UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROPOSTA DE PROJETO DE GRADUAÇÃO

LYS MENEGUELLI PEIXOTO

IMPLEMENTAÇÃO DE SISTEMA EMBARCADO DE DIAGNÓSTICO DE CAVITAÇÃO EM VÁLVULA DE CONTROLE

VITÓRIA – ES
DEZEMBRO/2016

LYS MENEGUELLI PEIXOTO

IMPLEMENTAÇÃO DE SISTEMA EMBARCADO DE DIAGNÓSTICO DE CAVITAÇÃO EM VÁLVULA DE CONTROLE

Parte manuscrita da Proposta de Projeto de Graduação do aluno **Nome do Autor**, apresentada ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheira Eletricista.

_____
Título e Nome completo
Orientador

_____
Lys Meneguelli Peixoto
Aluno

VITÓRIA – ES
DEZEMBRO/2016

RESUMO

No cenário empresarial, válvulas de controle são de suma importância para malhas de controle por representar a variação do fluxo de líquidos e sólidos entregues à planta. A manutenção feita nestas válvulas muitas vezes é tardia ou adiantada gerando gastos desnecessários às indústrias, por isso é interessante que os problemas sejam identificados em um tempo hábil para realizar a substituição das válvulas apenas quando necessário. Uma das principais falhas que acometem tais equipamentos é a cavitação, que é o surgimento e explosão de bolhas de vapor causando erosão nos internos da tubulação e da válvula. Neste projeto é desenvolvido um sistema embarcado baseado em cartas de controle que por meio de alterações no desvio padrão da emissão acústica da tubulação diagnostica a ocorrência de cavitação. O sistema foi validado com testes realizados na planta piloto da UFES onde foi corretamente identificada a cavitação em diferentes estágios. No desenvolvimento do projeto surgiram dificuldades que poderiam ser facilmente evitadas com a utilização de um microcontrolador diferente para a embarcação do mesmo.

LISTA DE FIGURAS

Figura 1 - Erosão causada pela cavitação em uma tubulação 6
Figura 2 - Diagrama de pressões ao longo do processo de cavitação 11
Figura 3 - Carta de Shewhart com limites superior e inferior.....12
Figura 4 - Distribuição de densidade normal 14
Figura 5 - Relação entre a densidade de dados da distribuição normal e os limites da carta de controle.....15
Figura 6 - Microfone de eletreto acoplado à tubulação. 16
Figura 7 - Esquemático do circuito de polarização e amplificação do microfone de eletreto 17
Figura 8 - Circuito de amplificação em *protoboard*. 18
Figura 9 - Arduino Uno.....19
Figura 10 - Display de LCD com teclado 20
Figura 11 - *Shield* de leitura e escrita no cartaõ SD.....20
Figura 12 - Diagrama de funcionamento do algoritmo.....22
Figura 13 - Registrador ADMUX 23
Figura 14 - Registrador ADCSRA 24
Figura 15 - Registrador ADCSRB 24
Figura 16 - Parte de código responsável pelas configurações da conversão AD.....25
Figura 17 - Linha de código que permite a escrita em binário no cartão SD.....26
Figura 18 - Diagrama do uso de dois buffers para coleta de dados e escrita no cartão SD 26
Figura 19 - de código responsável pelo cálculo dos limites superior e inferior da carta de controle.....28
Figura 20 - Display LCD: Menu inicial (a) e mensagem pós treinamento (b) 29
Figura 21 - Display LCD: Aviso de falta de treinamento (a) e em analise (b) 30
Figura 22 - Display LCD: Reporte de cavitação.....30
Figura 23 - Planta piloto na UFES 31
Figura 24 - Dados de treinamento (verdes) e dados de diagnóstico sem cavitação (rosa) 32
Figura 25 - Dados de treinamento (verdes) e dados de diagnóstico de cavitação fraca (azuis).....33
Figura 26 - Dados de treinamento (verdes) e dados de diagnóstico de cavitação média (azuis).....33
Figura 27 - Dados de treinamento (verdes) e dados de cavitação forte (azuis).....34

LISTA DE TABELAS

Tabela 1 - Tabela de componentes do circuito de polarização e amplificação do microfone de eletreto.....17

SUMÁRIO

1 INTRODUÇÃO 6
1.1 Apresentação e objeto de pesquisa 6
1.2 Motivação 7
1.3 Objetivo geral e objetivos específicos 8
1.4 Estrutura do trabalho 9
2 REFERENCIAL TEÓRICO 10
2.1 Cavitação em válvulas de controle 10
2.2 Cartas de controle 12
3 METODOLOGIA 16
3.1 Circuito amplificador do microfone de eletreto 16
3.2 Arduino Uno 18
3.3 *Shields* de display LCD e leitura/escrita em cartão SD 19
3.4 Algoritmo desenvolvido 21
3.4.1 Coleta e conversão de dados 22
3.4.2 Escrita e armazenamento no cartão SD 25
3.4.3 Cálculo de UCL e LCL e diagnóstico 27
4 APLICAÇÃO E DISCUSSÃO 29
4.1 Operação do protótipo 29
4.2 Testes realizados 31
5 CONCLUSÕES E TRABALHOS FUTUROS 35
REFERÊNCIAS BIBLIOGRÁFICAS 36
APÊNDICE A - ALGORITMO DESENVOLVIDO 38

1 INTRODUÇÃO

1.1 Apresentação e objeto de pesquisa

Válvulas de controle são importantes elementos da malha de controle por restringir ou permitir o fluxo do fluido a passar por elas. Muitas vezes, em um ambiente industrial, a manutenção das válvulas é feita de maneira cíclica, trocando-se os componentes periodicamente sem que os mesmos apresentem sinais de falha (BO-SUK et al., 2005).

A perda de produção em processos industriais devido à falta de confiabilidade varia de 2% a 16% (SHUKRI; MUN; IBRAHIM, 2011), como elemento chave em sistemas de controle, a monitoração de falhas em válvulas se torna crucial para evitar paradas e gastos excessivos no processo.

Um problema muito comum, e cuja identificação facilita a manutenção, em válvulas de controle é a cavitação. Cavitação é um evento que ocorre quando um líquido gera bolhas de vapor devido a uma redução de pressão durante o seu movimento e eclode gerando fortes vibrações e erosão na parte interna de válvulas e tubulações (PAES; MUNARO; CIARELLI, 2016), como é mostrado na Figura 1.

Figura 1 - Erosão causada pela cavitação em uma tubulação

Fonte: BO-SUK et al., 2005.

Os danos causados pela cavitação usualmente são inspecionados checando-se a tubulação à jusante da válvula, porém isto requisita que o processo fosse parado, gerando gastos e parando a produção. Há, portanto, uma clara necessidade da utilização de um procedimento não invasivo de monitoramento da válvula (ULANICKI; PICINALLI; JANUS, 2015).

Em Paes, Munaro e Ciarelli (2016) foram desenvolvidos dois procedimentos de diagnóstico de cavitação em válvulas de controle, um analisando o potencial da pressão à jusante por meio de *Support Vector Machines* (SVM) e o outro analisando a emissão acústica no entorno da válvula por meio de cartas de controle.

Um sistema embarcado é um sistema microprocessado com uma função específica já pré-definida. Tais sistemas são utilizados para que operações possam ser feitas no local de uso ao invés de necessitar recorrer a uma máquina central fixa para processamento. Em diagnósticos de falhas, um sistema embarcado seria proveitoso por realizar a coleta de dados e análise *in loco* possibilitando ao operador tomar medidas de manutenção em um tempo reduzido.

Este projeto busca programar uma solução embarcada de detecção de cavitação em válvulas baseando-se no projeto desenvolvido em Paes, Munaro e Ciarelli (2016) de análise de emissão acústica por meio de cartas de controle.

1.2 Motivação

Em muitas malhas de controle as válvulas são os últimos elementos a serem acionados controlando a quantidade de líquido a ser disponibilizada ao processo. Ocupando esta posição na malha elas se tornam um elemento crítico para o correto funcionamento do processo, o que faz com que a detecção de falhas nas mesmas seja de suma importância.

Sendo a cavitação um problema comum ocorrente em válvulas, o diagnóstico correto desta falha pode diminuir gastos com manutenções desnecessárias assim como o ruído ao redor da tubulação na qual a válvula se encontra.

Embarcar soluções de diagnóstico e disponibilizar uma interface local para a detecção da cavitação é proveitoso de forma a diminuir o tempo necessário para mudar de plataforma ao interpretar os dados recolhidos, possibilitar a fácil análise de diferentes válvulas e obter um indicador de falha local.

Todos os motivos supracitados justificam o interesse no projeto que este trabalho se propõe a fazer visto que a aplicação prática é clara. Tomando como base estudos anteriores, como Paes, Munaro e Ciarelli (2016) e Ulanicki, Picinalli e Janus (2015), têm-se técnicas que já foram testadas e avaliadas neste âmbito.

1.3 Objetivo geral e objetivos específicos

O objetivo geral deste projeto é elaborar um algoritmo de análise de emissão sonora por meio de cartas de controle e embarcá-lo em um Arduino para o diagnóstico de cavitação em uma válvula de controle da planta piloto localizada na Universidade Federal do Espírito Santo (UFES).

Os objetivos específicos deste trabalho são:

- Estudar sobre o problema de cavitação em válvulas e as formas de identificá-la;
- Estudar sobre a implementação e uso de cartas de controle em controle estatístico de qualidade;
- Utilizar a técnica citada no item anterior para desenvolver um algoritmo capaz de diagnosticar a ocorrência de cavitação;
- Desenvolver um menu para um display de LCD que permita a escolha entre treinamento do modelo ou diagnóstico do problema;
- Utilizar um cartão SD como armazenamento dos dados coletados nos períodos de treinamento e diagnóstico;
- Integrar o sistema de menu, armazenamento e carta de controle em um Arduino Uno;
- Realizar testes na planta piloto para confirmação do método.

1.4 Estrutura do trabalho

Este trabalho está organizado em cinco capítulos onde o primeiro descreve brevemente o problema a ser resolvido, os motivos pelos quais o projeto será realizado e os objetivos a serem alcançados com o mesmo. O segundo capítulo consta de um referencial teórico sobre os temas abordados pelo trabalho como o conceito de cavitação e cartas de controle. No Capítulo 3 é apresentada a metodologia utilizada no projeto, a descrição dos equipamentos necessários e o desenvolvimento do algoritmo.

O quarto capítulo mostra a aplicação do trabalho, explicitando as configurações necessárias para a operação e discutindo a montagem e os testes. As conclusões e observações finais encontram-se no Capítulo 5 e o código do algoritmo desenvolvido é disponibilizado no apêndice.

2 REFERENCIAL TEÓRICO

2.1 Cavitação em válvulas de controle

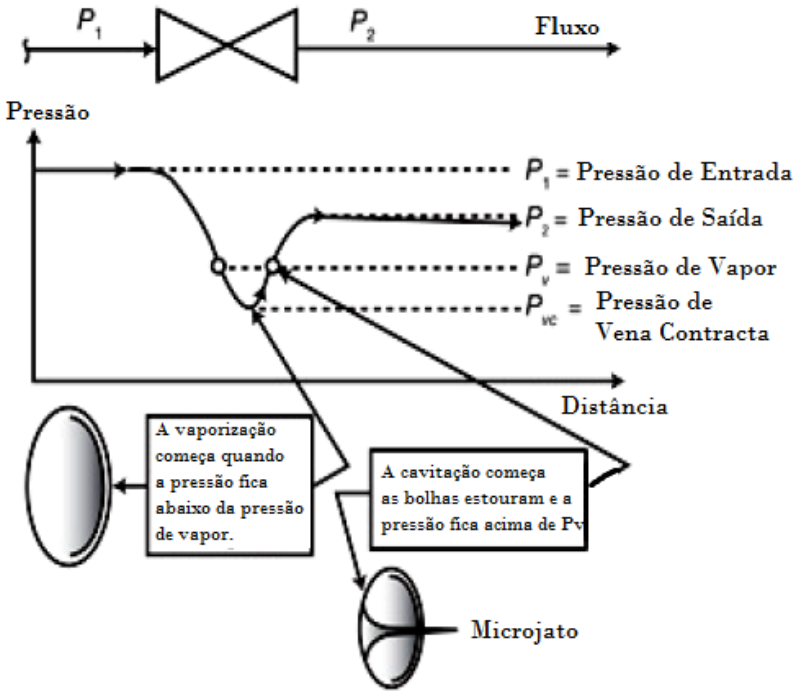
Em malhas de controle, as válvulas são responsáveis por controlar o fluxo de líquidos entregues ao processo modificando a abertura do obturador através de um atuador. Tais válvulas comumente operam com um obturador semiaberto, causando uma grande pressão diferencial entre o fluxo a jusante e a montante da válvula, o que pode causar cavitação e ruído (BO-SUK et al., 2005).

