UNIVERSIDADE FEDERAL DE VIÇOSA
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

LUCAS BRAGA CARDOSO

IMPLEMENTAÇÃO DE OPC SERVER EM SOFTWARE SUPERVISÓRIO VISANDO A INTEGRAÇÃO DE VARIÁVEIS TRANSMITIDAS POR DIFERENTES TECNOLOGIAS

VIÇOSA
2020

LUCAS BRAGA CARDOSO

IMPLEMENTAÇÃO DE OPC SERVER EM SOFTWARE SUPERVISÓRIO VISANDO A INTEGRAÇÃO DE VARIÁVEIS TRANSMITIDAS POR DIFERENTES TECNOLOGIAS

Monografia apresentada ao Departamento de Engenharia Elétrica da Universidade Federal de Viçosa em cumprimento ao requisito parcial para a obtenção do título de Bacharel em Engenharia Elétrica.

Orientador: Prof. André Gomes Torres

VIÇOSA

2020

LUCAS BRAGA CARDOSO

IMPLEMENTAÇÃO DE OPC SERVER EM SOFTWARE SUPERVISÓRIO VISANDO A INTEGRAÇÃO DE VARIÁVEIS TRANSMITIDAS POR DIFERENTES TECNOLOGIAS

Monografia apresentada ao Departamento de Engenharia Elétrica da Universidade Federal de Viçosa em cumprimento ao requisito parcial para a obtenção do título de Bacharel em Engenharia Elétrica.

Aprovada em 08 de dezembro de 2020.

Prof. Dr. André Gomes Torres - Orientador

Universidade Federal de Viçosa - DEL

Profa. Dra. Kétia Soares Moreira – Membro da Banca

Universidade Federal de Viçosa - DEL

Prof. Dr. Rodolpho Vilela Alves Neves – Membro da Banca

Universidade Federal de Viçosa - DEL

Aos meus pais, à minha noiva, aos meus irmãos e a todos os amigos que estiveram presentes durante a caminhada.

AGRADECIMENTOS

Agradeço aos meus pais, José Carlos e Marcia, por todo o apoio dado ao longo desses anos, mesmo que de tão longe. Vocês acreditaram no meu sonho e garantiram que eu tivesse a oportunidade de estudar e criaram as condições para que eu chegasse onde estou.

Agradeço à minha noiva, Isadora, a principal razão pela qual escolhi a UFV. Você plantou a semente muito tempo antes de eu me dar conta de para onde queria ir e por isso sou muito grato. Você foi o abraço nos momentos de estresse e frustação e o riso nos momentos de alegria.

Agradeço aos meus irmãos, Douglas e Maicon, pelas visitas, pela confiança em minhas capacidades, pelos deslocamentos quilométricos para fornecer uma carona e por todo o apoio.

Agradeço aos amigos que fiz nesse tempo que aqui passei. Os almoços, a companhia na volta para casa, os encontros na BBT e todas as alegrias compartilhadas tornaram os estudos muito mais suportáveis.

Agradeço aos professores que realmente fizeram diferença em minha graduação, em especial ao professor André, que confiou em minhas capacidades. A Iniciação Científica e todas as demais coisas decorrentes dela tiveram um impacto muito grande em minha vida acadêmica e acredito que ainda terão em um futuro próximo.

Agradeço aos funcionários do Departamento de Engenharia Elétrica que contribuíram para a minha pesquisa, em especial ao Lúcio e ao Cristiano, que tanto me socorreram nos momentos que tudo dava errado.

Agradeço à Viçosa e à UFV por todos os momentos que aqui passei. Não tenho dúvidas em dizer que esses foram os melhores anos que vivi até hoje.

Por fim, agradeço a todos que de alguma forma foram importantes para a minha conquista, perto ou longe. Meus sinceros agradecimentos.

A ciência de hoje é a tecnologia de amanhã.

Edward Teller

RESUMO

Esta monografia tem como objetivo descrever o desenvolvimento de um supervisório capaz de realizar a integração e o compartilhamento das grandezas temperatura, transmitida via rede Foundation Fieldbus, e tensão e corrente, transmitidas via USB. Os dados de temperatura foram armazenados em um servidor e seu acesso, a partir do supervisório, pôde ser realizado com a implementação de um OPC Server, transformando a máquina hospedeira do supervisório em um OPC Client. Os sinais de tensão e corrente foram coletados por uma placa de aquisição de dados com conexão USB. A metodologia empregada consistiu na melhoria de um protótipo existente, no acréscimo de novos dispositivos (com o intuito de inserir a tecnologia Fieldbus ao projeto), na programação do software supervisório e no desenvolvimento de sua interface gráfica. O destaque deste trabalho se encontra no uso de tecnologias de aquisição de dados presentes na indústria e em sistemas automatizados, permitindo a criação de novos meios de acesso à informação dentro das fábricas, mesmo que coletadas por sistemas diferentes. O software apresentou bom funcionamento, conseguindo não apenas receber e exibir as informações de temperatura armazenadas no servidor, como também exibir e fornecer ao servidor os dados de tensão e corrente, vindas da placa de aquisição.

ABSTRACT

This monograph aims to define the development of a supervisory capable of integrating and sharing the values of temperature, transmitted via the Foundation Fieldbus network, voltage and current, transmitted via USB. The temperature data was stored on a server and its access, from the supervisory, can be carried out with the implementation of an OPC server, transforming a machine that hosts the supervisory into an OPC client. The voltage and current signals were collected by a data acquisition board with a USB connection. The methodology employed consists of improving an existing prototype, without adding new devices (with or with the intention of inserting Fieldbus technology into the project), programming the software supervisory and developing its graphical interface. The highlight of this work is in the use of data acquisition technologies present in the industry and in automated systems, allowing the creation of new means of accessing information within the factories, even if collected by different systems. The software presented smooth operation, managing not only to receive and display the temperature information stored on the server, but also to display and provide the server with voltage and current data, coming from the acquisition board.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

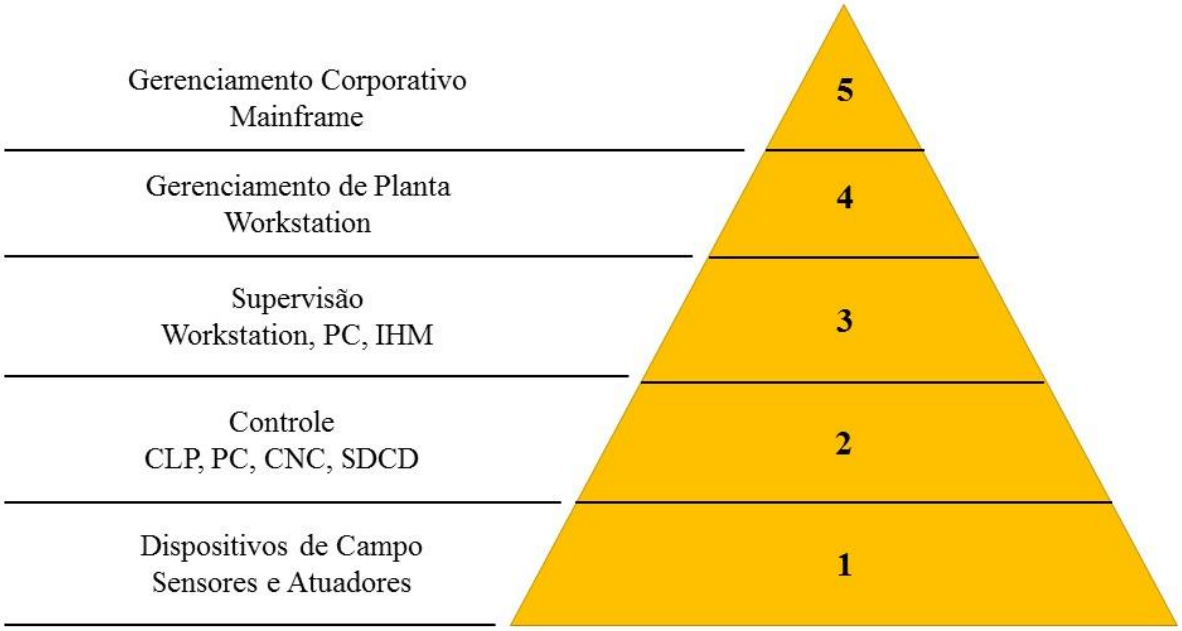
SUMÁRIO

1 INTRODUÇÃO

A automação industrial, mesmo sendo um assunto de destaque hoje, não é um tema novo. Historicamente, pode-se dizer que seu início ocorreu no século XVIII, durante a primeira revolução industrial e com o uso das máquinas a vapor para substituir grande parte dos trabalhos manuais. A evolução tecnológica que houve desde então levou o mundo a passar por mais outras duas revoluções no meio industrial, chegando ao ponto onde tornou-se necessário desenvolvimentos nos campos da aquisição, do processamento e da distribuição de informações; um dos efeitos observados neste século, devido ao rápido crescimento tecnológico, é o da convergência de todas essas áreas, culminando em um breve fim daquilo que difere coleta, processamento, transporte e armazenamento de dados (TANENBAUM; WETHERALL, 2011).

Sobre a organização da indústria no que se refere aos sistemas automatizados, Bruckner *et al*. (2019) afirma que a estrutura de muitos sistemas de automação atuais, conhecida como pirâmide de automação, data dos anos 1970 e foi desenvolvida como uma maneira de lidar com a complexidade dos sistemas que evoluíam na época. Sabe-se que hoje essa não é a única estrutura possível, porém, independentemente da forma de organização, o acesso à informação é fundamental na tomada de decisões de qualquer processo, devendo-se facilitar tal ação e torná-la mais eficiente. Na Figura 1 está ilustrado um exemplo da pirâmide de automação.

Figura 1 - Pirâmide da automação industrial.



Fonte: O Autor.

Para Eckhardt, Müller e Leurs (2018), no atual cenário da automação dos processos fabris, a troca de informações possui demasiada importância e isto tende a aumentar. Ainda, para os autores, deve-se fazer uso de uma única tecnologia de comunicação em todos os níveis da pirâmide de automação, desde o chão de fábrica até os instrumentos de decisão. O maior desafio é encontrar uma tecnologia de comunicação que seja capaz de atender todos os diferentes requisitos que cada nível possui.

Uma das formas de atingir tal objetivo é através do desenvolvimento ou aprimoramento de um sistema de aquisição de dados (*Data Acquisition System*, em inglês, ou DAQ *System*). De acordo com Measurement Computing (2012), um sistema de aquisição de dados é capaz de medir, analisar, exibir e armazenar informações provenientes das mais diversas fontes ao nosso redor. Logo, melhorar um sistema DAQ implica em criar um caminho mais fluido e confiável para o fluxo de informações.

