Controle e Supervisão de Sistema de Automação Utilizando Computação em Nuvem

José Antonio Toledo Júnior
Discente do Curso de Engenharia Mecatrônica
Universidade Federal de São João del-Rei
jatoledojunior@gmail.com

Guilherme Gomes da Silva
Departamento das Engenharias de Telecomunicações e Mecatrônica
Universidade Federal de São João del-Rei
guilhermegomes@ufsj.edu.br

***Resumo* — Os sistemas atuais mais comercializados de controle e supervisão para automação industrial já suportam comunicação TCP/IP para desenvolvimento remoto e tráfego de dados pela Internet. Mesmo assim, os sistemas SCADA são, em grande maioria, *softwares* proprietários que necessitam estar instalados localmente em computadores reservados, o que inviabiliza o uso dos mesmos longe desses locais. Este trabalho vem, então, propor o desenvolvimento de um sistema completo de controle e supervisão de uma planta industrial simplificada e deve ter as seguintes características: o sistema supervisório deve ser acessível de qualquer dispositivo conectado à Internet sem necessidade de instalação de *softwares* específicos; o sistema de controle deve ser de caráter modular e expansível; e a tratativa dos dados, incluindo armazenamento e disponibilização dos mesmos, deve ser feita através de computação em nuvem sem restrições de tempo real.**

***Palavras-chave* — Automação Industrial; Computação em Nuvem; Google App Script; Sistema de Controle; Sistema Supervisório.**

I. INTRODUÇÃO

A automação é a operação de uma máquina ou grupo de máquinas, local ou remotamente, com a mínima interferência do operador humano. Isso significa ter um mecanismo de atuação própria para fazer a ação requerida em tempo determinado ou em resposta a certas condições.

O PLC (*Programmable Logic Controller*) é o coração da automação industrial. Trata-se de um equipamento eletrônico, digital, microprocessado, que pode controlar um processo ou uma máquina e ser programado ou reprogramado rapidamente.

O sistema supervisório, ou SCADA (*Supervisory Control And Data Acquisition*), é um conjunto de *hardware* e *software* que permite ao operador ter acesso a informações de um processo de forma clara e objetiva.

Em aplicações industriais, as informações do SCADA normalmente são providas pelo PLC e devem ser apresentadas em formato padronizado e amigável, permitindo uma eficiente interação com o processo [1]. Um exemplo de arquitetura da automação clássica está apresentada na Figura 1 a seguir.

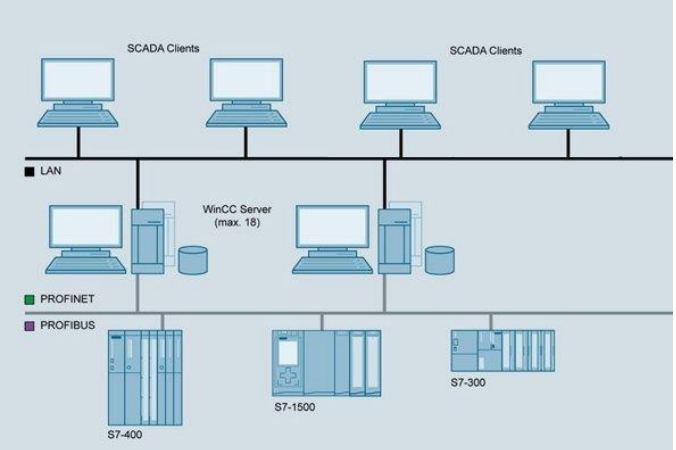

Figura 1 - Exemplo de arquitetura de automação clássica [2].

A Internet é baseada no protocolo TCP/IP de um conjunto de protocolos de comunicação [3]. Este nome é devido a união dos protocolos TCP (*Transmission Control Protocol*) e IP (*Internet Protocol*), localizados respectivamente nas camadas de transporte e de rede do modelo OSI (*Open System Interconnection*). Diante dessa realidade, os sistemas atuais mais comercializados de controle e supervisão para automação industrial já suportam comunicação TCP/IP para desenvolvimento remoto e tráfego de dados pela Internet.

Mesmo assim, os sistemas SCADA são, em grande maioria, *softwares* proprietários que necessitam estar instalados localmente em computadores reservados. Isso inviabiliza o uso do SCADA longe desses locais específicos, diminuindo consequentemente a sua utilização e necessitando de pessoas dedicadas a informar por rádio o *status* das variáveis e do processo para os mantenedores e operadores que estão no chão-de-fábrica realizando alguma atividade.

Este trabalho vem, então, propor o desenvolvimento de um sistema completo de controle e supervisão de uma planta industrial simplificada utilizando tecnologias não tradicionais

que contornem as dificuldades apresentadas. O projeto deve ter as seguintes características:

- O sistema supervisório deve ser acessível de qualquer dispositivo conectado à Internet sem necessidade de instalação de *softwares* específicos;
- O sistema de controle deve ser de caráter modular e expansível;
- A tratativa dos dados, incluindo armazenamento e disponibilização dos mesmos, deve ser feita através de computação em nuvem sem restrições de tempo real.

II. Revisão Bibliográfica

Nesse projeto, o Arduino terá a função de um PLC que adquire e controla os dados, enquanto o Raspberry realiza a troca deles com o banco de dados. O serviço de computação em nuvem utilizado será o GAS (*Google App Script*) e o sistema supervisório será criado através de programação web. Assim, um breve conceito dessas tecnologias é apresentado.

A. Arduino

O Arduino Mega 2560 é mais uma placa da plataforma Arduino que possui recursos bem interessantes para prototipagem e projetos mais elaborados. Baseada no microcontrolador ATmega2560, possui 54 pinos de entradas e saídas digitais, onde 15 destes podem ser utilizados como saídas PWM (*Pulse Width Modulation*), 16 entradas analógicas e 4 portas de comunicação serial [4].

Ele possui sua própria IDE (*Integrated Development Environment*) onde é possível criar os códigos e fazer *upload* para o dispositivo conectado ao computador. A linguagem do Arduino é C/C++ com pequenas modificações, sendo muito popular na comunidade de programadores.

B. Raspberry Pi

O Raspberry Pi tem o tamanho aproximado de um cartão de crédito. A versão mais recente é o Raspberry Pi 3, que tem processador de 1,2GHz e 1GB de memória, Wi-Fi e Bluetooth integrados. A placa permite, assim como o Arduino, ligar sensores, *displays* e outros componentes utilizando o conector GPIO (*General Purpose Input/Output*) de 40 pinos.

Raspbian é o sistema operacional oficial de todos os Raspberrys, porém é possível instalar outros sistemas como Ubuntu MATE, Snappy Ubuntu Core, Windows 10 IoT Core, entre outros. O Noobs é uma das formas mais simples e prática de instalação, pois é um pacote que contém várias versões do sistema operacional e ótima interface [5].

1) Python

Python é uma linguagem de programação com suporte à orientação de objetos, interpretada e interativa. Combina um poder notável com uma sintaxe muito clara. Ele possui módulos, classes, exceções, tipos de dados dinâmicos de alto nível e digitação dinâmica. Existem interfaces para muitas chamadas de sistema e bibliotecas, bem como para vários sistemas de janelas. Novos módulos embutidos são facilmente escritos em C ou C++ (ou outros idiomas, dependendo da implementação escolhida). O Python também pode ser usado como uma linguagem de extensão para aplicativos escritos em outras linguagens que precisam de interfaces de *script* ou automação fáceis de usar [6].

Há duas versões da linguagem: Python 2, que receberá atualizações de segurança até 2020, e Python 3, que introduziu algumas mudanças que quebraram a compatibilidade com a versão anterior e será a única a ser mantida no futuro [7]. O Raspbian já tem o interpretador de Python 2 e 3, além dos ambientes de desenvolvimento nativos IDLE2, IDLE3 e Thonny.

