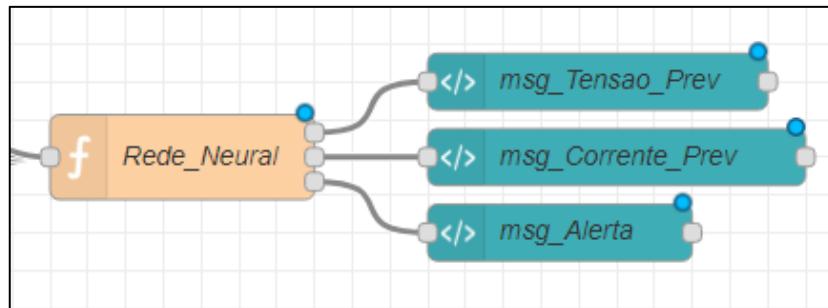
Código do nó “msg_Corrente”

```
<html>
<body>
<center ng-bind-html="msg.payload" style="font-size:150%"></center>
<center style="font-size:70%">A</center>
</body>
</html>
```

Código do nó “msg_Chuva”

```
<div height="50" style="height: 50px;">
<img src={ {msg.payload} } width="50" align="right"><br/>
</div>
```

Figura D.2 – Bloco de análise e exibição dos dados



Código do nó “Rede_Neural”

```
var v_temperatura = global.get("val_temperatura") || 0;
var v_radiacao = global.get("val_radiacao") || 0;
var v_tensao = global.get("val_tensao") || 0;
var v_corrente = global.get("val_corrente") || 0;

/* Normalização de entradas: */
const c_norm_in1 = 100;
const c_norm_in2 = 30;

/* Pré-processamento de entradas: */
const c_prepoc_in1_xmin = -0.9999280000000000;
const c_prepoc_in1_xmax = -0.9634300000000000;
const c_prepoc_in1_ymin = -1;
const c_prepoc_in1_ymax = 1;
const c_prepoc_in2_xmin = -0.41666666666667;
const c_prepoc_in2_xmax = -0.1400000000000000;
const c_prepoc_in2_ymin = -1;
const c_prepoc_in2_ymax = 1;

/* Bias da camada oculta: */
const c_bias_l1n1 = 1.744592669413621;
const c_bias_l1n2 = 2.161137518271462;
const c_bias_l1n3 = -1.873491712110173;
const c_bias_l1n4 = -4.464802813400797;

/* Pesos da camada oculta: */
const c_weight_in1_l1n1 = -3.320153717649573;
const c_weight_in1_l1n2 = -0.698082194798146;
const c_weight_in1_l1n3 = -0.779573714526757;
const c_weight_in1_l1n4 = -4.303239532683741;
const c_weight_in2_l1n1 = -1.744333905492800;
const c_weight_in2_l1n2 = 1.972406637573078;
const c_weight_in2_l1n3 = 1.509926543145421;
const c_weight_in2_l1n4 = -0.817957760297973;

/* Bias da camada de saída: */
const c_bias_out1 = -1.845291841818123;
```

```

/* Pesos da camada de saída: */
const c_weight_l1n1_out1 = -0.135903169939856;
const c_weight_l1n2_out1 = -0.628233562311729;
const c_weight_l1n3_out1 = -0.420651120867317;
const c_weight_l1n4_out1 = -2.820103596460168;

/* Pós-processamento de saída: */
const c_postproc_out1_xmin = 1.590000000000000;
const c_postproc_out1_xmax = 17.640000000000001;
const c_postproc_out1_ymin = -1;
const c_postproc_out1_ymax = 1;

/* Normalização de entradas: */
var v_in1 = (v_radiacao/c_norm_in1) - 1;
var v_in2 = (v_temperatura/c_norm_in2) - 1;

/* Pré-processamento de entradas: */
var v_in1_pp = ((c_prepoc_in1_ymax - c_prepoc_in1_ymin) * (v_in1 - c_prepoc_in1_xmin)) / (c_prepoc_in1_xmax - c_prepoc_in1_xmin)) + c_prepoc_in1_ymin;
var v_in2_pp = ((c_prepoc_in2_ymax - c_prepoc_in2_ymin) * (v_in2 - c_prepoc_in2_xmin)) / (c_prepoc_in2_xmax - c_prepoc_in2_xmin)) + c_prepoc_in2_ymin;

/* Cálculo da camada oculta: */
var v_l1n1 = (2 / (1 + Math.exp(-2 * (c_bias_l1n1 + (v_in1_pp * c_weight_in1_l1n1) + (v_in2_pp * c_weight_in2_l1n1)))))) - 1;
var v_l1n2 = (2 / (1 + Math.exp(-2 * (c_bias_l1n2 + (v_in1_pp * c_weight_in1_l1n2) + (v_in2_pp * c_weight_in2_l1n2)))))) - 1;
var v_l1n3 = (2 / (1 + Math.exp(-2 * (c_bias_l1n3 + (v_in1_pp * c_weight_in1_l1n3) + (v_in2_pp * c_weight_in2_l1n3)))))) - 1;
var v_l1n4 = (2 / (1 + Math.exp(-2 * (c_bias_l1n4 + (v_in1_pp * c_weight_in1_l1n4) + (v_in2_pp * c_weight_in2_l1n4)))))) - 1;

/* Cálculo da camada de saída: */
var v_out1 = c_bias_out1 + (v_l1n1 * c_weight_l1n1_out1) + (v_l1n2 * c_weight_l1n2_out1) + (v_l1n3 * c_weight_l1n3_out1) + (v_l1n4 * c_weight_l1n4_out1);

/* Pós-processamento de saída: */
var v_out1_pp = ((v_out1 - c_postproc_out1_ymin) * (c_postproc_out1_xmax - c_postproc_out1_xmin)) / (c_postproc_out1_ymax - c_postproc_out1_ymin)) + c_postproc_out1_xmin;

var str_c_out1_pp = { payload: 0.05 };
var str_v_out1_pp = { payload: v_out1_pp.toFixed(2) };

//Se o valor estiver fora do limite de 10% de tolerancia,
//exibe uma imagem de alerta.
var str_alert = { payload:"http://localhost:1880/medidas_icon/c_alerta.png" };

//Se o valor estiver dentro do limite de 10% de tolerancia,
//NÃO exibe uma imagem de alerta (imagem neutra).
if ((v_tensao < (v_out1_pp*(1.1))) && (v_tensao > (v_out1_pp*0.9))) {
    str_alert = { payload:"http://localhost:1880/medidas_icon/s_alerta.png" };
}

return [[str_v_out1_pp], [str_c_out1_pp], [str_alert]];

```

Código do nó “msg_Tensao_Prev”

```
<html>
<body>
<center ng-bind-html="msg.payload" style="font-size:150%"></center>
<center style="font-size:70%">V</center>
</body>
</html>
```

Código do nó “msg_Corrente_Prev”

```
<html>
<body>
<center ng-bind-html="msg.payload" style="font-size:150%"></center>
<center style="font-size:70%">A</center>
</body>
</html>
```

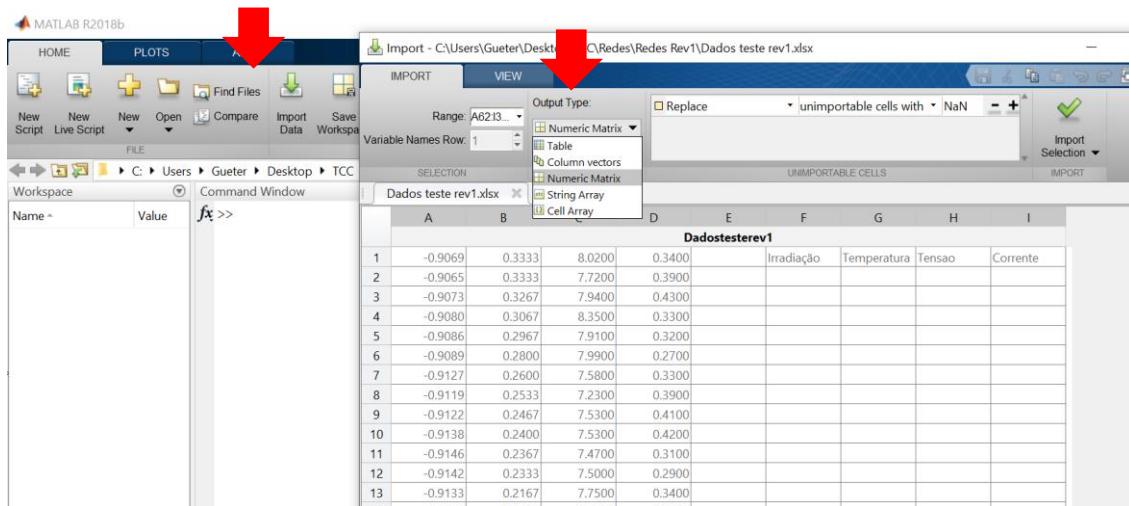
Código do nó “msg_Alerta”

```
<div height="50" style="height: 180px;">
<br/>
</div>
```

