UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE PESQUISA E PÓS GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS
DE COMUNICAÇÃO E AUTOMAÇÃO

ALLISON CRISTIAN DA CUNHA

UTILIZAÇÃO DE HARDWARE LIVRE PARA IMPLEMENTAÇÃO DE CONTROLADOR DE BOMBEIO CENTRÍFUGO SUBMERSO APLICADO EM POÇOS PRODUTORES DE PETRÓLEO

MOSSORÓ
2017

ALLISON CRISTIAN DA CUNHA

UTILIZAÇÃO DE HARDWARE LIVRE PARA IMPLEMENTAÇÃO DE CONTROLADOR DE BOMBEIO CENTRÍFUGO SUBMERSO APLICADO EM POÇOS PRODUTORES DE PETRÓLEO

Dissertação apresentada ao Mestrado do Programa de Pós-Graduação em Sistemas de Comunicação e Automação da Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Mestre em Sistemas de Comunicação e Automação.

Linha de pesquisa: Automação e Controle.

Orientador: Prof. Dr. Marcelo Roberto Bastos Guerra Vale - UFERSA

Co-orientadora: Profa. Dra. Fabiana Karla de Oliveira Martins Varella Guerra - UFERSA

MOSSORÓ
2017

C972u Cunha, Allison Cristian da.
UTILIZAÇÃO DE HARDWARE LIVRE PARA IMPLEMENTAÇÃO DE CONTROLADOR DE BOMBEIO CENTRÍFUGO SUBMERSO APLICADO EM POÇOS PRODUTORES DE PETRÓLEO / Allison Cristian da Cunha. - 2017.
91 f. : il.

Orientador: Marcelo Roberto Bastos Guerra Vale.
Coorientadora: Fabiana Karla de Oliveira Martins Varella Guerra.
Dissertação (Mestrado) - Universidade Federal Rural do Semi-árido, Programa de Pós-graduação em , 2017.

1. bcs. 2. bombeio centrífugo submerso. 3. elevação artificial. 4. controle fuzzy. 5. hardware livre. I. Vale, Marcelo Roberto Bastos Guerra, orient. II. Guerra, Fabiana Karla de Oliveira Martins Varella, co-orient. III. Título.

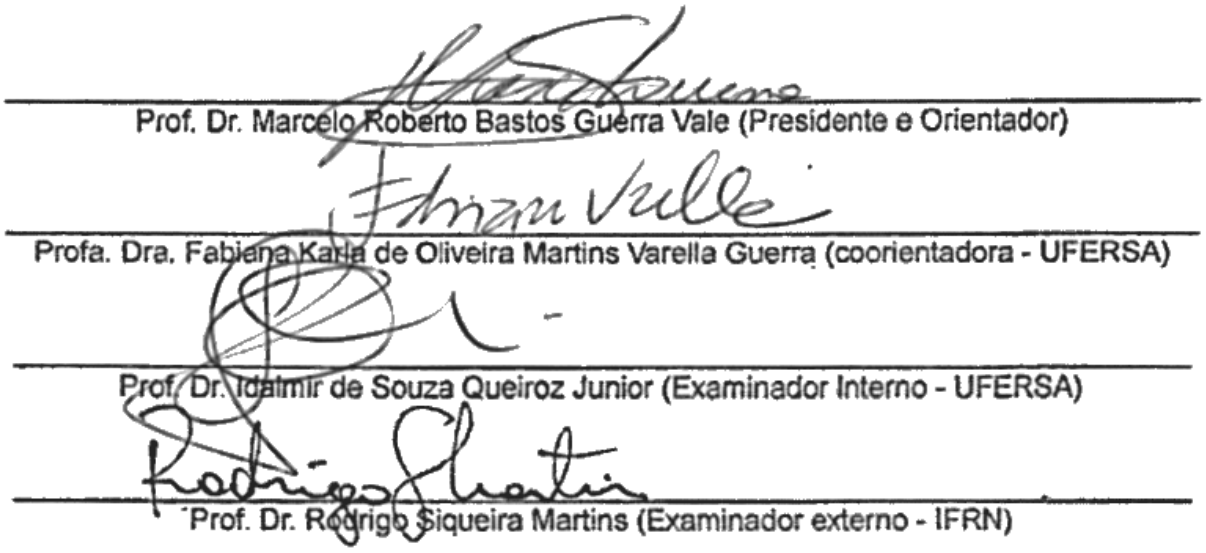
ALLISON CRISTIAN DA CUNHA

UTILIZAÇÃO DE HARDWARE LIVRE PARA IMPLEMENTAÇÃO DE CONTROLADOR DE BOMBEIO CENTRÍFUGO SUBMERSO APLICADO EM POÇOS PRODUTORES DE PETRÓLEO

Dissertação apresentada ao Mestrado do Programa de Pós-Graduação em Sistemas de Comunicação e Automação da Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Mestre em Sistemas de Comunicação e Automação.

Defendida em: 28 / 08 / 2017.

BANCA EXAMINADORA



DEDICATÓRIA

Dedico aos meus pais, esposa, irmãs, sobrinhos, afilhados, amigos e demais familiares.
A todos aqueles que lamentaram minha ausência, mas me esperaram,
compreensivamente, prestativamente, carinhosamente.
Um ideal conquistado é uma vitória, e essa vitória é para vocês!

AGRADECIMENTOS

A Deus, pela sorte e felicidade de ter uma vida maravilhosa, pela força de vontade indômita à mim presenteada, que me manteve seguindo mesmo quando acreditei não ser capaz.

Ao meu orientador Prof. Dr. Marcelo Roberto Bastos Guerra Vale, além da orientação, por me acolher quando precisei, e por impedir que eu desistisse nas vezes que eu hesitei.

Aos meus pais, Francisco Alves da Cunha e Edileuza Calixto da Cunha, a quem eu NUNCA serei capaz de retribuir tamanho amor, compreensão e sacrifício.

A minha esposa Sheila de Souza Xavier Ramos, por me amar, me ajudar, me motivar e caminhar ao meu lado.

Aos meus colegas do trabalho, confiando e me suportando nesta missão sem questionar minhas necessidades.

À Banca Examinadora pela oportunidade de aprovação.

A TODOS os meus amigos e familiares. MUITO OBRIGADO POR ESPERAREM!

“Corra até não poder mais, e então continue correndo.
O esforço constante é o maior atalho da vida”
Kamogawa Genji

RESUMO

A redução de consumo energético e do custo operacional na indústria de petróleo vem se tornando uma preocupação cada vez maior ao longo dos últimos anos. Nesse contexto a busca por novos métodos de produção de petróleo e a utilização de técnicas de controle integradas aos sistemas de elevação artificial para a otimização desse processo exercem um importante papel, e o desenvolvimento de controladores de bombeio é o principal objeto de estudo. O trabalho propõe a implementação de um sistema de controle para poços equipados com BCS (Bombeio Centrífugo Submerso), tendo como metodologia base o uso da plataforma Arduino®. O resultado desejado é o desenvolvimento de um sistema de controle de bombeio de baixo custo, com tecnologia de *software* que contemple fácil configuração e operação remota, e a validação do seu uso em um poço produtor de petróleo real. Os objetivos principais são a redução de custos através da utilização de *hardware* mais econômico, e a aceleração do desenvolvimento devida à migração para uma linguagem de programação de mais alto nível, ambos em relação às soluções de controle de bombeio atuais, baseadas no uso de CLP (Controlador Lógico Programável). Além disso, a proposta representa um grande passo para a disseminação do uso da filosofia de *hardware* livre para controle do processo de produção de petróleo, abrindo portas para a implementação de novos controladores para outros métodos de elevação.

Palavras-Chave: bcs, bombeio centrífugo submerso, elevação artificial, controle *fuzzy*, *hardware* livre.

ABSTRACT

The reduction of power consumption and operational costs in petroleum industry has become an increasingly concernment in the last years. Due to that the search for new petroleum production methods and the use of process controlling techniques integrated to its artificial lift systems for optimizing that process plays a main role, and the development of well pump controllers is the main study object. The work proposes the implementation of a control system for oil wells equipped with ESP (Electrical Submersible Pumping), having as main methodology the use of the Arduino® Platform. The desired result is the development of a low cost oil pump controller system, with software technology which brings easy configuration and remote operation, and validation of its use in a real oil well. The main goals are the cost reduction through the use of inexpensive hardware, and the the increase of developing speed by migrating to a higher level programming language, both compared to current well pumping controller solutions, based on use of PLC (Programmable Logic Controller). Other than that, the proposal represents a big step for the dissemination of open hardware philosophy for the petroleum production process control, opening doors for incoming implementation of new controllers for other artificial lift methods.

keywords: esp, electrical submersible pump, artificial lift, fuzzy controller, open hardware.

LISTA DE FIGURAS

NOMENCLATURA

Letras Latinas

d_c - Diâmetro dos cabos do motor

e_s - Erro de submergência

e_q - Erro de vazão

f - Frequência de operação do inversor

g - Aceleração da gravidade

I_f - Corrente após o secundário do trafo

I_{fs} - Incremento de frequência do controle de submergência

I_{fv} - Incremento de frequência do controle de vazão

I_T - Corrente de fase na saída do inversor

L_c - Comprimento do cabo do motor

L_s - Profundidade da bomba

p_{th} - Pressão na cabeça

p_{ch} - Pressão no revestimento

P - Potência ativa

p_d - Pressão na descarga da bomba

p_s - Pressão na sucção da bomba

q - Vazão

q_{max} - Vazão máxima da bomba

R_c - Resistência do cabo do motor

s - Submergência

T_m - Temperatura do motor

T_f - Temperatura do fluido

V_T - Tensão trifásica na saída do inversor

V_s - Tensão no secundário do trafo elevador

V_m - Tensão nos terminais do motor

Letras Gregas

Δp - Diferença entre p_d e p_a

ρ_0 - Massa específica do fluido

ρ - Resistividade do cobre

η_m - Rendimento do motor

φ - Ângulo de fase

η_b - Rendimento da bomba

Siglas

BCS - Bombeio Centrífugo Submerso

BEP - Best Efficiency Point

CLP - Controlador Lógico Programável

CPU - Central Processing Unit

CVPA - Coeficiente de Variação de Potência Ativa

DHS - Downhole Sensor

eFLL - embedded Fuzzy Logic Library

ESD - Electrostatic Discharge

ESP - Electrical Submersible Pump

IDE - Integrated Development Environment

OSI - Open System Interconnections

OSH - Open Source Hardware

PDU - Protocol Data Unit

PID - Proportional Integrative and Derivative

PIT - Pressure Indicator and Transducer

PLC - Programmable Logic Controller

PTC - Positive Temperature Coefficient

SCADA - Supervisory Control and Data Acquisition

TTL - Transistor-Transistor Logic

UART - Universal Asynchronous Receiver/Transmitter

UTR - Unidade Terminal Remota

VSD - Variable Speed Drive

SUMÁRIO

1 INTRODUÇÃO

A indústria do petróleo é composta por diversos cenários, compreendendo a exploração, produção, transporte, refino, distribuição e vendas. A produção de petróleo compreende toda infraestrutura e atividades necessárias para a elevação dos fluidos produzidos, até o seu escoamento para os pólos de processamento, onde os produtos serão acondicionado para transporte até as refinarias. O setor de elevação e escoamento compreende um conjunto de equipamentos e rotinas industriais que se espalham por todo o campo de produção, e que possuem potencial para utilização de tecnologias de otimização, controle e monitoramento remoto. Nesse âmbito os controladores de bombeio exercem um papel específico na indústria de petróleo, possuindo características de controle e automação específicas para cada método de elevação de petróleo utilizado. Por serem aplicações industriais bastante específicas, fragmentadas poço a poço, com diferentes fatores de segurança, condições de exposição ambiental e em grande parte livres de exposição à intempéries recorrentes de ambientes em plantas industriais, os estudos sobre inovações tecnológicas relacionados a essa indústria devem ser também específicos, para evitar que generalizações aplicáveis à outras indústrias penalizem seu desenvolvimento.

O cenário de produção de petróleo nacional se encontra atualmente em uma situação onde as margens para investimentos em inovações que não gerem retorno imediato são bastante estreitas, e por não existirem estudos suficientes capazes de quantificar detalhadamente a viabilidade dos investimentos, é bastante comum que projetos de implantação de técnicas de controle e otimização do bombeio de petróleo, sejam postos em segundo plano. Isso evidencia a necessidade de que sejam desenvolvidos projetos cada vez mais simples e mais econômicos.

A indústria de petróleo local dispõe de sistema de automação e monitoramento remoto dos sistemas de elevação utilizados em seus poços. Porém, nem todos os métodos de elevação dispõem de alguma técnica de controle para otimização da produção e segurança dos equipamentos, o que é visto como uma oportunidade para desenvolvimento de novas técnicas de controle utilizando a infra estrutura já utilizada para monitoramento.

Apesar de ser considerado economicamente viável, o uso de técnicas de controle no processo de produção de petróleo ainda não compõe o arranjo padrão de instalação de poços equipados com bombeio centrífugo submerso. Isso se deve parcialmente ao fato dessas técnicas serem recentes, mas em grande parte o motivo é o custo do investimento e a lentidão da adaptação cultural ao uso da técnica. Outro problema comum na indústria é a redução de contingente humano visando maior lucratividade, forçando um número cada vez menor de engenheiros a cuidar de números cada vez maiores de poços.

Os problemas listados apontam para a necessidade de desenvolvimento de controladores de baixo custo de aquisição, implantação e manutenção, para aumentar consideravelmente a margem de viabilidade econômica, e possibilitar finalmente a geração de controladores versáteis e eficientes no cenário de elevação de petróleo. O uso de técnicas de controle aumenta a autonomia do sistema de elevação, portanto reduz o tempo e os custos necessários para que sua operação seja mantida.

1.1 Justificativa

No processo de produção de petróleo, a viabilidade econômica de um poço depende do seu potencial de produção, da disponibilidade dos equipamentos e dos custos operacionais associados. Uma solução de controle pode aumentar a produção de um poço, além de estender a vida útil dos equipamentos e proporcionar custo operacional reduzido. Porém, quando são levados em consideração poços de baixo potencial de produção, a margem de viabilidade para novos investimentos se torna estreita, e portanto a solução é comumente rejeitada, embora tenha viabilidade comprovada. Algumas soluções baseadas no uso de CLP (Controlador Lógico Programável) em desenvolvimento no momento do início deste trabalho se mostram economicamente viáveis, mas o trabalho aqui proposto é uma alternativa que contribuirá significativamente com a implantação do sistema de controle idealizado, através da redução do investimento necessário, e da migração para uma arquitetura que tornará o desenvolvimento mais acessível. Como o custo de implantação está diretamente associado ao custo dos equipamentos e do desenvolvimento, o uso de *hardware* livre representa uma solução com maior capacidade de ser efetivamente implantada e amadurecida, além de

inspirar a expansão do uso dessa filosofia para a implantação de projetos de controle de baixo custo para outros métodos de elevação.

1.2 Objetivos

O trabalho propõe o desenvolvimento na plataforma Arduino®, de um painel de controle para variador de frequência, utilizado para bombeio centrífugo submerso, aplicado à poços produtores de petróleo. Para a técnica de controle, será implementado um controlador baseado em lógica *fuzzy*, com teoria disponível na literatura, capaz de controlar a frequência de alimentação do motor e consequentemente ajustar a rotação da BCS para o ponto de maior eficiência de bombeio possível automaticamente. Para atingir esse objetivo alguns aspectos preparatórios precisam ser abordados, como adequação de ambiente de simulação e especificação de *hardware* adicional suficiente para operação em campo. De forma ordenada e resumida , estes são os objetivos específicos almejados:

- Desenvolver de funcionalidades em simulador de BCS da PETROBRAS;
- Implementar programa de controle, e controlador fuzzy proposto por Costa (2012);
- Elaborar protótipo do controlador em plataforma Arduino®, e realizar testes de simulação;
- Montar painel com os recursos necessários para utilização em campo;
- Instalar em poço real, integrar a sistema SCADA para monitoramento e avaliar resultados.

2 REVISÃO DA LITERATURA

A composição final de um painel para controle e monitoramento remoto dos equipamentos de elevação de um poço faz uso de várias tecnologias. Neste capítulo serão abordadas cada uma das tecnologias utilizadas, explanando sua literatura, e seu uso nesta aplicação.

2.1 Introdução

Visando a contextualização da aplicação e da ferramenta, esta seção serão introduzidos aspectos relacionados à produção de petróleo e automação industrial, respectivamente.

2.1.1 Produção de petróleo

Poços produtores de petróleo são equipados para produzir os fluidos (óleo, gás e outros compostos) armazenados em um ou mais reservatórios, que são formações rochosas onde coincidem condições específicas para a geração, migração e contenção desses fluidos. Cada reservatório possui características de pressão de formação e porosidade das rochas distintas. O fluido de cada reservatório também terá características diferentes, como gás em solução e viscosidade. Após perfurado o poço, são abertos os canhoneados, que são fissuras laterais por onde o petróleo migrará de dentro da formação para o poço. A diferença entre a pressão da formação e a pressão no fundo do poço determinam a vazão do poço. A sua vazão, a viscosidade do seu fluido, e alguns outros parâmetros definem qual o método de elevação artificial, dentre os atualmente disponíveis, é o mais adequado (THOMAS, 2004).

No contexto de elevação de petróleo dois conceitos merecem atenção especial, por serem variáveis importantes para o controle utilizado neste trabalho, ilustrados na Figura 1.

- Nível dinâmico: Distância entre a superfície e o topo da coluna de fluido acumulada no espaço anular de um poço.

- Submergência: Distância entre o topo da coluna de fluido acumulada no espaço anular de um poço e a admissão da bomba equipada neste poço.

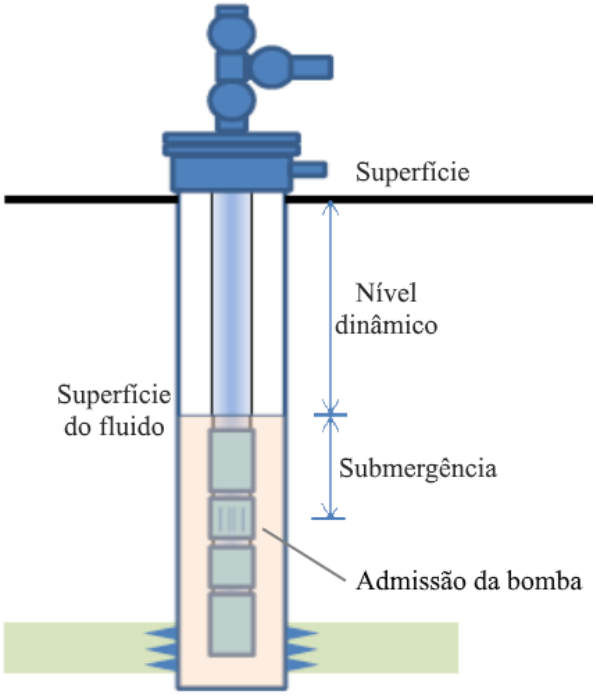


Figura 1: Nível dinâmico e submergência

Fonte: Adaptado de COSTA, 2012.

Vazões muito altas podem baixar muito a submergência e aumentar a entrada de gás livre na bomba e o risco de bombeio em vazio, condições que prejudicam a elevação e os equipamentos. Por outro lado, vazões muito baixas, podem aumentar muito a submergência, o que também aumenta a pressão contra os canhoneados no fundo do poço e consequentemente diminui a vazão do reservatório, reduzindo a sua produção. A técnica de controle abordada neste trabalho é aplicada ao método de elevação conhecido como Bombeio Centrífugo Submerso (BCS), com o propósito de manter o bombeio em níveis seguros de vazão e submergência.

2.1.2 Automação Industrial

O cenário industrial presenciou uma enorme revolução proveniente da utilização de computação em larga escala nos processos produtivos, por meio do uso de

microcontroladores. O Fenômeno da transformação de um processo industrial, por meio da substituição gradual de botões e lâmpadas por monitoramento e operação através de telas de computadores, e a migração da operação manual de um processo para o uso de ferramentas com diferentes graus de autonomia, para gerenciamento, controle e apoio à tomada de decisão em todas as camadas do processo, é definido pelo conceito de automação industrial (ALVES, 2005). O uso dessas ferramentas em conjunto dá origem a um sistema de automação, composto por uma série de sensores, atuadores, controladores e outros dispositivos conectados entre si por uma rede, os quais cooperam para realização de tarefas (LUGLI e SANTOS, 2010), além de trazer vantagens como agilidade, modularidade, confiabilidade e redução de custos, com relação aos processos manuais. Um sistema de automação é qualquer sistema que faz uso de computadores e substitui atividades manuais, beneficiando a segurança dos operadores, a velocidade e qualidade da produção, ou a redução dos custos, e assim contribuindo com melhorias para os principais pilares dos processos industriais (MORAES e CASTRUCCI, 2007). Os sistemas de automação modernos são compostos principalmente por dispositivos de campo, redes de comunicação e demais equipamentos e computadores (YAMAGUCHI, 2006). A automação de um processo permite através do uso de controladores, armazenar informações, processá-las, e atuar no processo de forma corretiva caso seja necessário (SOUZA, 2005).

O cenário atual da automação industrial é marcado pela predominância do uso de CLPs. Segundo Bolton (2015), um CLP é uma forma especial de controlador baseado em microprocessador que usa memória programável para armazenar instruções e para implementar funções como lógica, sequenciamento, temporização, contagem, e aritmética visando controlar máquinas e processos. O CLP nasceu praticamente dentro da indústria automobilística americana, especificamente na *Hydronic Division* da *General Motors*, em 1968, devido à grande dificuldade de mudar a lógica de controle de painéis de comando a cada mudança na linha de montagem. A arquitetura mais comum dentre os CLPs é composta por um *rack* que comporta uma fonte de alimentação, uma CPU, e *slots* para cartões de entrada ou saída analógica ou digital, comunicação entre outras funcionalidades.

Na indústria de petróleo o conceito de automação vem sendo aplicado de forma contínua e crescente em todas as camadas do processo. Assmann (2008) afirma ser de grande

valor um sistema local de controle do processo de elevação do petróleo capaz de mantê-lo no ponto ideal de operação, identificando as descontinuidades operacionais e retornando rapidamente ao ponto de operação após uma perturbação, de forma a recuperar a produção da forma mais rápida possível. Além destes aspectos outros importantes são: diagnosticar a causa de algum problema, transmitir ao sistema de supervisão sinais de alerta e até sugerir ações a serem tomadas. Baseado nesses aspectos o trabalho dará ênfase à otimização da produção, segurança de equipamentos, monitoramento remoto e identificação de perdas de produção.