Belchior (2014) caracteriza a medição como um conjunto de ações que objetiva a comparação de duas grandezas de uma mesma espécie e a determinação do valor momentâneo de uma em relação a outra, que é tomada como referência. Helfrick e Copper (1994) observam que, quase sempre, o ato de medir exige o uso de instrumentos adequados como uma forma de extensão das limitações das capacidades humanas. Existem instrumentos para a medição de praticamente qualquer grandeza, indo desde simples termômetros de bulbo até sofisticadas placas eletrônicas de aquisição.

Após uma grandeza ser mensurada, ela precisa ser transmitida para que os processos de análise e exibição sejam realizados. A comunicação é de suma importância para o trabalho desenvolvido, sendo considerada por Bruckner *et al.* (2019), um dos elementos mais importantes da pirâmide de automação. As informações capturadas pelos instrumentos de campo devem ser transmitidas até elementos inteligentes para que decisões sejam tomadas.

Buscando se manter em lugar de destaque no cenário econômico mundial, a Alemanha propôs um novo modelo de arquitetura industrial, utilizando tecnologias de alto nível, o que para muitos pode ser visto como a realização das fábricas digitais (CHEN; TAI; CHEN, 2017). Esse modelo, chamado de Indústria 4.0, ganhou força e espaço dentro e fora das fábricas (com a conhecida Internet das Coisas) e sua expansão tende a implicar em um grande aumento do número de instrumentos inteligentes no chão de fábrica, assim como dos protocolos digitais de comunicação utilizados por estes dispositivos.

Hoje, pode-se afirmar que o mercado dos protocolos de comunicação industrial é totalmente dominado por instrumentos que utilizam a rede Ethernet baseada em sistemas do

tipo *bus* (BRUCKNER *et al*., 2019). Um exemplo de protocolo muito difundido na indústria é o Fieldbus que, segundo o FieldComm Group (2020), é um protocolo digital que permite comunicação entre instrumentos e sistemas e que utilizam, basicamente, uma rede local.

Para Dongjiang e Ruiqi (2011), a falta de um padrão de comunicação universal entre os diversos protocolos coexistentes causa dificuldades à integração de diferentes sistemas e, mesmo que possuam requisitos próximos, suas implementações costumam divergir consideravelmente. Por mais que não seja um protocolo de comunicação universal, em 1996 a Microsoft anunciou o desenvolvimento de um padrão que serviria como uma interface para troca de dados, resolvendo o problema de interoperabilidade existente na comunicação entre protocolos e sistemas diferentes (SCHWARZ; BÖRCSÖK, 2013). Esse padrão recebeu originalmente o nome de OLE *for Process Control* (OPC) e atualmente uma vasta gama de fabricantes oferecem suporte ao padrão em seus instrumentos.

A aquisição de dados é apenas um dos muitos campos dominados pelos computadores, tendo em vista a grande capacidade de processamento que possuem e a velocidade com a qual são capazes de realizá-lo. Na verdade, os computadores foram introduzidos nos sistemas de controle e automação desde a década de 1960 (DONGJIANG; RUIQI, 2011). No mercado estão disponíveis inúmeros softwares que podem realizar os trabalhos de analisar e disponibilizar dados coletados, porém pode ser mais interessante em alguns casos (ou até mesmo necessário) que um programa específico seja utilizado para que a tarefa seja realizada sem problemas. Sistemas SCADA (*Supervisory Control and Data Acquisition)* são um desses programas especiais, responsáveis por receber, tratar e exibir dados, e ainda executar ações de controle nos sistemas onde estão inseridos.

Após medir, transportar e analisar as informações, restam exibi-las e armazená-las para finalizar um processo de aquisição de dados. A exibição de dados pode ser feita de várias formas, desde displays nos próprios instrumentos até telas de supervisórios. Computadores comerciais muitas vezes serão ótimos para o armazenamento de dados, mas em outras ocasiões não serão suficientes. Servidores podem ser alternativas viáveis e possuem a vantagem de possibilitarem seu acesso localmente, através de uma conexão à rede utilizando cabos, ou remotamente, via internet, por um ou mais sistemas e equipamentos.

Para a realização deste trabalho, desejou-se que um software supervisório fosse capaz de receber dados[1] de tensão e corrente, coletados a partir de uma placa de aquisição e transmitidos via USB, e dados de temperatura, coletados por um sensor e transmitidos via rede

[1] Todas as grandezas foram coletadas de um mesmo motor de indução trifásico.

Foundation Fieldbus. Além disso, também era de interesse que todos os dados permanecessem disponíveis em um servidor. Logo, fez necessário encontrar uma maneira de receber e compartilhar as informações transmitidas pelas duas tecnologias, atendendo aos requisitos de velocidade e confiabilidade. Diante do exposto, foi suposto que a implementação do padrão OPC ao sistema seria capaz de resolver o problema da incompatibilidade de protocolos.

Assim, o presente trabalho teve como objetivo geral o desenvolvimento de um supervisório que fosse capaz de integrar e compartilhar grandezas que foram medidas e transmitidas por protocolos de comunicação diferentes. Para atingir tal meta, seguiu-se os seguintes objetivos específicos:

- Melhorar o ambiente de trabalho existente, incorporando novos equipamentos no protótipo a ser utilizado nos testes;
- Instalar todos os softwares, bibliotecas e *assembles* necessários para utilizar os novos dispositivos e para realizar a implementação do padrão OPC no supervisório;
- Estabelecer a comunicação entre o supervisório e um servidor por intermédio de um OPC Server.

As grandezas integradas foram tensão e corrente, medidas por uma placa de aquisição de dados de padrão industrial e transmitidas via USB, e temperatura, medida por um termistor acoplado a um transmissor de temperatura e transportada pelo protocolo Fieldbus.

Buscou-se desenvolver um supervisório gratuito, capaz de fornecer dados confiáveis e que demandasse baixos recursos de processamento. Foram utilizados diversos dispositivos e tecnologias amplamente difundidas na indústria, tais como a placa de aquisição de dados e o protocolo digital de comunicação, possibilitando a criação de novos métodos de acesso às informações.

É importante destacar que a pesquisa aqui abordada foi desenvolvida durante um projeto de iniciação científica, fomentada pelo CNPq, e rendeu publicações em anais de congresso e em revista especializada em divulgação científica.[2]

[2] Trabalho publicado nos Anais do I Congresso Brasileiro Interdisciplinar de Ciência e Tecnologia (I CoBICET) e na revista Brazilian Journal of Development. DOI: 10.34117/bjdv6n9-496.

2 REVISÃO DE LITERATURA

Neste capítulo será realizada uma breve revisão sobre a rede Foundation Fieldbus, padrão OPC e programação orientada a objetos, com o intuito de apresentar conceitos e informações pertinentes para a compreensão do trabalho.

2.1 REDES DE COMPUTADORES E PROTOCOLOS DE COMUNICAÇÃO

Tanenbaum e Wetherall (2011) definem as redes de computadores como um conjunto de computadores separados, mas que estão interconectados, fisicamente ou não. Este conceito pode ser estendido não apenas para computadores comerciais, mas também para outros dispositivos inteligentes. Dentre as redes encontram-se as *Local Area Network* (LAN) ou redes locais, que são redes particulares cujo alcance é limitado ao espaço de um prédio e suas proximidades. O popularmente conhecido Wi-Fi é um exemplo de LAN.

De maneira simplificada, pode-se dizer que uma rede é comumente dividida em várias camadas ou níveis com a intenção de reduzir a sua complexidade. A quantidade de camadas e o conteúdo pelo qual cada uma é responsável é o que difere uma rede da outra. A intenção dessa divisão é fazer com que cada nível execute as ações para as quais foram programadas, sem precisar se preocupar com os detalhes de implementação da camada imediatamente abaixo.

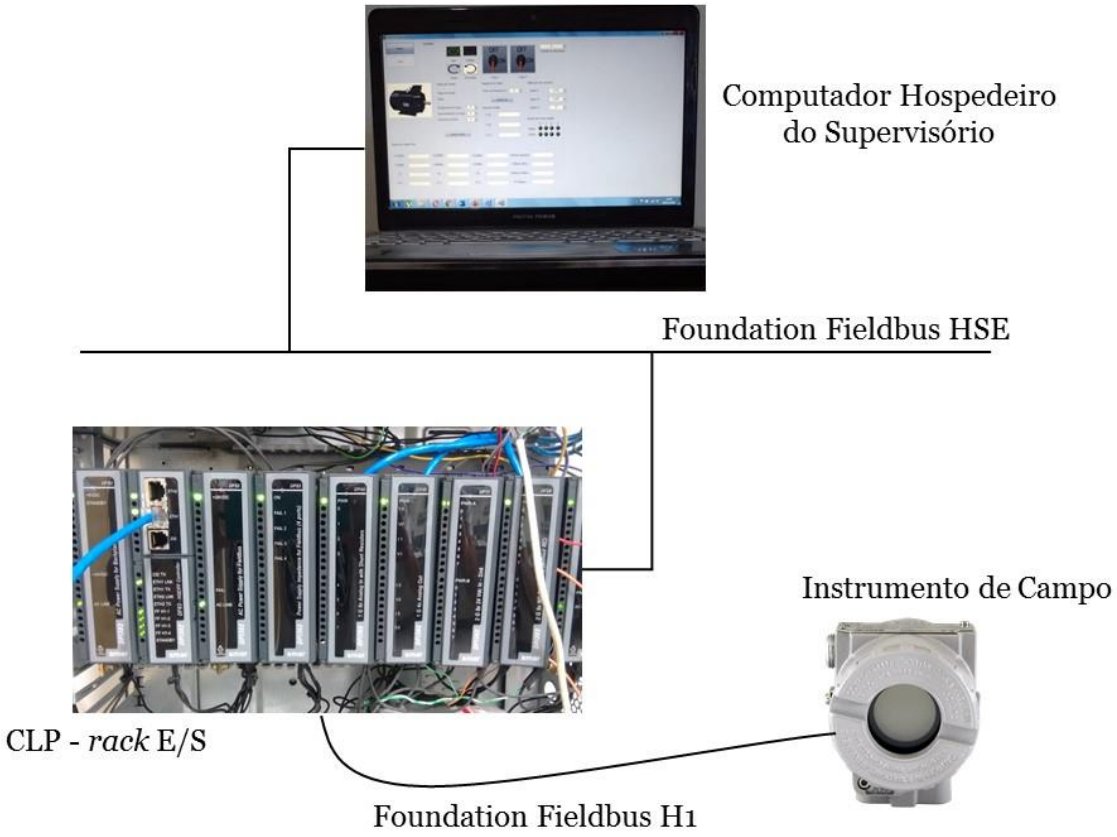
Neste contexto, é importante abordar o conceito de protocolo de comunicação. Um protocolo pode ser definido como as regras as quais duas camadas correspondentes estão sujeitas quando desejam estabelecer a comunicação, isto é, um protocolo é um acordo que dita como a comunicação entre duas partes ocorrerá (TANENBAUM; WETHERALL, 2011).