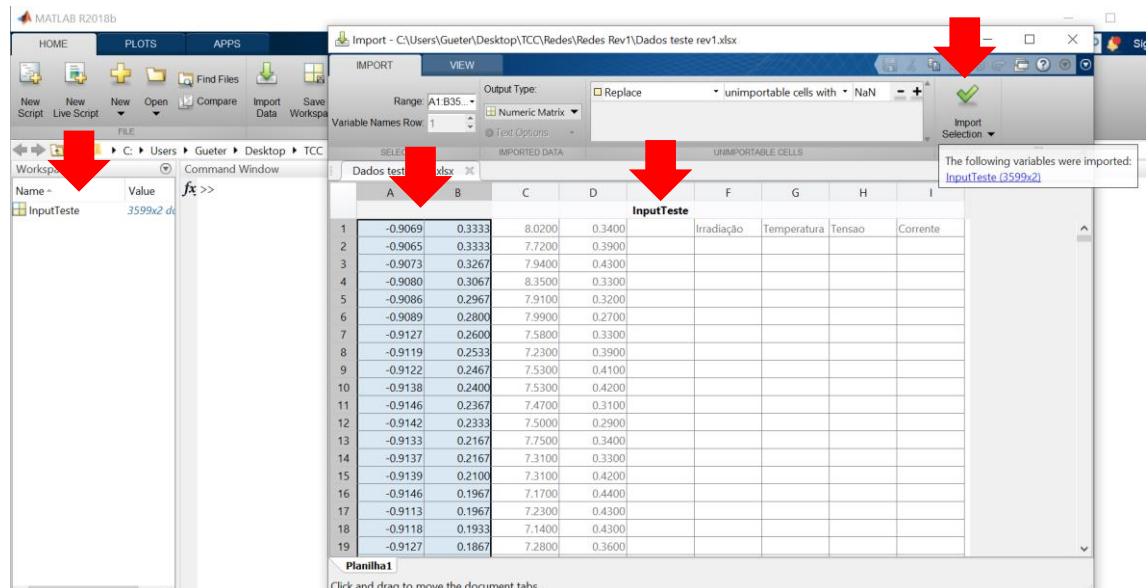

APÊNDICE E – CRIAÇÃO E SIMULAÇÃO DE UMA REDE NEURAL ARTIFICIAL NO MATLAB

Este apêndice tem por finalidade apresentar o passo-a-passo de como criar um Rede Neural Artificial com a ferramenta *nntool* do *Matlab*. Além disso, o guia ensina como treinar, testar e simular uma rede neural artificial no *Simulink*.

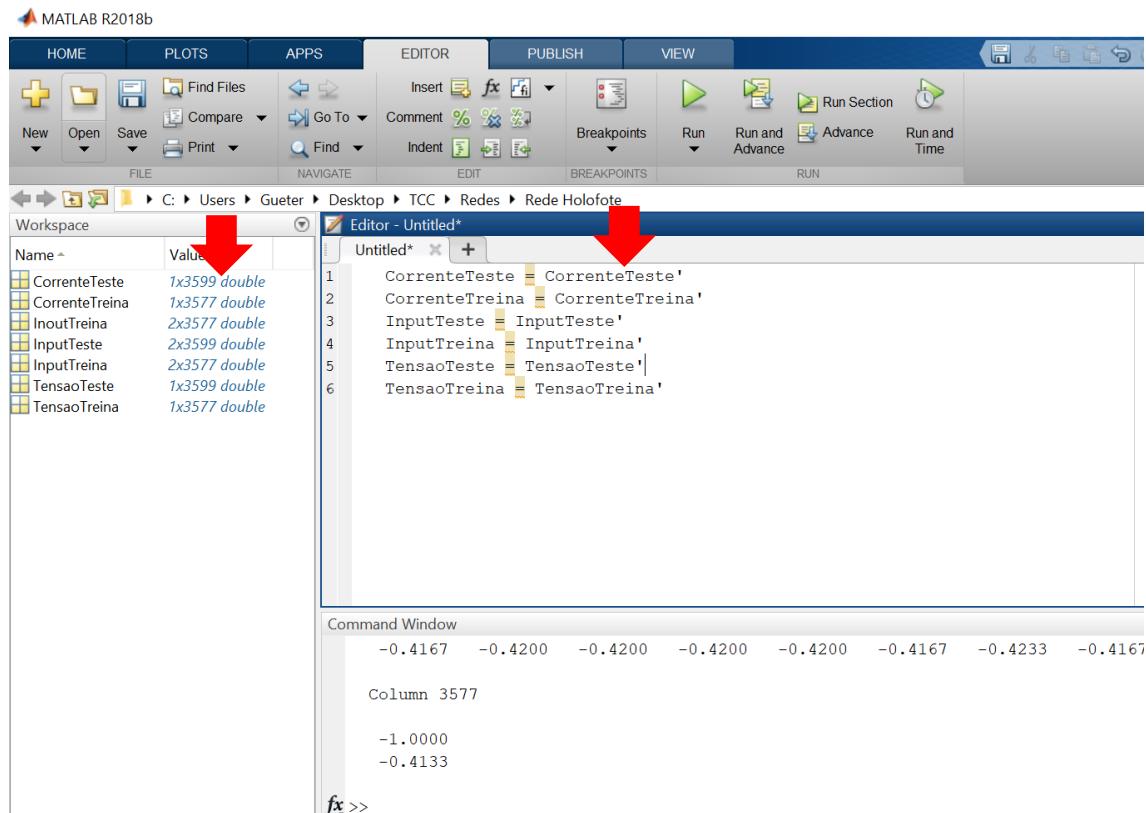
- O primeiro passo é importar os dados. No caso de os dados estarem em uma planilha do *Excel*, basta clicar no ícone **Import Data** e selecionar o arquivo com os dados desejados. Após selecionar o arquivo, mude o **Output Type** para **Numeric Matrix**.



- Selecionar os dados desejados, alterar o nome da matriz de saída e clicar em **Import Selection**. Note que ao fazer isso será criada uma matriz com as dimensões dos dados selecionados e o nome dado.

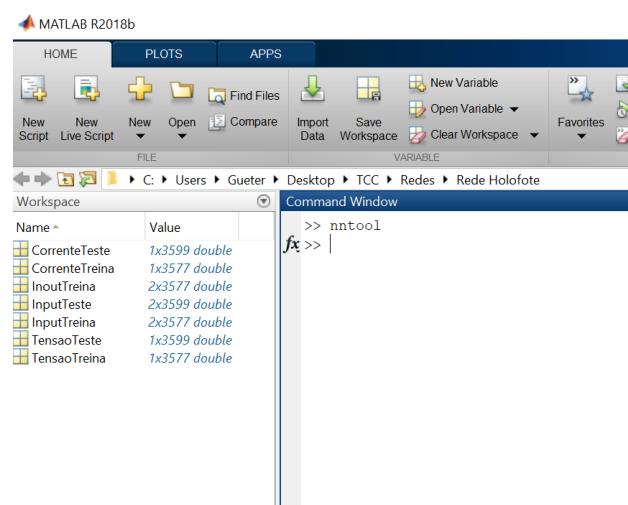


3. Os procedimentos **1** e **2** devem ser realizados para **todas as variáveis a serem importadas**, sendo elas, no caso desse trabalho, variáveis de entrada (temperatura e radiação solar) ou de saída (tensão e corrente) da rede neural, e variáveis de treinamento e de teste.
4. Uma vez com todas as variáveis importadas, deve-se transportar todas as matrizes, para torná-las compatíveis com a forma que a ferramenta ***nntool*** trabalha.