A cavitação é a súbita vaporização seguida da condensação de um líquido à jusante da válvula devido a zonas localizadas de baixa pressão (HUBBALLI; SONDUR, 2013, p.110). O fluxo de um líquido por uma tubulação sofre um aumento de velocidade quando este encontra uma restrição, como a válvula, causando um aumento da pressão dinâmica e uma diminuição da pressão estática de acordo com a Lei de Bernoulli expressa na Equação (1), onde ρ representa a densidade do fluido, P é a pressão, g a aceleração da gravidade, V é a velocidade do fluxo e z é a elevação do ponto medido (CARLSON, 2001).

$$\frac{V^2}{2} + \frac{P}{\rho} + gz = cte \tag{1}$$

Quando a velocidade do fluxo é suficientemente alta, a queda na pressão estática é tal que faz com que o líquido atinja o ponto de vaporização, formando bolhas de vapor. Após a passagem pela obstrução a velocidade e pressão voltam aos níveis anteriores fazendo com que estas bolhas eclodam, como mostrado na Figura 2.

Figura 2 - Diagrama de pressões ao longo do processo de cavitação



Fonte: HUBBALLI; SONDUR, 2013, modificada pela autora.

A rápida eclosão destas bolhas de vapor gera micro jatos de água de alta pressão em áreas muito pequenas que causam um alto ruído e, eventualmente, erosão dos internos da tubulação e da válvula. Assim, o rápido diagnóstico da cavitação permite que ações sejam tomadas de forma a mantê-la em níveis abaixo dos níveis de dano diminuindo também a vibração e o ruído do processo (HUBBALLI; SONDUR, 2013).

Existem vários indicadores que podem ser analisados para verificar a ocorrência da cavitação com ajuda de sensores como transdutores piezoeléctricos (SHUKRI; MUN; IBRAHIM, 2011), microfones capacitivos, acelerômetros (ULANICKI; PICINALLI; JANUS, 2015) e microfones de eletreto (PAES; MUNARO; CIARELLI, 2016).

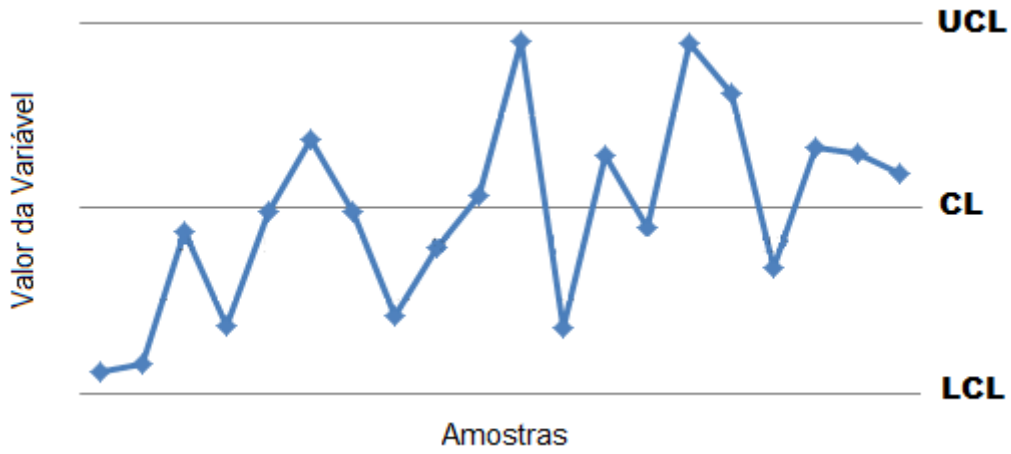
2.2 Cartas de controle

Em todos os processos de produção existe certa variabilidade dos elementos do processo, como ruído, vibração e etc. Quando esta variação é pequena e devida a causas inevitáveis as mesmas são consideradas "ruídos de plano de fundo" e é dito que o processo está em controle estatístico (MONTGOMERY; RUNGER, 2002).

Além desta variabilidade inerente, existem alguns fatores que podem afastar o funcionamento do processo do ponto esperado, como um comportamento atípico de um equipamento. Tais fatores, como a cavitação a ser diagnosticada neste projeto, tiram o processo do estado de controle estatístico e o fazem operar em níveis não desejados. A carta de controle é uma análise online utilizada para verificar esta variabilidade acima do normal e reportar a não conformidade do processo (MONTGOMERY; RUNGER, 2002).

Na Figura 3 está representada uma típica carta de controle de uma característica do processo. A carta é formada por uma linha central que indica a média dos valores que a variável analisada assume ao longo do processo, um limite de controle superior (UCL) e um limite de controle inferior (LCL) que delimitam a variabilidade inerente ao processo.

Figura 3 - Carta de Shewhart com limites superior e inferior



Fonte: Própria autora.

Após a definição dos limites superior e inferior e da linha central, o processo é monitorado de forma a identificar o comportamento da variável ao longo do tempo e verificar se a mesma ultrapassou os limites de controle, representando, assim, uma variação atípica do sistema.

De acordo com Montgomery e Runger (2002), ainda que todos os pontos analisados estejam entre os limites de controle, se eles se comportarem de forma padronizada e não aleatória isto é um indício de que sistema não está mais em controle estatístico.

Existem vários tipos de cartas de controle com diferentes definições dos limites superior e inferior e da linha central, mas apenas duas serão usadas neste projeto, a Carta de Shewhart e a Carta S.

Uma forma típica de determinar os limites de uma carta de controle é como mostrada pelas Equações (2), (3) e (4), onde μ é a média dos valores usados para o treinamento do modelo, σ é o desvio padrão das amostras e k é uma constante.

$$UCL = \mu + k\sigma \tag{2}$$

$$CL = \mu \tag{3}$$

$$LCL = \mu - k\sigma \tag{4}$$

O valor de k é determinado de acordo com a distribuição do sinal e usualmente é definido como sendo três (MONTGOMERY; RUNGER, 2002). Dr. Walter A. Shewhart foi um físico, engenheiro e estatístico e o primeiro a propor este modelo de carta de controle e por isso a mesma leva o seu nome (MONTGOMERY; RUNGER, 2002).

Em Montgomery e Runger (2002) mostra-se que a Carta S é uma carta de controle que avalia a variação do desvio padrão das amostras em relação ao modelo. A média dos desvios padrão usado nesta carta é calculado de acordo com a Equação (5), onde se tem m grupos de n amostras, σ_i representa o desvio padrão de cada grupo e os limites são calculados de acordo com as Equações (6), (7) e (8), onde c_4 é um parâmetro estatístico determinado de acordo com o número de grupos e amostras utilizados.

$$\bar{\sigma} = \frac{1}{m}\sum_{i=1}^{m}\sigma_i \quad (5)$$

$$UCL = \bar{\sigma} + k\frac{\bar{\sigma}}{c_4}\sqrt{1-c_4^2} \quad (6)$$

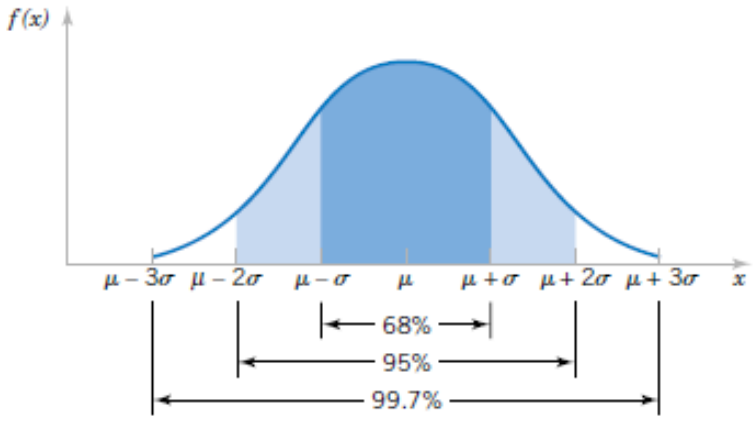
$$CL = \bar{\sigma} \quad (7)$$

$$LCL = \bar{\sigma} - k\frac{\bar{\sigma}}{c_4}\sqrt{1-c_4^2} \quad (8)$$

Uma análise dos comportamentos estatísticos da emissão sonora utilizada neste projeto foi feita em Paes, Munaro e Ciarelli (2016) e concluiu-se que as maiores variações do sinal são do fator forma (*s*) e do desvio padrão (σ), justificando o uso da Carta de Shewhart e da Carta S para o diagnóstico.

Em Paes, Munaro e Ciarelli (2016) foi realizado um teste de hipótese de Kolmogorov-Smirnov e verificado que os dados de emissão sonora possuem uma distribuição normal de densidade, representada na Figura 4, sendo este um pré-requisito para o uso de cartas de controle.

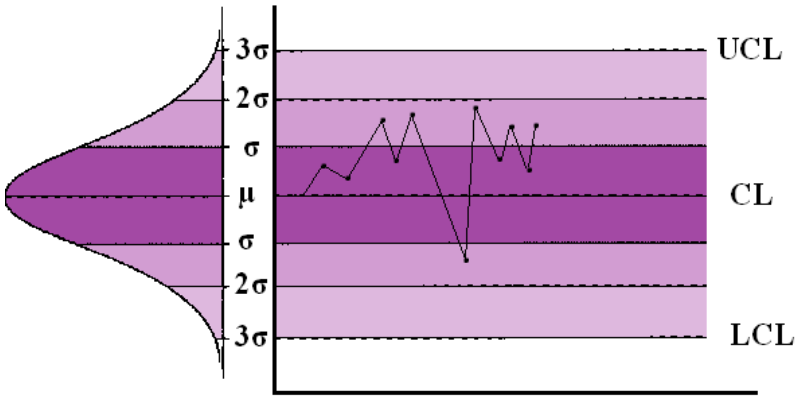
Figura 4 - Distribuição de densidade normal



Fonte: MONTGOMERY; RUNGER, 2002.

A Figura 5 mostra a relação entre os limites e o *k* escolhido. Para assegurar que os pontos fora dos limites representem a cavitação, escolheu-se um *k* igual a 2,5, de forma que no mínimo 98,758% das amostras analisadas deverão ficar dentro dos limites em caso de controle estatístico.

Figura 5 - Relação entre a densidade de dados da distribuição normal e os limites da carta de controle



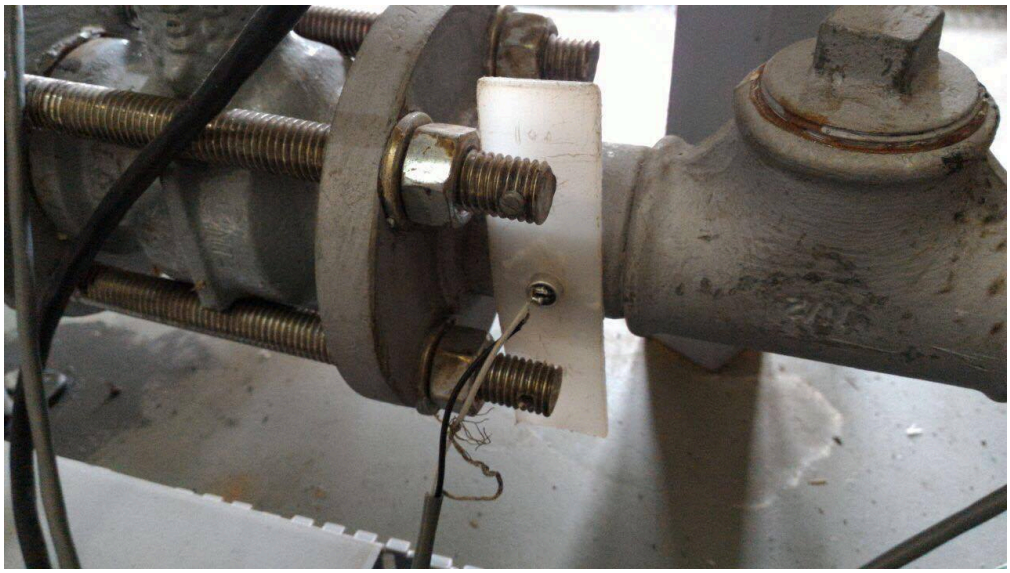
Fonte: Própria autora.