2.2 Bombeio Centrífugo Submerso

Método de elevação comumente utilizado em campos de produção terrestres e marítimos, para poços de maiores vazões. O método consiste na entrega de energia mecânica ao fluido bombeado através de uma bomba centrífuga de múltiplos estágios acoplada a um motor elétrico inserido no poço, e alimentado pela rede elétrica através de cabos específicos para esse tipo de aplicação.

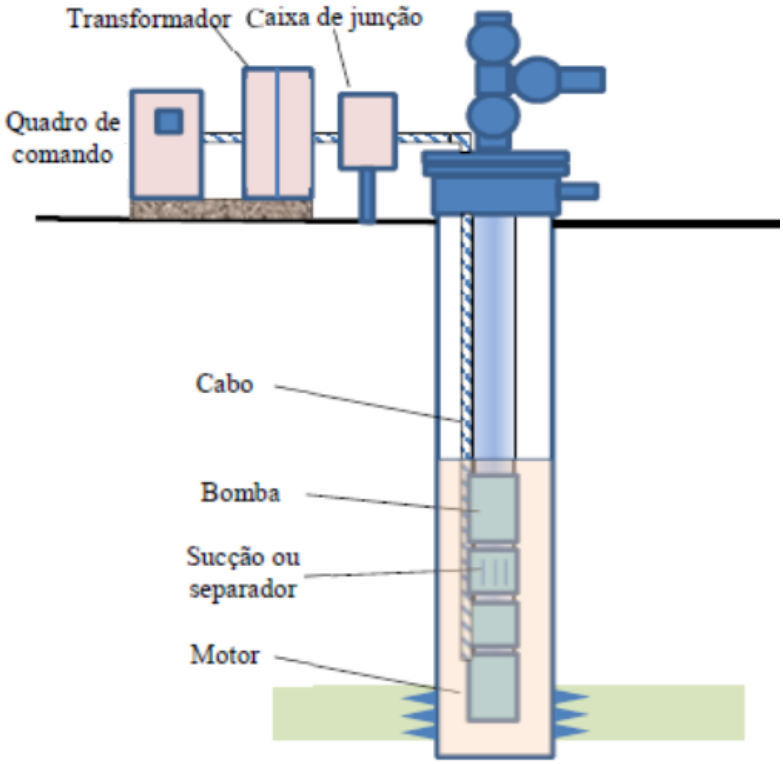


Figura 2 - Composição típica de um sistema de BCS

Fonte: Adaptado de COSTA, 2012.

O sistema para poços terrestres é tipicamente composto pelos elementos ilustrados na Figura 2 (TAKÁCS, 2009). Os equipamentos são descritos da seguinte forma:

- Quadro de comando: Elementos de controle e proteção do sistema, podendo ser equipado com variador de frequência, para suavização das partidas e ajuste da rotação do motor.

- Transformador: Necessário para a adequação da tensão da rede para ser transmitida ao motor, que opera com tensão superior.

- Caixa de junção: Compartimento com proteção contra intempéries que abriga a conexão do cabo elétrico de superfície com o cabo específico para BCS.

- Cabo de BCS: Cabeamento com isolação específica para as condições de tensão do motor, e de temperatura e pressão subsuperfície.

- Motor, Protetor, Sucção: Conjunto subsuperfície responsável por converter energia elétrica em energia mecânica, pela proteção do motor contra intrusão de impurezas, e admissão para dentro da bomba do fluido a ser elevado, respectivamente.

- Bomba Centrífuga Submersa: Responsável pela transmissão da energia mecânica oriunda do motor, para o fluido, através da rotação de múltiplos estágios. Cada estágio é um conjunto composto por um impelidor (rotor) e um difusor (estator). O impelidor transmite energia cinética ao fluido. O difusor direciona esse fluido em velocidade aumentada para o impelidor do estágio diretamente acima. Cada estágio adiciona uma parcela de pressão para o escoamento do fluido, e o processo se repete sucessivamente até o estágio mais acima (MAITELLI, 2010).

Todo método de elevação de petróleo possui alto custo associado à manutenção dos elementos subsuperfície. Vários fatores podem reduzir ou até comprometer a vida útil dos equipamentos. Para evitar que isso ocorra, geralmente esses métodos são monitorados por um corpo de engenharia qualificado, visando promover principalmente os seguintes cuidados (CAMILLERI & MACDONALD, 2010):

- Acompanhamento da temperatura do motor;
- Redução da quantidade de paradas e partidas;
- Presença de gás misturado ao fluido produzido.

Tais cuidados são baseados nas especificações técnicas fornecidas pelo fabricante de cada equipamento. Para BCS essas informações são fornecidas através de gráficos conhecidos como curvas características.

2.2.1 Curvas características de uma BCS

O dimensionamento de uma BCS consiste na seleção do conjunto adequado de bomba, motor e acessórios para produzir o máximo possível em um poço com características específicas. No que diz respeito à bomba, dada uma determinada vazão de fluido que o poço é capaz de fornecer, deseja-se saber qual a potência absorvida pela bomba e com qual rendimento energético e altura que a bomba conseguirá elevar aquele fluido. Essa altura é conhecida como *head*: altura manométrica em metros da coluna de fluido, definido como a energia por unidade de peso cedida ao fluido para uma dada vazão (MATOS & FALCO, 1998). Esses três critérios compõem um gráfico conhecido como curva característica da BCS, ilustrado na Figura 3.

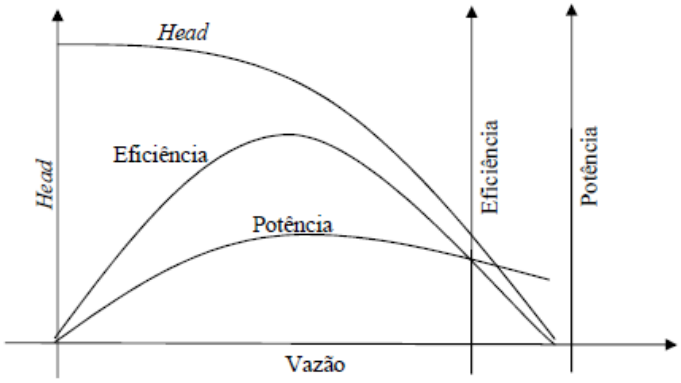


Figura 3 - Curvas características de uma BCS

Fonte: COSTA, 2012.

As curvas são fornecidas pelo fabricante da bomba, descrevendo seu comportamento bombeando água à 3500 rpm. O pico da curva de eficiência é conhecido como BEP (*Best Efficiency Point*). Cada fabricante define uma faixa de operação em torno do BEP, de forma que uma bomba adequada para um poço com determinado fluido, profundidade e vazão é aquela que consegue produzir o fluido dentro da faixa de operação.

2.2.2 Variadores de frequência

Vários fatores como a massa específica, a viscosidade, a pressão, a composição e a temperatura do fluido influenciam nos parâmetros descritos nas curvas características, tornando necessário para a equipe de engenharia do poço parar periodicamente, ajustar a vazão através restrição com válvula, recircular parte do fluido bombeado, ou ajustar a velocidade de rotação do motor através do uso de um variador de frequência, ou VSD (*Variable Speed Drive*). Esta última solução é a base para o uso da técnica de controle abordada neste trabalho.

A presença de gás na bomba pode ocasionar no aumento súbito da pressão no seu interior, prejudicando os estágios e consequentemente o bombeio. Este fenômeno prejudicial ao sistema é conhecido como *surging*. Para detectá-lo pode-se observar a taxa de variação da corrente elétrica do motor. Porém, segundo Takács (2009), qualquer instabilidade na rede elétrica que venha a ocasionar uma redução na tensão é compensada por um acréscimo na corrente elétrica fornecida para o motor, entretanto nesses casos a potência ativa se mantêm. Devido a isso o monitoramento do coeficiente de variação da potência ativa do motor (referido como CVPA ao longo deste trabalho) é uma prática mais adequada para a detecção.

Todos os possíveis problemas associados à produção de petróleo por BCS citados neste trabalho podem ser sanados ou amenizados com o uso de VSDs, variando-se a frequência de operação do motor com o intuito de ajustar vazão de produção, submergência e temperatura do motor. Quando se une o uso de um VSD a um sistema SCADA (*Supervisory Control And Data Acquisition*), que permite o monitoramento e a configuração remota dos equipamentos, torna-se muito mais fácil para o corpo de engenharia responsável por um poço de analisar sua performance e otimizar a sua produção e vida útil dos equipamentos,

representando um enorme salto desde as aplicações com velocidade fixa, sem monitoramento remoto e com controle através de válvulas manuais. Além disso o advento dos VSDs expandiu consideravelmente as pesquisas sobre aplicação de técnicas de controle em malha fechada de BCS (HAAPANEM & GAGNER, 2010).

2.3 *Hardware* Livre e Arduino®

Hardware Livre - às vezes abreviado por OSH (*Open Source Hardware*), é todo *hardware* cujo projeto fonte é publicamente disponível para qualquer pessoa usar, refabricar, modificar e revender. O movimento *Hardware* livre não é um novo conceito, mas sim uma revitalização de métodos históricos que foram sendo esquecidos conforme a fabricação moderna foi se tornando acessível, uma vez que ela produz hardware barato e eficiente, muito embora com limitações legais, e como resultado criou uma cultura de consumo, ao invés de uma cultura de desenvolvimento. A cultura OSH defende o compartilhamento, continuidade no desenvolvimento e a transparência plena. A preocupação com a transparência no desenvolvimento de alguma tecnologia, se dá devida à tendência natural de se tornarem cada vez mais incompreensíveis, ao passo que seus tamanhos diminuem, obrigando os usuários a confiarem nos fabricantes, e gerando uma relação de dependência. Em contraste, o *hardware* livre oferece liberdade de informação em um formato físico: os arquivos fonte do projeto (por exemplo diagramas, código, esquemas e instruções de montagem) são acessíveis e facilmente disponíveis para a reprodução dos mesmos (GIBB & ABADIE, 2014).

No contexto industrial, o uso do *hardware* livre ainda possui pouca participação, o que pode ser apenas uma questão cultural, tomando como exemplo a ascensão do uso do *software* livre pela indústria na última década. Mesmo contando com privilégios como transparência, baixo custo e acessibilidade, algumas barreiras precisam ser superadas, destacando-se como principais a robustez, confiabilidade e segurança. A validação da capacidade computacional do *hardware* de processamento principal, e a proposta de medidas para a superação das barreiras supracitadas e para aplicação satisfatória da filosofia de *hardware* livre na indústria de petróleo são objetivos deste trabalho.

A expressão mais popular da filosofia de *hardware* livre é através da utilização de plataformas de desenvolvimento e prototipagem. Elas consistem na associação de um microcontrolador a um conjunto de recursos de comunicação via diversos protocolos, e conexão com uma vasta variedade de sensores e atuadores. Estes possuem padronização satisfatória e estão disponíveis no mercado por custos bastante acessíveis. As plataformas também dispõem de facilidades de programação do microcontrolador na própria placa, através de linguagem de alto nível e ambiente de desenvolvimento também livre.

2.3.1 A plataforma de desenvolvimento Arduino®

Dentre as plataformas conhecidas, optou-se pela Arduino® (ARDUINO®, 2017a): Plataforma eletrônica de código aberto baseada em *hardware* e *software* de fácil utilização. Utiliza linguagem de programação *Arduino® Software Language* e dispões de suas próprias IDEs (*Integrated Development Environment*, termo utilizado para denominar o software de desenvolvimento). Nasceu no Ivrea Interaction Design Institute, com o intuito de ser uma ferramenta fácil para prototipagem rápida, com foco em estudantes sem muita experiência em eletrônica e programação. Apoiado pela filosofia de *software* livre e em constante evolução, rapidamente Arduino® se tornou uma ferramenta chave para o aprendizado de coisas novas. Pessoas interessadas conseguem começar algo apenas seguindo um passo-a-passo de um kit de desenvolvimento, ou compartilhando ideias online com outros membros da comunidade Arduino®. As principais vantagens sobre outras plataformas similares são:

- Baixo custo;
- Multiplataforma (Windows/Mac/Linux);
- Ambiente de programação simples;
- Software e Hardware livres e extensíveis.

O Arduino® se manteve em evolução constante, e de modo que seu uso foi se tornando popular, modelos com diferentes tamanhos e recursos foram criados. A Figura 4 ilustra uma placa Arduino® Mega 2560, modelo que será utilizado para desenvolvimento deste trabalho.

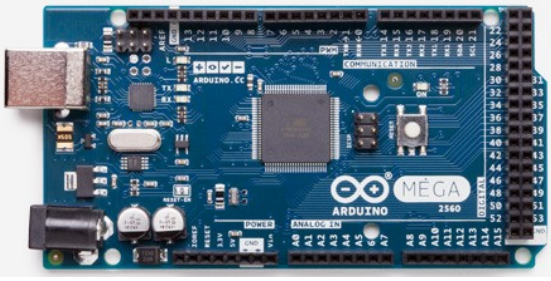


Figura 4 - Arduino® Mega 2560

Fonte: ARDUINO®, 2017e.

Há também uma vasta variedade de modelos, visando atender diferentes nichos de utilização (Robótica, *Internet of Things* e até moda), cada um com características e recursos diferentes. Além do Mega 2560, alguns dos principais modelos são: Uno, Leonard, 101, Esplora, Zero, Due, Mega ADK, Yún, Ethernet e MKR. Alguns dos principais recursos disponíveis, que podem ser encontrados já integrados, ou obtidos através de *shields* são:

- Entradas e saídas analógicas e digitais;
- Suporte a PWM (*Pulse Width Modulation*, ajuste do nível DC em uma porta digital pela modulação de pulsos de largura predefinida);
- Portas de comunicação serial, padrão RS232 TTL (padrão serial que utiliza sinais de 3,3V amplamente utilizado em dispositivos microcontrolados);
- Comunicação Ethernet, Wifi ou Bluetooth;
- Carregadores de baterias de LiPo (*Lithium Polymer*, baterias comuns nos *smartphones* atuais);
- Leitores de cartão de memória micro SD.

O site dispõe de uma tabela comparativa para que o modelo mais apropriado para cada aplicação possa ser selecionado de acordo com as funcionalidades necessárias. Caso outras funcionalidades além das integradas à placa principal venham a ser necessárias, placas adicionais, chamadas de *shields*, podem ser acopladas à principal, adicionando-a novos recursos sem a necessidade de solda ou outras modificações no *hardware*. Existe um vasto acervo de *shields* disponíveis atualmente, de baixo custo e de fácil aquisição pela internet.

Nesse projeto o Arduino® realizará basicamente 4 tarefas: consultar os equipamentos para leitura de variáveis, executar o controle *fuzzy* a partir delas, escrever a nova saída no VSD, e responder às consultas do sistema supervisório. Assim sendo, a única funcionalidade da placa efetivamente necessária será a comunicação serial, que será revisada mais adiante.

Por se tratar de uma plataforma de desenvolvimento, é recomendável que no final da fase de prototipagem um *hardware* específico seja dimensionado, com apenas os recursos que serão utilizados nesse projeto, e adicionado das funcionalidades de proteção necessárias. Todavia, embora o custo final de implantação em larga escala seja menor, há sempre um custo de tempo e dinheiro associado ao processo de customização. Para lidar com isso, o arranjo proposto neste trabalho visa prover recursos de condicionamento e segurança dos equipamentos. Isso pode ser alcançado através de acessórios periféricos, em maioria já utilizados e disponíveis comumente na indústria onde o trabalho será desenvolvido e inicialmente utilizado. Estas facilidades convergem com o principal objetivo deste trabalho: incorporar soluções de controle automático em poços produtores de petróleo com o menor custo e maior agilidade possível. Uma vez com um arranjo pronto disponível para uso, paralelo ao seu uso é recomendado e encorajado o desenvolvimento de novas versões deste projeto, com as funcionalidades periféricas integradas em um único hardware.

2.3.2 Hardware Livre na indústria

Com relação ao uso de hardware livre para desenvolvimento de controladores industriais, onde são de primordial importância atributos como confiabilidade, durabilidade, segurança, resistência à intempéries, fácil manutenção, intercambialidade, e tempo de resposta, existem duas frentes de desenvolvimento que valem à pena serem acrescidas à este contexto, e que serão expostas a seguir.

A primeira é a de desenvolvimento do OpenPLC Project (ALVES et al, 2014), ilustrado na Figura 5, que se propõe a utilizar o microcontrolador ATmega2560, o mesmo utilizado neste trabalho, para desenvolvimento de um controlador semelhante a um CLP.

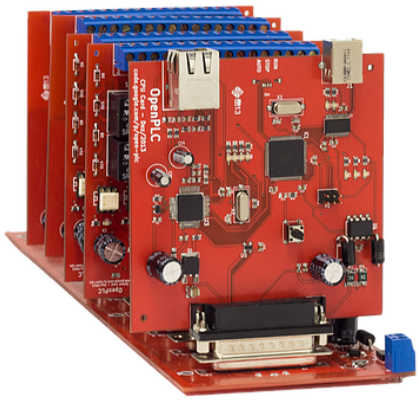


Figura 5 - The OpenPLC Prototype

Fonte: ALVES et al, 2014.

O protótipo é baseado em cartões de expansão, com proteção elétrica dos seus terminais, e com interface de desenvolvimento e linguagem de programação estilo *ladder*. Essa programação é feita através do *software* PLCOpen Editor, interface de desenvolvimento que atende às especificações da norma IEC-61131-3, que especifica os tipos de programação de CLP.

Além do *software* para programação e compilação, o projeto disponibiliza gratuitamente esquemas para construção de *hardware,* e *firmware* para utilização do *software* em algumas das mais populares plataformas de desenvolvimento atuais: Arduino® e Raspberry Pi, por exemplo. A relevância do OpenPLC no trabalho em questão está no fato dele ser um candidato em potencial à versão final do controlador, caso o mesmo se mostre economicamente viável em uma análise posterior.

A segunda diz respeito aos produtos da empresa Rugged Circuits (2017a), que produz versões com proteção de nível industrial dos modelos Uno e Mega, além de fornecer um exclusivo acervo de *cases*, *shields* e outros acessórios. A Figura 6 ilustra o modelo equivalente ao Mega. O *site* disponibiliza informações sobre os principais erros capazes de danificar um Arduino® (RUGGED, 2016d), e qual solução pode ser adotada, utilizada na sua versão do mesmo modelo, para impedir que o dano ocorra. Devido a isso o *site* é uma rica referência sobre utilização de proteção contra ESD (*Electrostatic Discharge*) e uso de PTC (*Positive Temperature Coefficient*), ambos recursos essenciais para aplicações de campo.

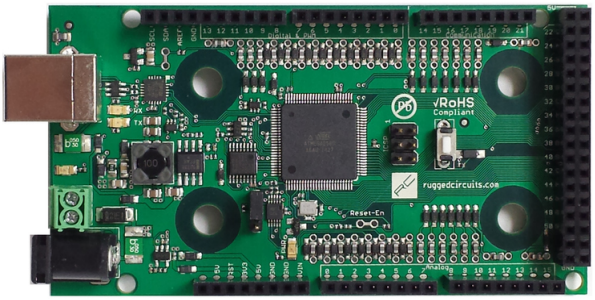


Figura 6 - Rugged MEGA, versão aprimorada do Arduino Mega 2560

Fonte: RUGGED, 2017b.

A relevância do Rugged MEGA neste trabalho também está no fato de ele ser um candidato em potencial à versão final do controlador, com as vantagens da compatibilidade total com o protótipo desenvolvido no Arduino® Mega, e sua ampla faixa de temperatura de operação, entre -10 a +85 ºC.

Além das limitações relacionadas à robustez e uso industrial, outro possível problema para o desenvolvimento de sistemas de maior porte na plataforma Arduino® é a limitação da capacidade de memória dos microcontroladores. As possíveis soluções e medidas adotadas serão discutidas na seção 3.3.

2.3.3 A linguagem de programação do Arduino®

A linguagem principal utilizada para a programação do Arduino® é modelada a partir de uma linguagem denominada *Processing*, baseada em linguagem *C*, que consolida facilidades para desenvolvimento e utilização dos recursos de *hardware.* Sua sintaxe é bastante similar à da linguagem C e suas variantes, tendo como característica principal o estilo e as facilidades de uma linguagem de alto nível, mas com um conjunto de métodos e funções de fácil configuração e acesso ao hardware, o que pode ser considerado uma vantagem ante as linguagens para software embarcado predecessoras. O projeto contendo o conjunto de arquivos que compõem o programa é conhecido popularmente na comunidade Arduino® como *Sketch* (“rascunho” em inglês). A sua compilação em código de máquina funciona da seguinte forma: os arquivos de código do *sketch* são consolidados e convertidos em outro código, em linguagem C. Esse novo código é passado ao compilador *open source* avr-gcc, que por sua

vez realiza a tradução final em código de máquina, compreendido pelo microcontrolador (BANZI & SHILOH, 2014).

A estrutura do *sketch* possui dois métodos principais, ilustrados na Figura 7.

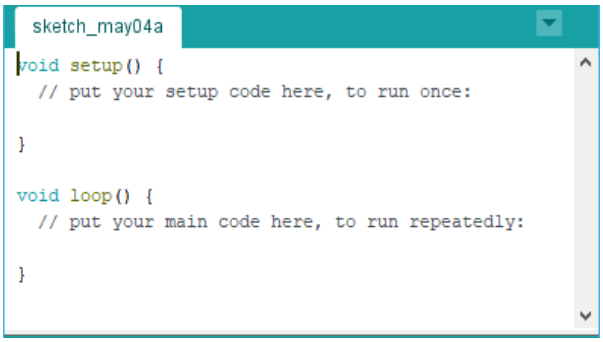


Figura 7 - *Sketch* padrão do Arduino® IDE

Fonte: ARDUINO®, 2017b.

- *Setup*( ): A função *setup* é chamada quando o programa inicia. Deve ser usada para iniciar variáveis, configurar pinos, instanciar bibliotecas, e qualquer outra funcionalidade que precise ser executada apenas uma vez, após cada inicialização ou *reset* da placa Arduino®.

- *Loop*( ): Após a execução da função *setup*, a função *loop* é iniciada e executada repetidamente, permitindo o programa mudar e responder. Ela é a rotina principal que será executada enquanto o Arduino® estiver funcionando apropriadamente, e deve ser usada para a execução do programa que controlará a placa.