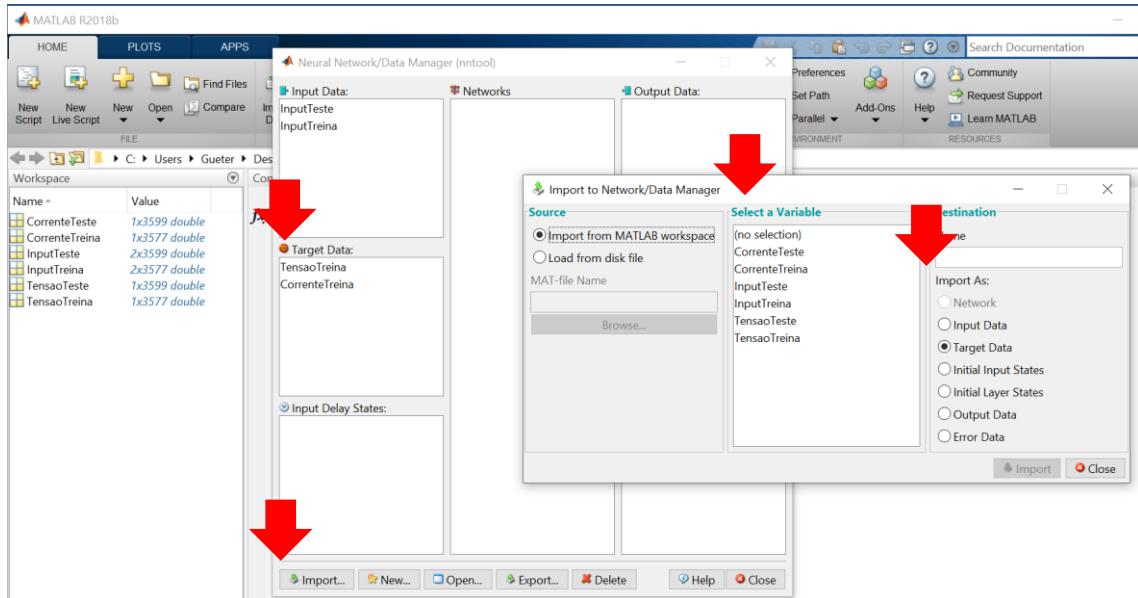


Matrizes já transpostas

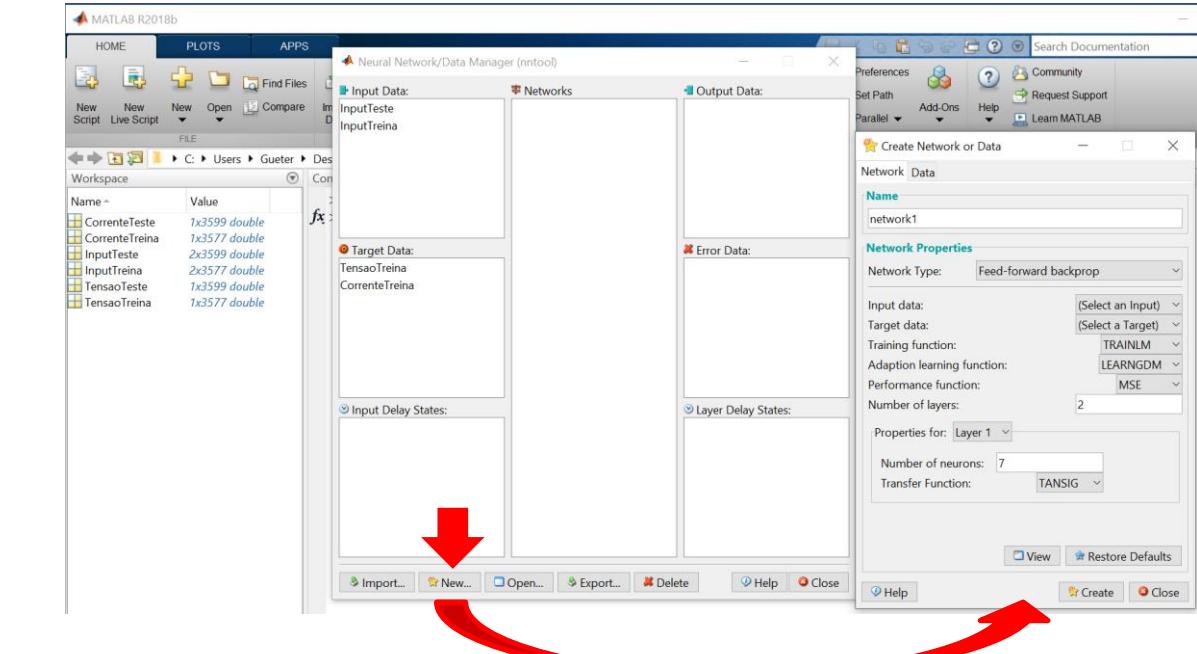
5. Agora com todos os dados importados, deve-se abrir o ***nntool*** digitando o comando ***nntool*** no **Matlab**



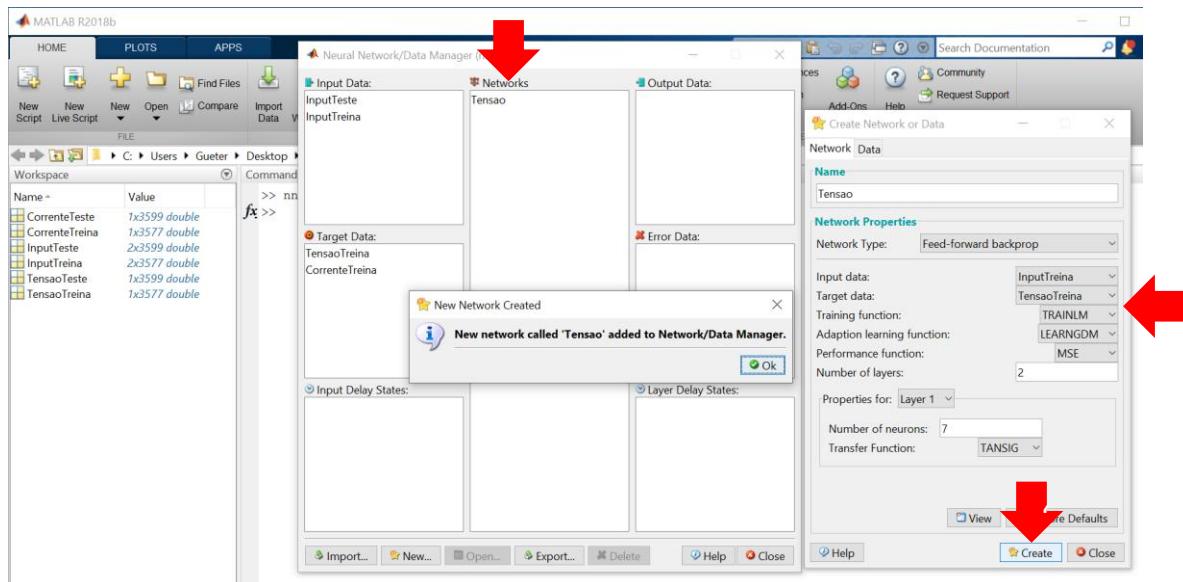
6. Uma vez aberto o **nntool**, clicar em **Import** para importar os dados do **Matlab** para o **nntool**. Os dados de entrada devem ser importados como **Input Data**. Já os dados de saída devem ser importados como **Target Data**.