C. Programação em Nuvem

A computação em nuvem surge da necessidade de construir infraestruturas de TI (Tecnologia da Informação) complexas, onde os usuários têm que realizar instalação, configuração e atualização de sistemas de *software*. Em geral, os recursos de computação e *hardware* são propensos a ficarem obsoletos rapidamente e a utilização de plataformas computacionais de terceiros é uma solução inteligente para os usuários lidarem com a infraestrutura de TI. Na computação em nuvem os recursos de TI são fornecidos como um serviço, permitindo que os usuários o acessem sem a necessidade de conhecimento sobre a tecnologia utilizada. Desse modo, os clientes passaram a acessar os serviços sob demanda e independente de localização, o que aumentou a quantidade de serviços disponíveis.

A computação em nuvem é uma evolução dos serviços e produtos de tecnologia da informação sob demanda, também chamada de *Utility Computing*. O objetivo da *Utility Computing* é fornecer componentes básicos como armazenamento, processamento e largura de banda de uma rede como uma "mercadoria" através de provedores especializados com um baixo custo por unidade utilizada. Usuários de serviços baseados em *Utility Computing* não precisam se preocupar com escalabilidade, pois a capacidade de armazenamento fornecida é praticamente infinita. A *Utility Computing* propõe fornecer disponibilidade total, isto é, os usuários podem ler e gravar dados a qualquer tempo, sem nunca serem bloqueados; os tempos de resposta são quase constantes e não dependem do número de usuários simultâneos, do tamanho do banco de dados ou de qualquer parâmetro do sistema. Os usuários não precisam se preocupar com *backups*, pois se os componentes falharem, o provedor é responsável por substituí-los e tornar os dados disponíveis em tempo hábil por meio de réplicas.

Uma razão importante para a construção de novos serviços baseados em *Utility Computing* é que provedores de serviços que utilizam serviços de terceiros pagam apenas pelos recursos que recebem, ou seja, pagam pelo uso. Não são necessários investimentos iniciais em TI e o custo cresce de forma linear e previsível com o uso. Dependendo do modelo do negócio, é possível que o provedor de serviços repasse o custo de armazenagem, computação e de rede para os usuários finais, já que é realizada a contabilização do uso [8].

1) Modelos de Serviço

O ambiente de computação em nuvem é composto de três modelos de serviços:

- *Software* como um Serviço (SaaS): Proporciona sistemas de *software* com propósitos específicos que são disponíveis para os usuários por meio da Internet e acessíveis a partir de vários dispositivos do usuário por meio de uma interface *thin client* como um navegador web. No SaaS, o usuário não administra ou controla a infraestrutura subjacente, incluindo rede, servidores, sistema operacional, armazenamento ou mesmo as características individuais da aplicação, exceto configurações específicas. O GAS é um exemplo de SaaS.
- Plataforma como um Serviço (PaaS): Fornece sistema operacional, linguagens de programação e ambientes de desenvolvimento para as aplicações, auxiliando a implementação de sistemas de *software*. Assim como no SaaS, o usuário não administra ou controla a infraestrutura subjacente, mas tem controle sobre as aplicações implantadas e, possivelmente, as configurações de aplicações hospedadas nesta infraestrutura. O GAE (*Google App Engine*) e Microsoft Azure são exemplos de PaaS.
- Infraestrutura como um Serviço (IaaS): Torna mais fácil e acessível o fornecimento de recursos, tais como servidores, rede, armazenamento e outros recursos de computação fundamentais para construir um ambiente de aplicação sob demanda, que podem incluir sistemas operacionais e aplicativos. Em geral, o usuário não administra ou controla a infraestrutura da nuvem, mas tem controle sobre os sistemas operacionais, armazenamento, aplicativos implantados e, eventualmente, seleciona componentes de rede, tais como *firewalls*. O Amazon EC2 (*Elastic Compute Cloud*) é um exemplo de IaaS [8].

2) Google App Script

Criado e mantido pelo Google, o GAS busca integrar os serviços do Google Apps através de *scripts* desenvolvidos em JavaScript. Essa linguagem apresenta várias ferramentas para programação, contando com serviços do G Suite, que inclui Calendar, Document, Drive, Gmail, Maps, Spreadsheet, entre outros. O principal objetivo desta linguagem de *Cloud Programming* é automatizar tarefas que integrem um ou mais serviços do Google [9].

Um recurso interessante do GAS é desenvolver *scripts* que apresentam um *endpoint* na rede por meio do protocolo HTTP (*Hypertext Transfer Protocol*), isto é, respondem a requisições do tipo *get* e *post*. Com esse mecanismo é possível criar API (*Application Programming Interface*), ou seja, um Web App para envio ou captura de dados. Uma aplicação cliente faz uma chamada HTTP para o GAS que faz o tratamento da requisição e devolve a resposta ao cliente [10].

D. Programação Web

Programação web é dividida em *front-end* e *back-end*. A primeira cuida da parte visual e interativa com o usuário, sendo importante que o desenvolvedor também se preocupe com a experiência do cliente. A segunda é a parte de armazenamento de dados em banco de dados.

O desenvolvedor *front-end* é responsável por "dar vida" à interface e deve ter conhecimento em HTML (*Hypertext Markup Language*), CSS (*Cascading Style Sheets*) e JavaScript. Para CSS e JavaScript há bibliotecas e *frameworks* que alguns profissionais se especializam como Angular e Bootstrap [11].

Bootstrap facilita a vida dos desenvolvedores web a criar sites com tecnologia mobile (responsivo) sem ter que digitar uma linha de CSS. Além disso, o Bootstrap possui uma diversidade de componentes (*plugins*) em JavaScript (jQuery) que auxiliam a implementação: *tootlip*, *menu-dropdown*, *modal*, *carousel*, *slideshow*, entre outros [12].

Quanto ao *back-end* há várias linguagens, como C#, PHP (*Hypertext Preprocessor*), Java, Python e Ruby. Cada uma possui vantagens e desvantagens em relação ao uso no desenvolvimento web, bem como no mercado de trabalho. Para que o aprendizado em *back-end* é preciso ter conhecimento em banco de dados, como MySQL e SQL Server [11].

III. METODOLOGIA

A planta escolhida para este projeto foi de um sistema de limpeza de torre de resfriamento, muito utilizado para remover particulados encrustados em trocadores de calor. A limpeza é realizada por um óleo específico que deve se manter a uma temperatura de aproximadamente 90ºC e o aquecimento desse óleo é realizado por uma serpentina com vapor superaquecido, controlado por uma válvula automática, dentro de um reservatório horizontal. O óleo é bombeado do reservatório para a torre através da bomba 1 e da torre retorna para o reservatório devido à bomba 2, efetuando assim a limpeza e garantia de bom funcionamento do equipamento.

Serão elaborados inicialmente fluxogramas para definir quais dados devem ser controlados e exibidos, principalmente em setores críticos. Em seguida dois esquemáticos devem ser feitos, um de montagem do sistema de controle e outro de comunicação entre as partes do projeto. Após isso, a implementação deve ser iniciada.

A. Fluxogramas

1) Operação normal

A operação normal do processo segue o fluxograma apresentado na Figura 2. O funcionamento normal ou cíclico do sistema se inicia a partir do botão *Iniciar Processo*. Algumas condições são verificadas para o correto funcionamento.

Para a bomba 1 existem 3 funções. A primeira delas é a *Manual – Desligado*, que desativa a bomba 1 e não realiza nenhuma ação no sistema. A segunda é a *Manual – Ligado*, que aciona a bomba 1 independentemente do sistema de controle de temperatura. A terceira é a *Automático*, que faz o sistema atuar de acordo com o controle de temperatura.