A Foundation Fieldbus (FF), utilizada neste trabalho, é um modelo de LAN desenvolvida exclusivamente para o controle de processos. O protocolo Fieldbus pode ser dividido em um nível físico, que diz respeito à interligação dos equipamentos, e níveis de software, para a comunicação digital entre os dispositivos. A camada física pode ser do tipo H1, destinada à conexão dos equipamentos de campo, ou HSE, para a conexão entre computadores e CLPs (AZEVEDO *et al*., 2017). Na Figura 2 exibe-se a estrutura física da rede FF, apresentando as camadas H1 e HSE.

Diferentemente da Foundation Fieldbus, o tão conhecido *Universal Serial Bus* (USB) não é uma rede, mas sim um padrão criado pela indústria para homogeneizar os conectores dos

equipamentos. A comunicação estabelecida pelo USB utiliza um protocolo próprio para executar a transferência de dados.

Figura 2 - Organização das camadas físicas da rede Foundation Fieldbus.



Fonte: O Autor.

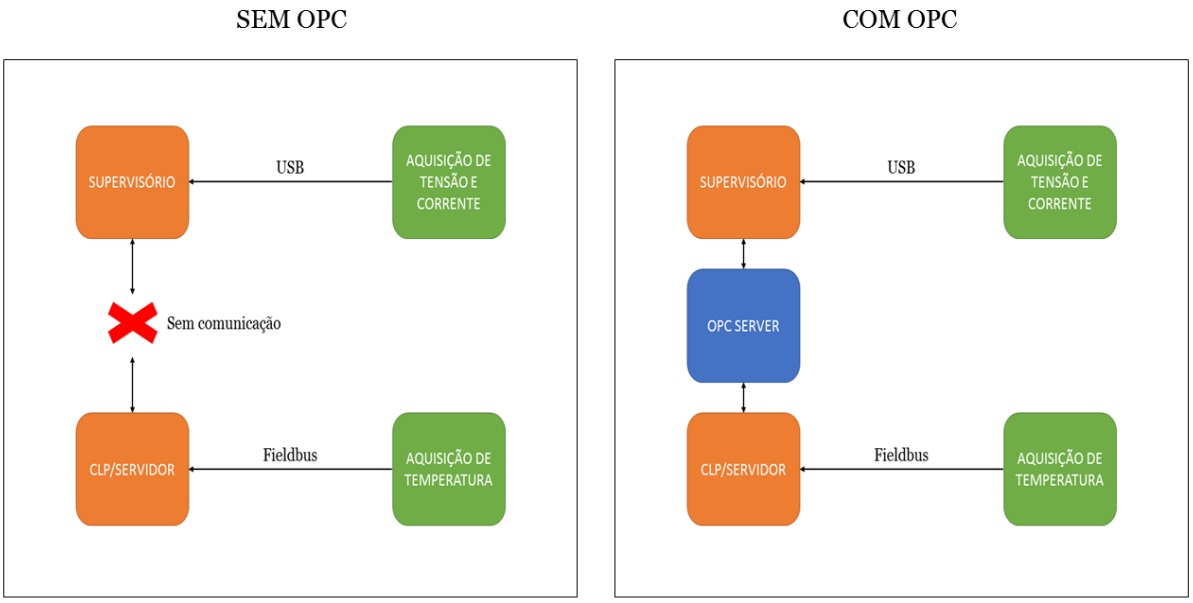
Não é o objetivo desta monografia focar nas diferentes redes e no modo como cada protocolo funciona, mas sim destacar que o protocolo Fieldbus é diferente do protocolo usado pelo padrão USB. Logo, como os protocolos são diferentes, a comunicação não respeita as mesmas regras, tornando-a impossível de ser realizada diretamente. Como no trabalho desenvolvido há uma placa de aquisição de dados USB e um transmissor de temperatura e um CLP que utilizam a tecnologia Fieldbus, não é possível que o supervisório leia e escreva dados do servidor de forma direta. Para isso faz-se necessária a implementação do OPC Server.

2.2 PADRÃO OPC E O OPC SERVER

Durante a década de 80, tanto a indústria quanto as universidades se empenharam no desenvolvimento de novos protocolos de comunicação, culminando em pelo menos 50 diferentes sistemas do tipo *bus* em alguns poucos anos (SCHWARZ; BÖRCSÖK, 2013). Porém, cada novo sistema dependia de um hardware ou de um software específico, diferente dos demais, e, se ocorresse alguma atualização, geralmente era necessária uma troca completa de dispositivos. Claramente, isso gerava um grande custo para as empresas, além do risco de falhas no processo de produção.

Com a intenção de resolver os problemas da incompatibilidade de protocolos, por volta de 1996, cinco empresas se uniram em um empreendimento. Desse esforço comum surgiu o padrão que hoje é conhecido como *OLE for Process Control* (OPC) e com isso uma maneira de sistemas ou dispositivos se comunicarem com outros sistemas diferentes ou semelhantes – a interoperabilidade. De acordo com a OPC Foundation (2020), a fundação responsável pela administração da tecnologia OPC, esse padrão foi construído com uma série de especificações usadas para estabelecer a comunicação entre dois servidores ou entre um servidor e um outro dispositivo, chamado de cliente. Na Figura 3 ilustra-se uma comparação entre dois processos de comunicação, onde o quadro da esquerda não possui o padrão OPC e o da direita possui.

Figura 3 - Comparação entre os processos de comunicação sem o uso do OPC e com o uso do OPC.



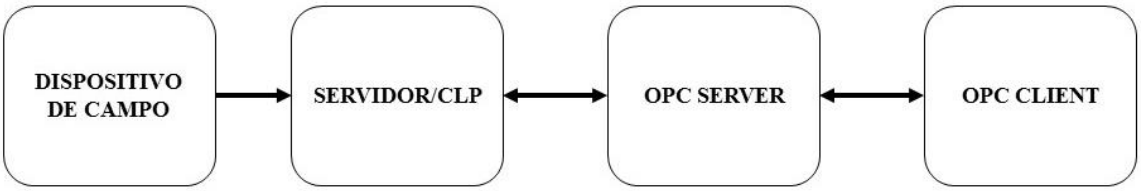
Fonte: O Autor.

A primeira versão, chamada de OPC *Data Access* (OPC-DA), possui suporte apenas para os sistemas operacionais do tipo Windows, pois a arquitetura cliente-servidor é baseada na tecnologia OLE, pertencente à Microsoft, que dominava o mercado de sistemas operacionais na época. Segundo Schwarz e Börcsök (2013), o padrão propõe que a troca de dados de processos ocorra seguindo um formato específico que é: *Process Value*, uma variável quantitativa, a qual indica o valor da grandeza transmitida; *Time Stamp*, que carrega a informação de data e hora da transmissão; e *Reliability*, uma variável qualitativa, que informa o nível de confiabilidade da informação transmitida, podendo ser *good*, *unknown* ou *bad*.

Um dos objetivos que motivou o desenvolvimento do OPC foi o de criar um conjunto de padrões de comunicação aos quais servidores de dados e softwares de controle estivessem submetidos, garantindo maior eficiência e estabilidade no acesso aos dados (QING; YONGSHENG, 2015).

Uma das ferramentas responsáveis pela capacidade de comunicação sem entraves é o OPC Server, que funciona como um conversor de protocolos. Dados presentes em CLP, servidores físicos ou virtuais, dispositivos ou interfaces de usuário são convertidos para o padrão OPC e dessa forma podem ser acessados por quem deseja as informações – os OPC Clients – (NICULESCU; SAVESCU; MITRU, 2018). Na Figura 4 exibe-se um esquema indicando a localização do OPC Server em um sistema de controle.

Figura 4 - Esquema ilustrativo da localização do OPC Server no sistema.



Fonte: O Autor.

2.3 PROGRAMAÇÃO ORIENTADA A OBJETOS

Em um ambiente industrial, espera-se que todo o processo de produção seja executado de forma precisa a fim de minimizar os gastos e por isso o uso de softwares profissionais é essencial. Entretanto, no presente trabalho, tendo como foco a criação de um programa computacional, isso não foi possível ou mesmo necessário. Para conseguir programar, necessita-se, obviamente, conhecer pelo menos uma dentre as inúmeras linguagens de

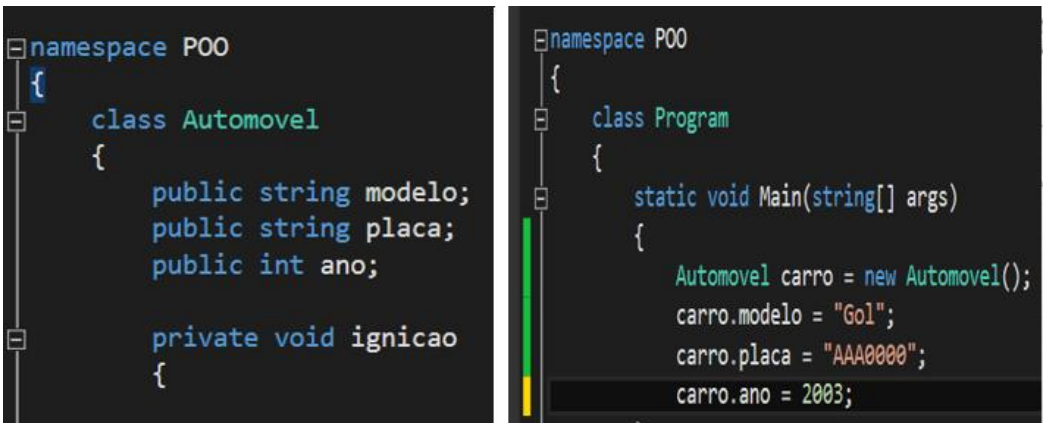
programação existentes e também do paradigma dentro da qual ela se localiza. Segundo Teixeira (2017), pode-se definir um paradigma de programação como a maneira sob a qual um código é estruturado ou qual é a forma que ele deve ter.

Existe hoje vários tipos de paradigmas, sendo o orientado a objetos o alvo de interesse. A programação orientada a objetos, também chamada de POO, apresenta diversas vantagens, como por exemplo uma maior proximidade com a linguagem humana, pois trata o código como a junção de objetos – motivo do nome –, tendo cada objeto propriedades distintas. Além disso, os conceitos de herança e polimorfismo se fazem muito importantes neste paradigma por estenderem definições já existentes e acrescentar maior dinamismo ao código. Muitas das mais conhecidas linguagens de programação pertencem ao paradigma orientado a objetos, dentre os quais pode-se destacar C++, Python, Java e C#, sendo a última utilizada no desenvolvimento do supervisório.

Para melhor compreendimento da POO, quatro conceitos devem ser conhecidos: classe, objeto, herança e polimorfismo. A classe é uma espécie de molde que define quais propriedades, ou atributos, serão aplicadas aos objetos daquela classe (CARVALHO; TEIXEIRA, 2012). Isso é realizado através dos métodos, que possuem uma relação direta com as funções presentes nas outras linguagens de programação.