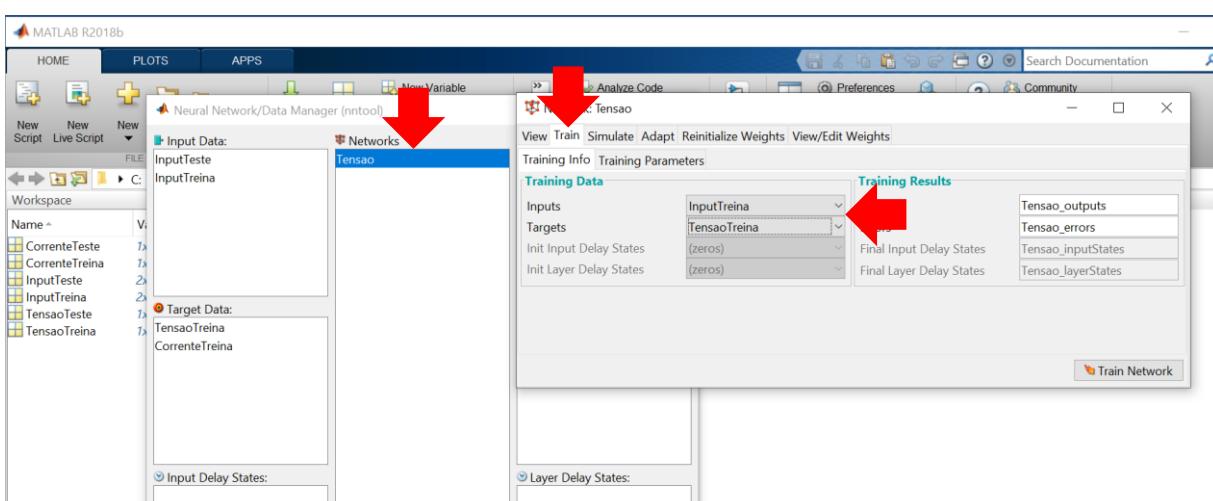
7. Com os dados importados no **nntool**, o próximo passo é criar a Rede Neural Artificial. Para isso deve-se clicar em **New** e abrirá a janela referente à criação da rede.



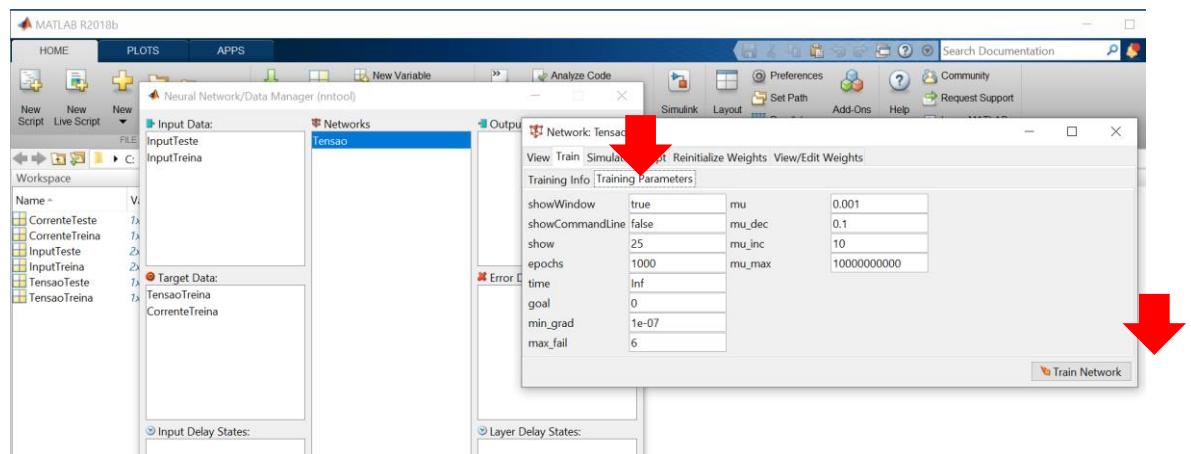
8. Na janela de criação da rede neural, deve-se configurar as características da rede, como o tipo de aprendizagem dela, o seu algoritmo de aprendizagem, o número de camadas com o número de neurônios e o método de critério de performance. Após configurar e nomear a rede, basta clicar no botão **Create** que a rede estará criada. Vale ressaltar que nos campos **Input Data** e **Target Data** deve-se colocar os dados de treinamento da rede.



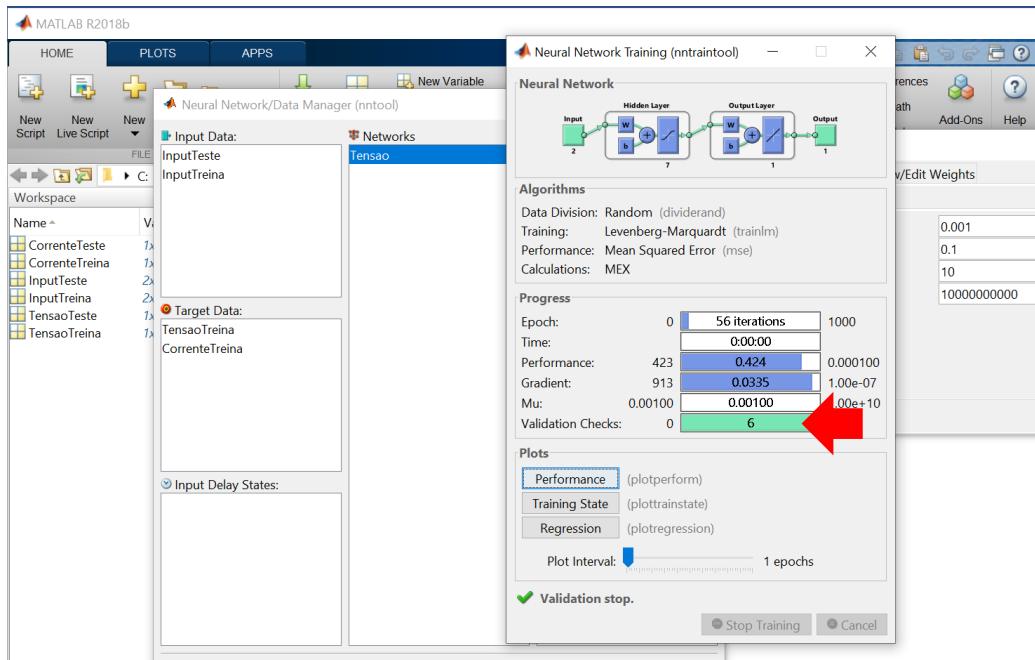
9. O próximo passo é treinar a rede. Para isso basta **clicar duas vezes** na rede neural, e irá abrir a janela da rede. Indo na aba **Train** define-se os **Inputs** e **Outputs** de treinamento.



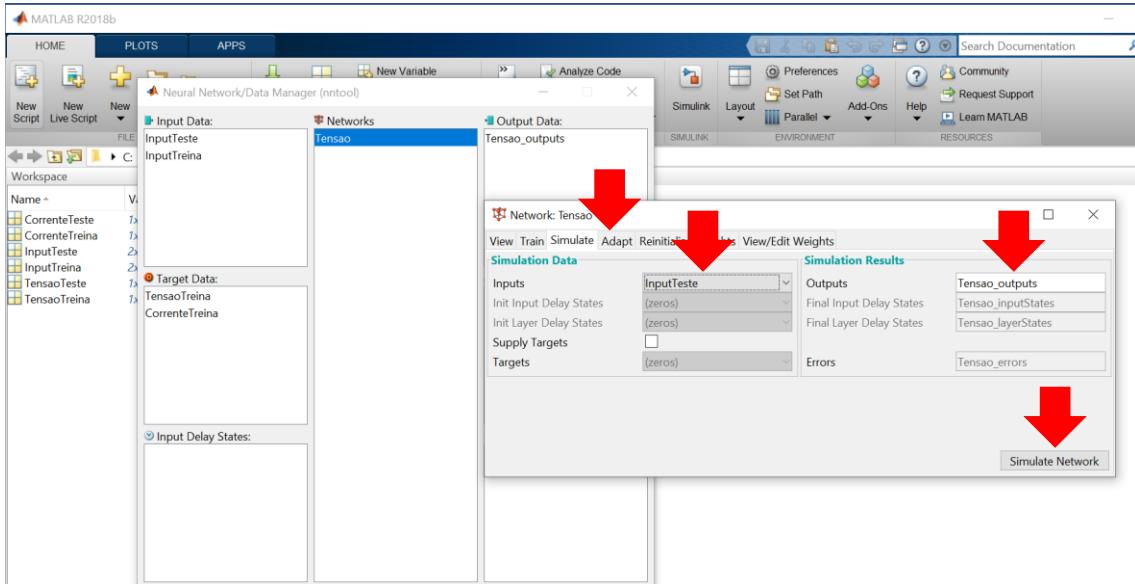
10. Antes de realizar o treinamento é preciso definir os parâmetros de treinamento. Basta clicar em **Training Parameters** e definir os valores desejados para os critérios de parada, como número de iterações (*epochs*) e a meta de performance (*goal*). Uma vez definido, basta clicar em **Train Network** e começará a aprendizagem da rede neural.