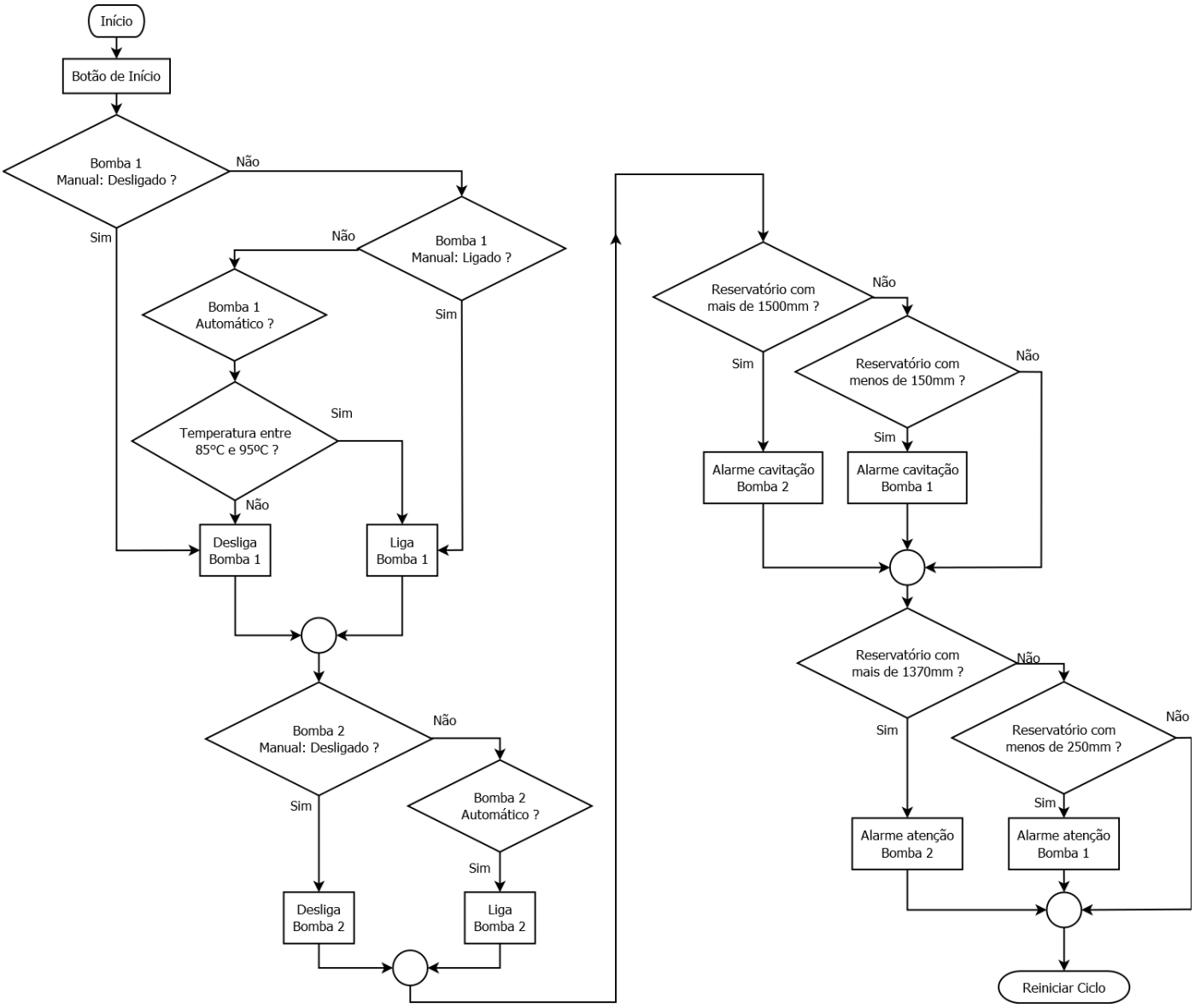

Figura 2 - Fluxograma de funcionamento normal do sistema.

Para a bomba 2 existem 2 funções. A primeira delas é a *Manual – Desligado*, que desativa a bomba 2 e não realiza nenhuma ação no sistema. A segunda é a *Automático* que aciona a bomba 2.

Quando o reservatório se encontra com nível alto de óleo, o alarme de atenção da bomba 2 é acionado. Quando o reservatório se encontra com nível baixo de óleo, o alarme de atenção da bomba 1 é acionado.

Quando o reservatório chega ao nível muito alto de óleo, o alarme de falha da bomba 2 é acionado e um *pop-up* de alerta é gerado indicando cavitação e desligamento dessa bomba. Quando o reservatório chega ao nível muito baixo de óleo, o alarme de falha da bomba 1 é acionado e um *pop-up* de alerta é gerado indicando cavitação e desligamento dessa bomba.

2) Controle de temperatura

O controle de temperatura é uma variável crítica e é a partir dele que o sistema busca se adequar para que a limpeza seja efetivamente concluída. A Figura 3 abaixo representa o fluxograma desse controle.

Durante o processo, quando a temperatura se encontra em mais de 95ºC, a válvula de vapor superaquecido é desligada e o sistema acusará temperatura alta. Quando a temperatura é menor que 85ºC, a válvula de vapor é acionada e o sistema acusará temperatura baixa. Quando a temperatura se encontra dentro do range ideal, entre 85ºC e 95ºC, a válvula se mantém ligada e o processo continua normalmente.

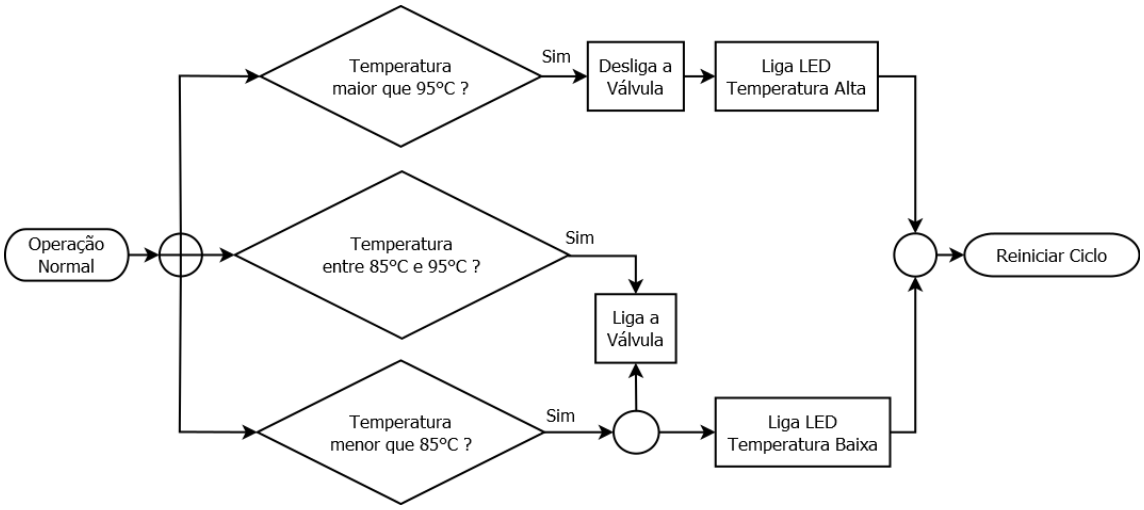

Figura 3 - Fluxograma de funcionamento da válvula de aquecimento do óleo.

3) Controle de emergência

O controle de emergência consiste numa rotina de segurança do processo caso haja alguma falha externa que o sistema supervisório não conseguiu tratar. O fluxograma desse controle está exibido na Figura 4 abaixo.

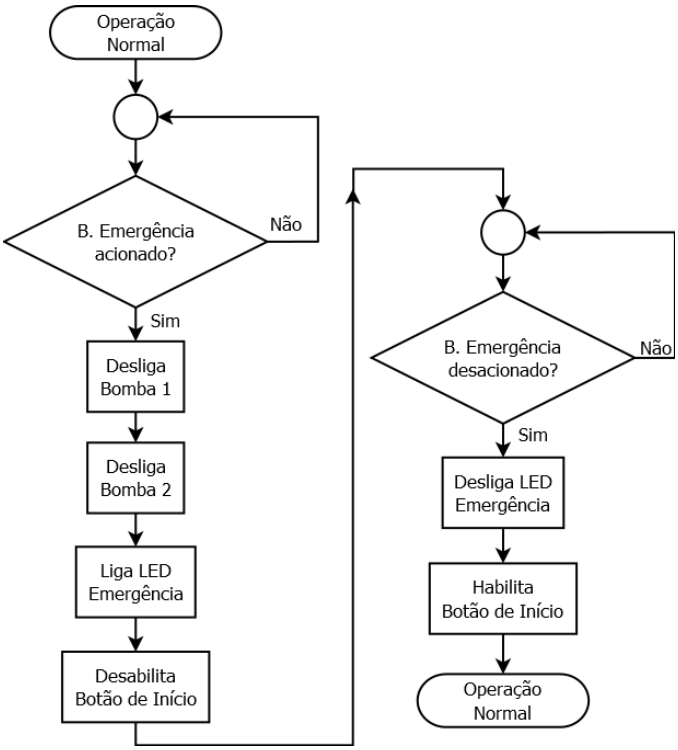

Figura 4 - Fluxograma de funcionamento do sistema de emergência.