3 METODOLOGIA

3.1 Circuito amplificador do microfone de eletreto

Na planta piloto usada para os testes do sistema há um microfone de eletreto acoplado à tubulação, como mostrado na Figura 6 para captar a emissão sonora da mesma. O microfone precisa ser polarizado e ter seu sinal amplificado para que possa ser processado e analisado em um microcontrolador.

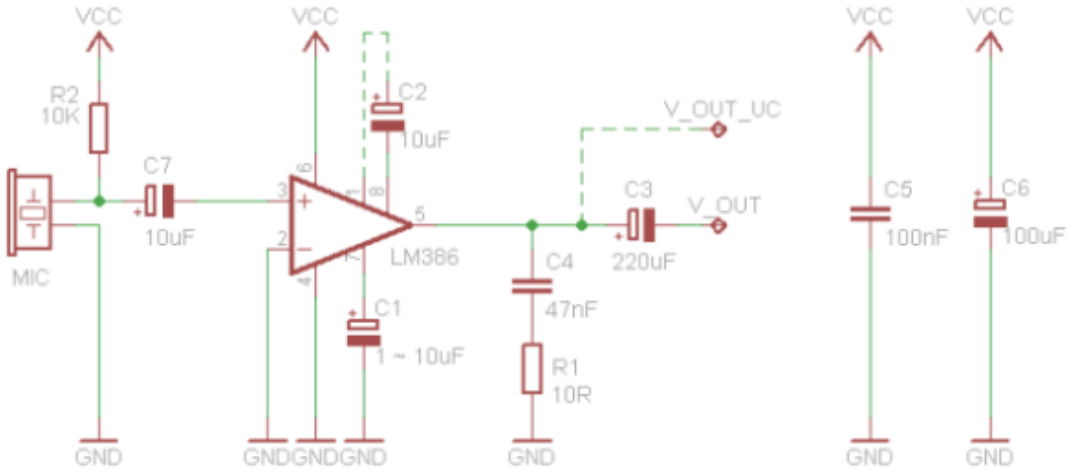
Figura 6 - Microfone de eletreto acoplado à tubulação.



Fonte: Própria autora.

Para a amplificação foi usado um amplificador de áudio LM386 de baixa tensão e o circuito mostrado na Figura 7 baseado no circuito sugerido na folha de dados do amplificador.

Figura 7 - Esquemático do circuito de polarização e amplificação do microfone de eletreto



Fonte: KOVACHEV, 2011.

A polarização do microfone é feita pelo resistor R2 e o capacitor C7, enquanto os capacitores C5 e C6 são usados para evitar ruídos de alimentação. O conjunto C4 e R1 é conhecido como célula de Boucherot e é usado para prevenir oscilações de alta frequência (KOVACHEV, 2011). O capacitor C3 é usado apenas para filtrar a componente de corrente contínua do sinal caso a saída vá para um autofalante. A Tabela 1 apresenta os valores e descrição dos componentes utilizados.

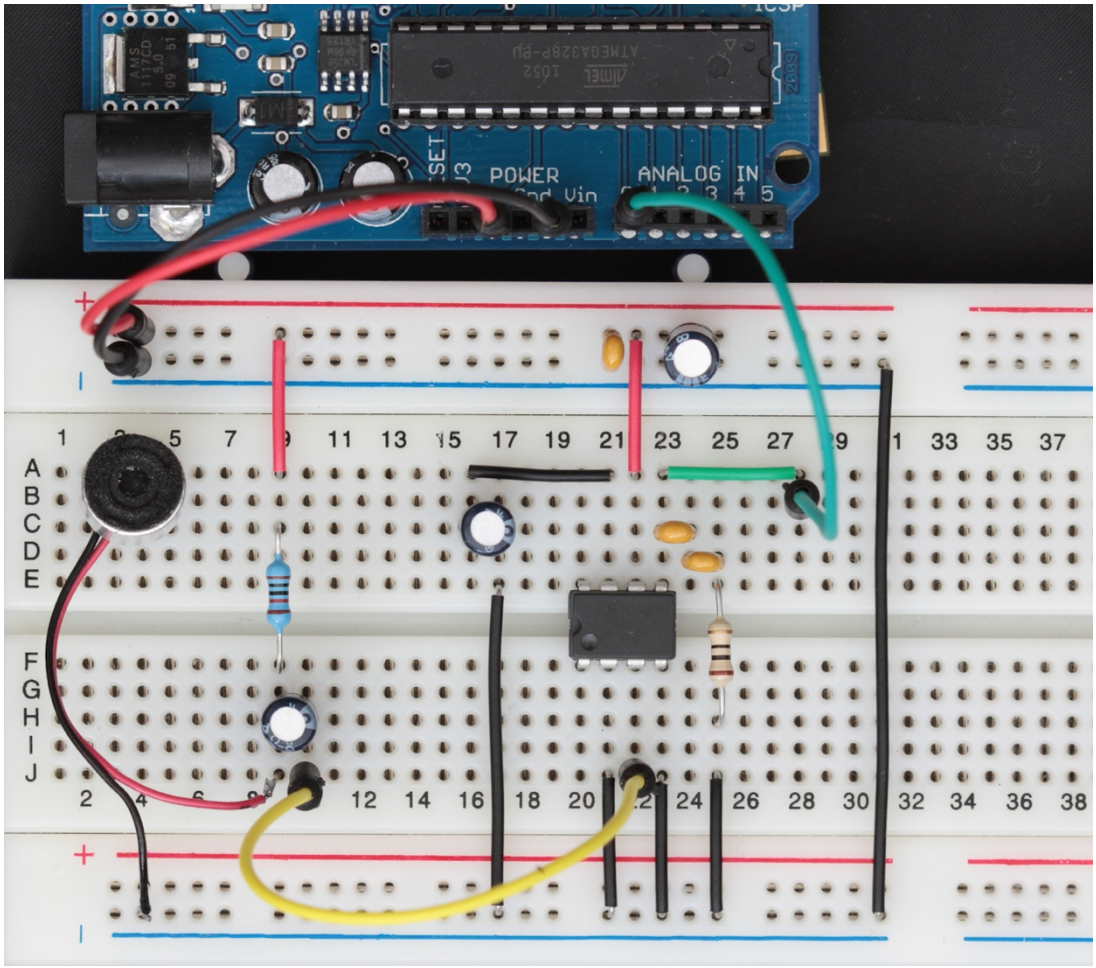
Tabela 1 - Tabela de componentes do circuito de polarização e amplificação do microfone de eletreto

Componente	Valor	Descrição
C1	10µF	Capacitor de desvio
C3	220µF	Capacitor de acoplamento de saída
C4	47nF	Célula Boucherot
C5	100nF	Filtro de alimentação
C6	100µF	Filtro de alimentação
C7	10µF	Polarizador do microfone
MIC		Microfone de eletreto
R1	10Ω	Célula Boucherot
R2	1 a 10KΩ	Polarizador do microfone
VSS	5V	Tensão de alimentação

Fonte: Própria autora.

Na Figura 8 tem-se um exemplo da montagem em *protoboard* do circuito de amplificação da saída do microfone com um ganho igual a 20.

Figura 8 – Circuito de amplificação em *protoboard*.

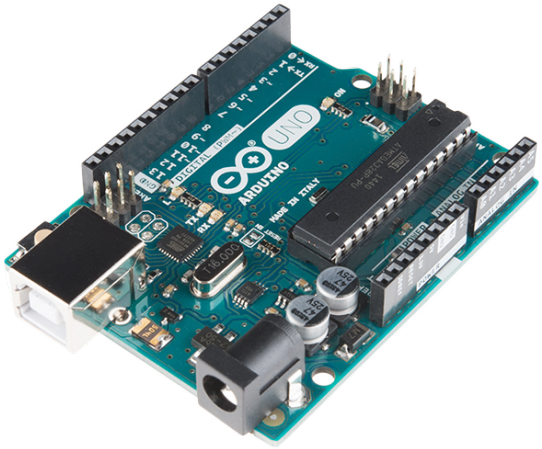


Fonte: KOVACHEV, 2010.

3.2 Arduino Uno

O Arduino é uma plataforma eletrônica de código aberto baseada em facilitar a implementação de *softwares* e *hardwares*. O Arduino Uno utilizado neste projeto é uma placa com um microcontrolador ATmega328P com um cristal de quartzo de 16MHz.

Figura 9 - Arduino Uno



Fonte: ARDUINO, 2016.

A Figura 9 mostra um exemplo da placa utilizada com 14 entradas/saídas digitais, seis entradas analógicas, uma conexão USB, uma entrada de alimentação e um botão de *reset*. O Arduino Uno possui 32KB de memória flash, 2K bytes de SRAM (*Static Random Access Memory* - Memória estática de acesso aleatório) e 1KB de EEPROM (*Electrically-Erasable Programmable Read-Only Memory* - Memória programável não volátil).

A escolha do Arduino Uno foi feita com o intuito de permitir fácil replicação do sistema montado no projeto em caso de aprimoramento ou somente aprendizado, além do seu preço acessível e sua vasta gama de bibliotecas disponíveis.

3.3 *Shields* de display LCD e leitura/escrita em cartão SD

Um dos objetivos do projeto é que o sistema possa ser operado *in loco*, ou seja, que a decisão de treinamento de dados ou diagnóstico do sistema possa ser tomada pelo operador sem a necessidade de um computador. Para tal é necessário um display onde seja possível selecionar as opções de um menu como o mostrado na Figura 10.

Figura 10 - Display de LCD com teclado

Fonte: DFROBOT, 2014.

Devido à pequena memória disponível no Arduino Uno e ao alto número de dados que precisarão ser armazenados e analisados no projeto (aproximadamente 14KB) a solução encontrada foi utilizar um *shield* que permite a leitura e escrita de dados em um cartão SD (*Secure Digital*), como o mostrado na Figura 11, dessa forma os dados de treinamento e diagnóstico ficarão armazenados e poderão ser utilizados para outros fins.

Figura 11 - *Shield* de leitura e escrita no cartão SD

Fonte: CATAZINELIVE, 2016.

3.4 Algoritmo desenvolvido

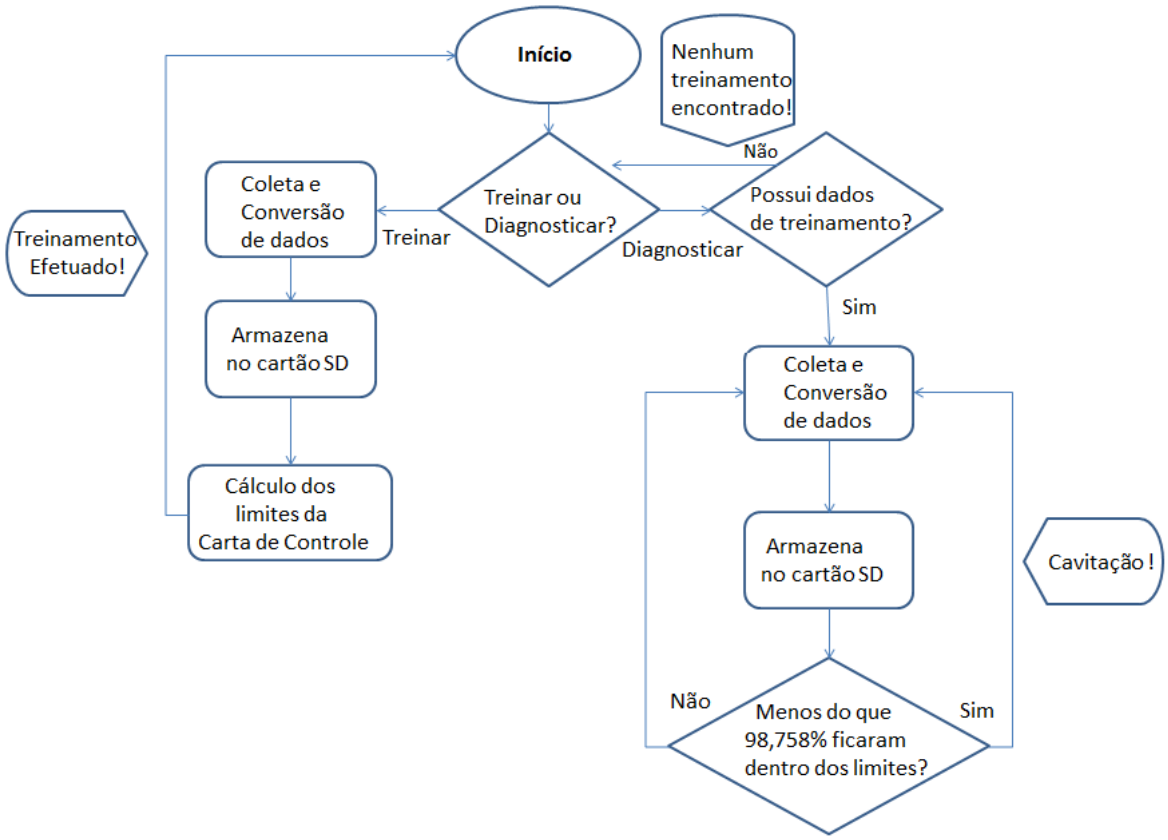
O algoritmo foi desenvolvido utilizando a linguagem de programação C e baseando-se no conceito de carta de controle descrito anteriormente. O código completo encontra-se no APÊNDICE A ao final deste manuscrito.