Objetos são instâncias de classes e possuem as propriedades por ela definidas. Praticamente tudo pode ser representado como um objeto. Normalmente, os atributos que um objeto possui não devem ser de conhecimento público e para ocultar tais informações existe o encapsulamento. Esse processo ocorre da mesma maneira que as camadas de uma rede, isolando toda a informação que um outro objeto não necessite de ter acesso. Na Figura 5 encontra-se um exemplo para melhor explicar a relação entre classe e objeto.

Figura 5 - Exemplo da relação entre classe e objeto.



Fonte: O Autor.

No quadro da esquerda está a declaração da classe automóvel, cujos atributos ou propriedades são modelo, placa e ano e possui o método de ignição. Ou seja, estas são características que todo objeto pertencente à classe automóvel precisa possuir. No quadro da direita há a criação do objeto carro, o qual possui um determinado modelo, uma placa e um ano de fabricação.

Segundo Ricarte (2001), a herança é um mecanismo exclusivo do paradigma orientado a objetos que permite que outras classes herdem os atributos de uma superclasse e adicionem novas propriedades se assim desejarem. No exemplo anterior, a classe motocicleta poderia ser uma subclasse da superclasse automóvel, herdando os atributos modelo, placa e ano e ainda adicionando a propriedade tamanho do tanque, por exemplo. A herança faz com que o processo se torne mais rápido e o código mais limpo, haja vista que não é necessário ficar reescrevendo atributos já existentes.

Por fim, o polimorfismo é uma propriedade que permite com que duas ou mais subclasses consigam invocar métodos que tem a mesma identificação que na superclasse, porém com comportamentos distintos. O nome polimorfismo vem justamente dessa habilidade de mudar de forma, de executar novas tarefas com o mesmo nome em outra classe.

3 MATERIAL E MÉTODOS

O protótipo desenvolvido para a execução deste trabalho possui aplicação voltada para os motores de indução trifásico. Na Figura 6 pode-se visualizar o protótipo, com algumas partes destacadas utilizando cores distintas. Dentre os dispositivos que o compõe, nota-se uma placa de aquisição de dados (retângulo preto), dois sensores de corrente (retângulo azul), quatro placas de circuito impresso (retângulos verdes) e um banco com três transformadores monofásicos (retângulo amarelo). Além disso, o motor utilizado para a pesquisa é do tipo gaiola de esquilo, modelo M 610-AC-1K17, fabricado pela Motron e seu acionamento é realizado a partir de um inversor de frequência, modelo CFW-11, produzido pela WEG. Nas próximas subseções cada componente será melhor detalhado.

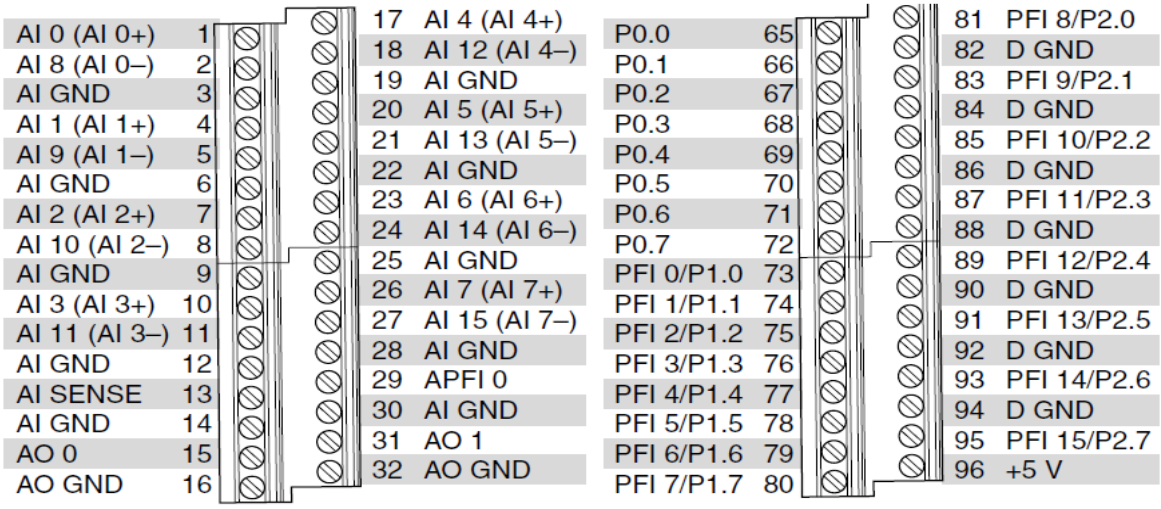
Figura 6 - Protótipo construído para realizar a aquisição de dados.

Fonte: Cardoso *et al.*, 2020.

3.1 PLACA DE AQUISIÇÃO DE DADOS

Para realizar a medição das tensões e correntes de entrada do motor, fez-se uso da placa de aquisição modelo NI USB-6351, de padrão industrial, fabricada pela National Instruments. O dispositivo é dotado de 16 entradas analógicas individuais que podem ser ligadas em pares no modo diferencial. Na Figura 7 apresenta-se um esquema dos bornes de ligação da placa e na Tabela 1 identifica-se qual borne é responsável pela coleta dos dados mais relevantes.

Figura 7 - Bornes de ligação da placa de aquisição de dados.



Fonte: *Datasheet* do dispositivo. Disponível em https://www.ni.com/pdf/manuals/374591d.pdf. Acesso em 22 de outubro de 2020.

Tabela 1 - Correspondência entre borne da placa de aquisição e sinal captado.

CANAL	BORNE	SINAL CAPTADO
AI 0	01	Tensão V_1
AI GND	03	Referência V_1
AI 2	07	Tensão V_2
AI GND	06	Referência V_2
AI 4	17	Tensão V_3
AI GND	12	Referência V_3
AI 1	04	Corrente I_1
AI GND	14	Referência I_1 e I_2
AI 3	10	Corrente I_2

Fonte: O Autor.

3.2 SENSORES DE CORRENTE

Para realizar a medição das correntes de entrada, utilizou-se dois sensores de corrente modelo ACS712. Considerando que o motor compõe um sistema trifásico equilibrado, sabe-se que, conforme afirma Oliveira *et al*. (1996), a soma vetorial das correntes nas três fases é igual a zero, escrito matematicamente em (1).

$$\mathbf{I_1} + \mathbf{I_2} + \mathbf{I_3} = \mathbf{0} \tag{1}$$

Por este motivo, somente dois sensores foram utilizados, ficando o cálculo da terceira corrente a encargo do software desenvolvido. As grandezas em (1) são vetoriais e por isto estão escritas em negrito.

3.3 PLACAS DE CIRCUITO IMPRESSO

Para a execução deste trabalho, não foi necessário construir qualquer tipo de placa de circuito impresso (PCI). Porém, para um entendimento mínimo do funcionamento do protótipo, é importante citar a função das placas desenvolvidas anteriormente nos trabalhos de Castro (2017) e Neiva (2018), reaproveitadas aqui.

De acordo com informações do *datasheet* da placa NI USB-6351, a amplitude máxima suportada pelos terminais do dispositivo é de ± 10 V. Como solução, utilizou-se um banco de transformadores monofásicos ligados em Δ-Y, com as fases do motor conectadas no primário. Dessa forma, a tensão foi reduzida de 220 V_{RMS} para aproximadamente 8,5 V_{RMS}. Como medida adicional de segurança, o secundário do transformador foi conectado a uma das PCI que contém um divisor de tensão junto a um filtro passivo de primeira ordem, antes de ser levado aos terminais da placa. A função do filtro é bloquear/filtrar quaisquer ruídos que possam influenciar os sinais recebidos pela placa, alterando seus valores reais e podendo levar a um resultado errôneo.

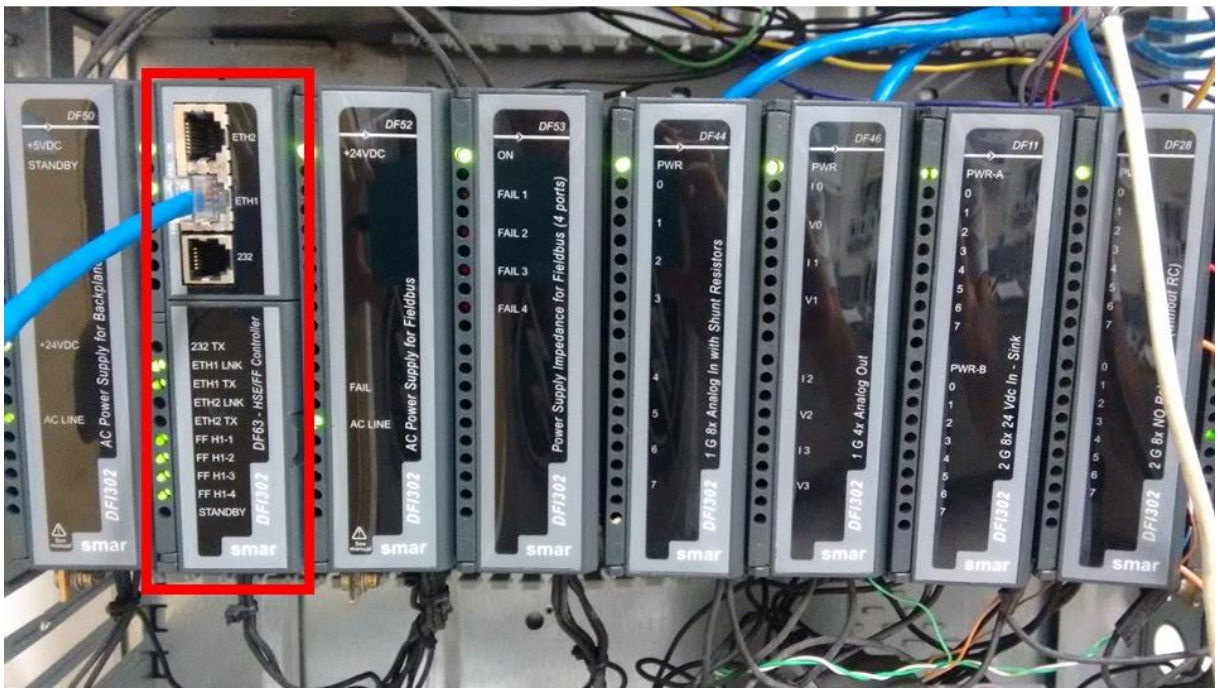
Outra das placas de circuito impresso contém um condicionador de corrente para cada fase para adequar os valores lidos, filtros para a retirada de offset característico do modelo ACS712 e amplificadores operacionais para aumentar o ganho do sinal de corrente.

