Universidade de São Paulo–USP
Escola de Engenharia de São Carlos
Departamento de Engenharia Elétrica e de Computação

Fabrício Tietz

Desenvolvimento de um gateway de internet das coisas para automação industrial

São Carlos
2017

Fabrício Tietz

Desenvolvimento de um gateway de internet das coisas para automação industrial

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos, da Universidade de São Paulo.

Curso de Engenharia Elétrica com ênfase em

Sistemas de Energia e Automação

Orientador: Dennis Brandão

São Carlos

2017

Tietz, Fabrício
T559d Desenvolvimento de um gateway de internet das coisas para automação industrial / Fabrício Tietz; orientador Dennis Brandão. São Carlos, 2017.

Monografia (Graduação em Engenharia Elétrica com ênfase em Sistemas de Energia e Automação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2017.

1. Automação Industrial. 2. Internet das Coisas. 3. Gateway. 4. Siemens S7. 5. Modbus TCP. 6. MQTT. I. Título.

FOLHA DE APROVAÇÃO

Nome: Fabrício Tietz

Título: "Desenvolvimento de um gateway de internet das coisas para automação industrial"

Trabalho de Conclusão de Curso defendido e aprovado em 23 / 11 / 17 ,

com NOTA 8,5 (oito , cinco), pela Comissão Julgadora:

Prof. Associado Dennis Brandão - Orientador - SEL/EESC/USP

Prof. Dr. Maximiliam Luppe - SEL/EESC/USP

Mestre Paulo Henrique Toledo de Oliveira e Souza - SEL/EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Rogério Andrade Flauzino

Agradecimentos

Agradeço a Deus.

Aos meus pais por sempre buscarem proporcionar uma boa educação e apoiar durante minha trajetória até aqui.

Ao meu irmão, Rodrigo, pelos primeiros ensinamentos em informática, fundamentais para minha escolha pela área de tecnologia. E por sempre me apoiar.

Agradeço a todos meus familiares e amigos que também me apoiaram. Em especial ao meu tio Nilson por sempre acreditar na minha capacidade e trabalho.

Ao Prof. Dr. Dennis Brandão pela orientação e oportunidade de realizar esse trabalho, bem como pelos ensinamentos transmitidos durante a graduação.

As colegas do Laboratório de Automação Industrial, pelo conhecimento compartilhado e apoio durante o desenvolvimento desse projeto. Em especial aos mestrandos Murilo Silveira Rocha e Luiz Ferreira Alves.

Aos grupos extra-curriculares SIEEL (Semana da Integração da Engenharia Elétrica), da qual tive o prazer de ser co-fundador, e SA-SEL (Secretaria Acadêmica da Engenharia Elétrica da EESC/USP) por todo o desenvolvimento profissional e pessoal proporcionados ao longo do período graduação.

À empresa Raízen Energia S/A pela oportunidade de desenvolver projetos na área de internet das coisas e automação agrícola durante meu período de estágio.

Resumo

Tietz, Fabrício **Desenvolvimento de um gateway de internet das coisas para automação industrial**. 91 p. Trabalho de Conclusão de Curso – Escola de Engenharia de São Carlos, Universidade de São Paulo, 2017.

Este trabalho apresenta o desenvolvimento de um *gateway* de internet das coisas para automação industrial. O *gateway* se comunica com o ambiente industrial por meio dos protocolos Siemens S7 e Modbus TCP e utiliza o protocolo MQTT para comunicação com um servidor (*broker*) na nuvem. Uma vez integrados os dois ambientes, o usuário pode interagir com a planta industrial utilizando uma *dashboard* em um navegador *web*. Durante o desenvolvimento visou-se a utilização de ferramentas *open-source* ou de baixo custo, bem como padrões e protocolos de grande difusão, tanto no ambiente industrial, quanto no ambiente de internet das coisas. Dessa forma o trabalho pode auxiliar em diversos projetos da industria para trazer maior conectividade a plantas industriais, como também ser aproveitado em outras pesquisas da área que necessitem desse tipo de conectividade.

Palavras-chave: Automação Industrial. Internet das Coisas. Gateway. Siemens S7. Modbus TCP. MQTT.

Abstract

Tietz, Fabrício **Development of an internet of things gateway for industrial automation**. 91 p. Undergraduate Final Project – São Carlos School of Engineering, University of São Paulo, 2017.

This paper presents the development of an industrial internet of things gateway. The gateway communicates with the industrial environment through Siemens S7 and Modbus TCP protocols, and uses MQTT protocol for the communication to a cloud server (broker). One time the environments are integrated, the user can interact with the industrial plant using a dashboard on a web browser. During the development, aimed to use open-source or low cost tools, both great defunded industrial and internet of things standards and protocols. On this way, the paper may collaborate to a lot of industry projects for provide more connectivity to industrial plants, as also be used in other researches that needs this type of connectivity.

Keywords: Industrial Automation. Internet of Things. Gateway. Siemens S7. Modbus TCP. MQTT.

Lista de ilustrações

Lista de tabelas

Sumário

Capítulo 1

Introdução

A industria já passou por diversas transformações ao longo da história e, segundo Castells (1999), pelo menos duas revoluções industriais podem ser bem definidas:

- Primeira Revolução Industrial: começou por volta da década de 1760, caracterizada de forma geral pela substituição de ferramentas manuais pelas máquinas, por meio de tecnologias como a máquina a vapor, a fiadeira e o processo Cort em metalurgia.

- Segunda revolução industrial: iniciada aproximadamente cem anos após a primeira, tem como destaque o desenvolvimento da eletricidade, o motor de combustão interna, produtos químicos com base científica, a fundição eficiente do aço e início das tecnologias de comunicação (telégrafo e telefone).

A crise do petróleo em 1973 evidenciou ainda mais as dificuldades do capitalismo, notadas desde a década de 1960 por fenômenos como a estagflação (estagnação econômica com alta dos preços), o significativo aumento das taxas de juros nos mercados financeiros internacionais e a instabilidade financeira. Esse cenário abriu espaço para mudanças, inovações tecnológicas proporcionadas principalmente pela microeletrônica passaram a ser incorporadas ao processo econômico e produtivo. A informação passou a ser uma mercadoria valiosa a partir de rápida mudança na capacidade dos equipamentos em processar, armazenar, distribuir e transmitir informações através das redes de comunicação (JúNIOR, 2017). Esse período pode ser considerado como o início da Terceira Revolução Industrial.

A partir do início do século XXI, com o avanço expressivo da internet, a conectividade é uma necessidade cada vez maior, tanto de pessoas, quanto de "coisas"(dispositivos e máquinas).

Na indústria esse fenômeno também é notado, e possivelmente o período destacado será classificado como a Quarta Revolução Industrial, ou Indústria 4.0 em referência ao projeto alemão intitulado *Industrie 4.0* sustentado por políticas governamentais, empreendedores e sindicatos. (HORN, 2017)

1.1 Motivação

Muitas industrias possuem equipamentos que ainda não estão preparados para as necessidade de conectividade da indústria 4.0, porém possuem vida útil considerável até o retorno do investimento. Dessa forma um dispositivo intermediário, também chamado *gateway*, entre redes de automação industrial tradicionais e o ambiente de internet das coisas pode ser utilizado em projetos de integração dessas plantas com as novas tecnologias de comunicação sem a necessidade de substituição dos equipamentos atuais, reduzindo em muito os investimentos necessários.

Além disso, o conceito de indústria 4.0 ainda não está totalmente definido, sua evolução é constante. A conexão de dispositivos tradicionais de automação industrial com novas tecnologias de comunicação pode auxiliar diversas outras pesquisas, como por exemplo o processamento na nuvem de malhas de controle de processos industriais.

1.2 Objetivos

Este trabalho tem como objetivo buscar uma solução de *hardware* e *software* para permitir a integração de ambientes industriais com o universo da internet das coisas e, assim, contribuir com projetos das indústria e com outras pesquisas científicas que necessitem dessa conectividade.

Também objetiva-se uma aplicação prática do desenvolvimento realizado neste trabalho com a planta didática da disciplina de laboratório SEL0431 - Laboratório de Controle de Processos Industriais, e dessa forma contribuir com o aprendizado de alunos de graduação em engenharia elétrica.

1.3 Estruturação do trabalho

O trabalho é dividido em 4 capítulos:

- Primeiro: contém a introdução
- Segundo: contém a revisão bibliográfica sobre os protocolos de comunicação utilizados no projeto e sobre o hardware *Raspberry Pi*
- Terceiro: contém o estudo de todas as ferramentas necessárias para o desenvolvimento do *gateway* proposto.
- Quarto: apresenta as aplicações práticas do trabalho realizado e também testes de tempo de resposta nos etapas de comunicação

Capítulo 2

Materiais e Métodos

2.1 Protocolos de comunicação

Para o *gateway* realizar a conexão entre os ambientes da industria e de internet das coisas são necessários três protocolos:

- Siemens S7: para ler e gravar dados em CLPs do fabricante Siemens;
- Modbus TCP: para ler e gravar dados em CLPs genéricos;
- MQTT: enviar e receber dados em internet das coisas.

2.1.1 MQTT - MQ Telemetry Transport

MQTT é um protocolo de mensagens extremamente simples de leve. Sua arquitetura o torna ideal para uso em ambientes de rede com largura de banda baixa ou com alta latência, e também para dispositivos remotos com limitações de processamento e memória. (LAMPKIN et al., 2012)

Sua história começou em 1999, quando Andy Stanford-Clark (IBM) e Arlen Nipper (Arcom, hoje Cirrus Link) precisavam criar um protocolo para conexão de oleodutos através de sinal de satélite com mínimo gasto de bateria e mínima largura de banda. Com isso, especificaram os objetivos que o protocolo deveria ter: fácil implementação, entrega de dados com nível de qualidade de serviço (QOS), leve e eficiente em termos de largura de banda, suporte a diferentes tipos de dados (*data agnostic*) e reconhecimento contínuo de sessão. Em 2010 a IBM publicou a versão 3.1 do protocolo, a primeira livre de *royalty*, e em 2014 foi pradronizado pela *Organization for the Advancement of Structured Information Standards (OASIS)* com o lançamento da versão 3.1.1. (HIVEMQ, 2016)

A arquitetura do protocolo MQTT é baseada em tópicos, onde um dispositivo publica uma mensagem em um determinado tópico e os dispositivos inscritos nesse tópico recebem a mensagem. Para isso possui dois atores:

- *Client*: dispositivos que coletam dados de telemetria e publicam as informações em tópicos. Também podem se inscrever em tópicos para receber as mensagens publicadas por outros dispositivos.

- *Broker*: servidor responsável por receber as publicações, filtrar as mensagens e encaminhar para os dispositivos inscritos no tópico em que a mensagem foi publicada. Também é responsável pelo controle de acesso dos dispositivos.

Nesse modelo de comunicação um dispositivo não precisa conhecer nenhuma informação sobre os demais dispositivos que irão receber suas mensagens, nem mesmo saber da existência deles, basta conhecer os parâmetros de conexão com o *broker* e os tópicos que deverá publicar suas mensagens. Dessa forma diversos dispositivos que acessam a internet através de roteadores com o recurso de *network address translation* (NAT), onde endereços de rede locais são traduzidos para um mesmo endereço público na internet, podem utilizar o protocolo MQTT para enviar e receber mensagens sem nenhuma configuração adicional no roteador para redirecionamento dos pacotes, uma vez que todo o processo de comunicação é iniciado pelo próprio dispositivo em direção ao *broker* e recebe as mensagens através da conexão estabelecida.

Para estabelecer conexão com o *broker*, o *client* envia um pacote do tipo ***CONNECT*** contendo sua identificação, parâmetros de autenticação, *Will Message* (mensagem que deverá ser enviada aos demais dispositivos em caso de desconexão inesperada), *Keep Alive* (tempo máximo de inatividade, quando o *client* deverá enviar um comando do tipo *ping* para o *broker* para confirmar atividade caso nenhuma mensagem seja enviada no intervalo) e *Clean Session* (indicador se a sessão deverá ser limpa ou não no final da conexão, ou seja, se os tópicos inscritos deverão persistir na próxima conexão e se as mensagens retidas deverão ser entregues). O *broker* responde com uma mensagem do tipo ***CONNACK*** para confirmar a conexão.

Ao publicar uma mensagem, o *client* envia um pacote do tipo ***PUBLISH***, que contem basicamente o nome do tópico que irá publicar (uma *string* com a hierarquia separada por barras), o nível de qualidade de serviço (QOS) e o *payload*, que trata-se da mensagem propriamente dita. Como o protocolo MQTT é *data-agnostic*, qualquer tipo de dado pode ser enviado, mas os mais comuns são *JSON (JavaScript Object Notation*, formatação de mensagens em formato texto composta coleção de pares/valor em uma lista ordenada), texto e binário. O *broker* não envia nenhum pacote de confirmação.

O pacote ***SUBSCRIBE*** é utilizado para inscrição em tópicos, especifica basicamente um ou mais tópicos para inscrição e o nível de qualidade de serviço (QOS) que deverá ser utilizado. Toda inscrição é confirmada pelo *broker* com o retorno de um pacote do tipo ***SUBACK***.

Outro pacote importante é o ***UNSUBSCRIBE*** para cancelar a assinatura de tópicos. O *broker* responde com um pacote do tipo ***UNSUBACK*** confirmando a operação.

Demais parâmetros e a especificação completa dos pacotes pode ser encontrada na documentação oficial do protocolo: (STANDARD, 2014)

2.1.2 Siemens S7

Siemens S7 *communication* é um protocolo de comunicação ethernet proprietário da empresa Siemens utilizado em CLPs (controladores lógicos programáveis) da linha S7 de sua fabricação. Sua utilização se dá para comunicação entre CLPs, acesso a informações do CLP por softwares supervisórios (SCADA), programação e execução de diagnósticos do controlador.

Segundo Nardella (2016), o protocolo S7 é orientado a função/comando, cada transmissão contém um comando ou uma réplica a um comando:

- Leitura/escrita de dados;
- Leitura/escrita cíclica;
- Informações de diretório;
- Informações do sistema;
- Transferência de blocos de programação;
- Controle do PLC (start/stop);
- Data e hora;
- Segurança;
- Programação/configuração.

A estrutura de comunicação do protocolo possui 3 elementos:

- Cliente (client): pode apenas requisitar uma informação
- Servidor (server): pode apenas responder uma requisição
- Par (partner/peer): pode requisitar, pode responder e pode enviar uma informação por sua própria iniciativa

A implementação do protocolo S7 em TCP/IP utiliza o serviço de transporte orientado a bloco ISO TCP (RFC1006). A tabela 1 indica os protocolos utilizados para comunicação S7 segundo o modelo OSI.

O protocolo também pode ser utilizado com outras camadas físicas e de rede, como RS-485 com MPI (*Multi-Point-Interface*)ou Profibus.

Wiens (2016) cita que para estabelecer conexão com um CLP utilizando o protocolo S7 em TCP/IP, são seguidos três passos:

Tabela 1 – Modelo OSI para o protocolo Siemens S7

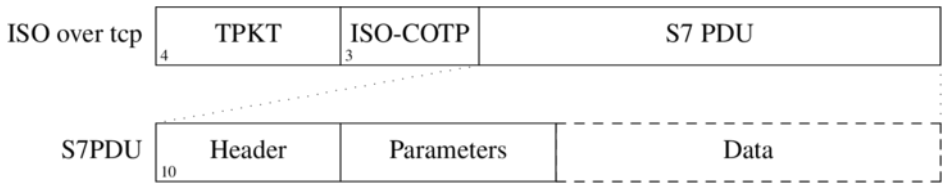
Camada OSI	Protocolo
7 - Aplicação	*S7 communication*
6 - Apresentação	*S7 communication*
5 - Sessão	*S7 communication*
4 - Transporte	ISO TCP (RFC1006)
3 - Rede	IP
2 - Enlace	*Ethernet*
1 - Física	*Ethernet*

Fonte: Adaptado de (WIENS, 2016)

- 1 - Conexão na porta TCP 102 do CLP
- 2 - Conexão na camada ISO
- 3 - Conexão na camada *S7 communication*

Cada bloco no protocolo é nomeado PDU (*Protocol Data Unit*), conforme estrutura apresentada na figura 2, seu tamanho depende do processador de comunicação (CP) do CLP utilizado e é negociado durante a conexão.

Figura 1 – Pacote Siemens S7



Fonte: (MIRU, 2016)

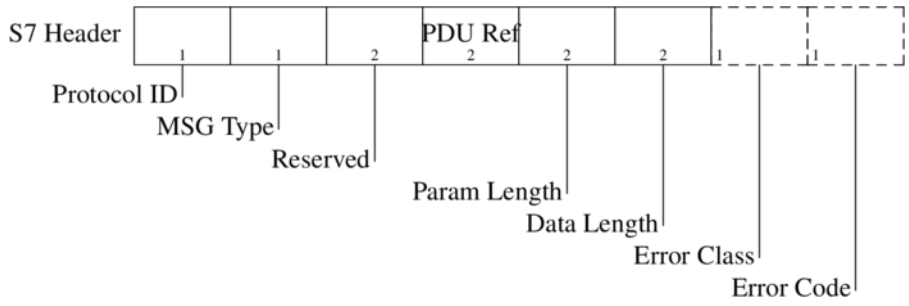
O PDU é composto por três partes principais:

- *Header*: contém o comprimento e referência do pacote, além do tipo de mensagem.
- *Parameters*: o conteúdo e estrutura do campo variam bastante com o tipo de função.
- *Data*: campo opcional que traz os dados para a função se existirem (com por exemplo, áreas de memória)

2.1.3 MODBUS TCP

MODBUS é um protocolo de transmissão desenvolvido no final da década de 1970 pela *Gould Modicon* (hoje *Schneider Electric*) para sistemas de controle de processos. Ao

Figura 2 – Formato do PDU Siemens S7



Fonte: (MIRU, 2016)

contrário de outros protocolos, não é definido um padrão físico (camada 1 do modelo OSI) para o MODBUS, o que torna possível a sua implementação de diferentes maneiras. (MACKAY, 2004)

Em 2014 a *Schneider Electric* transferiu os direitos do protocolo para a *Modbus Organization*, as especificações do MODBUS estão disponíveis para livre acesso via *download* e nenhuma taxa é requerida para sua utilização. (ORGANIZATION, entre 2005 e 2017)

Por ser simples, flexível e aberto, o MODBUS é um dos protocolos mais utilizados em automação industrial, principalmente para garantir a inter-operabilidade entre equipamentos de diferentes fabricantes. Os usuários de redes de automação geralmente estão cientes de que especificar a necessidade de compatibilidade com o protocolo é uma das formas mais eficientes de garantir a integração dos equipamentos com o menor custo possível. (MACKAY, 2004)

Outra liberdade do protocolo MODBUS está no modo de transmissão das mensagens, que pode ser:

- ASCII: mensagem "legível"no formato ASCII, utilizado principalmente para testes.
- RTU: mensagem em formato hexadecimal, normalmente utilizado em ambiente de produção por ser rápido e compacto.

