UNIVERSIDADE DE CAXIAS DO SUL

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

AUTOMAÇÃO DE ENSAIOS POR FLEXÃO DE CABOS ELÉTRICOS E ANÁLISE DA VIDA ÚTIL DOS CONDUTORES

Relatório de Trabalho de Conclusão de Curso, apresentado como requisito parcial à conclusão do curso de Engenharia de Controle e Automação
Orientador: Prof.Dr.StanislavTairov.

LEONARDO BORTOLUZ

CAXIAS DO SUL, 2015

LEONARDO BORTOLUZ

Aprovado (a) em ____/_____/_____

Banca Examinadora:

Presidente

Prof. (Titulação - Esp. Me.ou Dr. – e nome do orientador)
Universidade de Caxias do Sul - UCS

Examinadores:

Prof. (Titulação - Esp. Me.ou Dr. – e nome do professor examinador)
Universidade de Caxias do Sul - UCS

Prof. (Titulação - Esp. Me.ou Dr. – e nome do professor examinador)
Universidade de Caxias do Sul - UCS

AGRADECIMENTO

Aos meus pais, Adelina e Valdomiro, e a minha irmã, Luciana, pelo apoio, motivação e paciência concedidos a mim desde sempre, indispensáveis para chegar até aqui.

Ao professor orientador Dr. Stanislav Tairov pela disponibilidade e conhecimento aos quais a mim foram destinados ao longo da graduação, principalmente durante as orientações nos trabalhos de estágio e TCC.

Ao amigo Thiago Cândido, pelas aulas de programação em C# e SQL, oferecidas voluntariamente e essenciais a conclusão desse trabalho.

Ao engenheiro Valcir Paulo Rossetto Júnior, pelo auxílio durante as análises finais desse projeto.

Aos colegas e amigos de curso pela companhia e estudos em conjunto, dentre os quais são destacados: Alex Garbin Pereira, Diego Alex Marangon, Gustavo Henrique Soares, Henrique Maino, Jéssica Carissimi, Marcos Borsatti, Márcio Dall Amaria, Michel Pereira e Nathália Grazziotin

Aos professores que proporcionaram conhecimento durante todo o curso.

A todos que, de alguma forma, auxiliaram no progresso, não apenas nesse trabalho, mas em toda a minha vida acadêmica.

RESUMO

Este trabalho apresenta a automação de ensaios de condutores elétricos submetidos em flexão. Esses ensaios são utilizados pela engenharia da empresa Palfinger para a análise da vida útil dos cabos móveis empregados em seus produtos.

A automação é efetuada através da comunicação entre o controlador Arduino da máquina de testes com um PC, via rede local. Busca-se obter a supervisão remota dos ensaios, o armazenamento das informações através de banco de dados e o processamento dos resultados via PC.

É demonstrada a escolha do modelo Ethernet e dos protocolos TCP/IP que compõem o sistema de transferência de dados, bem como o *software* do Arduino responsável em integrar o controlador com a rede. Também é apresentado o aplicativo executado pelo PC e responsável pela interface gráfica, sendo desenvolvido em linguagem C# via ambiente Visual Studio.

Através da automação desenvolvida, realiza-se uma análise sobre a vida útil de cabos elétricos móveis aplicados em produtos da empresa. Como resultado, se obtêm os principais fatores que influenciam na fadiga de condutores submetidos em flexão, oferecendo uma base para futuros estudos a serem desenvolvidos pela engenharia da Palfinger, visando obter produtos com melhor qualidade.

Palavras-chaves: Automação. Testes de flexão. Arduino. C#.

ABSTRACT

This work shows the automation of electrical conductors tests submitted in bending. These tests are used by engineering Palfinger company to analyze the life of the moving cables in their products.

Automation is done through communication between the Arduino controller testing machine with a PC via LAN. It seeks to remote supervising of tests, the storage of information through database and processing of results via PC.

It is demonstrated that the choice of model Ethernet and TCP / IP protocols that compose the data transfer system and the Arduino software responsible to integrate the controller to the network. The application executed by the PC and responsible for the graphical user interface and is developed in in C # via Visual Studio environment also appears.

Through the developed automation, is realized an analysis of the life of movable electric cables applied in the company's products. As a result, we obtain the main factors that influence the fatigue submitted conductors flexed, providing a basis for future studies to be developed by the Engineering Palfinger, to obtain products with better quality.

Keywords: Automation. Tests of bending. Arduino. C #.

LISTAS DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

DDL	Linguagem de Definição de Dados
DML	Linguagem de Manipulação de Dados
DNS	*Domain Name Server*
DOE	*Design of Experiments*
EEPROM	*Electrically Erasable Programmable Read-Only Memory*
FTP	File Transfer Protocol
GUI	Interface Gráfica do Usuário
HTTP	*Hypertext Transfer Protocol*
ICSP	*In-Circuit Serial Programming*
IDE	*Integrated Development Environment*
IHM	Interface Homem-Máquina
IP	*Internet Protocol*
LCD	*Liquid Crystal Display*
MAC	*Media Acess Control*
PC	*Personal Computer*
PHY	*Physical Layer*
RJ-45	Registered Jack - 45
SD	*Secure Digital Card*
SPI	*Serial Peripheral Interface*
SQL	*Structured Query Language*
SRAM	*Static Random Access Memory*
TCP	*Transmission Control Protocol*
UART	*Universal Asynchronous Receiver/Transmitter*
USB	*Universal Serial Bus*
WI-FI	*Wireless Fidelity*

ÍNDICE

CAPÍTULO 1 - INTRODUÇÃO

Em produtos como guindastes e elevadores veiculares, é exigida a existência de cabos elétricos móveis responsáveis pela alimentação e comunicação dos elementos elétricos que sofrem deslocamentos transversais. Por essa característica, o projeto deve seguir rigorosos estudos sofre a interferência da movimentação no sistema, devido o surgimento de desgastes por flexão dos condutores aplicados. Essas fadigas são influenciadas por inúmeros fatores (secção dos fios, raio de curvatura do cabo, etc.) e, na maioria das vezes, tornam-se as características determinantes da vida útil do componente (Bortoluz, 2015).

Como ambos os produtos são destinados para o uso automotivo, durante o desenvolvimento de seus sistemas elétricos, é necessário seguir normas que auxiliem na aquisição de uma boa durabilidade no produto final. Essas normas especificam as principais características que os condutores elétricos devem possuir, tanto em seu processo de fabricação, quanto na aplicação ao projeto, informando dados importantes sobre as suas condições de uso (LS, 2005).

Durante o dimensionamento dos condutores a serem utilizados nesses projetos elétricos, primeiramente se busca atender os limites máximos de uso, como tensão, corrente e temperatura, e os limites físicos determinados pelo espaço dentro do equipamento. Essas características podem contribuir na origem de conflitos entre os elementos que influenciam na quebra dos condutores em flexão. Portanto, se evidencia a importância de análises que permitam conciliar determinados fatores, buscando construir um sistema que se enquadre à aplicação e que seja resistente à movimentação, garantindo a maior vida útil possível.

Com isso, visando tornar possível essa análise, é essencial que a engenharia desenvolva ensaios que simulem o funcionamento do equipamento com distintas configurações ou influências. Esses testes possibilitam, de maneira rápida, a coleta de informações sobre o comportamento desses sistemas ao longo do tempo, identificando a durabilidade do produto desenvolvido.

1.1 JUSTIFICATIVA DO TRABALHO

Sistemas elétricos utilizados em partes móveis de equipamentos, como guindastes e elevadores veiculares, necessitam de configurações especiais para que os mesmos sejam distribuídos corretamente durante a sua movimentação. De modo geral, a configuração destinada para essas aplicações corresponde na distribuição dos condutores em forma de “U” e organizados dentro de calhas ou esteiras porta-cabos (Figura 1), componentes responsáveis por proteger e guiar os condutores durante o seu deslocamento transversal.

Figura 1- Exemplo de aplicação de cabos móveis

Fonte: IGUS (2015)

Atuante no setor de produtos para transportes de cargas, a empresa Palfinger desenvolveu uma máquina eletropneumática de ensaios de flexão, a qual possuía operação altamente manual. A função desse equipamento é simular a fadiga por flexão dos chicotes elétricos móveis utilizados em seus produtos.

A automação desse equipamento foi desenvolvida durante a disciplina de Estágio em Engenharia de Controle e Automação, conforme Bortoluz (2015). O trabalho consistiu na implementação em nível de máquina, baseando-se em uma plataforma microcontrolada Arduino (Arduino, 2015). Dentre as melhorias proporcionadas com a automação, destaca-se a detecção de ruptura, que anteriormente era executada manualmente. Esse processo passou a ser realizado

pelo controlador, armazenando os dados sobre o rompimento dos cabos em um cartão de memória SD (*Secure Digital Card*). O controlador também contém um *display* responsável por informar importantes dados dos ensaios em execução, tais como a quantidade de ciclos realizadas e o total de condutores rompidos.

Apesar da automação se demonstrar bastante eficiente, a máquina ainda possui diversas oportunidades de melhorias. A primeira encontra-se na transmissão de dados da máquina para um PC (*Personal Computer*) desenvolvido através da automação em alto nível. Esse transporte ocorre via cartão SD, exigindo que seja efetuado manualmente pelo usuário, tornando-se um processo dependente da ação humana e impossibilitando o acompanhamento remoto dos ensaios enquanto ocorrem.

O próximo ponto se define na ausência de um *software* de alto nível de programação, responsável pelo processamento dos dados provenientes dos ensaios. Isso exige que o usuário execute tarefas, dentre as quais, cálculos estatísticos (média, desvio padrão, etc.) e levantamento dos resultados por meio de diversos *softwares*. Além de demandar elevado tempo de engenharia, a falha humana aumenta as incertezas nos resultados obtidos.

Com o aparecimento de novos produtos e projetos de chicotes, surge a necessidade da realização de ensaios e estudos referente à vida útil dos condutores submetidos em flexão. Portanto, se evidencia a importância do desenvolvimento de melhorias na comunicação e processamento dos dados proveniente da máquina de ensaios.

1.2 OBJETIVOS

1.2.1 Objetivo geral

Desenvolver sistema de transferência de dados entre controlador da máquina de flexão de cabos elétricos (primeiro nível) e PC (segundo nível) via rede local.

1.2.2 Objetivos específicos

- Supervisão remota dos testes via computador;
- Proporcionar, através de recursos de PC, processamento e análise sobre a vida útil dos condutores em flexão.
- Desenvolver estudo sobre vida útil dos condutores em diferentes configurações.

1.3 ÁREA DE TRABALHO

A comunicação entre a máquina e a rede local e a elaboração do *software* de processamento dos dados serão efetuadas em um laboratório computacional com acesso à rede local e a programas necessários para a tarefa.

A análise sobre a vida útil dos chicotes elétricos será desempenhada através de dados coletados da máquina de ensaios de flexão. Essa máquina está localizada dentro do pavilhão fabril da empresa Palfinger, em uma sala fechada utilizada pela engenharia para construção de protótipos e ensaios de componentes utilizados nos produtos.

Atualmente o grupo Palfinger, de origem austríaca, atua na produção de equipamentos para transportes de cargas (guindastes) e pessoas (elevadores veiculares), empregando cerca de 400 funcionários na planta de Caxias do Sul e mais de 6 mil em 29 unidades fabris pelo mundo (Palfinger, 2015).

1.4 LIMITES DO TRABALHO

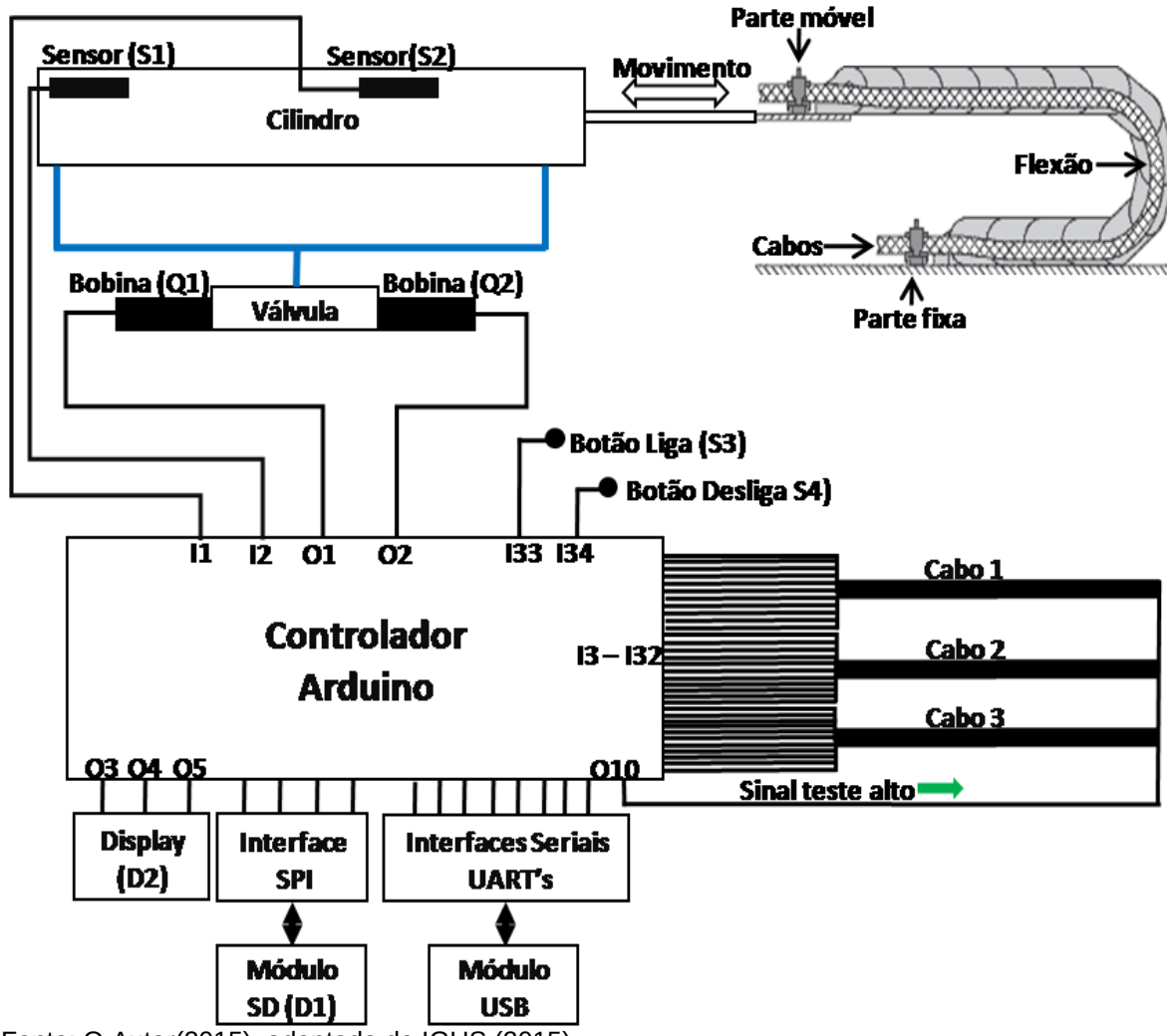
A etapa de desenvolvimento da comunicação em rede local se restringirá na atuação apenas em laboratório computacional. A ausência momentânea de pontos de acesso à rede, no ambiente onde se localiza a máquina, impede a efetivação da comunicação. No momento que ocorrer a implantação de acesso pela empresa, basta executar as alterações desenvolvidas nesse trabalho, as quais permitirão a interface proposta.

CAPÍTULO 2 - DESENVOLVIMENTO

2.1 SISTEMA ATUAL DE ENSAIOS

Durante a disciplina de Estágio em Engenharia de Controle e Automação da Universidade de Caxias do Sul (UCS), foi desenvolvido a automação da máquina de testes de flexão utilizada pela engenharia da empresa Palfinger e disponível em Bortoluz (2015). A Figura 2 mostra o diagrama dos principais elementos que compõe o sistema implantado.

Figura 2- Diagrama atual do sistema de testes



Fonte: O Autor(2015), adaptado de IGUS (2015)

Com processamento centralizado em uma plataforma microcontrolada modelo Arduino Mega 2560 (Apêndice A), a máquina exerce a movimentação dos

cabos elétricos e a detecção da ruptura de seus condutores. O equipamento comporta o máximo de 30 fios por ensaio, com comprimento de 185 mm a 800 mm.

O sistema é inicializado por meio dos botões S3 e S4 que habilitam ou desabilitam os testes. Quando pressionados simultaneamente, possibilitam o reset dos parâmetros e início de um novo ensaio.

Com a habilitação, o controlador aciona as bobinas Q1 e Q2, avançando e recuando o cilindro pneumático através da posição detectada pelos sensores eletromagnéticos S1 e S2. Devido ao fato dos cabos estarem fixados entre a haste do cilindro (parte móvel) e a base da máquina (parte fixa), esse deslocamento ocasiona aos condutores flexões idênticas quando aplicados aos produtos. A distância entre ambas as partes é regulável, possibilitando a alteração do raio de curvatura do chicote em teste.

Além de proporcionar o recuo do cilindro, a sub-rotina de testes de ruptura é habilitada, através de interrupção, pelo sensor S2. O controlador envia um sinal alto pela saída O10 e verifica o retorno nas entradas I3 a I32. Havendo a detecção de ruptura, os dados referentes ao cabo, fio e ciclo em que houve a quebra são armazenados em cartão de memória SD (D1), onde o módulo se comunica com o Arduino através da interface SPI (*Serial Peripheral Interface*).

O sistema também possui um display LCD (*Liquid Crystal Display*) gráfico D2, que fornece ao usuário informações dos ensaios em andamento, como número do teste, quantidade de ciclos e quantidade de fios rompidos até o momento. Ocorrendo a quebra em todos os condutores, a máquina é desabilitada.