Quando o botão de emergência é acionado, as bombas 1 e 2 são desligadas, o LED (*Light Emitting Diode*) de indicação de emergência é acionado e o botão de início do processo é desabilitado. Quando o botão de emergência é desabilitado, o LED de indicação de emergência é desligado e o sistema está pronto para voltar à operação normal quando o botão de início for acionado novamente.

B. Idealização do Sistema

A partir dos fluxogramas, um esquemático de montagem do sistema de controle está apresentado na Figura 5 abaixo.

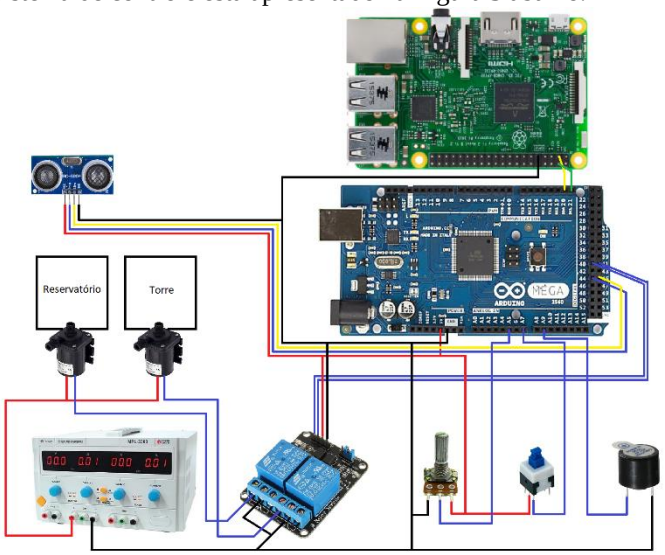

Figura 5 - Esquemático de montagem do sistema de controle.

São necessários, então, dois tanques para a simulação do reservatório e da torre de resfriamento, um sensor ultrassônico que medisse o nível de óleo do reservatório, um sensor de temperatura (simulado por um potenciômetro) para medir a temperatura do óleo do reservatório, duas bombas para mandar fluido de um tanque para outro, uma chave retentiva para simular o botão de emergência e um *buzzer* para representar a sirene caso o botão de emergência seja acionado.

Apenas as bombas do reservatório e da torre devem ser alimentadas com uma fonte ajustável de 12V, ficando o Arduino

responsável somente pelo acionamento das mesmas, já que sua saída é de apenas 5V.

A comunicação entre as partes do projeto está representada na Figura 6 abaixo.

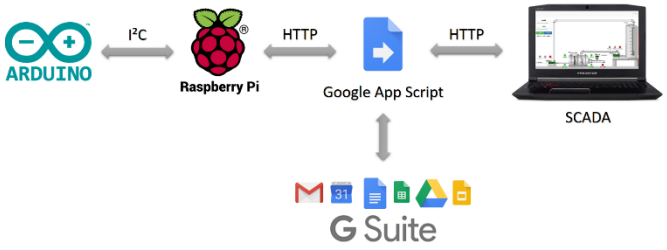

Figura 6 - Esquemático de comunicação.

O Arduino controla as variáveis do sistema e comunica com o Raspberry, que realiza requisições aos aplicativos web disponíveis pelo GAS, assim como a sistema SCADA. Esses aplicativos manipulam os dados e utilizam as ferramentas disponíveis do G Suite.

IV. Resultados

Por se tratar de um projeto completo de controle e supervisão de uma planta industrial simplificada, foi necessário desenvolver cada etapa individualmente e uni-las para obter o resultado esperado. No tópico IV.A estão definidas as *tags* do processo e a lógica de controle do Arduino seguindo os fluxogramas apresentados; nos tópicos IV.B e IV.C são discursadas as rotinas implementadas no Raspberry para troca de dados com o Arduino e o GAS, respectivamente; no tópico IV.D encontra-se informações sobre o funcionamento do banco de dados e os web apps desenvolvidos através do GAS; e por fim o tópico IV.E apresenta a interface supervisória da planta industrial trabalhada.

A. Sistema de Controle com o Arduino

1) Mapa de variáveis

Nem todas as variáveis do processo serão armazenadas no banco de dados, logo foram separadas na Tabela 1 as *tags* de controle e na Tabela 2 as de controle e armazenamento.

Tabela 1 - Mapa de variáveis apenas de controle.

Nome	Endereço	Tipo	Descrição
LOS_EMERGENCY	A9	Bool	Sinaleiro sonoro de emergência
POS_EMERGENCY	A7	Bool	Botoeira de emergência
FBK_OK_SYSTEM	M0	Bool	Verificação de sistema ok
KYV_PUMP1_START	D40	Bool	Relé de acionamento da bomba 1
KYV_PUMP2_START	D41	Bool	Relé de acionamento da bomba 2
SNS_R_OIL_LEVEL	D44/45	Float	Sensor ultrassônico de nível de óleo no reservatório
SNS_R_OIL_TEMPERATURE	A5	Int	Sensor potenciômetro de temperatura do óleo no reservatório

Tabela 2 - Mapa de variáveis de controle e armazenamento.

Nome	Endereço	Tipo	Descrição
COM_MODE_PUMP1	QW0	Int	Modo de operação da bomba 1
COM_MODE_PUMP2	QW1	Int	Modo de operação da bomba 2
COM_START	Q0.0	Bool	Comando de partida do processo
FBK_PUMP1_ALARM	I0.0	Bool	Alarme de possível falha da bomba 1
FBK_PUMP1_FAULT	I0.1	Bool	Falha da bomba 1
FBK_PUMP1_START	I0.2	Bool	Comando de acionamento da bomba 1
FBK_PUMP2_ALARM	I1.0	Bool	Alarme de possível falha da bomba 2
FBK_PUMP2_FAULT	I1.1	Bool	Falha da bomba 2
FBK_PUMP2_START	I1.2	Bool	Comando de acionamento da bomba 2
FBK_T_OIL_LEVEL	IW0	Float	Calculo do nível de óleo da torre
FBK_R_OIL_LEVEL	IW1	Float	Cálculo do nível de óleo do reservatório
FBK_R_OIL_TEMPERATURE	IW2	Float	Cálculo da temperatura do óleo do reservatório
FBK_SUPERHEATED_STEAM	I2.0	Bool	Válvula de controle de entrada de vapor superaquecido no reservatório
FBK_HLEVEL	I2.1	Bool	Nível alto de óleo no reservatório
FBK_HTEMP	I2.2	Bool	Temperatura alta do óleo no reservatório
FBK_LLEVEL	I2.3	Bool	Nível baixo de óleo no reservatório
FBK_LTEMP	I2.4	Bool	Temperatura baixa do óleo no reservatório
FBK_EMERGENCY	I3.0	Bool	Sinal de emergência

Para as variáveis apresentadas nas tabelas acima há padrões de prefixo, sendo eles:

- SNS: Sensor físico em área;
- POS: Botão físico em uma OS (*Operator System*), que é um painel de controle em área;
- LOS: Indicador físico em uma OS na área;
- KYV: Atuador, normalmente válvulas, porém utilizado aqui para relés de acionamento;
- FBK: Retorno de uma variável interna do sistema;
- COM: Comando interno do programa acionado por um usuário (botão virtual).