A estrutura padrão de um pacote MODBUS em modo RTU está ilustrada na figura 3.

Figura 3 – Formato padrão do *frame* MODBUS em modo RTU

Address field	Function field	Data field	Error check field
1 byte	1 byte	Variable	2 bytes

Fonte: (MACKAY, 2004)

Tabela 2 – Códigos de função do protocolo MODBUS

Código da função	Descrição
1	Leitura de bloco de bits do tipo coil(saída discreta)
2	Leitura de bloco de bits do tipo entradas discretas
3	Leitura de bloco de registradores do tipo holding
4	Leitura de bloco de registradores do tipo input
5	Escrita em um único bit do tipo coil(saída discreta)
6	Escrita em um único registrador do tipo holding
7	Ler o conteúdo de 8 estados de exceção
8	Prover uma série de testes para verificação da comunicação e erros internos
11	Obter o contador de eventos
12	Obter um relatório de eventos
15	Escrita em bloco de bits do tipo coil(saída discreta)
16	Escrita em bloco de registradores do tipo holding
17	Ler algumas informações do dispositivo
20	Ler informações de um arquivo
21	Escrever informações em um arquivo
22	Modificar o conteúdo de registradores de espera através de operações lógicas
23	Combina ler e escrever em registradores numa única transação
24	Ler o conteúdo da fila FIFO de registradores
43	Identificação do modelo do dispositivo

Fonte: (FREITAS, 2014)

Em uma requisição de um mestre (cliente) para um escravo (servidor), o primeiro campo campo contém o endereço do escravo para o qual a requisição é feita. O segundo campo contém o código da função, conforme tabela 2. O terceiro campo contém a informação necessária para completar a requisição, como um endereço para leitura. Já o último campo é utilizado para validação da integridade da mensagem. Ao responder uma requisição, o escravo preenche o primeiro campo do pacote com o seu próprio endereço, e terceiro campo traz informação requisitada. Os demais campos são preenchidos da mesma maneira que é feita na solicitação.

O protocolo MODBUS possui seu modelo de dados baseado em uma séria de tabelas com características distintas, as quatro primárias são apresentadas na tabela 3.

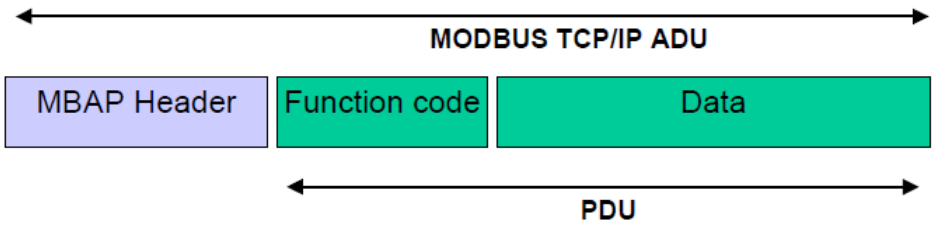
Tabela 3 – Tabelas primárias do modelo de dados do protocolo MODBUS

Tabela primária	Tipo de dados	Tipo de acesso	Comentários
Discretes Input	*bit*	somente leitura	entradas digitais
Coils	*bit*	leitura e escrita	saídas digitais
Input Registers	*16-bit word*	somente leitura	entradas analógicas
Holding Registers	*16-bit word*	leitura e escrita	saídas analógicas

Fonte: Adaptado de (ORGANIZATION, 2012)

MODBUS TCP é uma implementação do protocolo em meio físico *ethernet* utilizando uma pilha TCP/IP, o frame é apresentado na figura 4.

Figura 4 – Formato padrão do *frame* MODBUS TCP/IP



Fonte: (ORGANIZATION, 2006)

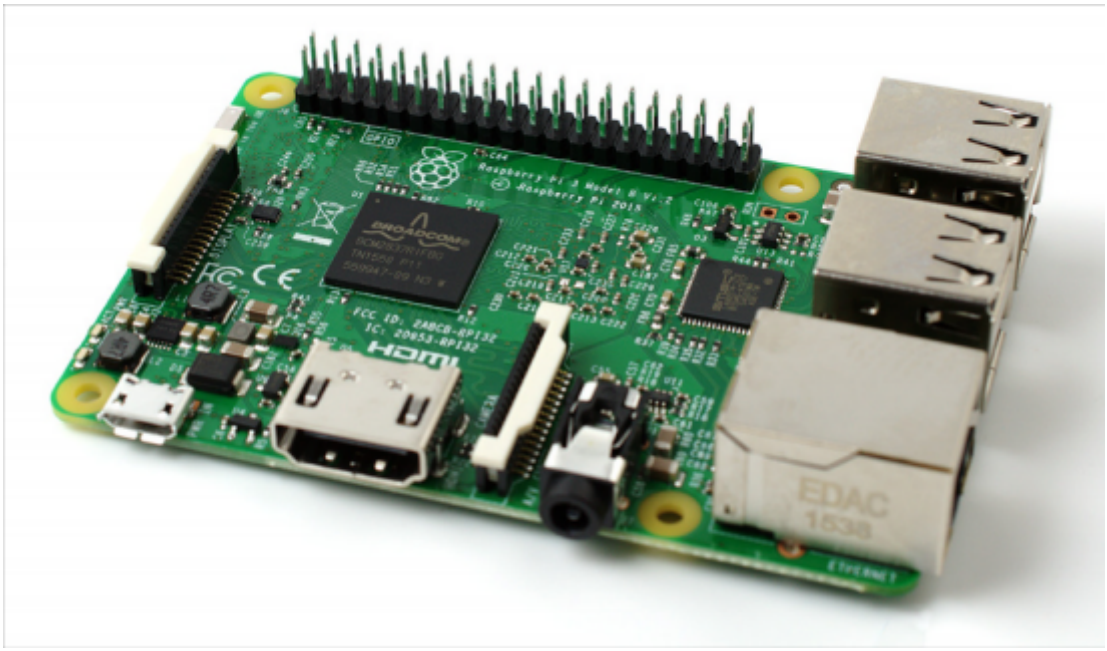
Em relação ao *frame* MODBUS convencional, o *adress field* que contém o endereço do escravo é substituído pelo cabeçalho MBAP com 7 *bytes*, o conceito de *slave address* é substituído pelo *unit identifier* de um *byte* utilizado para identificar uma unidade MODBUS em comunicações através de pontes, roteadores e *gateways*, permitindo que um único endereço IP possa representar diversos dispositivos MODBUS. A verificação de erros já é feita por padrão em *frames ethernet* através de CRC-32, tornando muito difícil a não identificação de erro em uma mensagem, dessa forma não é necessário utilizar um campo específico para isso no ADU (Application Data Unit).

2.2 Hardware Raspberry PI

Raspberry Pi é um pequeno computador *general-purpose* originalmente concebido para incentivar crianças a aprender programar. Os primeiros desenvolvimentos começaram com Eben Upton durante seus trabalhos na Universidade de Cambridge, onde construiu placas simples com processamento semelhante aos computadores de 8 *bits* originais. Concluindo que isso não seria suficiente para captar a imaginação de crianças acostumadas com equipamentos eletrônicos de última geração, Upton fundou a Raspberry Pi Foundation ao lado de outros amigos entusiastas e especialistas em tecnologia ao se transferir da universidade para a industria. Em 2012 chegaram ao mercado as primeiras versões da Raspberry Pi, com tamanho aproximado de um cartão de crédito o computador possuía porta HDMI para rápida conexão em televisores, portas USB para conexão com dispositivos padrão (por exemplo teclado e mouse), cartão de memória SD para armazenamento de dados e conector de energia do tipo micro USB (muito utilizado em carregadores de celular). Atualemente existem seis versões principais da Raspberry Pi: A+, B, 2, Zero, Zero W e 3. A Raspberry Pi 3 traz como novidade processador Broadcom BCM2837 de 64 bits, rádio wireless 2.4GHz integrado com suporte a Wi-Fi e Bluetooth, conector GPIO de 40

pinos, 4 portas USB, porta de rede cabeada 10/100 Mbps. Sua memória RAM é de 1GB (mesma quantidade do modelo 2). (UPTON; HALFACREE, 2016)

Figura 5 – Placa Raspberry Pi 3



Fonte: (UPTON; HALFACREE, 2016)

2.3 Segurança em soluções IoT

Em soluções conectadas em rede ou na internet a segurança é um fator fundamental, em internet das coisas isso não é diferente.

Segundo Shipley (2013), não existe um consenso sobre a melhor maneira de implementar segurança em soluções IoT, mas técnicas testadas e comprovadas de segurança em tecnologia da informação tradicional podem ser adaptadas às características de dispositivos embarcados (baixa capacidade de processamento e memória, baixo consumo de energia e conectividade limitada).

Dessa forma, Shipley (2013) ainda sugere algumas práticas:

- Inicialização segura: quando o dispositivo é inicializado (*boot*) a autenticidade e integridade de seu software devem ser verificadas através de assinaturas digitais criptografadas.

- Controle de acesso: o sistema operacional deve limitar os privilégios de componentes e aplicações para acessar somente os recursos que necessitam para o seu trabalho. Assim um recurso comprometido não compromete a segurança de todo o sistema.

- Autenticação de dispositivo: o dispositivo deve se autenticar ao ser conectado em uma rede antes de receber ou transmitir dados.

- *Firewall*: o dispositivo deve possuir recursos de software para filtrar pacotes de protocolos específicos de sua aplicação, pois os *firewalls* tradicionais estão preparados e encarregados de filtrar os pacotes de protocolos padrões de TI.

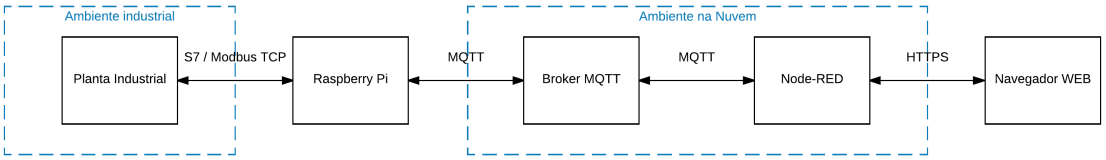
- Atualizações e correções: um dispositivo em operação não deve receber pacotes de correção ou atualização de forma automática, deve sempre partir do usuário ou administrador o início da ação e os dispositivos devem verificar a autenticidade e integridade dos pacotes submetidos antes de realizar o procedimento. Com isso não há o consumo desnecessário de banda na rede e a funcionalidade do dispositivo não é interrompida em momentos não planejados ou previstos.

Capítulo 3

Desenvolvimento do Gateway IoT

O objetivo do trabalho é desenvolver um *gateway* de internet das coisas para automação industrial, dessa forma são utilizadas diversas ferramentas de *software* e *hardware*, conforme diagrama da figura 6, para permitir a interação do usuário final com uma planta industrial através de um navegador *WEB*.

Figura 6 – Diagrama do *gateway*



Fonte: elaborada pelo autor

3.1 Sistema Operacional *Raspbian*, *softwares* utilizados e configurações da Raspberry Pi

Raspbian é um sistema operacional de código aberto baseado na distribuição *Linux Debian* de arquitetura ARM otimizada para o hardware da Raspberry Pi. Seu repositório conta com mais de 35.000 pacotes pré-compilados para fácil instalação. (RASPBIAN, entre 2012 e 2017)

Debian é uma das mais antigas e ativas distribuições *Linux*, seu foco é em estabilidade, alta compatibilidade, e excelente performance mesmo em hardwares modestos, o que torna um ótimo parceiro para a *Raspberry Pi*. (UPTON; HALFACREE, 2016)

Para o trabalho foi utilizada a versão *RASPBIAN JESSIE WITH DESKTOP* de 05/07/2017, que já possui ambiente gráfico e uma série de aplicativos necessários para esse desenvolvimento já instalados, como os interpretadores *Python 2* e *Python 3*, ambiente

de desenvolvimento *Python IDLE* e a ferramenta de acesso remoto *Real VNC Server*. Adicionalmente foi instalado o software *Team Viewer* para permitir acesso remoto sem a necessidade de redirecionamento de portas específicas.

A placa de rede cabeada é utilizada para conexão com a rede de automação industrial e a placa de rede *wireless* usada para conexão com a internet, já que em geral ambientes de automação industrial são isolados da internet para garantir maior segurança contra ataques externos. Uma alternativa é utilizar um placa de rede *ethernet* adicional via porta USB para conexão à internet por rede cabeada e conseguir uma maior performance.

3.2 *Broker MQTT* com *IBM Watson IoT platform*

IBM Bluemix é uma plataforma em computação na nuvem criada pela companhia americana de tecnologia IBM (International Business Machines Corporation) em 2014 e ofertada para o público na modalidade Paas (Platform-as-a-Service, plataforma como serviço na tradução do inglês). (TOMALA-REYES, 2014)

Seus principais atrativos são o acesso instantâneo a todas as tecnologias e a alta escalabilidade da oferta de serviços, ou seja é possível iniciar com um serviço disponibilizado para um único usuário e em questão de minutos expandir a infraestrutura para atender milhões de usuários para a mesma aplicação.

Ao todo estão disponíveis mais de 130 serviços, como bancos de dados, servidores de aplicação, gateways para internet das coisas, back-end para aplicativos móveis, computação cognitiva e diversas outras tecnologias.

Nessa gama de serviços está inserido o *IBM Watson IoT platform*, que reúne *broker* MQTT, interface para gerenciamento de dispositivos IoT (controle de ID, tipo e chave de acesso), gerenciamento de nível de acesso de aplicativos, monitoramento de dados trafegados, regras para alertas sobre dispositivos, integração com banco de dados externo para armazenamento de dados históricos, e *dashboard* em formato de placas e cartões para trazer dados e indicadores dos dispositivos de forma gráfica ao administrador do sistema.

Os tópicos no *IBM Watson IoT platform* seguem o seguinte formato:

"iot-2/evt/event_id/fmt/format_string", onde "event_id"corresponde à identificação do evento (no projeto temos os eventos que indicam as entradas, saídas e variáveis auxiliares dos CLPs).

Dispositivos devem conectar utilizando nome de usuário "use-token-auth"e como senha o *token* de acesso gerado no momento de criação do dispositivo. Já aplicativos devem autenticar-se utilizando nome de usuário no padrão "a/orgId/APIkey", onde "orgId"corresponde ao identificador da organização e "APIkey"corresponde à chave da aplicação gerada automaticamente no momento da criação das credenciais de aplicação, bem como o *token* de acesso utilizado no campo senha.

Para o gateway desenvovido neste trabalho são utilizados os recursos de *broker* MQTT, gerenciamento de dispositivos (criação do tipo "CLP"e de três dispositivos nomeados "s71200", "temperatura", "nivel"e "vazao") e a geração de uma chave de acesso do tipo "Aplicativo Padrão"para acesso dos códigos desenvolvidos em *python* ao serviço. Os tópicos do *broker* são criados automaticamente quando é feita a primeira publicação com determinado nome de tópico para cada dispositivo.

3.3 Biblioteca *Eclipse Paho MQTT*

O projeto *Eclipse Paho* foi criado para prover implementações de código aberto e escaláveis de protocolos abertos e padrões voltados para as novas, existentes e emergentes aplicações de máquina-para-máquina (M2M) e internet das coisas (IoT). Inicialmente foi realizada a implementação de um *client* MQTT para uso em plataforma embarcadas e futuramente trará suporte a funções de servidor (FOUNDATION, 2015). O projeto possui suporte ativo da empresa IBM.

3.3.1 Importação da biblioteca no ambiente *Python* e construção da instância *client*

```
1 import paho.mqtt.client as mqtt
2
3 client = mqtt.Client(client_id)
4 client.username_pw_set(username, password)
5 client.on_message=on_message
6 client.connect(host, port, keepalive, bind_address)
```

Onde:

- ***client_id***: identificar exclusivo do *client* durante a conexão. Se o parâmetro for omitido será gerado deforma randômica.
- ***username***: nome de usuário para conexão.
- ***password***: senha para conexão.
- ***on_message***: método a ser executado quando uma mensagem for recebida.
- ***host***: endereço IP ou *hostname* do *broker* remoto, dados do tipo ***string***
- ***port***: número da porta utilizada pelo serviço, dados do tipo **inteiro**
- ***keepalive***: máximo tempo (em segundos) entre comunicações com o *broker*, se não houver comunicação no intervalo o *client* enviará mensagens de *ping* para o *broker*

- ***bind_address***: endereço IP da interface de rede para associar o *client* quando o dispositivo possui múltiplas interfaces.

3.3.2 Publicação de mensagens em tópicos

```
1 client.publish(topic, payload, qos, retain)
```

Onde:

- ***topic***: tópico do *broker* no qual a mensagem será publicada, dados do tipo ***string***.
- ***payload***: mensagem a ser publicada, valor do tipo ***JSON*** ou ***string***.
- ***qos***: nível de qualidade de serviço, valor do tipo **inteiro** (0, 1 ou 2).
- ***retain***: flag para retenção ou não da mensagem no tópico, valor do tipo **bool**

3.3.3 Assinatura de tópicos

Assinatura de um único tópico:

```
1 client.subscribe(topic, qos)
```

Onde:

- ***topic***: tópico do *broker* no qual a mensagem será publicada, dados do tipo ***string***.
- ***qos***: nível de qualidade de serviço, valor do tipo **inteiro** (0, 1 ou 2).

Assinatura de múltiplos tópicos com um único comando:

```
1 client.subscribe([(topic1, qos1),(topic2, qos2)])
```

Nesse caso no primeiro parâmetro é passada uma lista de valores do tipo *tuple* contendo os tópicos para inscrição e seus respectivos níveis de qualidade de serviço (*qos*). O segundo parâmetro é omitido.