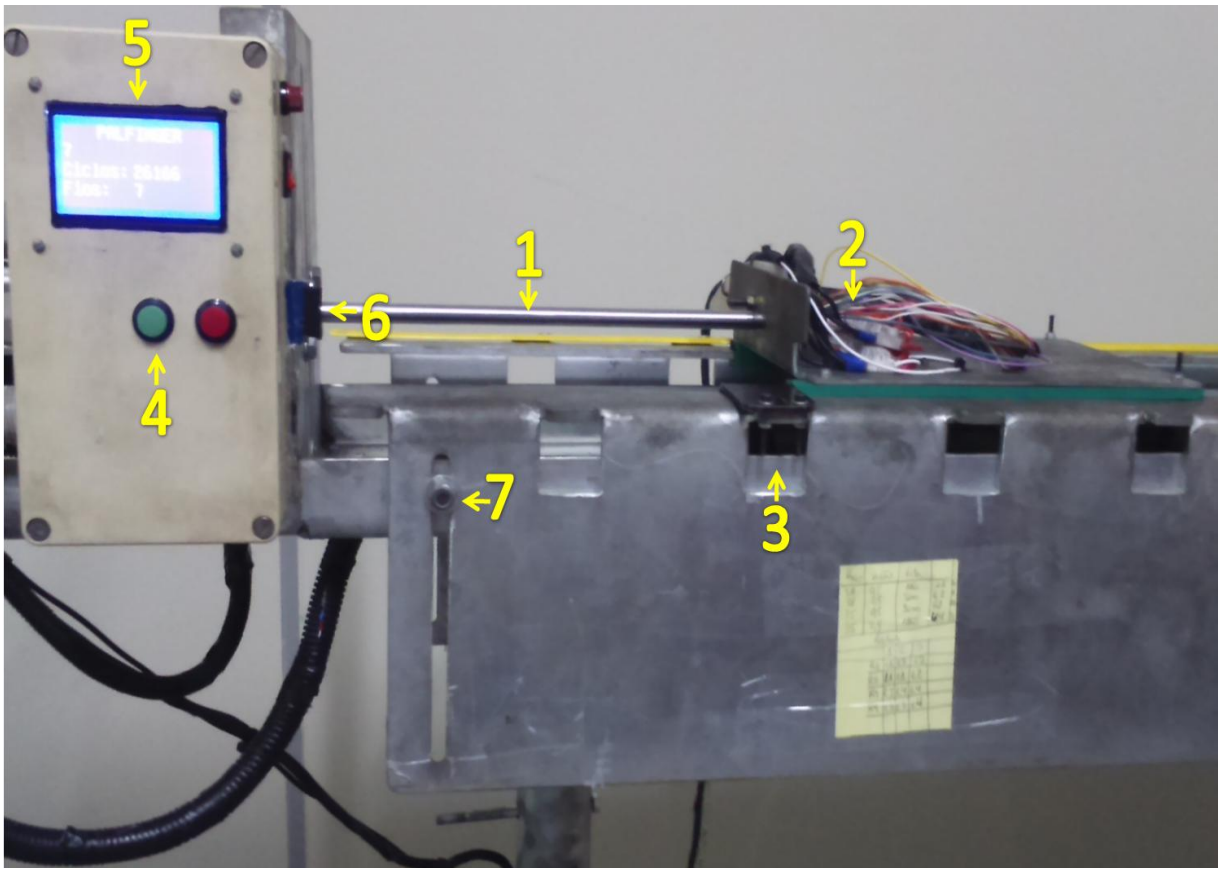
Atualmente, o único método de comunicação com um nível de processamento por PC é efetuado pelo usuário através da transferência manual do cartão SD. O Arduino permite a implementação de métodos melhores para o transporte de dados através da utilização de módulos específicos, os quais utilizam as interfaces representadas na Figura 2.

Conforme Arduino (2015), a UART (*Universal Asynchronous Receiver/Transmitter*) é majoritariamente utilizada para a depuração de softwares entre o microprocessador e o computador via módulo integrado USB (*Universal Serial Bus*). Já a SPI é um padrão utilizado em diversos módulos externos, dentre os quais o de memória SD empregado no sistema atual.

Na Figura 3 é apresentada a imagem da máquina física desenvolvida conforme o diagrama da Figura 2.

Figura 3- Sistema de testes atual.

Legenda: cilindro pneumático (1), estrutura móvel (2), base fixa (3), botões de comando(4), bloco de controle Arduino com display LCD (5), módulo de memória SD (6), regulagem do raio dos chicotes (7).



Fonte: Palfinger (2015)

As interfaces aqui apresentadas permitem o desenvolvimento posterior do sistema de comunicação e processamento de dados conforme proposto.

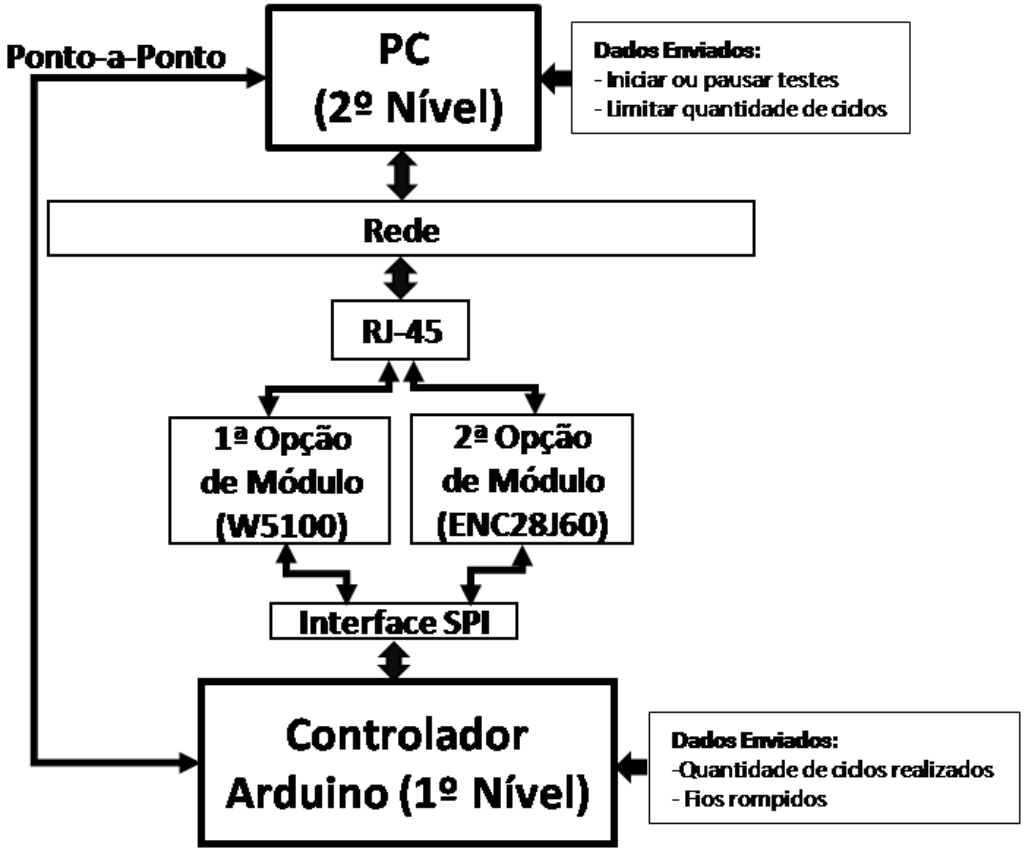
2.2 SISTEMA PROPOSTO PARA TRANSFERÊNCIA DE DADOS

Conforme Rosário (2009), a inclusão de sistemas de comunicação no desenvolvimento de automações é de grande importância, pois oferecem vantagens como agilidade e facilidade no acompanhamento de processos e redução de custos de instalação e operação dos equipamentos.

A Figura 4 apresenta o diagrama do sistema de transferência de dados proposto, buscando permitir a supervisão remota da máquina de ensaios. O diagrama é composto por um esboço dos métodos de comunicação entre os níveis e

as mensagens que se deseja transmitir. O Arduino será responsável por gerar as informações dos ensaios em andamento e receber os comandos do PC, o qual exerce a interface com o usuário. Esse fluxo bi-direcional exige uma comunicação *full* ou *half duplex*.

Figura 4- Diagrama do Sistema de transferência de dados



Fonte: o Autor (2015)

Segundo Langbridge (2015), o principal método de comunicação do controlador Arduino com níveis superiores é realizada através de um módulo USB, integrado à plataforma via conexão UART, o que desenvolve uma ligação ponto a ponto com o PC. Essa interface é utilizada para envio ou recebimento de dados seriais através da própria IDE (Ambiente de Desenvolvimento Integrado) do Arduino.

Apesar de não exigir a adição de módulos ao controlador, a necessidade de desenvolvimento de um canal próprio que possibilite a transmissão de dados via ponto a ponto descarta a aplicação desse sistema ao projeto. Sendo assim, é definido utilizar o método via rede para comunicação entre os níveis.

2.2.1 Definição da rede

Em Lamb (2015) é apresentado diversos tipos de redes aplicados em automações, onde destacam-se a Ethernet, o Wireless e padrões destinados para o uso industrial como Device, CANopen, PROFIBUS e o grupo FieldBus, este último desenvolvido com foco no controle em tempo real.

Dentre esses recursos, o controlador Arduino permite apenas aplicações através das redes Ethernet ou Wireless (Wi-Fi - *Wireless Fidelity*, Bluetooth ou ZigBee), porém exige a adição de módulos específicos e o desenvolvimento de *softwares* para a interface com o usuário.

Portanto, foi escolhido trabalhar com o padrão de rede Ethernet devido as vantagens, sobre os demais, demonstradas abaixo:

- A empresa possui redes Ethernet no ambiente fabril próximo ao local onde se encontra a máquina de ensaios de flexão;
- Existência de módulos que permitem a integração da Ethernet ao controlador da máquina;
- Flexibilidade na alteração do destino dos dados, devido a Ethernet permitir a conexão multiponto resultante da sua topologia por barramento e métodos utilizados na transferência das mensagens (Fourouzan, 2008);
- A posição do controlador na máquina é fixo, possibilitando o acesso à rede por meio físico (cabo);
- A comunicação via cabo Ethernet é menos suscetível às interferências que a *wireless,* visto que a máquina encontra-se em ambiente fabril.

Conforme estruturado na Figura 4, para a aplicação da rede desejada são disponibilizados dois módulos compatíveis com o Arduino: o módulo W5100 (1ª opção), produzido pela Wiznet, e o módulo Ethernet ENC28J60 (2ª opção), produzido pela Microchip. Ambos recursos desenvolvem a comunicação através de um sistema próprio, com compatibilidade aos protocolos TCP/IP (*Transmission Control Protocol* / *Internet Protocol*), transmissão *full* ou *half-duplex* e clock de 25MHz. A conexão com o Arduino é realizada através da interface SPI e a conexão com a rede por meio do conector RJ-45 (*Registered Jack* - 45).

Como principais diferenças entre cada opção, segundo Wizinet (2015), o W5100 implementa a camada física (PHY), o controle de acesso à mídia (MAC) e as camadas de rede e transporte, com *buffer* de transmissão de 16KBytes e velocidades de 10 ou 100 Mbps. Além disso, esse componente possui um sistema integrado de memória SD. O módulo ENC28J60, conforme Microchip (2012), implementa apenas o PHY e o MAC da rede e realiza tratamentos de colisões aos pacotes enviados, com buffer de 8KBytes e velocidade de 10Mbps. Demais atuações como pilha de dados (*stack*) são desenvolvidas na memória do próprio Arduino.

Como vantagens do módulo ENC28J60, temos que o mesmo atende aos requisitos de memória do projeto, visto que o controlador possui velocidade de processamento otimizada (utilização de interrupções), com custo de R$ 35,90, enquanto o da Wizinet é de R$ 69,90 (FilipeFlop, 2015). Apesar disso, é escolhido trabalhar com o módulo W5100, tendo como principal fator de escolha a existência da memória SD integrada. A função da memória SD é realizar o armazenamento local dos dados permitindo um *backup* caso ocorra problemas na comunicação com a rede.

Como o Arduino possui apenas uma interface SPI, o desenvolvimento do sistema de comunicação exige a substituição do módulo de memória SD pelo módulo de rede Ethernet, visto que ambos utilizam a mesma conexão. Como o módulo W5100 possui a memória SD integrada, é possível implementar ambos os sistemas através do mesmo hardware, comutando as operações via *software*.

Definida a utilização do módulo W5100 ao projeto, desenvolve-se a rede Ethernet através da aplicação dos protocolos TCP/IP. É definido seguir esse grupo de protocolos devido a sua ampla utilização em sistemas de comunicação e a compatibilidade com o módulo escolhido.

2.2.2 Os protocolos TCP/IP

Em Farrel (2005), protocolos são definidos por um conjunto de regras que capacitam a comunicação entre sistemas distintos, definindo as suas operações na rede. Por sua vez, o TCP/IP é um conjunto de protocolos divididos em quatro camadas distintas (Tabela 1). Essas camadas organizam os protocolos conforme as

suas funções e ordem de atuação durante a transmissão e recepção dos dados da rede.

Tabela 1- Camadas TCP/IP

Camada	Modelo TCP/IP	Protocolo
4	Aplicação	TELNET
3	Transporte	TCP
2	Internet	IP
1	Interface de Rede	Ethernet

Fonte: Ross (2008)

2.2.2.1. Primeira Camada - Interface de Rede

Segundo Tanembaum (2003), a primeira camada desenvolve a interface entre o controlador da máquina e a rede, permitindo o tráfego independente dos dados até o destino. Essa camada pode ser dividida em nível físico e nível de enlace.

A aplicação física é implementada através do módulo W5100 via cabo 100Base-T (padrão IEEE 802.3), recurso compatível com o sistema, limitando velocidade dos dados na rede em 100Mps. Esse padrão atende ao sistema de comunicação proposto, visto que as mensagens enviadas (caracteres e variáveis numéricas conforme a Figura 4) são composta por valores inferiores a 16 bits e o tempo máximo para a transmissão é de três segundos, menor tempo entre o acionamento das interrupções do controlador durante os ensaios.

No nível de enlace é implementada a Ethernet, arquitetura escolhida e responsável em segmentar a mensagem em quadros (frames), gerenciar a disponibilidade e envio de dados no canal de comunicação e aplicar o indicador MAC. Esse indicador atribui um número único ao controlador Arduino, permitindo o endereçamento dos dados na rede (Spurgeon, 2000). O MAC é implementado via *software* do controlador em conjunto com o módulo W5100.

2.2.2.2. Segunda Camada - Internet

A camada de internet atua no roteamento das mensagens na rede (caminho que a informação deve seguir), sendo responsável pela identificação correta do

destino dos dados e controle de congestionamentos (Tanembaum, 2003). O protocolo empregado será o IP, compatível ao módulo W5100 e responsável pelo endereçamento do controlador. A versão utilizada é a IPv4, que permite endereços de até 32 bits.

Conforme Farrel (2005), o objetivo do IP é definir a identidade de cada componente conectado através da sua codificação por meio de quatro octetos. Com isso, é definido um número único que representa a rede e o host do dispositivo, possibilitando que mensagens cheguem até ele. O IP utilizado nas redes Ethernet da empresa são de classe A, ou seja, o primeiro octeto simboliza o código da rede e os três restantes o terminal. Como exemplo empregado, temos que o número IP do computador localizado no setor da engenharia é 10.70.18.100 e o IP do Arduino é definido, via software, como 10.70.18.156. O último octeto é modificado por um valor de IP inexistente, permitindo que o controlador faça parte da mesma rede, porém sendo um terminal único.

2.2.2.3. Terceira Camada - Transporte

Em Farrel (2005), a camada de transporte é definida pela função de certificar a qualidade das mensagens e reordenação dos pacotes recebidos no destino. Os principais protocolos que atuam nessa área são o TCP e o UDP (*User Datagram Protocol*). Esses protocolos utilizam o conceito de portas, onde as informações são distribuídas em diferentes acessos, de acordo com o seu tipo de serviço. Essa atuação possibilita a múltipla comunicação do dispositivo a vários servidores.

Kretcheu (2013) descreve que o protocolo UDP não verifica a integridade das mensagens recebidas pelo destino. Essa característica o torna um protocolo pouco confiável, porém de rápido processamento com possibilidade de envios simultâneos de mensagens, devido à inexistência de retransmissões. É utilizado quando a qualidade dos arquivos não compromete o sistema, como em transferências de áudios ou vídeos, não aplicável a esse projeto.

O TCP, protocolo orientado por conexão, atua verificando a integridade dos arquivos recebidos e solicitando a sua retransmissão, caso os mesmos tenham sidos danificados ou perdidos. Esse protocolo se identifica pela sua alta confiabilidade na troca de informações, porém com um custo elevado de processamento (Kretcheu, 2013). Como o sistema de comunicação efetua a transferência de importantes

informações com baixa exigência em velocidade, é definido o TCP como protocolo da camada de transporte. Esse protocolo é implementado por bibliotecas do Arduino que atuam sobre o módulo de Ethernet e por classes específicas utilizadas durante o desenvolvimento do aplicativo em alto nível.

2.2.2.4. Quarta Camada - Aplicação

De acordo com Murhammer (2000), a última camada determina a interface entre a rede de comunicação de dados com o aplicativo de alto nível, o qual realiza o consumo das informações transferidas e a interface com o usuário. Nessa camada são disponíveis protocolos para diferentes funções, como acesso remoto (TELNET), transferência de arquivos (FTP - *File Transfer Protocol*), resolução de nomes de acesso (DNS - *Domain Name Server*) ou distribuição de hipermídia (HTTP - *Hypertext Transfer Protocol*).

Como o sistema de comunicação proposto não exige o acesso a Internet ou operações com transferência de arquivos ou *e-mails*, o protocolo utilizado na camada de aplicação é semelhante ao TELNET, o qual permite o acesso remoto a máquina. A interface com o aplicativo é desenvolvida com base na comunicação entre servidor e cliente. O servidor (Arduino) aguarda a conexão de um cliente (PC) para que seja efetivada a rede de dados. A implementação dessa etapa ocorre durante o desenvolvimento dos *softwares* presentes no controlador e no aplicativo de processamento de dados.

2.2.3 Efetivação do sistema de transferência de dados

Durante a primeira fase de automação dos ensaios de flexão foi desenvolvido o controle dos ensaios e armazenamento dos dados via memória SD em nível de máquina (Bortoluz, 2015). Como a automação proposta implementa a comunicação entre o controlador da máquina e o PC, é necessário realizar adaptações no *hardware* e *software* em que atribui-se ao sistema de interface com o usuário e transferência de dados, permitindo a supervisão e processamento das informações via rede.

2.2.3.1. Adaptação do Hardware para conexão com a rede

Como citado anteriormente, o Arduino Mega 2560 possui apenas uma interface SPI para a comunicação com recursos externos, utilizado atualmente no armazenamento de dados via cartão de memória SD. Para o desenvolvimento da rede proposta, é necessária a sua substituição pelo módulo W5100, o qual utiliza a mesma interface com o Arduino e implementa a comunicação via Ethernet, mantendo o armazenamento via memória SD.

Devido ao módulo W5100 exigir do Arduino o controle sobre a comutação entre a rede Ethernet e a memória SD, é necessário alterar as ligações de dois pinos digitais do controlador, que são responsáveis por essa função. A ligação física do módulo Ethernet ao controlador Arduino é apresentada no Apêndice B.