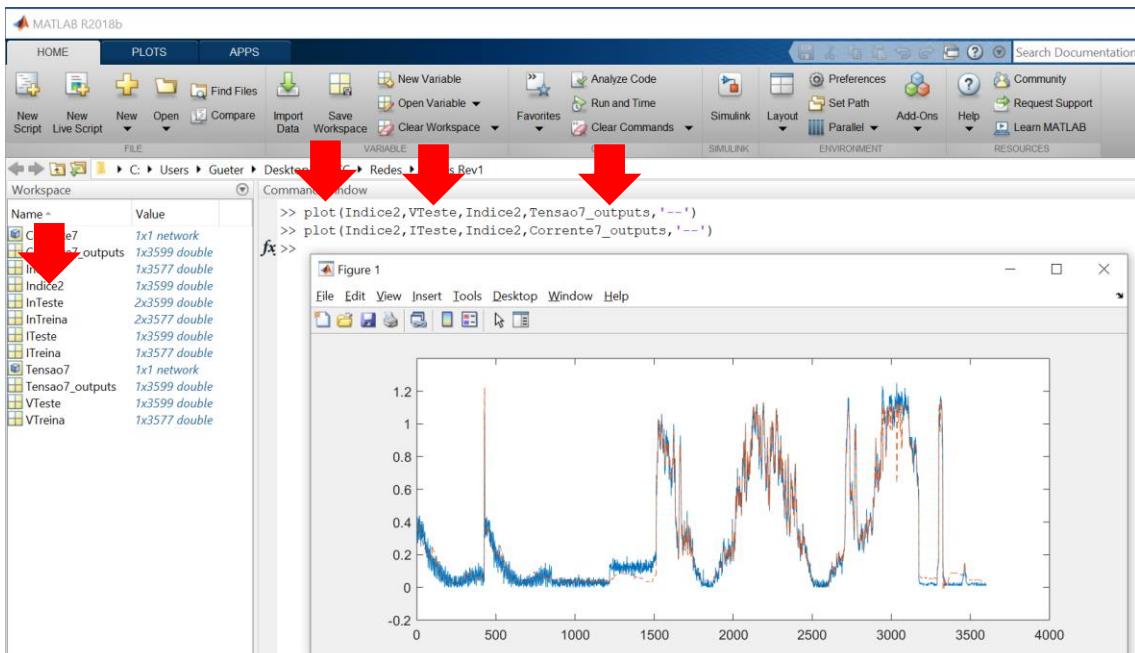
11. Após esperar o treinamento ser realizado e algum critério de parada ser atingido, deve-se fechar a tela de treinamento e fazer a simulação da rede.



12. Para testar a rede, deve-se voltar na janela da rede neural, clicar em **Simulate** e definir os **Inputs** de simulação, os quais são os dados de teste importados anteriormente. Também deve-se nomear os **Outputs** da rede, que serão comparados com os valores reais. Para simular deve-se clicar no botão **Simulate Network**.

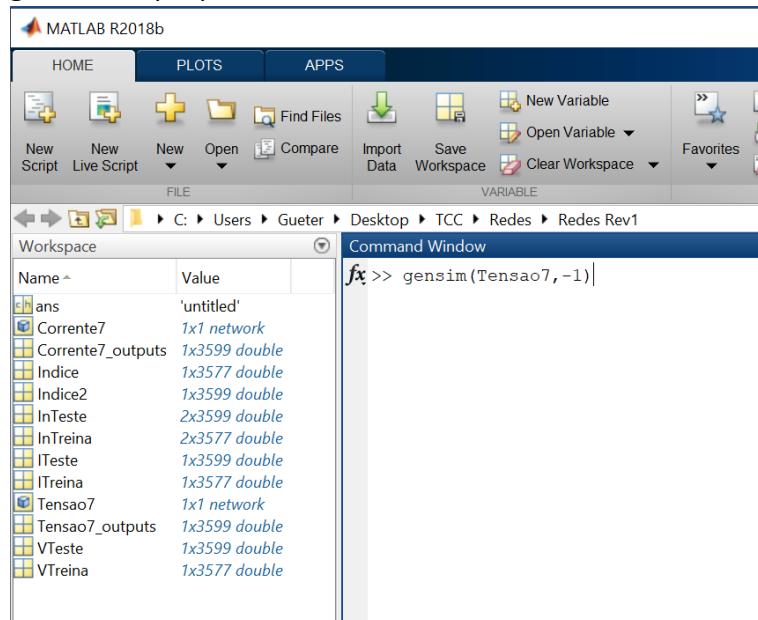


13. Após simular ambas as redes de corrente e de tensão com os dados de teste e exportar os **Outputs** gerados pela rede do *nntool* para o *Matlab* (mesmo procedimento de importar do passo 6), pode-se plotar os valores previstos e medidos para comparar e checar a qualidade da rede neural artificial. Para isso usa-se o comando *plot*. Vale ressaltar que é necessário criar um matriz índice do tamanho das amostras para ser o eixo X do plot.

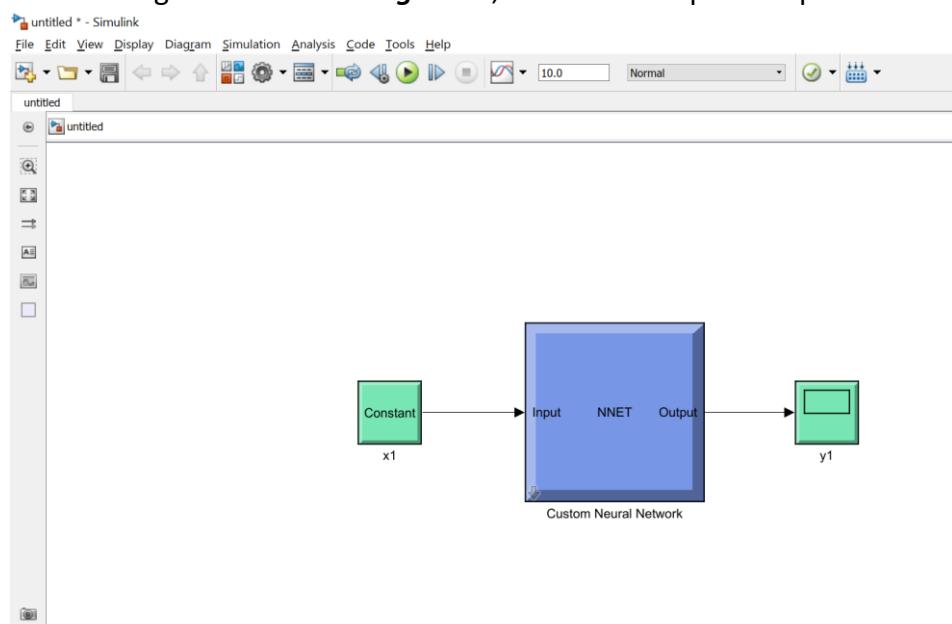


14. O processo deve ser repetido até encontrar uma configuração de rede neural artificial que após treinada satisfaça os objetivos desejados. Uma vez que isso aconteça deve-se exportar os *pesos* e os *bias* para poder implementar a rede neural onde desejar. É possível consultar o Apêndice F – Implementação manual de uma Rede Neural do tipo Perceptron Multicamadas (MLP) gerada em *MATLAB* para ter informações detalhadas de como exportar e depois implementar os parâmetros de uma rede neural artificial do *Matlab*.