3.4 CLP E TRANSMISSOR DE TEMPERATURA

A integração da tecnologia Fieldbus ao projeto ocorreu com a medição da temperatura do enrolamento do estator durante o funcionamento do motor. Para a coleta do sinal de temperatura, optou-se por um transdutor do tipo termistor NTC200, acoplado a um transmissor de temperatura, modelo TT302, fabricado pela SMAR. O transmissor é alimentado pela rede H1 e os sinais medidos são enviados para um Controlador Lógico Programável (CLP), modelo DF63, da mesma fabricante. Para a correta calibração do sensor de temperatura, seguiu-se os procedimentos indicados no manual de instruções do TT302, fornecido por SMAR (2006).

O CLP faz parte de um conjunto de módulos e cartões com aplicação voltada à automação de processos industriais, sendo alimentado por uma fonte de 24 V presente no rack. Para os fins deste trabalho, somente o módulo controlador DF63 foi necessário. Na Figura 8 pode-se visualizar o rack, com o CLP em destaque.

Figura 8 - Localização do CLP no rack modular.



Fonte: O Autor.

Todos os sinais de temperatura medidos, além de serem exibidos no transmissor de temperatura, também são enviados para um servidor próprio, identificado como “Smar.hseoleserver.0”. Por simplificação, neste trabalho, a menos que indicado o contrário, servidor fará referência ao Smar.hseoleserver.0.

3.5 MICROSOFT VISUAL STUDIO

Para a programação do supervisório, necessitou-se escolher um ambiente de desenvolvimento integrado, mais conhecidos como IDE. A escolha do ambiente, além da linguagem de programação utilizada, levou em consideração os aspectos referentes ao custo – buscando plataformas gratuitas que oferecessem um bom suporte – e ao nível da linguagem, optando pelas de alto nível e orientada a objetos. O escolhido foi o Visual Studio, criado pela Microsoft; nele, encontram-se uma vastidão de ferramentas que tornam possíveis a edição de código e a criação de interfaces gráficas com aparência sofisticada (MICROSOFT, 2019). A linguagem de programação escolhida foi a C# (lê-se C Sharpe) e que possui suporte dentro do ambiente de desenvolvimento.

Dentre as muitas vantagens oferecidas pela linguagem C# e pelo Visual Studio está a possibilidade de usar *threads* e *tasks*. Tratam-se de processos que são executados simultaneamente, agilizando a execução das tarefas. Sem o uso dos *threads*, cada tarefa só poderia ser realizada quando a anterior estivesse concluída, levando a possíveis sobrecargas de dados no processador e consequentemente um péssimo desempenho do computador, com excessivos travamentos.

3.6 SOFTWARE ADVOSOL

Para acessar os serviços e vantagens que o uso do padrão OPC oferece, necessita-se de softwares específicos, em sua grande maioria pertencente a terceiros. Este é o caso do Advosol, pertencente à empresa homônima Advosol Inc. e um dos mais recomendados pela própria OPC Foundation. Trata-se de uma empresa especializada em OPC, ofertando produtos, programas e conjuntos de ferramentas para trabalhar com o padrão (ADVOSOL, 2020). Mesmo sendo a fonte de receitas da empresa, felizmente há disponível em seu site uma versão gratuita para trabalhar com OPC DA. Tal versão, mesmo limitada, atendeu plenamente aos propósitos fornecendo a possibilidade de leitura e escrita em servidores, além de bibliotecas e *assembles* para trabalhar diretamente no Visual Studio.

3.7 SOFTWARE SYSCON

Para trabalhar com o transmissor de temperatura e com o CLP, suas configurações precisaram ser ajustadas seguindo os protocolos da SMAR, a fabricante. O System302 engloba todo o sistema de automação industrial da empresa. Já o Syscon trata-se de uma ferramenta responsável por configurar, supervisionar e operar os diversos equipamentos oferecidos (SMAR, 2012). É através do Syscon que tanto controladores, como o DF63, quanto dispositivos de campo, como o TT302, podem ser configurados, parametrizados em relação ao sensor que estão utilizando, ter novos blocos funcionais criados em sua programação, dentre outras funcionalidades.

3.8 MÉTODOS

Uma grande parte do protótipo foi aproveitada dos trabalhos desenvolvidos por Castro (2017) e Neiva (2018), como por exemplo as PCI citadas anteriormente. Entretanto, algumas mudanças precisaram ser realizadas, começando pela alocação de todos os elementos em um quadro metálico. Novos dispositivos foram adicionados ao protótipo, como o transmissor de temperatura, o cabo de rede necessário para a conexão física do CLP ao computador, assim como os cabos para ligação do TT302 à rede H1. Tais mudanças foram executadas ao longo de alguns meses e poderiam ser melhor detalhadas, porém, como o foco deste trabalho é o desenvolvimento do software supervisório, a descrição da parte física do projeto se aterá a este sucinto parágrafo.

Como dito anteriormente, toda a programação ocorreu dentro do ambiente de desenvolvimento Visual Studio. Ao longo de todo o código, o uso de *threads* e *tasks* contribuíram com uma menor repetibilidade de linhas e com um comportamento mais ágil do software durante sua execução. O supervisório é a parte mais importante do projeto, pois é através dele que as variáveis são integradas e exibidas ao usuário. Ressalta-se que a placa de aquisição de dados realiza a medição apenas dos sinais de tensão e corrente de alimentação do motor, de modo que as outras variáveis são obtidas através de cálculos via software. Na Figura 9 encontra-se uma parte extraída das linhas de código onde pode-se visualizar os comandos para executar os cálculos.

Figura 9 - Trecho do código utilizado para o cálculo das tensões e correntes de entrada do motor.

```
V_12 = (Convert.ToSingle(v_1[i]) * (220 / 6) - Convert.ToSingle(v_2[i]) * (220 / 6)) * 2 / 3 * 17400 / 10000 * 215 / 225;  // V1 - V2
V_23 = (Convert.ToSingle(v_2[i]) * (220 / 6) - Convert.ToSingle(v_3[i]) * (220 / 6)) * 2 / 3 * 17400 / 10000 * 215 / 225;  // V2 - V3
V_31 = (Convert.ToSingle(v_3[i]) * (220 / 6) - Convert.ToSingle(v_1[i]) * (220 / 6)) * 2 / 3 * 17400 / 10000 * 218 / 225;  // V3 - V1
I_1 = ((Convert.ToSingle((i_1[i]) - Convert.ToDouble(nud_Ajuste_i1.Value))) * (295 / 100)); // I1
I_2 = ((Convert.ToSingle((i_2[i]) - Convert.ToDouble(nud_Ajuste_i2.Value))) * (295 / 100)); // I2
I_3 = -(I_1 + I_2);      // I3 Calculada
```

Fonte: O Autor.

Como as tensões de entrada são calculadas entre fase e terra e as tensões mostradas são entre fases, utilizou-se (2) para o cálculo.

$$\mathbf{V_{12} = V_1 - V_2} \tag{2}$$

As demais constantes vistas na imagem possuem o papel de adequar, de forma proporcional, os pequenos sinais medidos aos seus valores equivalentes reais. O mesmo vale para as correntes, onde a variável subtraída diz respeito ao nível de offset do sensor.

Dentre os dados de placa do motor não se encontrava o fator de potência (representado neste trabalho por cos θ), porém ensaios anteriores foram o suficiente para determinar seu valor. Esta informação foi de crucial importância para que o código pudesse calcular as potências ativas monofásicas (P1, P2 e P3) e as potências reativa e aparente trifásicas, de acordo com Alexander e Sadiku (2013), expostas nas equações de (3) a (5), respectivamente. A potência ativa trifásica foi obtida a partir da soma das três potências monofásicas.

$$P = V_{RMS} * I_{RMS} * \cos(\theta) \tag{3}$$

$$Q = 3 * V_{RMS} * I_{RMS} * \text{sen}(\theta) \tag{4}$$

$$S = 3 * V_{RMS} * I_{RMS} \tag{5}$$

Até o momento foi explicado como ocorreu a construção do protótipo e a programação do supervisório para que a aquisição de dados a partir da placa funcionasse corretamente, restando apenas o terceiro grande elemento: o acesso ao servidor a partir do OPC Server. Como já fora explicado, o OPC Server atua como um conversor de protocolos. Ele pega informações presentes no servidor, as converte para seu próprio padrão e as entrega para o demandante, onde há um programa capaz de interpretar corretamente este novo padrão e permite manipulá-lo para que a informação faça sentido para o usuário. Esta é a descrição do procedimento de leitura do servidor; o processo de escrita ocorre da mesma maneira, porém com o fluxo de dados no sentido contrário.

O programa utilizado para transformar o computador hospedeiro em um OPC Client foi o Advosol. Dentre todos os itens oferecidos na versão gratuita, os mais interessantes foram as bibliotecas e *assembles* compatíveis com o Visual Studio. Tais bibliotecas serviram de base para a implementação do padrão OPC no supervisório. Na Figura 10 mostra-se um trecho do código utilizado para a programação de tais funcionalidades.

Figura 10 - Trecho de código utilizado na programação de funcionalidades do OPC Server.

```
private void btn_Conectar_Click(object sender, EventArgs e)
{
    btn_Ler.Enabled = true;
    int rtc;

    if (Srv == null)
    {
        try
        {
            txt_Status.Text = "Conectando ao servidor OPC...";
            this.Update();
            Srv = new OpcServer();
            rtc = Srv.Connect("Smar.hseoleserver.0");
            if (HRESULTS.Failed(rtc))
            {
                txt_Status.Text = "Erro" + rtc.ToString() + "para conectar ao servidor";
                Srv = null;
                return;
            }
        }
        catch (Exception ex)
        {
            txt_Status.Text = ex.Message;
            Srv = null;
            return;
        }

        SERVERSTATUS stat;
        rtc = Srv.GetStatus(out stat);
        if (HRESULTS.Succeeded(rtc))
        {
            txt_Status.Text = "Smar.hseoleserver.0";
        }
```

Fonte: O Autor.