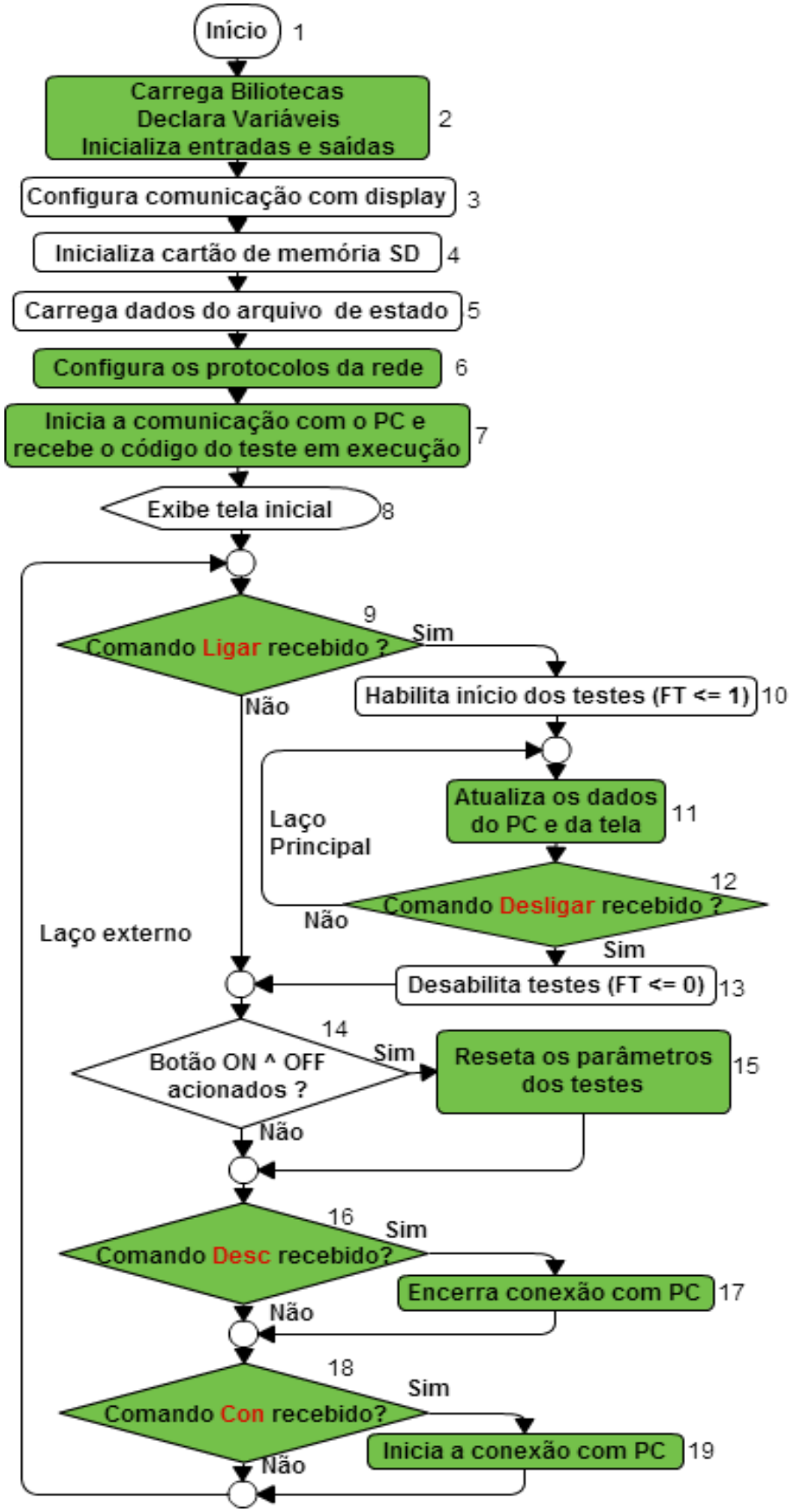
2.2.3.2. Modificação do software para comunicação com a rede

O *software* do controlador da máquina é responsável por efetuar os testes de flexão e o armazenamento local dos dados dos ensaios (cartão SD). Devido a implementação da rede Ethernet e a integração com o segundo nível de processamento, é necessário o desenvolvimento de um software para o Arduino, baseado em primeiro nível de programação, o qual permite o acesso à rede local e a supervisão remota dos ensaios.

Para a construção desse programa foi utilizada a linguagem própria do controlador (semelhante a linguagem C) e a IDE livre disponibilizada pelo próprio Arduino. No apêndice D é apresentado o código desenvolvido, onde as linhas cinzas simbolizam as partes modificadas nesse trabalho.

Na Figura 5 é demonstrado o fluxograma do *software* desenvolvido, onde os blocos de cor verde simbolizam as funções modificadas ou inseridas nesse projeto.

Figura 5- Fluxograma principal do software do controlador



Fonte: o Autor (2015)

O *software* inicia-se com a declaração de variáveis, definição e inicialização das portas do controlador e carregamento das bibliotecas utilizadas (bloco 2) e descritas na Tabela 2. É necessária a inclusão da biblioteca que opera o módulo Ethernet W5100, a declaração de variáveis que implementam os protocolos de rede (endereço MAC e IP) e de um servidor vinculado a uma porta de conexão. O número da porta deve ser entre 0 e 65535, portanto utiliza-se o valor 5555, visto que números inferiores a 1024 geralmente são utilizados para outros processos na rede.

Tabela 2- Bibliotecas utilizadas

Nome	Função
U8glib.h	Configura comandos para utilização de *displays* gráficos LCD com 8 bits.
Ethernet.h	Implementa a comunicação Ethernet e seus protocolos com o módulo W5100.
SD.h	Implementa comandos utilizados pela memória SD.
SPI.h	Configura a comunicação mestre-escravo (SPI) do Arduino para os módulo de Ethernet e memória SD.

Fonte: Truchsess(2014), Arduino (2015).

Nos blocos 3 a 5 ocorre a inicialização do display LCD e a configuração e carregamento dos dados presentes na memória SD. Esse processo possibilita com que os ensaios sejam pausados e retornados, sem ocorrer a perda de informações obtidas até o momento. O arquivo que armazena é denominado como arquivo de estado.

O bloco 6 configura a comunicação com a rede Ethernet, sendo implementado o protocolo IP e o endereço MAC. Após, o controlador é habilitado como servidor e aguarda o acesso do PC (bloco 7). Com o acesso, o PC envia um número que identifica o teste em execução e a quantidade máxima de ciclos a serem realizados. Todos os dados transferidos entre controlador e PC carregam um caractere que identifica o tipo de informação.

Após a exibição da quantidade de ciclos e fios rompidos na tela principal (bloco 8), o programa entra no laço externo, o qual opera em ciclo a espera de comandos enviados pelo usuário para o controlador.

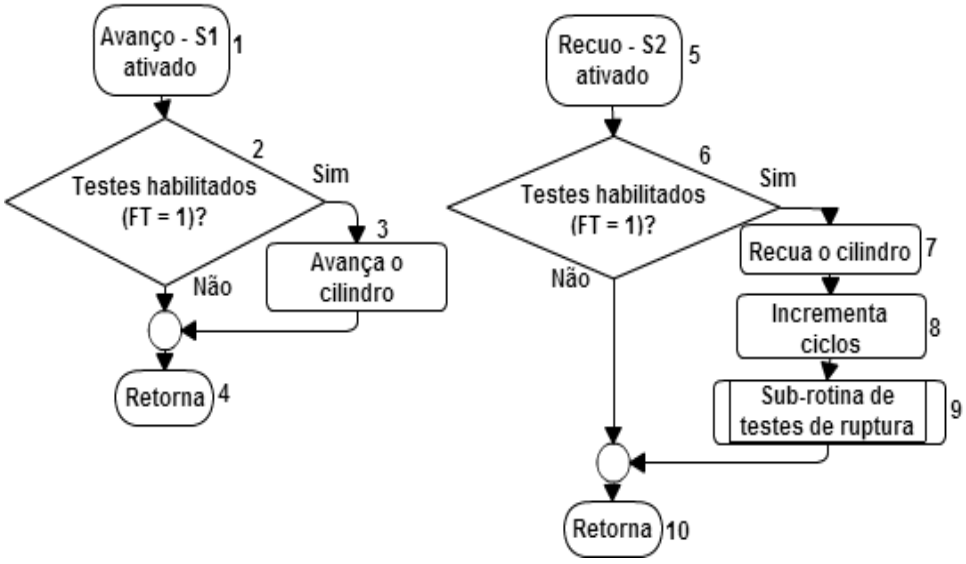
Os testes são habilitados através do comando "*Ligar"*, originado por meio do botão S3 ou remotamente via PC e rede local. Com a habilitação, o software entra no laço principal que envia dados de quantidade de ciclos executados para o *display* e PC, sempre quando houver atualizações (bloco 11). Nesse laço também ocorre a

habilitação do sistema de interrupções, que realizam as movimentações de avanço e recuo do cilindro e execução dos testes de rupturas. Caso o comando "*Desligar*" for detectado (bloco 12), os testes são desabilitados (bloco 13) e retornado para o laço externo. Esse comando é enviado para a máquina via botão S4 ou remotamente pelo PC.

O *reset* dos parâmetros dos ensaios (blocos 14 e 15) simboliza o início de um novo teste e é executado através do acionamento simultâneo dos botões S3 e S4 ou pelo PC. Com a realização do reset, o número do teste é incrementado (via botões) ou atribuído (via PC) e as suas variáveis de testes são zeradas. Os blocos 16 a 19 gerenciam o acesso à rede local, permitindo que o operador realize os ensaios com ou sem a comunicação com o PC.

A Figura 6 apresenta as sub-rotinas de interrupções, responsáveis pela movimentação do cilindro (avanço e recuo), as quais não sofreram alterações no programa atual. A sub-rotina de avanço (blocos 1 a 4) atua quando S1 é ativado, e realiza a operação de avanço do cilindro mediante a habilitação dos testes (comando "*Ligar"* recebido). A sub-rotina de recuo atua quando S2 é ativado (bloco 6 a 10), onde, com o *comando "Ligar"* ativo, o cilindro recua, o contador de ciclos é incrementado e os testes de ruptura nos condutores são executados por meio de uma sub-rotina.

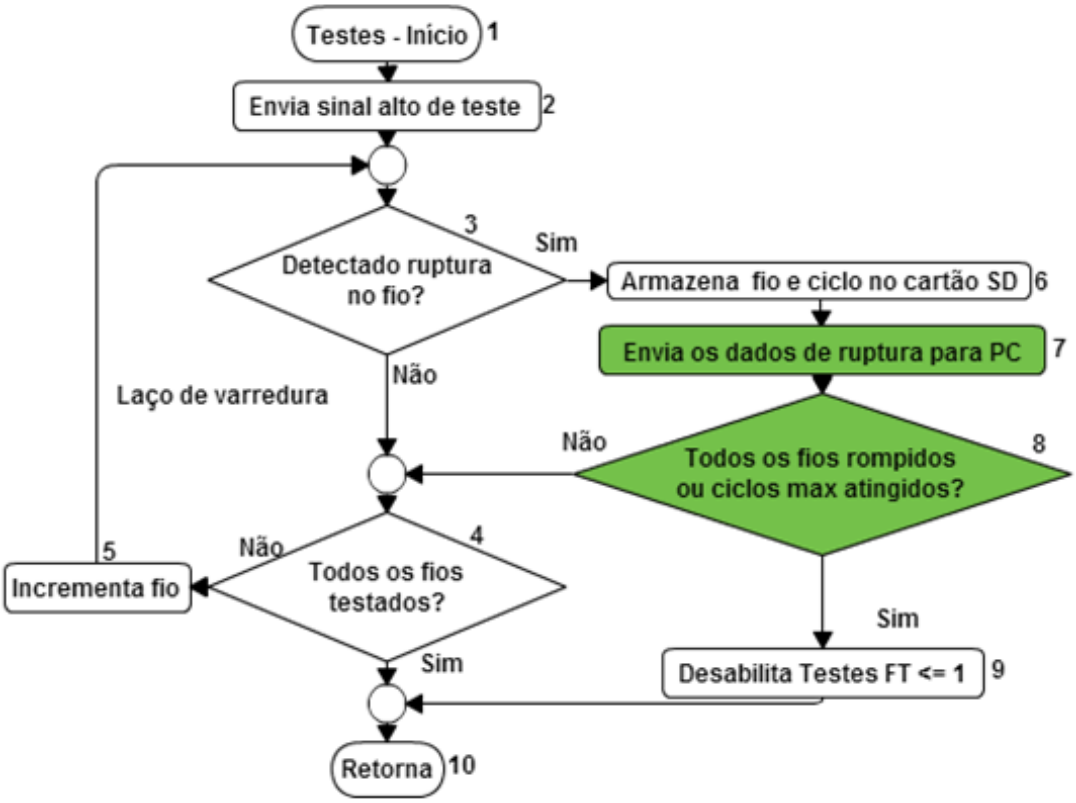
Figura 6- Fluxograma das sub-rotinas de movimentação do cilindro



Fonte: o Autor (2015)

A Figura 7 apresenta a sub-rotina de testes que inicia-se emitindo um sinal de nível alto em todos os condutores, verificando seu retorno por meio de varredura (blocos 2 a 5). Detectada a ruptura, dados referente ao número do cabo, condutor e ciclo de rompimento são gravados em um arquivo na memória SD com nome referente ao número do ensaio (bloco 6). Esses mesmos dados são enviados via rede local para o PC (bloco 7). Caso todos os condutores se rompam ou o número de ciclos executados alcance o valor máximo, a máquina é desabilitada (blocos 8 e 9).

Figura 7- Fluxograma da sub-rotina de testes de ruptura



Fonte: o Autor (2015)

Com isso, são finalizadas as modificações do software responsável pelo controle em primeiro nível da máquina, da rede e do recebimento dos dados por parte do controlador Arduino. Para possibilitar a completa automação proposta, é necessário desenvolver o *software* de segundo nível, a qual implementa o cliente (PC) que se conecta ao Arduino.

2.3 SISTEMA PROPOSTO PARA PROCESSAMENTO DE DADOS VIA PC

Conforme Aguilar (2008), a programação em segundo nível ou alto nível permite o desenvolvimento de aplicações com alta portabilidade e maior facilidade na escrita e compreensão dos códigos.

Para a efetivação da transferência e processamento dos dados dos testes de nível de máquina para nível alto, é desenvolvido um aplicativo para o PC que recebe as informações dos ensaios em andamento. Esse aplicativo encontra-se sobre a camada de aplicação do protocolo TCP/IP, onde suas principais funções são:

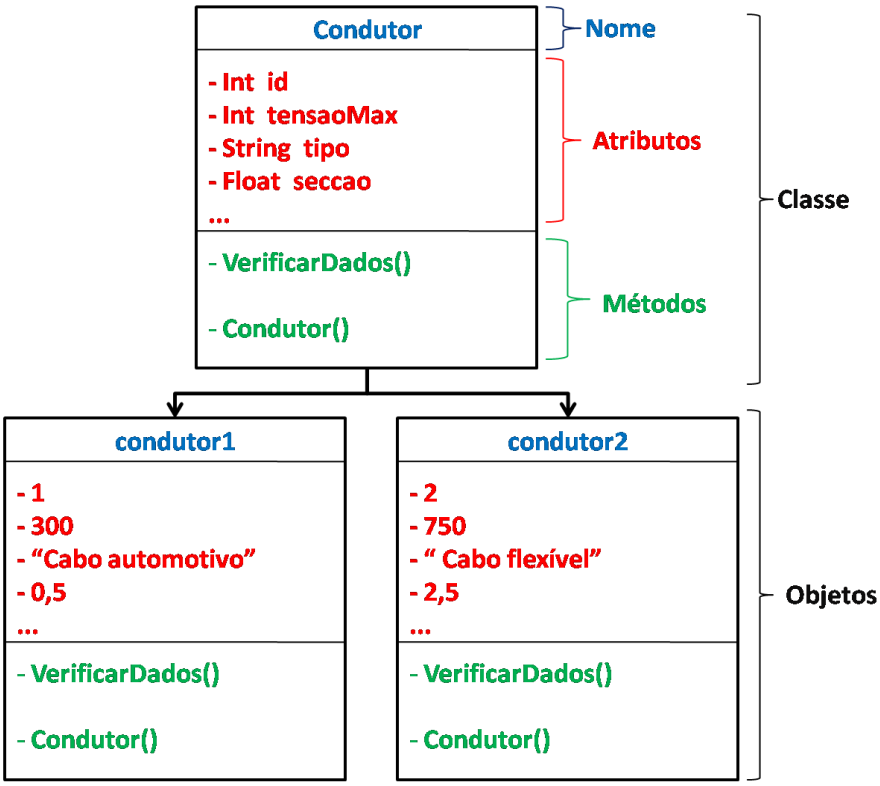
- Desenvolver a interface com o usuário para a supervisão remota dos ensaios em andamento e cadastramento de novos testes e condutores utilizados;
- Receber os dados dos ensaios emitidos pelo controlador através da rede local ou por arquivos via cartão SD;
- Armazenar as informações dos ensaios em banco de dados;
- Desenvolver e apresentar resultados estatísticos das informações coletadas conforme exigido pelo usuário

Para essa implementação será utilizada a IDE Visual Studio Community 2015, desenvolvida pela empresa Microsoft, que a disponibiliza gratuitamente para desenvolvedores individuais (pessoas físicas) ou em aplicações limitadas dentro de organizações (pesquisas, ensino ou projetos em código aberto). O Visual Studio permite a programação em alto nível através de diversos tipos de linguagens como C++, C#, JavaScript, Visual Basic, HTML e Python, contando com um sistema de compilação e depuração avançada dos códigos (Microsoft, 2015).

A linguagem de programação escolhida será a C# (Csharp), orientada a objetos e compatível ao desenvolvimento de aplicações como manipulações em banco de dados e interfaces gráficas ao usuário (*GUI*). Esse modelo de programação, implementado via C#, atua através da criação de objetos (instância) que seguem as características de um molde. Esse molde é chamado de classe, o qual é responsável em representar um conjunto de objetos que possuem características e comportamentos em comum (Dionisyo, Tavares e Júnior, 2013).

Na Figura 8 é apresentado o modelo de classe e suas instâncias através de um exemplo que armazena e trata os dados dos condutores utilizados nos testes.

Figura 8- Exemplo de classes e objetos



Fonte:o Autor (2015)

Segundo Deitel (2003), a classe exibida é dividida em três importantes campos de acordo com as suas funções. Conforme o exemplo, o nome (Condutor) identifica a classe, permitindo ser referenciada pelo *software*. Os atributos (id, tensaoMax, seccao, etc.) armazenam as variáveis que a classe possui. Os métodos (VerificarDados e Condutor) definem as funções executadas pela classe. Nesse último campo é apresentado uma função especial, a qual recebe o mesmo nome da classe (Condutor). Esse método é denominado como construtor e difere-se dos demais por ser executado sempre ao criar-se um novo objeto. O objetivo principal do construtor é permitir a inicialização de variáveis ou processos quando efetuada a instância.

Os condutores 1 e 2 representam os objetos instanciados, o que permitem o recebimento de distintas características (variáveis), mantendo os atributos e métodos da classe referenciada.

2.3.1 Recursos disponíveis utilizados no aplicativo

A linguagem C# e as amplas bibliotecas existentes no Visual Studio permitem a utilização de comandos existentes que implementam diversas funções essenciais para o desenvolvimento do aplicativo. Segue nas próximas tabelas os recursos utilizados de maior relevância no trabalho e disponibilizados na IDE escolhida, sendo divididas em métodos, classes ou estruturas.

A Tabela 3 apresenta os recursos básicos utilizados em funções para a apresentação de dados, conversões, gerenciamento de arquivos externos e representação de variáveis.

Tabela 3- Recursos básicos

Tipo	Recurso	Função
Método	Control.Find	Procura controles em um formulário conforme o nome atribuído. Utilizado na atualização dinâmica dos dados da interface gráfica
Estrutura	DateTime	Estrutura que representa uma variável de tempo, utilizado para armazenamento de data e hora.
Classe	Convert	Converte expressões para o tipo desejado.
	Interation	Cria uma caixa de diálogo que recebe dados inseridos pelo usuário.
	File	Implementa métodos para o gerenciamento de arquivos .
	MessageBox	Cria caixas de diálogo para exibição de mensagens ao usuário
	Path	Instância um arquivo conforme o caminho informado.

Fonte:Microsoft (2015)

A Tabela 4 contém os recursos utilizados para a comunicação com o Arduino, bem como o recebimento e envio dos dados via rede local, essencial para a supervisão dos ensaios em andamento.

Tabela 4- Recursos utilizados na comunicação com o Arduino

Tipo	Recurso	Função
Método	Socket.Connect	Estabelece a comunicação com a rede através de um ponto de acesso informado.
	Socket.Receive	Recebe os dados em bytes do canal de comunicação criado.
	Socket.Send	Envia os dados em bytes para o canal de comunicação criado.
Classe	Socket	Implementa um canal que utiliza uma porta para comunicação, permitindo a troca de informações com a rede. Para a utilização é necessário informar parâmetros como o método de comunicação através do protocolo TCP / IP e o fluxo de dados por Stream.
	Thread	Cria e gerencia processos de divisão de tarefas em mais de um segmento para a execução de funções de modo concorrente. Esse recurso é necessário quando deseja operar vários processos em conjunto, fazendo com que a Thread comute os comandos das funções e os processos aparentam operarem em paralelo.