O código ilustrado na Figura 8 é um exemplo simples de programa que lê a entrada digital 7, definida na variável *'inPin'* na segunda linha, e escreve seu valor na saída digital 13, definida na variável *'ledPin'* na primeira linha. O exemplo permite acender um LED ao pressionar um botão.

```
int ledPin = 13;    // o LED do exemplo será conectado ao pino 13
int inPin = 7;      // o botão do exemplo será conectado ao pino 7
int val = 0;        // variável para guardar o valor lido

void setup()
{
  pinMode(ledPin, OUTPUT);      // configura o pino 13 como saída
  pinMode(inPin, INPUT);        // configura o pino 7 como entrada
}

void loop()
{
  val = digitalRead(inPin);     // lê o pino ligado ao botão
  digitalWrite(ledPin, val);    // escreve o valor do botão no pino ligado ao LED
}
```

Figura 8 - Exemplo de *sketch* simples

Fonte: ARDUINO®, 2017b.

Para desenvolvimento dos *Sketches*, a comunidade Arduino® disponibiliza dois ambientes de desenvolvimento principais: o Arduino® *Desktop IDE*, e o Arduino® *Web Editor*. Este segundo tem parte de sua interface ilustrada na Figura 9.

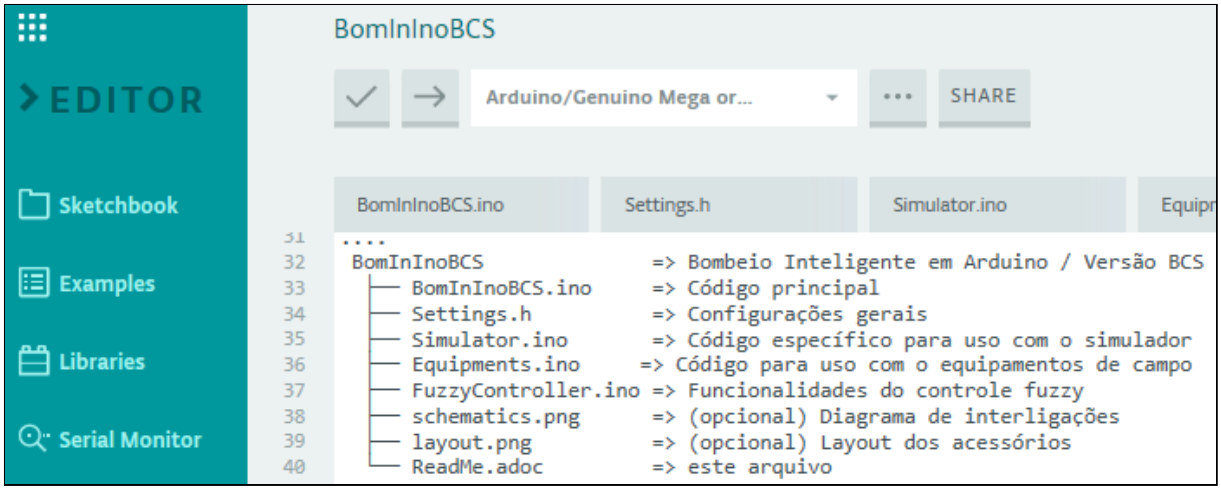


Figura 9 - Arduino® Web Editor

Fonte: Autoria Própria.

As duas interfaces são essencialmente similares, entretanto existem algumas vantagens sobre a interface *web* que motivaram seu uso neste projeto:

- Os *sketches* são salvos automaticamente em nuvem, o que garante segurança contra perda de dados e facilidade de acesso em outros computadores;

- A interface *web* requer apenas instalação do serviço agente que realizará a comunicação do *browser* (*software* de navegação na *internet*) com o *hardware*. A IDE requer download e instalação maiores.

- As bibliotecas disponibilizadas pela comunidade são constantemente atualizadas, e na *Desktop IDE* fica sob responsabilidade do usuário. Já no *Web Editor* tem-se a garantia de que seu projeto estará sempre utilizando as versões mais atualizadas disponíveis.

O Arduino® Web Editor é uma das ferramentas da plataforma *online* Arduino® Create, que consolida os principais serviços utilizados pela comunidade Arduino®: desenvolvimento, loja, tutoriais, repositório de projetos, serviços de nuvem e fórums de discussão.

2.4 Comunicação e Instrumentação

Nesta seção serão descritos o protocolo serial TTL, o RS232-C (*Recommended Standard 232*) e sua variante RS485.

O Arduino® e a maioria dos microcontroladores de hoje em dia possuem UARTs (*Universally Asynchronous Receiver/Transmitter*), que são módulos de *hardware* de comunicação utilizados para receber e transmitir dados serialmente. UARTs transmitem um bit por vez, em uma taxa específica. Esse método de comunicação serial é comumente conhecido como TTL (SPARKFUN, 2010).

O RS232-C difere do TTL apenas a nível de hardware: O nível lógico ALTO é representado por uma tensão negativa, entre -3V e -25V, e o nível BAIXO por uma tensão positiva entre +3V e +25V. Na maioria dos computadores esses sinais flutuam de -13V para +13V. A Figura 10 ilustra um exemplo de transmissão do byte 10101010 nos dois protocolos.

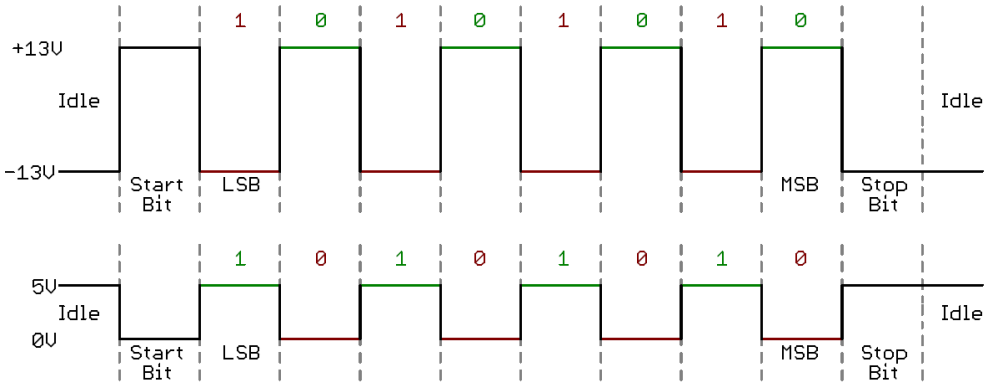


Figura 10 - Exemplos de sinais RS232-C e TTL

Fonte: SPARKFUN, 2010.

Por utilizar limiares de tensão mais extremos, o RS232-C se torna menos suscetível a ruído, interferência e atenuação que o TTL, e portanto consegue atingir maiores distâncias ou ser usado em cenários com ambientes externos mais ruidosos.

Quando a aplicação demanda maiores distâncias, taxas de comunicação ou comunicação multiponto, utiliza-se o protocolo RS485 (nome popular convencionado da interface serial especificada pelo documento TIA/EIA-485). A Figura 11 ilustra os sinais (A e B) dos dois canais de uma rede RS485, e o sinal diferencial (diff) que será justamente a diferença de tensão entre os dois.

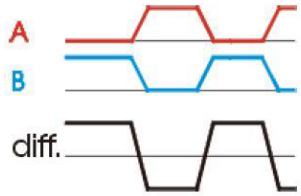


Figura 11 - Exemplo de sinal RS485

Fonte: EMCU, 2009.

Em cenários mais agressivos, onde os *drivers* podem estar sofrendo influência de flutuações eletromagnéticas, sujeitos a possíveis descargas elétricas, ou até mesmo para evitar ligações erradas é importante que os terminais estejam protegidos. Essa proteção por ser alcançada através de combinações do uso de resistores como divisores de tensão e limitadores

de corrente, diodos *schottky* como reguladores e bloqueadores de corrente reversa, e fototransistores como optoacopladores.

O conjunto elaborado neste trabalho utilizará todos os 3 protocolos citados nesta sessão. Esses protocolos são caracterizados como de camada física, especificando as características elétricas dos sinais de comunicação. Para formatação das mensagens a serem enviadas entre os equipamentos, faz-se necessário a utilização de um protocolo de camada superior. Nesta aplicação será usado o Modbus: protocolo da camada de aplicação comum entre os equipamentos que constituem o sistema. Suas principais características serão abordadas na sessão seguinte.

2.4.1 Protocolo Modbus

Uma vez que o protocolo de comunicação comum entre os equipamentos mais comumente utilizados em poços de BCS *on-shore* atualmente é o Modbus, se fez necessária a implementação de suporte a este protocolo, e nesta seção serão descritas as suas principais características.

O Modbus é um protocolo de mensagem para comunicação entre dispositivos. Com relação ao modelo OSI (*Open Systems Interconnection*, modelo de redes mantido pela norma ISO/IEC 7498-1), o Modbus atua na camada de aplicação, e pode ser usado em diferentes tipos de redes. A Figura 12 ilustra a pilha de comunicação Modbus e os principais exemplos de protocolos de camadas inferiores.

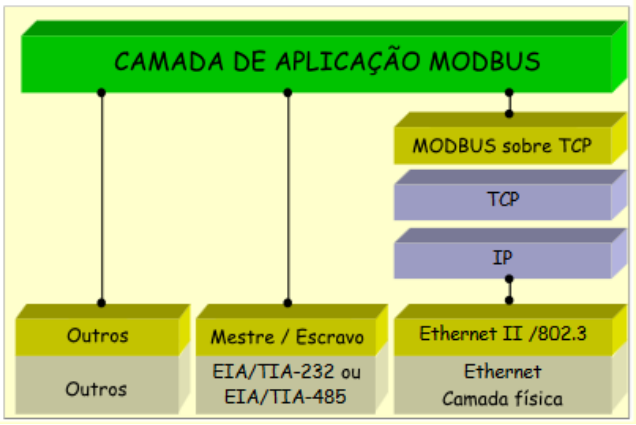


Figura 12 - Pilha de comunicação do protocolo Modbus

Fonte: Adaptado de MODBUS, 2012.

Uma das principais características do Modbus é a facilidade de utilização independente dos protocolos das camadas inferiores, sendo necessário apenas o mínimo de *overhead* para qualquer encapsulamento do datagrama original, que é composto basicamente pelo código da função utilizada, e os dados transmitidos como parâmetros dessa função. Adicionalmente são integrados ao datagrama informações a nível de aplicação, como dados de endereçamento e checagem de erro. A Figura 13 descreve o datagrama convencional utilizado em comunicação serial: PDU (*Protocol Data Unit*) representa o datagrama original Modbus, e ADU (*Application Data Unit*) representa a mensagem já encapsulada de acordo com a aplicação.

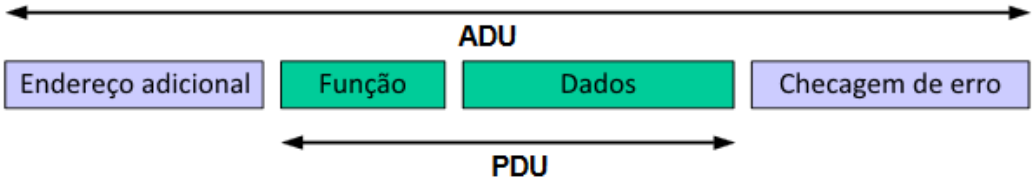


Figura 13 - Datagrama Modbus convencional para comunicação serial

Fonte: Adaptado de MODBUS, 2012.

Como todas as 3 interfaces Modbus que serão implementadas neste trabalho utilizam comunicação serial, então todas usarão a mesma codificação, que é o Modbus RTU. O termo RTU vem de *Remote Terminal Unit*, utilizado para os terminais de comunicação serial. Existe ainda uma segunda codificação serial menos utilizada denominada Modbus ASCII, que utiliza caracteres ASCII ao invés de binários, e com isso consegue robustez em cenários de maior latência, mas ao custo do dobro da banda (um caractere binário/hexadecimal possui 4 bits enquanto um ASCII possui 8).

Como em toda interface de comunicação serial, deve ser feita a configuração apropriada de taxa de transmissão, paridade, número de *bits* de início e de parada do canal, em cada nó da rede, para que os sinais recebidos sejam interpretados apropriadamente. Além disso, para evitar que o mestre espere indefinidamente por uma resposta que pode nunca chegar, é comum o uso de parâmetros de *timeout*, tempo de espera por resposta, e *retry*, número de tentativas antes de desistir da comunicação. A implementação do mestre e do servidor também difere em outros aspectos: O mestre precisa conhecer os endereços, as funções suportadas e os mapas de registradores dos escravos a serem lidos/escritos. Os

escravos, por sua vezes, precisam da implementação do mapa de registradores, da detecção de chegada de solicitação, e do seu processamento de acordo com as funções suportadas e registradores disponíveis. Neste trabalho serão desenvolvidas 3 interfaces Modbus, para uso em 3 redes distintas:

- Mestre no Arduino®: para comunicação do controlador com um computador durante as simulações, e para que o controlador possa realizar a aquisição de dados do VSD, DHS (*Downhole Sensor*, ou sensor de fundo) e *hub* (concentrador) de entradas analógicas, escrita da referência de saída do controlador no VSD, e eventual escrita de parâmetros de configuração em cada um dos 3 escravos na aplicação real.
- Escravo em simulador: para processamento das requisições do mestre no Arduino® durante as simulações, e adicionalmente para servir de interface de configuração do controlador:
- Escravo no Arduino®: para que o controlador possa responder à solicitações do sistema SCADA responsável pelo monitoramento do poço na aplicação real.

2.4.2 Entradas analógicas

A instrumentação de superfície utilizada nesta aplicação, que compreende um par de PITs (Pressure Indicator & Transmitter), utiliza o padrão de 4-20mA + HART: protocolo que sobrepõe comunicação digital ao padrão de 4-20mA, injetando séries de oscilações de tensão na forma de ondas senoidais, cuja média de tensão é 0, e portanto não interferindo e nem requerendo a interrupção do sinal original de 4-20mA do instrumento (HOWART, 1994). A leitura do instrumento é escalonada entre 4 e 20mA. É usado 4mA (zero vivo) ao invés de 0mA (zero morto) para que uma leitura real do valor 0, possua valor em mA diferente de um circuito aberto ou um equipamento inoperante. A configuração da escala de medição do instrumento é feita através de equipamentos de interfaces de configuração HART.

A entrada analógica do Arduino® é feita com um conversor A/D de resolução de 10 bits, o que significa que sinais entre 4 e 20 mA serão convertidos para valores entre 0 e 1023 (ARDUINO®, 2017c). A escala configurada no PIT precisa ser conhecida pelo software

executado no Arduino®, para que a conversão desse valor no correspondente à grandeza física seja possível.

Para que esse padrão industrial seja utilizado em uma placa Arduino®, cujas entradas analógicas permitem valores de tensão entre 0 e 5V, um resistor de 250 ohm precisa ser ligado entre os terminais do instrumento para que uma queda de tensão entre 1 (4mA x 250ohms) e 5 (20mA x 250ohms) volts seja gerada, conforme ilustrado na Figura 14.

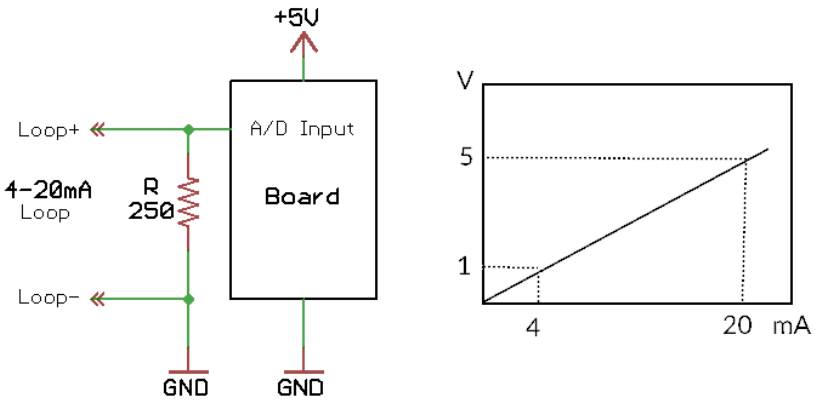


Figura 14 - Adequação de instrumento 4 a 20 à entrada analógica de um Arduino®

Fonte: Adaptado de RUGGED, 2017a.

Além disso, filtros RC para remoção de ruído, diodos *schottky* para proteção de sobretensão, e circuitos com optoacoplamento para isolação galvânica, projetados para esses níveis de tensão e corrente podem ser utilizados em conjunto para aumentar a proteção contra erros de instalação e intempéries do uso industrial. A decisão de uso deve ser calculada a partir da avaliação da qualidade do sinal, e dos riscos associados à cada sistema.

Na aplicação em questão, visando a proteção do *hardware* principal, considerando a vantagem de usar acessórios industriais de fácil aquisição e baixo custo, e o benefício de poder fazer leitura remota dos instrumentos na ausência do controlador, optou-se pela utilização de um *hub* de entradas analógicas com comunicação serial Modbus.

2.5 Técnica de controle

Nesta seção serão abordados os conceitos relacionados à lógica de controle utilizada, e alguns conceitos relacionados à terminologia utilizada serão definidos. Com o intuito do

melhor entendimento das seções seguintes, a desambiguação de dois termos semelhantes e frequentemente utilizados se faz necessária:

- Controle: Termo teórico que se refere apenas à técnica utilizada para a realização do ajuste automático de um sistema, independente das funcionalidades que o envolvem para que ele se adeque a esse sistema. Por exemplo: controle *fuzzy*.
- Controlador: Termo prático que se refere ao conjunto composto por equipamentos, recursos de instrumentação, comunicação, lógica de segurança e técnica de controle automático de algum sistema. Por exemplo: Controlador *fuzzy* para BCS em Arduino®.

2.5.1 Lógica *fuzzy*

Conceito de lógica não binária proposto por (ZADEH, 1965) nos Estados Unidos, base da teoria dos conjuntos *fuzzy*, inspirado pela necessidade de resolução de diversos paradoxos existentes, propostos por filósofos e outros estudiosos, relacionados à não discreticidade da realidade, e não resolvíveis pela lógica binária tradicional. Também conhecida como lógica difusa ou nebulosa. É a área da lógica computacional que visa mapear a forma humana de solucionar problemas em um universo lógico (SIMÕES e SHAW, 2007). A teoria de conjuntos *fuzzy* possui sua própria lista de conceitos e definições. Distingue-se da teoria de conjuntos tradicional, onde cada elemento pertence ou não a um conjunto, principalmente pela introdução do conceito de pertinência dos elementos sobre os conjuntos.

Para exemplificar melhor, considere um determinado universo *U, x* como elemento de *U*, e *A* como subconjunto de *U.* Os conjuntos tradicionais apenas nomeiam os seus elementos:

$$A = \{x_1, x_2, ..., x_n\}$$

ou os especifica através de uma ou mais regras *C* na forma:

$$A = \{x \mid x \vdash C\}$$

Definindo-se uma função de pertinência μ sobre o subconjunto *A*, que representa o quanto um elemento *x* faz parte dele, ela teria a forma:

$$\mu_A(x) = \left\{ \begin{array}{l} 0, \textit{ se e somente se } x \notin A \\ 1, \textit{ se e somente se } x \in A \end{array} \right\}$$

Isso porque na teoria tradicional, um elemento ou pertence ou não a um conjunto. Em termos matemáticos a função μ_A realiza o seguinte mapeamento:

$$\mu_A : x \rightarrow \{0,1\}$$

No entanto, quando se trata de problemas do mundo real, o conceito de fronteira de um subconjunto não é discreto, mas sim contínuo, o que evidencia a necessidade de uma representação também contínua dos seus elementos.

Nos conjuntos *fuzzy* os componentes são definidos pelo par elemento e função de pertinência que atribui um valor de afinidade do elemento ao conjunto:

$$A = \{(x_1, \mu_A(x_1)), (x_2, \mu_A(x_2)), ..., (x_n, \mu_A(x_n))\}$$

com as funções de pertinência sendo qualquer função que resulte em valores entre 0 e 1, ou seja, que seja compreendida pelo seguinte mapeamento:

$$\mu_A : x \rightarrow [0,1]$$

o que permite que cada elemento seja tratado de forma proporcional à sua afinidade com aquele conjunto. Isso resolve o problema da representação discreta, mas introduz a descaracterização das operações com conjuntos. Isso torna necessária uma redefinição das principais operações, que são o complemento, a união e a interseção.

No caso do complemento o mesmo princípio é válido, e portanto a operação pode ser descrita da seguinte forma:

$$\mu_{\underline{A}}(x) = 1 - \mu_A(x)$$

Para a união, qualquer operação que satisfaça a condição "$f(x,y) = 1$, se QUALQUER argumento for igual a 1" pode ser utilizada. A operação que satisfaz essa condição mais comumente utilizada é a função *max*, que retorna o maior dentre os argumentos:

$$\mu_{A \cup B}(x) = max(\mu_A(x), \mu_B(x))$$

Para a intersecção, qualquer operação que satisfaça a condição "$f(x,y) = 1$, se AMBOS os argumentos forem iguais a 1" pode ser utilizada. A operação que satisfaz essa condição mais comumente utilizada é a função *min*, que retorna o menor dentre os argumentos:

$$\mu_{A \cap B}(x) = min(\mu_A(x), \mu_B(x))$$

A lógica *fuzzy* introduz ainda diversos outros termos, conceitos e definições. Ela também possui vasta gama de aplicações relacionadas às áreas de modelagem, controle e inteligência artificial, além de muitas vezes ser aplicada em conjunto com outras técnicas computacionais, como por exemplo controladores PID e redes neurais artificiais.

Esta dissertação propõe-se a comprovar a viabilidade do uso de *hardware* livre para a implementação de sistemas de controle na indústria do petróleo, em particular controladores de bombeio, o que justifica a abordagem secundária e resumida da técnica utilizada para o controle. Entretanto a implementação de controladores de bombeio *fuzzy* aplicado a poços produtores de petróleo é um tema ainda pouco explorado, com base na exclusividade da proposta de Costa (2012). O mesmo ocorre com o desenvolvimento de controladores *fuzzy* para dispositivos embarcados, normalmente devido às limitações desse tipo de *hardware*. Portanto, as seções seguintes deste trabalho terão ênfase no desenvolvimento de controle *fuzzy* utilizado em controladores de bombeio desenvolvidos com utilização de *hardware* livre.