2) Rotinas de controle

Toda a lógica do processo foi feita através de 4 rotinas construídas no Arduino, sendo elas: *main_routine()*, *acionamento()*, *reservatorio()* e *torre()*.

a) main_routine()

Essa rotina tem como função iniciar o processo quando o botão de início é pressionado no supervisório e ainda fazer o gerenciamento da situação emergência. Assim ela não permite que as bombas sejam ligadas quando o botão de emergência estiver acionado.

b) acionamento()

Essa rotina faz o controle das duas bombas do processo. Para ligar a bomba 1, a rotina confere o modo de operação (*Automático*, *Manual – Ligado* e *Manual – Desligado*), faz a checagem de temperatura caso necessário, verifica o comando de início do processo e se a bomba 1 não está em falha.

Já que a bomba 2 tem apenas os modos de operação (*Automático* e *Manual – Desligado*), uma vez que a mesma não tem sistemas críticos, de maneira análoga a bomba 1 o sistema confere os intertravamentos e liga a bomba caso tudo esteja em condições de operações.

As duas bombas ainda podem entrar em falha e desligar com o objetivo de evitar cavitação, sendo novamente acionadas somente se a condição normal de operação for restabelecida. A bomba 1 entra em falha caso o nível do reservatório esteja abaixo de 150mm e a bomba 2 caso o nível calculado da torre esteja abaixo de 72mm.

c) reservatorio()

Essa rotina controla o processo no que tange ao nível e a temperatura do reservatório, fazendo ainda a conversão da leitura analógica do nível (através de um sensor ultrassônico, cuja biblioteca está no *link* em [13]) e da temperatura (simulada através do potenciômetro). Essas leituras são convertidas para milímetros e graus Celsius, respectivamente.

A partir dos valores obtidos na conversão o código tem condições de acionar as variáveis booleanas com *links* aos indicadores do supervisório relacionados ao reservatório, sendo eles: nível alto, nível baixo, temperatura alta e temperatura baixa.

Com relação à temperatura, o controle da válvula automática do trocador de calor do reservatório tem a função de manter a temperatura do óleo sempre no range estabelecido (85-95ºC). Dessa maneira a válvula permanece aberta em temperaturas até 95ºC e fechada acima deste valor.

Quanto ao nível, são gerados alarmes caso o tanque fique abaixo de 250mm, nessa condição a bomba já fica em estado de alarme.

d) torre()

Essa rotina tem como função realizar o controle das funções relacionadas à torre de resfriamento. Para o cálculo do nível da torre utilizou-se o artifício de estimação do nível de óleo, economizando assim um sensor no projeto. Uma vez que o sistema é fechado e são conhecidas as dimensões do reservatório e da torre, a ideia de conservação de massa é aplicada, ou seja, a quantidade de óleo que sai do reservatório entra na torre e vice-versa. Essa rotina ainda gera um alarme na bomba 2 quando o nível na torre está abaixo de 164mm.

B. Troca de Dados Arduino-Raspberry

As informações fluem do Arduino para o Raspberry e vice-versa através da comunicação I²C (*Inter-Integrated Circuit*), que é um protocolo facilmente implementado nas duas plataformas.

Ele trabalha no modelo mestre-escravo, com pelo menos um dispositivo atuando como mestre e os demais dispositivos (até 127) como escravos. Para este projeto, o Raspberry será o mestre da comunicação e o Arduino será o escravo.

A função do mestre é coordenar a comunicação podendo solicitar a qualquer um dos escravos da rede para enviar ou receber dados, enquanto os escravos ficam na escuta e respondem às solicitações do mestre.

No Arduino a comunicação I²C é configurada através da inclusão da biblioteca nativa *Wire.h*. É necessário para cada escravo definir um endereço hexadecimal exclusivo para que o mestre possa estabelecer comunicação. A comunicação é aberta através da função *Wire.begin()*, em que o endereço do Arduino é passado como parâmetro, e as escutas através das funções *Wire.onReceive()* e *Wire.onRequest()*, passando a função para receber e enviar dados, respectivamente.

No Raspberry a comunicação I²C vem desativada inicialmente e para habilitá-la basta abrir o terminal de comandos e digitar *sudo rasp-config*. Uma interface irá abrir, selecione *5 Interfacing Options* e habilite a comunicação em *P5 I2C*. É necessário instalar pelo terminal os pacotes *I2C-Tools* e *smbus* através do comando *sudo apt-get install i2c-tools python-smbus*, que possibilitam o uso do barramento I²C com a linguagem Python e o uso de ferramentas de configuração. Se tudo estiver correto é possível visualizar os escravos conectados através do terminal pelo comando *i2cdetect –y 0* ou *i2cdetect –y 1*.

A classe *I2CProtocol* foi criada, sendo necessário importar dentro dela a biblioteca *smbus* ou *smbus2*. O construtor dessa classe recebe o endereço hexadecimal do Arduino, define o barramento através da função *smbus.SMBus()*, o *offset* nulo para o vetor de dados e o tamanho desse vetor. Dois métodos são criados, *read_block_data()* e *write_block_data()*, para respectivamente receber e enviar um bloco de informações.

Esses dois métodos enviam ou recebem um vetor de até 32 *bytes* do tipo *char*, que deve ser tratado posteriormente. Assim, não há como passar em uma requisição o nome ou *tag* mais o valor do dado para identificação do receptor, o que se faz a necessidade de enviar os dados em uma sequência conhecida entre as duas partes. Isso é um ponto negativo e de atenção durante o projeto, pois a ordem de envio de dados terá de ser atualizada no mestre e nos escravos em uma modificação do projeto original.

Apenas as três primeiras *tags* da Tabela 2 são enviadas do Raspberry para o Arduino. Esse processo utiliza somente uma requisição do protocolo I²C, pois como esses valores são booleanos e ocupam uma posição cada do vetor de dados, os três são agrupados em sequência conhecida e enviados pelo Raspberry.

Todas as variáveis da Tabela 2, incluindo a *tag* COM_START, são enviadas do Arduino para o Raspberry. Esse processo utiliza uma requisição por valor de *tag* do protocolo I²C, pois há valores *floats* entre esses dados que necessitam de um número variável de posições no vetor de dados. Assim, o valor é tratado no emissor, mandado individualmente e também tratado no receptor.

C. Troca de dados Raspberry-GAS

Os dados que foram recebidos do Arduino precisam ser salvos num banco de dados para serem historiados e acessados pelo supervisório. O banco de dados está localizado na nuvem do Google e está acessível por Web Services disponíveis através da ferramenta GAS.

Para acessar este banco é necessário realizar requisições *post*, *get*, *put* e *delete* do protocolo HTTP, porém o GAS tem suporte nativo apenas para os métodos *post* e *get*, logo, todos os dados transitados utilizam um desses dois métodos. Dados que devem ser atualizados no banco são passados através do método *post*, porém quando é necessário pegar algum dado do banco o método *get* que é chamado. Fazer requisições HTTP pela linguagem Python necessita instalar a biblioteca *requests* pelo terminal através do comando *pip install requests*.

Ao iniciar o Raspberry é necessário fazer uma sincronização com o servidor para obter todas as variáveis do banco, utilizando para isso as classes implementadas *SyncVariables* e *Variable*, detalhadas a seguir.

A classe *Variable* é inicializada com as seguintes informações da variável: *name, description, tag, type, eng_unit, l_limit, h_limit, ll_limit, hh_limit* e *last_value*. Cada parâmetro da variável contém os métodos *getters()* e *setters()*, além do método *print_variable_info()* que imprimi todos os parâmetros para visualização.

A classe *SyncVariables* é inicializada com a URL (*Uniform Resource Locator*) do aplicativo web que trata de sincronização. O método *synchronize()* faz a requisição *get* na URL e recebe um JSON (*JavaScript Object Notation*) com todas as informações das variáveis no banco. O JSON recebido é tratado e a lista *variables* que foi passada de parâmetro é preenchida com os dados.