	IPEndPoint	Implementa um ponto de acesso, permitindo a comunicação entre o aplicativo e a rede através de um endereço IP e número da porta.
	IPAddress	Fornece um endereço IP a partir de uma cadeia de caracteres

Fonte:Microsoft (2015)

Por fim, a Tabela 5 apresenta os recursos utilizados para a operação e gerenciamento das informações em banco de dados através da inserção da biblioteca SQLite. Maiores detalhes desse sistema são descritos no Apêndice C e na descrição da classe desenvolvida para esse fim (Seção 2.3.2.2).

Tabela 5- Recursos para banco de dados

Tipo	Recurso	Função
Método	SQLiteCommand. ExecuteReader	Executa comandos para leituras de linhas nas tabelas de dados
	SQLiteConnection .CreateCommand	Cria um comando a ser enviado para o banco de dados. Esse método permite o gerenciamento dos dados em linguagem SQL através de cadeia de caracteres em C#
Classe	SQLiteConnection	Cria uma vínculo com o banco de dados especificado, permitindo métodos que abrem ou encerram a sua comunicação
	SQLiteConnection.CreateFile	Cria o arquivo do sistema de banco de dados
	SQLiteDataReader	Responsável pela leitura de registros presente no banco de dados

Fonte:Microsoft (2015)

2.3.2 Classes desenvolvidas para o aplicativo

Seguindo a orientação a objetos, são desenvolvidos cinco classes organizadas conforme as suas funções e comportamentos, as quais, em conjunto com os recursos disponíveis, permitem o funcionamento do aplicativo proposto.

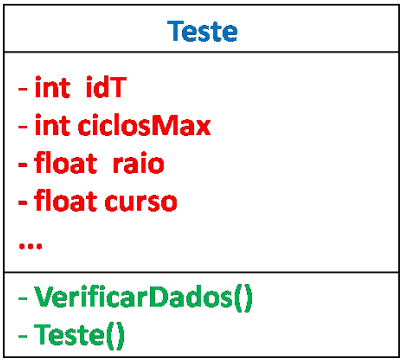
2.3.2.1 Classe Condutor

Classe apresentada na Figura 8, a qual representa os condutores cadastrados para os ensaios. Os seus atributos definem o código para a identificação do condutor (ID), características físicas (secção, material, fabricante, etc.) e descrições adicionais. Seus métodos desenvolvem a verificação da consistência dos dados recebidos e a inicialização das variáveis pelo construtor.

2.3.2.2 Classe Teste

Demonstrada na Figura 9, essa classe recebe as informações dos ensaios cadastrados com atributos que definem o código para identificação (idT), características do ensaio (raio de curvatura dos cabos, comprimento do curso, ciclo máximo, etc.) e a data de início do teste. Possui os mesmos métodos que a classe Condutor.

Figura 9- Classe Teste



Fonte:o Autor (2015)

2.3.2.3 Classe BancodeDados

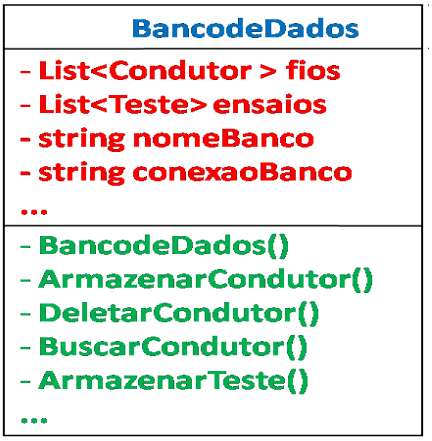
Como a principal função do aplicativo é o cadastramento de testes, recebimento de dados na supervisão dos ensaios e o desenvolvimento de resultados estatísticos, é essencial a implementação de um processo para o gerenciamento e armazenamento de informações. Para isso, é utilizado o sistema de banco de dados que organizam os registros por meio de arquivos divididos em tabelas.

O banco de dados é implementado através da biblioteca SQLite, de domínio público e robusta (SQLite, 2015), podendo ser integrada facilmente ao Visual Studio através de seu gerenciador de bibliotecas (*NuGet*). Os comandos que executam as funções de gerenciamento do banco de dados são desenvolvidos na linguagem SQL (*Structured Query Language*), projetada para essa funcionalidade.

Conforme Prescott (2015), os comandos em SQL são classificados em duas categorias. A primeira é a DDL (Linguagem de definição de dados), utilizada na criação e alteração de estruturas no banco de dados, como tabelas ou objetos. A segunda é a DML (Linguagem de Manipulação de Dados) que atribui comandos para a manipulação dos dados presentes nas tabelas ou objetos.

Essas funções são implementadas através da Classe *BancodeDados* (Figura 10), onde atuam funções responsáveis pela conexão com o banco de dados e envio dos códigos em SQL através de comandos implementados pelo biblioteca SQLite (Tabela 5). Seu construtor é responsável pela criação do arquivo de armazenamento e de suas tabelas. Os demais métodos são responsáveis pelo armazenamento, exclusão, e carregamento de dados.

Figura 10- Classe BancodeDados

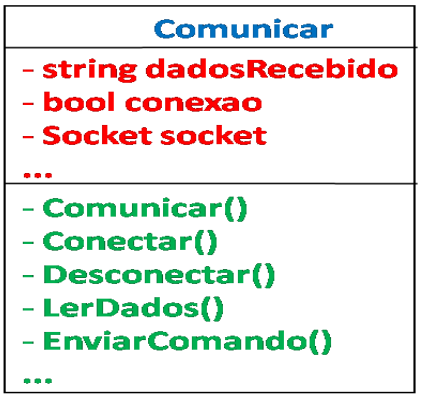


Fonte:o Autor (2015)

2.3.2.4 Classe Comunicar

Classe demonstrada na Figura 11 e utilizada durante a supervisão dos ensaios. Introduz métodos que desenvolve a comunicação com o Arduino, leitura dos dados recebidos pela rede local, identificação e seleção dos dados obtidos, envio de comando para o controlador da máquina e encerramento da conexão.

Figura 11- Classe Comunicar

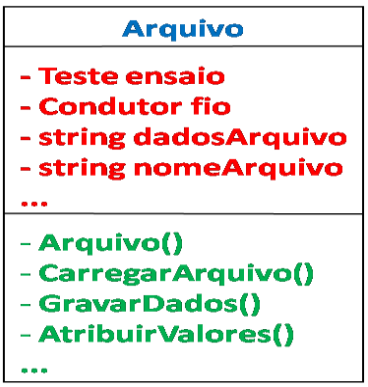


Fonte:o Autor (2015)

2.3.2.5 Classe Arquivo

Utilizada para o carregamento de dados através de arquivos e apresentada na Figura 12. Essa classe atua quando o usuário desejar transferir os arquivos para o software através da memória SD. É responsável por desenvolver a leitura dos dados e o tratamento das informações, as quais são posteriormente enviadas para a classe responsável pelo armazenamento no banco de dados.

Figura 12- Classe Arquivo



Fonte:o Autor (2015)

2.3.2.6 Classe Resultados

Atua no desenvolvimento dos cálculos estatísticos utilizados no processamento de resultados para o usuário e representada na Figura 13. É composta por métodos que agregam os testes e seus condutores em um único objeto, filtragem de dados e operações matemáticas (média, desvio padrão, etc.) das informações conforme solicitado na interface.

Figura 13- Classe Resultados

Fonte:o Autor (2015)

2.3.3 Telas e funcionamento do aplicativo

O desenvolvimento do aplicativo pelo Visual Studio envolve a criação de interfaces entre o sistema e o usuário. Devido a isso, utiliza-se a implementação de telas através da *WinForms*. Conforme Deitel (2003), a *WinForms* é um conjunto de bibliotecas (*frameworks*) fornecido pelo Visual Studio que implementa a interface gráfica com o usuário através da criação de formulários (janelas ou caixas de diálogos) e da definição do comportamentos de seus componentes de controle (botões, caixas de texto, barras, etc.). Para cada formulário criado é atribuída uma nova classe que carrega as funções executadas pelo mesmo. Na Tabela 6 são exibidos os principais componentes fornecidos pelo *WinForms* e utilizados no desenvolvimento do projeto

Tabela 6- Principais componentes *WinForms* utilizados

Componente	Função
Button	Cria um botão no formulário.
Chart	Insere e controla a inserção de gráficos.
CheckBox	Cria uma caixa de seleção.
ComboBox	Permite que o usuário selecione um item de uma lista de opções cadastradas.
DataGridView	Estrutura e exibe dados em uma tabela no formulário.
GroupBox	Permite agrupamento de diversos componentes, delimitando-os no formulário.
Label	Permite a adição de textos no formulário (somente leitura).
MaskedTextBox	Desenvolve uma máscara, padronizando o modelo do texto apresentado ou recebido pelo usuário.
MenuStrip	Fornece sistema de menus.
OpenFileDialog	Cria uma caixa de diálogo que desenvolve a interface para seleção de arquivos ao usuário.
RichTextBox	Permite a edição e apresentação de textos com melhor formatação que a TextBox.
TextBox	Permite a edição e apresentação de textos no formulário.

Fonte:Microsoft (2015)

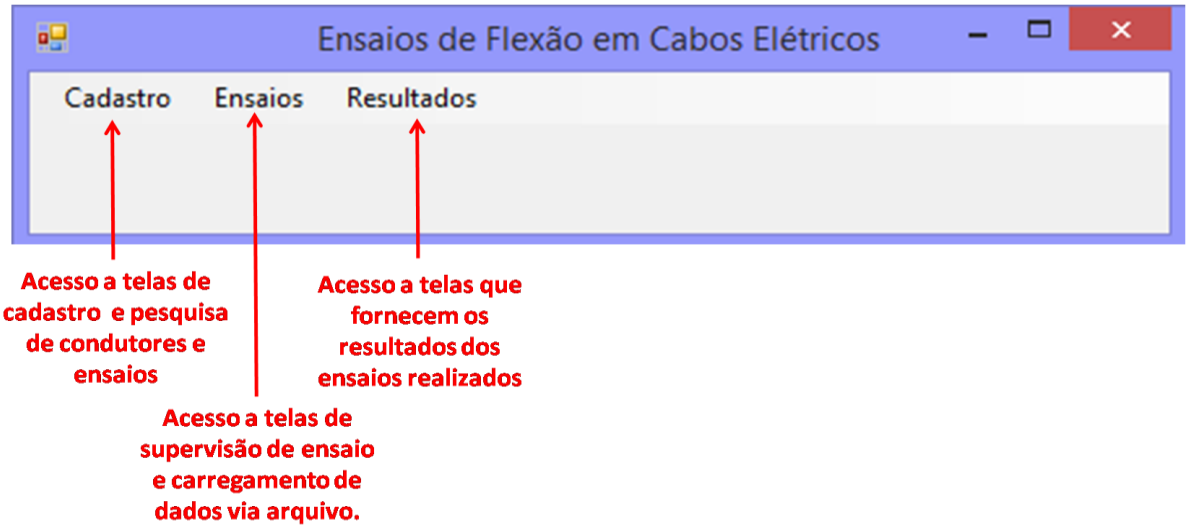
A seguir são exibidos os formulários desenvolvidos com suas determinadas funções.

2.3.3.1 Tela Principal

Quando iniciado, o aplicativo abre a Tela Principal (Figura 14), formulário que exerce a função de menu, permitindo o acesso aos demais formulários

presentes no aplicativo. Durante seu carregamento, o banco de dados do aplicativo é inicializado, criando os arquivos e estruturas que o compõe, caso inexistente. No Apêndice C encontram-se maiores detalhes sobre esse processo.

Figura 14- Tela Principal



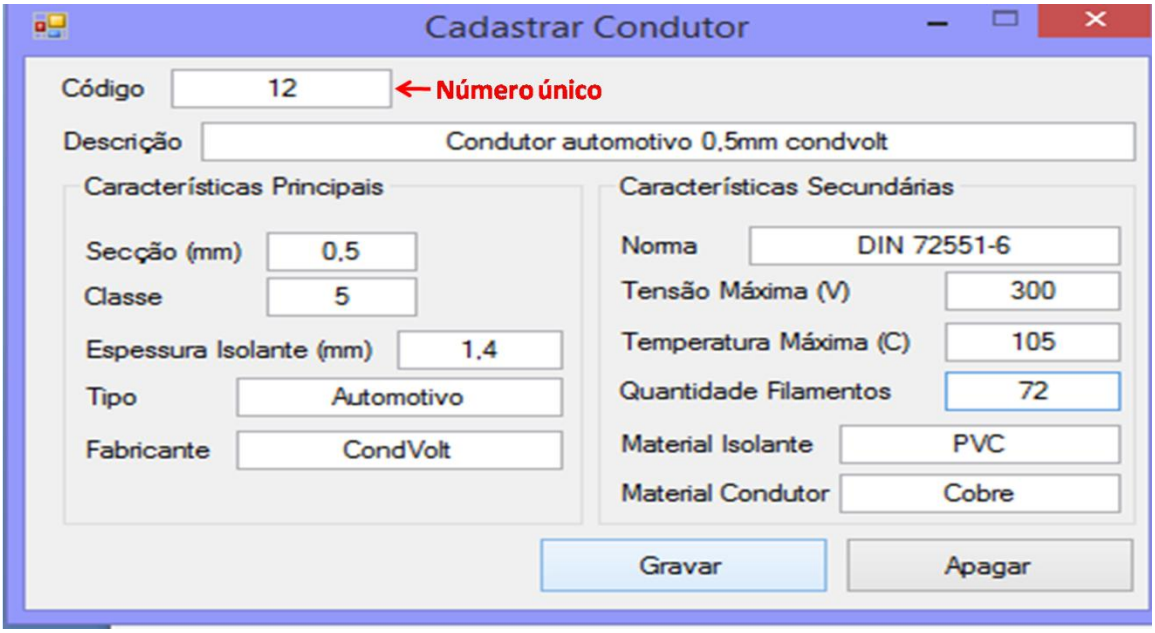
Fonte:o Autor (2015).

Na tela principal, assim como nas telas de cadastramento e supervisão de dados, quando o usuário deseja fechá-las, uma caixa de diálogo de confirmação é apresentada via função *MessageBox* (Tabela 3). Esse processo dificulta a perda de dados caso o usuário as finalize por engano

2.3.3.2 Tela de Cadastro de Condutor

Acessando os botões "*Cadastro"* e "*Condutor" na* tela inicial, é exibido o formulário para cadastro de condutores (Figura 15). O objetivo desse cadastro é permitir a identificação das características dos condutores em teste, visto que a máquina não difere, apenas detecta o ciclo da sua ruptura e o relaciona a um código de identificação.

Figura 15- Tela Cadastro de Condutores



Fonte:o Autor (2015).

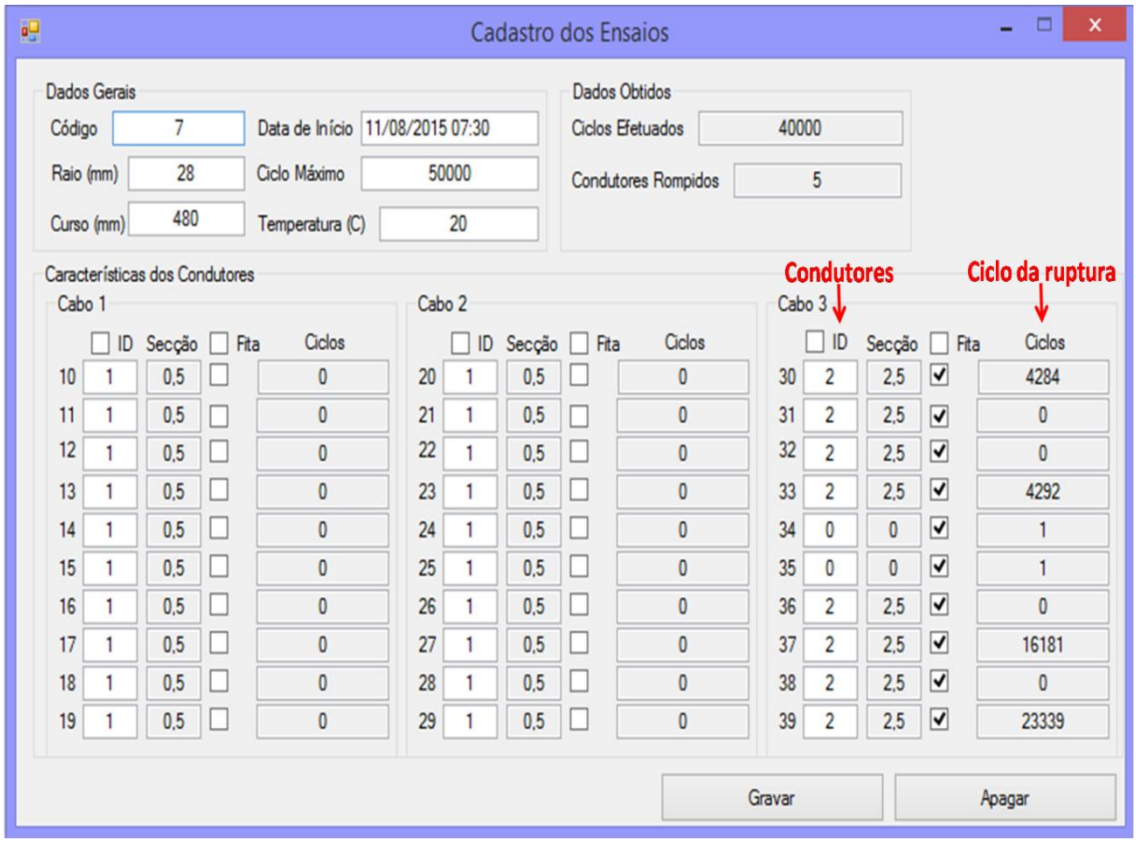
Através da inserção de um número único de endereçamento, o formulário permite a criação, alteração ou exclusão de registros dos condutores e suas características no banco de dados do aplicativo. Caso algum dado inserido se demonstre inconsistente ou o código não for informado, uma tela de erro alerta o usuário e a operação não é executada.

2.3.3.3 Tela de Cadastro de Ensaios

Na barra da *Tela Principal*, acessando as opções *"Cadastro"* e *"Ensaio"*, o formulário *Cadastro de Ensaios* (Figura 16) é carregado. Essa tela é responsável pela criação, alteração ou exclusão de registros dos ensaios a serem executados. Permite a inserção de dados referente a configuração da máquina (raio de curvatura, ciclos máximos, presença de fita nos cabos etc.) e dos condutores que compõe o teste, relacionando-os com a numeração utilizada pela máquina no reconhecimento do condutor rompido (números de 10 a 39).

Essa tela também proporciona o acesso das informações dos ensaios finalizados, apresentando a quantidade total de ciclos efetuados e os condutores rompidos, exibindo o número do ciclo da sua ocorrência. Condutores não rompidos carregam a quantidade de ciclos zerada.