Inicialmente, é dada a opção de treinamento ou diagnóstico ao usuário. Selecionado o treinamento, inicia-se a coleta e conversão de dados do sinal analógico de saída do microfone de eletreto.

Após a coleta dos dados, são calculadas a média e o desvio padrão destes e definidos os limites superior e inferior da carta de controle. O usuário é informado pelo display LCD que o treinamento foi efetuado e lhe é dada novamente a opção de escolha entre realizar um novo treinamento ou iniciar o diagnóstico.

Selecionada a opção de diagnóstico, são coletadas novas janelas de dados e é verificado se uma porcentagem suficiente de dados se encontra acima do limite superior ou abaixo do limite inferior da carta. Encontrada essa situação é reportado, via display LCD, o diagnóstico de cavitação leve, média ou forte. A Figura 12 mostra um fluxograma de funcionamento do algoritmo.

Figura 12 - Diagrama de funcionamento do algoritmo



Fonte: Própria autora.

3.4.1 Coleta e conversão de dados

Os dados coletados pelo Arduino são a saída do circuito amplificador do microfone apresentado na Figura 7, ou seja, são sinais analógicos. Para a correta interpretação dos dados pelo microcontrolador, é necessário que haja uma conversão dos dados de analógicos para digitais (ADC).

Apesar da frequência da unidade central de processamento (CPU) do Arduino ser de 16MHz, a frequência de conversão AD utilizada em comandos como "*analogRead( )*" é menor, dividida por um fator de escala, além de levar 13 ciclos de relógio do conversor para ser completada (ATMEL CORPORATION, 2015).

Para atingir uma frequência de coleta de 9600Hz, é preciso levar em consideração tanto a velocidade de conversão AD quanto o tempo de escrita no cartão SD visto que o Arduino não possui memória SRAM suficiente para armazenar aproximadamente 14KB (7112 inteiros de 2 bytes cada) de dados por treinamento.

Contornando o uso do "*analogRead( )*", a conversão pode ser feita a nível de registradores, evitando operações desnecessárias e modificando o fator de escala do relógio ADC para assegurar a alta velocidade desejada.

Os registradores associados à conversão AD são o multiplexador de seleção (ADMUX), o de controle e status A e B (ADCSRA e ADCSRB) e os de dados (ADCL e ADCH). O conversor AD tem 10 bits de resolução (ATMEL CORPORATION, 2015), ou seja, os valores estarão entre 0 e 1023. A referência de conversão indicará qual tensão corresponderá ao valor máximo.

Figura 13 - Registrador ADMUX

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Fonte: ATMEL CORPORATION, 2015.

No registrador ADMUX, representado na Figura 13, os bits de acordo com a folha de dados são:

- REFS0 e REFS1: Referência de tensão para conversão;
- ADLAR: Define como o resultado será armazenado nos registradores ADCL e ADCH e não será usado neste projeto;
- MUX0 a MUX3: Definem o canal analógico a ser lido.

Figura 14 - Registrador ADCSRA

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Fonte: ATMEL CORPORATION, 2015.

Ainda de acordo com a folha de dados do Arduino, o registrador ADCSRA, representado na Figura 14, possui os seguintes bits:

- ADEN: Permite que a conversão ocorra;
- ADSC: Inicia a conversão quando em nível lógico alto e volta para zero ao final da conversão;
- ADATE: Permite o gatilho automático de conversão, ou seja, permite que a conversão aconteça assim que, como usado neste projeto, por exemplo, uma interrupção de tempo termine;
- ADIF: Passa para nível lógico alto quando a conversão termina;
- ADIE: Permite uma interrupção assim que a conversão termine;
- ASPS0 e ADPS1: Definem o fator de escala do conversor.

Figura 15 - Registrador ADCSRB

Bit	7	6	5	4	3	2	1	0	
(0x7B)	-	ACME	-	-	-	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Fonte: ATMEL CORPORATION, 2015.

Por fim, no registrador ADCSRB, mostrado na Figura 15, serão usados apenas os bits ADTS0 a ADTS2 que selecionam a fonte do gatilho para a conversão.

Além de garantir que a conversão ocorra rapidamente, é necessário que ela seja feita no intervalo correto de tempo para assegurar a frequência de 9600Hz que permite que a cavitação seja identificada (PAES; MUNARO; CIARELLI, 2016). Para tal, foi utilizada uma

interrupção de tempo na frequência determinada que ativa a conversão AD. Na Figura 16 é mostrado o pedaço do código onde foi implementada a conversão.

Figura 16 - Parte de código responsável pelas configurações da conversão AD

```
//A1 de entrada e Vcc de referência
ADMUX = (1 & 7) | (1 << REFS0)

//Define a interrupção do Timer 1 como gatilho da conversão AD
ADCSRB = (1 << ADTS2) | (1 << ADTS0)

//Enable, interrupção como gatilho, limpa interrupção, interrupção AD após conversão, prescaler 64
ADCSRA = (1 << ADEN) | (1 << ADATE) | (1 << ADIF)| (1 << ADIE) | (1 << ADPS2) | (1 << ADPS1)
```

Fonte: Própria autora.

Como mostrado na Figura 16, um fator de escala (*prescaler*) de 64 foi definido para a conversão AD, pois ele equivale a uma frequência de 250KHz por ciclo, e como cada conversão tem 13 ciclos, resulta em aproximadamente 19KHz de frequência de conversão, mais do que suficiente para que a conversão termine antes que uma outra interrupção de tempo (9600Hz) ocorra, resolvendo o problema de leitura e conversão de dados.

3.4.2 Escrita e armazenamento no cartão SD

Como dito anteriormente, o tempo de escrita no cartão SD é importante para garantir que a coleta de dados ocorra corretamente. Os dados devem ser passados para o cartão à medida que forem coletados, pois o Arduino não possui memória suficiente para armazenar todos os dados de treinamento da carta de controle.

Para otimizar o tempo de escrita no cartão SD primeiro deve-se pensar no tamanho do arquivo a ser escrito. A forma normal de escrita de dados no SD escreve arquivos em um formato legível por humanos (ASCII), porém tais arquivos acabam sendo maiores do que os dados escritos de forma crua (binário) o que faz com que o tempo de escrita seja maior.

Para evitar a escrita de arquivos maiores do que o estritamente necessário utiliza-se uma técnica chamada serialização que é a tradução de uma grande estrutura de dados para um formato facilmente legível pela máquina, desta forma, evita-se os bytes extras.

Esta forma de escrita é exemplificada na Figura 17, onde o endereço da variável a ser escrita é passado como parâmetro para que a função de escrita veja-o apenas como um arranjo de bytes, ao invés de um inteiro.

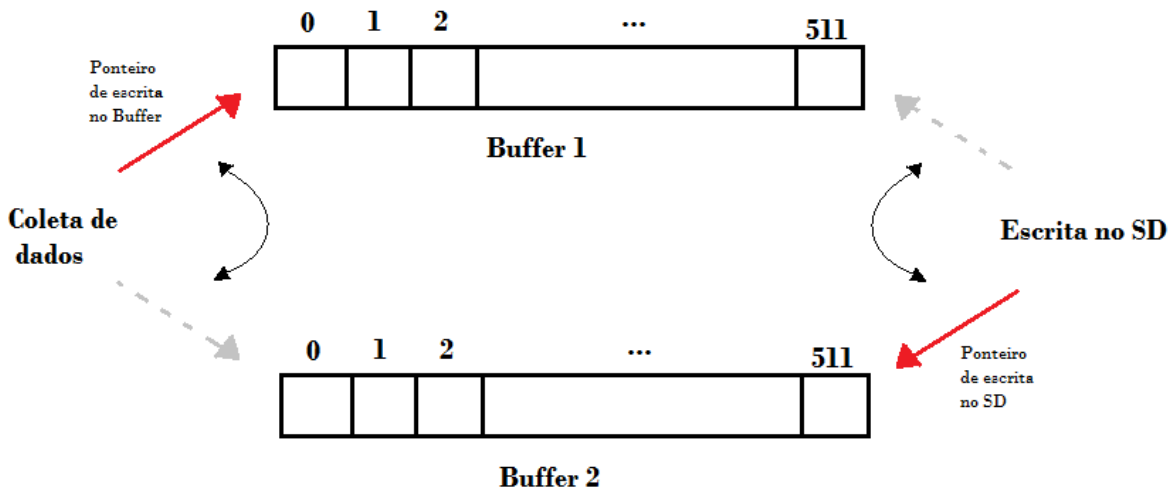
Figura 17 - Linha de código que permite a escrita em binário no cartão SD

```
sd.card()->writeData((uint8_t*)pBlock)  //Escreve um array de 512 bytes
```

Fonte: Própria autora.

O segundo problema é determinar se os dados serão escritos dado a dado no cartão SD ou em blocos. O cartão SD possui uma memória *cache* de 512 bytes que é carregada e escrita de uma vez no arquivo determinado. Assim, para otimizar o bloco de dados escritos, utilizou-se um método de dois buffers em que enquanto um é preenchido com os valores da conversão AD, o outro é escrito no cartão SD, como esquematizado na Figura 18.

Figura 18 - Diagrama do uso de dois buffers para coleta de dados e escrita no cartão SD



Fonte: Própria autora.

Os dados são escritos, então, em blocos de 512 bytes, onde os primeiros quatro bytes são um cabeçalho e os 508 restantes são as informações desejadas, ou seja, 254 inteiros de dois bytes cada.

A interrupção de tempo implementada na coleta de dados assegura por meio de uma bandeira que o carregamento e escrita dos buffers não se sobreponham, reportando um erro se isso acontecer.

3.4.3 Cálculo de UCL e LCL e diagnóstico

Com os dados coletados e armazenados corretamente, os cálculos dos limites superior (UCL) e inferior (LCL) da carta de controle podem ser feitos.

São coletadas 7112 amostras a uma frequência de 9600 Hz que posteriormente são divididas em 56 grupos de 127 amostras cada. Esta divisão da janela de amostragem é feita para calcular o desvio padrão de acordo com a Equação (5) para uma Carta S.

Como o número de amostras usada em cada grupo é alto (n maior do que 25), o valor de c_4 se aproxima suficientemente de um para que os cálculos dos limites da Carta S se tornem irrelevantes, sendo jogados os valores calculados para a Carta de Shewhart (MONTGOMERY; RUNGER, 2002).

Como visto no Capitulo 2, o valor de k escolhido para os limites foi igual a 2,5, de forma que os cálculos da média, de UCL e LCL são feitos de acordo com o código mostrado pela Figura 19.