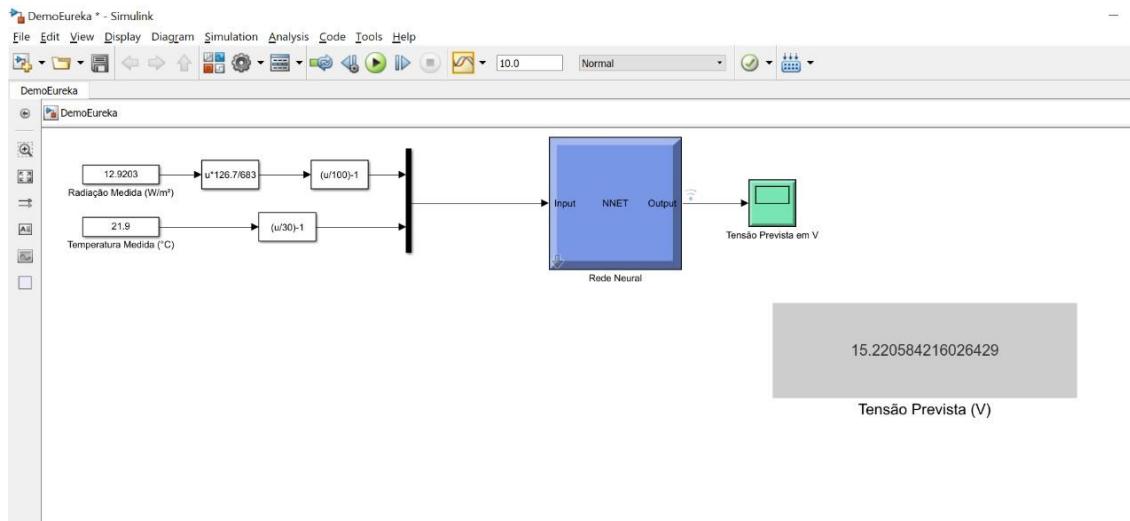
15. Caso deseje exportar a rede para o ***Simulink*** no *Matlab*, deve-se digitar o comando ***gensim*** no próprio *Matlab*.



16. Uma vez digitado o comando ***gensim***, a rede será exportada para o ***Simulink***.



17. Uma vez no Simulink, podem efetuados diversas modificações no *software*, a fim de facilitara a aplicação da rede neural. Veja abaixo um exemplo de onde colocando o valor de temperatura ($^{\circ}\text{C}$) e radiação solar (W/m^2), esses valores são normalizados, entram como *inputs* da rede, e é exibido o valor de *output*.



APÊNDICE F – IMPLEMENTAÇÃO MANUAL DE UMA REDE NEURAL DO TIPO PERCEPTRON MULTICAMADAS (MLP) GERADA EM MATLAB

Este apêndice foi elaborado em conjunto com o Engenheiro Rodrigo de Marca França e tem como objetivo Exportar uma Rede Neural do tipo Perceptron Multicamadas (MLP) criada e treinada na ferramenta MATLAB para qualquer outra aplicação que permita o a entrada de equações matemáticas.

A ferramenta MATLAB possui um ambiente bem robusto que permite a concepção, treinamento e testes de Redes Neurais MLP com baixo esforço e alta eficiência. No entanto, as redes geradas nele tem aplicação limitada em ambientes e ferramentas externas. Para esses ambientes, é possível montar manualmente as equações que geram a Rede Neural, usando os parâmetros gerados na ferramenta MATLAB. A única restrição é que o ambiente deve ser capaz de realizar equações com números flutuantes (preferencialmente com precisão “double”) e funções matemáticas (sendo a função exponencial uma das mais comuns). Exemplos de ambiente onde seria possível equacionar uma rede desse tipo são:

- Microcontroladores;
- Softwares de alto nível;
- Aplicações em Nuvem;
- Aplicativos móveis;
- Entre outros.

Será realizada uma demonstração rápida do equacionamento de uma Rede Neural MLP, seguido de um exemplo de aplicação onde uma rede desse tipo é utilizada para prever a tensão gerada em um painel solar, baseada na temperatura e radiação solar aos quais ela está exposta. Essa rede foi desenvolvida e aplicada por alunos para o trabalho de graduação “IoT aplicada à energia renovável”.

1. Demonstração genérica do cálculo de uma Rede Neural MLP:

Uma Rede Neural do tipo Perceptron Multicamadas (MLP) é composta, basicamente, por:

- Uma camada de entrada, onde a quantidade de neurônios é igual a quantidade de entradas da rede, totalmente conectada (usualmente não é contabilizada na quantidade de camadas da rede);

- N camadas ocultas, onde o número de neurônios pode variar por camada, totalmente conectada;
- Uma camada de saída, onde a quantidade de neurônios é igual a quantidade de saídas da rede.

Desconsiderando os neurônios de entrada, um neurônio genérico dessa rede pode ser descrito de acordo com a Figura F.1, a seguir.

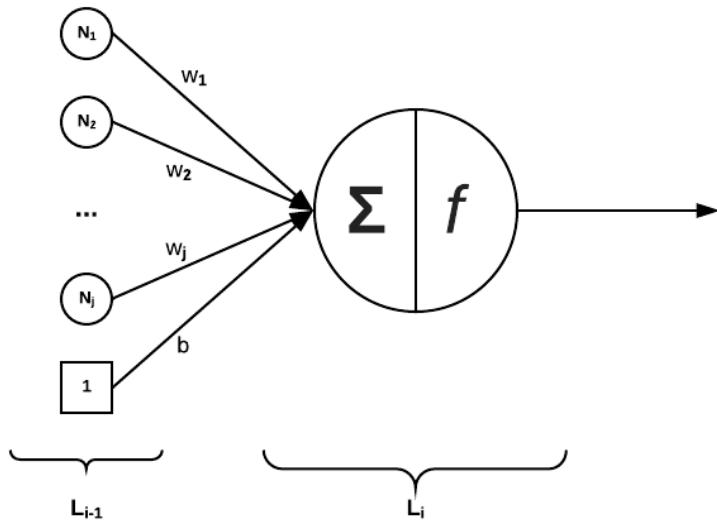


Figura F.1 - Neurônio genérico de uma MLP

A operação executada por um neurônio pode ser descrita de maneira simplificada pela execução de sua função de ativação usando como parâmetro a somatória de todas as suas entradas. Ou seja, $N(x) = f(\sum \text{entradas})$. Onde $f(n)$ é a função de ativação do neurônio, que depende da camada onde ele se encontra. Em uma rede MLP, os resultados dos neurônios de uma camada são utilizados como entrada para a próxima camada, até a camada de saída (última camada).

Usualmente, as funções de ativação $f(n)$ de redes MLP operam no intervalo de valores de -1 a $+1$. Dessa forma, é conveniente que todas as operações da rede estejam confinadas nesse mesmo intervalo. Como normalmente deseja-se que uma rede neural MLP trabalhe com faixas genéricas de valores, torna-se interessante realizar um pré-processamento de

suas entradas, assim como um pós-processamento de suas saídas. Uma maneira simples de realizar essas operações é com uma normalização das entradas e uma normalização inversa das saídas. Podemos observar a aplicação dessas funções na Figura F.2, abaixo:

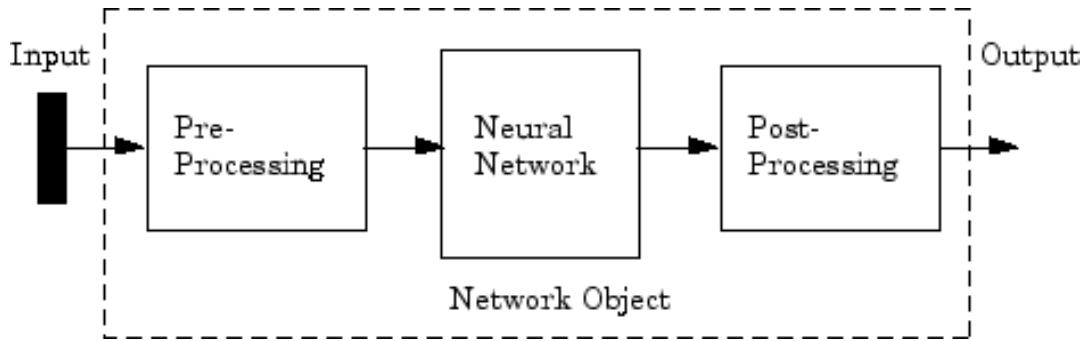


Figura F.2 - Exemplo de funcionamento do Pré e Pós processamento em uma Rede Neural MLP

Logo, a saída de qualquer Rede Neural MLP, com l camadas, pode ser calculado seguindo os seguintes $l + 2$ passos:

- Passo 1: Cálculo de \vec{I}_p , usando \vec{I} ;
- Passo 2: Cálculo de \vec{N}_1 , usando W_1 , \vec{I}_p e \vec{B}_1 ;
- Passo 3 Cálculo de \vec{N}_2 , usando W_2 , \vec{N}_1 e \vec{B}_2 ;
- ...
- Passo l : Cálculo de \vec{N}_{l-1} , usando W_{l-1} , \vec{N}_{l-2} e \vec{B}_{l-1} ;
- Passo $l + 1$: Cálculo de $\vec{O}_p = \vec{N}_l$, usando W_l , \vec{N}_{l-1} e \vec{B}_l ;
- Passo $l + 2$: Cálculo de \vec{O} , usando \vec{O}_p ;

Onde podemos definir:

$$\vec{I}_p = f_{\text{pré-processamento}}(\vec{I})$$

Sendo \vec{I}_p o vetor de entradas pré-processadas e \vec{I} o vetor de entradas da rede.

$$\vec{N}_1 = f_1(W_1 * \vec{I}_p + \vec{B}_1)$$

Sendo \vec{N}_1 o vetor resultado dos neurônios da primeira camada, $f_1(n)$ a função de ativação da primeira camada, W_1 a matriz de pesos da primeira camada, \vec{I}_p o vetor de entradas pré-processadas e \vec{B}_1 o vetor de fatores de viés (*bias*) da primeira camada.

$$\vec{N}_l = f_i(W_i * \vec{N}_{l-1} + \vec{B}_i), i = 2, 3, 4 \dots (l - 1)$$

Sendo \vec{N}_l o vetor resultado dos neurônios da camada i , $f_i(n)$ a função de ativação da camada i , W_i a matriz de pesos da camada i , \vec{N}_{l-1} o vetor resultado dos neurônios da camada $i - 1$ e \vec{B}_i o vetor de fatores de viés (*bias*) da camada i . O índice i pode assumir valores entre 2 e l .

$$\vec{O}_p = \vec{N}_l$$

Sendo \vec{O}_p o vetor resultado da rede, que é igual ao vetor resultado da última camada da rede (camada l).

$$\vec{O} = f_{pós-processamento}(\vec{O}_p)$$

Sendo \vec{O} o vetor resultado pós-processado da rede MLP, assim como o vetor de saída da rede MLP.

2. Exportando uma Rede Neural MLP criada em MATLAB:

A ferramenta MATLAB utiliza, por padrão, pré-processamento de entrada e pós-processamento de saída durante o treinamento e execução das redes. Com todo o equacionamento acima, podemos então, replicar uma Rede Neural MLP do MATLAB desde que tenhamos:

- Os parâmetros e algoritmos utilizados no pré-processamento das entradas;
- Os parâmetros e algoritmos utilizados no pós-processamento das saídas;
- As funções de ativação de cada camada;
- Os pesos conectando cada neurônios de rede;
- Os fatores de viés (*bias*) de cada neurônio da rede.

No MATLAB, Redes Neurais MLP são representadas pela estrutura “*network*”. Essa estrutura contém todas as informações referentes a rede, incluindo as informações listadas acima. Será mostrado aqui apenas como pegar os parâmetros necessários para o cálculo manual de uma rede.

Os parâmetros de pré-processamento se encontram dentro da subestrutura “*inputs*”, dentro da estrutura “*network*”. Essa subestrutura é um conjunto cujo tamanho depende da quantidade de entradas da rede, que é diferente da quantidade de neurônios de entrada, e descreve as características das entradas da rede. Estamos especialmente interessados nos campos “*processFcns*” e “*processSettings*”, que descrevem, respectivamente, as funções de pré-processamento e seus parâmetros.

Os parâmetros de pós-processamento se encontram dentro da subestrutura “*outputs*”, dentro da estrutura “*network*”. Essa subestrutura é um conjunto cujo tamanho depende da quantidade de saídas da rede, que é diferente da quantidade de neurônios de saída, e descreve as características das saídas da rede. Estamos especialmente interessados nos campos “*processFcns*” e “*processSettings*”, que descrevem, respectivamente, as funções de pré-processamento e seus parâmetros.

As funções de ativação de cada camada se encontram dentro da substrutura “*layers*”, dentro da estrutura “*network*”. Essa subestrutura é um conjunto cujo tamanho depende da quantidade de camadas da rede, e descreve as características das camadas da rede. Estamos especialmente interessados no campo “*transferFcn*” que descreve a funções de ativação da camada.

Os pesos conectando os neurônios da rede se encontram dentro das substruturas “*IW*” e “*LW*”, dentro da estrutura “*network*”. A subestrutura “*IW*” é um conjunto cujo tamanho depende da quantidade de camadas e de entradas (diferente da quantidade de neurônios de entrada) da rede, e descreve os pesos de cada entrada por camada. Já a subestrutura “*LW*” é um conjunto cujo tamanho depende da quantidade de camadas da rede, e descreve os pesos de cada camada interconectada.

Os fatores de viés (*bias*) de cada neurônio da rede se encontram dentro da substrutura “*b*”, dentro da estrutura “*network*”. A subestrutura “*b*” é um conjunto cujo tamanho depende da quantidade de rede, e descreve o *bias* de cada neurônio por camada.

3. Exemplo de aplicação:

Com o objetivo de prever a tensão gerada em um painel solar, baseada na temperatura e radiação solar aos quais ela está exposta, foi criada uma rede neural com a ferramenta “*nntool*”, do MATLAB. No entanto, se tornou necessário executar essa Rede MLP fora do ambiente MATLAB, portanto é necessária exporta-lá.

A estrutura da rede criada pode ser vista na Figura F.3, abaixo. Analisando a estrutura “*network*” da rede MLP criada, sabemos que ela possui:

- 2 entradas, com função de pré-processamento $f(x) = \text{mapminmax}(x)$;
- 2 camadas:
 - 1 camada oculta com 4 neurônios e função de ativação $f(n) = \text{tansig}(n)$;
 - 1 camada de saída com um neurônio e função de ativação $f(n) = \text{purelin}(n)$.
- 1 saída, com função de pós-processamento $f(y) = \text{mapminmax}^{-1}(y)$;

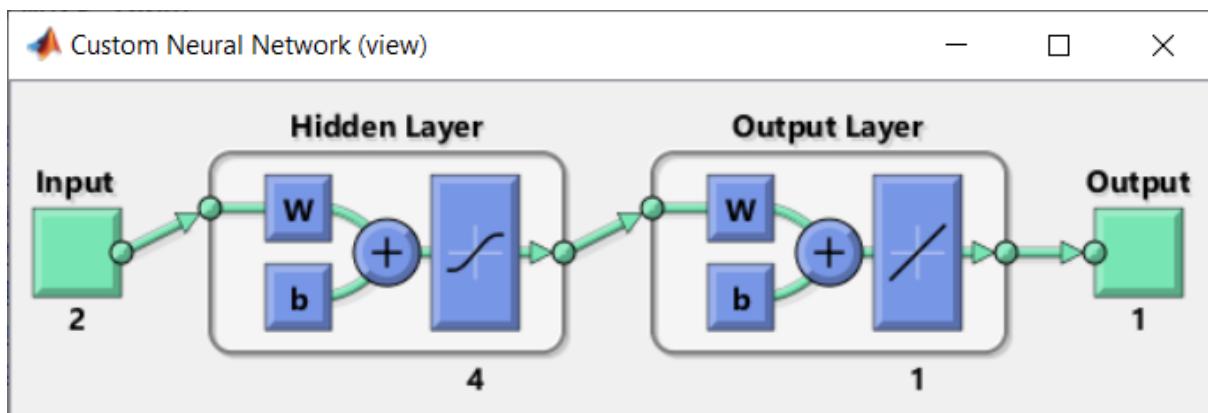


Figura F.3 - Estrutura da rede criada, pelo método "view()"

4. Implementação do algoritmo de cálculo da Rede Neural MLP:

Agora que localizamos todas as informações necessárias para replicar a rede, incluindo sua estrutura, podemos efetivamente fazer uma implementação manual dela. As

implementações demonstradas nesses documentos serão feitas em script MATLAB, e podem ser exportadas para outras linguagens conforme necessidade.