Figura 16- Tela de Cadastro de Ensaios

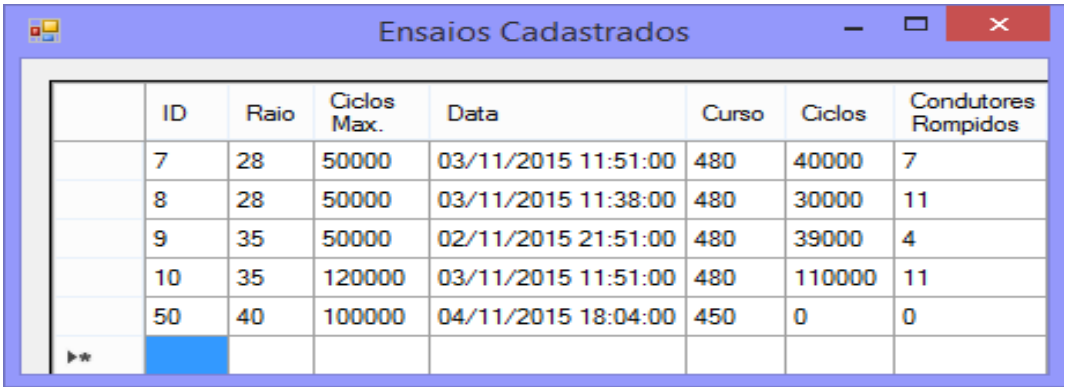


Fonte:o Autor (2015).

2.3.3.4 Telas de Pesquisa de Condutores e Ensaios

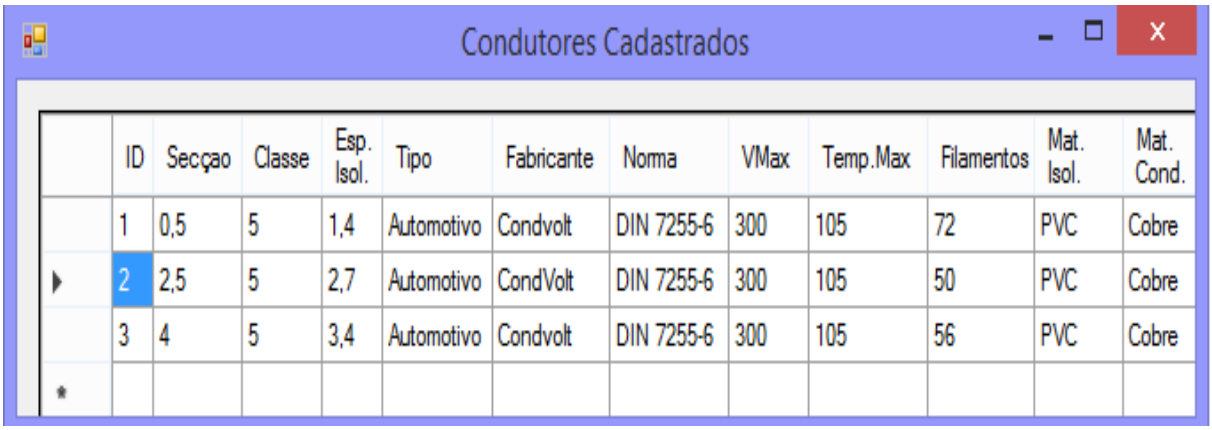
Acessando as opções no menu de *"Cadastro"* e "*Pesquisar"*, é disponibilizado o acesso a dois formulários (Figuras 17 e 18) que informam ao usuário todos os condutores e testes cadastrados no sistema. Para a exibição dos dados via tabela na tela, é utilizado o componente *DataGridView*.

Figura 17- Tela de Pesquisa dos Condutores



Fonte:o Autor (2015).

Figura 18- Tela de Pesquisa dos Testes

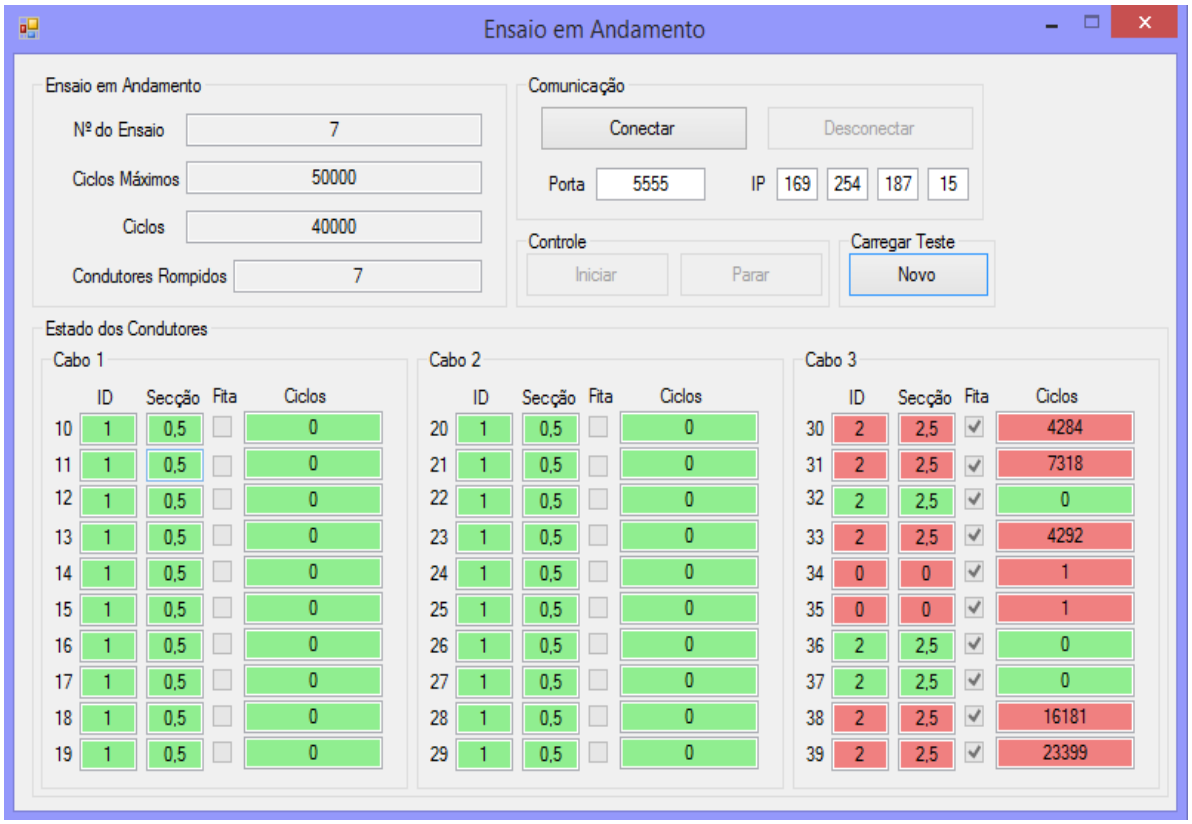


Fonte:o Autor (2015).

2.3.3.5 Tela de Supervisão

A Tela de Supervisão (Figura 19), acessada através das opções *"Ensaios"* e *"Supervisão de Ensaio"* , desenvolve a interligação entre o usuário e a máquina de ensaios de flexão de maneira dinâmica.

Figura 19-Tela de Supervisão



Fonte:o Autor (2015).

Com o carregamento do código do ensaio e dos parâmetros de conexão com o Arduino (utilização da classe *Socket),* o aplicativo habilita o envio de comandos que iniciam ou pausam a máquina e o recebimento, via Threads, de dados de ruptura e quantidade de ciclos. Os condutores em condução são marcados em verde, enquanto os rompidos em vermelho. Para a seleção do número do teste a ser carregado é utilizada uma caixa de diálogo implementada pela classe *Interation*.

Durante esse processo de supervisão, inúmeras informações são transferidas entre o controlador e o computador. Para isso, são adicionados caracteres únicos nos dados transmitidos, permitindo com que o Arduino e o PC identifiquem o tipo de mensagem recebida e a processem de maneira correta. Na Tabela 7 são apresentados as caracteres utilizados e o tipo de informação correspondente.

Tabela 7- Caracteres para identificação de dados

Caractere recebido pelo Arduino	**Tipo de informação**	**Caractere recebido pelo PC**	**Tipo de informação**
M	Quantidade máxima de ciclos	R	Ruptura de condutor
I	Número do teste	T	Total de rupturas
D	Encerramento da conexão	C	Quantidade de ciclos
L	Iniciar os testes		
P	Parar os testes		
F	Término da string de dados		
R	Vetor de ruptura e reset		
C	Quantidade de ciclos		

Fonte:o Autor (2015).

O Arduino, ao receber um comando, divide o mesmo em caracteres que representam letras dos que representam números. Sabendo a letra recebida, o controlador consegue identificar o tipo de dado recebido e processá-lo de maneira correta. No final de cada mensagem enviada, um caractere F é adicionado indicando o término da cadeia.

De forma semelhante, o aplicativo presente no PC identifica o caractere de identificação através do método *EndsWith*, permitindo o tratamento correto da mensagem. Na Figura 20 é apresentado um exemplo dessa aplicação.

Figura 20- Exemplo de dados transferidos

Informação recebida pelo Arduino
M50000F
Informação recebida pelo PC
29R

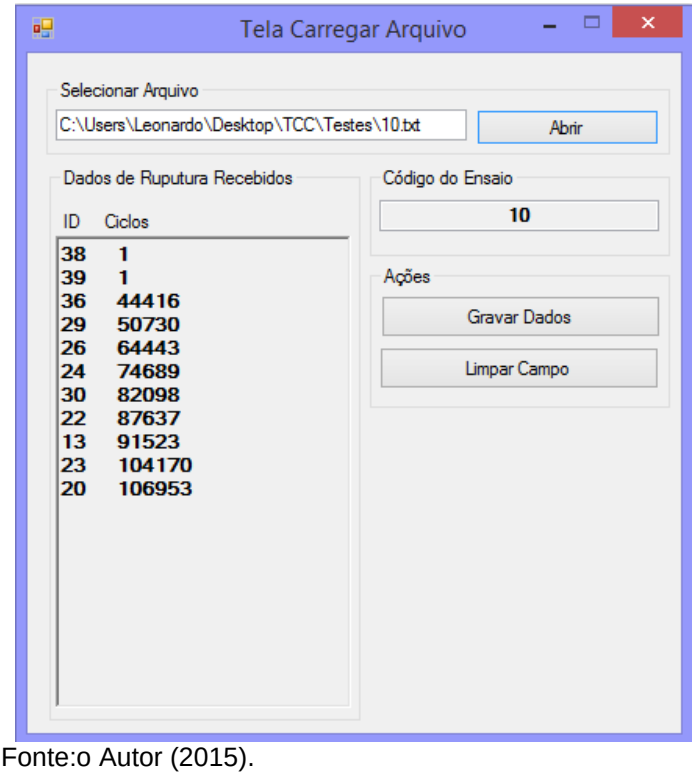
Fonte:o Autor(2015)

O dado recebido pelo Arduino representa uma informação de quantidade máxima de ciclos (caractere M) de 50000. O caractere F simboliza o término do número informado. A informação recebida pelo PC representa que ocorreu uma ruptura (caractere R) no condutor de posição 29 da máquina.

2.3.3.6 Tela de Arquivo

Caso o usuário deseje carregar os dados dos testes via arquivos presentes na memória SD da máquina, é disponibilizado o formulário da Figura 21, acessado através dos botões *"Ensaio"* e *"Carregar Arquivo"* no menu principal. Esse formulário permite o carregamento de arquivos e o armazenamento de suas informações no banco de dados, vinculados a um teste previamente cadastrado.

Figura 21- Tela de Arquivos



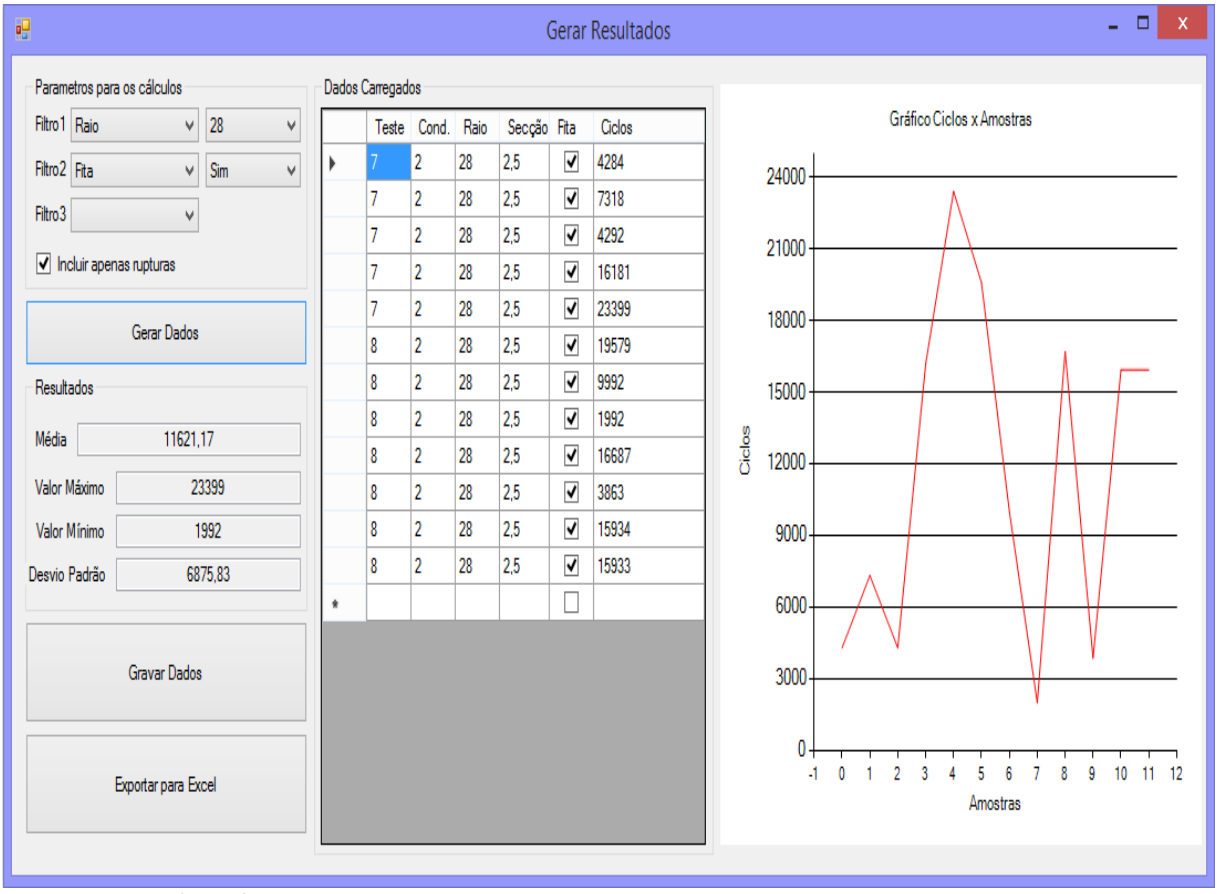
Fonte:o Autor (2015).

A seleção do arquivo pelo usuário é desenvolvida através do componente *OpenFileDialog* (Tabela 6), onde é criada uma caixa de diálogo padrão do *Windows* e que possibilita a busca dos dados pelo computador.

2.3.3.7 Tela de Resultados e Pesquisa de Resultados

Através da aba *"Resultados"* no menu principal, o usuário tem acesso as opções: *"Gerar Resultados"* e *"Pesquisar Resultados".* Escolhendo a primeira é apresentado o formulário da Figura 22 , o qual exibe os resultados estatísticos dos testes executados.

Figura 22- Tela Gerar Resultados



Fonte:o Autor (2015).

Para a apresentação dos resultados, o usuário deve fornecer os parâmetros a serem utilizados como filtro de dados, onde é possível a inserção de até três características simultâneas. Após o processamento, é exibido um gráfico e os resultados estatísticos sobre a quantidade de ciclos de rompimento dos condutores. Essas informações podem ser armazenadas internamente via banco de dados ou exportá-las para uma tabela no *Excel*, *software* de planilhas utilizado pela empresa.

O último formulário desenvolvido (Figura 23) é acessado pela segunda opção da aba *"Resultados"*, e apresenta todos os registros armazenados no banco de dados conforme gerados no formulário anterior. Há também a possibilidade de exportação para planilhas no Excel. Como exemplo, temos na primeira linha (ID 1), os resultados gerados pelos filtros com raio de 28mm, secção de 2,5mm² e com fita, resultando na média de 11621, desvio padrão de 6875, valor máximo de 23399 e valor mínimo de 1992. Esses valores estatísticos referem-se na quantidade de ciclos para a ruptura dos condutores amostrados.

Figura 23- Tela Pesquisar Resultados

	ID	1ª Par.	Valor 1	2ª Par.	Valor 2	3ª Par.	Valor 3	Média	DesPa	ValMax	ValMin
	1	Raio	28	Secção	2,5	Fita	true	11621,1...	6875,83	23399	1992
	2	Raio	35	Secção	2,5	Fita	false	63257	18841	82098	44416
	3	Raio	28	Secção	2,5			11621,1...	6875,83	23399	1992
	4	Ensaio	8					11997,1...	6333,1	19579	1992
	5	Fita	false					47520	17826,64	82098	31473
	6	Fita	false	Secção	2,5			63257	18841	82098	44416

Fonte:o Autor (2015).

Com a implementação dos formulários de interface gráfica (telas) descritos, se obtém a finalização do aplicativo presente no PC, responsável pelo gerenciamento em segundo nível dos ensaios de flexão. No Apêndice E é apresentado o código em C# desenvolvido, referente ao processo de comunicação entre o Arduino e o aplicativo.

No Capítulo 3 é demonstrada uma aplicação prática da automação implementada, sendo destinada aos produtos produzidos pela empresa Palfinger.

CAPÍTULO 3 - PESQUISA DA VIDA ÚTIL DOS CONDUTORES APLICADOS EM ELEVADORES VEICULARES

Efetivada a automação da máquina de ensaios de flexão em cabos elétricos proposta, desenvolve-se nesse capítulo um estudo sobre a vida útil dos condutores utilizados nos elevadores veiculares da Palfinger. O objetivo dessa pesquisa é demonstrar a aplicação prática do sistema de ensaios implementado, oferecendo importantes informações para a empresa. São realizados ensaios para a coleta de dados, análises dos resultados e conclusões sobre as características que influenciam na vida útil dos condutores, buscando obter informações que auxiliem em futuras pesquisas realizadas pela engenharia da Palfinger.