Após a inscrição em um tópico o *client* passará a receber as respectivas mensagens e irá disparar o método definido no parâmetro *on_message* especificado na contrução da instância. Abaixo segue um exemplo de método para apresentar o conteúdo e informações de mensagens recebidas:

```
def on_message(client, userdata, message):
        print("Menssagem recebida: " , str(message.payload.decode("utf-8")))
        print("Tópico da menssagem: ", message.topic)
        print("Nível de qualidade de serviço (QOS): ", message.qos)
        print("Retenção: ", message.retain)
```

3.4 Biblioteca *Snap7*

A biblioteca *Snap7* é uma *suite* de código aberto e multiplataforma de comunicação ethernet para interface nativa com CLPs Siemens utilizando protocolo S7. Possui compatibilidade com as linguagens $C/C++$, $.NET/Mono$, Pascal, LabVIEW, Phyton, Node.js e com as arquiteturas $i386/x86_64$, ARM, Sun Sparc e Mips, é altamente escalável e permite a integração de sistemas baseados em PC com automação industrial através de três componentes especializados: *client*, *server* e *partner*. (NARDELLA, 2016)

Por padrão todo CLP Siemens S7 opera como um servidor S7 e permite acesso às áreas de memória independente do programa que estiver em execução na CPU. Dessa forma o *gateway* desenvolvido atua como *client* no ambiente de comunicação S7 para permitir a leitura de entradas e saídas, bem como leitura e escrita de variáveis auxiliares sem que haja a necessidade de uma programação específica no CLP.

Importação da biblioteca no ambiente *Python*:

```
import snap7
from snap7.snap7types import *
```

Criação de duas classes *client* (uma para leitura, outra para escrita) e conexão com CLP S7-1200:

```
clp_r = snap7.client.Client()
clp_r.connect(ip, rack, slot)
clp_w = snap7.client.Client()
clp_w.connect(ip, rack, slot)
```

Onde:

- ***ip***: endereço IP do CLP, dados do tipo *string*
- ***rack***: número do *rack* configurado na programação do CLP
- ***slot***: número do *slot* configurado na programação do CLP

Leitura e escrita de dados da memória:

```
leitura = clp_r.read_area(area, dbnumber, start, size)
clp_w.write_area(area, dbnumber, start, data)
```

Onde:

- ***area***: área de memória que será acessada. Pode ser especificada pelo endereço ou constante conforme a tabela 4. Valor do tipo **inteiro**.
- ***dbnumber***: número do bloco de dados, para as demais áreas deve ser utilizado o valor 0. Valor do tipo **inteiro**.
- ***start***: *offset* do endereço para o início do acesso. Valor do tipo **inteiro**.
- ***size***: quantidade de *bytes* de memória que serão acessados. Valor do tipo **inteiro**.
- ***data***: dados que serão escritos na memória. Valor do tipo ***bytearray***.
- ***leitura***: dados retornados na leitura da memória. Valor do tipo ***bytearray***

Tabela 4 – Endereços e constantes para áreas de memória com biblioteca *Snap7*

Área	Endereço	Constante
Entradas de processo	0x81	*S7AreaPE*
Saídas de processo	0x82	*S7AreaPA*
Variáveis auxiliares	0x83	*S7AreaMK*
Blocos de dados	0x84	*S7AreaDB*
Contadores	0x1C	*S7AreaCT*
Temporizadores	0x1D	*S7AreaTM*

Fonte: Adaptado de (NARDELLA, 2016)

3.4.1 Tratando *arrays, bytes e bits*

Importação das demais bibliotecas necessárias:

```
import json
import numpy
import math
import struct
```

Na leitura de áreas de memória com a função *read_area* os dados retornados são do tipo *bytearray*, onde temos um vetor de *bytes* de oito *bits*. Para trabalharmos com informações dos tipos destacados na tabela 5, precisamos tratar os dados.

Tabela 5 – Tipos de dados para CLP Siemens S7

Variável	Tipo
Entradas digitais	*bit*
Saídas digitais	*bit*
Variáveis auxiliares	*bit*
Entradas analógicas	*word (16 bits)* ou *dword (32 bits)*
Saídas analógicas	*word (16 bits)* ou *dword (32 bits)*

Para dados do tipo *bit*, podemos converter o *bytearray* para uma matriz de *bits* através das funcionalidades da biblioteca *numpy*. No exemplo abaixo são lidas as 14 entradas digitais do CLP, o que correspondem a 2 *bytes* de memória, na variável *in_byte_array*. Na sequência os dados são convertidos para o tipo *numpy.ndarray*, onde teremos um vetor de *bytes* de oito *bits* expecificado pelo parâmetro *numpy.uint8*, e gravados na variável *in_byte_nparray*. Finalmente é feita a conversão para uma matriz de *bits* com duas linhas e oito colunas (parâmetro *reshape(2,8)*) e os dados são gravados na variável *in_bit_nparray*:

```
1 in_byte_array = clp_r.read_area(S7AreaPE, 0, 0, 2)
2 in_byte_nparray = numpy.frombuffer(in_byte_array, dtype=numpy.uint8)
3 in_bit_nparray = numpy.unpackbits(in_byte_nparray).reshape(2,8)
```

Comparação dos tipos de dados:

```
1 >>> print(in_byte_array)
2 bytearray(b'\x01\x00')
3 >>> print(in_byte_nparray)
4 [1 0]
5 >>> print(in_bit_nparray)
6 [[0 0 0 0 0 0 0 1]
7 [0 0 0 0 0 0 0 0]]
```

No código acima podemos observar as diferenças entre os tipos de dados utilizados. No exemplo apenas a primeira entrada digital do CLP (I0.0) estava ativa, como pode ser observado pelo valor 1 na posição (1,8) da matriz *in_bit_nparray*.

Para dados do tipo *word (16 bits)* e *dword (32 bits)* podemos seguir o mesmo procedimento e converter os dados lidos para o tipo *numpy.ndarray* especificando o tipo de dados pelo parâmetro *numpy.uint16* ou *numpy.uint32*. No exemplo abaixo é feita a leitura da entrada analógica IW64. Como o CLP trabalha com o formato de dados *big-endian* e o sistema operacional utilizado (*Raspbian*) utiliza o formato *litle-endian*, é necessário inverter a sequência de *bytes* através do comando *byteswap()*.

```
IW64_byte_array = clp_r_n.read_area(S7AreaPE, 0, 64, 2)
IW64_byte_nparray = numpy.frombuffer(IW64_byte_array, dtype=numpy.uint16).byteswap()

>>> print(IW64_byte_array)
bytearray(b'\x08\x02')
>>> print(IW64_byte_nparray)
[2050]
```

Para gravação de dados na memória do CLP, também é necessário trabalhar com dados do tipo *bytearray*, o exemplo abaixo mostra como converter um vetor de bits para o formato exigido pela biblioteca *Snap7* e a gravação na área de memória correspondente às variáveis auxiliares do CLP:

```
bit_array = [0, 0, 0, 0, 1, 0, 0, 1, 1, 0]

byte_array = [0,0]

i = 0
j = 0

while (i<10):
if (i<8):
byte_array[0] = int(byte_array[0] + (bit_array[i] * math.pow(2, i)))
else:
byte_array[1] = int(byte_array[1] + (bit_array[i] * math.pow(2, i-8)))
i = i+1

data = struct.pack('BB', byte_array[0], byte_array[1])

clp_w.write_area(S7AreaMK, 0, 0, data)

>>> print(bit_array)
[0, 0, 0, 0, 1, 0, 0, 1, 1, 0]
>>> print(byte_array)
[144, 1]
>>> print(data)
b'\x90\x01'
```

O código contém um ciclo *while* para varredura do vetor de *bits* e conversão dos valores binários para decimais arranjados em um vetor de dois inteiros com os valores dos *bytes* para a área de memória do CLP. Na sequência os dados são convertidos em *bytearray* com o comando *struct.pack*. Para variáveis do tipo *word* e *dword* basta preencher o vetor de inteiros com uma palavra em cada posição e especificar o tamanho corretamente conforme documentação da biblioteca *python struct* disponível em (FOUNDATION, 2017b).

3.5 Biblioteca *pymodbus*

A biblioteca *pymodbus* é uma implementação completa do protocolo *Modbus* em python e pode ser utilizada sem nenhuma outra dependência externa quando projetos mais leves são requeridos. Possui as classes *client* e *server* e pode operar nas variantes *TCP, UDP, Serial ASCII, Serial RTU, e Serial Binary* do protocolo nos modos síncrono e assíncrono. (RIPTIDEIO, 2017)

Alguns CLPs já operam como padrão na função de servidor *Modbus TCP*, como é caso do CLP Fertron Citrino segundo informações do manual (FERTRON, 2007). Dessa forma o *gateway* desenvolvido atua como *client Modbus TCP* para a comunicação com esse tipo de equipamento sem a necessidade de uma programação específica do CLP. Já equipamentos como o Siemens S7-1200 requerem a adição de um bloco de programação para executar a função servidor *Modbus TCP*.

Importação da biblioteca no ambiente *Python* para utilização da classe *client* no modo síncrono:

```
1 from pymodbus.client.sync import ModbusTcpClient
2 clp = ModbusTcpClient(ip, port)
```

Onde:

- ***ip***: endereço IP do CLP, dados do tipo *string*
- ***port***: número da porta do serviço *Modbus TCP* no servidor (CLP), dados do tipo inteiro.

Leitura de saídas digitais (*coils*), entradas digitais (*input registers*), saídas analógicas (*holding registers*) e entradas analógicas (*input registers*):

```
1 do = clp.read_coils(start, count, unit)
2 di = clp.read_discrete_inputs(start, count, unit)
3 ao = clp.read_holding_registers(start, count, unit)
4 ai = clp.read_input_registers(start, count, unit)
```

Onde:

- ***start***: endereço de memória para início da leitura, dados do tipo **inteiro**.
- ***count***: número de registros a serem lidos, dados do tipo **inteiro**.
- ***unit***: identificador da unidade (equipamento) que será acessado, dados do tipo **inteiro**.

Os valores retornados nas leituras (**do, di, ao, ai**) são estruturas que possuem o status da operação e os valores lidos em vetores, acessíveis através do campo *bits* para variáveis discretas (no exemplo *do.bits* e *di.bits*) e *registers* para variáveis analógicas (no exemplo *ao.registers* e *ai.registers*).

A classe *pymodbus* possui as funções *write_coils* e *write_registers* para escrita em saídas digitais (*coils*) e registradores (*holding registers*) exemplificadas abaixo:

```
1 coils_data = [1, 0, 0]
2 registers_data = [10, 20, 30]
3
4 start = 0
5
6 clp.write_coils(start, coils_data, unit=0)
7 clp.write_registers(start, registers_data, unit=0)
```

Onde:

- ***start***: endereço de memória para início da escrita, dados do tipo **inteiro**.
- ***coils_data***: registros discretos a serem escritos, dados do tipo **lista de inteiros**.
- ***registers_data***: registros a serem escritos, dados do tipo **lista de inteiros**.
- ***unit***: identificador da unidade (equipamento) que será acessado, dados do tipo **inteiro**.

3.6 Painel de controle com *Node-RED*

Após a integração dos CLPs com o *broker MQTT* é necessário permitir a interação do usuário com a planta industrial, para isso pode ser implementada uma *dashboard* em Node-RED.

Node-RED é uma ferramenta de programação para conexão de dispositivos de hardware, *APIs* e serviços online em novas e interessantes formas. Fornece um editor baseado em navegador *web* que torna fácil a criação de fluxos através da conexão entre a grande quantidade de nós disponíveis na biblioteca. (FOUNDATION, 2017a)

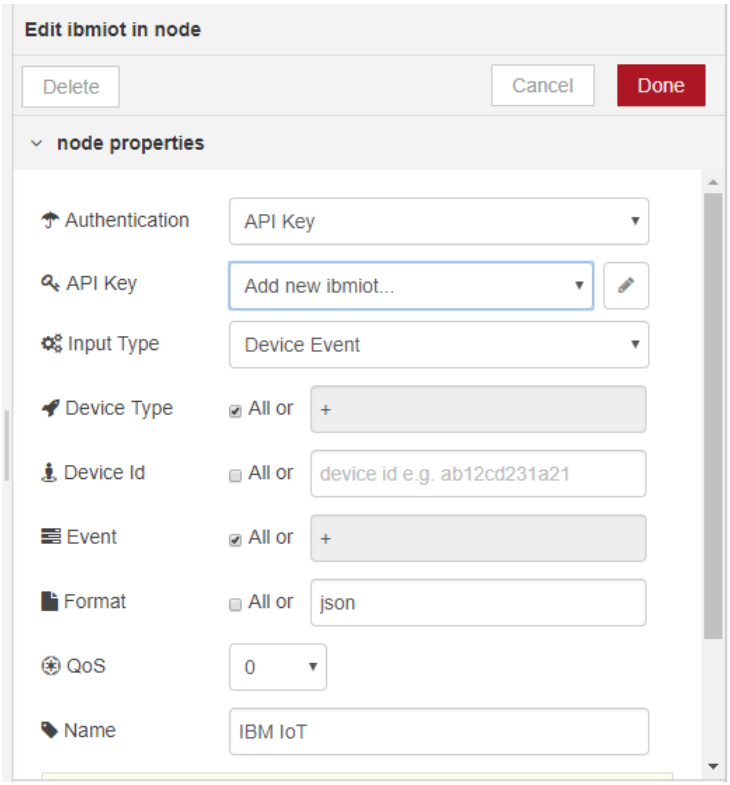
As figuras 7 a 14 apresentam detalhes dos nós utilizados no projeto.

- ***IBM IoT In***: nó para receber informações do serviço *IBM Watson IoT Platform* através do protocolo MQTT. Os parâmetros de configuração incluem o tipo de dispositivo, identificação de dispositivo, tipo de evento e formato de dados.

Figura 7 – Node-RED nó *IBM IoT In*

Fonte: elaborada pelo autor

Figura 8 – Configurações do nó *IBM IoT In*



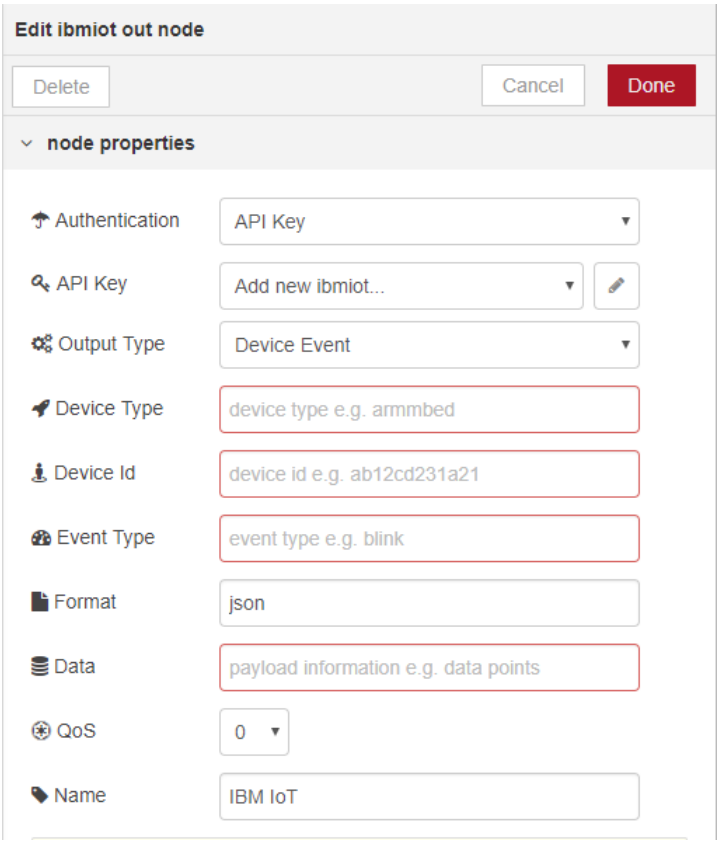
Fonte: elaborada pelo autor

- ***IBM IoT Out***: nó para enviar informações para serviço *IBM Watson IoT Platform* através do protocolo MQTT. Os parâmetros de configuração incluem o tipo de dispositivo, identificação de dispositivo, tipo de evento e dados a serem enviados e formato.

Figura 9 – Node-RED nó *IBM IoT Out*

Fonte: elaborada pelo autor

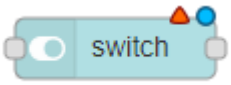
Figura 10 – Configurações do nó *IBM IoT Out*



Fonte: elaborada pelo autor

- ***Dashboard Switch***: botão deslizante utilizando em *dashboard* para controle de variáveis binárias.

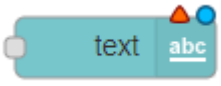
Figura 11 – Node-RED nó *Dashboard Switch*



Fonte: elaborada pelo autor

- ***Dashboard Text***: campo para apresentação de textos em *dashboard*.

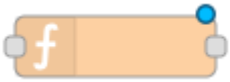
Figura 12 – Node-RED nó *Dashboard Text*



Fonte: elaborada pelo autor

- ***JavaScript Function***: bloco para utilização de *scritps* em linguagem *JavaScript*.

Figura 13 – Node-RED nó *JavaScript Function*



Fonte: elaborada pelo autor

- ***Json***: nó para conversão de tipos entre *strings JSON* e objetos *JavaScript*.

Figura 14 – Node-RED nó *Json*

Fonte: elaborada pelo autor

3.7 Caixa para instalação da placa *Raspberry Pi* em painel de automação industrial

Para tornar o *gateway* desenvolvido preparado para aplicações industriais, a placa Raspberry Pi foi acomodada em uma caixa modular do fabricante alemão *Phoenix Contact*, modelo *RPI-BC 107,6 DEV-KIT KMGY*, em conformidade com a norma DIN 43880.

A caixa permite a montagem em trilhos do padrão DIN, facilitando a instalação em ambientes industriais. Além de acomodar a Raspberry Pi, o produto permite fácil acesso aos pinos GPIO através de uma tampa na superfície e também permite a conexão com outras caixas da mesma linha de produtos do fabricante através de conectores *HBUS DIN*.

Figura 15 – Raspberry Pi utilizada no trabalho instalada em uma caixa para trilhos DIN

Fonte: elaborada pelo autor

3.8 Painel para controle remoto de um CLP Siemens S7-1200

Para testar as funcionalidades do *gateway* de envio bidirecional de informações entre o ambiente industrial e o ambiente IoT, foi realizada uma implementação das ferramentas

estudadas em conjunto com um CLP Siemens S7-1200 disponível no LAI (Lanoratório de Automação Industrial) da EESC/USP.