Figura 19 - Código responsável pelo cálculo dos limites superior e inferior da carta de controle

```
while (binFile.read(&buf , 512) == 512) {
  if (buf.count == 0) {
    break;
  }

  for (uint16_t i = 0; i < buf.count/2; i++) {
    Xi += buf.data[i];              //somatório de Xi
    Xiq += (buf.data[i]*buf.data[i]);     //somatório de Xi^2
  }
  dp += sqrt((Xiq-(Xi*Xi)/127))/126);     //somatório dos desvios padrão das 127 amostras
  media += Xi/127;                                //média das 127 amostras
  Xi = 0;
  Xiq = 0;
  for (uint16_t i = buf.count/2; i < buf.count; i++) {
    Xi += buf.data[i];
    Xiq += (buf.data[i]*buf.data[i]);
  }
  dp += sqrt((Xiq-((Xi*Xi)/127))/126);
  media += Xi/127;
  Xi = 0;
  Xiq = 0;
}
dp = dp/56;                     //desvio padrão médio dos 56 grupos
media = media/56;               //média das médias dos 56 grupos
UCL = (2.5*dp)+media;                //calcula limite superior
LCL = media-(2.5*dp);                //calcula limite inferior
```

Fonte: Própria autora.

Na etapa de diagnóstico, foram coletadas 1016 amostras por grupo e verificado se cada uma destas se encontrava dentro dos limites estabelecidos. Se mais do que 13 amostras (mais do que 1,242% do total) estavam acima de UCL ou abaixo de LCL o sistema acusa o indício de cavitação.

A divisão dos níveis em cavitação fraca, média e forte é feita a partir do número de pontos fora dos limites da carta traçada. Os limites utilizados neste projeto foram entre 14 a 50 pontos para cavitação fraca, 50 a 200 para cavitação média e maior do que 200 para cavitação forte.

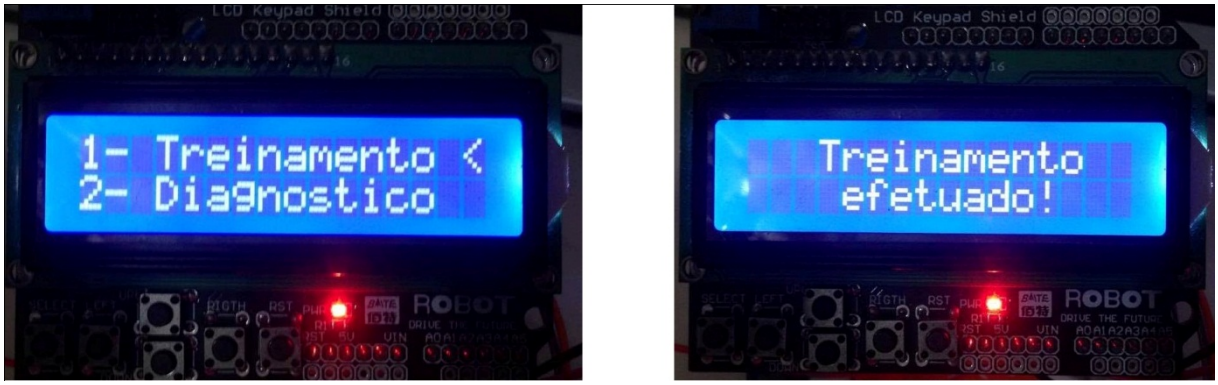
4 APLICAÇÃO E DISCUSSÃO

4.1 Operação do protótipo

O sistema foi programado para ser operado apenas pelo teclado do display LCD, dispensando qualquer tipo de configuração prévia.

Ao alimentar o Arduino, o display mostra duas opções para o usuário, Figura 20 (a), realizar o treinamento do sistema ou realizar o diagnóstico de cavitação. Se o usuário selecionar a opção de treinamento, o mesmo é efetuado e uma mensagem, Figura 20 (b), indica o seu término, retornando ao menu inicial.

Figura 20 - Display LCD: Menu inicial (a) e mensagem pós treinamento (b)

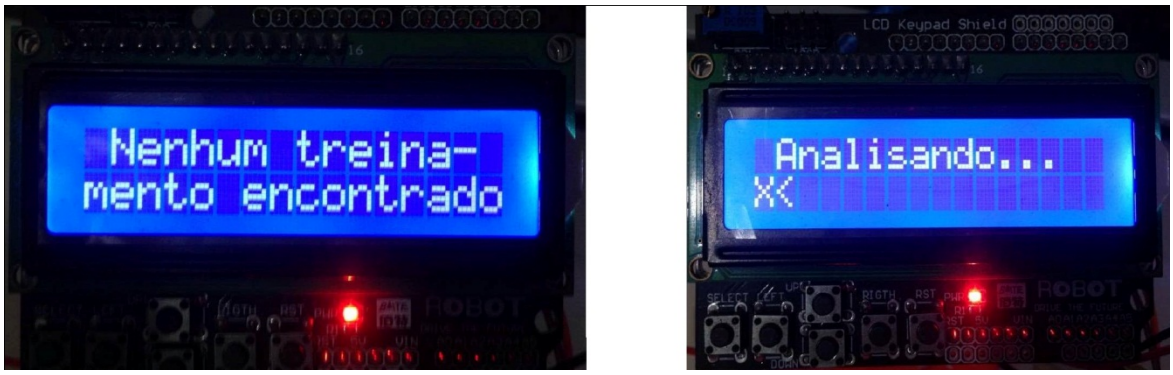


(a) (b)

Fonte: Própria autora.

Se a opção de diagnóstico for selecionada e não houver registro de um treinamento anterior o display apresenta uma mensagem de erro, Figura 21 (a), e retorna ao menu inicial. Se um treinamento foi identificado, o programa inicia o diagnóstico mostrando a tela de análise, Figura 21 (b), da qual o usuário pode sair pressionando o ‘X’.

Figura 21 - Display LCD: Aviso de falta de treinamento (a) e em analise (b)



(a) (b)

Fonte: Própria autora.

A cavitação detectada foi dividida em três níveis: fraca, média e forte; e são reportadas de acordo com o mostrado na Figura 22.

Figura 22 - Display LCD: Reporte de cavitação

Fonte: Própria autora.

Todas as manipulações do display são feitas apertando as teclas de “cima”, “baixo” e “selecionar” escritas no próprio teclado.

4.2 Testes realizados

A planta piloto da UFES mostrada pela Figura 23 foi usada como locação para os testes. A válvula usada é do tipo globo rotacional, modelo 35-35212, diâmetro interno de 1”1/2, ANSI 150, contendo um obturador excêntrico e operando com água.

Figura 23 - Planta piloto na UFES

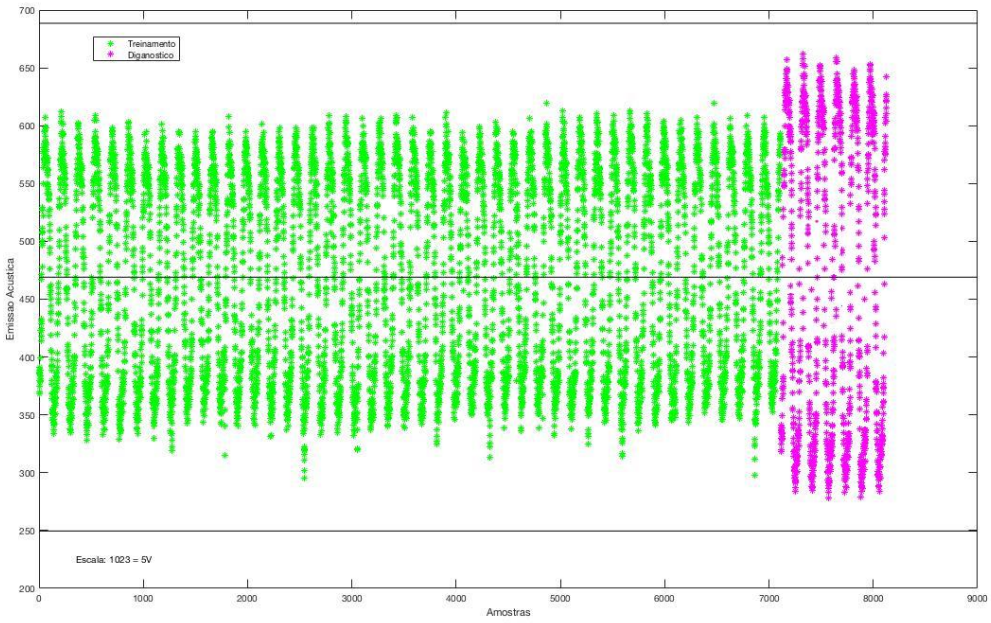
Fonte: Própria autora.

Durante os testes a temperatura da água foi mantida ambiente (27ºC) e foi utilizada uma bomba de múltiplos estágios tipo *Booster*, para fazer com que a água circulasse pela tubulação. Para controlar a abertura da válvula foi usado um controlador Freelancer 2000 Select da ABB disponível na planta que enviava sinais de 4 a 20mA correspondendo a 0 e 100% de abertura, respectivamente.

Para realizar o treinamento da carta de controle fixou-se a abertura da válvula em 50% e os diagnósticos foram realizados com 50, 20, 15 e 10% de abertura. Tal faixa de treinamento foi escolhida por representar o funcionamento normal do sistema. A válvula começa a apresentar sinais de cavitação quando a abertura da mesma está abaixo de 30% (PAES; MUNARO; CIARELLI, 2016), assim, o protótipo diagnosticou corretamente a cavitação fraca à 20% de abertura, média à 15%, forte à 10% e sem cavitação à 50%.

Na Figura 24 são apresentados os dados coletados durante o treinamento e os diagnósticos. Os primeiros 7112 dados são os de treinamento, e os 2032 seguintes são os de diagnóstico. Nota-se que os pontos que se encontram fora dos limites da carta de controle são cada vez mais numerosos à medida que acontece mais cavitação.

Figura 24 - Dados de treinamento (verdes) e dados de diagnóstico sem cavitação (rosa)

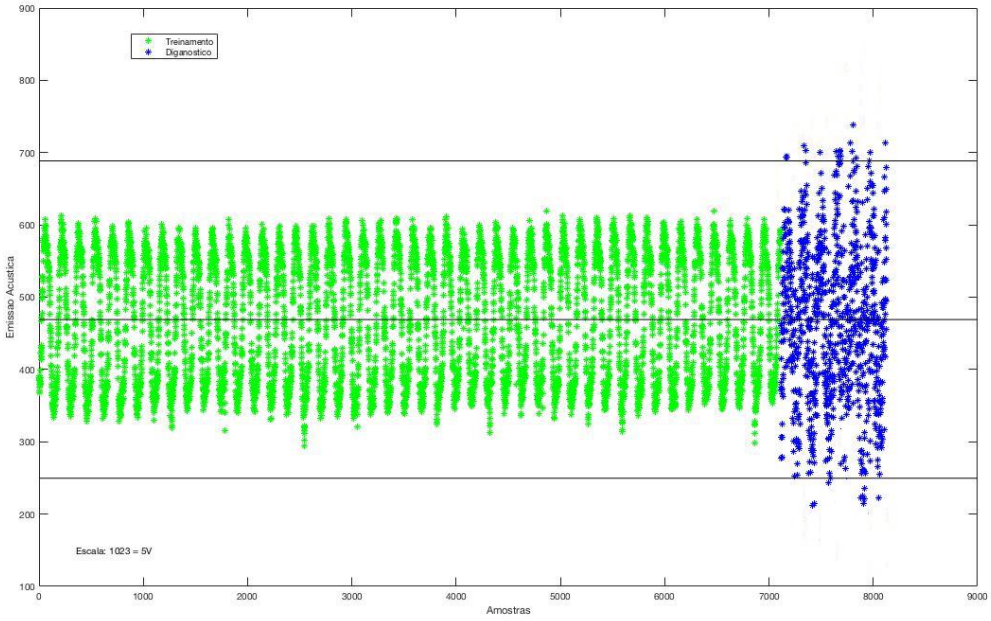


Fonte: Própria autora.

Os dados dos testes são armazenados em diferentes arquivos no cartão SD, separados por treinamento e níveis de cavitação. A Figura 24 mostra a emissão acústica (em valores digitalizados) pelo número de amostras, onde o treinamento foi feito com a válvula 50% aberta e o teste sem cavitação com a válvula 30% aberta o que permite notar que quando não acontece cavitação os limites da carta de controle não são ultrapassados.