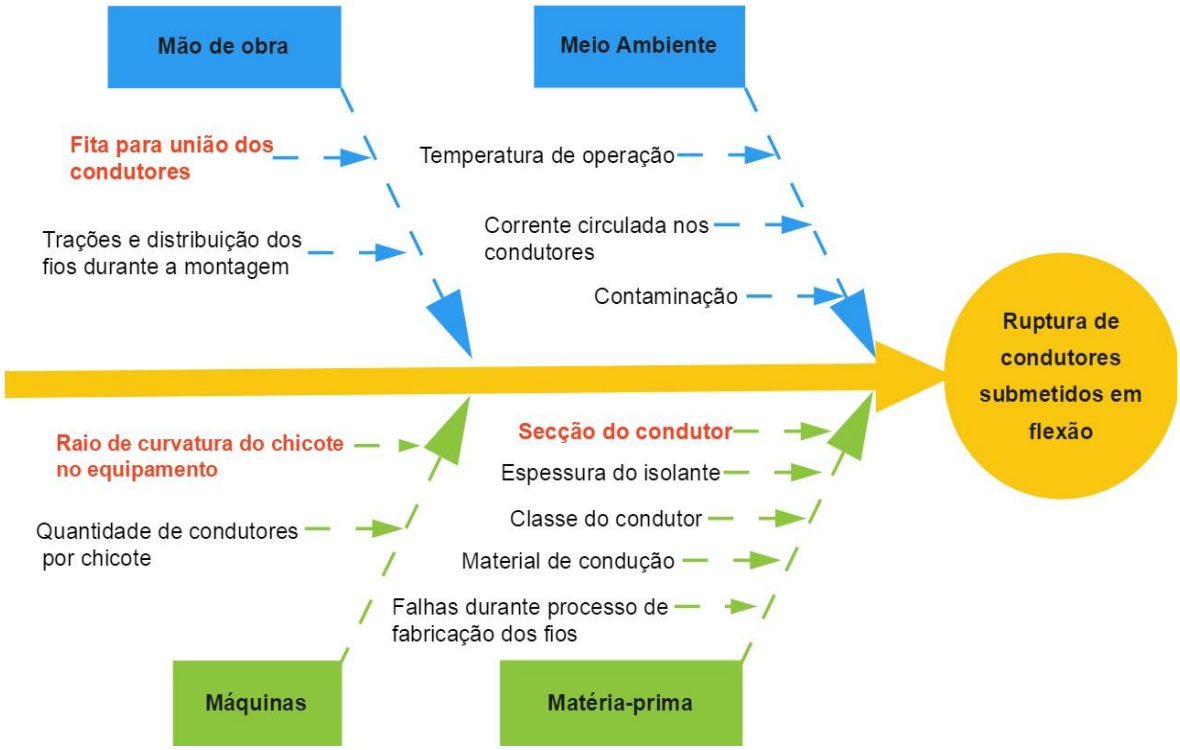
3.1 DEFINIÇÃO DOS FATORES DE RUPTURA EM ESTUDO

Inicia-se o estudo através da análise prévia sobre os principais parâmetros que, possivelmente, influenciam no desgastes de chicotes em flexão, definindo assim as variáveis dos ensaios a serem executados. Para isso, é utilizado o diagrama *Ishikawa*, ferramenta gráfica de qualidade que auxilia na identificação de causas de um determinado problema (César, 2011).

Conforme César (2011), o diagrama *Ishikawa* é iniciado através da definição do problema alvo, no caso, a ruptura de condutores submetidos em flexão. As possíveis causas do problema são adicionadas ao diagrama, sendo classificadas conforme a sua natureza de origem.

No diagrama *Ishikawa* aplicado a pesquisa sobre a vida útil dos chicotes elétricos (Figura 24), foram detectados 12 possíveis causas de ruptura dos condutores presentes nos elevadores veiculares, sendo divididas em quatro classificações: mão-de-obra, meio ambiente, máquinas e matéria-prima. As causas em vermelho representam as variáveis escolhidas para a configuração dos ensaios.

Figura 24- Diagrama *Ishikawa* desenvolvido



Fonte:o Autor (2015).

A Tabela 8 contém as análises responsáveis pela inclusão ou exclusão das variáveis nos ensaios, apresentadas no diagrama *Ishikawa*. Esse processo é desenvolvido pelo método *brainstorming*, onde são inseridos os conceitos prévios por meio de sugestões (César, 2011).

Tabela 8- Análise sobre possíveis causa de ruptura

Causa	Análise	Parecer
1- Fita para união dos condutores	Durante a fabricação dos chicotes elétricos, seus condutores são envolvidos com uma fita isolante, que facilita a sua confecção. Essa fita exerce uma resistência na flexibilidade do cabo e pode influenciar na sua movimentação.	Aceita devido a facilidade de inclusão nos ensaios e possibilidades de alteração no produto final.
2- Trações e distribuição dos fios do chicote **3- Falhas na fabricação dos fios.**	Durante o manuseio dos condutores na montagem do chicote, existe o risco dos mesmos sofrerem esforços demasiados ou má distribuição. Essas características podem comprometer a integridade do cabo ou torná-lo mais suscetível ao desgaste devido ao acomodamento incorreto dos fios durante a movimentação. Falhas na produção ou baixa qualidade dos condutores também podem agravar o problema.	Rejeitada devido a sua alta variação, dificuldade na inclusão dos testes e dificuldade na tomada de ações para a solução. Será tratada através da quantidade de amostras e margem de erro dos ensaios.

4- Temperatura de operação do equipamento **5- Corrente de circulação nos condutores**	A elevação de temperatura nos condutores originada pelo ambiente ou pela corrente que circula podem ocasionar a degradação das suas propriedades , incluindo a flexibilidade.	Rejeitada devido a máquina de ensaios não permitir a inclusão dessas variáveis.
6- Contaminação	Presença de resíduos que ocasionem o aumento de desgastes dos chicotes em flexão, seja por atrito, corrosão ou redução da flexibilidade.	Rejeitada devido a sua alta variação. São inseridas, no manual do equipamento, instruções de limpeza periódica, a fim de minimizar esse risco.
7- Raio de curvatura do chicote	O raio possui influência nos desgastes dos cabos em flexão devido a atenuação da "dobra" dos condutores originadas da sua distribuição em forma de "U".	Aceita devido a possibilidade na inclusão dos ensaios e alteração no produto final.
8- Quantidades de condutores por chicote	Influencia a distribuição dos condutores durante a sua movimentação e possibilita um aumento no atrito com as paredes da esteira porta-cabos que o envolve.	Rejeitada devido a dificuldade de alteração no produto. Nos ensaios realizados, são definidos a quantidade máxima utilizada de condutores (10 fios).
9- Espessura da isolação **10- Classe do condutor** **11- Material de condução**	Características de fabricação dos condutores que exercem influência na sua flexibilidade. Dentre as variáveis citadas, a classe é a que mais se sobressai, pois define a quantidade e diâmetro dos filamentos que compõe cada condutor, caracterizando o grau de flexibilidade (Santos, 2005).	Rejeitada devido a impossibilidade de alteração no produto final. Isso se deve pois, como os chicotes são utilizados em elevadores veiculares, o fabricante segue normas regulamentadoras que certificam a aplicação. A norma referida é a DIN 72551 e a ISO 6722, que definem as características de condutores de baixa tensão para uso automotivo (LS, 2005).
12- Secção do condutor	A secção pode interferir na flexibilidade e atenuações da "dobra" dos condutores durante a flexão.	Aceita devido a possibilidade de inserção dos testes e alteração no produto final.

Fonte: o Autor (2015), Santos (2005), LSCable (2005).

Após escolhidos os fatores, são definidos dois valores (níveis) para cada um, os quais são utilizados na composição dos ensaios, o que permite a análise das suas influências na ruptura dos cabos. Segue abaixo a relação entre os fatores e os níveis definidos:

- Fita para união dos condutores - Os dois níveis escolhidos são condutores com fita e sem fita. Esse fator representa a fita isolante que envolve os condutores durante a sua montagem pelo fabricante. É realizado ensaios com fios livres e unidos com fita isolante em três pontos (extremidades e centro), conforme padrão utilizado no projeto.
- Raio de curvatura do chicote - Seus níveis são representados por ensaios com raio de curvatura de 28mm e 35mm. Esses valores são definidos através dos limites informados pelo fabricante da esteira porta-cabos utilizada (Igus, 2015).
- Secção do condutor - Os níveis definidos são condutores com secções de 0,5mm² e 2,5mm², que representam os fios utilizados no modelo de elevador veicular em estudo e definidos pela engenharia da Palfinger. A menor secção é utilizada em cabos de controle, enquanto a maior em cabos de alimentação de motores.

3.2 DEFINIÇÃO DOS ENSAIOS REALIZADOS

Com a definição dos fatores e valores de seus níveis, configura-se os testes através do método de planejamento de experimentos (DOE) *Taguchi*, o qual utiliza matrizes ortogonais que permitem a redução na quantidade de ensaios e melhor aproveitamento das informações coletadas (Swaab e Pinto, 2011). Como o estudo é composto por três fatores (fita, secção e raio), com dois níveis cada, é escolhido aplicar a matriz L4 do método *Taguch*i (Tabela 9), conforme orientado em Cardoso (2014).

Tabela 9- Matriz Taguchi para definição dos experimentos

L4	Fatores e níveis		
Experimentos	Raio(mm)	Secção(mm²)	Fita
1	28	0,5	Não
2	28	2,5	Sim
3	35	0,5	Sim
4	35	2,5	Não

Fonte:adaptado de Cardoso (2014).

Devido a máquina comportar o ensaio de até 30 condutores simultaneamente, determina-se na Tabela 10 a composição de cada teste, relacionando a capacidade da máquina, com os experimentos configurados na matriz *Taguchi*. A quantidade de amostras de cada condutor é definida em três chicotes para cada configuração, compostos cada um de 10 condutores para a secção de 0,5mm² e 8 para a secção de 2,5mm². A redução do número de fios de 2,5mm² ocorre devido o espaço existente na esteira porta-cabos não comportar maiores quantidades.

Tabela 10- Ensaios a serem realizado pela máquina

	Máquina		
Ensaios	Cabo 1	Cabo 2	Cabo 3
A	10 x Exp. 1	10 x Exp. 1	8 x Exp. 2
B	10 x Exp. 1	8 X Exp. 2	8 x Exp. 2
C	10 x Exp. 3	10 x Exp. 3	8 x Exp. 4
D	10 x Exp. 3	8 x Exp. 4	8 x Exp. 4

Fonte:o Autor (2015).

Por fim, é estimado a quantidade mínima de ciclos que deve ser executado em cada teste. Essa quantidade é definida através da Equação (1), que relaciona a média de ciclos efetuados por cada elevador em campo e o tempo de vida útil ideal.

$$Cmin = \mathrm{N} \,.\, \mathrm{Cd} \,.\, \mathrm{d}. \,\mathrm{A} \qquad (1)$$

Onde:

C_{min}=Quantidade de ciclos mínimo dos ensaios

N= Ciclos executados pelo chicote durante uma operação do elevador

Cd= Quantidade estimada de operação do elevador por dia

d= Dias em um ano

A = Vida útil estimada [anos]

Utilizando dados fornecidos pela engenharia da Palfinger referente a vida útil esperada do equipamento e a sua média de uso, encontra-se o seguinte resultado:

$$Cmin = 2\,.\,8\,.\,265.\,4 \quad (2)$$

$$Cmin = 16960 \approx \mathbf{17000\ ciclos} \quad (3)$$

Com o cálculo, define-se que a quantidade de ciclos a serem executadas durantes os ensaios não deve ser inferior a 17 mil ciclos, garantindo uma correta análise sobre a vida útil dos condutores.

Na Figura 25 é exibido um exemplo de chicote utilizado nos experimentos, composto por oito condutores de 2,5mm² de secção, sem a união com fita.

Figura 25- Exemplo de chicote utilizado

Legenda: condutores de 2,5mm² (1), terminal tubular para conexão com a máquina (2), esteira porta-cabos (3), capa para proteção externa (4).

Fonte:o Autor (2015).

3.3 RESULTADOS DOS ENSAIOS

A seguir é apresentado os resultados obtidos no desenvolvimento de cada experimento. A ID da máquina informa o código do fio rompido, onde o primeiro algarismo identifica o chicote e o segundo o condutor. No ensaio A (Tabela 11) são realizados 42 mil ciclos, sendo detectadas rupturas em cinco condutores com secção de 2,5mm² e três de 0,5mm² .

Tabela 11- Dados do ensaio A

Raio: 28mm		**Total de ciclos: 42000**
ID Maq.	**Condutor**	**Ciclo de ruptura**
30	2,5mm² c/ Fita	**4284**
33	2,5mm² c/ Fita	**4292**
31	2,5mm² c/ Fita	**7318**
38	2,5mm² c/ Fita	**16181**
39	2,5mm² c/ Fita	**23399**
25	0,5mm² c/ Fita	**31473**
24	0,5mm² c/ Fita	**37913**
20	0,5mm² c/ Fita	**41700**

Fonte:o Autor (2015).

No ensaio B (Tabela 12), são detectados a ruptura em sete condutores de 2,5mm² com a execução de 25 mil ciclos.

Tabela 12- Dados do ensaio B

Raio: 28mm		**Total de ciclos: 25000**
ID Maq.	**Condutor**	**Ciclo de ruptura**
32	2,5mm² c/ Fita	**1992**
36	2,5mm² c/ Fita	**3863**
30	2,5mm² c/ Fita	**9992**
37	2,5mm² c/ Fita	**15933**
37	2,5mm² c/ Fita	**15934**
35	2,5mm² c/ Fita	**16687**
11	2,5mm² c/ Fita	**19579**

Fonte:o Autor (2015).

Com a modificação do raio de curvatura de 28mm para 35mm, durante a coleta de dados ensaio C não foram detectadas nenhuma ruptura, mesmo com o desenvolvimento de quase 40 mil ciclos.

Devido a alteração positiva na vida útil dos condutores após o aumento do raio, no último teste (Tabela 13) realiza-se mais de 100 mil ciclos. Com a quantidade seis vezes acima do mínimo necessário, consegue identificar a ruptura em dois condutores de 2,5mm² e sete de 0,5mm².

Tabela 13- Dados do ensaio D

Raio 35mm		**Total de ciclos: 110000**
ID Maq.	**Condutor**	**Ciclo de ruptura**
36	2,5mm² s/ Fita	**44416**
29	0,5mm² c/ Fita	**50730**
26	0,5mm² c/ Fita	**64443**
24	0,5mm² c/ Fita	**74689**
30	2,5mm² s/ Fita	**82098**
22	0,5mm² c/ Fita	**87637**
13	0,5mm² c/ Fita	**91523**
23	0,5mm² c/ Fita	**104170**
20	0,5mm² c/ Fita	**106953**

Fonte:o Autor (2015).

Por fim, agrupa-se os dados obtidos nos quatro ensaios conforme cada fator, calculando a média dos ciclos de ruptura, obtendo a Tabela 13

Tabela 14- Média dos dados obtidos

Raio(mm)	**Secção(mm²)**	**Fita**	**Média**
28	0,5	Não	37028,66
28	2,5	Sim	11621,17
35	0,5	Sim	82877,86
35	2,5	Não	63257

Fonte:o Autor (2015).

Com isso, é finalizado a coleta das informações, seguindo para a análise de cada parâmetro. A Figura 26 apresenta um chicote do Ensaio A de 2,5mm² com fita. É identificado que as rupturas geralmente ocorrem na posição central do cabo, região onde a "dobra" dos condutores é mais acentuada durante a flexão.

Figura 26- Chicote rompido

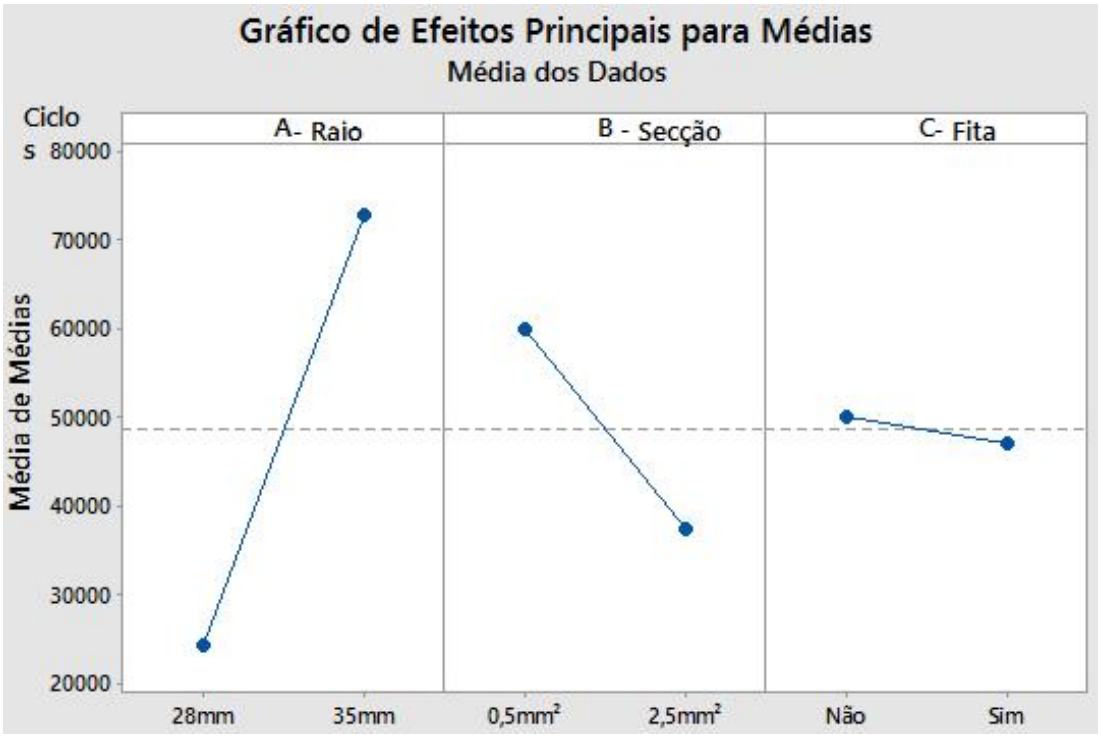
Fonte:o Autor (2015).

3.4 ANÁLISE DOS RESULTADOS

Com as informações obtidas nos testes e através do método DOE *Taguchi,* estimam-se os efeitos das características presentes nos ensaios. Para isso, é utilizado o software de cálculos estatísticos Minitab, onde é inserido a Tabela 14 que informa os fatores, níveis e a média dos resultados obtidos.

Seguindo orientações de Minitab (2015), é gerado o gráfico da Figura 27, que representa os efeitos de cada característica. Conforme Cardoso (2014), quanto maior a inclinação das retas, maior o efeito do fator. As retas em aclive representam interferência positiva na vida útil dos fios sobre flexão, já em declive a interferência negativa.

Figura 27- Gráfico de influência dos fatores



Fonte:o Autor (2015), Minitab (2015).

A análise do gráfico de efeitos permite identificar os seguintes aspectos:

- Raio - Apresentou um efeito positivo, ao aumentar o raio de curvatura a vida útil dos condutores em flexão acresce.
- Secção - Efeito negativo, ao aumentar a secção ocorre uma diminuição considerável na vida útil do condutor.
- Fita - Apesar da leve característica negativa na presença da fita, o declive pouco acentuado da reta indica um efeito de baixa relevância ao estudo.

Com essas informações, identifica-se a importância dos projetos de sistemas elétricos móveis incluírem, em suas análises, o raio de curvatura do chicote e a secção dos condutores. Esses fatores apresentaram grande influência nos ensaios realizados e possuem uma alta probabilidade de interferirem na durabilidade dos chicotes elétricos desenvolvidos.

CONCLUSÃO

O trabalho desenvolveu a automação do sistema de ensaios de flexão em cabos elétricos, onde a comunicação entre o controlador Arduino com o PC, via rede local, permite a supervisão remota dos testes. Já a criação do aplicativo em nível alto de programação possibilita o processamento dos dados obtidos.

O sistema implementado traz como benefício para o usuário o controle e monitoramento dinâmico dos ensaios, visto que os mesmos são executados em ambiente distinto ao da engenharia da empresa. Por meio do pré-cadastramento das características dos condutores e experimentos, é identificado uma melhor organização nas informações obtidas pelos testes. Esses dados são centralizados em um único aplicativo, proporcionando o rápido acesso e processamento dos resultados estatísticos por meio de interfaces gráficas com o usuário.

Através da análise prática utilizando o sistema implementado, foram identificados os fatores que exercem grande impacto ao desgaste de flexão em condutores elétricos. Essa análise proporcionou importantes informações que permitem o guiamento de futuras análises, essenciais para garantir uma melhor durabilidade de condutores empregados nos produtos da empresa e que se submetem a flexão.