Posteriormente, o programa entra em um ciclo que realiza requisição *get* para pegar os valores das 3 primeiras variáveis da Tabela 2, envia para o Arduino essas informações, recebe do Arduino os valores das outras variáveis da Tabela 2 incluindo a COM_START e, por fim, realiza requisição *post* para salvar no banco. A taxa de atualização foi definida em 2 segundos, ou seja, a cada 2 segundos este processo é repetido.

As requisições para o servidor são feitas pelas classes *GetRequestPLC* e *PostRequestPLC*. Essas classes são inicializadas com suas respectivas URLs que tratam das requisições vindas do Raspberry. O método *post* necessita ainda de um usuário e senha com permissão para poder modicar os dados do banco, logo no corpo da mensagem são passados esses dois parâmetros para posterior autenticação. Uma codificação básica utilizando Base64 é aplicada.

Quando uma chamada HTTP é realizada pelo programa, a *thread* que realizou a chamada fica travada até receber um retorno do servidor. Para contornar este problema é necessário utilizar programação paralela. Dessa forma, utilizou-se a biblioteca *threading* para realizar este paralelismo (esse método não utiliza paralelismo real, porém atende bem à necessidade do projeto). Dessa forma, é necessário inserir o comando *threading.Thread.__init__(self)* no construtor da classe e sobrescrever o método *run()*, que efetivamente irá realizar as requisições.

D. Banco de Dados e Aplicativos Web com o GAS

Com uma conta Google criou-se uma planilha Google denominada *db.gsheet*, contendo 3 abas: *credentials*, *variables_info* e *variables_hist*.

A primeira aba contém usuário, senha, e-mail, data de criação e grupo de todas as contas cadastradas. Todos os grupos têm permissão de realizar requisições *post* no sistema, sendo que o grupo de administradores contém os DBDs (*Database Developers*) e os DBAs (*Database Administrators*), o grupo de controladores contém os Raspberrys do processo e o grupo de supervisores contém aos usuários que utilizam o sistema de supervisão pela web.

A segunda aba contém todos os dados das variáveis que são utilizadas no sistema SCADA. Na última coluna está o último valor da variável, valores estes recebidos através de requisições *post* do sistema de controle e supervisão. As informações relacionadas ao modo de funcionamento das bombas e botão de *Iniciar Processo* são atualizadas pelo sistema supervisório, enquanto as outras variáveis através do processo de controle.

A terceira aba contém o histórico das variáveis, de forma vertical, sendo inseridos novos dados sempre na linha superior, deslocando os mais antigos para baixo. Na primeira coluna contém a data e hora de quando esses valores foram gravados pelo servidor. Esta aba é inteiramente alimentada pelas requisições *post* do sistema de controle.

Um *script* do Google pode ser criado individualmente ou pode estar associado a uma planilha, não sendo disponível o acesso sem sua planilha. Para o projeto foram criados 5 *scripts* individuais: *sync_variables*, *requests_plc*, *requests_scada*,

hist_requests_scada e *auth_user*. A extensão de todos esses scripts é *gscript*.

O *script sync_variables* é responsável por enviar um vetor com todas as informações das variáveis do banco no formato JSON quando uma requisição *get* for feita. Da mesma forma, os *scripts requests_plc* e *requests_scada* têm métodos *get* e *post* implementados para enviar e receber os últimos valores das variáveis, tendo uma função de autenticação inclusa para as requisições *post*. O *script hist_requests_scada* envia todos os valores historiados de uma variável com o nome passado por parâmetro no método *get*. Por último, o *script auth_user* é responsável por receber um usuário e senha pelo método *post* e verificar se o mesmo está cadastrado na aba de credenciais.

Dentro de cada função de autenticação acontece primeiramente a decodificação Base64 do usuário e senha passados como parâmetros, para depois a verificação com a aba de credenciais. Por mais que o método *post* passe os parâmetros por meio do corpo da mensagem, e não no cabeçalho como no método *get*, caso a informação seja interditada por alguém as credenciais ficam facilmente acessíveis, logo é necessário, numa aplicação real, utilizar métodos de criptografia e autenticação mais seguros e completos, como o SHA-256.

Para publicar esses *scripts* como Web Apps, basta ir em *Publicar* e clicar em *Implantar como aplicativo da web*. Uma janela irá ser aberta contendo a URL, versão, modo de execução e permissão de acesso do aplicativo. Como ele será utilizado por diversas fontes, é necessário autorizá-lo para *Qualquer pessoa, mesmo anônima*.

Ao publicar os *scripts*, basta ter acesso à URL do aplicativo que os métodos *doGet()* e *doPost()* responderão às requisições que chegarem a ele, enviando ou recebendo dados de acordo com a lógica desenvolvida e a permissão de cada usuário que tenta acessar. É possível ainda que um *script* seja programado para executar uma determinada função de tempos em tempos.

Através dos serviços do G Suite, com poucas linhas de código, é possível mandar e-mail alertando sobre uma parada inesperada do processo, enviar um relatório de produção para a gerência, armazenar todos os documentos técnicos em uma pasta compartilhada, e outras inúmeras aplicações que automatizam e tratam melhor os dados.

Quando o Raspberry faz uma requisição para salvar os dados no banco, três variáveis são previamente analisadas (FBK_PUMP1_FAULT, FBK_PUMP2_FAULT e FBK_EMERGENCY) e caso uma delas estiver acionada um e-mail é enviado para todos os usuários do grupo de administradores dizendo que o processo foi interrompido devida àquela variável. A conta utilizada é gratuita, logo o Google disponibiliza apenas 100 e-mails diários, caso seja necessário utilizar mais que este número é possível contratar planos.

E. Sistema de Supervisão via Web

Inicialmente foi desenvolvido uma página de abertura do projeto a qual serviu de aprendizado e treino da programação web *front-end*. Nela há um botão para direcionar até o repositório do GitHub [14] em que pode ser baixado todos os arquivos deste trabalho.

Ao clicar no projeto *Automação Industrial*, o usuário será redirecionado para o supervisório exibido na Figura 7. Esse sistema SCADA tem a imagem de fundo estática apresentada na Figura 8, inserida no HTML para representar o reservatório, a torre, as bombas e o encanamento.