2.5.2 Controle *fuzzy*

Técnica de controle não linear baseada em lógica *fuzzy*. Enquanto os controles clássicos são eficientes em sistemas de controle lineares, pessoas experientes têm maior facilidade para operar sistemas não lineares ou com múltiplas variáveis, baseado no seu conhecimento sobre o comportamento das variáveis de processo envolvidas.

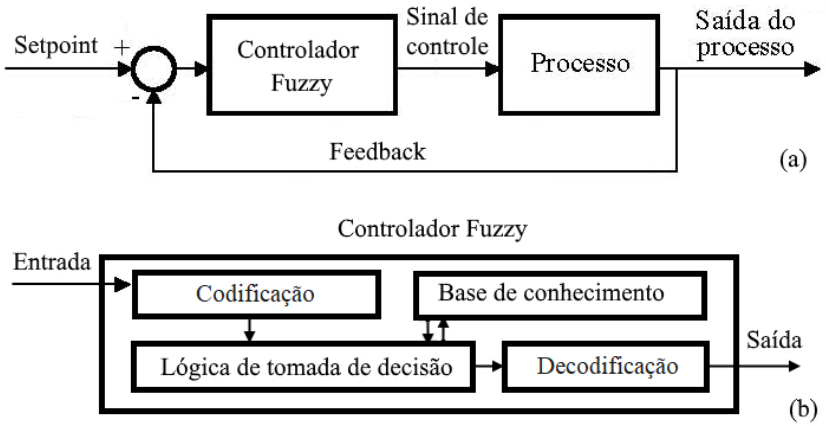


Figura 14 - Sistema de controle em malha fechada (a), etapas do controlador *fuzzy* (b)

Fonte: Autoria própria.

Um sistema de controle *fuzzy* possui a mesma estrutura de qualquer outro sistema de controle em malha fechada, conforme ilustrado na Figura 14 (a). O controlador é composto por 4 módulos, ilustrados na figura 14 (b): Interface de codificação (*fuzzificação*), Lógica de tomada de decisão, Base de conhecimento e Interface de decodificação (*defuzzificação*) (SIMÕES & SHAW, 2007).

- Interface de codificação: Módulo de entrada do controlador, responsável pela codificação de valores exatos das variáveis do sistema em conjuntos *fuzzy*, através de funções de pertinência. As funções de pertinência são definidas pelo projetista para cada variável de acordo com o conhecimento do seu comportamento. Cada função está associada a uma variável linguística, que representa um estado *fuzzy* da variável medida, (Baixo, Médio e Alto são exemplos). Qualquer função com imagem compreendida no intervalo [0,1] pode ser utilizada como função de pertinência. Cabe ao projetista escolher de acordo com a natureza da variável a ser codificada, entretanto existem três tipos principais de funções de pertinência que são comumente utilizadas por representar variáveis de natureza linear satisfatoriamente bem com baixo custo computacional: Trapezoidal aberta à esquerda, Triangular e Trapezoidal aberta à direita, conforme exemplo ilustrado na Figura 16.

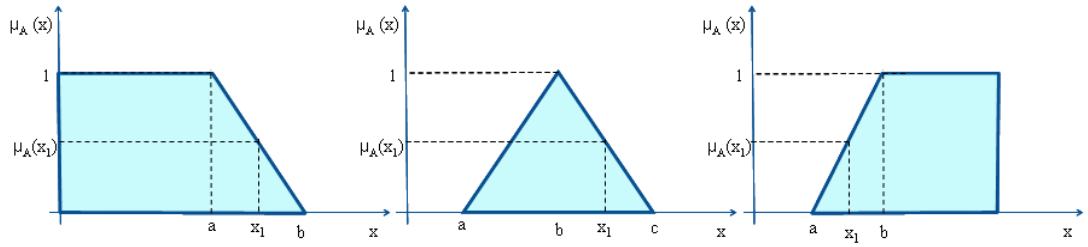


Figura 16 - Exemplos de funções de pertinência

Fonte: COSTA, 2012.

- Base de conhecimento: Conjunto de funções de pertinência e regras *fuzzy* linguísticas que representam o conhecimento integrado ao controlador. As funções de pertinência fornecem as definições numéricas necessárias usadas no conjunto de regras *fuzzy*. Essas, por sua vez, caracterizam a estratégia de controle utilizada de acordo com os valores oriundos da codificação, que são processados na etapa de tomada de decisão.

- Lógica de tomada de decisões: Utiliza implicações *fuzzy* para processar os valores oriundos da codificação para simular a tomada de decisões humana, e gerar ações de controle baseadas nessas decisões. As regras de tomada de decisão são geralmente do tipo "SE condição ENTÃO decisão". A condição pode combinar múltiplas variáveis linguísticas através dos operadores lógicos E e OU, representados matematicamente pelas funções *min* (mínimo valor pertinência entre as variáveis *fuzzy*, implementa a intersecção entre as funções), e max (máximo, implementa a união entre as funções) respectivamente.

- Interface de decodificação: Consiste na consolidação de um valor utilizável no mundo real, a partir de diferentes valores *fuzzy* resultantes da tomada de decisões. O objetivo é obter-se um único valor numérico discreto que melhor represente os valores *fuzzy* da variável linguística de saída, a partir da consolidação do valor *fuzzy* resultante de cada regra associada à essa variável. Os métodos mais utilizados são centro da área, centro dos máximos e média dos máximos (SIMÕES & SHAW, 2007).

Neste trabalho, outros técnicas de controle poderiam ser utilizadas, uma vez que sua a proposta não é criar um modelo de controlador, mas sim de desenvolver recursos de suporte e

condicionamento para sua implementação em uma plataforma de *hardware* livre, e seu uso em contexto industrial. Entretanto o controlador *fuzzy* utilizado, além de atualmente validado e sendo utilizado na empresa onde serão realizados os testes deste trabalho, é visto como uma boa oportunidade para explorar o potencial da plataforma de desenvolvimento utilizada de rodar um controle *fuzzy* de porte industrial.

2.6 Estado da arte

Os principais temas associados ao desenvolvimento deste trabalho são: controle de BCS, controle *fuzzy* e controle industrial em Arduino®. Nesta seção serão apresentados os trabalhos disponíveis na literatura, relacionados aos temas envolvidos.

2.6.1 Controle automático de BCS

Os principais documentos publicados sobre controle de BCSs estão elencados em (COSTA, 2012) na seção 3.3, página 57. No mesmo trabalho ele desenvolveu uma arquitetura *fuzzy* para controle não linear de vazão e submergência pelo ajuste da frequência, com regras baseadas em seu conhecimento especialista, capaz de realizar controle da rotação da bomba em cenários com instrumentação minimalista. O sistema possui versão para CLP ainda em desenvolvimento, mas já em fase experimental com mais de 4 anos de operação contínua.

Este trabalho utiliza o mesmo controle proposto por Costa (2012), porém o objetivo principal é explorar uma nova possibilidade: a implementação de um sistema para controle de BCS com uso de *hardware* livre. Isso envolve a criação de servidor de comunicação serial para simulação do controle, desenvolvimento do programa de controle com suporte a comunicação serial com simulador, supervisório e equipamentos, e idealização de arranjo de hardware adequado para condicionamento da operação do controlador em campo.

2.6.2 Controle *fuzzy*

O controle *fuzzy* possui vasta literatura, comumente apresentado como alternativa para controles PID em plantas complexas ou não lineares, além de representar naturalmente uma

forma de aproximar o conhecimento de operadores experientes a um sistema de controle automático, sem a necessidade de conhecimento aprofundado da planta e de técnicas de análise de estabilidade.

Dorjee (2014) explora o uso de CLPs como controlador *fuzzy* de VSDs para controle de esteiras, que também usam motores de indução trifásicos. No trabalho o autor elenca diversas aplicações industriais, que também fazem uso de controladores *fuzzy* e CLP, visando melhor monitoramento dos parâmetros elétricos do motor, economia de energia, redução de perdas e compensação da rotação em função de variações na carga.

Campos (2005), elenca algumas das principais implementações de controle *fuzzy* na indústria de petróleo. Dentre elas se destacam um controlador para a otimização da produção de uma plataforma; e controle de carga dos fornos, balanceamento dos passes dos fornos e otimização da injeção de vapor de uma unidade de destilação.

No tocante à implementação de controladores de bombeio em particular, destaca-se principalmente a proposta de Costa (2012) para controle de BCS, cujo conceito e projeto do controlador será usado na implementação deste trabalho.

2.6.3 Controle com Arduino®

O uso de hardware livre para aplicações industriais ainda é pouco difundido devido a fatores como a fragilidade, limitação dos recursos, e aspectos de segurança relacionados à abertura do *hardware* e do *software*. Na indústria do petróleo, controladores para métodos de elevação em poços *onshore* sempre estarão próximos ao poço, em lugares arejados, expostos apenas à intempéries climáticas e eventuais descargas atmosféricas. O uso de revestimento, arrefecimento e proteções elétricas adequadas tornam seu uso tão seguro quanto o de qualquer outra solução existente até o momento.

Para controle de velocidade de motores de indução em geral, Kumar (2013) apresenta alternativamente uma proposta de controle da rotação através do ajuste de fase da tensão de alimentação do motor utilizando TRIACs controlados por Arduino®.

Também na indústria de petróleo, Scholl (2014) utilizou 3 placas Arduino® Nano, para implementação de um sistema de controle e monitoramento de um veículo autônomo

subaquático: um para monitoramento interno e controle do motor de propulsão, um para aquisição de dados de pressão e direção, e outro para entrada dos comandos de controle do veículo.

2.6.4 Controlador *fuzzy* para BCS com Arduino®

Com relação ao uso de *hardware* livre para esse tipo de aplicação, não foram encontrados outros desenvolvimentos validados na literatura, sendo este trabalho uma solução pioneira, e portanto motivadora de expansão para outros métodos de elevação de petróleo.

Os sistemas de elevação de petróleo são aplicações industriais com características um pouco distintas das aplicações mais populares como controle de produção e plantas de grande porte, complexas e perigosas. Instalado próximo aos poço, o controlador vai estar protegido por todo o aparato de segurança necessário para evitar danos patrimoniais e ao meio ambiente.

No caso de um controlador de BCS, a técnica mais utilizada atualmente se baseia no controle da frequência de saída de um VSD, para aumentar ou reduzir a rotação da bomba e consequentemente a energia transferida para elevar o fluido do poço. Por sua vez o VSD deve ser obrigatoriamente dotado de recursos de proteção contra rotações reversas, e de configurações para proteção contra correntes, tensões e frequências fora das faixas seguras de operação. Isso isenta o controlador de grande parte da responsabilidade, de forma que eventuais falhas no seu sistema apenas manteriam a operação em uma condição de operação no pior caso indesejada, mas nunca fora dos limites seguros de operação.

Além disso, a natureza lenta do processo migração do óleo da formação para o poço, dispensa a necessidade de controladores com características de velocidade muito superiores às disponíveis nos modelos atuais de Arduino. Esses aspectos sustentam a hipótese de que a solução é viável, encorajando o desenvolvimento para proporcionar resultados à literatura.

3 MATERIAIS E MÉTODOS

Uma vez revisados os conceitos de automação na indústria do petróleo, elevação com uso de BCS e VSD, *hardware* livre e Arduino®, comunicação serial, instrumentação e controle *fuzzy*, temos a proposta deste trabalho situada na literatura. Os capítulos 3 e 4 abordarão as características específicas da implementação e dos resultados deste projeto.

Em termos práticos, o produto final deste trabalho é um painel de controle de rotação de uma BCS através de um VSD. O painel se integra aos demais equipamentos que compõem o sistema de elevação de petróleo. O arranjo desses elementos está ilustrado na Figura 17.

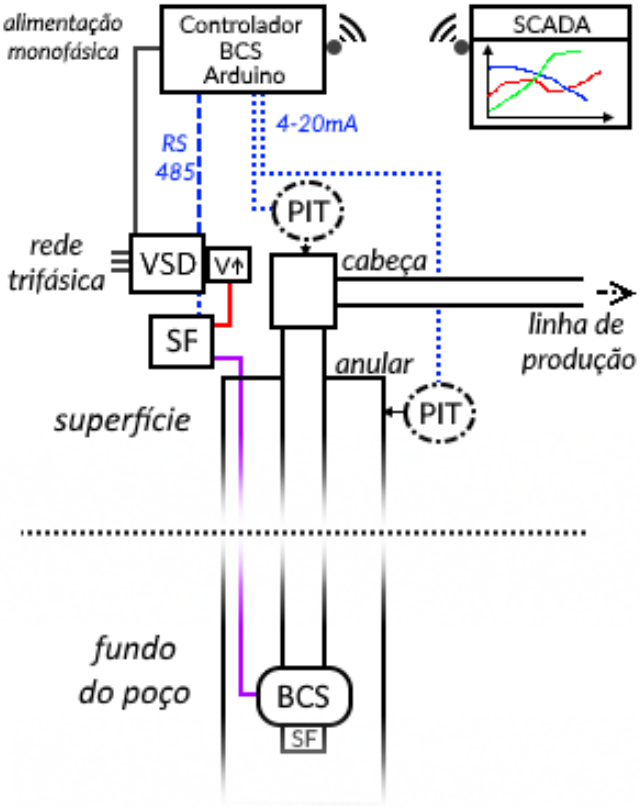


Figura 17 - Infraestrutura do sistema de BCS automatizado

Fonte: Autoria própria.

A composição final deve conter minimamente os seguintes elementos (Costa, 2012):

- VSD: Para permitir o controle da rotação da BCS pela frequência de alimentação, e para aquisição dos parâmetros elétricos de operação;
- DHS: Para aquisição da pressão de fundo do poço e da temperatura do motor.

- Transmissores de pressão interligados à cabeça de produção e ao anular do poço.
- Controlador do VSD: O objeto de estudo.

A metodologia utilizada para o desenvolvimento deste trabalho está detalhada em cinco etapas, enumeradas de 1 a 5 na Figura 18. Além disso, para a futura implantação desse projeto em maior escala, faz-se necessária também a realização da análise de custos de implantação, manutenção e operação. Essa etapa não será compreendida neste trabalho, mas sua necessidade merecem ser levada em consideração.

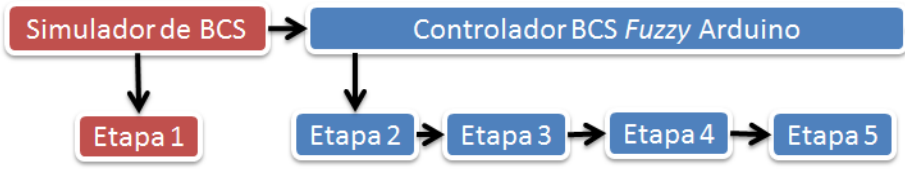


Figura 18 - Etapas de desenvolvimento do projeto

Fonte: Autoria própria.

A primeira etapa consiste no desenvolvimento do módulo de comunicação Modbus do *software* Simulador de BCS de propriedade da PETROBRAS. Essa etapa é necessária para que o controlador desenvolvido neste trabalho possa ser simulado sem *hardware* adicional. O *software* é desenvolvido em C++, e portanto serão utilizadas bibliotecas que implementam o protocolo Modbus RTU escravo na mesma linguagem para comunicação via porta serial.

A segunda etapa consiste no desenvolvimento do programa de controle, e se divide em três fases: Aquisição de dados, controle e segurança. A fase de aquisição de dados consiste na estruturação das memórias, e na implementação das rotinas de consulta ao simulador e aos equipamentos e sensores conectados; A fase de controle contempla a idealização de uma arquitetura fuzzy que implemente o controle proposto por Costa (2012). Nessa etapa também serão definidos os parâmetros de configuração do controlador, necessários para a alteração da condição operacional atual do controle *fuzzy*. A conclusão dessa etapa habilita o controlador para testes com o *software* simulador de BCS; A fase de segurança contempla o dimensionamento e implementação das lógicas de segurança dos equipamentos externos, dispostas de forma a sobrepor o controle parando o equipamento e disparando alarmes se

forem identificadas condições operacionais inseguras baseando-se nas informações dos sensores e equipamentos.

A terceira etapa refere-se às simulações e correções de *bugs* no programa de controle. Eventuais ajustes identificados como necessários para complementar o controle em si surgem nesta etapa. Cenários diferentes de simulação devem ser testados, e as anomalias sanadas até que se obtenha um gráfico de simulação que ilustre a operação adequada do controlador em diferentes cenários.

A quarta etapa consiste na elaboração do painel de controle, para transformar o controlador em um equipamento de campo, utilizando acessórios para condicionar seu uso, que possa ser reproduzido e facilmente instalado e configurado. A etapa também envolve a montagem do equipamento em campo, e a interligação dos equipamentos externos.

Na quinta etapa a robustez e a segurança do controlador serão abordadas através de medições de parâmetros de operação, e podendo acontecer aprimoramentos no *hardware e software* idealizados, visando o melhor condicionamento da operação e das comunicações.

3.1 Simulador de BCS

Para que um controlador em desenvolvimento possa ser simulado e validado com segurança, se faz necessário o uso de um software simulador, modelado de forma a representar satisfatoriamente o fenômeno físico a ser controlado. Simulação Computacional consiste em conduzir experimentos em um computador, envolvendo relações de conteúdo lógico e matemático, necessários à descrição do comportamento e da estrutura de um problema real, em períodos de tempo bem definidos (ABREU & RANGEL, 1999).

Como o trabalho em questão propõe um controlador de BCS, se fez necessário buscar na literatura um simulador para esse método de elevação. O simulador selecionado foi o desenvolvido pela equipe do projeto AUTOPOC, uma parceria entre PETROBRAS e a UFRN, cuja interface foi desenvolvida por Barbosa (2008): “Propõe-se o desenvolvimento de uma ferramenta computacional para simulação do conjunto BCS que seja capaz de representar o comportamento dinâmico deste método buscando proporcionar um maior domínio de conhecimento acerca do mesmo.” A Figura 19 ilustra a sua tela principal:

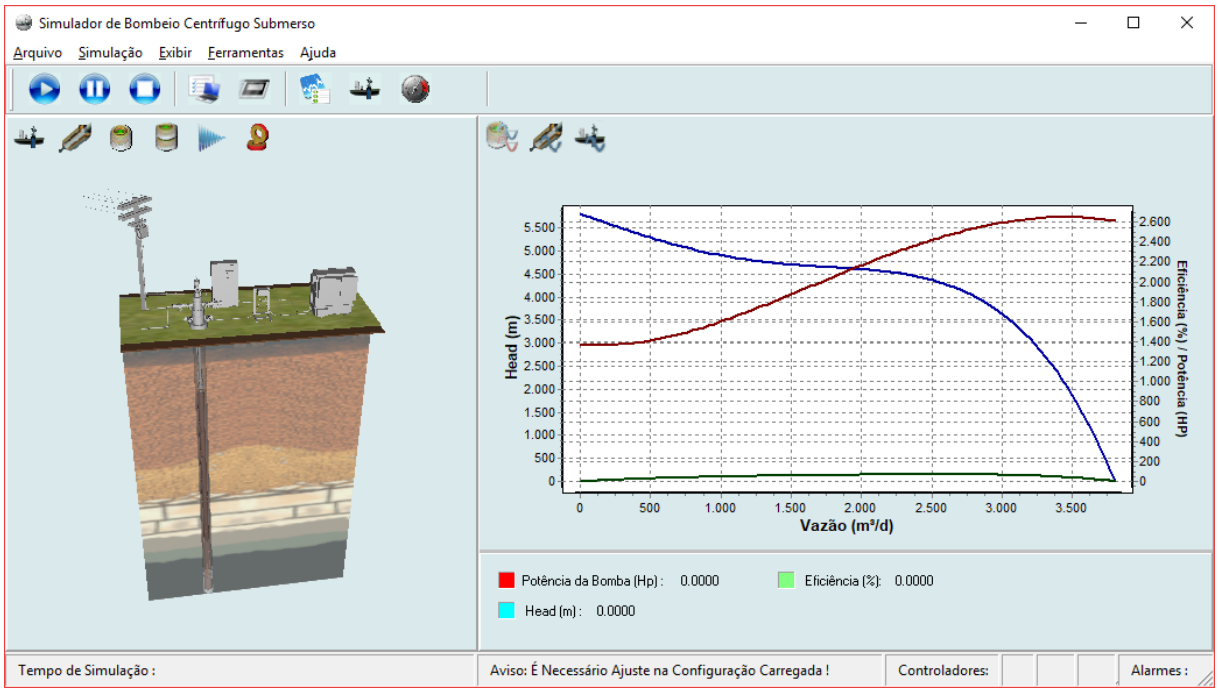


Figura 19 - Tela principal do simulador de BCS

Fonte: Software de simulação de BCS da PETROBRAS.

Os motivos principais que guiaram a escolha, foram o fato de a sua modelagem ser a mesma utilizada na idealização do controlador desenvolvido, além de todas as facilidades de uso e desenvolvimento, uma vez que os direitos autorais são possuídos pela empresa que dará suporte na implementação e testes do controlador em questão. Além disso o simulador é desenvolvido com a interface de desenvolvimento C++ Builder da Borland®, que possui rica e difundida literatura disponível na internet.

Um dos aspectos principais relacionados ao projeto do controlador é o protocolo de comunicação utilizado na comunicação entre os equipamentos. No projeto em questão o protocolo escolhido foi o Modbus, necessariamente por ser o protocolo utilizado pelos equipamentos que compõem o sistema de elevação, mas também por atender aos requisitos de controle, pela ampla compatibilidade com inversores, e pelo baixo custo com desenvolvimento e hardware adicional. Entretanto, se o controlador desenvolvido e o simulador utilizado não dispuserem de suporte ao mesmo protocolo, que é o caso, faz-se necessário que o protocolo de um seja desenvolvido no outro. Como desenvolver suporte à

um novo protocolo em um controlador dedicado apenas encareceria o projeto, optou-se por desenvolver o suporte ao protocolo Modbus no Simulador de BCS.

3.1.1 Servidor Modbus

Até o desenvolvimento deste trabalho, o simulador suportava comunicação através de um servidor implementado sobre a pilha TCP/IP. O servidor permite acesso de leitura a parâmetros dos componentes virtuais simulados (VSD, DHS, e equipamentos de elevação do poço) assim como acesso de escrita da frequência de operação do VSD, que é o parâmetro de saída do controlador.