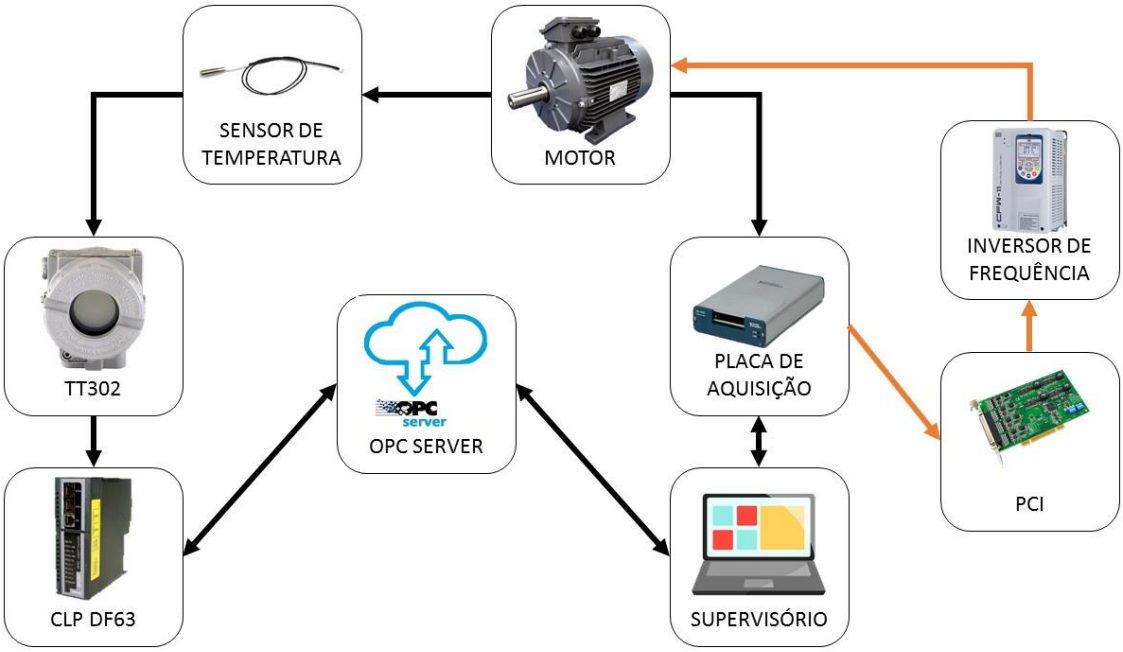
O restante do código utilizado na programação do acesso ao OPC pode ser conferido no Apêndice A. O OPC Server foi implementado com duas finalidades:

- Permitir que o supervisório lesse os sinais de temperatura presentes no servidor;
- Permitir que o supervisório escrevesse, no servidor, as médias dos valores de corrente e tensão e a potência ativa do motor.

Para alcançar tais metas, o Syscon foi utilizado. Tal programa garantiu o acesso às configurações internas do módulo DF63, permitindo a inserção de um bloco funcional que possibilitou a escrita das diversas variáveis no CLP e, consequentemente, tornou-as disponíveis no servidor. Para validar se a escrita foi executada de forma correta, programou-se o supervisório para que ele passasse a ler os dados que foram escritos.

Tendo finalmente descrito todo o processo de construção do protótipo e programação do supervisório, expõe-se um esquemático na Figura 11 para sintetizar tudo o que foi explicado. Os principais componentes são apresentados e as setas que os interligam indicam o fluxo das informações.

Figura 11 - Esquema ilustrativo da composição do sistema global e do fluxo de informações.



Fonte: O Autor.

O motor é a fonte dos dados. A seta preta que vai para a placa de aquisição indica o sentido do fluxo dos sinais de corrente e tensão; esses sinais, coletados pela placa, são transferidos para o supervisório. A outra seta preta que sai do motor (e vai até o sensor de temperatura), indica o fluxo dos sinais de temperatura; o sinal é captado pelo sensor e então passa para o TT302, o qual transmite os dados para o CLP/servidor. As setas duplas ligadas ao OPC Server indicam que:

- Os dados de temperatura, armazenados no servidor, podem ser lidos pelo OPC Server e repassados para o supervisório (CLP/servidor – OPC Server – Supervisório);
- Os dados de tensão e corrente (que são repassados para o supervisório via placa de aquisição), podem ser lidos pelo OPC Server e escritos no servidor (Supervisório – OPC Server – CLP/servidor).

As setas laranjas indicam que o motor pode ser controlado pelo supervisório. O software envia as ordens para a placa de aquisição; a placa transmite a mensagem para uma das PCI através de suas portas digitais e a placa de circuito impresso repassa o sinal para o inversor de frequência. Este último age no motor, podendo controlar sua velocidade, torque e sentido de rotação.

Diversos ensaios foram realizados com o desejo de auferir o desempenho do sistema como um todo. O termistor foi inserido no enrolamento do estator e acoplado ao transmissor de temperatura, que por sua vez foi conectado à rede H1. Conectou-se a placa de aquisição de dados no computador através de um cabo USB; a ligação do computador ao CLP se deu via rede HSE, utilizando um cabo de rede. Alocou-se os instrumentos de medição em suas posições como forma de validação da leitura exibida na tela do supervisório. A análise dos resultados obtidos é apresentada no próximo capítulo.

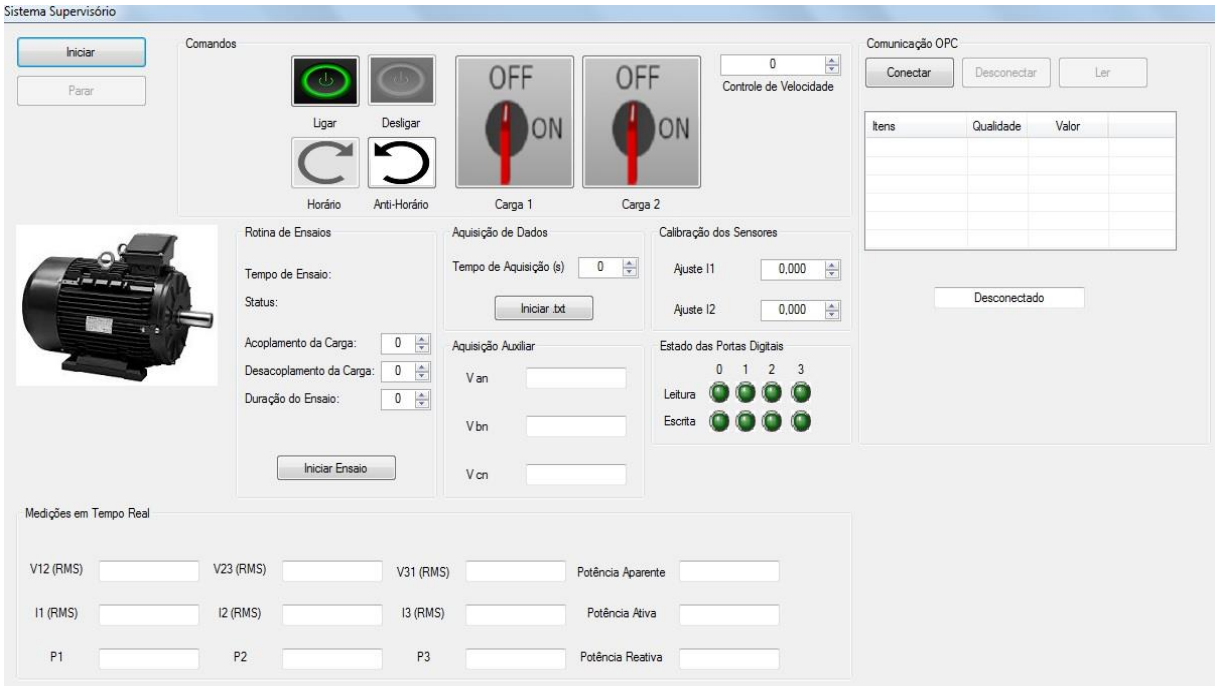
4 RESULTADOS E DISCUSSÃO

Para analisar o comportamento do programa desenvolvido e verificar, na prática, se ele iria funcionar da forma adequada e esperada, realizou-se uma sequência de testes. Foram realizados, ao todo, seis diferentes testes divididos em dois dias. Por simplificação e para evitar repetibilidade, nesta seção será realizada a análise de apenas um dos ensaios.

A realização dos ensaios buscou estudar o supervisório nas tarefas de receber e de exibir corretamente os sinais de tensões e correntes vindos da placa de aquisição, de realizar os cálculos corretos das potências e, com especial atenção, de adquirir e compartilhar dados via OPC Server. Desejava-se saber se o OPC Server seria capaz de ler as informações de temperatura presentes no servidor, assim como de escrever as médias das grandezas de interesse nele.

A Figura 12 mostra a tela principal do programa, onde percebe-se que todas as grandezas de interesse (tensão e corrente em cada fase, potência ativa por fase e total, potência reativa e potência aparente) estão à disposição do usuário/operador através da interface gráfica criada. No canto superior direito encontra-se a área reservada para variáveis relacionadas com a tecnologia OPC.

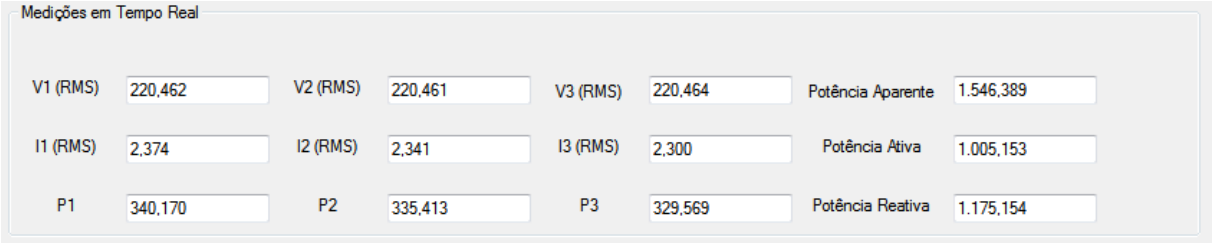
Figura 12 - Tela principal do supervisório desenvolvido.



Fonte: Cardoso *et al.*, 2020.

Após o motor ser ligado e o botão “Iniciar” ser pressionado no software, os valores de tensão e corrente em cada fase do motor, assim como das potências, passaram a ser exibidos em tela. A frequência de amostragem escolhida para a placa de aquisição foi de 10 kHz, o que significa que o código desenvolvido foi capaz de calcular, a cada segundo, os valores RMS de tensão e corrente dentre um intervalo de 10 mil amostras e a partir daí calcular todas as potências. A cada atualização da tela, realizadas em intervalos de um segundo, todos os cálculos eram refeitos e os valores exibidos mudavam com base nas novas contas. Na Figura 13, mostra-se em destaque a área reservada para a exposição das grandezas de interesse e os valores exibidos na tela do supervisório.

Figura 13 - Exibição das medições realizadas a partir da placa de aquisição.



Fonte: Cardoso *et al.*, 2020.

A validação das grandezas que eram apresentadas em tela foi realizada através de medições das correntes de entrada e tensões de alimentação do motor, com o auxílio de instrumentos adequados. A leitura obtida do voltímetro indicou tensão de 220 V_{RMS} nas três fases, algo já esperado a partir da hipótese de se tratar de uma máquina equilibrada; considerou-se que a diferença de 0,2% entre a leitura e a maior tensão mostrada na Figura 13 (V_3) ocorreu dada a resolução do aparelho utilizado e a consequente ausência de casas decimais. O amperímetro indicou o valor de aproximadamente 2,4 A_{RMS} por fase do motor, implicando na diferença de cerca de 5% entre a referência (amperímetro) e o maior valor de corrente exibido em tela, visto na Figura 13 (I_3); acredita-se que a discrepância possa ter sido causada por dois fatores, simultaneamente ou não: má calibração ou precisão limitada dos sensores de correntes. Além disso, como I_3 depende diretamente das outras duas correntes, erros em I_1 e I_2 podem ter um grande reflexo no valor da terceira corrente. Na Tabela 2 estão os dados exibidos na Figura 13 e os valores medidos pelos instrumentos, para melhor comparação.