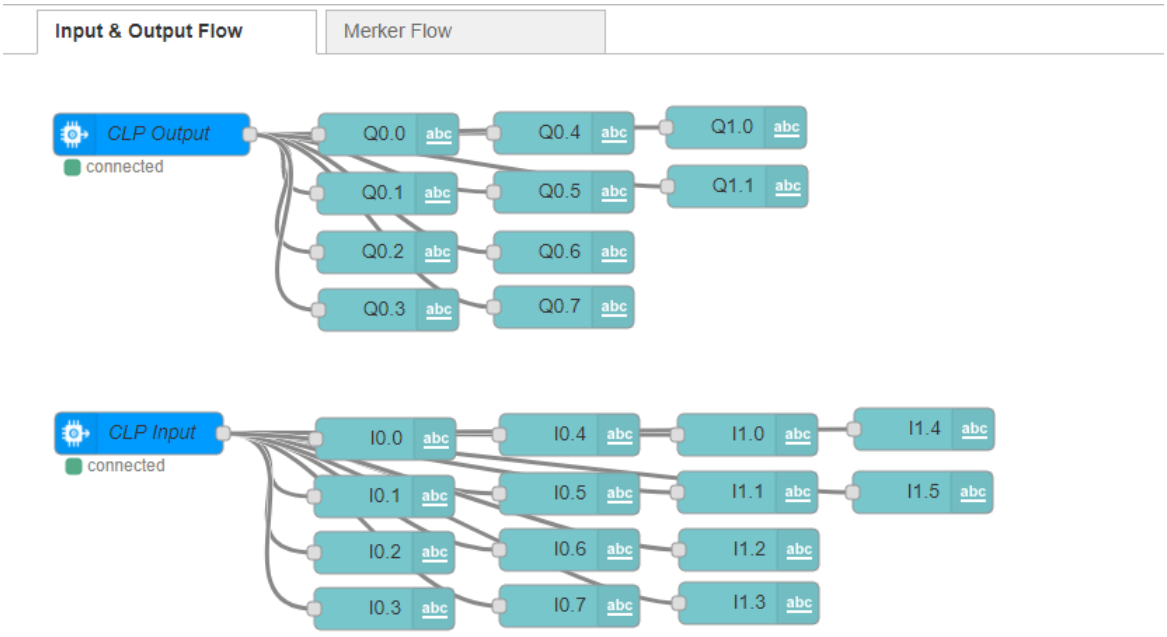
É realizada a leitura cíclica (a cada 10ms) das entradas e saídas digitais do CLP, havendo alteração de valores entre uma leitura e outra, os dados são enviados para o *broker MQTT*, uma *dashboard* em Node-RED recebe as mensagens e apresenta para o usuário final através do navegador *web*. A *dashboard* também é utilizada para controle das variáveis auxiliares do CLP, ao alterar o estados dos botões, é feita uma publicação no *broker*, o *gateway* então recebe a mensagem e altera os valores na memória do CLP. Para finalizar é utilizado um programa em LADDER (disponível no Anexo B) para alterar as saídas digitais do CLP conforme os valores das variáveis auxiliares.

Abaixo encontra-se o código em *python* para a implementação, as imagens 16 e 17 apresentam os fluxos em Node-RED e a figura 18 traz a *dashbard* criada.

```
1   import time
2   import json
3   import paho.mqtt.client as mqtt
4   import snap7
5   from snap7.snap7types import *
6   import numpy
7   import math
8   import struct
9
10  global out_byte_array_old
11  global in_byte_array_old
12
13  def merker(merker_payload):
14  merker_json = json.loads(merker_payload.decode('utf-8'))
15  merker_bit = [int(merker_json["M00"]) , int(merker_json["M01"]) ,
    ↪  int(merker_json["M02"]) , int(merker_json["M03"]) ,
16  int(merker_json["M04"]) , int(merker_json["M05"]) , int(merker_json["M06"]) ,
    ↪  int(merker_json["M07"]) ,
17  int(merker_json["M10"]) , int(merker_json["M11"])]
18
19  merker_byte = [0,0]
20
21  i = 0
22  j = 0
23
24  while (i<10):
25  if (i<8):
26  merker_byte[0] = int(merker_byte[0] + (merker_bit[i] * math.pow(2, i)))
27  else:
28  merker_byte[1] = int(merker_byte[1] + (merker_bit[i] * math.pow(2, i-8)))
29  i = i+1
30
31  merker_data = struct.pack('BB', merker_byte[0], merker_byte[1])
32  clp_w.write_area(S7AreaMK, 0, 0, merker_data)
33
34  def on_message(client, userdata, message):
35
36  if (message.topic=="iot-2/type/raspberry/id/pi1/evt/merker/fmt/json"):
```

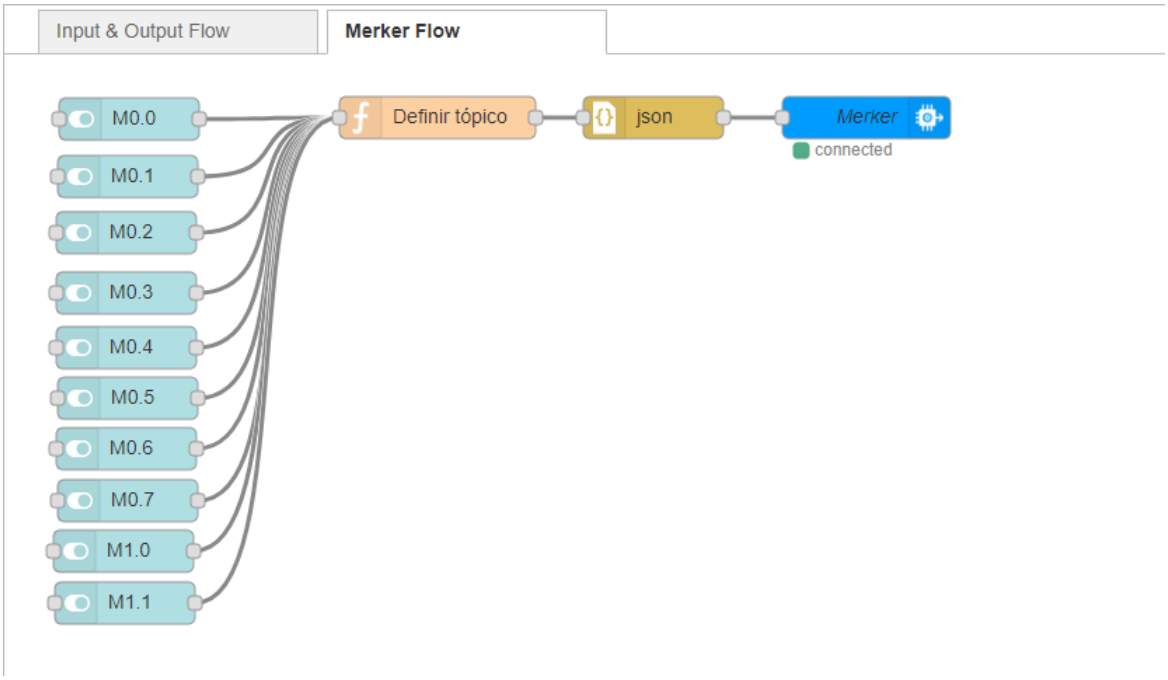
```
37          merker(message.payload)
38
39          def clp_read():
40          global out_byte_array_old
41          global in_byte_array_old
42
43          out_byte_array = clp_r.read_area(S7AreaPA, 0, 0, 2)
44          if (out_byte_array != out_byte_array_old):
45          out_byte_nparray = numpy.frombuffer(out_byte_array, dtype=numpy.uint8)
46          out_bit_nparray = numpy.unpackbits(out_byte_nparray).reshape(2,8)
47          clp_output = {'Q00': str(out_bit_nparray[0][7]), 'Q01': str(out_bit_nparray[0][6]),
            ↪  'Q02': str(out_bit_nparray[0][5]), 'Q03': str(out_bit_nparray[0][4]),
48                  'Q04': str(out_bit_nparray[0][3]), 'Q05': str(out_bit_nparray[0][2]), 'Q06':
                    ↪  str(out_bit_nparray[0][1]), 'Q07': str(out_bit_nparray[0][0]),
49                  'Q10': str(out_bit_nparray[1][7]), 'Q11': str(out_bit_nparray[1][6])}
50          client.publish('iot-2/type/raspberry/id/pi1/evt/output/fmt/json',
            ↪  json.dumps(clp_output), 1, True)
51          out_byte_array_old = out_byte_array
52
53          in_byte_array = clp_r.read_area(S7AreaPE, 0, 0, 2)
54          if (in_byte_array != in_byte_array_old):
55          in_byte_array = clp_r.read_area(S7AreaPE, 0, 0, 2)
56          in_byte_nparray = numpy.frombuffer(in_byte_array, dtype=numpy.uint8)
57          in_bit_nparray = numpy.unpackbits(in_byte_nparray).reshape(2,8)
58          clp_input = {'I00': str(in_bit_nparray[0][7]), 'I01': str(in_bit_nparray[0][6]), 'I02':
            ↪  str(in_bit_nparray[0][5]), 'I03': str(in_bit_nparray[0][4]),
59                  'I04': str(in_bit_nparray[0][3]), 'I05': str(in_bit_nparray[0][2]), 'I06':
                    ↪  str(in_bit_nparray[0][1]), 'I07': str(in_bit_nparray[0][0]),
60                  'I10': str(in_bit_nparray[1][7]), 'I11': str(in_bit_nparray[1][6]), 'I12':
                    ↪  str(in_bit_nparray[1][5]), 'I13': str(in_bit_nparray[1][4]),
61                  'I14': str(in_bit_nparray[1][3]), 'I15': str(in_bit_nparray[1][2])}
62          client.publish('iot-2/type/raspberry/id/pi1/evt/input/fmt/json', json.dumps(clp_input),
            ↪  1, True)
63          in_byte_array_old = in_byte_array
64
65
66          #S7 Client
67          clp_r = snap7.client.Client()
68          clp_r.connect('10.235.10.20',0,0)
69          clp_w = snap7.client.Client()
70          clp_w.connect('10.235.10.20',0,0)
71          out_byte_array_old = clp_r.read_area(S7AreaPA, 0, 0, 2)
72          in_byte_array_old = clp_r.read_area(S7AreaPE, 0, 0, 2)
73
74          #MQTT Client
75          client = mqtt.Client(client_id='a:****:****')
76          client.username_pw_set('****', '****')
77          client.on_message=on_message
78          client.connect('****.messaging.internetofthings.ibmcloud.com', 1883, 60)
79          client.subscribe('iot-2/type/raspberry/id/pi1/evt/merker/fmt/json', 1)
80          client.loop_start()
81
82          while True:
83          clp_read()
84          time.sleep(0.01)
```

Figura 16 – Fluxo em *Node-RED* para supervisão de entradas e saídas do CLP Siemens S7-1200



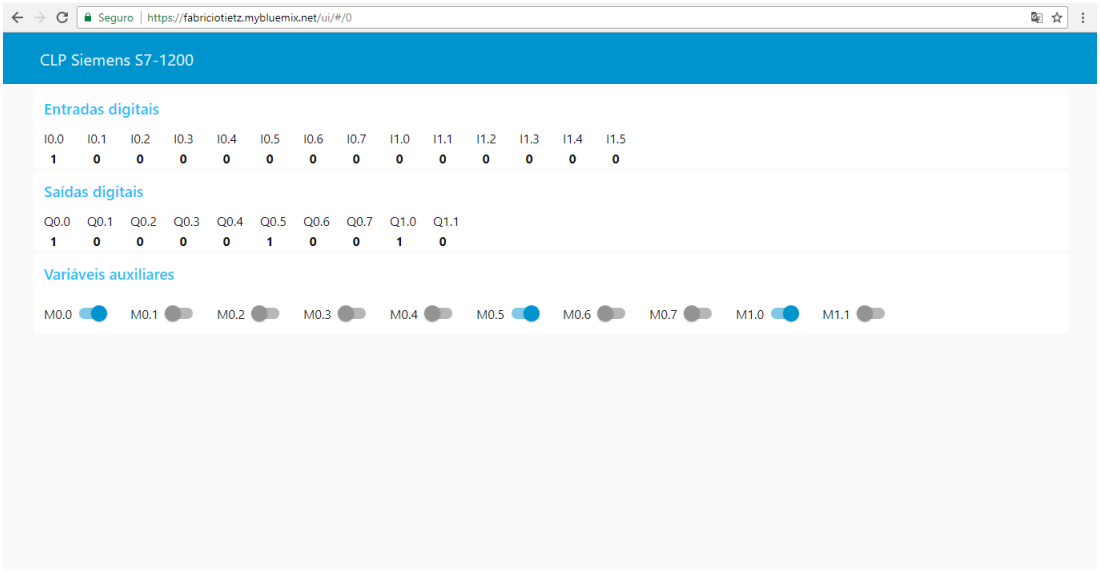
Fonte: elaborada pelo autor

Figura 17 – Fluxo em *Node-RED* para controle de variáveis auxiliares do CLP Siemens S7-1200



Fonte: elaborada pelo autor

Figura 18 – *Dashboard* para controle remoto do CLP Siemens S7-1200



Fonte: elaborada pelo autor

3.9 Supervisório para a planta didática da disciplina SEL0431

Para testar o *gateway* em um cenário real, foi feita a implementação para supervisão da planta didática da disciplina SEL0431 - Laboratório de Controle de Processos Industriais.

Na planta didática utilizada na disciplina, concebida por Oliveira (2016), são realizados os controles de processos em três malhas diferentes: temperatura, nível e vazão. As duas primeiras utilizam um CLP Siemens modelo S7-1200, enquanto a terceira utiliza um CLP Fertron Citrino. Ainda existe um quarto CLP, modelo Siemens S7-1200, para a segurança da planta. O Anexo C contém os roteiros da disciplina com os mapas de memória dos CLPs.

Na implementação é realizada a leitura ciclica (a cada 10ms) das entradas e saídas dos CLPs (tanto digitaism quanto analógicas) que são utilizadas nos processos. As informações são enviadas para o *broker* em caso de atualização entre uma leitura e outra (digitais) ou variação maior que 10% (analógicas).

O código em *python* encontra-se abaixo, as figuras 19, 20 e 21 apresentam os fluxos em Node-RED e as figuras 22, 23 e 24 apresentam as *dashboards* de supervisão.

```
1  import time
2  import json
3  import numpy
4  import paho.mqtt.client as mqtt
5  import snap7
6  from snap7.snap7types import *
7  from pymodbus.client.sync import ModbusTcpClient
8
9
10 global out_byte_array_old_t
11 global in_byte_array_old_t
12 global analog_IW64_byte_nparray_old_t
13 global analog_IW66_byte_nparray_old_t
14 global out_byte_array_old_n
15 global in_byte_array_old_n
16 global analog_QW80_byte_nparray_old_n
17 global analog_IW64_byte_nparray_old_n
18 global analog_ID266_byte_nparray_old_n
19 global ai_old_v
20 global di_old_v
21 global ao_old_v
22 global do_old_v
23
24 def clp_read_t():
25         global out_byte_array_old_t
26         global in_byte_array_old_t
27         global analog_IW64_byte_nparray_old_t
28         global analog_IW66_byte_nparray_old_t
29
30         out_byte_array_t = clp_r_t.read_area(S7AreaPA, 0, 0, 2)
```

```
31          if (out_byte_array_t != out_byte_array_old_t):
32                  out_byte_nparray = numpy.frombuffer(out_byte_array_t, dtype=numpy.uint8)
33                  out_bit_nparray = numpy.unpackbits(out_byte_nparray).reshape(2,8)
34                  clp_output = {'Q00': str(out_bit_nparray[0][7]), 'Q01':
                    ↪  str(out_bit_nparray[0][6]), 'Q02': str(out_bit_nparray[0][5]), 'Q03':
                    ↪  str(out_bit_nparray[0][4]), 'Q04': str(out_bit_nparray[0][3]), 'Q05':
                    ↪  str(out_bit_nparray[0][2]), 'Q06': str(out_bit_nparray[0][1]), 'Q07':
                    ↪  str(out_bit_nparray[0][0]), 'Q10': str(out_bit_nparray[1][7]), 'Q11':
                    ↪  str(out_bit_nparray[1][6])}
35                  client.publish('iot-2/type/CLP/id/temperatura/evt/output/fmt/json',
                    ↪  json.dumps(clp_output), 1, True)
36                  out_byte_array_old_t = out_byte_array_t
37          #Entradas analógicas
38          analog_IW64 = clp_r_t.read_area(S7AreaPE, 0, 64, 2)
39          analog_IW64_byte_nparray = numpy.frombuffer(analog_IW64, dtype=numpy.uint16).byteswap()
40          analog_IW64_up = int(analog_IW64_byte_nparray_old_t[0]) * float(1.1)
41          analog_IW64_down = int(analog_IW64_byte_nparray_old_t[0]) * float(0.9)
42
43          analog_IW66 = clp_r_t.read_area(S7AreaPE, 0, 66, 2)
44          analog_IW66_byte_nparray = numpy.frombuffer(analog_IW66, dtype=numpy.uint16).byteswap()
45          analog_IW66_up = int(analog_IW66_byte_nparray_old_t[0]) * float(1.1)
46          analog_IW66_down = int(analog_IW66_byte_nparray_old_t[0]) * float(0.9)
47
48          #Entradas digitais
49          in_byte_array_t = clp_r_t.read_area(S7AreaPE, 0, 0, 2)
50
51          #Verificação de alteração
52          if (in_byte_array_t != in_byte_array_old_t) or (int(analog_IW64_byte_nparray_old_t[0])
            ↪  < int(analog_IW64_down)) or (int(analog_IW64_byte_nparray_old_t[0]) >
            ↪  int(analog_IW64_up)) or (int(analog_IW66_byte_nparray_old_t[0]) <
            ↪  int(analog_IW66_down)) or (int(analog_IW66_byte_nparray_old_t[0]) >
            ↪  int(analog_IW66_up)):
53                  in_byte_nparray = numpy.frombuffer(in_byte_array_t, dtype=numpy.uint8)
54                  in_bit_nparray = numpy.unpackbits(in_byte_nparray).reshape(2,8)
55                  clp_input = {'I00': str(in_bit_nparray[0][7]), 'I01':
                    ↪  str(in_bit_nparray[0][6]), 'I02': str(in_bit_nparray[0][5]), 'I03':
                    ↪  str(in_bit_nparray[0][4]), 'I04': str(in_bit_nparray[0][3]), 'I05':
                    ↪  str(in_bit_nparray[0][2]), 'I06': str(in_bit_nparray[0][1]), 'I07':
                    ↪  str(in_bit_nparray[0][0]), 'I10': str(in_bit_nparray[1][7]), 'I11':
                    ↪  str(in_bit_nparray[1][6]), 'I12': str(in_bit_nparray[1][5]), 'I13':
                    ↪  str(in_bit_nparray[1][4]), 'I14': str(in_bit_nparray[1][3]), 'I15':
                    ↪  str(in_bit_nparray[1][2]), 'IW64' : str(analog_IW64_byte_nparray[0]),
                    ↪  'IW66' : str(analog_IW66_byte_nparray[0])}
56                  client.publish('iot-2/type/CLP/id/temperatura/evt/input/fmt/json',
                    ↪  json.dumps(clp_input), 1, True)
57                  in_byte_array_old_t = in_byte_array_t
58                  analog_IW64_byte_nparray_old_t = analog_IW64_byte_nparray
59                  analog_IW66_byte_nparray_old_t = analog_IW66_byte_nparray
60
61  def clp_read_n():
62          global out_byte_array_old_n
63          global in_byte_array_old_n
64          global analog_QW80_byte_nparray_old_n
65          global analog_IW64_byte_nparray_old_n
66          global analog_ID266_byte_nparray_old_n
67
```