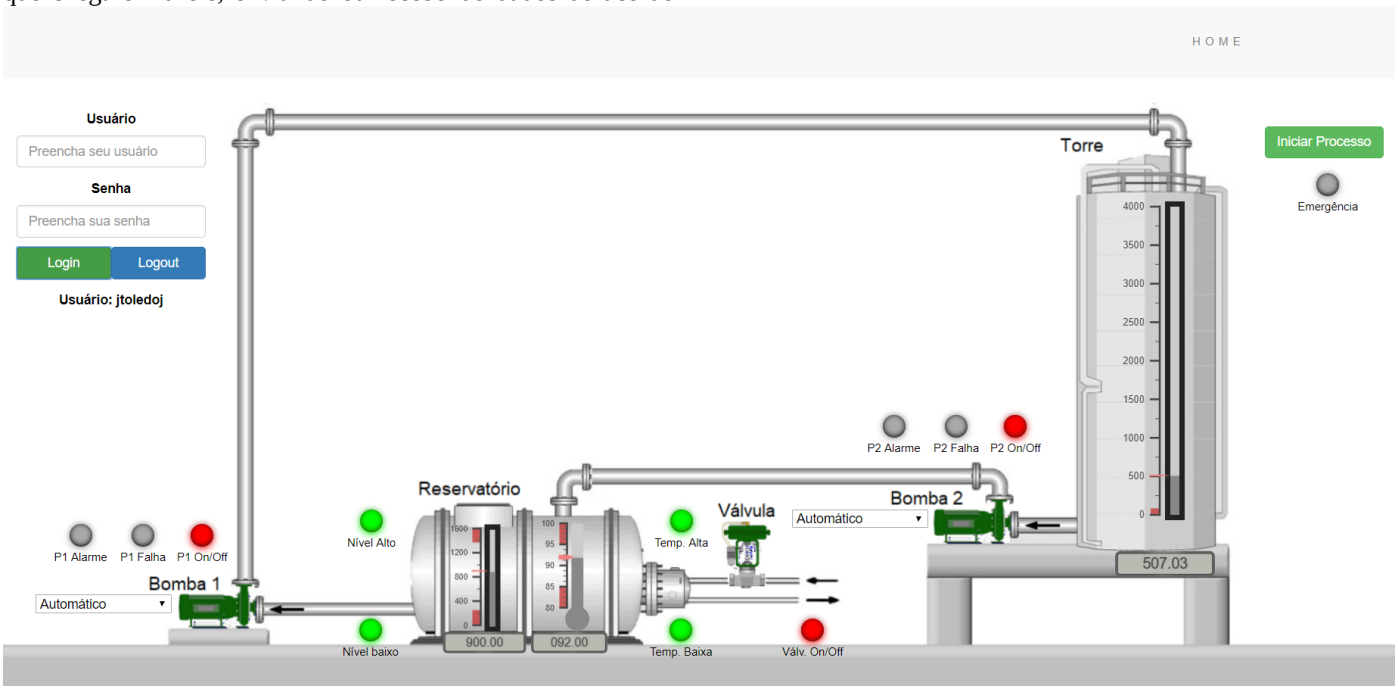

Figura 7 - Sinótico para desktop do sistema proposto.

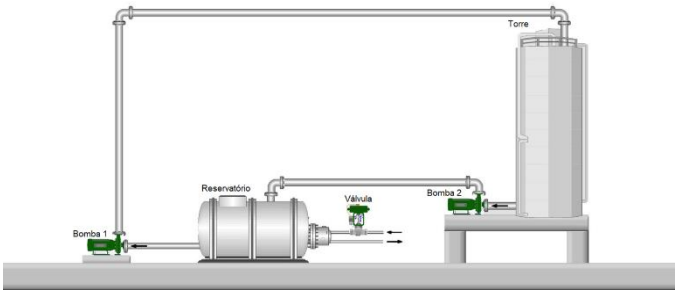

Figura 8 - Imagem de fundo do sinótico representado na Figura 7.

Os campos de autenticação no canto superior esquerdo são formados pela *tag* <*form*>. O botão de *Login* não é um botão de *submit* do formulário, mas chama a função *login()* do JavaScript, que pega o usuário e senha, aplica a codificação Base64 e realiza uma requisição *post* através de JQuery para o *script auth_user*. O retorno dessa requisição é analisado para emitir um *pop-up* com a função *alert()* dizendo se o usuário tem ou não permissão de *login* no sistema. O usuário fica logado e com seu *login* exibido abaixo dos botões até que o mesmo clique no botão *Logout*, que chama a função *logout()* para realizar tal atividade.

Após fazer *login* no sistema, o botão *Iniciar Processo* e os modos de funcionamento das bombas passarão a surtir efeito no processo controlado, modificando assim seus respectivos valores na aba *variables_info* da planilha de banco de dados.

Os LEDs fazem parte de um projeto *open source* [15] e são basicamente *tags* <*div*> estilizadas com as classes *led-red*, *led-green*, *led-gray*, *led-yellow*, *led-red-blink* e *led-yellow-blink*. Todos eles foram posicionados manualmente utilizando a programação em CSS. Dependendo dos valores de cada variável, a classe atual de estilização do LED é removida e outra classe é adicionada. O padrão de cores para as variáveis está listado abaixo:

- Indicadores: cor vermelha se o bit for verdadeiro (1) e verde se for falso (0);
- Alerta: cor amarela intermitente se o bit for verdadeiro (1) e cinza se for falso (0);
- Falha ou emergência: cor vermelha intermitente se o bit for verdadeiro (1) e cinza se for falso (0);

Os *gauges* de nível e temperatura são *tags* <*canvas*> desenhados através da biblioteca *canvas-gauge* [16], onde são instanciados alguns *LinearGauges* e parametrizados de acordo com o desenvolvedor. O posicionamento deles também foi feito através do arquivo CSS responsável.

Ao clicar nos *gauges* um gráfico é exibido abaixo do sinótico como mostra na Figura 9. Para isso, a função *atualiza_grafico()* é chamada recebendo o nome da variável do *gauge* que foi clicado. Logo, essa função faz uma requisição *get* para o *script hist_requests_scada*, que retorna um vetor com as informações historiadas da variável em questão. O botão fechar esconde o gráfico atual. A biblioteca utilizada é a *Chart.js*, cheia de ferramentas de personalização de gráficos e muito prática de ser implementada.

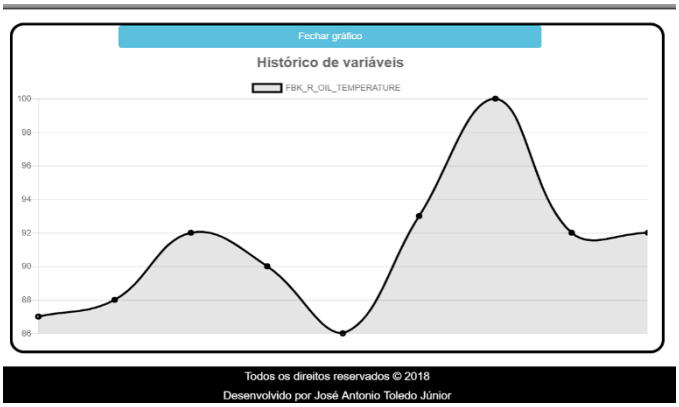

Figura 9 - Exemplo de gráfico de uma variável historiada.

Duas funções são executadas a cada 1 segundo para tornar dinâmico o sistema de supervisão. A função *recebe_dados()* faz uma requisição *get* para o *script requests_scada*, atualizando os valores dos *gauges* e alterando as classes de estilização dos LEDs. A função *envia_dados()* faz uma requisição *post* para o mesmo aplicativo da web e envia os parâmetros setados no sistema supervisório utilizando o usuário e senha que está logado no sistema.

O sistema é totalmente responsivo aos diversos tamanhos de dispositivos. A Figura 10 abaixo ilustra o design responsivo para um smartphone, o que permite a utilização fácil e rápida do sistema SCADA no chão-de-fábrica ou em uma viagem a negócios, por exemplo.

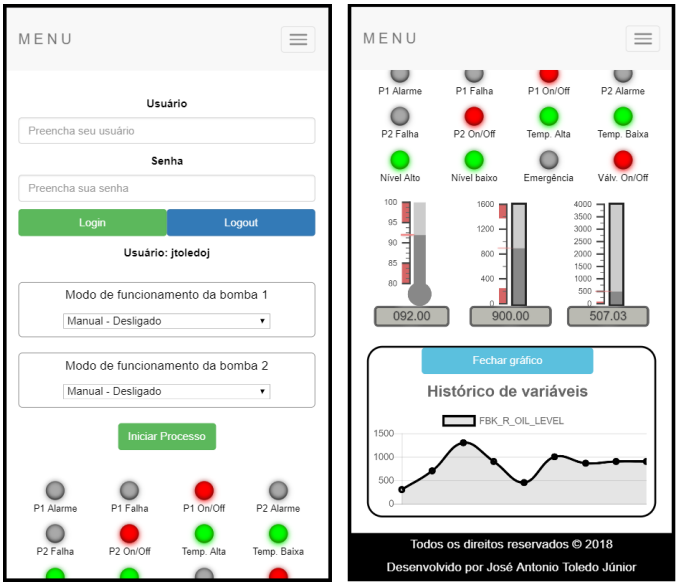

Figura 10 - Sinótico para mobile do sistema proposto.

Para entrar na aplicação a partir de qualquer dispositivo conectado à Internet é necessário hospedá-la em um provedor de hospedagem de sites, como a HostGator e a Uol Host, onde um endereço IP real e URL serão definidos para acesso.

V. Conclusão

O presente trabalho integrou diversas tecnologias e conceitos de programação e automação para obter seus resultados. Assimilar tantas ferramentas, linguagens e metodologias distintas é um processo difícil e demorado, mas foi sendo individualmente dominado e funcionou muito bem em conjunto com as partes.