Como o desenvolvimento de suporte à comunicação TCP/IP no controlador encareceria o projeto sem trazer ganhos consideráveis, optou-se pelo desenvolvido de um servidor Modbus via porta de comunicação serial no software simulador de BCS, para suporte de controladores com comunicação serial Modbus e realização das mesmas operações de leitura e escrita disponíveis no servidor TCP. Quanto ao desenvolvimento do lado do controlador, o mesmo mestre Modbus desenvolvido para comunicação com os equipamentos da aplicação real pode ser utilizado, o que torna o uso desse protocolo ainda mais vantajoso, uma vez que o Modbus é o único protocolo suportado por todos esses equipamentos.

Com isso tornou-se possível para o controlador se comunicar com o simulador utilizando apenas o mesmo hardware que seria necessário para comunicação com os equipamentos de campo, apenas através de um conversor RS232 TTL / USB para conexão da placa Arduino® com o computador em que o simulador está sendo executado.

Uma vez concebida permissão para desenvolvimento do suporte a controladores Modbus no simulador de BCS, foi realizado o levantamento dos recursos necessários para o desenvolvimento:

- Desenvolvimento de interface gráfica para configuração do servidor;
- Capacidade de enumeração de portas de comunicação serial do Windows®, Sistema operacional para o qual o simulador foi desenvolvido;

- Desenvolvimento de servidor de comunicação serial orientado a eventos e de execução paralela, para garantir o mínimo de consumo de processamento, e não sobrecarregar o simulador;
- Implementação do Modbus: Processamento de mensagens e mapa de memória;
- Desenvolvimento de interface gráfica para configuração do controlador durante a simulação.

O diagrama na Figura 20 ilustra quais classes e unidades foram integradas ao simulador de BCS, e como elas se dispõem à aplicação original sob uma visão hierárquica.

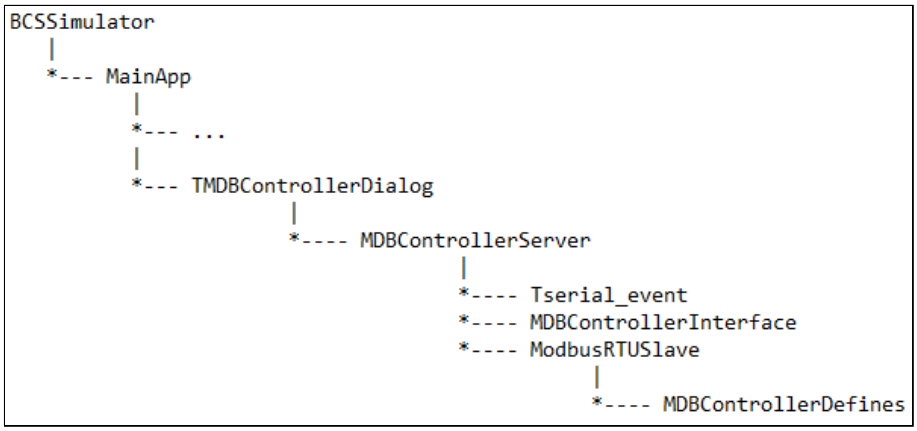


Figura 20 - Classes integradas ao simulador de BCS

Fonte: Autoria própria.

A interface gráfica foi desenvolvida seguindo a mesma filosofia dos outros servidores já existente no simulador, representada pela classe *TMDBControllerDialog* e adicionada ao mesmo *menu*, conforme a Figura 21.

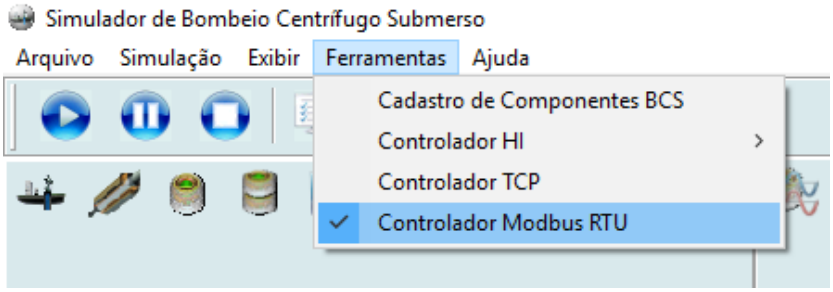


Figura 21 - Menu do controlador Modbus

Fonte: Autoria própria.

A enumeração das portas seriais disponíveis, para população da caixa de seleção da porta conectada ao controlador foi feita utilizando a idéia proposta por Afanasyev (2003), modificada para aquisição do nome amigável de cada dispositivo, ao invés de apenas o número de cada porta. Além da seleção da porta de comunicação, apenas a taxa de comunicação (*baudrate*) e o endereço do nó Modbus foram mantidos configuráveis. O controle de fluxo foi mantido como automático, sem paridade, 8 *bits* de início e 1 de parada. A Figura 22 ilustra a tela de configuração desenvolvida.

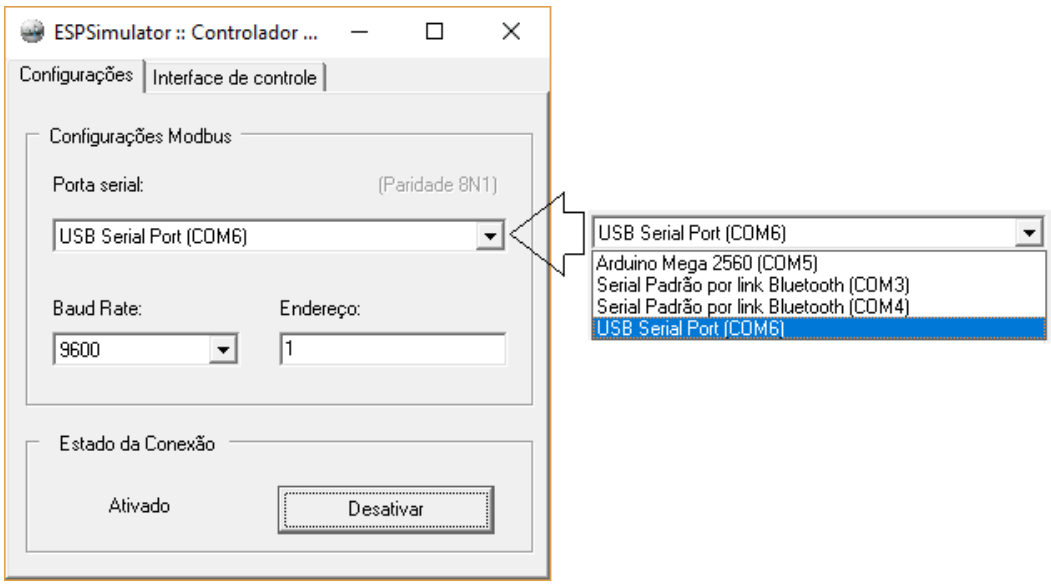


Figura 22 - Configuração do servidor Modbus

Fonte: Autoria própria.

O desenvolvimento do servidor, através da implementação da classe *MDBControllerServer,* consiste na implementação de rotinas que permitam à aplicação detectar a chegada de uma mensagem na porta à qual o servidor esteja conectado, e das rotinas de validação e processamento conforme a aplicação.

Para detecção da chegada de mensagens, durante os testes das primeiras concepções desenvolvidas, observou-se que as rotinas de checagem da chegada de mensagens, somada ao processamento delas, durante a execução da simulação, estavam causando atrasos no tempo da simulação, influenciado negativamente na resposta do controlador. Diversos testes e aprimoramentos foram desenvolvidos, entretanto sob a mesma filosofia não se chegou a uma solução viável. Após pesquisa posterior, a solução para esses problemas foi obtida através da

implementação de um servidor de execução paralela e orientado a evento de chegada de dados na porta serial. Para tal foi utilizada a idéia proposta por Schneider (2001), implementando a classe *Tserial_event* como objeto do servidor Modbus (escravo), que conteria a rotina a ser executada pelos eventos de chegada das mensagens do controlador (mestre) na porta serial.

O Servidor Modbus por sua vez, representado pela classe *ModbusRTUSlave*, foi implementado utilizando as rotinas de processamento de mensagens Modbus desenvolvidas por Romic (2015), customizadas para atender às funções de leitura e escrita dos parâmetros do simulador. Para o mapa de memória, foram definidos 27 registradores (de 10 a 36) para cada um dos parâmetros de simulação que foram estimados como necessários para o controle. Além disso, uma vez que o protocolo Modbus convencionalmente prevê leitura de registradores de 16 *bits* apenas, o uso de valores decimais (com ponto flutuante) é desencorajado, uma vez que a representação computacional convencional desses valores demandaria 32 *bits.* Devido a isso uma boa prática é converter cada valor para uma representação em escala de base 10, de forma a transmitir sempre um inteiro de 16 bits com melhor representação possível para cada parâmetro. Ao chegar no destino, com conhecimento prévio do escalar utilizado para aquele parâmetro, o inteiro recebido é convertido de volta em valor decimal. Os 27 registradores citados anteriormente, e seus respectivos escalares foram definidos na unidade *MDBControllerDefines.* De posse dessas definições foi possível para o servidor Modbus RTU encaminhar leituras e escritas para cada respectivo parâmetro, conforme solicitado em cada mensagem, e com sua respectiva conversão inteiro/decimal.

Por último, para que fosse possível ativar e desativar o controle, ajustar parâmetros de referência e observar a frequência de saída sem que fosse necessário qualquer desenvolvimento adicional no controlador, optou-se por desenvolver no simulador a classe *MDBControllerInterface*, para prover suporte a essas funcionalidades, ilustrada na Figura 23.

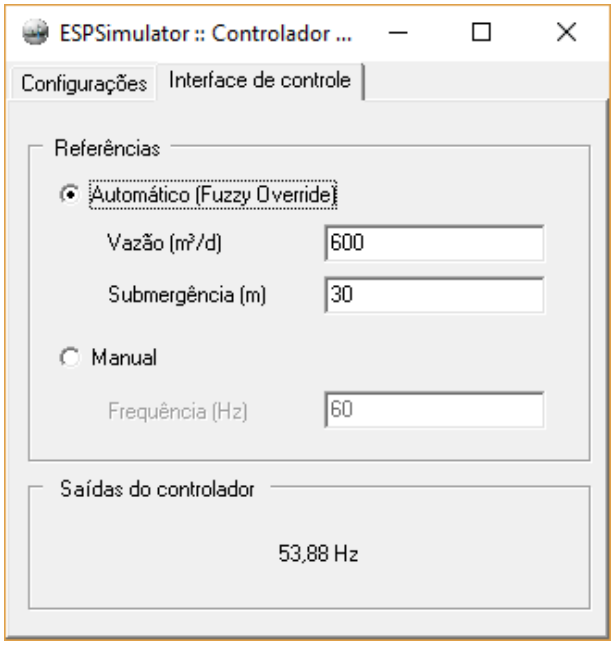


Figura 23 - Interface de controle implementada no simulador

Fonte: Autoria própria.

Além dos recursos acima, algumas adaptações na interface e correção de bugs foram realizadas, favorecendo o projeto do simulador. Uma vez desenvolvidas as funcionalidades necessárias, foi iniciada a segunda etapa, para desenvolvimento do programa de controle para a placa Arduino®.

3.2 Controlador de BCS Arduino®

A seleção do Arduino® para o protótipo foi feita inicialmente pela tabela disponível no site (ARDUINO®, 2017d). Os pré requisitos foram:

- No mínimo duas UARTs, para comunicação interna com os equipamentos, e externa com o sistema SCADA;
- EEPROM disponível (no mínimo 120 bytes), para armazenamento não volátil das configurações;
- Operação preferível com 24V ou em segundo caso com 12V.
- Menor custo possível.

Isso estreitou as opções aos modelos Mega 2560 e MEGA ADK. O último é uma versão do primeiro, mas com funcionalidades específicas para o desenvolvimento de aplicações integradas à dispositivos Android. Essa funcionalidade pode ser anotada como um trabalho posterior, uma vez que haveriam utilidades em se ter um Android como IHM (interface homem máquina) para o controlador, como carga e descarga de configurações, e monitoramento em tempo real através de gráficos. Por fim o Mega 2560 foi selecionado principalmente por seu menor custo. As suas especificações estão na Figura 24.

Microcontrolador	ATmega2560	Memória Flash	256 KB
Tensão de operação	5 V	SRAM	8 KB
Tensão de entrada (recomendada)	7-12 V	EEPROM	4 KB
Tensão de entrada (limite)	6-20 V	Freq. processador	16 MHz
Pinos de E/S digital	54	Pino do LED integrado	13
Pinos de entrada analógica	16	Comprimento	10,15 cm
Corrente DC por pino de E/S	20 mA	Largura	5,33 cm
Corrente DC do pino 3.3 V	50 mA	Peso	37 g

Figura 24 - Especificações do Mega 2560

Fonte: Adaptado de ARDUINO®, 2017e.

O Mega 2560 possui EEPROM de 4kB, bem mais do que o suficiente para a aplicação. Também possui 4 UARTs, uma conectada à USB, que como boa prática é reservada para programação, depuração ou configuração. As outras 3 estão disponíveis em pares de portas digitais para Tx (transmissão) e Rx (recepção) , e níveis TTL, e portanto os dois conversores TTL/232 citados anteriormente foram utilizados para adequar os níveis de tensão das portas aos outros equipamentos. Para interconexão com os outros equipamentos, por não possuírem pinagem RS232 DTE convencional, que consiste em um conector DB9 macho com o Tx no pino 2 e o Rx no pino 3, cabos específicos foram confeccionados, conforme a Figura 25.

Figura 25 - Cabos para comunicação com rádio e conversor 232/485

Fonte: Autoria própria.

O fato de as portas de comunicação serial serem a única interface que o Mega 2560 fará com o restante do sistema, excluindo a fonte que é obrigatória, é um aspecto favorável à segurança e robustez da placa Arduino®, já que o uso de outras interfaces, como por exemplo portas digitais e analógicas, aumentariam a sua exposição a sinais externos indesejados. Além disso, ambas as portas seriais passam pelos conversores TTL/232, e dentre as duas a que sai para o inversor passa antes pelo conversor 232/485 que possui isolação galvânica. Isso anula as chances de dano ao controlador oriundo de sinais indesejados dos outros equipamentos.

3.2.1 Aquisição de variáveis

As equações para submergência e vazão foram deduzidas e validadas por Costa (2012), onde ele admitiu algumas condições predominantes para simplificação dos cálculos. As variáveis utilizadas nas equações estão descritas na lista de dados lidos, ou na lista de parâmetros de configuração descritas a seguir. Os dados lidos são solicitadas via rede Modbus. O diagrama da Figura 26 ilustra a rede e os diferentes protocolos utilizados:

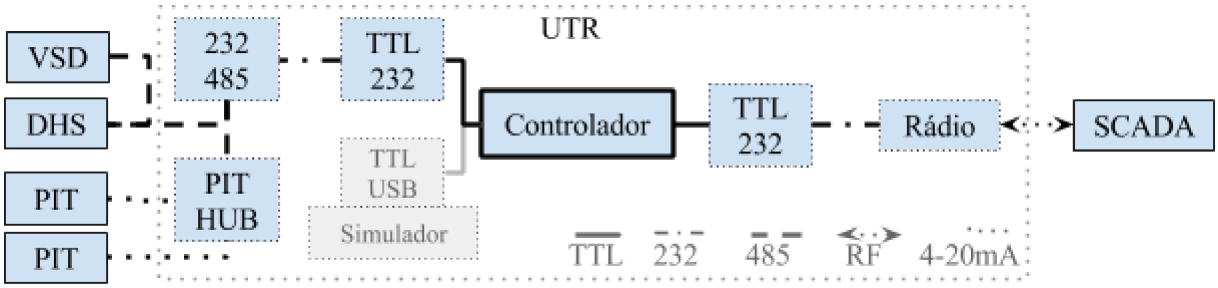


Figura 26 - Diagrama de comunicação do sistema

Fonte - Autoria própria

As solicitações são feitas ciclicamente pela porta do mestre Modbus, conectada à rede interna, representada no diagrama pelos componentes à esquerda do controlador. Os dados lidos são:

- VSD:
  - Frequência de saída do inversor, f (Hz);
  - Corrente de fase na saída do inversor, I_T (A);
  - Tensão trifásica na saída do inversor, V_T (V);
  - Dados do motor:
    - Potência ativa, P (w);
    - Fator de potência, $cos\varphi$ (adimensional);
- DHS:
  - Pressão na admissão da bomba: p_s (kgf/cm²);
  - Temperatura do motor, T_m (K)
  - Temperatura do fluido, T_f (K);
- *Hub* de PITs
  - Pressão de cabeça, que é a pressão na tubulação de produção medida na cabeça do poço (*tubing head*): p_{th} (kgf/cm²);
  - Pressão no anular, que é a pressão no revestimento medida na cabeça do poço (*casing head*): p_{ch} (kgf/cm²);

Para a estimativa da vazão é necessário o uso de parâmetros conhecidos do poço e dos equipamentos. Esses parâmetros de configuração estão listados abaixo:

- Relação de espiras do transformador elevador: β_T (adimensional > 1);
- Diâmetro d_c (m) e comprimento L_c (m) dos cabos do motor, para cálculo da sua resistência R_c (ohm);
- Vazão máxima da bomba obtida na curva correspondente a 60 Hz: q_{max} (m³/d);
- Rendimento da bomba, no ponto de melhor eficiência, η_b^{BEP} (adimensional);
- Rendimento do motor, η_m (adimensional);
- Profundidade da bomba L_s (m);

O cálculo para estimativa da submergência é representado pela Equação 1:

$$s = \frac{p_s - p_{ch}}{\rho_0 g} \tag{1}$$

Sendo ρ_o a massa específica do fluido no anular (kg/m³), e g a aceleração da gravidade, aproximada para 10m/s².

Para a estimativa de vazão uma série de cálculos deve ser realizada. A partir da relação de espiras do transformador elevador obtém-se a tensão nos terminais do secundário do transformador elevador, V_S (V), e a corrente de fase, I_f (A), através das Equações 2 e 3 respectivamente:

$$V_S = V_T \times \beta_T \tag{2}$$

$$I_f = \frac{I_T}{\beta_T} \tag{3}$$

A resistência elétrica dos cabos do motor pode ser adquirida através do conhecimento das suas dimensões, conforme a Equação 4:

$$R_c = \frac{4\sqrt{3}}{\pi} \times \frac{\rho L_c}{d_c^2} \tag{4}$$

onde ρ corresponde à resistividade do cobre (1,72 x 10^{-8} ohm x metro). Com isso é possível obter-se a queda de tensão nos cabos, para então medir a tensão nos terminais do motor V_m (V), através da Equação 5:

$$V_m = V_s - (R_c \times I_f) \tag{5}$$

Com a medida da pressão de cabeça e conhecendo-se a profundidade da bomba, pode-se obter a pressão de descarga da bomba através da Equação 6:

$$p_d = p_{th} + \rho_0 g L_s \tag{6}$$

Uma vez adquiridos todos os parâmetros necessários, a vazão pode ser estimada pela Equação 7 (CAMILLERI et al., 2009):

$$q = q_{max} \frac{f}{60} \left[1 - \frac{q_{max} \Delta p}{\sqrt{3} V_m I_f \cos\varphi \eta_m 4 \eta_b^{BEP}} \left(\frac{f}{60}\right)\right] \tag{7}$$

Onde Δp (Pa) é a diferença entre a pressão de descarga e a pressão de sucção.

De posse dos parâmetros necessários o algoritmo segue para a execução do controle.

3.2.2 Características do controle

O controlador utilizado nesse projeto é do tipo *override* (sobreposição): As saídas de controle de múltiplos controladores, serão comparadas, prevalecendo a que resultar no valor de saída mais seguro, o que nesse caso significa menor frequência. Para exemplificar, considere que a vazão atual está muito abaixo da referência, o que resulta em um acréscimo na frequência de saída, porém a referência de submergência está sendo atingida no mesmo momento, e como um aumento de vazão implicaria numa redução indesejada da submergência, o controle de submergência sobrepõe o de vazão. De forma análoga, se a submergência estiver muito acima da referência, mas a vazão já estiver no limite, o controle de vazão sobrepõe o de submergência.

Controladores tipo *override* demandam cuidados especiais: cada variável controlada pode ter tempos de resposta distintos, demandando tempos de controle mais lentos ou mais agressivos. Tempos muito baixos podem ocasionar escritas em excesso de frequência. Tempos muito altos podem impactar no tempo de resposta e provocar *overshoot* adicional.

Cada conjunto poço, motor e transformador vai possuir limitações que refletem na frequência de operação possível para aquele poço. Devido a isso, no comissionamento do VSD é necessário que sejam ajustados limites mínimo e máximo de frequência, garantindo que nenhuma interface que ajuste a frequência de operação seja capaz de conduzir o sistema a uma condição insegura. Dessa forma é impossível que o controlador possa prejudicar os outros equipamentos do sistema. Esses limites são lidos juntamente com as outras variáveis do inversor, para que o controlador possa alertar condições de controle saturado.

A aplicação não possui criticidade de velocidade, em termos de período de execução e tempo de resposta, uma vez que mesmo um *overshoot* com duração na ordem de segundos, não chega a ser um problema, devido à lentidão na resposta da submergência. Além disso os pontos mais críticos são o bombeio fora da faixa segura de frequência, o esvaziamento do

poço (submergência = 0) e presença de gás na bomba, ambos já previstos e com soluções já integradas ao controlador.

Para evitar que erros de regime fiquem causando oscilação desnecessária no VSD, utilizou-se um parâmetro de variação mínima da saída. Frequências abaixo desse valor são descartadas. Para evitar que o VSD dê grandes saltos de frequência, é importante que o VSD possua ajuste gradual da frequência de saída, até atingir a referência. Caso algum modelo de inversor utilizado não possua a funcionalidade, utilizou-se um passo de frequência. Assim o controlador envia referências de frequência gradativas até que a frequência de saída do controlador seja atingida.