Tabela 2 - Comparação entre os valores medidos por instrumentos e os exibidos no supervisório.

Grandeza	Valor Lido	Valor de Referência	Diferença (%)
V_1	220,462 V	220 V	0,21
V_2	220,461 V	220 V	0,21
V_3	220,464 V	220 V	0,21
I_1	2,374 A	2,417 A	1,78
I_2	2,341 A	2,411 A	2,90
I_3	2,300 A	2,409 A	4,52

Fonte: O Autor.

Como informado anteriormente, o fator de potência do motor se fazia conhecido a partir de ensaios anteriores, tendo o valor aproximado de 0,65. Vale ressaltar que o fator de potência não constava dentre os dados de placa do motor, assim como o modelo não pôde ser localizado no site do fabricante. Porém, motores de modelos semelhantes da mesma marca foram utilizados como fonte de comparação, levando à conclusão de que mesmo sendo um valor bem abaixo do recomendado, se faz condizente com as características de um motor para fins didáticos. Através das equações, do fator de potência encontrado e dos dados de tensões e correntes retirados da Figura 13, pode-se facilmente calcular todas as potências.

Na Figura 14, consegue-se ver em detalhes os campos reservados para a obtenção dos dados presentes no servidor. Enquanto o botão “Conectar” não fosse pressionado, não haveria conexão estabelecida entre o servidor e o supervisório. Caso ocorresse alguma falha e a conexão não fosse bem sucedida, a caixa de texto na parte inferior da mesma imagem apresentaria uma mensagem de erro. No ensaio não houve falhas na comunicação e após pressionar o botão “Ler”, a exibição dos dados começou e o nome do servidor “Smar.hseoleserver.0”, hospedeiro das informações, surgiu na mesma caixa de texto onde apareceria a mensagem de erro citada anteriormente.

Figura 14 - Leitura das variáveis presentes no servidor.

Fonte: Cardoso *et al.*, 2020.

Um primeiro olhar sobre as informações presentes no servidor, ilustradas na Figura 14, não possibilita a identificação de a qual variável cada item se refere. Isso ocorre, pois ao enviar cada variável para o servidor, o CLP realiza a identificação através da atribuição de *tags* ao invés dos nomes habituais. A Tabela 3 apresenta a qual grandeza cada *tag* representa, assim como seus valores.

Tabela 3 - Correspondência entre as *tags* e as grandezas medidas.

Tag	**Grandeza Correspondente**	**Valor Medido**
Motor_TT302_AI.OUT.VALUE	Temperatura do estator	32,76887 °C
Motor_DF63_CONST.CT_VAL_1	Média das tensões de entrada	219,8 V
Motor_DF63_CONST.CT_VAL_2	Média das correntes de entrada	2,332 A
Motor_DF63_CONST.CT_VAL_3	Potência ativa trifásica	999,337 W

Fonte: O Autor.

Buscando correspondência entre a Tabela 3 e a Figura 13, destaca-se que as variáveis tensão e corrente representam, na verdade, as médias (RMS) de V_1, V_2 e V_3 e de I_1, I_2 e I_3, respectivamente; a potência ativa é a mesma da imagem – soma de P_1, P_2 e P_3 ou simplesmente a potência ativa total. A presença destas três grandezas no servidor, tomadas pela placa de

aquisição de dados, mostra que a escrita de dados realizada pelo supervisório funcionou da maneira esperada.

Também pode-se afirmar que a leitura das informações presentes no servidor ocorreu de maneira correta, pois tanto os dados escritos quanto a temperatura puderam ser lidos pelo supervisório. Na Figura 15 exibe-se o transmissor de temperatura TT302 utilizado; tal dispositivo é dotado de um display que mostra, em tempo real, a temperatura medida pelo termistor acoplado a ele. Comparando a leitura do transmissor de temperatura com os valores exibidos na tela da Figura 14 – ou reescritos na Tabela 3 – percebe-se uma mínima diferença de aproximadamente 0,2%, causada principalmente pelo *delay* de atualização dos dados no programa – que ocorrem a cada segundo.

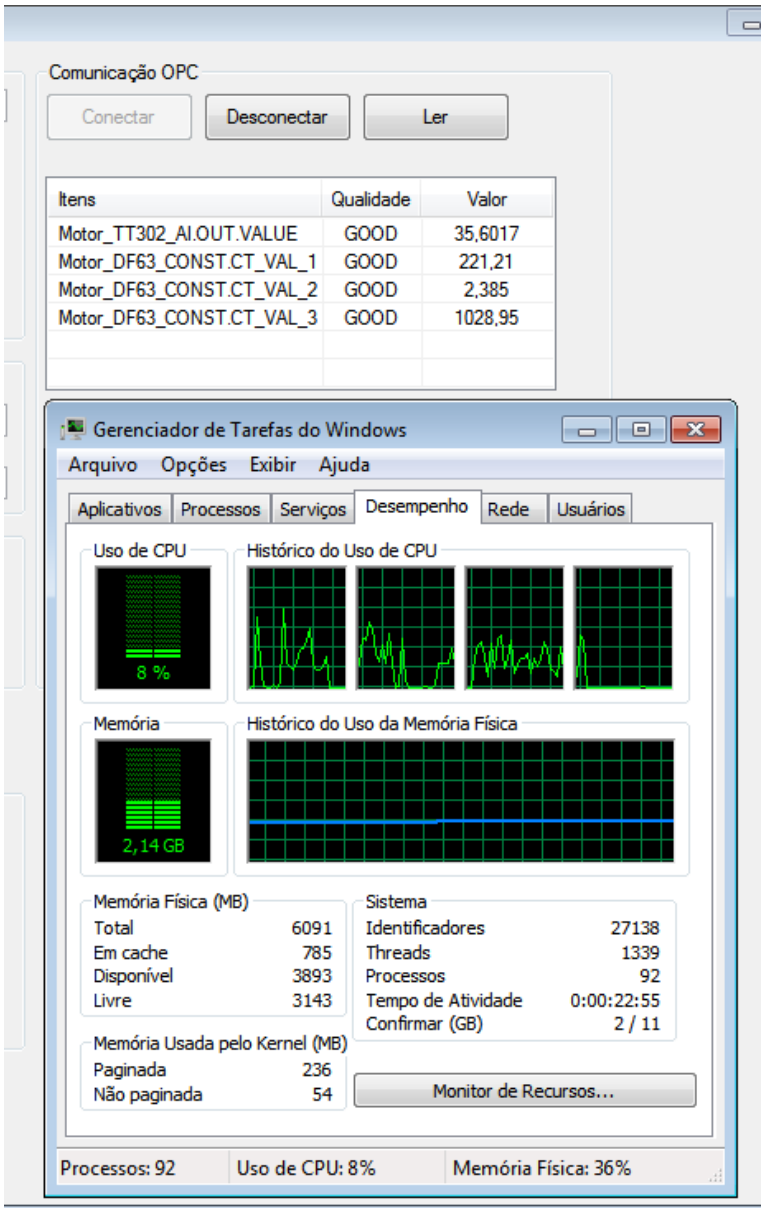
Figura 15 - Display do transmissor de temperatura TT302.

Fonte: Cardoso *et al*., 2020.

Durante o processo de programação do supervisório surgiu a preocupação relacionada ao seu custo de execução. Não seria viável, do ponto de vista da aplicabilidade, desenvolver um programa capaz de receber corretamente os dados vindos da placa de aquisição e se comunicar com um servidor, usufruindo dos processos de leitura e escrita, e possuir um alto gasto da capacidade de processamento do computador hospedeiro. A Figura 16 apresenta o resultado obtido. Pode-se notar que o valor de temperatura presente na Figura 16 é diferente daquele apresentado na análise anterior (Figura 14). Isso ocorreu, pois a tela do supervisório com o

gerenciador de tarefas, apresentada na Figura 16, foi “fotografada” após a captura da tela do supervisório presente na Figura 14. A temperatura do motor aumentou no intervalo de tempo entre as duas capturas de tela.

Figura 16 - Desempenho do computador durante a execução do supervisório.



Fonte: O Autor.

Através do Gerenciador de Tarefas, ferramenta nativa do sistema operacional Windows, constatou-se qual o uso de CPU da máquina durante a execução do supervisório. Como percebe-se, apenas 8% do processador foi utilizado[3], não comprometendo o funcionamento do

[3] Nenhum outro programa estava em execução no momento dos ensaios.

computador. Tal resultado foi possível devido ao uso de *threads* e *tasks* durante a programação, além de inserir a menor quantidade possível de imagens na interface gráfica.

5 CONCLUSÃO

Ao longo desta monografia, discorreu-se principalmente sobre os materiais utilizados e os procedimentos seguidos para o desenvolvimento de um supervisório, buscando integrar sinais de tensão e corrente – coletados de um motor a partir de uma placa de aquisição de dados USB – e dados de temperatura – medidos no mesmo motor, mas transmitidos pela tecnologia Fieldbus, amplamente difundida na indústria.

Os resultados mostraram que o supervisório conseguiu receber os dados da placa, interpretá-los, fazer todos os cálculos exigidos e exibir o valor correto de cada grandeza de interesse; a tarefa de ler os dados de temperatura e escrever as informações de corrente e tensão no servidor também foram executadas. Todas as informações foram atualizadas em tela no intervalo de um segundo, tempo considerado satisfatório para os propósitos ensejados. A análise deste compilado de observações comprova que o supervisório desenvolvido apresentou o funcionamento e o OPC Server cumpriu com o objetivo de liberar o acesso ao servidor. Além disso, o processador do computador não ficou sobrecarregado, demonstrando um desempenho excelente por parte do software.

Durante a execução do programa, qualquer computador ou módulo controlador com acesso ao sistema cliente/servidor OPC e que esteja conectado ao servidor utilizado (Smar.hseoleserver.0), poderá ter acesso aos dados de temperatura e também aos de corrente, tensão e potência ativa do motor, facilitando possíveis ações de controle a serem tomadas.

Sugere-se que, como trabalhos futuros, faça-se a portabilidade do OPC Server utilizado, OPC DA, para a versão mais nova, OPC UA. Trata-se de uma tecnologia que vem ganhando força no mercado e muitos pesquisadores acreditam que seja a peça chave para resolver os problemas presentes na chamada “Internet das Coisas”.

REFERÊNCIAS

ADVOSOL. **Advosol specializes in OPC solutions on the .NET platform**. Disponível em: http://www.advosol.com/topic/about. Acesso em 20 fev 2020.