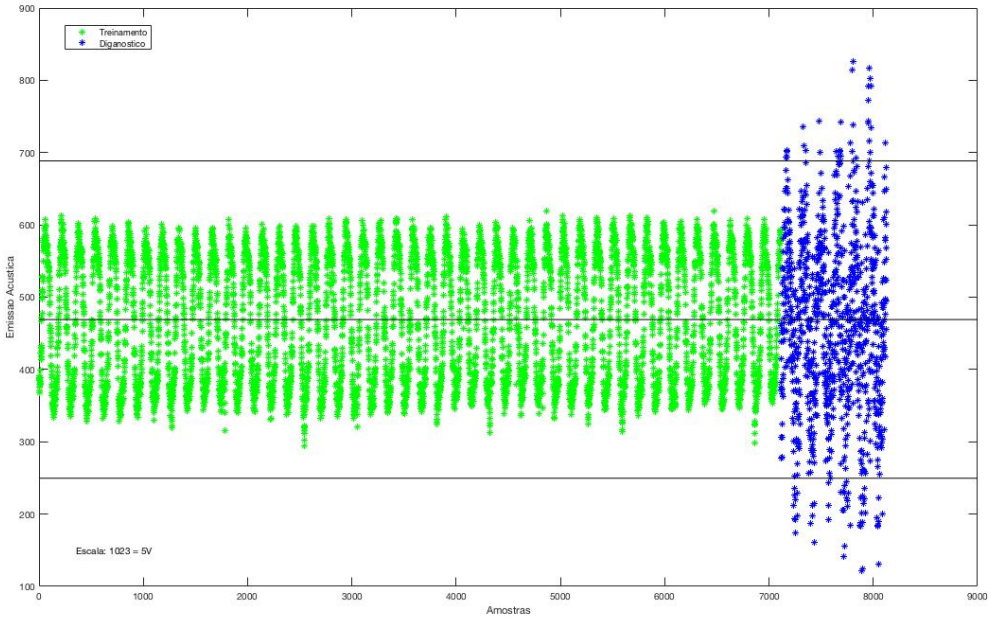
Já nas Figuras 25, 26 e 27 são apresentados os dados com cavitação fraca (válvula 20% aberta), média (válvula 15% aberta) e forte (válvula 10% aberta) respectivamente. Tais figuras exemplificam claramente o aumento do número de pontos fora dos limites da carta indicando um aumento na cavitação.

Figura 25 - Dados de treinamento (verdes) e dados de diagnóstico com cavitação fraca (azul)



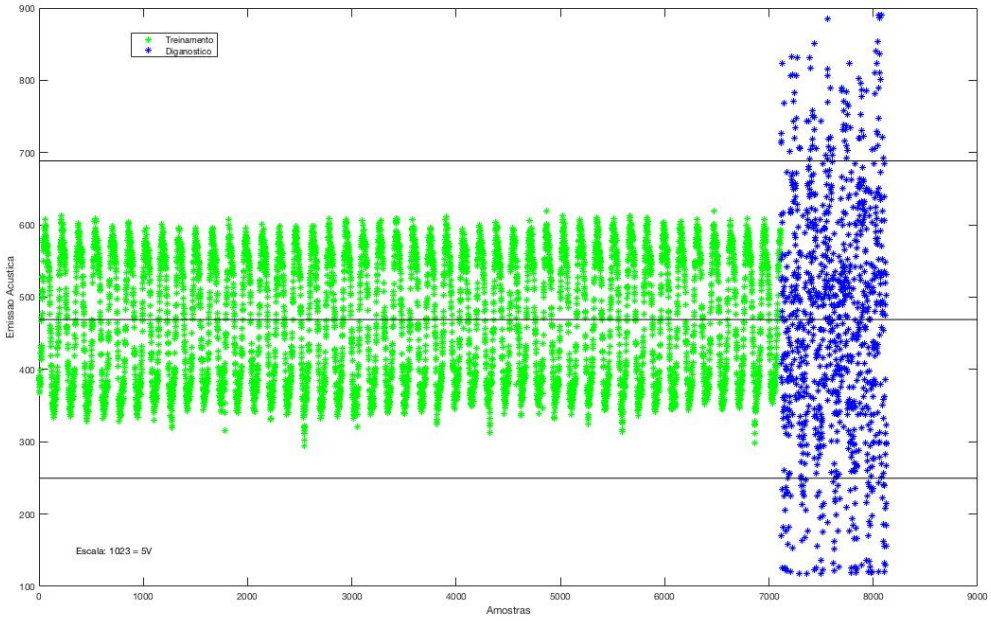
Fonte: Própria autora.

Figura 26 - Dados de treinamento (verdes) e dados de diagnóstico com cavitação média (azul)



Fonte: Própria autora.

Figura 27 - Dados de treinamento (verdes) e dados de diagnóstico com cavitação forte (azul)



Fonte: Própria autora.

5 CONCLUSÕES E TRABALHOS FUTUROS

Neste projeto foi mostrado um sistema embarcado de diagnóstico de cavitação em válvulas de controle por meio de análise de emissões sonoras. Foi discutida a metodologia utilizada para otimizar a coleta e conversão de dados, assim como a escrita em um cartão SD além de aplicada uma técnica de análise estatística conhecida como de carta de controle.

Foi comprovada a eficácia de cartas de controle no diagnóstico proposto e da utilização de métodos como serialização e manipulação de registradores para diminuir o tempo de leitura e escrita do Arduino Uno. Obtiveram-se também gráficos representativos do sistema estudado em casos de funcionamento normal ou funcionamento em cavitação.

Apesar de o projeto ter alcançado o seu objetivo principal, detectar a cavitação, a escolha do microcontrolador para embarcar o sistema poderia ter sido mais adequada à alta frequência de coleta de dados requisitada, tanto com uma frequência maior de relógio ou uma capacidade maior de armazenamento de dados. Além disto, visto o intuito do projeto em desenvolver uma plataforma autossuficiente, recomenda-se que em trabalhos futuros seja considerado o uso de uma bateria para facilitar a autonomia do projeto.

Alternativas para trabalhos futuros são implementar outro método estatístico para análise da emissão sonora da tubulação ou embarcar outras soluções já existentes, como a utilização de *Support Vector Machines* feita em Paes, Munaro e Ciarelli (2016). Tais projetos poderiam trabalhar em conjunto com este trabalho para aumentar a confiabilidade da detecção, evitando possíveis erros devidos a fatores não considerados no escopo do trabalho.

Ressalta-se que as dificuldades encontradas no desenvolvimento do projeto serviram como enriquecimento da discussão feita pelo mesmo ao invés de gerarem grandes atrasos. Por fim, conclui-se que abordagens de matemática estatística, eletrônica e computação são ótimas ferramentas para solucionar problemas de controle e instrumentação.

REFERÊNCIAS BIBLIOGRÁFICAS

ARDUINO. **Arduino UNO & Genuino UNO**. 2016. Disponível em: <https://www.arduino.cc/en/Main/ArduinoBoardUno>. Acesso em: 05 dez. 2016.

ATMEL CORPORATION. **ATmega48A/PA/88A/PA/168A/PA/328/P**: Atmel 8-bit microcontroller with 4/8/16/32KBytes in-system programmable flash datasheet. 2015. Disponível em: <http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf>. Acesso em: 05 dez. 2016.

BO-SUK, Y.; WON-WOO, H.; MYUNG-HAN, K.; SOO-JONG, L. Cavitation Detection of Butterfly Valve Using Support Vector Machines. **Journal of Sound and Vibration**, v.287, p. 25-43. 2005.

CARLSON, B. Avoiding Cavitation in Control Valves. **ASHRAE Journal**, v.43, n.6, p. 58-63. 2001.

CATAZINELIVE. **SD Card Reader Using Arduino**. 2015. Disponível em: <http://catazinelive.net/2013/06/sd-card-reader-using-arduino/>. Acesso em: 05 dez. 2016.

DFROBOT. **Arduino LCD Keypad Shield (SKU: DFR0009)**. 2014. Disponível em: <https://www.dfrobot.com/wiki/index.php/Arduino_LCD_KeyPad_Shield_(SKU:_DFR0009)>. Acesso em: 05 dez. 2016.

HUBBALLI, B. V.; SONDUR, V. B. A Review on the Prediction of Cavitation Erosion Inception in Hydraulic Control Valves. **International Journal of Emerging Technology and Advanced Engineering**, v.3, n.1, Jan. 2013.

KOVACHEV, D. **LM386 Microphone Amplifier**. 2011. Disponível em: <https://lowvoltage.wordpress.com/2011/05/15/lm386-mic-amp/>. Acesso em: 05 dez. 2016.

MONTGOMERY, D. C.; RUNGER, G. C. **Applied Statistics and Probability for Engineers**. John Wiley & Sons, inc., p. 595-612. 2002.

PAES, A. A.; MUNARO, C. J.; CIARELLI, P. M.; Diagnóstico de Cavitação nos Internos de Válvulas de Controle. **Congresso SBA**. 2016.

SHUKRI, I. N. B. M.; MUN, G. Y.; IBRAHIM, R. B. A Study on Control Valve Fault Incipient Detection Monitoring System Using Acoustic Emission Technique. **3rd International Conference on Computer Research and Development (ICCRD)**. 2011.

ULANICKI, B.; PICINALI, L.; JANUS, T. Measurements and Analysis of Cavitation in a Pressure Reducing Valve During Operations – A Case Study. **13th Computer Control for Water Industry Conference**. 2015.