A maneira mais simples e eficaz de executar uma rede neural no ambiente MATLAB é simplesmente chamar a estrutura “*network*” da rede. Ao chamar a estrutura, o ambiente MATLAB realiza todos os cálculos da rede e fornece um vetor de saídas para um vetor de entradas. Assim também obtemos um valor de referência para validar outras implementações. Esta implementação pode ser encontrada abaixo.

```
%%
% Resolução direta, pela estrutura de network do MATLAB

% Vetor de entradas do sistema (valor não processado)
In = [
    -0.963430000000000;
    -0.967795000000000
];

% Vetor de saída do sistema (valor pós-processado)
Out_pp = Tensao4(In);
disp(Out_pp);
```

O resultado obtido por esse método é $Out_{pp} = 24,760732076591324$.

Para permitir a obtenção do mesmo resultado em outros ambientes, precisamos os passos e equações já definidos neste documento para:

1. Escrever as equações matriciais e vetoriais necessárias para a execução da rede gerada;
2. Converter as equações matriciais e vetoriais em equações lineares, permitindo sua utilização em qualquer ambiente;
3. Realizar manualmente o cálculo das funções de pré-processamento e pós-processamento, assim como das funções de ativação de cada camada;
4. Extrair da estrutura “*network*” da rede todas as constantes necessárias para os cálculos.

Executando todos esses passos, podemos reescrever a implementação acima como:

```

%%

% Resolução completamente manual, equacionamento não matricial,
% constantes do MATLAB, sem funções

% Pesos da primeira camada oculta
weight_l1n1_in1 = Tensao4.IW{1}(1, 1);
weight_l1n1_in2 = Tensao4.IW{1}(1, 2);
weight_l1n2_in1 = Tensao4.IW{1}(2, 1);
weight_l1n2_in2 = Tensao4.IW{1}(2, 2);
weight_l1n3_in1 = Tensao4.IW{1}(3, 1);
weight_l1n3_in2 = Tensao4.IW{1}(3, 2);
weight_l1n4_in1 = Tensao4.IW{1}(4, 1);
weight_l1n4_in2 = Tensao4.IW{1}(4, 2);

% Pesos da camada de saída
weight_out1_l1n1 = Tensao4.LW{2,1}(1, 1);
weight_out1_l1n2 = Tensao4.LW{2,1}(1, 2);
weight_out1_l1n3 = Tensao4.LW{2,1}(1, 3);
weight_out1_l1n4 = Tensao4.LW{2,1}(1, 4);

% Viés da primeira camada oculta
bias_l1n1 = Tensao4.b{1}(1);
bias_l1n2 = Tensao4.b{1}(2);
bias_l1n3 = Tensao4.b{1}(3);
bias_l1n4 = Tensao4.b{1}(4);

% Viés da camada de saída
bias_out1 = Tensao4.b{2};

% Parâmetros de pré-processamento da entrada 1
prep_in1_xmin = Tensao4.inputs{1}.processSettings{1}.xmin(1);
prep_in1_xmax = Tensao4.inputs{1}.processSettings{1}.xmax(1);
prep_in1_ymax = Tensao4.inputs{1}.processSettings{1}.ymax;
prep_in1_ymin = Tensao4.inputs{1}.processSettings{1}.ymin;

% Parâmetros de pré-processamento da entrada 2
prep_in2_xmin = Tensao4.inputs{1}.processSettings{1}.xmin(2);
prep_in2_xmax = Tensao4.inputs{1}.processSettings{1}.xmax(2);
prep_in2_ymax = Tensao4.inputs{1}.processSettings{1}.ymax;
prep_in2_ymin = Tensao4.inputs{1}.processSettings{1}.ymin;

% Parâmetros de pós-processamento da saída 1
posp_out1_xmin = Tensao4.outputs{2}.processSettings{1}.xmin;
posp_out1_xmax = Tensao4.outputs{2}.processSettings{1}.xmax;
posp_out1_ymax = Tensao4.outputs{2}.processSettings{1}.ymax;
posp_out1_ymin = Tensao4.outputs{2}.processSettings{1}.ymin;

% Vetor de entradas do sistema (valor não processado)
In = [
    -0.9634300000000000;
    -0.9677950000000000
];

% Entradas individuais da rede (valor pré-processado)
% -- mapminmax(x) = (ymax-ymin)*(x-xmin)/(xmax-xmin) + ymin;
in1_pp = ((prep_in1_ymax - prep_in1_ymin) * (In(1) - prep_in1_xmin)) /
(prep_in1_xmax - prep_in1_xmin) + prep_in1_ymin;
in2_pp = ((prep_in2_ymax - prep_in2_ymin) * (In(2) - prep_in2_xmin)) /
(prep_in2_xmax - prep_in2_xmin) + prep_in2_ymin;

```

```
% Cálculo dos neurônios da primeira camada oculta
% -- tansig(n) = 2/(1+exp(-2*n))-1;
l1n1 = (2 / ( 1 + exp(-2 * (weight_l1n1_in1 * in1_pp + weight_l1n1_in2 * in2_pp + bias_l1n1)))) - 1;
l1n2 = (2 / ( 1 + exp(-2 * (weight_l1n2_in1 * in1_pp + weight_l1n2_in2 * in2_pp + bias_l1n2)))) - 1;
l1n3 = (2 / ( 1 + exp(-2 * (weight_l1n3_in1 * in1_pp + weight_l1n3_in2 * in2_pp + bias_l1n3)))) - 1;
l1n4 = (2 / ( 1 + exp(-2 * (weight_l1n4_in1 * in1_pp + weight_l1n4_in2 * in2_pp + bias_l1n4)))) - 1;
% Cálculo dos neurônios da camada de saída, equivalente as saídas individuais da rede (valor não processado)
% -- purelin(n) = n;
out1 = (weight_out1_l1n1 * l1n1 + weight_out1_l1n2 * l1n2 + weight_out1_l1n3 * l1n3 + weight_out1_l1n4 * l1n4 + bias_out1);

% Saídas individuais do sistema (valor pós-processado)
% -- mapminmax^(-1)(y) = (y-ymin)*(xmax-xmin)/(ymax-ymin) + xmin;
out1_pp = ((out1 - posp_out1_ymin) * (posp_out1_xmax - posp_out1_xmin) / (posp_out1_ymax - posp_out1_ymin)) + posp_out1_xmin;
% Vetor de saída do sistema (valor pós-processado)
Out_pp = [out1_pp];
disp(Out_pp);
```

Onde temos apenas equações lineares e operações matemáticas simples, com exceção da função exponencial (que pode ser facilmente encontrada em outros ambientes e ferramentas).

O resultado obtido por essa implementação é $Out_{pp} = 24,760732076591324$, que é numericamente igual ao obtido usando a estrutura interna de redes do MATLAB.

A implementação demonstrada acima independe do ambiente MATLAB e pode ser facilmente exportada para outros ambientes e ferramentas, sendo necessário apenas:

- Que o ambiente ou ferramenta suporte contas com pontos flutuantes e o cálculo da função exponencial (e^x);
- A substituição de todas as constantes da estrutura “*network*” por seus valores numéricos.

APÊNDICE G – TABELA DE CUSTOS

RELAÇÃO DE PREÇOS PARA PROJETO		
QTDE	MATERIAIS	PREÇO
1	Sensor de Chuva - YL-83	R\$ 14,90
1	Sensor de Temperatura - DS18B20	R\$ 18,90
1	Sensor de Tensão	R\$ 25,00
1	Sensor de Luz - BH1750FVI (LUX)	R\$ 16,90
1	Sensor de Corrente - ACS712 5A	R\$ 19,00
1	Módulo Regulador De Tensão Step-up	R\$ 13,05
1	BMS - Bms 3s 10a Carregador Bateria Li-íon	R\$ 20,00
1	Baterias - 3 Baterias 18650 2200mah	R\$ 39,90
1	Antena p/ Shield Draguino	R\$ 14,90
1	Ardino MEGA	R\$ 149,41
1	Caixa usinada em ABS - 120g	R\$ 6,00
1	Módulo Bluetooth - HM 10	R\$ 16,50
1	Shield Draguino	R\$ 209,90
1	Conectores Painel solar	R\$ 14,90
1	Cabo de 4mm2	R\$ 26,90
15	Σ	R\$ 606,16