ALEXANDER, C. K.; SADIKU, M. N. O. **Fundamentos de circuitos elétricos**. 5. ed. Porto Alegre: AMGH, 2013.

AZEVEDO, D. L. D. *et al*. Análise comparativa de protocolos Fieldbus (Wirelesshart, Profibus e Foundation Fieldbus). *In*: 19º Seminário de Automação e TI Industrial, 2015, Rio de Janeiro. **Anais do seminário de automação & TI**. Rio de Janeiro: 2015. p. 81-89.

BELCHIOR, Fernando Nunes. **Apostila de medidas elétricas**. Itajubá: UNIFEI, 2014.

BRUCKNER, D. *et al*. An introduction to OPC UA TSN for industrial communication systems. **Proceedings of the IEEE**, v. 107, n. 6, p. 1121-1131, 2019.

CARDOSO, L. B. *et al*. Desenvolvimento de supervisório visando a integração entre variáveis Fieldbus e de sistemas de aquisição de dados. **Brazilian Journal of Development**, v. 6, n. 9, p. 70684-70694, 2020.

CARVALHO, V. A. D.; TEIXEIRA, G. F. **Programação orientada a objetos**: curso técnico de informática. Colatina: IFES, 2012.

CASTRO, Pedro Henrique Gonçalves Rigueira Pinheiro. **Sistema de aquisição de dados de tensões e de correntes trifásicas**. 2017. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Elétrica) – Universidade Federal de Viçosa, Viçosa, 2017.

CHEN, J.; TAI, K.; CHEN, G. Application of programmable logic controller to build-up an intelligent industry 4.0 platform. **Procedia CIRP**, v. 63, p. 150-155, 2017.

DONGJIANG, L.; RUIQI, S. Implement of communication between configuration software and OPC server based on Modbus/TCP. In: **IEEE 2011 10th International Conference on Electronic Measurement & Instruments**. IEEE, 2011. p. 218-221.

ECKHARDT, A.; MÜLLER, S.; LEURS, L. An evaluation of the applicability of OPC UA Publish Subscribe on factory automation use cases. *In*: **2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)**. IEEE, 2018. p. 1071-1074.

FIELDCOMM GROUP. **Driving the industrial digital transformation for over 20 years**. Disponível em: https://www.fieldcommgroup.org/technologies/foundation-fieldbus. Acesso em 25 fev 2020.

HELFRICK, A. D.; COOPER, W. D. **Instrumentação eletrônica moderna e técnicas de medição**. Rio de Janeiro: Prentice Hall do Brasil, 1994.

MEASUREMENT COMPUTING. **Data acquisition handbook**: a reference for DAQ and analog & digital signal conditioning. 3. ed. 2012.

MICROSOFT. **Bem-vindo ao IDE do Visual Studio**. Disponível em: https://docs.microsoft.com/pt-br/visualstudio/get-started/visual-studio-ide?view=vs-2017&redirectedfrom=MSDN. Acesso em 23 fev 2020.

NATIONAL INSTRUMENTS. **NI 6351**: device specifications. 2016.

NEIVA, Diego Romie. **Supervisório para o acionamento de motor de indução trifásico utilizando inversor de frequência**. 2018. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Elétrica) – Universidade Federal de Viçosa, Viçosa, 2018.

NICULESCU, F.; SAVESCU, A.; MITRU, A. Transmitting data over the network using an OPC server. In: **MATEC Web of Conferences**. EDP Sciences, 2018. p. 03002.

OLIVEIRA, C. C. B. D. *et al*. **Introdução a sistemas elétricos de potência**: componentes simétricas. 2. ed. São Paulo: Edgard Blücher, 1996.

OPC FOUNDATION. **What is OPC?** Disponível em: https://opcfoundation.org/about/what-is-opc/. Acesso em 24 fev 2020.

QING, L.; YONGSHENG, Q. Development of OPC server in a remote industrial control system. In: **2015 12th IEEE International Conference on Electronic Measurement & Instruments (ICEMI)**. IEEE, 2015. p. 552-557.

RICARTE, Ivan Luiz Marques. **Programação orientada a objetos**: uma abordagem com Java. Campinas: UNICAMP, 2001.

SCHWARZ, M. H.; BÖRCSÖK, J. A survey on OPC and OPC-UA: About the standard, developments and investigations. In: **2013 XXIV International Conference on Information, Communication and Automation Technologies (ICAT)**. IEEE, 2013. p. 1-6.

SMAR. **Transmissor de temperatura Fieldbus**: manual de instruções, operação e manutenção. 2006.

SMAR. **Syscon - configurador de sistema**: manual de operação e instalação. 2012.

TANENBAUM, A. S.; WETHERALL, D. J. **Redes de computadores**. 5. ed. Pearson, 2011.

TEIXEIRA, André Lobo. **Desenvolvimento de um cliente OPC e uma WEB Application para disponibilizar informações de plantas industriais na internet**. 2017. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Elétrica) – Universidade Federal de Viçosa, Viçosa, 2017.

APÊNDICE

APÊNDICE A – Trecho de código utilizado para a programação das funcionalidades do OPC

```
private void btn_IniciarEnsaio_Click(object sender, EventArgs e)
        {
            contador = 0;
            tmr_Ensaio.Enabled = true;
            flag = true; // Inicia a aquisição de dados
        }

        private void btn_Conectar_Click(object sender, EventArgs e)
        {
            btn_Ler.Enabled = true;
            int rtc;

            if (Srv == null)
            {
                try
                {
                    txt_Status.Text = "Conectando ao servidor OPC...";
                    this.Update();
                    Srv = new OpcServer();
                    rtc = Srv.Connect("Smar.hseoleserver.0");
                    if (HRESULTS.Failed(rtc))
                    {
                        txt_Status.Text = "Erro" + rtc.ToString() + "para conectar ao
servidor";
                        Srv = null;
                        return;
                    }
                }
                catch (Exception ex)
                {
                    txt_Status.Text = ex.Message;
                    Srv = null;
                    return;
                }

                SERVERSTATUS stat;
                rtc = Srv.GetStatus(out stat);
                if (HRESULTS.Succeeded(rtc))
                {
                    txt_Status.Text = "Smar.hseoleserver.0";
                }
                else
                {
                    txt_Status.Text = "Erro" + rtc.ToString() + "ao obter o Status";
                    Srv = null;
                    return;
                }

                try
                {
                    ReadWriteGroup = new SyncIOGroup(Srv);
                    DataChangeEventHandler dch = new
DataChangeEventHandler(this.DataChangeHandler);
                    AsyncRefrGroup = new RefreshGroup(Srv, dch);
```

```
                btn_Conectar.Enabled = false;
                btn_Desconectar.Enabled = true;
            }
            catch (OPCException ex)
            {
                txt_Status.Text = ex.Message;
                Srv = null;
                return;
            }

            tmr_OPC.Enabled = true;
        }
    }

    private void btn_Desconectar_Click(object sender, EventArgs e)
    {
        if (Srv != null)
        {
            AsyncRefrGroup.Dispose();
            ReadWriteGroup.Dispose();
            Srv.Disconnect();
            Srv = null;
            txt_Status.Text = "Desconectado";
        }
        btn_Conectar.Enabled = true;
        btn_Desconectar.Enabled = false;
        btn_Ler.Enabled = false;
        lv_Atualizar.Items.Clear();
    }

    public void DataChangeHandler(object sender, DataChangeEventArgs e)
    {
        if (this.InvokeRequired)
        {
            this.BeginInvoke(new DataChangeEventHandler(DataChangeHandler), new
object[] { sender, e });
            return;
        }
        for (int i = 0; i < e.sts.Length; ++i)
        {
            int hnd = e.sts[i].HandleClient;
            object val = e.sts[i].DataValue;
            string qt = AsyncRefrGroup.GetQualityString(e.sts[i].Quality);
            DateTime dt = DateTime.FromFileTime(e.sts[i].TimeStamp);

            OPCDA.NET.ItemDef item = AsyncRefrGroup.FindClientHandle(hnd);
            if (item != null)
            {
                string name = item.OpcIDef.ItemID;
                for (int l = 0; l < lv_Atualizar.Items.Count; ++l)
                {
                    if (lv_Atualizar.Items[l].Text == name)
                    {
                        while (lv_Atualizar.Items[l].SubItems.Count > 1)
                        {
                            lv_Atualizar.Items[l].SubItems.RemoveAt(1);
                        }
                        lv_Atualizar.Items[l].SubItems.Add(qt);
                        lv_Atualizar.Items[l].SubItems.Add(val.ToString());
                    }
                }
```

```
            }
        }
    }

    private void btn_Ler_Click(object sender, EventArgs e)
    {
        int Temperatura;
        int Tensao;
        int Corrente;
        int Potencia;

        Temperatura = AsyncRefrGroup.Add("Motor_TT302_AI.OUT.VALUE");
        if (HRESULTS.Failed(Temperatura))
        {
            lbl_LeituraErro.Text = ReadWriteGroup.GetErrorString(Temperatura);
        }
        else
        {
            lv_Atualizar.Items.Add("Motor_TT302_AI.OUT.VALUE");
        }

        Tensao = AsyncRefrGroup.Add("Motor_DF63_CONST.CT_VAL_1");
        if (HRESULTS.Failed(Tensao))
        {
            lbl_LeituraErro.Text = ReadWriteGroup.GetErrorString(Tensao);
        }
        else
        {
            lv_Atualizar.Items.Add("Motor_DF63_CONST.CT_VAL_1");
        }

        Corrente = AsyncRefrGroup.Add("Motor_DF63_CONST.CT_VAL_2");
        if (HRESULTS.Failed(Corrente))
        {
            lbl_LeituraErro.Text = ReadWriteGroup.GetErrorString(Corrente);
        }
        else
        {
            lv_Atualizar.Items.Add("Motor_DF63_CONST.CT_VAL_2");
        }

        Potencia = AsyncRefrGroup.Add("Motor_DF63_CONST.CT_VAL_3");
        if (HRESULTS.Failed(Potencia))
        {
            lbl_LeituraErro.Text = ReadWriteGroup.GetErrorString(Potencia);
        }
        else
        {
            lv_Atualizar.Items.Add("Motor_DF63_CONST.CT_VAL_3");
        }
    }

    private void tmr_OPC_Tick(object sender, EventArgs e)
    {
        object tensao = ((v1_rms + v2_rms + v3_rms) / 3).ToString("n3");
        int Tensao = ReadWriteGroup.Write("Motor_DF63_CONST.CT_VAL_1", tensao);

        object corrente = ((i1_rms + i2_rms + i3_rms) / 3).ToString("n3");
        int Corrente = ReadWriteGroup.Write("Motor_DF63_CONST.CT_VAL_2",
corrente);

        object potencia = txt_PotenciaAtiva.Text;
```

```
            int Potencia = ReadWriteGroup.Write("Motor_DF63_CONST.CT_VAL_3",
potencia);
        }
```