APÊNDICE A – ALGORITMO DESENVOLVIDO

```
#include <LiquidCrystal.h>
#include <SPI.h>
#include "SdFat.h"
#include "AnalogBinLogger.h"

// --- Variáveis para o LCD ---
void keyboardRead();
boolean direita  = 0x00, butt01 = 0x00,
        cima     = 0x00, butt02 = 0x00,
        baixo    = 0x00, butt03 = 0x00,
        esquerda   = 0x00, butt04 = 0x00,
        select = 0x00, butt05 = 0x00;

LiquidCrystal disp(8,  //RS no digital 8
                   9,  //EN no digital 9
                   4,  //D4 no digital 4
                   5,  //D5 no digital 5
                   6,  //D6 no digital 6
                   7); //D7 no digital 7

int tecla = 0x00;                       //recebe o valor do teclado do LCD

//------------------------------------------------------------------------
-----

const uint8_t PIN_LIST[] = {1};
//------------------------------------------------------------------------
-----
const float SAMPLE_RATE = 9600;  // Frequência de aquisição de dados em Hz

const float SAMPLE_INTERVAL = 1.0/SAMPLE_RATE; //Tempo entre intervalos
baseado na sample_rate definida

#define ROUND_SAMPLE_INTERVAL 1  //Arredonda o tempo entre as amostras para
garantir uma taxa de coleta estável
//------------------------------------------------------------------------


uint8_t const ADC_REF = (1 << REFS0);  //Seta o VCC como referência para a
resolução da conversão (Tabela 24-3 no datasheet)

//------------------------------------------------------------------------

uint32_t FILE_BLOCK_COUNT = 29;   //(Quantidade de blocos de 512 bytes a
serem escritos)+1

#define FILE_BASE_NAME "bnalog"

//------------------------------------------------------------------------

const uint8_t SD_CS_PIN = 10;  //pino de chip select do SD
//------------------------------------------------------------------------


const uint8_t BUFFER_BLOCK_COUNT = 1;  //Usa dois buffers, o cache do SD+1
```

```
const uint8_t QUEUE_DIM = 4;  // Tamanho da fila de buffers

//================================================================================
#define TMP_FILE_NAME "tmp_log.bin"

const uint8_t PIN_COUNT = sizeof(PIN_LIST)/sizeof(PIN_LIST[0]);

const uint16_t MIN_ADC_CYCLES = 13;  //quantidade mínima de ciclos de clock
ADC por converção (o Uno normalmente leva 13 ciclos para completar uma
conversão)



const uint16_t ISR_TIMER0 = 160;  //Máximo número de ciclos para o system
interrupt do timer0 (millis, micros..)
//================================================================================
SdFat sd;

SdBaseFile binFile;

char binName[13] = FILE_BASE_NAME "00.bin";  //MODIFICAR PARA CADA ESCRITA


const size_t SAMPLES_PER_BLOCK = 254;  //Quantas amostras terão em um bloco
(2 bytes/amostra) menos 4bytes de cabeçário (512bytes/bloco)
typedef block16_t block_t;  //Define o tipo block16_t da biblioteca como
block_t


block_t* emptyQueue[QUEUE_DIM];   //Cria uma fila de buffers vazios com a
dimensão definida pelo número de buffers
uint8_t emptyHead;                //Head da fila vazia
uint8_t emptyTail;                //Tail da fila vazia

block_t* fullQueue[QUEUE_DIM];    //Cria uma fila de buffers cheios com a
dimensão definida pelo número de buffers
volatile uint8_t fullHead;        //Head da fila cheia
uint8_t fullTail;                 //Tail da fila cheia


inline uint8_t queueNext(uint8_t ht) {   //passa para o próximo elemento da
fila
  return (ht + 1) & (QUEUE_DIM -1);
}
//================================================================================
=====

block_t* isrBuf;    //Ponteiro para o buffer sendo utilizado


bool isrBufNeeded = true;   //Precisa pegar um buffer novo porque o antigo
já encheu

uint16_t isrOver = 0;       //contagem de overruns

// ADC
uint8_t adcmux[PIN_COUNT];
uint8_t adcsra[PIN_COUNT];
```

```
uint8_t adcsrb[PIN_COUNT];
uint8_t adcindex = 1;


volatile bool timerError = false;   //Flags de controle se a conversão AD
demorou mais tempo do que a interrupção de tempo leva para ser ativada
volatile bool timerFlag = false;
//-------------------------------------------------------------------------
-----
//Variáveis para definições da Carta de Shewhart

float media;          //média das amóstras
float dp;             //desvio padrão das amostras
uint16_t famostras;   //amostras fora dos limites da carta
float UCL;            //Upper Control Limit
float LCL;            //Lower Control Limit

boolean treinou = 0;            //treinamento finalizado(1)
boolean pos = 0;                //variável auxiliar de controle de posição
LCD: diagnóstico(1) ou treinamento(0)

//-------------------------------------------------------------------------
-----

ISR(ADC_vect) {   //Interrupção de conclusão ADC

  uint16_t d = ADC;  //Lê o dado do ADC (ADCL primeiro)

  if (isrBufNeeded && emptyHead == emptyTail) {  //se precisa de um novo
buffer mas não há nenhum na lista de buffers vazios

    if (isrOver < 0XFFFF) {   //incrementa contagem de overrun
      isrOver++;
    }
    timerFlag = false;  //já entrou na interrupção AD depois da de tempo
    return;
  }
  // Começa conversão

    timerFlag = false; //já entrou na interrupção AD depois da de tempo

  if (isrBufNeeded) {  //Checa se precisa de um buffer novo
    isrBuf = emptyQueue[emptyTail];       //isrBuf aponta para o ultimo
buffer livre da fila
    emptyTail = queueNext(emptyTail);     //o ultimo é incrementado
    isrBuf->count = 0;                    //contagem é zerada
    isrBuf->overrun = isrOver;            //e o número de overruns é
passado
    isrBufNeeded = false;                 //não precisa mais de buffer
porque já pegamos um
  }
  // Guarda o dado no Buffer
  isrBuf->data[isrBuf->count++] = d;      //adiciona o dado na posição
'count' do buffer e incrementa count


  if (isrBuf->count >= SAMPLES_PER_BLOCK) { //Checa se o Buffer está cheio
    uint8_t tmp = fullHead;
```

```
    fullQueue[tmp] = (block_t*)isrBuf;   //insere o buffer cheio na fila de
buffers cheios
    fullHead = queueNext(tmp);

    isrBufNeeded = true;     //preencheu um buffer então precisa de outro
    isrOver = 0;            //zera os overruns pro próximo buffer
  }
}
//------------------------------------------------------------------------
-----
// timer1 interrupt to clear OCF1B
ISR(TIMER1_COMPB_vect) { //Timer 1 interrupt de 9,6KHz
  if (timerFlag) {   //Conferir se a ADC aconteceu entre interrupções de
tempo
    timerError = true;
  }
  timerFlag = true;
}
//========================================================================
=====
// Mensagens de erro guardadas na memória flash para economizar RAM
#define error(msg) errorFlash(F(msg))
//------------------------------------------------------------------------
-----
void errorFlash(const __FlashStringHelper* msg) {
  sd.errorPrint(msg);
}
//------------------------------------------------------------------------
-----
//========================================================================
=====
#if ADPS0 != 0 || ADPS1 != 1 || ADPS2 != 2  //checa se os bits do prescaler
estão corretos
#error bits de ADC prescaler errados
#endif
//------------------------------------------------------------------------
-----
// Inicializa ADC e timer1
void adcInit(metadata_t* meta) {
  uint8_t adps;  //bits de prescaler para o ADCSRA
  uint32_t ticks = F_CPU*SAMPLE_INTERVAL + 0.5;  // Intervalo de amostragem
dos ciclos do cpu

  if (ADC_REF & ~((1 << REFS0) | (1 << REFS1))) { //Confere se a setagem
dos pinos de referência está correta
    error("Invalid ADC reference");
  }

  int32_t adcCycles = (ticks - ISR_TIMER0);

  for (adps = 7; adps > 0; adps--) {              //Define o prescaler do
ADC baseado no número de ciclos escolhido (para 13 vai ser um prescaler de
64)
    if (adcCycles >= (MIN_ADC_CYCLES << adps)) {
      break;
    }
  }

  meta->adcFrequency = F_CPU >> adps;  //250KHz
```

```
  if (meta->adcFrequency > 1000000) {
    error("Sample Rate Too High");
  }
#if ROUND_SAMPLE_INTERVAL
  // Arredonda para que o intervalo seja múltiplo do clock do ADC
  ticks += 1 << (adps - 1);
  ticks >>= adps;
  ticks <<= adps;
#endif  // ROUND_SAMPLE_INTERVAL

  meta->pinCount = PIN_COUNT;     // VOLTAR TIRANDO TODOS OS PIN_COUNT E
PIN_LIST

  for (int i = 0; i < PIN_COUNT; i++) {
    uint8_t pin = PIN_LIST[i];
    meta->pinNumber[i] = pin;


    adcmux[i] = (pin & 7) | ADC_REF;  //Recebe para setar a referência
escolhida antes e o pino de entrada (01000001)


    adcsrb[i] = i == 0 ? (1 << ADTS2) | (1 << ADTS0) : 0;    //Recebe para
fazer A INTERRUPÇÃO DE TEMPO TRIGGAR A CONVERSÃO AD

    adcsra[i] = (1 << ADEN) | (1 << ADIE) | adps;   //Recebe para dar
enable na conversão, liberar a interrupt de ADC e passar o prescaler
(10001110)
    adcsra[i] |= i == 0 ? 1 << ADATE : 1 << ADSC;  //liberar o auto-trigger
  }

  // Setup timer1
  TCCR1A = 0;
  uint8_t tshift;
  if (ticks < 0X10000) {
    // sem prescale, CTC mode
    TCCR1B = (1 << WGM13) | (1 << WGM12) | (1 << CS10);
    tshift = 0;
  } else if (ticks < 0X10000*8) {
    // prescale 8, CTC mode
    TCCR1B = (1 << WGM13) | (1 << WGM12) | (1 << CS11);
    tshift = 3;
  } else if (ticks < 0X10000*64) {
    // prescale 64, CTC mode
    TCCR1B = (1 << WGM13) | (1 << WGM12) | (1 << CS11) | (1 << CS10);
    tshift = 6;
  } else if (ticks < 0X10000*256) {
    // prescale 256, CTC mode
    TCCR1B = (1 << WGM13) | (1 << WGM12) | (1 << CS12);
    tshift = 8;
  } else if (ticks < 0X10000*1024) {
    // prescale 1024, CTC mode
    TCCR1B = (1 << WGM13) | (1 << WGM12) | (1 << CS12) | (1 << CS10);
    tshift = 10;
  } else {
    error("Sample Rate Too Slow");
  }
  // divide pelo prescaler
  ticks >>= tshift;
```

```
  // seta TOP para timer reset
  ICR1 = ticks - 1;
  OCR1B = 0;

  // multiplica pelo prescaler
  ticks <<= tshift;

  // Amostra o intervalo em CPU clock ticks.
  meta->sampleInterval = ticks;
  meta->cpuFrequency = F_CPU;
  meta->recordEightBits = 0;
}
//------------------------------------------------------------------------------
//da o enable do ADC e do timer1 interrupt
void adcStart() {
  // inicializa ISR
  isrBufNeeded = true;
  isrOver = 0;
  adcindex = 1;

  // Limpa qualquer interrupt pendente
  ADCSRA |= 1 << ADIF;

  // Setup para o primeiro pino
  ADMUX = adcmux[0];
  ADCSRB = adcsrb[0];
  ADCSRA = adcsra[0];

  // Enable timer1 interrupts.
  timerError = false;
  timerFlag = false;
  TCNT1 = 0;
  TIFR1 = 1 << OCF1B;
  TIMSK1 = 1 << OCIE1B;
}
//------------------------------------------------------------------------------
void adcStop() {  //para os interrupts
  TIMSK1 = 0;
  ADCSRA = 0;
}
//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
void dumpData(boolean limites) {
  block_t buf;
//  if(!binFile.open("bnalog00.bin", O_READ)){
//    Serial.println("naoleu");
//  }
  if (!binFile.isOpen()) {
    Serial.println(F("No current binary file"));
    return;
  }
  binFile.rewind();
  if (binFile.read(&buf , 512) != 512) {
    error("Read metadata failed");
  }
```

```
  uint32_t Xi = 0;      //Elemento i do buffer
  uint32_t Xiq = 0;     //Elemento i ao quadrado

  if(limites){      //limites == 1 se quiser fazer a carta e 0 se quiser
diagnosticar
    dp = 0;
    media = 0;
    while (binFile.read(&buf , 512) == 512) {
      if (buf.count == 0) {
        break;
      }

      for (uint16_t i = 0; i < buf.count/2; i++) {
        Xi += buf.data[i];              //somatório de Xi
        Xiq += pow(buf.data[i],2);      //somatório de Xi^2
//        Serial.println(buf.data[i]);
      }
      dp += sqrt((Xiq-(pow(Xi,2)/127))/126);     //somatório dos desvios
padrão das 127 amostras
      media += Xi/127;                                  //média das 127
amostras
      Xi = 0;
      Xiq = 0;
      for (uint16_t i = buf.count/2; i < buf.count; i++) {
        Xi += buf.data[i];
        Xiq += pow(buf.data[i],2);
 //        Serial.println(buf.data[i]);
      }
      dp += sqrt((Xiq-(pow(Xi,2)/127))/126);
      media += Xi/127;
      Xi = 0;
      Xiq = 0;
    }
    dp = dp/56;                       //desvio padrão médio dos 56 grupos
    media = media/56;                 //média das médias dos 56 grupos
    UCL = (2.5*dp)+media;                  //calcula limite superior
    LCL = media-(2.5*dp);                  //calcula limite inferior
  }
  else{
    famostras = 0;   //AMOSTRAS FORA DOS LIMITES!!!
    while (binFile.read(&buf , 512) == 512) {
      if (buf.count == 0) {
        break;
      }

      for (uint16_t i = 0; i < buf.count; i++) {
    //            Serial.println(buf.data[i]);
        if(buf.data[i]>UCL || buf.data[i]<LCL)   //Se a amostra estiver
fora dos limites determinado, possibilidade de cavitação
        {
          famostras += 1;
        }
      }

    }

  }
  Serial.println(F("Done"));
}
```

```
//------------------------------------------------------------------------------
-----
uint32_t const ERASE_SIZE = 29;
void logData() {
  uint32_t bgnBlock, endBlock;

  block_t block[BUFFER_BLOCK_COUNT];  //Aloca a quantidade de Buffers extra
determinada

  adcInit((metadata_t*) &block[0]);   //Inicializa o ADC e o timer1
interrupt