Além dos benefícios para a Palfinger, o projeto concedeu importantes análises sobre o conceito da automação aplicado em situações reais dentro de empresas. A atividade de estágio em conjunto com o TCC, proporcionou uma visão geral das diversas etapas desse processo. Iniciando com a implantação de um controlador, sensores e atuadores que criaram o nível de máquina até na implementação de um aplicativo que finalizou a automação em alto nível. Durante esse processo, é identificado a utilização de diversos conceitos, como a comunicação de dados através da definição da rede Ethernet, a escolha dos protocolos TCP/IP e a atuação do módulo que permite a compatibilidade da máquina com a rede local. Por fim, a programação em nível alto possibilita a elaboração do aplicativo que concede, dentre vários aspectos, uma evidente evolução no controle do sistema.

REFERÊNCIAS

AGUILAR, Luis Joyanes. **Fundamentos de Programação** : Algoritmos, estrutura de dados e objetos. 3ª ed. São Paulo: McGraw, 2008. 720p.

ARDUINO. **Learning:** Reference. 2015. Disponível em: http://www.arduino.cc/en/Reference/HomePage. Acesso em 05 setembro de 2015.

ARDUINO. **Products:** Arduino Mega 2560. 2015. Disponível em: http://www.arduino.cc/en/Main/ArduinoBoardMega2560. Acesso em 16 setembro de 2015.

ARTOFCIRCUITS. **W5100 Ethernet Shield for Arduino**. 2015. Disponível em: http://artofcircuits.com/product/arduino-ethernet-shield. Acesso em 11 outubro de 2015.

BORTOLUZ, Leonardo. **Automação de Testes de Flexão em Cabos Elétricos.** Caxias do Sul: Universidade de Caxias do Sul, 2015. 62p.

CARDOSO. Diana E. V. C. **Otimização da remoção de photoresists no processo eWLB na indústria de semicondutores.** 2014**.** FEUP. Disponível em: http://sigarra.up.pt/feup/pt/publs_pesquisa.show_publ_file?pct_gdoc_id=382129&pct_publ_id=98808.

CÉSAR, Francisco I. G. **Ferramentas Básicas da Qualidade.** Instrumentos para gerenciamento de processos e melhoria contínua. 1ª ed. São Paulo: Biblioteca24horas, Seven System International, 2011. 132p.

COSTA, Rogério L. C. **SQL: Guia Prático.** 2ªed. Rio de Janeiro: Brasport. 2007. 248p.

DEITEL, Harvey M. et al. **C# - Como Programar.** São Paulo: Makron Books, 2003. 1153p.

DIONYSIO, Rosana C.C.; TAVARES, Nelson S.; JÚNIOR, Carlos I.S. **C#: Introdução a Programação Orientada a Objetos**. 1ªed. Taquaritinga: AgBook, 2013. 55p.

FILIPEFLOP. **Módulos.** 2015. Disponível em: http://www.filipeflop.com/modulos-ct-41d96. Acesso em 27 setembro de 2015.

FARREL, Adrian. **A Internet e seus Protocolos** : Uma análise comparativa. Rio de Janeiro: Elsevier, 2005. 608p.

FOROUZAN, Behrouz A. **Comunicação de Dados e Redes de Computadores**. 4ªed. São Paulo: McGraw-Hill, 2008. 1132p.

TRUCHSESS, Norbert. **Arduino UIP.** 2014. Disponível em: https://github.com/ntruchsess. Acesso em 2 outubro de 2015.

IGUS, Plastics for longer life. **Calha articulada para espaços reduzidos.** 2015. Disponível em: http://www.igus.com.br/wpck/1767/overview_E2micro Acesso em 12 agosto de 2015.

IGUS, Plastics for longer life. **Elementos de Fixação.** 2015. Disponível em: http://www.igus.com.br/wpck/2915/designing_strainrelief?C=BR&L=pt. Acesso em 30 agosto de 2015.

KRETCHEU, Paulo. **Protocolos TCP e UDP**. 2013. Disponível em http://kretcheu.com.br/videos/protocolos-tcp-e-udp/. Acesso em 30 de setembro de 2015.

LAMB, Frank. **Automação Industrial na Prática** - Série Tekne . Porto Alegre: AMGH, 2015 . 376p.

LANGBRIDGE, James A. **Arduino Sketches**: Tools and Techniques for Programming Wizardry. Indianopolis: JonhWiley& Sons, 2015. 444 p.

LS, Group. **LS Automotive Wire & Cable:** Environment-friendly cable for the safety or you car. 2005. Disponível em: http://m.lscns.co.kr/catalogs/LS_Automotive_Wire&Cable_EN_05.pdf. Acesso em 11 agosto de 2015.

MINITAB, Inc. **Experimentos Taguchi.** 2015. Disponível em: http://support.minitab.com/pt-br/minitab/17/topic-library/modeling-statistics/doe/taguchi-designs/taguchi-designs/. Acesso em 13 novembro de 2015.

MCROBERTS, Michael. **Arduino Básico.** 1ª Ed. São Paulo: Novatec .2011. 456p.

MICROCHIP, Technology. **ENC28j60:** Stand-Alone Ethernet Controller with SPI Interface. 2012. Disponível em: http://ww1.microchip.com/downloads/en/DeviceDoc/39662e.pdf . Acesso em 27 setembro de 2015.

MICROSOFT, Company. **Visual Studio.** 2015. Disponível em: https://www.visualstudio.com/. Acesso em 22 outubro de 2015

MONK, Simon. **Programação com Arduino II**: passos avançados com sketches. Porto Alegre: Bookman, 2015. 260p.

MURHAMMER, Martin W. et al. **TCP/IP - Tutorial e Técnico.** São Paulo: Makron, 2000. 690p.

PALFINGER, Group. **O Grupo Palfinger.** 2015. Disponível em https://www.palfinger.com/pt-BR/bra/sobre-a-empresa. Acesso em 25 agosto de 2015.

PRESCOTT, Preston. **SQL Para Iniciantes.** Madrid: Babelcube, 2015. 30p.

ROSS, Julio. **Rede de Computadores.** Rio de Janeiro: Antenna, 2008. 148p.

ROSÁRIO, João Maurício. **Automação Industrial**. São Paulo: Baraúna, 2009. 515p.

SANTOS, J. Neves dos. **Condutores e Cabos de Energia.** 2005. Disponível em https://web.fe.up.pt/~jns/material_didatico/APONTAMENTOS_CABOS%20de%20Energia_FINAL.pdf. FEUP. Acesso em 11 novembro de 2015.

SCHWAAB, Marcio; PINTO José C. **Análise de Dados Exerimentais, volume II**: planejamento e experimentos. Rio de Janeiro: E-papers, 2011. 514p.

SPURGEON, Charles F. **Ethernet** : O guia definitivo.Rio de Janeiro: Campus, 2000. 478p.

SQLite. **SQLite.** 2015. Disponível em https://www.sqlite.org Acesso em 25 outubro de 2015.

TANEMBAUM, Andrew S. **Rede de computadores**. 4ª ed. Rio de Janeiro: Elsevier, 2003. 887p.

WIZINET. Internet Offload Processor Provider. 2015. **W5100.** Disponível em: http://www.wiznet.co.kr/product-item/w5100/. Acesso em 27 setembro de 2015.

APÊNDICE A - CONTROLADOR ARDUINO MEGA 2560

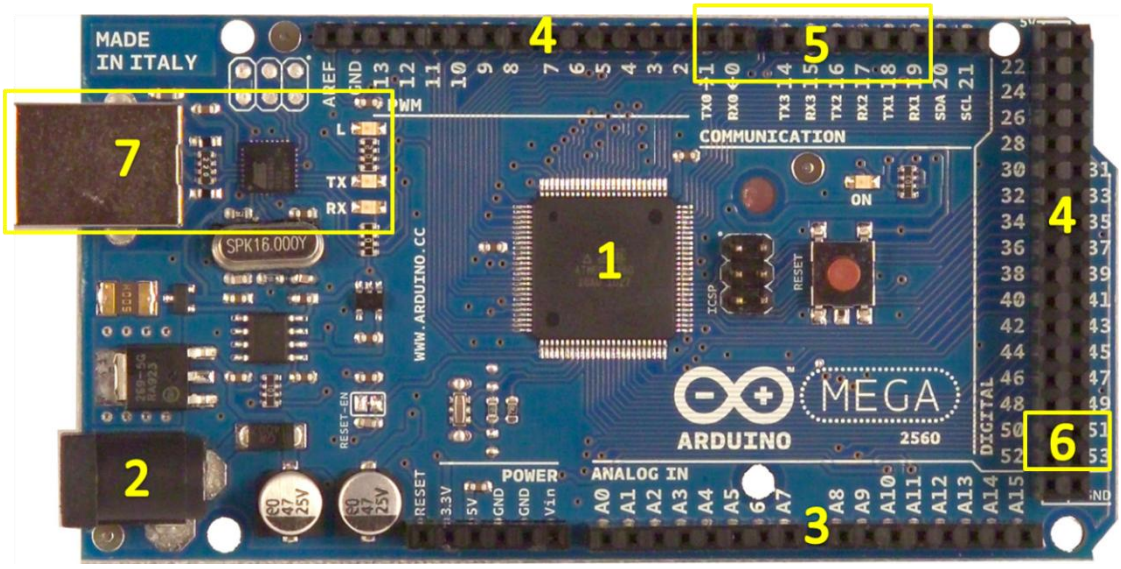
Conforme Monk (2015), originalmente projetado pela empresa italiana de mesmo nome, o Arduino é uma plataforma de desenvolvimento baseada em um microcontrolador produzido pela empresa ATMEL. Com a sua rápida programação, e adição de módulos específicos, o controlador se torna versátil, permitindo a construção de diversos modelos de sistemas para transferência de dados, detecção de grandezas físicas e interfaces homem-máquina (IHMs). Demais características como baixo custo, *hardwares* e *softwares* livres (*open source*), durabilidade e facilidade de implementação permitem a sua ampla difusão em projetos eletrônicos.

Durante a automação da máquina de ensaios foi empregado o modelo Mega 2560, apresentado na Figura 28.

Conforme informado em Arduino (2015), o modelo apresentado possui um microcontrolador ATMega2560 (1) com 256KB de memória *flash*, 8KB de SRAM (*Static Random Access Memory*) e 4KB de EEPROM (*Electrically Erasable Programmable Read-Only Memory*). Para interface, são disponibilizadas 16 entradas analógicas (3) e 54 entradas ou saídas digitais (4), configuradas via *software,* dentre as quais cinco podem funcionar como interrupções.

Figura 28- Arduino Mega 2560

Legenda: microcontrolador (1), alimentação (2), entradas analógicas (3), entradas/saídas digitais (4), conexão UART (5), conexão SPI (6), UART padrão(7).



Fonte: adaptado de Arduino (2015)

Dentre as portas digitais disponíveis, é possível a implementação de quatro canais de comunicação serial através da conexão UART (5) e um canal por meio da conexão SPI (6). Para permitir a compilação de softwares enviados por computadores, a primeira conexão UART é convertida em USB (7) por meio do componente ATmega16U2.

Atualmente, a máquina de ensaios de flexão utiliza 44 entradas/saídas digitais do controlador, sendo representadas na Tabela 15 conforme a numeração do próprio Arduino.

Tabela 15- Utilização das portas do controlador

Portas digitais	**Utilização**
0 e 1	Botões ON e OFF
2 e 3	Sensores S1 e S2 (interrupção)
7 e 5	Bobinas Q1 e Q2
6	Sinal de Teste
8,9,13	Interface serial com display LCD
11,12,22 a 49	Detecção dos condutores
50 a 54	Módulo SD (SPI)

Fonte: o Autor (2015)

APÊNDICE B - MÓDULO ETHERNET W5100 E A SUA COMUNICAÇÃO

Para possibilitar a comunicação de dados via rede Ethernet é utilizado o módulo W5100, dispositivo projetado para uso específico em plataformas Arduino, sendo denominado como *shield*. Os shields se diferem dos demais módulos por seguirem os mesmos padrões de conexões do controlador, possibilitando que a interface seja realizada apenas conectando o *shield* sobre o Arduino, descartando o uso de cabos (Arduino, 2015).

O módulo W5100 opera em 3,3V (alimentado pelo controlador) e a interface SPI com o Arduino Mega 2560 é realizada conforme a Tabela 16. Para melhor conexão, as ligações do pino 11 a 13 do módulo podem ser executadas através do método ICSP (*In-Circuit Serial Programming*), o que libera fisicamente as entradas de mesmo nome do Arduino para demais funções. O pino 4 é utilizado para ativar a memória SD, enquanto o pino 10 o módulo Ethernet, sendo alternados as suas habilitações através do *software* presente no controlador e conforme a necessidade de uso.

Tabela 16- Comunicação do módulo com o Arduino

Pino	Conexão	Função
GND	0V	Alimentação negativa do módulo
3V3	3.3V	Alimentação positiva do módulo
4 - CS(Chip Select)	Porta 4 do Arduino	Habilita a memória SD
10 - CS (*Chip Select*)	Porta 10 do Arduino	Habilita o módulo Ethernet
13 - SCK (*Serial Clock*)	Porta 52 do Arduino	Gerencia a velocidade de transmissão
12 - MOSI (*Master Out Slave In)*	Porta 51 do Arduino	Canal do mestre para o escravo
11 - MISO(*Master In Slave Out*)	Porta 50 do Arduino	Canal do escravo para o mestre

Fonte: Arduino (2015)

Na figura 29 é exibido o módulo com a identificação de seus principais componentes e pinos de ligação. O conector ICSP (2) é localizado sob a placa, e facilita a conexão com a ICSP do Arduino Mega 2560.

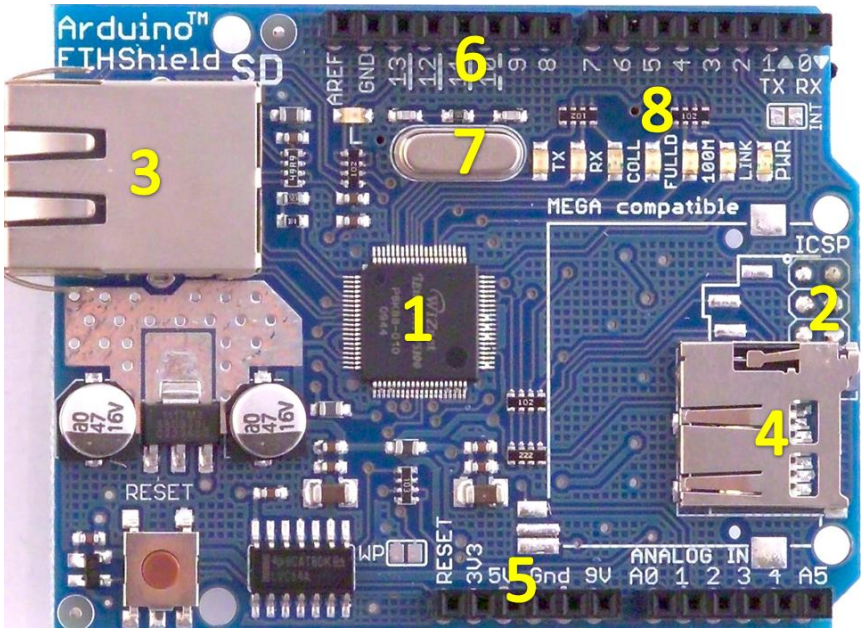
O módulo W5100 possui sete LED's (8) responsáveis por indicar a presença das seguintes situações da rede:

- PWR - Alimentação do módulo
- LINK - Comunicação com a rede Ethernet
- 100M - Conexão a 100Mbps
- FULLD - Método *full duplex*

- COLL - Colisão
- TX e RX - Transmissão e recepção de dados

Figura 29- Módulo Ethernet ENC28J60

Legenda: microcontrolador W5100 (1), interface ICSP (2), conector RJ-45(3), entrada para cartão de memória SD (4), pinos de alimentação (5), interface SPI (6), cristal 25MHz, indicadores de comunicação (8) .



Fonte:adaptado de ArtofCircuits (2015).

APÊNDICE C - BANCO DE DADOS UTILIZADOS NO PROJETO

O armazenamento e gerenciamento dos dados dos ensaios de flexão é efetuado pelo aplicativo através do sistema de banco de dados SQLite. Para o seu funcionamento são desenvolvidas seis tabelas que permitem o gerenciamento dos registros. A criação das estruturas do banco e as funções de controle das informações são implementadas pela linguagem SQL, onde os principais comando empregados é demonstrado na Tabela 17

Tabela 17- Principais comandos SQL utilizados

Comando	Função
CREATE TABLE	Cria a tabela conforme nome e campos informados
INSERT INTO	Insere novos dados nos campos da tabela de destino
UPDATE	Atualiza os dados existentes na tabela de destino
DELETE FROM	Exclui os dados referentes a um endereço
PRIMARY KEY	Determina o campo utilizado para endereçamento dos dados
SELECT	Seleciona os dados de uma tabela ou linha

Fonte:Costa (2007).

A seguir são apresentados as tabelas que compõe a estrutura do sistema de dados aplicado ao projeto, divididas conforme os tipos de informações armazenadas.

A Figura 30 exibe a tabela que armazena as características dos condutores cadastrados, utilizados nos testes de flexão. O número ID é um registro único utilizado no endereçamento de cada item adicionado. Os demais campos correspondem a características específicas do condutor .

Figura 30- Tabela de cadastro de condutores

RecNo	ID	DESCR	SEC	CLASS	ESPISOL	TIPO	FABRI	NORMA	VMAX	TMAX	FILM	MATISOL	MATCOND
Click here to define a filter													
1	1	Cabo Auto Flexível 0,5mm	0,5	5	1,4	Automotivo	CondVolt	DIN72551 / ISO6722	300	100	19	PVC	Cobre
2	2	Cabo automotivo 2,5mm	2,5	5	2,7	Automotivo	Multisul	DIN72551 / ISO6722	300	100	50	PCV	Cobre

Fonte:o Autor(2015)

A Figura 31 exibe a tabela que armazena as características dos ensaios cadastrados pelo usuário e os dados de quantidade de ciclos e condutores rompidos obtidos nos ensaios de flexão. Da mesma maneira que a tabela dos condutores, um número de identificação único (IDT) é utilizado para o endereçamento dos testes. Também é armazenado um vetor de 39 posições (VETRUP), o qual é utilizado pelo Arduino para a identificação dos fios que se encontram rompidos.

Figura 31- Tabela de cadastro de ensaio

R...	IDT	RAIO	CURSO	TEMP	CICMAX	DATA	CICLOS	CONDR	VETRUP
Click here to define a filter									
1	7	28	480	20	50000	03/11/2015 11:51:00	40000	7	0000000000000000000000000001101110011
2	8	28	480	20	50000	03/11/2015 11:38:00	30000	11	0100000000100000010000000001010111111
3	9	35	480	20	50000	02/11/2015 21:51:00	39000	4	0000000000000000110000000000000000011
4	10	35	480	20	120000	03/11/2015 11:51:00	110000	11	0000000000001000000101110100110000001011

Fonte:o Autor(2015)

A tabela da Figura 32 armazena dados dos condutores que se encontram em teste. A identificação de cada registro é realizado através do código do ensaio a qual pertencem (IDTESTE) e o número de identificação presente na máquina (IDMAQ).