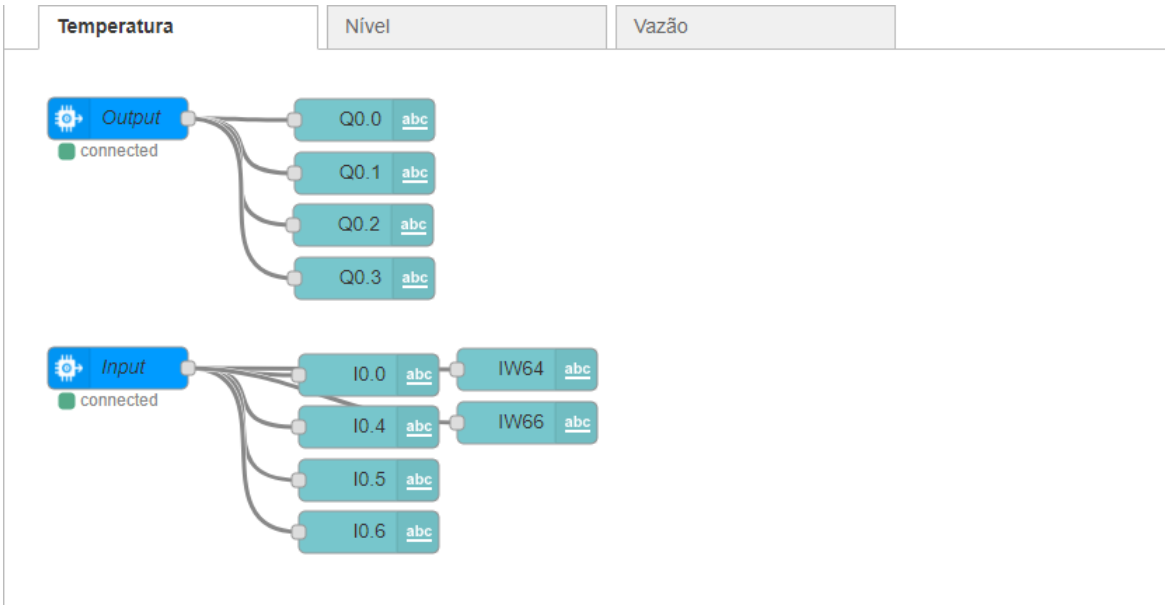
```
68          #Saídas analógicas
69          analog_QW80 = clp_r_n.read_area(S7AreaPA, 0, 80, 2)
70          analog_QW80_byte_nparray = numpy.frombuffer(analog_QW80, dtype=numpy.uint16).byteswap()
71          analog_QW80_up = int(analog_QW80_byte_nparray_old_n[0]) * float(1.1)
72          analog_QW80_down = int(analog_QW80_byte_nparray_old_n[0]) * float(0.9)
73
74          #Saídas digitais
75          out_byte_array_n = clp_r_n.read_area(S7AreaPA, 0, 0, 2)
76
77          #Verificação de alteração
78          if (out_byte_array_n != out_byte_array_old_n) or
            ↪  (int(analog_QW80_byte_nparray_old_n[0]) < int(analog_QW80_down)) or
            ↪  (int(analog_QW80_byte_nparray_old_n[0]) > int(analog_QW80_up)):
79              out_byte_nparray = numpy.frombuffer(out_byte_array_n, dtype=numpy.uint8)
80              out_bit_nparray = numpy.unpackbits(out_byte_nparray).reshape(2,8)
81              clp_output = {'Q00': str(out_bit_nparray[0][7]), 'Q01':
                ↪  str(out_bit_nparray[0][6]), 'Q02': str(out_bit_nparray[0][5]), 'Q03':
                ↪  str(out_bit_nparray[0][4]), 'Q04': str(out_bit_nparray[0][3]), 'Q05':
                ↪  str(out_bit_nparray[0][2]), 'Q06': str(out_bit_nparray[0][1]), 'Q07':
                ↪  str(out_bit_nparray[0][0]), 'Q10': str(out_bit_nparray[1][7]), 'Q11':
                ↪  str(out_bit_nparray[1][6]), 'QW80' : str(analog_QW80_byte_nparray[0])}
82              client.publish('iot-2/type/CLP/id/nivel/evt/output/fmt/json',
                ↪  json.dumps(clp_output), 1, True)
83              out_byte_array_old_n = out_byte_array_n
84              analog_QW80_byte_nparray_old_n = analog_QW80_byte_nparray
85
86          #Entradas analógicas
87          analog_IW64 = clp_r_n.read_area(S7AreaPE, 0, 64, 2)
88          analog_IW64_byte_nparray = numpy.frombuffer(analog_IW64, dtype=numpy.uint16).byteswap()
89          analog_IW64_up = int(analog_IW64_byte_nparray_old_n[0]) * float(1.1)
90          analog_IW64_down = int(analog_IW64_byte_nparray_old_n[0]) * float(0.9)
91
92          analog_ID266 = clp_r_n.read_area(S7AreaPE, 0, 266, 4)
93          analog_ID266_byte_nparray = numpy.frombuffer(analog_ID266,
            ↪  dtype=numpy.uint32).byteswap()
94          analog_ID266_up = int(analog_ID266_byte_nparray_old_n[0]) * float(1.1)
95          analog_ID266_down = int(analog_ID266_byte_nparray_old_n[0]) * float(0.9)
96
97          #Entradas digitais
98          in_byte_array_n = clp_r_n.read_area(S7AreaPE, 0, 0, 2)
99
100         #Verificação de alteração
101         if (in_byte_array_n != in_byte_array_old_n) or (int(analog_IW64_byte_nparray_old_n[0])
            ↪  < int(analog_IW64_down)) or (int(analog_IW64_byte_nparray_old_n[0]) >
            ↪  int(analog_IW64_up)) or (int(analog_ID266_byte_nparray_old_n[0]) <
            ↪  int(analog_ID266_down)) or (int(analog_ID266_byte_nparray_old_n[0]) >
            ↪  int(analog_ID266_up)):
102             in_byte_nparray = numpy.frombuffer(in_byte_array_n, dtype=numpy.uint8)
103             in_bit_nparray = numpy.unpackbits(in_byte_nparray).reshape(2,8)
104             clp_input = {'I00': str(in_bit_nparray[0][7]),
105             'I01': str(in_bit_nparray[0][6]), 'I02': str(in_bit_nparray[0][5]),
106             'I03': str(in_bit_nparray[0][4]), 'I04': str(in_bit_nparray[0][3]),
107             'I05': str(in_bit_nparray[0][2]), 'I06': str(in_bit_nparray[0][1]),
108             'I07': str(in_bit_nparray[0][0]), 'I10': str(in_bit_nparray[1][7]),
109             'I11': str(in_bit_nparray[1][6]), 'I12': str(in_bit_nparray[1][5]),
110             'I13': str(in_bit_nparray[1][4]), 'I14': str(in_bit_nparray[1][3]),
```

```
111                 'I15': str(in_bit_nparray[1][2]), 'IW64' : str(analog_IW64_byte_nparray[0]),
112                 'ID266' : str(analog_ID266_byte_nparray[0])}
113                 client.publish('iot-2/type/CLP/id/nivel/evt/input/fmt/json',
                    ↪  json.dumps(clp_input), 1, True)
114                 in_byte_array_old_n = in_byte_array_n
115                 analog_IW64_byte_nparray_old_n = analog_IW64_byte_nparray
116                 analog_ID266_byte_nparray_old_n = analog_ID266_byte_nparray
117
118 def clp_read_v():
119         global ai_old_v
120         global di_old_v
121         global ao_old_v
122         global do_old_v
123
124         #Saídas digitais
125         do_v = clp_r_v.read_coils(1, 4, unit=255)
126         #Saída analógica
127         ao_old_up = int(ao_old_v.registers[0]) * float(1.1)
128         ao_old_down = int(ao_old_v.registers[0]) * float(.9)
129         ao_v = clp_r_v.read_holding_registers(5000, 1, unit=255)
130         if (do_v.bits != do_old_v.bits) or (int(ao_v.registers[0]) < int(ao_old_down)) or
            ↪  (int(ao_v.registers[0]) > int(ao_old_up)):
131                 clp_output = {'DO2': int(do_v.bits[0]), 'DO3': int(do_v.bits[1]), 'DO4':
                    ↪  int(do_v.bits[2]), 'DO5': int(do_v.bits[3]), 'AO1':
                    ↪  str(int(ao_v.registers[0]))}
132                 client.publish('iot-2/type/CLP/id/vazao/evt/output/fmt/json',
                    ↪  json.dumps(clp_output), 1, True)
133                 do_old_v = do_v
134                 ao_old_v = ao_v
135
136         #Entradas digitais
137         di_v = clp_r_v.read_discrete_inputs(2, 4, unit=255)
138         #Entrada analógica
139         ai_old_up = int(ai_old_v.registers[0]) * float(1.1)
140         ai_old_down = int(ai_old_v.registers[0]) * float(.9)
141         ai_v = clp_r_v.read_input_registers(0, 1, unit=255)
142         if (int(ai_v.registers[0]) < int(200)): #Limite mínimo para considerar zero
143                 ai_v.registers[0] = 0
144         if (di_v.bits != di_old_v.bits) or (int(ai_v.registers[0]) < int(ai_old_down)) or
            ↪  (int(ai_v.registers[0]) > int(ai_old_up)):
145                 clp_input = {'DI3': int(di_v.bits[0]), 'DI4': int(di_v.bits[1]), 'DI5':
                    ↪  int(di_v.bits[2]), 'DI6': int(di_v.bits[3]), 'AI1':
                    ↪  str(int(ai_v.registers[0]))}
146                 client.publish('iot-2/type/CLP/id/vazao/evt/input/fmt/json',
                    ↪  json.dumps(clp_input), 1, True)
147                 di_old_v = di_v
148                 ai_old_v = ai_v
149
150
151 #S7 Client
152 #CLP malha temperatura
153 clp_r_t = snap7.client.Client()
154 try:
155         clp_r_t.connect('10.235.10.20',0,0)
156         out_byte_array_old_t = clp_r_t.read_area(S7AreaPA, 0, 0, 2)
157         in_byte_array_old_t = clp_r_t.read_area(S7AreaPE, 0, 0, 2)
```

```
158         analog_IW64_byte_nparray_old_t  = clp_r_t.read_area(S7AreaPE, 0, 64, 2)
159         analog_IW66_byte_nparray_old_t  = clp_r_t.read_area(S7AreaPE, 0, 66, 2)
160 except Exception as e:
161         print('Falha na conexão com o CLP 10.235.10.20. Reinicie a aplicação para tentar
            ↪  novamente: ' + str(e))
162         wait = input('Pressione ENTER para sair da aplicação')
163         exit()
164
165
166 #CLP malha nivel
167 clp_r_n = snap7.client.Client()
168 try:
169         clp_r_n.connect('10.235.10.21',0,0)
170         out_byte_array_old_n = clp_r_n.read_area(S7AreaPA, 0, 0, 2)
171         in_byte_array_old_n = clp_r_n.read_area(S7AreaPE, 0, 0, 2)
172         analog_QW80_byte_nparray_old_n = clp_r_n.read_area(S7AreaPA, 0, 80, 2)
173         analog_IW64_byte_nparray_old_n = clp_r_n.read_area(S7AreaPE, 0, 64, 2)
174         analog_ID266_byte_nparray_old_n = clp_r_n.read_area(S7AreaPE, 0, 266, 4)
175 except Exception as e:
176         print('Falha na conexão com o CLP 10.235.10.21. Reinicie a aplicação para tentar
            ↪  novamente: ' + str(e))
177         wait = input('Pressione ENTER para sair da aplicação')
178         exit()
179
180 #CLP malha vazão
181 try:
182         clp_r_v = ModbusTcpClient("10.235.10.202", 502)
183         ai_old_v = clp_r_v.read_input_registers(0, 1, unit=255)
184         di_old_v = clp_r_v.read_discrete_inputs(2, 4, unit=255)
185         ao_old_v = clp_r_v.read_holding_registers(5000, 1, unit=255)
186         do_old_v = clp_r_v.read_coils(1, 4, unit=255)
187 except Exception as e:
188         print('Falha na conexão com o CLP 10.235.10.202. Reinicie a aplicação para tentar
            ↪  novamente: ' + str(e))
189         wait = input('Pressione ENTER para sair da aplicação')
190         exit()
191
192 #MQTT Client
193 client = mqtt.Client(client_id='a:****:gateway')
194 client.username_pw_set('a-****', '****')
195 try:
196         client.connect('****.messaging.internetofthings.ibmcloud.com', 1883, 60)
197         client.subscribe('iot-2/type/raspberry/id/pi1/evt/merker/fmt/json', 1)
198         client.loop_start()
199 except Exception as e:
200         print('Falha na conexão com o broker MQTT. Reinicie a aplicação para tentar novamente:
            ↪  ' + str(e))
201         wait = input('Pressione ENTER para sair da aplicação')
202         exit()
203
204 while True:
205 try:
206         clp_read_t()
207 except Exception as e:
208         print('Falha na leitura do CLP de Temperatura: ' + str(e))
209         pass
```

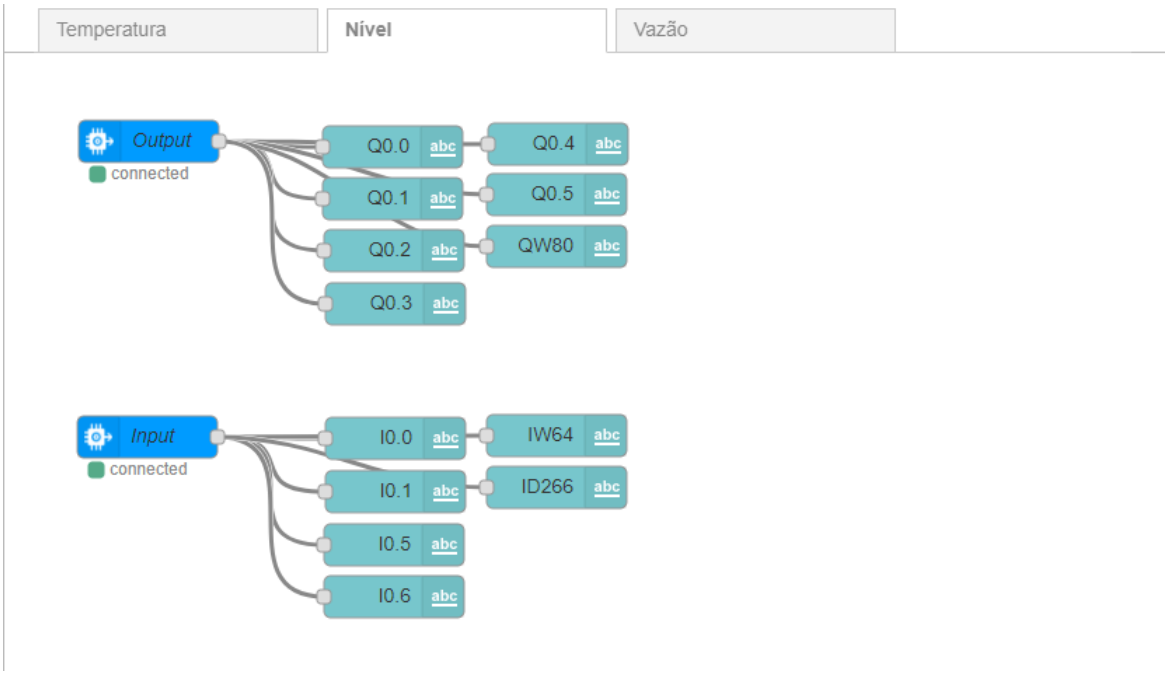
```
time.sleep(0.01)

try:
        clp_read_n()
except Exception as e:
        print('Falha na leitura do CLP de Nivel: ' + str(e))
        pass
time.sleep(0.01)

try:
        clp_read_v()
except Exception as e:
        print('Falha na leitura do CLP de Vazao: ' + str(e))
        pass
time.sleep(0.01)
```

Figura 19 – Fluxo em *Node-RED* para *dashboard* de supervisão da malha de temperatura