  // Cria um novo arquivo
  binFile.close();
  if (!binFile.createContiguous(sd.vwd(),
                                binName, 512 * FILE_BLOCK_COUNT)) {
    error("createContiguous failed");
  }
  // Pega o endereço do arquivo no SD
  if (!binFile.contiguousRange(&bgnBlock, &endBlock)) {
    error("contiguousRange failed");
  }
  // Usa o cache do SD como buffer
  uint8_t* cache = (uint8_t*)sd.vol()->cacheClear();
  if (cache == 0) {
    error("cacheClear failed");
  }

  // Apaga todos os possíveis dados do arquivo
  uint32_t bgnErase = bgnBlock;
  uint32_t endErase;
  while (bgnErase < endBlock) {
    endErase = bgnErase + ERASE_SIZE;
    if (endErase > endBlock) {
      endErase = endBlock;
    }
    if (!sd.card()->erase(bgnErase, endErase)) {
      error("erase failed");
    }
    bgnErase = endErase + 1;
  }
  // Começa a escrita de múltiplos blocos
  if (!sd.card()->writeStart(bgnBlock, FILE_BLOCK_COUNT)) {
    error("writeBegin failed");
  }
  // Write metadata.
  if (!sd.card()->writeData((uint8_t*)&block[0])) {
    error("Write metadata failed");
  }
  // Inicializa filas
  emptyHead = emptyTail = 0;
  fullHead = fullTail = 0;

  //Usa o cache como o primeiro bloco
  emptyQueue[emptyHead] = (block_t*)cache;
  emptyHead = queueNext(emptyHead);

  // Coloca os outros buffers na fila vazia
```

```
  for (uint8_t i = 0; i < BUFFER_BLOCK_COUNT; i++) {
    emptyQueue[emptyHead] = &block[i];
    emptyHead = queueNext(emptyHead);
  }
  // Dá um tempo pro SD se preparar para a grande escrita
  delay(1000);

  uint32_t bn = 1;      //numero do bloco
  uint32_t t0 = millis();
  uint32_t t1 = t0;
  uint32_t overruns = 0;
  uint32_t count = 0;
  uint32_t maxLatency = 0;

  // Começa os interrupts
  adcStart();
  while (1) {
    if (fullHead != fullTail) {   //Se tem algum bloco cheio na fila

      block_t* pBlock = fullQueue[fullTail]; //pega o endereço do bloco
cheio

      // Escreve o bloco no SD.
      uint32_t usec = micros();
      if (!sd.card()->writeData((uint8_t*)pBlock)) {
        error("write data failed");
      }
      usec = micros() - usec;
      t1 = millis();
      if (usec > maxLatency) {
        maxLatency = usec;
      }
      count += pBlock->count;   //A contagem total de itens escritos é
somada com a quantidade de itens escritos no ultimo bloco

      //Adiciona overruns
      if (pBlock->overrun) {
        overruns += pBlock->overrun;

      }
      // Move o bloco para a fila de blocos vazios
      emptyQueue[emptyHead] = pBlock;
      emptyHead = queueNext(emptyHead);
      fullTail = queueNext(fullTail);
      bn++;
      if (bn == FILE_BLOCK_COUNT) {
        // Escreveu o número de blocos definidos, parar interrupções
        adcStop();
        break;
      }
    }
    if (timerError) {
      error("Missed timer event - rate too high");
    }

  }

  if (!sd.card()->writeStop()) {   //Para a escrita
    error("writeStop failed");
```

```
  }

}
//-------------------------------------------------------------------------
-----



//-------------------------------------------------------------------------
-----------------

void setup(void) {

  Serial.begin(9600);


  analogRead(PIN_LIST[0]);  //Faz a primeira conversão AD pra incializar o
ADC (a primeira conversão demora 25 ciclos ADC)

//------------ Inicialização SD -------------

  if (!sd.begin(SD_CS_PIN, SPI_FULL_SPEED)) { //inicializa o cartão SD
    sd.initErrorPrint();
  }
//------------ Inicialização LCD -------------
     disp.begin(16,2);                                  //Inicializa LCD
16 x 2

     disp.setCursor(0,0);                               //Posiciona
cursor na coluna 2, linha 1
     disp.print("1- Treinamento <");                      //Imprime
mensagem
     disp.setCursor(0,1);
     disp.print("2- Diagnostico  ");



}
//-------------------------------------------------------------------------
-----
void loop(void) {
//---------- Leitura dos botões do LCD ------------

    keyboardRead();  //lê teclado

    if(direita == 0x01)                                 //tecla direita
pressionada?
    {
      direita = 0x00;                                   //limpa flag da tecla
    } //end if direita

    if(cima == 0x01)                                    //tecla cima
pressionada?
    {
      cima = 0x00;                                       //limpa flag da tecla
      if (pos)                                           //seta(<) em
diagnóstico, tem como ir pra cima
      {
        pos = 0;                                        //treinamento
```

```
        disp.setCursor(0,0);                          //Posiciona cursor na
coluna 1, linha 1
        disp.print("1- Treinamento <");               //Põe a seta(<) em
treinamento
        disp.setCursor(0,1);
        disp.print("2- Diagnostico  ");
      }

    } //end if cima

    if(baixo == 0x01)                                 //tecla baixo
pressionada?
    {
      baixo = 0x00;                                   //limpa flag da tecla
      if (pos==0)                                    //seta(<) em
treinamento, tem como ir pra baixo
      {
      pos = 1;                                       //diagnóstico
      disp.setCursor(0,0);                           //Posiciona cursor na
coluna 1, linha 1
      disp.print("1- Treinamento  ");                //Põe a seta(<) em
diagnóstico
      disp.setCursor(0,1);
      disp.print("2- Diagnostico <");
      }
    } //end if baixo

    if(esquerda == 0x01)                                 //tecla esquerda
pressionada?
    {
      esquerda = 0x00;                                   //limpa flag da
tecla

    } //end if esquerda

    if(select == 0x01) //select foi pressionado
    {
      select = 0x00;

      if (pos && treinou)     //selecionou diagnostico (pos==1) e já existe
uma carta (treinou==1)
      {
        disp.setCursor(0,0);
        disp.print(" Analisando...  ");
        disp.setCursor(0,1);
        disp.print("X<              ");

        binName[0] = 'd';
        FILE_BLOCK_COUNT = 5;
        while(1){
              keyboardRead();  //lê teclado

              if(select == 0x01) //select foi pressionado
              {
                    select = 0x00;
                    disp.setCursor(0,0);
                    disp.print("1- Treinamento <");
                    disp.setCursor(0,1);
                    disp.print("2- Diagnostico  ");
```

```
                pos = 0;
                delay(1500);
                break;
            }
            logData();
            dumpData(0);
            if(famostras >= 14 && famostras < 50 ){
                if (sd.exists("cavfraca.bin")) {
                   if (!sd.remove("cavfraca.bin")) {
                     error("Can't remove cavfraca file");
                   }
                }
                disp.setCursor(0,0);
                disp.print("   Cavitacao:   ");
                disp.setCursor(0,1);
                disp.print("     Fraca!     ");
                delay(1500);
                disp.setCursor(0,0);
                disp.print(" Analisando...  ");
                disp.setCursor(0,1);
                disp.print("X<              ");
                if (sd.exists(binName)) {
                  if (!binFile.rename(sd.vwd(), "cavfraca.bin")) {
//renomeia o arquivo temporário para o nome definido antes
                    error("Can't rename Cf file");
                  }
                }
            }
            else if (famostras >= 50 && famostras < 200){
                if (sd.exists("cavmedia.bin")) {
                   if (!sd.remove("cavmedia.bin")) {
                     error("Can't remove cavmedia file");
                   }
                }
                disp.setCursor(0,0);
                disp.print("   Cavitacao:   ");
                disp.setCursor(0,1);
                disp.print("     Media!     ");
                delay(1500);
                disp.setCursor(0,0);
                disp.print(" Analisando...  ");
                disp.setCursor(0,1);
                disp.print("X<              ");
                if (sd.exists(binName)) {
                  if (!binFile.rename(sd.vwd(), "cavmedia.bin")) {
//renomeia o arquivo temporário para o nome definido antes
                    error("Can't rename CM file");
                  }
                }
            }
            else if (famostras >= 200){
                if (sd.exists("cavforte.bin")) {
                   if (!sd.remove("cavforte.bin")) {
                     error("Can't remove cavforte file");
                   }
                }
                disp.setCursor(0,0);
                disp.print("   Cavitacao:   ");
                disp.setCursor(0,1);
```

```
                    disp.print("     Forte!     ");
                    delay(1500);
                    disp.setCursor(0,0);
                    disp.print(" Analisando...  ");
                    disp.setCursor(0,1);
                    disp.print("X<              ");
                      if (!binFile.rename(sd.vwd(), "cavforte.bin")) {
 //renomeia o arquivo temporário para o nome definido antes
                        error("Can't rename CF file");
                      }
                }
                 else
                 {
                       if (sd.exists("diadados.bin")) {
                        if (!sd.remove("diadados.bin")) {
                          error("Can't remove diag file");
                        }
                       }
                       if (!binFile.rename(sd.vwd(), "diadados.bin")) {
 //renomeia o arquivo temporário para o nome definido antes
                        error("Can't rename diag file");
                      }
                 }

                }

        }

      else if (pos && treinou == 0)   //selecionou diagnostico e não existe
uma carta (treinou==0)
      {
        disp.setCursor(0,0);
        disp.print(" Nenhum treina- ");         //imprime aviso
        disp.setCursor(0,1);
        disp.print("mento encontrado");
        delay(1500);
        pos = 0;                                //volta display para config
inicial
        disp.setCursor(0,0);
        disp.print("1- Treinamento <");
        disp.setCursor(0,1);
        disp.print("2- Diagnostico  ");
      }
      else if(pos==0)            //selecionou treinamento
      {
        disp.setCursor(0,0);
        disp.print("  Treinando...  ");
        disp.setCursor(0,1);
        disp.print("                ");

        binName[0] = 'b';
        FILE_BLOCK_COUNT = 29;
        if (sd.exists("tredados.bin")) {
          if (!sd.remove("tredados.bin")) {
            error("Can't remove trei file");
          }
        }
        logData();
        dumpData(1);
```

```
        if (!binFile.rename(sd.vwd(), "tredados.bin")) {  //renomeia o
arquivo temporário para o nome definido antes
                        error("Can't rename trei file");
        }

        treinou=1;

        disp.setCursor(0,0);
        disp.print("   Treinamento  ");
        disp.setCursor(0,1);
        disp.print("    efetuado!   ");
        delay(1500);
        disp.setCursor(0,0);
        disp.print("1- Treinamento <");
        disp.setCursor(0,1);
        disp.print("2- Diagnostico  ");
      }
    }
}


void keyboardRead()
{
      ADMUX &= 11110000;                                          //Seta A0
como entrada
      ADCSRA = (1<<ADEN) | (1<<ADPS2) | (1<<ADPS1)| (1<<ADPS0);  //Volta o
prescaler pra 128
      ADCSRA |= (1<<ADSC);                                        //Inicia a
conversão
      while (ADCSRA & (1<<ADSC));                                 //Espera a
conversão acabar
      tecla = ADC;                                                //lê o
valor dos registradores ADCH e ADCL
      ADCSRA = 0;                                                 //limpa
ADCSRA


     // --- Testa se os botões foram pressionados ---
     // Se foi pressionado, seta a respectiva flag
     if      (tecla <  50)                    butt01 = 0x01; //direita
     else if (tecla > 103 && tecla < 200) butt02 = 0x01;  //cima
     else if (tecla > 250 && tecla < 380) butt03 = 0x01;  //baixo
     else if (tecla > 450 && tecla < 550) butt04 = 0x01;  //esquerda
     else if (tecla > 700 && tecla < 800) butt05 = 0x01;  //select



     // --- Testa se os botões foram liberados ---
     //
     if (tecla > 50 && butt01)             //Botão foi solto e a flag ta
setada?
     {
         butt01 = 0x00;                          //Limpa flag butt01
         direita  = 0x01;                        //Seta flag direita

     } //end direita
     if (tecla > 200 && butt02)
     {
         butt02 = 0x00;
```

```
            cima     = 0x01;

        } //end cima
        if (tecla > 380 && butt03)
        {
            butt03 = 0x00;
            baixo   = 0x01;

        } //end baixo
        if (tecla > 550 && butt04)
        {
            butt04 = 0x00;
            esquerda   = 0x01;

        } //end esquerda
        if (tecla > 800 && butt05)
        {
            butt05 = 0x00;
            select = 0x01;
        } //end select


} //end keyboardRead
```