3.3 Programa de controle

Sendo o Arduino® Mega 2560 o mais simples dentre os modelos da plataforma que contém as funcionalidades necessárias para a aplicação em questão, a preocupação para o desenvolvimento do programa seria elaborar uma arquitetura cujo *sketch* coubesse em seus 256 kB, e durante a execução não excedesse os 8kB de RAM. Para estimativa do tamanho necessário, foram implementadas e instanciadas as principais estruturas de dados das principais funcionalidades. São estas:

- Inicialização da estrutura *fuzzy*: Responsável pelo consumo da maior parte da memória. Compreende a instanciação das funções de pertinência para os conjuntos de entradas e de saídas, e a interligação de ambos através das regras *fuzzy*. Inicialmente a implementação do controlador *fuzzy* em sua estrutura original provou-se inviável, consumindo toda a memória RAM disponível. Porém após novas pesquisas 3 novos aspectos foram decisivos para que fosse desenvolvida uma nova versão do programa consumindo apenas 75% da memória disponível. São estes:
    - Utilização da biblioteca eFLL, biblioteca detalhada na seção 3.3.1. Seu uso permitiu redução no consumo de memória e simplificou drasticamente a implementação do controle fuzzy;

    - Reestruturação da arquitetura do controlador, para redução da quantidade de entradas e saídas;
    - Utilização da memória de programa (256kB) para armazenamento das constantes utilizadas. Sem as técnicas utilizadas o controlador não teria sido possível na plataforma escolhida.

- Aquisição e preparação de dados: Consulta ao simulador (durante a simulação) ou aos equipamentos de campo (VSD, DHS e PITs), para leitura das variáveis de controle, composição e normalização das entradas para execução do controlador *fuzzy*.

- Execução do controle: Rotina para execução do controlador, e processamento de cada saída para lógicas de ajuste da frequência final a ser enviada para o VSD.

- Escrita da frequência: Envio da nova frequência para o VSD.

As funcionalidades citadas acima compõem o necessário para o funcionamento do sistema de controle, habilitando-o para uso com o simulador. Para a versão de campo, funcionalidades adicionais são necessárias:

- Carga de configuração: Leitura dos parâmetros de configuração para cada poço. Sem isso seria necessário escrever um *sketch* diferente toda vez que o sistema fosse utilizado em um poço diferente, ou toda vez que fosse alterada alguma característica do poço ou de seus respectivos equipamentos de elevação. Essa funcionalidade também demanda cuidados como geração e comparação de CRC (*Cyclic Redundancy Check*, padrão de verificação de integridade de dados), para que não sejam utilizadas configurações corrompidas, e rotina de salvamento toda vez que um desses parâmetros for alterado remotamente pelo usuário.

- Escravo Modbus para o sistema SCADA: O controlador estará sendo monitorado remotamente em tempo integral, e portanto precisa implementar as funcionalidades de escravo Modbus para leitura de variáveis e escrita de configurações remotamente.

- Diagnóstico e segurança: Como o sistema SCADA possui uma taxa de amostragem limitada, uma vez que o mesmo monitora outras centenas de poços, o controlador deve possuir funcionalidades de monitoramento para detecção e registro de falhas.

Com relação ao uso de boas práticas relacionadas à redução do consumo de memória durante a execução do programa, o principal recurso utilizado foi o salvamento de constantes na memória de programa (ARDUINO®, 2017f). A utilização da palavra chave PROGMEM para constantes estáticas, conforme as primeiras declarações na Figura 27, orienta o compilador a armazenar essas constantes na memória do programa, não ocupando a memória RAM, que é mais escassa, durante a execução do programa.

```
#ifdef PROGMEM_RULES
//para armazenamento das regras na PROGMEM
  const static byte fRulesV [81] PROGMEM = {P,P,N,P,Z,N,P,N,N,P,Z,N,P,Z,N,P,
  const static byte fRulesS [81] PROGMEM = {N,N,Z,N,Z,P,Z,P,P,N,N,Z,N,Z,Z,N,
#else
```

Figura 27 - Armazenamento de constantes na memória de programa

Fonte: Autoria própria.

Os rótulos N, Z e P, representam 0, 1 e 2 respectivamente. Esses valores são os índices para acesso às três funções de pertinência do incremento de frequência.

A Figura 28 ilustra a recuperação das regras para as duas formas citadas acima, respectivamente, através do uso da função *pgm_read_byte_near*:

```
#ifdef  PROGMEM_RULES
        //para recuperação das regras da PROGMEM
        frc_freq->addOutput(freqSetsV[pgm_read_byte_near(fRulesV + r)]);
        frc_freq->addOutput(freqSetsS[pgm_read_byte_near(fRulesS + r)]);
#else
```

Figura 28 - Recuperação de constantes da memória de programa

Fonte: Autoria própria.

Como medida antecipativa, para caso o consumo de RAM final excedesse a capacidade do Mega 2560, foi adquirido o *shield* de expansão QuadRAM®, ilustrado na Figura 29. Apesar de seu uso não chegar a ser necessário, as pesquisas realizadas e merecem ser registradas.

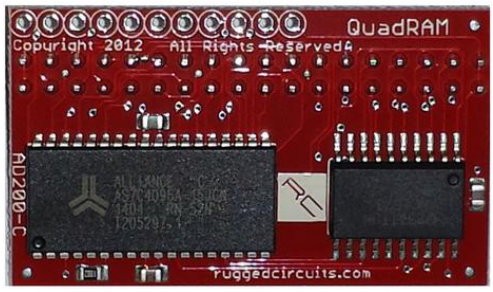


Figura 29 - QuadRAM® *Shield*

Fonte: RUGGED, 2017c.

O QuadRAM® expande a capacidade de memória do Mega com 8 bancos de 64 kB de RAM cada, requisitando apenas o uso da biblioteca *xmem*, que implementa as funcionalidades de acesso.

3.3.1 Biblioteca eFLL

Uma das vantagens do uso de hardware livre e é a abundância de software gratuito que pode ser utilizado para dar agilidade ao projeto. Neste projeto o uso da biblioteca eFLL (*Embedded Fuzzy Logic Library*), biblioteca *fuzzy* para sistemas embarcados (ALVES et al, 2012) foi de importância crucial para a agilidade e redução do programa desenvolvido.

A biblioteca foi desenvolvida pelo *Robotic Research Group* (RRG) na Universidade Estadual do Piauí (UESPI-Teresina). Seu uso simplifica a implementação de um controlador *fuzzy* enquanto mantém um código intuitivo e de fácil leitura, sem uso desnecessário de memória. Suas principais estruturas são:

- *Fuzzy*: Classe principal. Engloba todo o sistema *fuzzy*. Através dele, é possível manipular os conjuntos *fuzzy*, as regras linguísticas, entradas e saídas.
- *FuzzySet*: Permite modelar cada conjunto *fuzzy* do sistema. Atualmente a biblioteca suporta as funções de pertinência triangular, trapezoidal e singleton
- *FuzzyInput*: Agrupa os conjuntos *fuzzy* de entrada que pertencem ao mesmo domínio.
- *FuzzyOutput*: Equivalente ao *FuzzyInput*, porém para as variáveis de saída.

- *FuzzyRule*: Engloba base de regras do objeto *Fuzzy*, que contém um ou mais deste.
- *FuzzyRuleAntecedent*: Usado para compor o objeto *FuzzyRule*. Responsável por montar a parte condicional (SE) das regras.
- *FuzzyRuleConsequent*: Usado para compor o objeto *FuzzyRule*. Responsável por montar a parte consequencial (ENTÃO) das regras.

O uso da biblioteca se resume a 5 procedimentos principais: Criação e associação dos conjuntos fuzzy às entradas e saídas, criação e associação das regras às entradas e saídas, execução da codificação a partir dos dados de entrada, obtenção da saída a partir da decodificação.

3.3.2 Estrutura do programa

Visando maior modularização e facilidade na expansão para outros métodos de elevação ou outras técnicas de controle, optou-se pelo desenvolvimento do *sketch* em múltiplos arquivos, sendo alguns deles de compilação condicional, de acordo com a aplicação e o hardware utilizados.

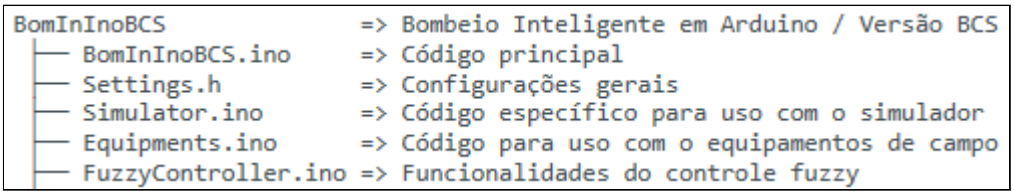


Figura 30 - Estrutura do *sketch*

Fonte: Autoria própria.

Os arquivos utilizados na versão atual estão ilustrados na Figura 30 e suas funções estão descritas a seguir:

- *BomInInoBCS.ino*: Código principal. O *setup* inicializa as configurações, as comunicações e a construção do objeto *fuzzy*. O *loop* executa a leitura das variáveis, o controle e a escrita da saída final no simulador ou VSD. Nesse módulo o uso de simulador ou equipamentos de campo é totalmente transparente.

- *Settings.h*: Definições gerais; Valores de configuração inicial, endereçamento das variáveis, passo de frequência e períodos de controle.

- *FuzzyController.ino*: Funcionalidades específicas do controlador *fuzzy*, como criação das entradas, saídas, associação das regras, codificação e decodificação. Nesse módulo o uso de simulador ou equipamentos de campo é totalmente transparente.

- *Simulator.ino*: Funcionalidades específicas do simulador; Definições de endereçamento e escalares dos registradores do escravo Modbus no simulador; rotinas consulta, processamento e escrita. Código compilado apenas se a chave que habilita o simulador estiver ativada.

- *Equipments.ino*: Funcionalidades específicas dos equipamentos periféricos; Definições de endereçamento e escalares dos registradores de cada um dos 3 escravos Modbus: VSD, DHS e *hub* dos PITs; rotinas consulta, processamento e escrita. Implementação do escravo para o sistema SCADA. Código compilado apenas se a chave que habilita o simulador estiver desativada.

3.3.3 Arquitetura *fuzzy*

Nesta seção serão descritas as principais características do controlador *fuzzy* proposto por Costa (2012), utilizado para desenvolvimento do protótipo.

Com relação ao proposto por Costa (2012), até o momento houveram apenas duas modificações relevantes: A normalização dos erros de vazão e submergência e a estrutura do controlador.

A normalização do erro de vazão foi feita em função de um número arbitrário que representasse significativamente bem um limite máximo de vazão para esse tipo de aplicação. O valor de 1200 m³/d foi adotado mediante conhecimento especialista de profissionais da área (COSTA, 2012). A Equação 8 descreve o cálculo do erro de vazão normalizado:

$$e_q = \frac{q_i - q_{sp}}{q_{max}} \tag{8}$$

Onde (e_q) corresponde ao erro de vazão; (q_i) à vazão instantânea; (q_{sp}) à referência para vazão desejada e (q_{max}) vazão máxima adotada, todos em m³/d.

A normalização do erro de submergência foi feita em função da profundidade da bomba, que é o limite físico máximo possível para a submergência. A Equação 9 descreve o cálculo do erro de submergência normalizado:

$$e_s = \frac{s_i - s_{sp}}{s_{max}} \tag{9}$$

Onde (e_s) corresponde ao erro de submergência; (s_i) à submergência instantânea; (s_{sp}) à referência para vazão desejada e (s_{max}) à profundidade da bomba, todos em metros.

Com relação à estrutura do controlador, a sua forma original está ilustrada na Figura 31, e possui as seguintes características:

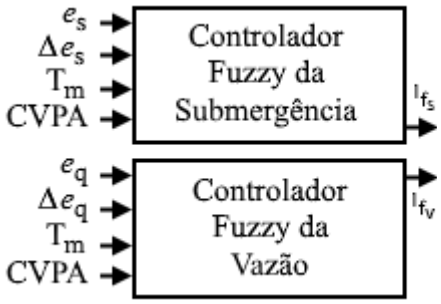


Figura 31 - Estrutura original do controlador *fuzzy*

Fonte: Autoria própria.

- 2 controladores *fuzzy*, para submergência e vazão;
- 4 Entradas em cada:
    - Erro de submergência/vazão;
    - Variação do erro de submergência/vazão;
    - Temperatura do motor;
    - CVPA;
- 1 Saída em cada:
    - Incremento de frequência;
- 2 Bases de regras, 1 para cada saída, com 81 entradas cada (Costa, 2012).

O controlador implementado conforme o esquema acima demonstrou-se inviável devido a consumir quase totalmente a memória RAM disponível no microcontrolador. Após vários experimentos, visando a redução do uso de memória, diversas configurações com consumo de memória gradativamente reduzido foram elaboradas. A configuração mais compacta obtida para o controlador *fuzzy* está descrita na Figura 32.

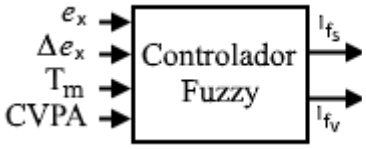


Figura 32 - Estrutura final do controlador *fuzzy*

Fonte: Autoria própria.

- 1 controlador *fuzzy* de 4 entradas apenas. A redução foi possível pois uma vez que as funções de pertinência sugeridas por Costa (2012) eram iguais para as entradas Erro de vazão e de submergência, e da mesma forma para o par de entradas de variação do erro, uma mesma entrada poderia servir para as duas variáveis de controle, desde que fossem utilizadas alternadamente.

- 2 saídas: Incremento de frequência para submergência e para vazão. Semelhante às entradas, a princípio apenas uma saída seria suficiente, uma vez que as funções de pertinência das duas saídas também são iguais. Entretanto isso demandaria uma re-associação da base de regras, uma vez que a mesma difere para as duas saídas. Como esse processo demandaria processamento e memória adicionais consideráveis, optou-se pelo uso de duas saídas, embora com funções de pertinência idênticas, mas com bases de regras distintas previamente associadas na inicialização, o que foi primordial na redução do consumo de memória.

- 1 Base de regras com 162 entradas, 81 para cada saída, baseado no original.

O controlador atua da seguinte forma:

- Iteração 1:
    - Entradas:

        - Erro de vazão;
        - Variação do erro de vazão;
        - Temperatura do motor;
        - CVPA;
    - Saída relevante: Incremento de frequência para vazão (Saída 1).
- Iteração 2:
    - Entradas:
        - Erro de submergência;
        - Variação do erro de submergência;
        - Temperatura do motor;
        - CVPA;
    - Saída relevante: Incremento de frequência para submergência (Saída 2).

Em cada iteração as respectivas entradas são codificadas, e após a decodificação apenas a saída relevante daquela iteração é coletada, e através da lógica de *override* a saída final é selecionada. Com relação à execução, a forma proposta não apresenta desvantagens. O tempo de execução das duas iterações é da ordem de milissegundos, e portanto não prejudica o processo.

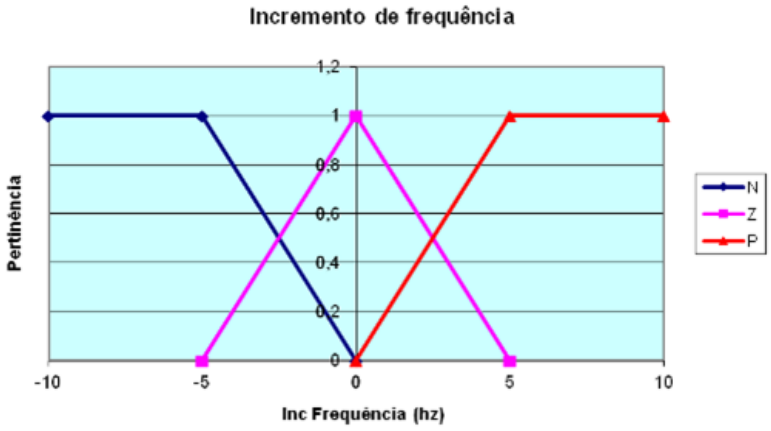


Figura 33 - Fuzzificação do incremento de frequência

Fonte: COSTA, 2012.

As variáveis de entrada de erro, variação de erro e as saídas (incrementos de frequência), tem funções de pertinência do tipo N (negativo), Z (zero) e P (positivo), enquanto

as entradas de temperatura do motor e CVPA são do tipo L (baixo), M (médio) e H (alto). Todas com funções de pertinência seguindo o mesmo padrão do exemplo na Figura 33 (trapezoidal aberta à esquerda, triangular e trapezoidal aberta à direita), mudando apenas os valores de cada função.

A base de conhecimento utilizada possui um total de 162 regras: 81 para controle de submergência, e 81 para controle de vazão. Todas possuem o seguinte formato:

$$SE\ (e_{v,s} = v)\ E\ (\Delta e_{v,s} = w)\ E\ (T_M = x)\ E\ (CVPA = y)\ ENTÃO\ (I_f = z)$$

Onde v,w,x,y,z representam as possíveis variáveis linguísticas para cada entrada e saída: N, Z, P, L, M ou H.

3.4 Infraestrutura de campo

Nesta seção serão abordados os elementos relacionados especificamente ao protótipo de campo, que são o painel de controle e os demais equipamentos do sistema, e as adequações realizadas no sistema SCADA para suporte ao controlador em questão.

3.4.1 Elementos do sistema

Para integração com os equipamentos do poço e utilização no campo, o controlador é integrado a uma caixa com os outros equipamentos necessários para sua alimentação, comunicação e proteção. Esse conjunto de equipamentos é conhecido como painel de controle ou UTR (Unidade Terminal Remota) (COSTA, 2012). Esses equipamentos são essencialmente o que torna este trabalho viável. Um controlador desenvolvido a partir de um protótipo feito com Arduino® pode ser usado no campo ou em outros ambientes industriais desde que possua condicionamento térmico e elétrico adequado, afinal a diferença entre um Arduino® e qualquer outro controlador industrial julgado mais robusto está exatamente nos aparatos de segurança integrados utilizados. Os equipamentos designados para o condicionamento do Mega 2560 neste projeto são:

- Alimentação:
    - Fonte de alimentação de 110VAC para 24VAC, para alimentação dos equipamentos de comunicação da rede interna e também para os PITs;
    - Fonte de alimentação de 110VAC para 12VDC, para alimentação do Mega 2560. Nota: O microcontrolador ATmega2560 opera com 5V, e a tensão de alimentação recomendada é de 7V, conforme ilustrado na Figura 22, pois o consumo de potência do regulador integrado à placa aumenta gradativamente com o aumento da tensão de entrada. Entretanto, além de 12V ainda estar dentro do limite recomendado, fontes de 7V não são comerciais, e já as de 12V são mais populares nos setores de automação industrial, juntamente com as de 24V.

- Comunicação:
    - Rádio serial na faixa de 900MHz, com fonte de alimentação própria de 110V, para comunicação da rede externa. Precisa ser de um modelo comunicável com a rede de rádios do sistema SCADA.
    - Conversor serial RS485/RS232 optoisolado, Para garantia de isolação galvânica entre o Mega 2560 e as conexões que vem do campo.
    - *Hub* de entradas analógicas, para conversão dos PITs para Modbus em RS485. Sua porta serial é conectada à rede do campo, portanto sua proteção não é garantida. Este risco foi assumido para isentá-lo da necessidade de ser optoisolado e garantir o isolamento do controlador na rede interna. Uma vantagem do seu uso é possibilitar que o SCADA monitore todas as variáveis do sistema apenas por conectar o rádio ao conversor RS485/RS232, útil em casos onde seja necessário remover o controlador para manutenção ou desenvolvimento
    - Dois conversores seriais RS232/TTL, com proteção contra ESD de 1.5kV. Ambos conectados ao Mega 2560: Um para comunicação com o rádio serial, e outro para conexão com o conversor RS485/RS232. A Figura 34 ilustra o Mega 2560 após equipado com os conversores, com proteção de acrílico e engate para trilho DIN.

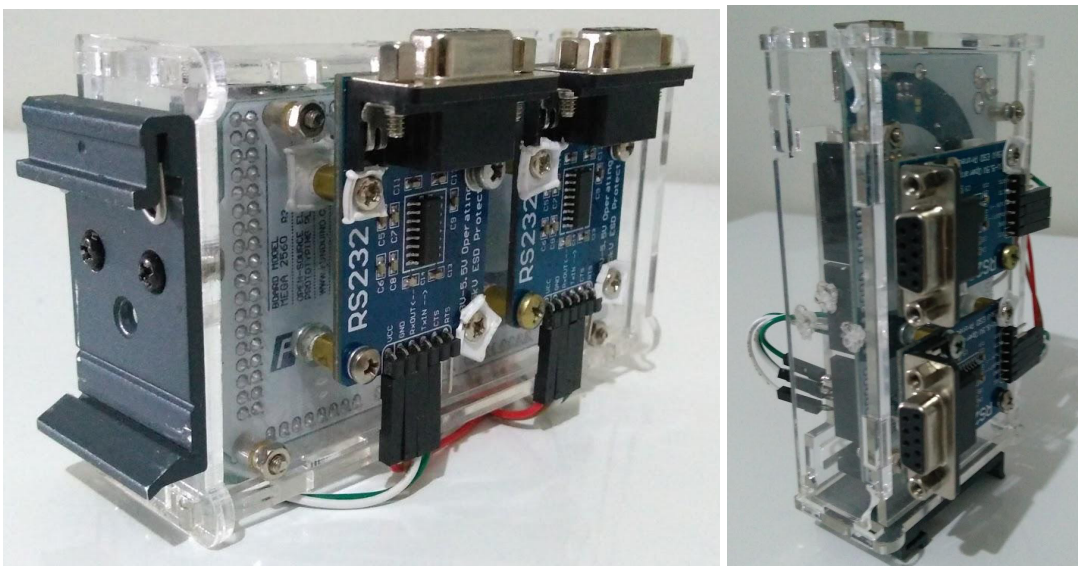


Figura 34 - Mega 2560 equipado para utilização no campo

Fonte: Autoria própria.

- Proteção
    - Disjuntor de 6A: Para proteção contra curto circuitos e descargas elétricas na parte de alimentação CA. O lado CC é protegido pelos fusíveis de 1A e 2A das fontes de 12V e 24V respectivamente.
    - Supressor de surto para 150VAC, conectando uma das fases ao terra ao atuar, e portanto o uso de aterramento é obrigatório.

O painel de controle com a qual o protótipo foi montado foi reaproveitado de uma outro utilizada na empresa, fabricada pela HI Tecnologia (2017). O resultado da integração desses equipamentos está ilustrado na Figura 35. Algumas adaptações foram feitas para atender ao arranjo proposto neste trabalho. Seu uso foi conveniente por possuir tamanho adequado e por trazer alguns dos equipamentos do mesmo fabricante necessários já instalados.