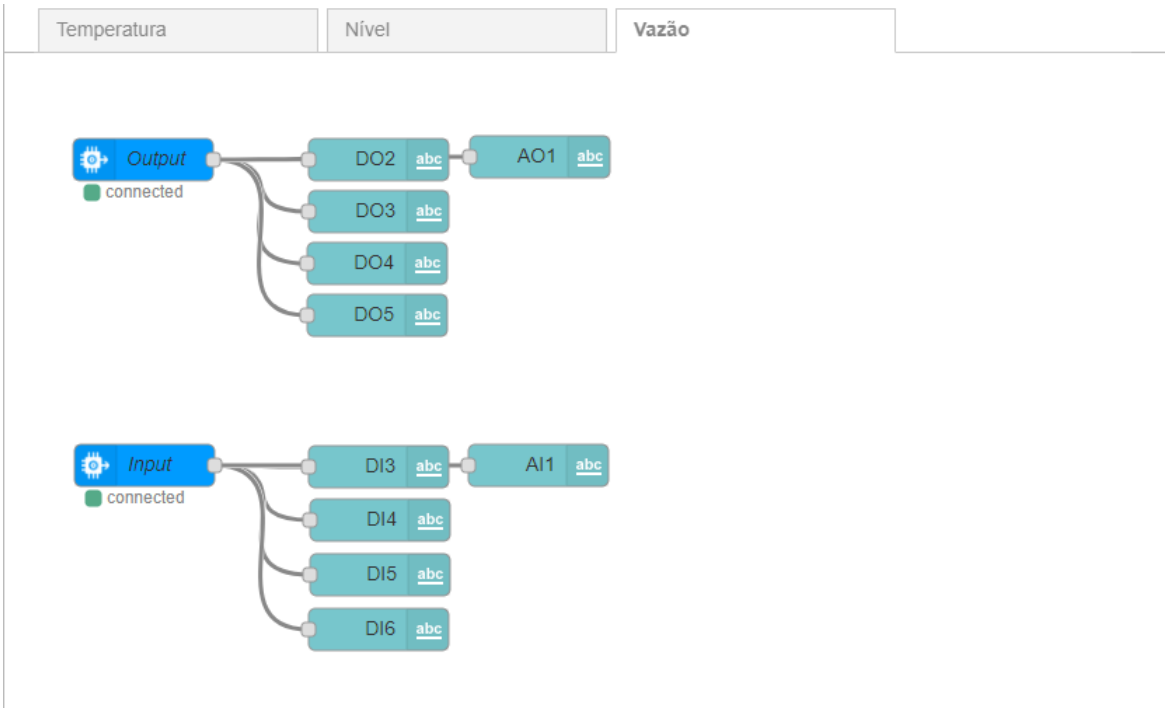
Fonte: elaborada pelo autor

Figura 20 – Fluxo em *Node-RED* para *dashboard* de supervisão da malha de nível



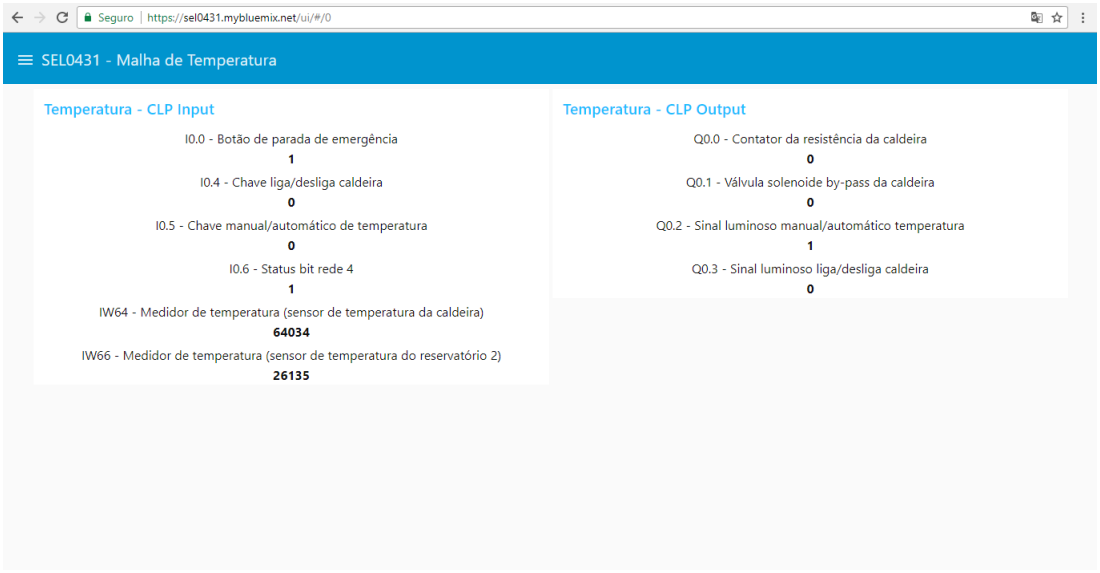
Fonte: elaborada pelo autor

Figura 21 – Fluxo em *Node-RED* para *dashboard* de supervisão da malha de vazão



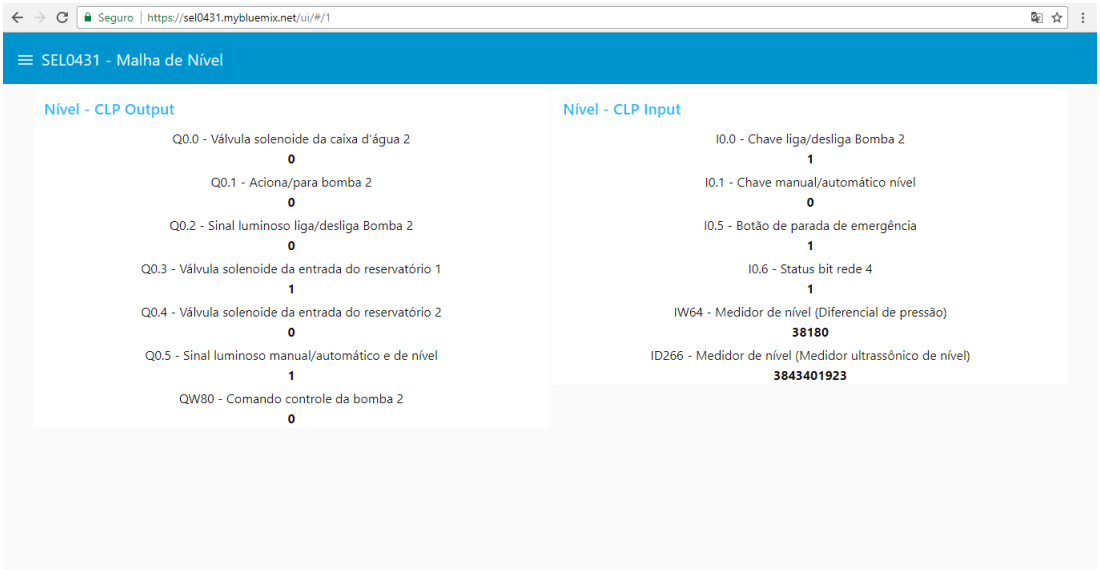
Fonte: elaborada pelo autor

Figura 22 – *Dashboard* para supervisão da malha de temperatura



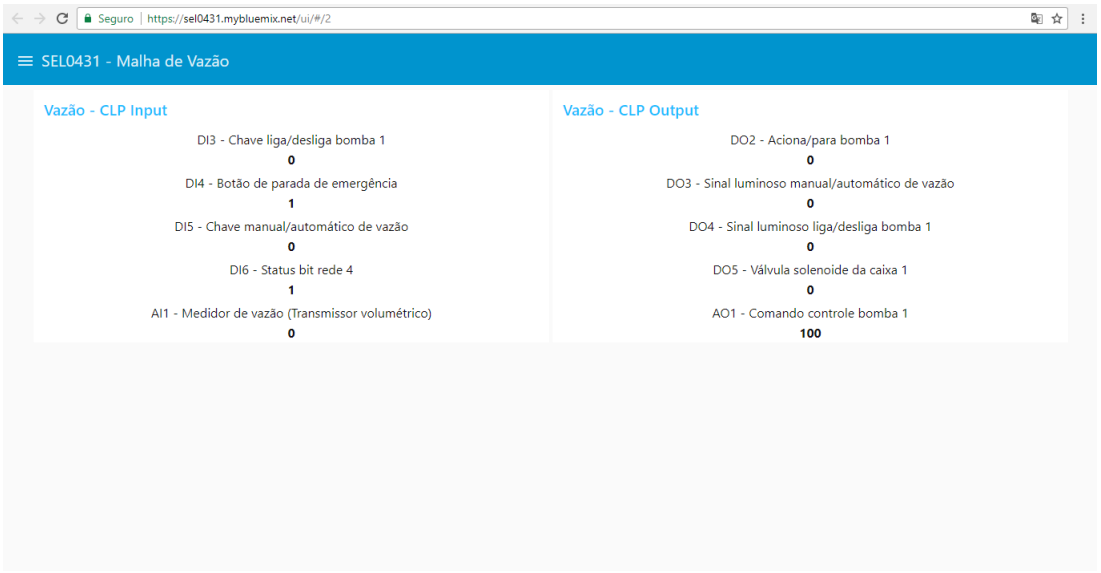
Fonte: elaborada pelo autor

Figura 23 – *Dashboard* para supervisão da malha de nível



Fonte: elaborada pelo autor

Figura 24 – *Dashboard* para supervisão da malha de vazão



Fonte: elaborada pelo autor

Capítulo 4

Resultados

4.1 Tempo de resposta para leitura de informações do CLP, publicação de mensagem no *broker* e retorno da mensagem

Para avaliar o tempo de resposta das operações do *gateway* foi implementado um programa que registra o tempo utilizado na leitura de um *byte* de memória de um CLP S7-1200, correspondente a 10 saídas digitais, e também o tempo decorrido entre a publicação de uma mensagem no *broker MQTT* e o retorno da mesma mensagem para o dispositivo. A figura 25 apresenta o fluxograma dos testes realizados.

A conexão com o CLP é feita com o protocolo Siemens S7 através da placa de rede cabeada (*ethernet*) e a placa de rede sem fio (*wireless*) é utilizada para conexão com a rede *WI-FI* para acesso à internet e permitir a conexão com o *broker* através do protocolo MQTT.

A tabela 6 traz os resultados para 100 testes realizados, no Anexo A encontra-se a tabela com os tempos de resposta para cada um dos testes.

Tabela 6 – Tempos de resposta para as operações testadas

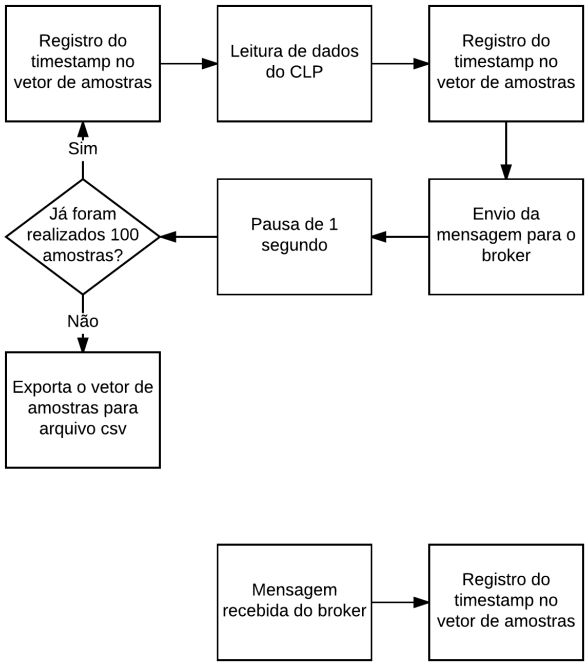
Operação	Média (ms)	Mínimo (ms)	Máximo (ms)	Desvio padrão (ms)
1	011,2177	001,9898	055,0528	014,3072
2	215,5126	149,1167	372,2243	048,6220

Operação: 1 - Leitura de dados do CLP, 2 - Publicação no *broker* e retorno da mensagem

Fonte: elaborada pelo autor

Como indicado pelo desvio padrão maior que a média, a leitura de dados do CLP apresenta grande variação de tempo, mas os resultados não permitem concluir a causa dessa variação.

Figura 25 – Fluxograma do teste de tempos de resposta



Fonte: elaborada pelo autor

Os valores de tempo para comunicação com o *broker* possuem maior regularidade e podem ser reduzidos se a conexão com a internet for realizada por uma placa de rede cabeada adicional ao invés da placa de rede sem fio utilizada no teste.

Abaixo encontra-se o código em *python* utilizado para registrar os tempos de resposta.

```
import time
import numpy
import json
import paho.mqtt.client as mqtt
import snap7
from snap7.snap7types import *

def on_message(client, userdata, message):
global timestamp
global offset
if (message.topic=="iot-2/type/raspberry/id/pi1/evt/time/fmt/json"):
data_json = json.loads(message.payload)
time_id = data_json["id"]
timestamp[time_id][2]=time.time()-timestamp[i][1]-timestamp[i][0]-offset

global timestamp
global offset

```

```
#S7 Client
clp = snap7.client.Client()
clp.connect('192.168.10.105',0,0)

#MQTT Client
client = mqtt.Client(client_id='a:***:***')
client.username_pw_set('****', '****')
client.on_message=on_message
client.connect('***.messaging.internetofthings.ibmcloud.com', 1883, 60)
client.subscribe('iot-2/type/raspberry/id/pi1/evt/time/fmt/json', 1)
client.loop_start()

offset = time.time()
timestamp = numpy.zeros((100,3), dtype=numpy.float64)
i=0

while (i<100):
timestamp[i][0]=time.time()-offset
leitura = clp.read_area(S7AreaPA, 0, 0, 2)
timestamp[i][1]=time.time()-timestamp[i][0]-offset
data = {'id' : i, 'leitura' : str(leitura)}
client.publish('iot-2/type/raspberry/id/pi1/evt/time/fmt/json', json.dumps(data), 1, True)
time.sleep(1)
i=i+1

numpy.savetxt('c:/dados.csv', timestamp, fmt='%.14f', delimiter=';')
```

Conclusão

Esse trabalho teve como objetivo propor uma solução para integração de ambientes industriais tradicionais e novas tecnologias de internet das coisas através do desenvolvimento de um *gateway*.

No desenvolvimento foram utilizadas tecnologias de código aberto como a linguagem de programação *python* e bibliotecas livres para utilização dos protocolos de comunicação. Também foi utilizado o *hardware Raspberry Pi*, de baixo custo e fácil acesso no mercado. Todo o desenvolvimento realizado é multiplataforma, pode ser rapidamente adaptado a outros sistemas operacionais, arquiteturas ou plataformas de hardware, o que permite a utilização em projetos de diferentes tipos e portes.

Para a realização do projeto foram essenciais os aprendizados de disciplinas como Redes de Computadores, Linguagens de Programação e Aplicações, Aplicação de Microprocessadores I e II, Automação e Redes de Comunicação Industrial, bem como os conhecimentos adquiridos no período de estágio com o desenvolvimento de um projeto utilizando *Raspberry Pi* e linguagem de programação *python*.

Por meio das aplicações realizadas, foi possível verificar a funcionalidade do *gateway* desenvolvido no monitoramento remoto de uma planta didática e controle remoto de um CLP. Todos os códigos desenvolvidos estão integralmente disponibilizados, possibilitando o aproveitamento em outros projetos e pesquisas.

Trabalhos futuros podem aproveitar o *gateway* proposto para realizar o processamento e controle de plantas industriais através da nuvem, também podem ser estudados outros protocolos de comunicação industrial e IoT, bem como explorar as funcionalidades da Raspberry Pi conectada diretamente a sensores e atuadores através dos pinos de GPIO.

Referências

CASTELLS, M. A sociedade em rede, vol. 1. **São Paulo: Paz e Terra**, 1999. v. 8, 1999.

FERTRON. **Manual CLP Citrino**. jun. 2007. Acessado: 23 out. 2017. Disponível em: <http://www.fertron.com.br/site_2014/imagens/uploads/produto/7/77/101/arq-/manual_clp_citrino.pdf>.

FOUNDATION, E. **Paho**. nov. 2015. Acessado: 10/08/2017. Disponível em: <http://wiki.eclipse.org/Paho>.

FOUNDATION, J. **Node-RED**. 2017. Acessado: 31/10/2017. Disponível em: <https://nodered.org/>.

FOUNDATION, P. S. **7.1. struct - Interpret bytes as packed binary data - Python 3.6.3 documentation**. out. 2017. Acessado: 31/10/2017. Disponível em: <https://docs.python.org/3/library/struct.html>.

FREITAS, C. M. **Protocolo Modbus: Fundamentos e Aplicações**. abr. 2014. Acessado: 20/10/2017. Disponível em: <https://www.embarcados.com.br/protocolo-modbus/>.

HIVEMQ. **MQTT Essentials: Part 1 - Introducing MQTT**. 2016. Disponível em: <https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt>.

HORN, L. S. A estratégia alemã para a indústria digital. 2017. 2017.

JúNIOR, M. F. F. A terceira revolução industrial e o novo paradigma produtivo: algumas considerações sobre o desenvolvimento industrial brasileiro nos anos 90. **Revista da FAE**, 2017. v. 3, n. 2, 2017.

LAMPKIN, V.; LEONG, W.; OLIVERA, L.; RAWAT, S.; SUBRAHMANYAM, N.; XIANG, R.; KALLAS, G.; KRISHNA, N.; FASSMANN, S.; KEEN, M. et al. **Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry**. [S.l.]: IBM Redbooks, 2012. (IBM redbooks). ISBN 9780738437088.

MACKAY, S. **Practical Industrial Data Networks: Design, Installation and Troubleshooting**. [S.l.]: Newnes, 2004. (IDC Technology). ISBN 9780750658072.

MIRU, G. **The Siemens S7 Communication - Part 1 General Structure**. jan. 2016. Acessado: 25/06/2017. Disponível em: <http://gmiru.com/article/s7comm/>.

NARDELLA, D. **Snap7 Reference Manual**. [S.l.], dezembro 2016. Disponível em: <https://sourceforge.net/projects/snap7/files/1.4.2/>.

OLIVEIRA, M. A. **Construção e implementação de uma planta didática multiprocessos para uso em laboratório de ensino**. Monografia (Trabalho de Conclusão de Curso) — UNIVERSIDADE DE SÃO PAULO, 2016.

ORGANIZATION, M. **MODBUS Messaging on TCP/IP Implementation Guide V1.0b**. out. 2006. Acessado: 20/10/2017. Disponível em: <http://www.modbus.org-/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf>.

_____. **MODBUS Application Protocol Specification V1.1b3**. abr. 2012. Acessado: 20/10/2017. Disponível em: <http://www.modbus.org/docs-/Modbus_Application_Protocol_V1_1b3.pdf>.

_____. **Modbus FAQ**. entre 2005 e 2017. Acessado: 20/10/2017. Disponível em: <http://www.modbus.org/faq.php>.

RASPBIAN. **FrontPage - Raspbian**. entre 2012 e 2017. Acessado: 02/11/2017. Disponível em: <http://www.raspbian.org/>.

RIPTIDEIO. **Pymodbus**. ago. 2017. Acessado: 23 out. 2017. Disponível em: <http://riptideio.github.io/pymodbus/>.

SHIPLEY, A. Security in the internet of things, lessons from the past for the connected future. **Security Solutions, Wind River, White Paper**, 2013. 2013.