A ideia do trabalho de desenvolver um sistema completo de controle e supervisão de uma planta industrial simplificada cumpre os objetivos propostos permitindo: a supervisão através de qualquer dispositivo conectado à Internet necessitando apenas do navegador web; o controle flexível do processo com o Arduino e Raspberry; e a tratativa dos dados através de computação em nuvem, sem nenhuma restrição de tempo real, utilizando as diversas ferramentas disponíveis pelo GAS.

O processo industrial de limpeza de torre de resfriamento, mesmo que simplificado, está bastante completo no quesito de intertravamentos para funcionar sem danos aos equipamentos, com suas funções de controle de operação normal, temperatura e situações de emergência integradas.

O Arduino se mostrou eficiente no processamento dos dados de controle e o Raspberry realizou com facilidade a troca de informações entre o Arduino e o banco de dados na nuvem. Fica proposto de melhoria para sistema definir um protocolo de envio e recebimento dos dados utilizando a comunicação I²C ou mesmo testar outros tipos de comunicação entre Arduino e Raspberry para eliminar a necessidade de conhecer a sequência dos dados enviados entre as duas plataformas.

Os recursos oferecidos pelo GAS têm grande utilidade na indústria a fim de automatizar relatórios e notificações de falhas, por exemplo, porém a tentativa de substituição de um banco de dados tradicional por uma planilha do Google não foi a melhor escolha, pois é gasto um tempo razoavelmente grande para se escrever um dado na planilha – aproximadamente 250ms.

O sistema supervisório teve seu sinótico muito fidedigno ao sistema real, sendo desenvolvido com tecnologias já tradicionais da programação web e responsivo à diversos dispositivos. A visualização da informação se mostra clara e objetiva através dos *gauges* e LEDs, e a análise do dado pode ser facilmente feita através dos seus valores históricos exibidos em forma de gráficos.

Atualmente existem editores de site que permitem criar páginas na web sem a necessidade de conhecer nada de programação, como os ofertados pela Wix e pelo HostGator. Porém tais empresas não disponibilizam o código fonte para alterações internas e manipulação dos dados do processo, o que limita construir através dessas ferramentas um sistema SCADA completo. Dessa forma, caso alguma empresa disponibilize de edição de sites e código fonte, grandes sistemas supervisórios totalmente acessíveis pelo navegador podem ser desenvolvidos.

Por fim, a questão da segurança da informação trafegada entre os sistemas não foi o foco deste projeto, mas é de fundamental importância para um sistema funcional real. Aqui todos os dados de autenticação foram passados no corpo da requisição com apenas uma codificação Base64, dessa forma fica de sugestão para melhorias o aumento de complexidade na segurança dos dados e a utilização de padrões de autenticação já bem definidos na comunidade de desenvolvedores, como o SHA-256.

Agradecimentos

Primeiramente a Deus, Aquele que se apresentou e se apresenta nas mais variadas formas, me dando forças para finalizar esta etapa da minha vida e continuar buscando sempre melhorar profissional e pessoalmente.

Em especial aos meus pais, a quem devo a vida, a formação moral e a criação da minha identidade. Meu reconhecimento e gratidão por toda a paciência, amor, compreensão e apoio constante.

Aos meus irmãos e minha namorada, que me trazem uma sensação de felicidade e tranquilidade para continuar lutando e poder aproveitar a vida ao lado deles.

Aos professores, os parabenizo pelo compromisso e dedicação em compartilhar e desenvolver conhecimento nessa jornada trabalhosa de pesquisa, ensino e extensão. Meus eternos agradecimentos pelos inúmeros ensinamentos e bom convívio.

Aos meus queridos colegas e amigos, pelos momentos de convívio, risos, troca de conhecimentos, afetos e intrigas, e que por meio delas me fez amadurecer.

Referências

[1] VILELA, P. S. C.; VIDAL, F. J. T. *Automação Industrial*. maio 2003. 5p. Trabalho apresentado como requisito parcial para aprovação na disciplina Redes para Automação Industrial, Universidade Federal do Rio Grande do Norte (UFRN), Natal, Brasil, 2003. Disponível em: <https://www.dca.ufrn.br/~affonso/FTP/DCA447/trabalho1/trabalho1_19.pdf>. Acesso em: 20 nov. 2018.

[2] SIEMENS. *WinCC/Server*. Disponível em: <https://w3.siemens.com/mcms/human-machine-interface/en/visualization-software/scada/wincc-options/wincc-server/pages/default.aspx>. Acesso em: 04 dez. 2018

[3] McHUGH, D. *Implementing TCP/IP in SCADA Systems*. ago. 2003. 66p. Tese (Mestrado em Engenharia) – Galway-Mayo Institute of Technology (GMIT), Galway, Ireland, 2003. Disponível em: <https://research.thea.ie/handle/20.500.12065/398>. Acesso em: 20 nov. 2018.

[4] EMBARCADOS. *Arduino MEGA 2560*. 28 abr. 2014. Disponível em: <https://www.embarcados.com.br/arduino-mega-2560/>. Acesso em: 21 nov. 2018.

[5] FILIPEFLOP. *Primeiros passos com Raspberry Pi e Linux*. 1 fev. 2018. Disponível em: <https://www.filipeflop.com/blog/primeiros-passos-raspberry-pi-e-linux/>. Acesso em: 21 nov. 2018.

[6] PYTHON Wiki. *The Python Wiki*. 16 set. 2018. Disponível em: <https://wiki.python.org/moin/FrontPage/>. Acesso em: 21 nov. 2018.

[7] PYTHON Brasil. *QUAL Python?*. Disponível em: <https://python.org.br/qual-python/>. Acesso em: 21 nov. 2018.

[8] SOUSA, F. R. C.; et al. *Gerenciamento de Dados em Nuvem:* Conceitos, Sistemas e Desafios. dez. 2011. In: SWIB 2010. cap. 4. Disponível em: <http://200.17.137.109:8081/novobsi/Members/josino/fundamentos-de-banco-de-dados/2012.1/Gerenciamento_Dados_Nuvem.pdf >. Acesso em: 21 nov. 2018.

[9] VIEIRA, J. C. *Linguagem de Programação para Nuvem:* Experiências com Google App Script. 2014. 62p. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Universidade Federal de Santa Maria (UFSM), Santa Maria, Brasil, 2014. Disponível em: <http://docplayer.com.br/20294205-Linguagens-de-programacao-para-nuvem-experiencias-com-google-apps-script.html >. Acesso em: 21 nov. 2018.

[10] FAZER Lab. *Dados em tempo real com planilha do Google Docs*. 24 out. 2017. Disponível em: <https://fazerlab.wordpress.com/2017/10/24/dados-em-tempo-real-com-planilha-do-google-docs/>. Acesso em: 20 nov. 2018.

[11] TREINAWEB. *O que é front-end e back-end?*. 30 jan. 2017. Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-front-end-e-back-end/>. Acesso em: 22 nov. 2018.

[12] TUTORIAL Web Design. *O que é Bootstrap?*. 29 abr. 2014. Disponível em: <http://www.tutorialwebdesign.com.br/o-que-e-bootstrap/>. Acesso em: 22 nov. 2018.

[13] GITHUB. *Biblioteca Sensor Ultrassonico HC-SR04*. 27 jul. 2015. Disponível em: <https://github.com/filipeflop/Ultrasonic/>. Acesso em: 15 set. 2018.

[14] GITHUB. *Automacao Industrial*. 22 nov. 2018. Disponível em: <https://github.com/JTOLEDOJ/Automacao-Industrial/>. Acesso em: 22 nov. 2018.

[15] CODEPEN. *CSS LED Lights*. Disponível em: <https://codepen.io/fskirschbaum/pen/MYJNaj/>. Acesso em: 8 jul. 2018.

[16] CANVAS Gauges. *Other Custom Linear Gauges*. Disponível em: <https://canvas-gauges.com/documentation/examples/>. Acesso em: 7 jul. 2018.