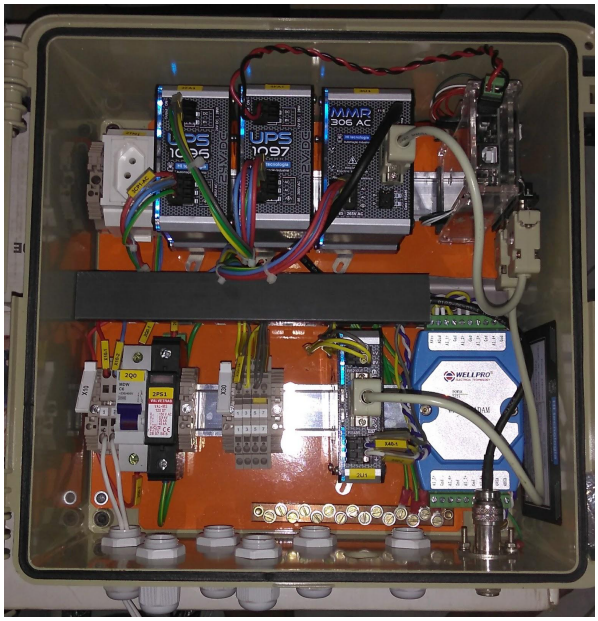


Figura 35 - Painel de controle pronto para instalação no campo

Fonte: Autoria própria.

É importante lembrar que para a implementação do protótipo a utilização de equipamentos já disponíveis é prioritária à aquisição de equipamentos que representem melhor o dimensionamento de potência efetivamente necessário. Em uma etapa posterior do projeto, é recomendado um dimensionamento mais rigoroso dos componentes, para construir uma versão final do controlador.

3.4.2 Supervisão remota

Para supervisão remota durante o período de testes foi utilizado o SISAL: Sistema Supervisório para Automação da Elevação, supervisório de poços corporativo da PETROBRAS, desenvolvido inicialmente por uma equipe de alunos da UFRN, que mais tarde se tornou a empresa RN Tecnologia, e que manteve-se realizando desenvolvimento e manutenção do sistema até o último ano. O sistema possui facilidades de integração de protótipos através de comunicação serial via rádio, com capacidade de historiamento de variáveis disponíveis no controlador, configuração de alarmes e monitoramento 24/7. A Figura 36 ilustra a janela do SISAL para operação de poços equipados com BCS.

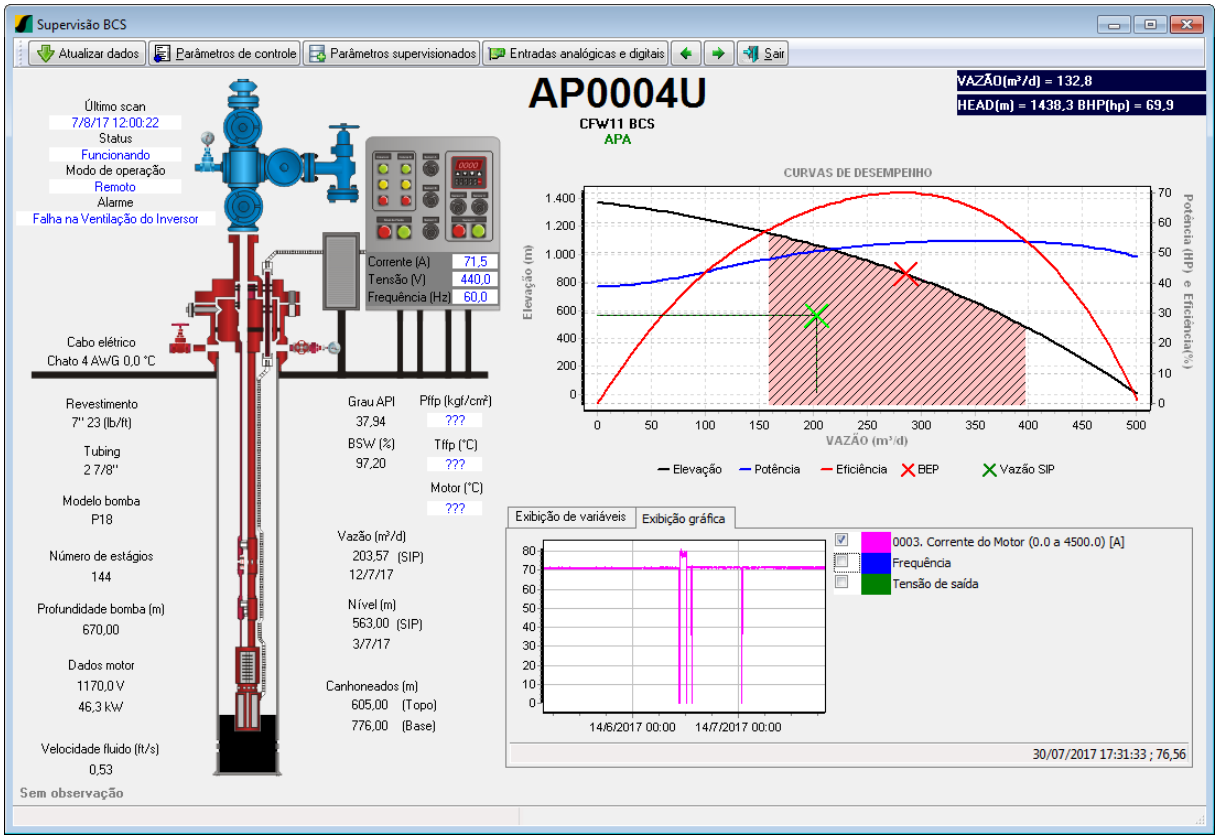


Figura 36 - Tela de operação de BCS

Fonte: Supervisório SISAL.

A integração de um novo controlador ao supervisório é realizada pela equipe de desenvolvimento e acontece em dois passos: Cadastro como controlador genérico, sem método de elevação associado, e posteriormente associação com um método de elevação.

Estando o painel de controle habilitado para comunicação, dotado de rádio serial sincronizado com o respectivo rádio do mestre de campo, configurado com o mesmo endereço previamente cadastrado no supervisório, o equipamento passa a ser monitorado pelo mestre de campo ciclicamente. A partir daí deve-se cadastrar cada variável, ou expressões baseadas nessas variáveis, que se deseje realizar o monitoramento e historiamento. A montagem do painel de controle, sua instalação no poço, interconexão com os outros equipamentos e sua integração remota com o SISAL habilitam o protótipo para ser efetivamente utilizado em um poço real, para a realização de testes, adequações e avaliação dos resultados.

4 ANÁLISE DOS RESULTADOS

A análise dos resultados foi dividida em duas seções. Na primeira serão mostrados os resultados obtidos relacionados ao Mega 2560 e ao simulador, onde ao comportamento do controle fuzzy em si será melhor ilustrado. Em seguida, serão apresentados os resultados adicionais relacionados ao uso em campo, que introduz aspectos relacionados ao painel de controle e ao supervisório.

4.1 Simulações e experimentos

O propósito da simulação foi auxiliar no desenvolvimento do programa de controle, e validar os seguintes fatores:

- Convergência do programa de controle, apesar de não ser o foco deste trabalho, mas o programa precisaria estar operante para ser testado em campo;
- Memória disponível suficiente para instanciar o controlador fuzzy e demais funcionalidades do programa;
- Estabilidade do programa, com relação ao consumo de memória ao longo do tempo.

Com relação ao funcionamento do controlador fuzzy implementado, foram realizadas diversas simulações com poços virtuais de diferentes características, para testar a sintonia do controle proposto. A versão atual do programa de controle está funcionando conforme o esperado, e com as mesmas características propostas por Costa (2012). A Figura 37 ilustra um caso de simulação, enfatizando pontos relevantes com relação a atuação do controlador.

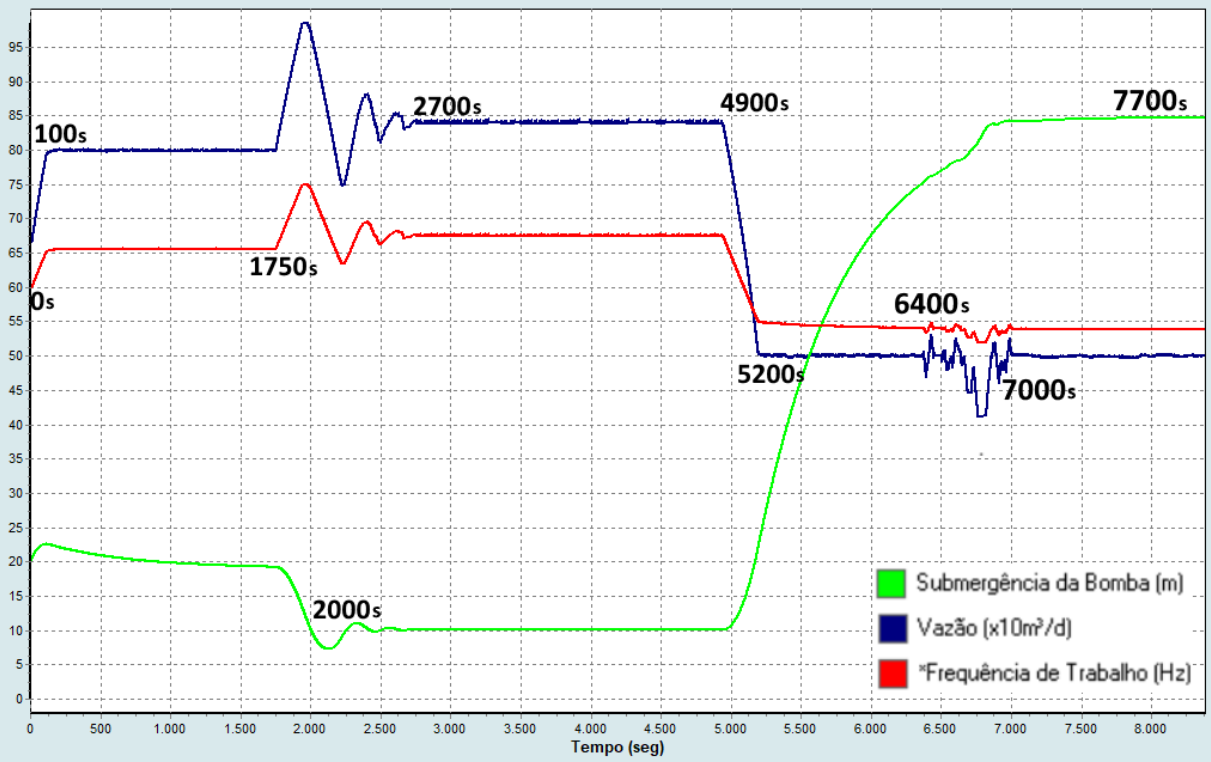


Figura 37 - Simulação do controle

Fonte: Autoria própria.

O gráfico da Figura 37 foi extraído do simulador, e possui duas modificações visuais implementadas para melhorar a exibição de testes de controle: A variável frequência de trabalho, que corresponde ao grupo de variáveis elétricas, foi inserida provisoriamente no gráfico de variáveis do poço. Além disso, a vazão foi reescalada para 10% do seu valor original, de forma a ter as curvas com valores mais próximos e permitir um gráfico menos disperso.

Na simulação apresentada foram destacados 10 momentos significativos:

- T = 0s: Inicia-se o bombeio a 60hz, frequência com a qual se consegue prover uma vazão em torno de 670 m³/d nesse poço, e com 20m de submergência. As referências foram ajustadas para 800m³/d e 10m. O controle segue na direção da referência mais segura.

- T = 100s: A referência de vazão é atingida, e o controlador se mantém, apesar de a submergência estar em torno de 23m. A submergência se mantém em decréscimo.
- T = 1750s: A referência de vazão é aumentada para 1000m³/d, permitindo ao controlador alimentar mais a vazão e reduzir a submergência mais rapidamente.
- T = 2000s: A referência de submergência é atingida. Durante a simulação o período de controle utilizado para o controle de submergência estava de 15 segundos, e como a submergência caiu muito mais rápido foi ocasionado um overshoot de aproximadamente 2,5m. Esse tempo se mostrou inapropriado para poços capazes de operar com vazões dessa ordem, e decidiu-se manter os tempos de controle como parâmetros para serem configurados para cada poço, ao invés de fixos.
- T = 2700s: O controlador conseguiu estabilizar a submergência em torno da referência de 10m. Observa-se um pequeno *ripple* na vazão, uma vez que por questões de resolução o controle continua realizando pequenos ajustes na frequência para manter a submergência, e pequenas variações de frequência são diretamente refletidas na vazão, que é diretamente proporcional.
- T = 4900s: Após 1:21 minutos a referência de vazão foi alterada para 500m³/d. Isso fez com que o controle de vazão passasse a estar em uma situação crítica. Logo o controle *override* comuta para reduzir a vazão, e começa a baixar a frequência, saindo do regime estacionário da submergência.
- T = 5200s: A nova referência de vazão é atingida e mantida. A submergência continua a subir, o que significa que a formação injeta mais fluido no poço do que a vazão de 500m³/d pode tirar.
- T = 6400s: Simulou-se instabilidade na comunicação com o simulador, o que causa um efeito de período de controle aleatório. Isso fez com que o controle passasse a atuar de forma irregular, mas ainda tentando atingir a referência com os dados recebidos ocasionalmente.
- T = 7000s: O ruído simulado foi removido, e o controlador estabilizou-se rapidamente em torno da referência de vazão.
- T = 7700s: A submergência atingiu o nível dinâmico do poço, ponto onde a bomba está tirando a mesma quantidade de fluido que entra no reservatório. Embora a

referência esteja em 10m, o controle *override* permanece ponderando por controlar a vazão, que se encontra em condição mais crítica.

Embora o resultado sirva para validar o algoritmo de controle utilizado neste trabalho, seu ponto principal é mostrar que nenhuma das dificuldades exaltadas na simulação está relacionada a limitações do *hardware*, e que uso adequado dos seus recursos permite que soluções de porte superior ao das soluções atualmente disponíveis na literatura façam uso da tecnologia. A experiência mostra a capacidade de controle do microcontrolador utilizado.

O próximo passo seria desenvolver o restante das funcionalidades do controlador de BCS. Ao término do desenvolvimento das principais funcionalidades o programa ocupou apenas 8% da memória de programa disponível, conforme ilustrado na Figura 37.

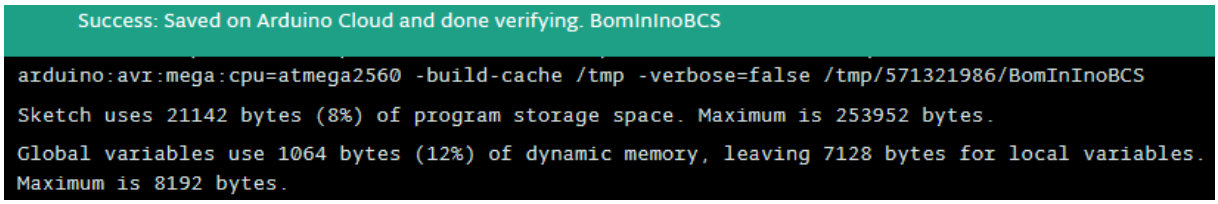


Figura 38 - Consumo da memória de programa do controlador

Fonte: Autoria própria.

A variáveis globais declaradas utilizaram 12% da memória dinâmica, restando 7128. Destes, as estruturas relacionadas ao controlador *fuzzy*, e as demais variáveis utilizadas consumiram cerca de 5,8 kB, deixando aproximadamente 1,2kB livre para a execução, conforme exibido na Figura 39, que ilustra trecho da tela de depuração do programa em dois momentos distintos.

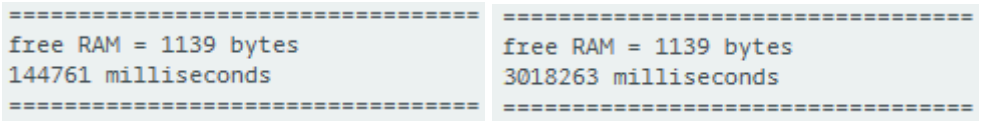


Figura 39 - Memória RAM livre do controlador

Fonte: Autoria própria.

As duas imagens na Figura 39 foram registradas em um intervalo de aproximadamente 47 minutos de diferença. Note que não houve variação na memória RAM disponível, o que indica que não há vazamentos de memória no programa. A versão atual do programa

disponibiliza a memória livre no mapa Modbus do controlador. Isso permite historiar e detectar vazamentos de memória para auxiliar em desenvolvimentos posteriores.

Com relação ao servidor Modbus desenvolvido no simulador, enquanto os testes eram realizados o consumo de CPU e de memória da aplicação foi constantemente monitorado, através da ferramenta *resmon*, o monitor de recursos do *Windows*®, visando avaliar o impacto do escravo Modbus. Quando a simulação está rodando, e há um servidor habilitado, o consumo de memória aumenta consideravelmente, dobrando de cerca de 120kB para 240kB, porém esse número é insignificante, considerando que os computadores atuais possuem memórias da ordem de *gigabytes*. Além disso, foram realizadas simulações de mais de 20 horas contínuas, sem que fosse observada tendência de aumento na memória consumida pelo simulador.

4.2 Testes de campo

Para realização dos testes de utilização no campo o painel de controle foi instalado em um poço real na cidade de Mossoró/RN. A Figura 40 ilustra os equipamentos que compõem o sistema.

(a) (b) (c) (d) (e)

Figura 40 - Componentes de um sistema de BCS automatizado: (a) Cabeça de produção, (b) PITs, (c) Painel do VSD, (d) Painel do DHS e (e) Painel de controle.

Fonte: Autoria própria.

Os principais aspectos a serem observados com os testes são a comunicação da rede interna, a capacidade de controle, a temperatura de operação, e a operação contínua sem travamentos.

A comunicação se mostrou satisfatória durante os 4,8 dias em que foram realizados os testes. A Figura 41 mostra os gráficos de vazão, submergência e frequência de operação monitorados a partir do supervisório.

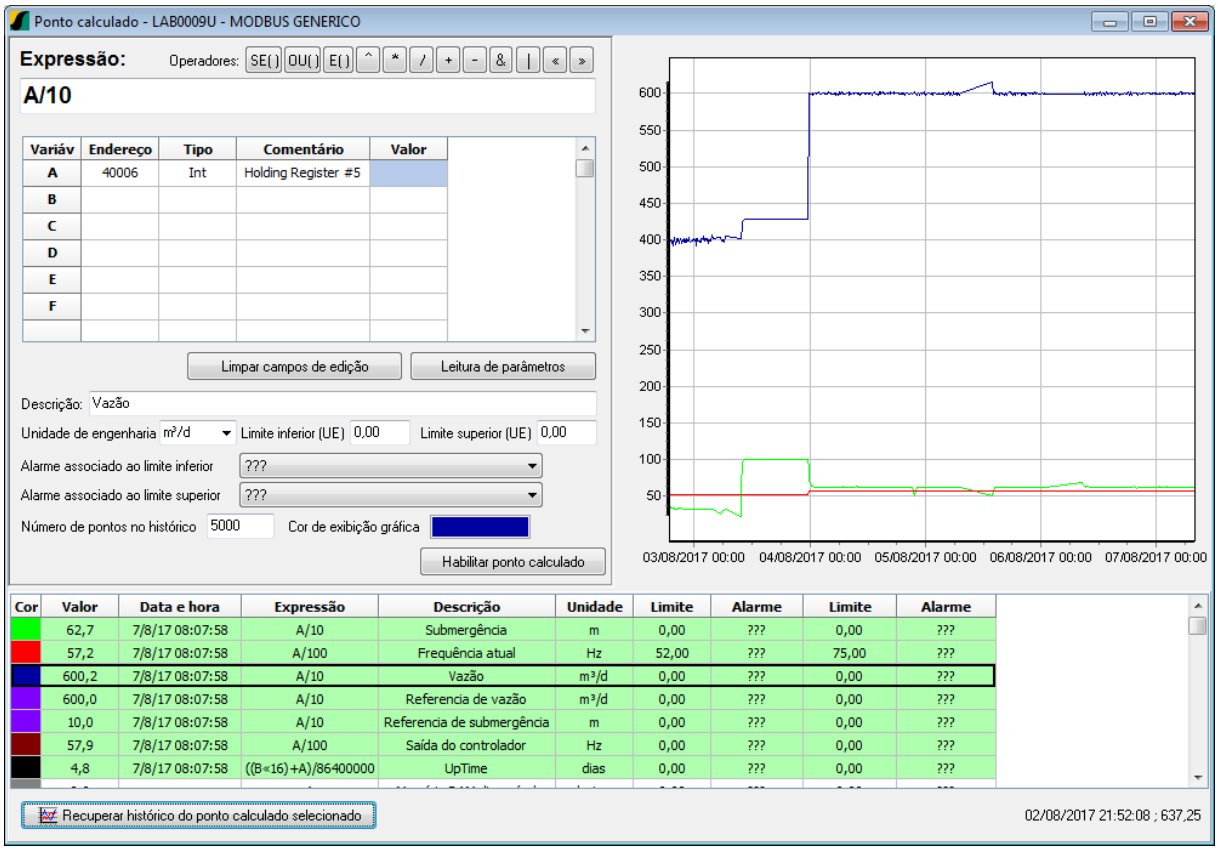


Figura 41 - Monitoramento remoto de variáveis

Fonte: Supervisório SISAL.

Uma vez que o equipamento estava instalado em um poço real, operando em regime estável, era esperado que não houvessem maiores variações nos gráficos. Porém, duas observações foram feitas:

- O degrau entre os dias 3 e 4 se deve a uma modificação operacional realizada no poço;
- Nesse período e no dia 6, a rede ficou indisponível por algumas horas, mas o controlador não foi desligado.

O controlador mostrou operação coerente e ininterrupta enquanto todos os elementos estavam disponíveis.

Outro aspecto relevante a ser validado é a capacidade do microcontrolador manter-se confiável durante o uso no campo, exposto à temperaturas de um ambiente externo, que pode ocasionar em travamentos. Para constatação foi realizado o seguinte procedimento: durante os 4 primeiros dias após a instalação do protótipo no poço, um multímetro foi inserido dentro do painel de controle, em modo de registro de limiares de temperatura, com um termopar fixado ao microcontrolador, conforme ilustrado na Figura 42.