STANDARD, O. **MQTT Version 3.1.1**. out. 2014. Disponível em: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf>.

TOMALA-REYES, A. **O que é IBM Bluemix?** 2014. Acessado: 21/06/2017. Disponível em: <https://www.ibm.com/developerworks/br/cloud/library/cl-bluemixfoundry/index.html>.

UPTON, E.; HALFACREE, G. **Raspberry Pi User Guide**. [S.l.]: Wiley, 2016. ISBN 9781119264378.

WIENS, T. **S7 Communication (S7comm)**. maio 2016. Acessado: 20/06/2017. Disponível em: <https://wiki.wireshark.org/S7comm>.

Anexos

ANEXO A

Tabela de resultados para o teste de tempo de resposta

Teste	Início do ciclo (s)	Tempo leitura CLP(s)	Publicação e retorno do broker (s)
1	0	0,033021927	0,329221249
2	1,033691406	0,002001286	0,159106731
3	2,036367178	0,002011299	0,20712328
4	3,039035559	0,001997471	0,154104233
5	4,041705847	0,001997471	0,184123516
6	5,044375896	0,002001524	0,168108702
7	6,047045946	0,002001524	0,208139658
8	7,049716949	0,002001286	0,163105011
9	8,052386522	0,001997709	0,184123993
10	9,05505681	0,002001762	0,160106421
11	10,05772567	0,022014618	0,249163389
12	11,11543083	0,020013571	0,205135345
13	12,17013931	0,002001047	0,207138062
14	13,17280483	0,037028313	0,279184103
15	14,24852824	0,021014214	0,207137823
16	15,27021194	0,002000809	0,159102917
17	16,27289057	0,003993511	0,200128794
18	17,27755237	0,002000332	0,149116755
19	18,28022242	0,002002239	0,178118706
20	19,28289318	0,004002094	0,237159252
21	20,28756428	0,002001286	0,20413661
22	21,29022956	0,002001524	0,19513154
23	22,29290438	0,002001286	0,203132629
24	23,29557467	0,002000093	0,200134993
25	24,29824471	0,001998186	0,201138496
26	25,30091476	0,002002001	0,204132557
27	26,30358529	0,002001286	0,20713377
28	27,30625534	0,001997948	0,205140591
29	28,30892539	0,037030458	0,23314786
30	29,41365933	0,053036928	0,266178131
31	30,50138855	0,019018412	0,209136486
32	31,55809712	0,002001286	0,17411685
33	32,56076789	0,002000809	0,166110992
34	33,56343794	0,002001047	0,156104088
35	34,56610584	0,0020082	0,160101891
36	35,5687778	0,002001286	0,158102274
37	36,57144523	0,002001286	0,158105612
38	37,57411814	0,002001762	0,200133085
39	38,57678843	0,022015095	0,277181864
40	39,63449526	0,020020723	0,175109863
41	40,65517735	0,002001524	0,151100874
42	41,6578548	0,00199914	0,186120033
43	42,66051841	0,038031578	0,292185545
44	43,73423934	0,022032976	0,372224331
45	44,79194236	0,021013975	0,239157677
46	45,84864855	0,002001524	0,189126253
47	46,85131955	0,001996517	0,201138735
48	47,85398889	0,00400281	0,200130701
49	48,85866022	0,002001524	0,201134443

50	49,86133051	0,002001286	0,201131821
51	50,8640008	0,002000332	0,205135345
52	51,86667895	0,001992226	0,208141327
53	52,8693428	0,002000332	0,200133085
54	53,87201476	0,002001047	0,200130701
55	54,87468123	0,002000332	0,202132702
56	55,87735152	0,002001524	0,168109179
57	56,88002229	0,002000809	0,203135729
58	57,88269258	0,00200057	0,201134682
59	58,88536215	0,004001379	0,278183699
60	59,89003348	0,002001524	0,205133438
61	60,89270377	0,021013975	0,267179251
62	61,95041108	0,002007723	0,297190666
63	62,95308065	0,002000332	0,19913435
64	63,95575166	0,00200057	0,204136372
65	64,95842123	0,002001286	0,221147776
66	65,961092	0,00200057	0,202135086
67	66,96376204	0,00200057	0,203133106
68	67,96643162	0,002001524	0,188126087
69	68,96910286	0,002000332	0,176151276
70	69,97177172	0,002001524	0,209169388
71	70,97445345	0,001989841	0,212143183
72	71,97711277	0,001997948	0,201137304
73	72,97977948	0,004006147	0,154102564
74	73,98445511	0,021013021	0,254169464
75	75,12722111	0,021013975	0,275176048
76	76,1859355	0,021008015	0,294189692
77	77,24163222	0,019008398	0,191131353
78	78,26131248	0,002001286	0,289191008
79	79,26398253	0,051029682	0,288192511
80	80,35070562	0,035023212	0,228153229
81	81,4214325	0,001993418	0,179120064
82	82,42409134	0,032022476	0,270180225
83	83,49180841	0,021015882	0,217139959
84	84,54851484	0,002018213	0,15408659
85	85,55118537	0,001996756	0,175121069
86	86,55385566	0,004002094	0,203132868
87	87,55852628	0,040033817	0,309196949
88	88,66926551	0,01901269	0,260174513
89	89,74298215	0,020014286	0,217148542
90	90,80169463	0,039028883	0,343226194
91	91,87640905	0,020013332	0,325216293
92	92,93211913	0,021013021	0,223149061
93	93,98882723	0,021011353	0,30720067
94	95,04553056	0,038031816	0,279177427
95	96,22331834	0,022029161	0,170095444
96	97,24601054	0,001992702	0,158105135
97	98,24867225	0,002003431	0,167105436
98	99,25134182	0,021014452	0,291192293
99	100,3250542	0,055052757	0,255156517

100	101,4848437	0,054022789	0,244167328

ANEXO **B**

Programação em LADDER para controle remoto de CLP Siemens S7-1200

Main [OB1]

Main Properties					
General					
Name	Main	**Number**	1	**Type**	OB.ProgramCycle
Language	LAD				
Information					
Title	"Main Program Sweep (Cycle)"	**Author**	Fabrício Tietz	**Comment**	
Family		**Version**	0.1	**User-defined ID**	

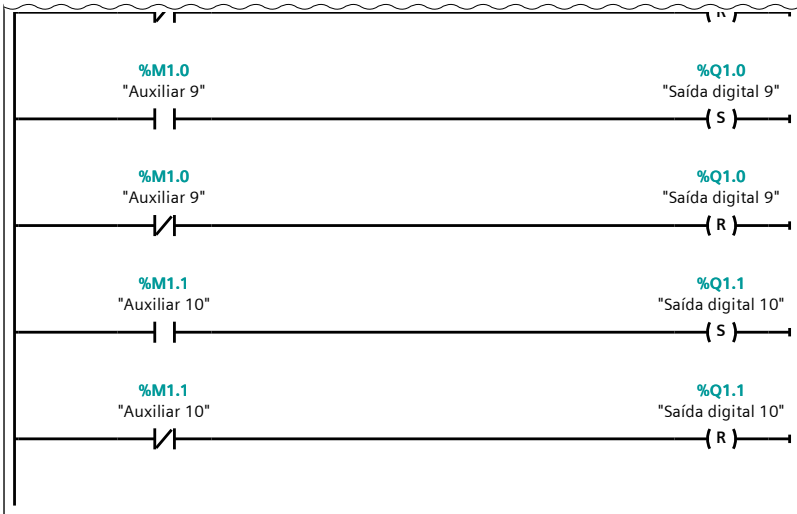
Name	Data type	Offset	Comment
Temp			

Network 1:

Network 1: (1.1 / 2.1)

Network 1: (2.1 / 2.1)

1.1 (Page1 - 2)



Symbol	Address	Type	Comment
"Auxiliar 1"	%M0.0	Bool	
"Saída digital 1"	%Q0.0	Bool	
"Auxiliar 2"	%M0.1	Bool	
"Saída digital 2"	%Q0.1	Bool	
"Auxiliar 3"	%M0.2	Bool	
"Saída digital 3"	%Q0.2	Bool	
"Auxiliar 4"	%M0.3	Bool	
"Saída digital 4"	%Q0.3	Bool	
"Auxiliar 5"	%M0.4	Bool	
"Saída digital 5"	%Q0.4	Bool	
"Auxiliar 6"	%M0.5	Bool	
"Saída digital 6"	%Q0.5	Bool	
"Auxiliar 7"	%M0.6	Bool	
"Saída digital 7"	%Q0.6	Bool	
"Auxiliar 8"	%M0.7	Bool	
"Saída digital 8"	%Q0.7	Bool	
"Auxiliar 9"	%M1.0	Bool	
"Saída digital 9"	%Q1.0	Bool	
"Auxiliar 10"	%M1.1	Bool	
"Saída digital 10"	%Q1.1	Bool	

ANEXO C

Roteiros de laboratório da disciplina SEL0431

Universidade de São Paulo Escola de Engenharia de São Carlos

Departamento de Engenharia Elétrica

SEL 0431 - Lab. Controle Proc. Industrial

Profs. Dennis Brandão

Controle da malha de temperatura

Duração da prática: 3 semanas

Atividade proposta:

1. Programar no CLP S7-1200 uma lógica de controle do tipo ON/OFF:

- O Endereço IP e a máscara de sub rede do CLP são **10.235.10.20** e **255.255.254.0** respectivamente.
- A variável manipulada (MV) deve ser o acionamento da resistência elétrica na caldeira.
- A referência ou se point (SP) de temperatura deve ser definido entre 28˚C e 30˚C (os valores lidos no CLP estão em hexadecimal, ver Tabela de Conversão).
- A variável primária ou de processo (PV) deve ser a temperatura dentro da caldeira (ver entrada IW64).
- A entrada IW65, referente à temperatura do reservatório 2, é apenas para conferência.

Introduzir comandos para o acionamento manual do atuador:

- Quando a chave seletora "manual/automático" da malha de temperatura estiver na posição manual, deve-se desabilitar o modo automático do controle.
- Nesta condição, ao ligar a chave de comando nomeada "aquecimento", o contator que liga a resistência elétrica dentro da caldeira deve ser acionado.
- Os respectivos sinaleiros deverão ser acionados adequadamente: quando a resistência da caldeira estiver ligada, o LED referente a esta (%Q0.3) deve estar aceso, e quando o modo manual for ligado, o LED da saída %Q0.2 deve acender.

2. Elaborar um relatório com os seguintes itens:

- Lógica Ladder comentada.
- Lista de variáveis.
- Limites de histerese do controlador ON/OFF.
- Configuração da rede de comunicação utilizada, se for o caso.
- Tabela de parâmetros da rede de comunicação utilizada, se for o caso.

Sugestão: Controlar de forma adequada o acionamento da válvula solenoide by-pass da caldeira para ter uma temperatura mais estável antes de misturar com a água do reservatório.

Lista de instrumentos da malha de temperatura:

- CLP S7-1200
- Dois disjuntores de 10 A
- Resistência elétrica de 40 Ω, deve sempre ser acionada com água na caldeira
- Fonte de alimentação da rede de 24Vdc
- Contator para acionamento
- Caldeira elétrica
- Sensor de nível vertical
- Reservatório
- Dois sensores de temperatura (pt1000) analógicos 0-10V
- Dois registros esféricos
- Válvula solenoide (by pass)
- Dois sinaleiros
- Duas chaves seletoras
- Botão de emergência (localizado no painel elétrico).

Observações e restrições importantes:

- Utilizar a entrada normalmente aberta "status bit da rede 4" na lógica do programa desabilitando qualquer saída do CLP caso esta entrada não estiver acionada.

- Antes de começar a programar, reconheça o circuito hidráulico na malha bem como todos os seus componentes.

- Antes de executar a lógica, atente para se os registros esféricos estão abertos nas tubulações da malha de controle.

- Atente para a torre sinalizadora, ela indica se a planta está pronta para operar (sinal verde ou amarelo) ou se há problema em algum equipamento (sinal vermelho).

- Atente para o indicador de temperatura, pois este possui um relé de proteção que é desativado se a temperatura dentro da caldeira atinge 50°C. Se isso acontecer, espere a temperatura da água restabelecer dentro dos padrões para voltar a operar.

- Nunca deixe a temperatura da água ultrapassar os 45 ˚C em nenhum ponto de medição, pois pode danificar a tubulação de PVC e outros equipamentos.

- Em caso de problemas para o funcionamento, verifique se os disjuntores das fases estão ligados.

-Verifique se a bomba 1 está em funcionamento e a válvula solenoide da caixa d´água 1 está desligada, isso irá garantir o fluxo de água para resfriar a caldeira.

Tutorial para este experimento

O tutorial "Programação do CLP Siemens S7-1200" tem o objetivo de indicar os passos necessários para se realizar um primeiro projeto de automação, utilizando-se o software da Siemens TIA Portal V11.

Tabela de conversão de variável para o medidor de temperatura:

Hexadecimal	Decimal	Temp (ºC)	Tensão (V) na entrada analógica
1AA6	6822	21,0	2,47
1B7F	7039	21,5	2,54
1CD4	7380	22,0	2,65
1D8D	7565	22,5	2,71
1E85	7813	23,0	2,79
20B3	8371	24,0	2,97
254D	9549	25,0	3,35
284C	10316	26,0	3,57
2A63	10851	27,0	3,77
2DC6	11718	28,0	4,05
3051	12369	29,0	4,26
329E	12958	30,0	4,45
348E	13454	31,0	4,61
3776	14198	32,0	4,85
3A7D	14973	33,0	5,10
3D07	15623	34,0	5,31
408E	16526	35,0	5,55
4334	17204	36,0	5,82
45A0	17824	37,0	6,02
48A7	18599	38,0	6,27
4B51	19281	39,0	6,49
4E57	20055	40,0	6,74
513F	20799	41,0	6,98

Mapa de entradas e saídas da CPU 1214C DC/DC/DC da Siemens

Entradas:

%I0.0 - Botão de parada de emergência
%I0.4 – Chave liga/desliga caldeira
%I0.5 – Chave manual/automático de temperatura
%I0.6 – Status bit rede 4
%IW64 – Medidor de temperatura (sensor de temperatura da caldeira)
%IW65 – Medidor de temperatura (sensor de temperatura do reservatório 2)

Saídas:

%Q0.0 – Contator da resistência da caldeira
%Q0.1 – Válvula solenoide by-pass da caldeira
%Q0.2 – Sinal luminoso manual/automático temperatura
%Q0.3 - Sinal luminoso liga/desliga caldeira

Universidade de São Paulo
Escola de Engenharia de São Carlos
Departamento de Engenharia Elétrica
SEL 0431 - Lab. Controle Proc. Industrial
Prof. Dennis Brandão

Controle da malha de nível

Duração da prática: 3 semanas

Atividade proposta:

1. Programar no CLP S7-1200 uma lógica de controle do tipo PID:

- O endereço IP e a máscara de sub rede do CLP são **10.235.10.21** e **255.255.254.0**, respectivamente.
- A variável manipulada (MV) deve ser o comando para o inversor de frequências do acionamento da bomba 2 (atuador).
- A variável primária ou de processo (PV) deve ser o nível no reservatório 1 ou o nível no reservatório 2. Escolha um deles.
- A referência ou set point (SP) deve ser definido entre 12 e 18 cm de altura da coluna d'água, caso o reservatório utilizado seja o reservatório 1; ou definido entre 12 e 18 cm, caso o reservatório utilizado seja o reservatório 2.
- O controlador também deverá atuar nas válvulas solenoides que estão posicionadas nas tubulações de entrada dos reservatórios e da caixa d´água 2, com a finalidade de manter o nível nos reservatórios.

Introduzir comandos para o acionamento manual do atuador:

- Quando a chave seletora "manual/automático" da malha de nível estiver na posição manual, deve-se desabilitar o modo automático do controlador PID.
- Nesta condição (manual), acionando-se a chave de comando nomeada "bomba 2", a bomba 2 atua na frequência do inversor que a aciona.
- Os respectivos sinaleiros do painel deverão ser acionados adequadamente: quando a bomba 2 estiver ligada, o sinal luminoso referente a ela (%Q0.2) deve estar ligado; quando o modo manual estiver acionado pelo botão, o sinal %Q0.5 deve acender.

2. Elaborar um relatório com os seguintes itens:

- Lógica Ladder comentada.
- Lista de variáveis.
- Tabela de ganhos do controlador.
- Configuração da rede de comunicação utilizada, se for o caso.
- Tabela de parâmetros da rede de comunicação utilizada, se for o caso.

Tutorial para este experimento

O tutorial "Programação do CLP Siemens S7-1200 com Profibus" tem o objetivo de indicar os passos necessários para se realizar um primeiro projeto de automação com rede Profibus, utilizando-se o software da Siemens TIA Portal V11.

Há um vídeo auxiliando a implementação do bloco PID no CLP S7-1200.

Lista de instrumentos da malha de nível:

- CLP S7-1200
- Disjuntor tripolar (para o inversor)
- Coupler Profibus DP/PA
- Caixa d'água
- Fonte de alimentação 24Vdc
- Dois reservatórios
- Transmissor ultrassônico de nível
- Transmissor de pressão diferencial
- Quatro registros esféricos
- Três válvulas solenoides
- Dois sinaleiros
- Duas chaves seletoras
- Botão de emergência (localizado no painel elétrico).

Algumas restrições e avisos devem ser observados:

- Utilizar a entrada normalmente aberta "status bit da rede 4" na lógica do programa desabilitando qualquer saída do CLP caso esta entrada não estiver acionada.

- Ler o tutorial "Ajuste matemático na malha de nível", ele corrige uma incompatibilidade de hardware entre o cartão de saída analógica do CLP e a entrada do inversor.

- Antes de começar a programar, reconheça o circuito hidráulico na malha bem como todos os seus componentes.

- Antes de executar a lógica atente para se os registros esféricos estão abertos nas tubulações da malha de controle.

- Ao ligar a bomba 2, pelo menos uma das válvulas solenoides (%Q0.3 ou %Q0.4) que direciona água para os reservatórios deverá estar aberta.

- Atente à torre sinalizadora, ela indica se a planta está pronta para operar (sinal verde ou amarelo) ou se há problema em algum equipamento.