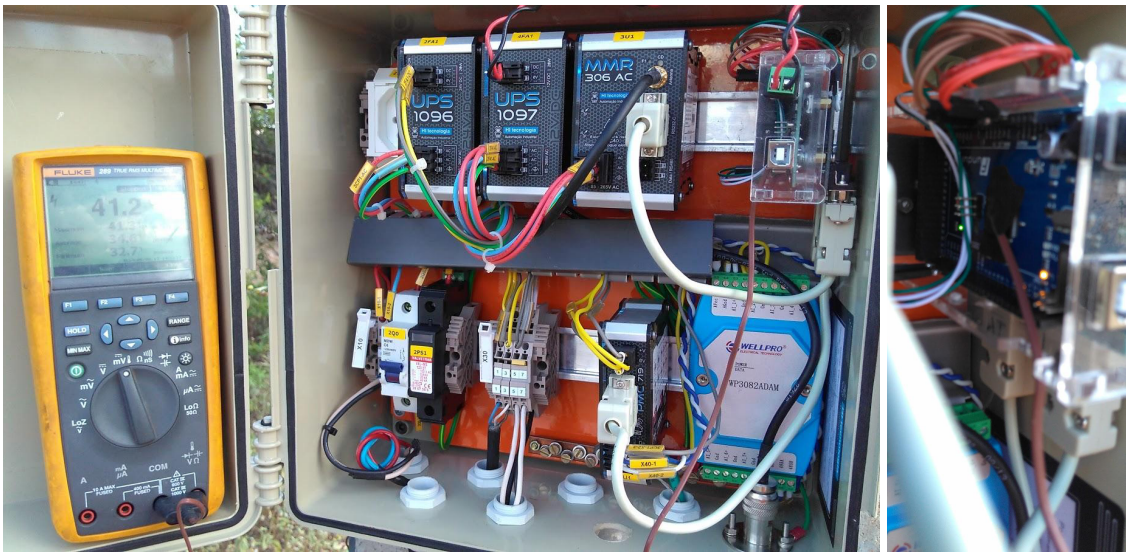


Figura 42 - Monitoramento da temperatura do microcontrolador em campo (esquerda). Termopar fixado em contato com o microcontrolador (direita).

Fonte: Autoria própria.

A inserção do multímetro para medição de temperatura foi feita todos os dias entre as 8 e 10 da manhã, e removido sempre após as 15 horas, visto que os picos de temperatura ocorrem por volta do meio dia, e monitorar além disso seria desperdício da bateria do multímetro. Após os 4 dias a máxima temperatura registrada foi 41 ºC, valor bastante distante do limite de 85 ºC do microcontrolador (ATMEL, 2014). Adicionalmente, o programa em teste possui tempo de espera (*delay*) igual a zero propositalmente, para testar a operação do microcontrolador de forma exaustiva. Após os 4 dias de monitoramento, devido à estabilidade das medições, optou-se pela conclusão dos testes.

A idéia inicial para a realização dos testes de temperatura era utilizar um *shield* para conexão do termopar diretamente ao Mega 2560, e então historiar a temperatura pelo supervisório. Entretanto visto que essa idéia demandaria custo e desenvolvimento adicional, optou-se pelo teste realizado que atingiria o objetivo com menor custo e maior precisão.

Mesmo com todos os aspectos supracitados validados, para cobrir eventuais causas imprevistas de travamento, se fez necessário o acompanhamento contínuo do funcionamento do Mega 2560. Para comprovação de que não houveram travamentos e reinícios durante a operação, foi disponibilizado no mapa Modbus do controlador o tempo acumulado desde a sua última inicialização, obtido através da função *millis*() que retorna esse tempo em milisegundos. Em seguida o parâmetro com esse tempo foi cadastrado para monitoramento no SISAL. Como a função retorna um inteiro de 32 bits, foram necessários dois registradores do mapa Modbus (16bit/registrador) para armazenar o valor, e no SISAL foi necessário recombinar esses dois registradores, e converter o resultado de milissegundos para dias, utilizando a expressão na Figura 43, onde também é exibido o histórico do tempo durante o período de testes.

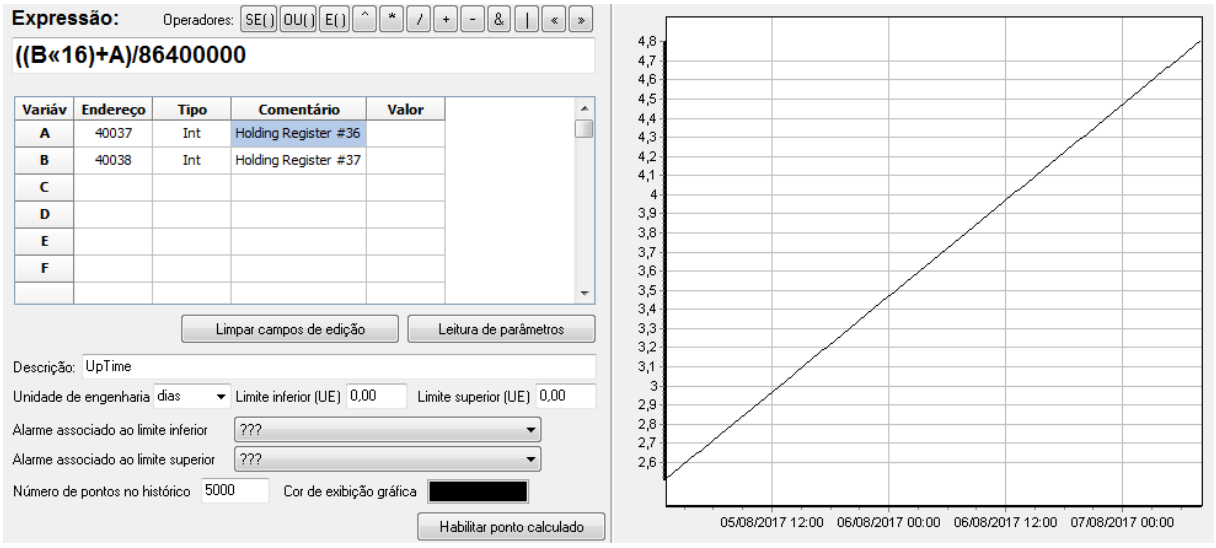


Figura 43 - Monitoramento do tempo de operação do microcontrolador

Fonte: Supervisório SISAL.

Conforme desejado, o valor apenas cresceu continuamente. Qualquer reinício do microcontrolador teria recomeçado a contagem, identificado por uma queda no gráfico.

Quando o teste foi interrompido o tempo registrava 4,8 dias, que equivale à aproximadamente 4 dias e 19 horas. Em posse destes resultados o teste foi considerado como conclusivo.

Para que fique claro: O controlador rodou exaustivamente por esse período, coletando variáveis por uma porta serial, executando o algoritmo de controle, escrevendo a saída de frequência pela mesma serial, e aproximadamente a cada 5 minutos respondendo ao mestre do sistema SCADA com os dados atuais do processo pela segunda porta serial, sem problemas com velocidade, sem reinícios por estouros de memória ou superaquecimento, protegido do sol e da poeira apenas por uma caixa de plástico, sem ventilação adicional. Se for levada em consideração a impressão popular de fragilidade da plataforma de desenvolvimento utilizada, o resultado é surpreendente, e sua divulgação pode servir de motivação para novas experiências do seu uso no contexto industrial.

5 CONSIDERAÇÕES FINAIS

Os testes demonstraram o potencial da solução, embora limitada em alguns aspectos, mas concluindo os objetivos da dissertação com resultados favoráveis ao seu uso. Nesta seção será abordada a conclusão dos objetivos propostos inicialmente.

As funcionalidades adicionais implementadas no simulador de BCS da PETROBRAS atenderam satisfatoriamente ao objetivo inicial, uma vez que o desenvolvimento do servidor Modbus e da interface de controle no simulador exerceram papel determinante na realização do projeto. Isso porque durante a implementação do controlador há a necessidade de testar diferentes valores de entradas e outras condições adversas, às vezes impossíveis de serem reproduzidas no campo, sem prejudicar o processo. Além disso, a ocorrência frequente de reprogramação, reinícios, e outras condições indesejadas esperadas comprometeria o funcionamento de equipamentos reais, tornando o desenvolvimento inicial inviável.

Outro benefício do desenvolvimento é a possibilidade de uso posterior da nova funcionalidade suportada, uma vez que o protocolo Modbus é predominante nos equipamentos de elevação utilizados em poços de campos terrestres.

A maior dificuldade relacionada ao desenvolvimento do programa de controle foi a necessidade de um programa rápido e compacto, devido às limitações de hardware. Neste aspecto as habilidades primordiais foram o conhecimento da linguagem de programação Arduino® e seus tipos de dados, a distinção apropriada entre constantes e variáveis para utilização da memória de programa, e o uso das bibliotecas Modbus e eFLL.

A separação do código fonte em diferentes módulos aconteceu aos poucos, e portanto noções de abstração e modularização foram se tornando necessárias. Programas desenvolvidos para essas plataformas de desenvolvimento costumam ser protótipos de menor porte, e portanto não apresentando necessidade de segmentação do projeto. Um vez estando bem isolados os módulos de comunicação, controle e segurança, suporte a novos métodos e novos equipamentos poderá ser integrado com mais facilidade, dando espaço para o uso de

produção compartilhada, que é uma das principais características do desenvolvimento de *software* livre, mas pouco aplicada no contexto de desenvolvimento para *hardware* livre.

Com relação à plataforma utilizada, o Mega 2560 se mostrou capaz de exercer a função de um controlador fuzzy para poços produtores de petróleo equipados com variador de frequência e outros sensores, com monitoramento remoto, operando de forma ininterrupta durante dias sem apresentar falhas.

Todas as soluções atuais para automação de poços produtores de petróleo, por mais robusta que a sua unidade de processamento principal seja, funcionam munidas de aparato adicional para condicionamento e proteção. O painel desenvolvido segue o padrão industrial utilizado por outras empresas com experiência em aplicações do gênero. Além disso, é importante levar em consideração que existem diversos tipos de ambientes industriais, e cada cenário deve ser analisado individualmente.

Sobre a integração com os sensores e com o VSD, além dos cuidados naturalmente necessários para a comunicação apropriada de cada dispositivo, modelos diferentes de fabricantes diferentes costumam apresentar pequenas divergências nas suas implementações dos protocolos utilizados para o canal de comunicação, o que normalmente demandam cuidados especiais para adequação dos sinais, como opto-isoladores, equalização de fontes e casadores de impedância. Contudo nenhum desses aspectos foi um problema, uma vez que por experiência prévia as devidas proteções contra ruídos e as boas práticas de cabeamento necessárias foram utilizadas.

O uso de *hardware* livre na sua forma elementar pode ser inviável, mas realizando-se o projeto adequado para cada cenário, pode-se ter soluções tão seguras quanto às soluções proprietárias, mas bem mais econômicas. Além disso, a facilidade superior da aquisição de equipamentos já prontos nem sempre é uma vantagem. A falta de suporte técnico com relação aos produtos de automação de poços adquiridos de companhias estrangeiras é um ponto fraco. Uma vez que a empresa mantenedora do conhecimento especialista, nesse caso a PETROBRAS, possa ser a criadora, desenvolvedora e mantenedora dos seus próprios painéis de controle, essa dependência de companhias estrangeiras deixará de ser um problema.

A utilização de *hardware* livre permite às empresas não precisarem pagar pela tecnologia, de forma que o custo de aquisição corresponda apenas à montagem e ao material utilizado. Além disso, é natural que o próprio consumidor seja o maior retentor do conhecimento especialista necessário para controle do processo, uma vez que ele possui experiência em controlá-lo manualmente. Deixar de depender de *hardware* proprietário faz com que o consumidor não seja mais obrigado a fornecer seu conhecimento para um fabricante, que o solicita para customização do seu produto, e posteriormente o vende de volta dentro de um produto fechado, forçando o consumidor a depender de seu suporte técnico, o que implica em novos custos. Sendo as empresas os próprios desenvolvedores, e sabendo como seus controladores funcionam internamente, elas se libertam da dependência do suporte técnico dos fabricantes, que por interesse próprio buscam sempre uma forma de vender mais produtos, ao invés de sanar a necessidade da empresa consumidora de seus produtos.

Dada a capacidade de modularização e facilidade de desenvolvimento das plataformas de *hardware* livre, e considerando a abundância dessa literatura, seu uso pode muitas vezes se tornar mais ágil do que o uso de soluções fechadas, principalmente em aplicações customizadas.

RECOMENDAÇÕES E TRABALHOS FUTUROS

Para que se possa quantificar os benefícios de se investir na iniciativa de *hardware* livre, é primordial a conclusão da análise de custos, que não foi realizada até a conclusão deste trabalho. O que também é necessário para a expansão do uso da solução. Sobre esse aspecto também recomenda-se o desenvolvimento do suporte aos demais modelos de VSDs utilizados na empresa, mantendo-se o padrão de abstração para as funcionalidades das camadas superiores. Recomenda-se ainda o desenvolvimento de um *hardware* com dimensionamento mais rigoroso, e com funcionalidades suficientes para todos os métodos de elevação aplicáveis, para redução de custos, e para que o projeto possa ser reproduzido de forma escalar.

Com relação ao *software* desenvolvido, sugere-se a criação de um *software* modelo, com camadas de abstração de *hardware* para que o programa seja migrado com mais facilidade e agilidade para diferentes arranjos de equipamentos e outras arquiteturas de controladores. Além disso, as seguintes funcionalidades adicionais podem ser estudadas: Carga e descarga de configuração via microSD, wifi, ou *bluetooth*; utilização de RTC (*Real Time Clock*) para registro histórico de falhas e alarmes; e desenvolvimento de aplicativo Android para visualização gráfica dos dados do controlador. Uma vez amadurecido o projeto, sugere-se também a criação de controladores para os métodos de elevação Bombeio Mecânico e Bombeio por Cavidade Progressiva, utilizando a mesma filosofia de desenvolvimento da versão desenvolvida neste trabalho.

Em termos de aumento de produtividade e vida útil dos conjuntos, também é importante que seja feito um acompanhamento, especialmente em poços com produção de gás associado, em contribuição aos testes do controle *fuzzy* utilizado.

REFERÊNCIAS

ABREU, A. M. M.; RANGEL, J. J. A. **Simulação Computacional - Uma Abordagem Introdutória**. Centro Federal de Educação Tecnológica de Campos. 1999.

AFANASYEV, V. **Serial ports. Enumeration and FIFO control**. Code Project. 2003. Disponível em: <http://www.codeproject.com/articles/4187/serial-ports-enumeration-and-fifo-control>. Acessado em 22/07/2017.

ALVES, J. L. L. **Instrumentação, Controle e Automação de Processos**. Rio de Janeiro: Livros Técnicos e Científicos (LTC). 2005.

ALVES, A. J.; LEMOS, M.; KRIDI, D. S. & LEAL, K. **eFLL - Uma Biblioteca Fuzzy para Arduino® e Sistemas Embarcados**. Robotic Research Group (RRG). Universidade Estadual do Piauí (UESPI). 2012. Disponível em: <http://www.zerokol.com/2012/09/arduino fuzzy-uma-biblioteca-fuzzy-para.html>. Acessado em: 25 jul. 2017.

ALVES, T. R.; BURATTO, M.; SOUZA, F. M.; & RODRIGUES, T. V. **OpenPLC: An open source alternative to automation**. In Global Humanitarian Technology Conference (GHTC), IEEE. 2014. p. 585-589.

ARDUINO. **What is Arduino®?** Disponível em: <http://www.arduino.cc/en/Guide/Introduction>. Acessado em: 26 jul. 2017a. **Arduino® Language Reference**. Disponível em: <http://www.arduino.cc/en/Reference/HomePage>. Acessado em: 19 jul. 2017b. **Analog Input Pins**. Disponível em <http://www.arduino.cc/en/Tutorial/AnalogInputPins>. Acessado em 19 jul. 2017c. **Compare board specs**. Disponível em: <http://www.arduino.cc/en/Products/ Compare>. Acessado em 29 jul. 2017d. **Arduino® Mega 2560 Rev3**. Disponível em: <http://store.arduino.cc/usa/arduino-mega-2560-rev3>. Acessado em 29 jul. 2017e. **PROGMEM.** Disponível em: <http://www.arduino.cc/en/Reference/PROGMEM>. Acessado em 05 ago. 2017f.

ASSMANN, B. W. **Estudo de Estratégias de Otimização para Poços de Petróleo com Elevação por Bombeio de Cavidades Progressivas**. Tese de Doutorado. Universidade Federal do Rio Grande do Norte. 2008.

ATMEL. **ATmega2560 Datasheet**. Disponível em: <http://www.atmel.com/pt/br/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf>. Acessado em: 06 ago. 2017.

AXELSON, J. L. **Serial Port Complete: Programming and Circuits for RS-232 and RS-485 Links and Networks**. Lakeview Research. Madison, WI 53704. 1999.

BANZI, M., & SHILOH, M. **Getting Started with Arduino®: The Open Source Electronics Prototyping Platform**. Maker Media, Inc. 2014.

BARBOSA, T. S. **Desenvolvimento da Interface Gráfica para um Simulador Computacional do Sistema de Elevação por Bombeio Centrífugo Submerso**. Trabalho de conclusão de curso (Engenharia da Computação). Universidade Federal do Rio Grande do Norte. Natal, RN. 2008.

BOLTON, W. **Programmable logic controllers**. Newnes. 2015.

CAMPOS, M. C. M. M. **Aplicações de Controladores Fuzzy na Petrobras**. Centro de Pesquisa e Desenvolvimento - CENPES. Rio de Janeiro. 2005.

CAMILLERI, L. A. P., BANCIU, T., DITOIU, G. **First Installation on Five ESPs Offshore Romania - a case study and lessons learned**. Electric Submersible Pump Workshop. Woodlands, Tx: SPE. 2009.

COSTA, R. O. **Controle Aplicado a Poços Com Método de Elevação Bombeio Centrífugo Submerso**. Tese de Doutorado. Natal, UFRN. 2012.

DORJEE, R. G. **PLC and Fuzzy Logic Control of a Variable Frequency Drive**. International Journal of Engineering Trends and Technology (IJETT), Advanced Technical Training Centre, Bardang, Singtam, Sikkim, India. 2014.

EMCU. **RS485**. Disponível em: <http://www.emcu.it/MCUandPeriph/RS485/RS485uk.html>. Acessado em: 08 ago. 2017.

GIBB, A., & ABADIE, S. **Building open source hardware: DIY manufacturing for hackers and makers**. Pearson Education. 2014.

HAAPANEM, B. E., & GAGNER, M. G. **Remote Monitoring and Optimization of Electrical Submersible Pumps Utilizing Control Algorithms.** (Paper 134109). Canadian Unconvencional Resources & International Petroleum Conference. Calgary, Canadá. CSUG/SPE. 2010.

HI TECNOLOGIA. **Site da HI Tecnologia**. Disponível em: <http://www.hitecnologia.com.br>. Acessado em: 08 ago. 2017.

HOWART, M. **HART - Standard for 4-20mA Digital Communications**. Measurement and Control, 27(1), 5-7. 1994.

KUMAR, Y. N., Bindu, P. H., Sneha, A. D., & Sravani, A. **A Novel Implementation of Phase Control Technique for Speed Control of Induction Motor Using ARDUINO**. International Journal of Emerging Technology and Advanced Engineering, v. 3, n. 4, p. 469-473, 2013.

LUGLI, A. B., SANTOS, M. M. D. **Redes Industriais para Automação Industrial: AS-I, PROFIBUS E PROFINET**. 1ed. São Paulo. 2010.

MAITELLI, C. W. **Simulação do Escoamento Monofásico em Um Estágio de Uma Bomba Centrífuga Utilizando Técnicas de Fluidodinâmica Computacional**. Tese de Doutorado. Natal, UFRN. 2010.

MATOS, E., FALCO, R. **Bombas Industriais**. Rio de Janeiro: Interciência. 1998.

MODBUS, I. D. A. **Modbus application protocol specification v1.1b3**. North Grafton, Massachusetts. Disponível em: <http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf>. Acessado em: 24 mai. 2017.

MORAES, C. C., CASTRUCCI, P. L. **Engenharia de Automação Industrial**. 2ed. Rio de Janeiro: Livros Técnicos e Científicos (LTC). 2007.

ROMIC, M. **riModbus RTU - C/C++ Modbus library - Master / Slave.** Disponível em <http://www.easy4engineers.com/rimodbus-rtu-cc-modbus-library-master-slave>. Acessado em 23 mai. 2017.

RUGGED Circuits. **Site da Rugged Circuits.** Disponível em: <http://www.rugged-circuits.com>. Acessado em: 11 jul. 2017a. **The Rugged MEGA**. Disponível em: <http://www.rugged-circuits.com/mega-tech>. Acessado em: 11 jul. 2017b. **QuadRAM**. Disponível em: <http://www.rugged-circuits.com/new-products/quadram>. Acessado: em 11 jul. 2017c. **10 Ways to Destroy an Arduino**. Disponível em: <http://www.ruggedcircuits.com/10-ways-to-destroy-an-arduino>. Acessado em: 11 jul. 2017d.

SCHNEIDER, T. **Serial Communication for WIN32**. Disponível em: <http://www.tetraedre.com/advanced/serial2.php>. Acessado em: 23 jul. 2017.

SCHOLL, M. V.; ROCHA, C. R. **Sistema de Acompanhamento de Missão Para Uma Plataforma Experimental De Robótica Subaquática**. Universidade Federal do Rio Grande (FURG) - Campus Carreiros. Rio Grande (RS). Brasil. 2014.

SERIAL. **Serial Programming Wikibook**. Disponível em: <https://en.wikibooks.org/w/index.php?oldid=3198488>. Acessado em: 07 jul. 2017.

SIMÕES, M. G.; SHAW, I. **Controle e Modelagem *Fuzzy***. São Paulo, Blucher. 2007.

SOUZA, R. B. **Uma Arquitetura para Sistemas Supervisórios Industriais e sua Aplicação em Processos de Elevação Artificial de Petróleo**. Dissertação de Mestrado. Universidade Federal do Rio Grande do Norte. 2005.

SPARKFUN. **RS-232 vs. TTL Serial Communication**. Disponível em: <http://www.sparkfun.com/tutorials/215>. Acessado em: 21 jul. 2017.

TAKÁCS, G. **Electrical Submersible Pumps Manual: design operations and maintenance**. Oxford, UK, Elsevier. 2009.

THOMAS, J. E. **Fundamentos da Engenharia de Petróleo**, 2a edição, Ed. Interciência. 2004.

YAMAGUCHI, M. Y. **Sincronização das Bases de Tempo de CLPs Distribuídos numa Rede de Automação de Processo Industrial**. Dissertação de Mestrado. Escola Politécnica da Universidade de São Paulo. 2006.

ZADEH, L. A. **Fuzzy sets. Information and control**, 8(3), 338-353. University of California, Berkeley, California. 1965.