Seguem as tabelas de conversão de nível para cada instrumento:

Medidor: Ultrassônico Endress+Hauser Prossonic T (reservatório 2)

Nível (cm)	Valor aferido (Real)
1,5	-340,36
5	-326,42
11	-306,8
30	-275

Medidor: Diferencial de Pressão Rosemount (reservatório 1)

Range (%)	Nível (cm)	Tensão (V)	Hexadecimal
100	30	9,54	66DF
95	28,5	9,19	6314
90	27	8,78	5EA2
85	25,5	8,44	5AF3
80	24	8,08	570C
75	22,5	7,64	5248
70	21	7,3	4E99
65	19,5	6,93	4A96
60	18	6,56	4694
55	16,5	6,2	42AD
50	15	5,8	3E58
45	13,5	5,43	3A55
40	12	5,07	366F
35	10,5	4,66	31FD
30	9	4,33	2E6A
25	7,5	3,94	2A30
20	6	3,6	2681
15	4,5	3,2	222B
10	3	2,81	1DF1
6	1,7	2,46	1A27
0	0	1,93	1468

***A altura da coluna d'água e os valores medidos crescem linearmente.**

Mapa de entradas e saídas da CPU 1214C DC/DC/DC da Siemens

Entradas-

%I0.1 – Chave liga/desliga Bomba 2
%I0.2 – Chave manual/automático nível
%I0.5 – Botão de parada de emergência
%I0.6 – Status bit rede 4
%IW64 – Medidor de nível (Diferencial de pressão)
%ID266 – Medidor de nível (Medidor ultrassônico de nível)

Saídas-

%Q0.0 – Válvula solenoide da caixa d'água 2
%Q0.1 – Aciona/para bomba 2
%Q0.2 – Sinal luminoso liga/desliga Bomba 2
%Q0.3 - Válvula solenoide da entrada do reservatório 1
%Q0.4 - Válvula solenoide da entrada do reservatório 2
%Q0.5 - Sinal luminoso manual/automático e de nível
%QW80 – Comando controle da bomba 2

Universidade de São Paulo
Escola de Engenharia de São Carlos
Departamento de Engenharia Elétrica
SEL 0431 - Lab. Controle Proc. Industrial
Prof. Dennis Brandão

Controle da malha de vazão

Duração da prática: 3 semanas

Atividade proposta:

1. Programar no CLP CITRINO uma lógica de controle do tipo PID:

- O Endereço IP e a máscara de sub rede do CLP são **10.235.10.202** e **255.255.254.0** respectivamente.
- A variável manipulada (MV) deve ser o comando para o inversor de frequências de acionamento da bomba 1 (atuador).
- A variável primária ou de processo (PV) deve ser a vazão da água medida pelo transmissor de vazão.
- A referência ou set point (SP) deverá ser definido entre 900L/h e 1000L/h.
- O controlador também deverá atuar nas válvulas solenoides que estão posicionadas nas tubulações de entrada dos reservatórios e da caixa d´água 1, com a finalidade de manter a vazão de água pelo transmissor.

Introduzir comandos na lógica para o acionamento manual do atuador:

- Quando a chave seletora "manual/automático" da malha de vazão estiver na posição manual, deve-se desabilitar o modo automático de controlador PID.
- Nesta condição (manual), acionando-se a chave seletora "bomba 1", deve- se atuar na frequência do inversor de acionamento da bomba 1.
- Os respectivos sinaleiros do painel deverão ser acionados adequadamente: DO3 ligado quando o modo manual estiver ativo e DO4 quando a bomba 1 estiver acionada.

2. Elaborar um relatório técnico com os seguintes itens:

- Lógica Ladder comentada
- Lista de variáveis
- Tabela de ganhos do controlador
- Configuração da rede de comunicação utilizada, se for o caso.
- Tabela de parâmetros da rede de comunicação utilizada, se for o caso.

Tutorial para este experimento

Os tutoriais "Configuração do CLP Citrino" e "Configuração de controlador PID no CLP Citrino" têm o objetivo de indicar os passos necessários para se realizar um primeiro projeto de automação utilizando-se o software Citrino tools 2.0.

Lista dos instrumentos da malha de vazão:

- CLP CITRINO (MCPU-1) composto pelos cartões:
  - Fonte de alimentação
  - CPU
  - Dois cartões de entrada (um analógico e um digital)
  - Dois cartões de saída (um analógico e um digital)
- Disjuntor tripolar
- Bomba hidráulica
- Fonte de alimentação de 24Vdc
- Transmissor volumétrico de água
- Caixa d'água
- Dois reservatórios
- Dois sensores indutivos
- Cinco registros esféricos
- Três válvulas solenoides de 24Vdc
- Dois sinaleiros
- Duas chaves seletoras
- Botão de emergência (localizado no painel elétrico).

Algumas restrições e avisos devem ser observados:

- Utilizar a entrada normalmente aberta "status bit da rede 4" na lógica do programa desabilitando todas as saídas do CLP caso esta entrada não estiver acionada.

- Antes de começar a programar, reconheça o circuito hidráulico na malha bem como todos os seus componentes.

- Antes de executar a lógica atente para se os registros esféricos estão abertos nas tubulações da malha de controle.

- Atente à torre sinalizadora, ela indica se a planta está pronta para operar (sinal verde ou amarelo) ou se há problema em algum equipamento (sinal vermelho).

Seguem as tabelas de conversão de nível para cada instrumento:

Medidor: Transmissor volumétrico

Frequência	Vazão (L/h)	Hexadecimal
20	315	10B5
21	336,5	1208
22	356,4	1320
23	375	1460
25	412	16F3
28	472	1A64
30	510,5	1CAD
30,1	511	1CCB
30,2	512,2	1CDE
34,5	590,9	21B9

37	634,5	2469
40	682	2757
45	761,7	2C41
48	807	2F0D
50	847	3184
54	900	34C9
58	950	37DE
60	1004	3B33
63,5	1014,8	3BDE
66	1045	3DBA

Sendo que a vazão cresce aproximadamente linear com os parâmetros citados.

Mapa de entradas e saídas do CLP Citrino

Entradas-

AI 1 – Medidor de vazão (Transmissor volumétrico)
DI 3 – Chave liga/desliga bomba 1
DI 4 – Botão de parada de emergência
DI 5 – Chave manual/automático de vazão
DI 6 – Status bit rede 4

Saídas-

AO 1 – Comando controle bomba 1
DO 2 – Aciona/para bomba 1
DO 3 – Sinal luminoso manual/automático de vazão
DO 4 – Sinal luminoso liga/desliga bomba 1
DO 5 - Válvula solenoide da caixa 1

Universidade de São Paulo

Escola de Engenharia de São Carlos Departamento de Engenharia Elétrica

SEL 0431 - Lab. Controle Proc. Industrial

Profs. Dennis Brandão

Sistema Supervisório

Duração da prática: 3 semanas

Atividade proposta:

1. Criar um sistema supervisório para as três malhas de controle no software Elipse SCADA.

2. Elaborar um relatório contendo os seguintes itens:

- Estrutura e elementos do sistema supervisório desenvolvido
- Instruções de uso do sistema supervisório
- Imagens das telas do sistema supervisório em operação
- Gráficos de tendências das três malhas de controle

Descrição da proposta:

- Para a supervisão de cada malha, deverá ser executado:

- Monitoramento e edição dos SPs
- Monitoramento das MVs:
    - %Q0.0 – contator da resistência da caldeira
    - %QW80 – comando para a bomba 2 (nível)
    - AO 1 – comando para a bomba 1 (vazão)
- Monitoramento das PVs
    - %IW64 – sensor de temperatura da caldeira
    - %IW64 – medidor de nível do reservatório 1
    - %ID266 – medidor de nível do reservatório 2
    - A1 1 – medidor de vazão
- Gráfico de tendência das PVs e SPs
- Campo de monitoramento do modo do controlador
- Geração de relatório em tabela ou gráfico
- Controle de acesso
- Registro de alarmes

Utilizar em cada malha estilos e formas diferentes de se representar as variáveis (gráficos, bargraphs, gauges, ou similares). Separar as malhas em páginas diferentes.

Algumas restrições e atenções devem ser respeitadas:

- Fique atento ao sinalizador luminoso, ele indica se a planta está realmente pronta para operar ou se há problema de segurança em alguma das malhas.

Tutorial para o 4º experimento

Os tutoriais “Configuração do Sistema Supervisório Elipse SCADA” têm o objetivo de indicar os passos necessários para se realizar um primeiro projeto de supervisório utilizando o software Elipse SCADA.

ANEXO **D**

Phoenix Contact RPI-BC 107,6 DEV-KIT KMGY - Instrução de Montagem

PHOENIX CONTACT GmbH & Co. KG
Flachsmarktstraße 8, 32825 Blomberg, Germany
Fax +49-(0)5235-341200, Phone +49-(0)5235-300
phoenixcontact.com

MNR 0195416 - 00 2016-04-11

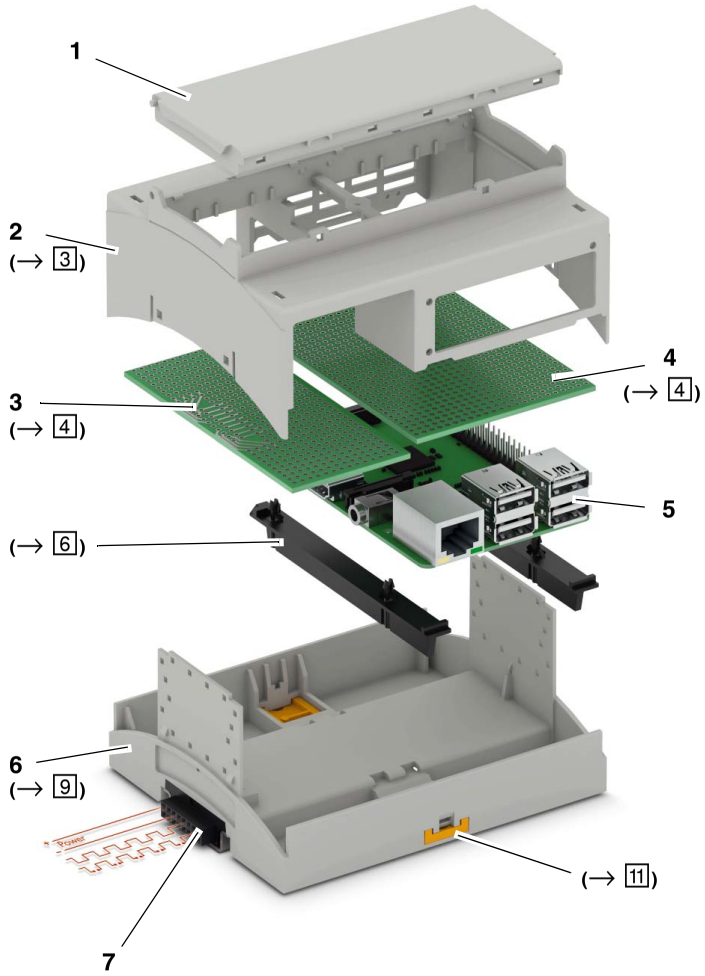
Documentation

DE Montageanleitung
EN Assembly Instructions
FR Instructions de montage
IT Istruzioni di montaggio
ES Instrucciones de montaje
PT Instrução de montagem
TR Montaj talimatları
RU Инструкция по монтажу

RPI-BC 107,6 DEV-KIT KMGY

2202874

2



1 BC 107,6 DKL R KMGY (Not removable!)
(option: 2896131, BC 107,6 DKL S TRANS (Removable))

2 RPI-BC 107,6 DEV-KIT KMGY P10

3 (option: 2202995, RPI-BC EXT-PCB HBUS SET)

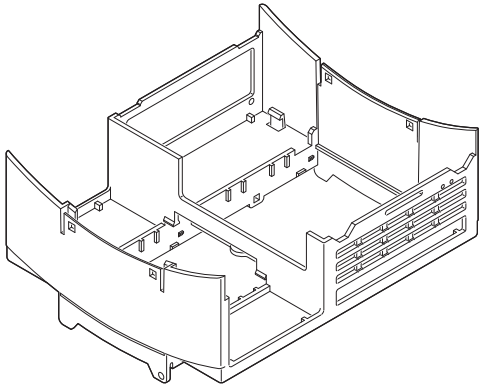
4 (option: 2202994, RPI-BC INT-PCB SET)

5 Raspberry Pi Board (not included)

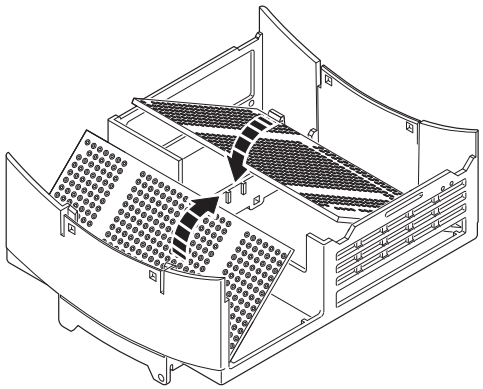
6 RPI-BC 107,6 UT HBUS KMGY
(option: 2896270, RPI-BC 107,6 UT HBUS BK)

7 (option: 2202876, HBUS 107,6-16P-1S BK P1)

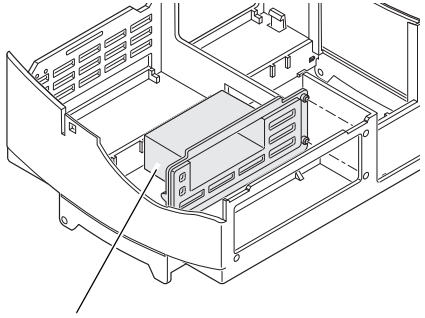
3



4



5

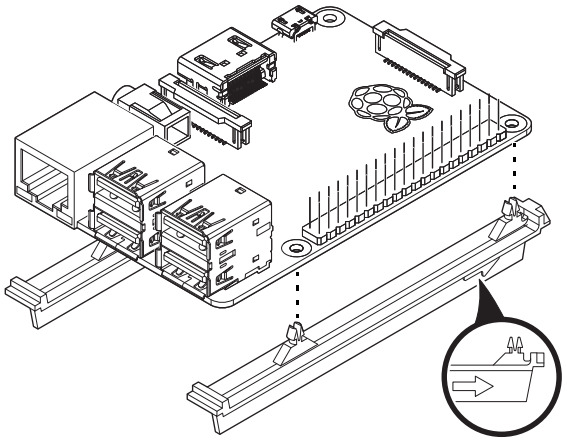


(option: 2202906, RPI-BC A+-ADAPTER ATGY)

1

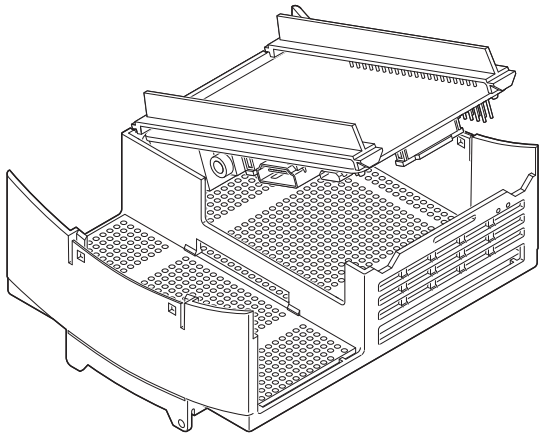
- Beachten Sie auch die Angaben im Datenblatt unter phoenixcontact.net/qr/2202874.
- Also observe the specifications in the data sheet at phoenixcontact.net/qr/2202874.
- Tenir également compte des indications figurant dans la fiche technique sous phoenixcontact.net/qr/2202874.
- Osservare anche le indicazioni fornite nella scheda tecnica disponibile nel sito phoenixcontact.net/qr/2202874.
- Observe asimismo las especificaciones de la hoja de características que hallará en phoenixcontact.net/qr/2202874.
- Observe também as informações contidas na ficha técnica em phoenixcontact.net/qr/2202874.
- phoenixcontact.net/qr/2202874 adresindeki veri sayfasında verilen teknik özellikler de dikkat edin.
- Также соблюдать требования технического паспорта на сайте phoenixcontact.net/qr/2202874.

6

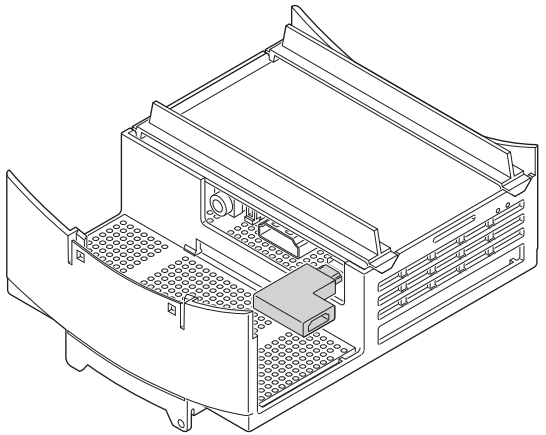


(→ 7)

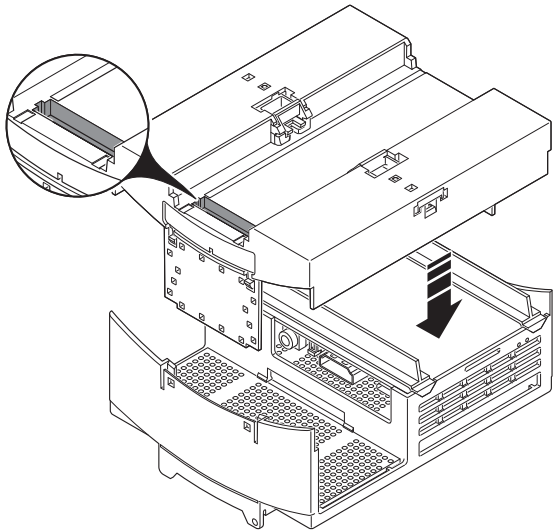
7



8



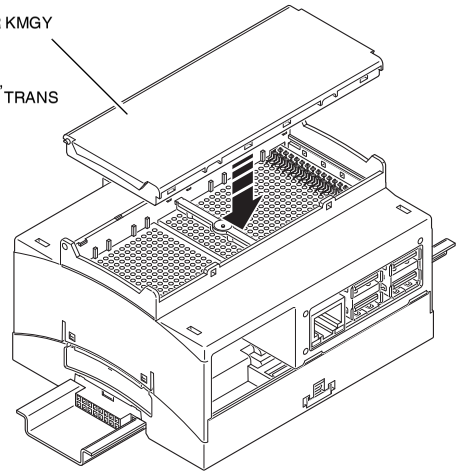
9



10

BC 107,6 DKL R KMGY

option: 2896131,
BC 107,6 DKL S TRANS



11