Figura 32- Tabela de condutores em teste

RecNo	IDTESTE	IDCOND	IDMAQ	SEC	CICLOS	FITA
Click here to define a filter						
98	10	1	17	0,5	(null)	1
99	10	1	18	0,5	(null)	1
100	10	1	19	0,5	(null)	1
101	10	1	20	0,5	(null)	1
102	10	1	21	0,5	(null)	1
103	10	1	22	0,5	87637	1
104	10	1	23	0,5	104170	1
105	10	1	24	0,5	74689	1
106	10	1	25	0,5	(null)	1
107	10	1	26	0,5	64443	1
108	10	1	27	0,5	(null)	1
109	10	1	28	0,5	(null)	1
110	10	1	29	0,5	50730	1
111	10	2	30	2,5	82098	0

Fonte:o Autor(2015)

A quinta tabela utilizada (Figura 33) armazena os dados de conexão da rede desenvolvida entre o computador e o controlador da máquina.

Figura 33- Tabela de registros da rede

RecNo	IDREDE	PORTA	IP
Click here to define a filter			
1	1	5555	169.254.187.15

Fonte:o Autor(2015)

A última tabela desenvolvida (Figura 34) armazena os resultados estatísticos das rupturas, gerados conforme filtros selecionados pelo usuário. Identificado através de um código único (IDRES), cada registro possui as opções a qual foram gerados e os resultados dos cálculos de média, desvio padrão, valor máximo e valor mínimo.

Figura 34- Tabela de resultados

RecNo	IDRES	OPC1	VAL1	OPC2	VAL2	OPC3	VAL3	MEDIA	DPADRAO	VMAX	VMIN
Click here to define a filter											
1	2	Raio	28	Secção	2,5	Ensaio	7	6934,25	8023,36	23399	0
2	1	Raio	28	Condutor	2			11621,1666666667	6875,83	23399	1992
3	4	Raio	35	Condutor	2			63257	18841	82098	44416
4	6	Raio	35	Fita	true			82877,8571428571	19163,74	106953	50730
5	3	Raio	35					78517,6666666667	20762,05	106953	44416
6	8	Raio	35	Secção	2,5			63257	18841	82098	44416
7	10	Raio	28					3321,47619047619	6407,94	23399	1
8	11	Raio	28	Condutor	2			11621,1666666667	6875,83	23399	1992
9	20	Ensaio	8					11997,1428571429	6333,1	19579	1992
10	22	Ensaio	8	Fita	true			11997,1428571429	6333,1	19579	1992
11	234	Raio	28					3321,47619047619	6407,94	23399	1
12	23	Fita	true	Secção	2,5	Raio	28	5810,58333333333	7576,37	23399	0

Fonte:o Autor(2015)

APÊNDICE D - PROGRAMA DO MICROCONTROLADOR ARDUINO

Segue o *software* presente no microcontrolador Arduino. As linhas demarcadas em cinza representam os códigos desenvolvidos nesse projeto. As demais linhas foram implementadas durante o trabalho de estágio. As referências das funções e lógicas presentes no código encontram-se em Arduino (2015) e McRoberts (2011).

```
//Carrega bibliotecas
#include <SD.h>                                                    //Biblioteca SD card
#include <SPI.h>                                  //Biblioteca de comunicação com SD card
#include "U8glib.h"                                        //Biblioteca do display grafico
#include <Ethernet.h>                                  //Biblioteca da comuniçao Ethernet

//Define Variáveis utilizadas na rede Ethernet
byte mac[] = { 0xAB, 0xCD, 0x12, 0x34, 0xFF, 0xCA };            // Define o numero MAC
int porta = 5555;
IPAddress ip(169, 254, 187, 15);                 //Define o endereços IP do controlador
EthernetServer server(porta);                 // Define um servidor, atribuindo a porta
EthernetClient client;                                             // Define um cliente

char flag = 0;                                                  // Recebe dados da rede
int vldata = 0;                                               // Recebe chars da rede
String strnum = "";                                           // Recebe números da rede

//Define Variáveis utilizadas na maquina
int pinSD = 4;                                    // Pino de habilitação do modulo Sd
int pinEthernet = 10;                           // Pino de habilitação do moduloEthernet
int tam_vet = 39;                                             // tamanho vetor de fios
int pri_ent = 11;                                    // entrada primeiro fio conectado
int quant_fio = 10;                                   //quantidade de fios p/ cabo
int ON = 0;                                                                    // S3
int OFF = 1;                                                                   // S4
int S1 = 2;                                              // Sensor_recuo (interrupcao)
int S2 = 3;                                              // Sensor_avanco(interrupcao)
int Q1 = 7;                                                 // Solenoide de avanco
int Q2 = 5;                                                  // Solenoide de recuo
int ST = 6;                                                // Sinal de teste (O11)
int SD_CS = 53;                                               // Slave select - SD
int SD_MOSI = 51;                                          // Master Out Slave In - SD
int SD_MISO = 50;                                          // Master in SLave Out - SD
int SD_SCK = 52;                                              // Serial clock - SD
int DI_E = 8;                                                  // Enable - Display
int DI_RW = 9;                                             // Read_Writer - Display
int DI_RS = 13;                                          // Register Select - Display
int Q_ant = 0;                                  // Armazenar estado anterior solenóides
int total_fios = 0;                       // Armazena quantidade total de fios rompidos
int cabo = 0;                                   // Armazena tipo de cabo do fio rompido
long ciclos = 0;                                                    // Armazena ciclos
int fio_num = 0;                                  // Armazena o numero do fio rompido
long ciclosMax = 1000000;                       // Armazena quantidade de ciclos máximos
int Qant = 0;                                   // Armazena estado anterior das bobinas
int estado_fios[39];                                  // Vetor armazenar estado fios
bool fc = false;                                         // Habilita contagem ciclos
bool auxreset = false;                                              // Habilita reset
```

```
bool ft = false;                                    // Habilita inicio do processo
bool sair_reset = true;                           // Habilita tela  do menu de reset
bool fd = true;                                   // Habilita atualizacao do display
bool ftf = true;                                    // Habilita atualizacao dos fios
bool testenovo = false;                           // Indica o inicio de um novo teste
char filename[8];                                          //Armazena nome do arquivo
int file_num = 1;                                        //Armazena numero do arquivo
String ext_name = ".txt";                              //Armazena extensÃ£o do Arquivo
String test_name;                                 //Armazena o nome final do arquivo
File arq_teste;                          // Arquivo armazenamento de dados dos testes
File arq_estados;        // Arquivo armazenamento de estados anteriores do dispositivo

//Configura  display
U8GLIB_ST7920_128X64 u8g(DI_RS, DI_RW, DI_E, U8G_PIN_NONE);        //Comunicação com o
                                                          //display (Enable, RW, RS, RESET)
Define e inicializa entradas e saídas
void setup()
{
    pinMode(pinSD, OUTPUT);                          //Saída para seleção do modulo SD
    pinMode(pinEthernet, OUTPUT);               //Saída para seleção do modulo Ethernet
    pinMode(Q1, OUTPUT);                                          // Saida solenoide Q1
    pinMode(Q2, OUTPUT);                                          // Saida solenoide Q2
    pinMode(ST, OUTPUT);                                        // Saida sinal de teste
    pinMode(SD_CS, OUTPUT);                          // Saida para comunicao com SD card
    pinMode(ON, INPUT_PULLUP);     //Entrada botÃ£o S3 - utilizamos o resistor pull-up
    pinMode(OFF, INPUT_PULLUP);                                   // Entrada para botao S4
    pinMode(11, INPUT); // C1_01              //Configura entrada para os cabos e fios
    pinMode(12, INPUT); // C1_02
    pinMode(22, INPUT); // C1_03
    pinMode(23, INPUT); // C1_04
    pinMode(24, INPUT); // C1_05
    pinMode(25, INPUT); // C1_06
    pinMode(26, INPUT); // C1_07
    pinMode(27, INPUT); // C1_08
    pinMode(28, INPUT); // C1_09
    pinMode(29, INPUT); // C1_10
    pinMode(30, INPUT); // C2_01
    pinMode(31, INPUT); // C2_02
    pinMode(32, INPUT); // C2_03
    pinMode(33, INPUT); // C2_04
    pinMode(34, INPUT); // C2_05
    pinMode(35, INPUT); // C2_06
    pinMode(36, INPUT); // C2_07
    pinMode(37, INPUT); // C2_08
    pinMode(38, INPUT); // C2_09
    pinMode(39, INPUT); // C2_10
    pinMode(40, INPUT); // C3_01
    pinMode(41, INPUT); // C3_02
    pinMode(42, INPUT); // C3_03
    pinMode(43, INPUT); // C3_04
    pinMode(44, INPUT); // C3_05
    pinMode(45, INPUT); // C3_06
    pinMode(46, INPUT); // C3_07
    pinMode(47, INPUT); // C3_08
    pinMode(48, INPUT); // C3_09
    pinMode(49, INPUT); // C3_10

    memset(estado_fios, (int)0, sizeof(int) * tam_vet);       // Inicializa vetor de
                                                                 // estados dos fios
                                                                 //Inicializa Saidas
    digitalWrite(Q2, LOW);                                      // Desliga bobina Q2
    digitalWrite(Q1, LOW);                                      // Desliga bobina Q1
```

```
    digitalWrite(ST, LOW);                                  // Desliga sinal de teste

    // Inicializa cartao de memoria e cria arquivos
    AtivarSD();                                                 //Ativa modulo SD
    SD.begin(pinSD);                                        // Inicializa cartao SD
    if (SD.exists("Estad.txt") == false)
    {                                                       // Se arquivo nao existe
        arq_estados = SD.open("Estad.txt");                    // Cria novo arquivo
        arq_estados.close();                                     // Fecha o arquivo
        Grava_estados();          // Inicializa arquivo de estados com dados zerados
    }
    //Carregar estados inicias do arquivo de estados
    arq_estados = SD.open("Estad.txt", FILE_READ);        // Abre o arquivo de estados
    if (arq_estados)
    {                                                                //Se arquivo abriu
        file_num = (arq_estados.parseInt());                // Carrega numero do teste
        ciclos = (arq_estados.parseInt());              // Carrega quantidade de ciclos
        Qant = (arq_estados.parseInt());                       // Carrega estado bobina
        for (int i = 0; i < tam_vet; i++)
        {                                                   // Carrega vetor de ruptura
            estado_fios[i] = (arq_estados.parseInt());  // Carrega caracteres do vetor
            total_fios += estado_fios[i];       // Carrega quantidade de fios rompidos
        }
        arq_estados.close();                                        // Fecha o arquivo
    }
    test_name = file_num + ext_name;              // configura nome do arquivo de teste
    test_name.toCharArray(filename, sizeof(filename));     // configura o tipo do arq.

    Define interrupções
    attachInterrupt(1, Recuo, RISING);              // interrupção 1 quando S0 acionado
    attachInterrupt(0, Avanco, RISING);             // interrupção 1 quando S1 acionado

    //Configura parÃ£metros de imagem do display
    if (u8g.getMode() == U8G_MODE_R3G3B2)                  // Configura cor da escrita
        u8g.setColorIndex(255);                                                 // Branca
    else if (u8g.getMode() == U8G_MODE_GRAY2BIT)              // Configura intensidade
        u8g.setColorIndex(1);                                    // Máxima intensidade
    else if (u8g.getMode() == U8G_MODE_BW)                 // Retornar modo de exibição
        u8g.setColorIndex(1);

    // Inicia a rede
    AtivarEthernet();                                       // Ativa o modulo de Ethernet
    Ethernet.begin(mac, ip);             // Cria a rede atraves dos enderecos MAC e IP
}

//Programa principal
void loop()
{
    if (fd == true)
    {                           // Atualiza tela principal quando nao conectado a rede
        display_ciclos();                          // Projeta tela principal (sub-rotina)
        Grava_estados();                                                // Grava estados
        fd = false;
    }
    client = server.available();           // Recebe um cliente que conectou no server
   //EM REDE
    while (client && client.connected())
    {                                                       // Conectado a um cliente
        LerDadosPC();             // Atualiza tela e envia dados de ciclos para a rede
        if (fd)
        {
            display_ciclos();                   // Projeta tela principal (sub-rotina)
```

```
            Grava_estados();                                                // Grava estados
            client.print(String(ciclos) + "C");     // Envia dados de ciclos pela rede
            if (ftf)
            {                                       // Verifica se houve novas rupturas
                client.print(String(total_fios) + "T");// Envia total de ciclos a rede
                ftf = false;              // Desabilita atualizacao de total de ciclos
            }
            fd = false;                 // Desabilita atualizacao do display e da rede
        }
       switch (vldata){                                 // Verifica as flags recebidas
             case 'M': ciclosMax = strnum.toInt();                  // Armazena ciclos
máximos
                 break;
      case 'I': if(file_num != strnum.toInt()){         // Verifica se nº novo teste é
                  testenovo = true;      // Habilita flag para inicio de um novo teste
                  file_num = strnum.toInt();           // Atribui numero do novo teste
                  test_name = file_num + ext_name;        // Configura nome do arquivo
                  test_name.toCharArray(filename, sizeof(filename));// tipo do arquivo
                }
                break;
      case 'D': client.stop();              // Encerra a comunicação com o cliente(PC)
                break;
      case 'C': if(testenovo){         // Atribui os ciclos apenas se é um novo teste
                 ciclos = strnum.toInt();                    // Atribui os ciclos
                 testenovo = false;                     // Desativa flag de novo teste
                }
                break;
      case 'L': Ligar();                          // Subrotina para habilitar a máquina
                break;
      case 'P': Desligar();                      // Subrotina para desabilitar a máquina
                break;
      case 'R': if(testenovo){                           // Verifica se é um novo teste
                  ResetarON();                                  // Reseta os parametros
                  for (int i = 0; i<tam_vet ; i++ ) {      // Carrega vetor de ruptura
                    if(strnum[i] == '1'){                 // Atribui rupturas ao vetor
                    estado_fios[i] = 1;                         // de estados dos fios
                    total_fios++;                       // Recebe total de fios rompidos
                    }
                  }
                }
                break;
     }
     if ( digitalRead(ON) == LOW && ft == false )
     {                                                          // Verifica inicio dos testes
        Ligar();                                    // Subrotina para habilitar a máquina
     }
     if (  digitalRead(OFF) == LOW && ft == true)
     {                                                          // Veridica pausa dos testes
        Desligar();                                // Subrotina para desabilitar a máquina
     }
        Resetar();                                 // Subrotina de teste de reset via botão
    }
    //SEM REDE
    if (digitalRead(ON) == LOW && ft == false)
    {                                                           // Testa estado dos botoes
        Ligar();                                          // Subrotina para habilitar teste
    }
    if ((digitalRead(OFF) == LOW) && ft == true)
    {                                                           // Testa estado dos botoes
        Desligar();                                 //Subrotina para desabilitar a maquina
    }
```

```
    Resetar();                                  //Subrotina de teste de reset via botao
}

// Movimentacao de cilindro -  Interrupcaoes
void Avanco()
{                                                   // Interrupcao de avanco(S1 acionado)
    if (ft == true)
    {                                                       // Testa se testes habilitados
        digitalWrite(Q2, LOW);                                      // Desiga Solenoide Q2
        Delay(500);                                                  // Tempo de comutacao
        digitalWrite(Q1, HIGH);                                       // Liga Solenoide Q1
        fc = true;                                          // Habilita contagem dos ciclos
        Qant = 0;
    }
}
void Recuo()
{                                                   // Interrupcao de recuo (S2 acionado)
    if (ft == true)
    {                                                       // Testa se testes habilitados
        digitalWrite(Q1, LOW);                                      // Desiga Solenoide Q1
        Delay(500);                                                  // Tempo de comutacao
        digitalWrite(Q2, HIGH);                                       // Liga Solenoide Q2
        if (fc == true)
        {                                           // Testa se contagem de ciclos habilitada
            Teste_cabos();        // Realiza o teste de ruptura em todos os condutores
            fd = true;                              // Habilita atualizaÃ§Ã£o do display
            fc = false;                               // Desabilita incremento de ciclos
            Qant = 1;
        }
    }
}
// Grava estados
void Grava_estados()
{                                       // Grava dados atuais no arquivo de estados
    AtivarSD();                                             // Ativa modulo de memoria SD
    SD.remove("Estad.txt");                     // Deleta o arquivo de estados existente
    arq_estados = SD.open("Estad.txt", FILE_WRITE);             // Cria um novo arquivo
    if (arq_estados)
    {                                                                   // Arquivo aberto
        arq_estados.println(file_num);                      // Armazena numero do teste
        arq_estados.println(ciclos);                                 // Armazena ciclos
        arq_estados.println(Qant);                          // Armazena estado das bobinas
        for (int i = 0; i < tam_vet; i++)
        {
            arq_estados.println(estado_fios[i]);          // Armazena vetor de ruptura
        }
        arq_estados.close();                                             // Fecha arquivo
    }
    AtivarEthernet();                                       // Ativa modulo Ethernet
}

// Realiza testes
void Teste_cabos()
{                                                       // Testa a ruptura nos cabos
    digitalWrite(ST, HIGH);                 // Envia sinal alto nos condutores em teste
    Delay(500);
    ciclos++;                                           // Incrementa contador de ciclos
    for (int i = pri_ent; i < 50; i++)
    {                                                   // Efetua  varredura nos 30 fios
        if ((i <= 12 || i >= 22) && estado_fios[i - pri_ent] == 0)
        {
            if (digitalRead(i) == LOW)
```

```
            {                                                   // Testa se o fio esta rompido
                if (i < 30)
                {                                           // Determina o cabo onde rompeu o fio
                    cabo = 1;                                                       // Cabo 1
                }
                else
                {
                    if (i < 40)
                    {
                        cabo = 2;                                                   // Cabo 2
                    }
                    else
                    {
                        cabo = 3;                                                   // Cabo 3
                    }
                }
                Grava_testes(i);  // Grava o os dados no arquivo de teste (sub-rotina)
                estado_fios[i - pri_ent] = 1; // Atualiza o vetor que armazena os fios
                total_fios++;                   // Incrementa numero de fios rompidos
                ftf = true;                   // Ativa atualização de rupturas a rede
            }
        }
    }
    digitalWrite(ST, LOW);                          // Desliga saida de sinal dos testes
    if (total_fios == 30 || ciclos >= ciclosMax)
    {                                                       // Testa condicao de termino
        ft = false;                                                 // Desabilita testes
        digitalWrite(Q1, LOW);                                          // Desliga Q1
        digitalWrite(Q2, LOW);                                          // Desliga Q2
    }
}

//Grava arquivo de testes
void Grava_testes(int i)
{                                                               // Grava dados dos testes
    AtivarSD();                                                         // Ativa modulo SD
    arq_teste = SD.open(filename, FILE_WRITE);       // Abre arquivo em modo de edicao
    if (i <= 12)
    {                                                   // Verifica numero do fio rompido
        fio_num = i - pri_ent;
    }
    else
    {
        fio_num = i - ((quant_fio * cabo) + quant_fio);
    }
    if (arq_teste)
    {                                               // Arquivo aberto
        arq_teste.print(cabo);                                  // Numero do cabo
        arq_teste.print((fio_num));                               // Numero dofio
       arq_teste.print("     ");                  // Grava informacaoes dos ciclos
        arq_teste.println(ciclos);                          // Numero de ciclos
        arq_teste.close();                                        // Fecha arquivo
    }
    AtivarEthernet();             // Ativa Ethernet e envia dados de ruptura pela rede
    client.print(String(cabo) + String(fio_num) + String(ciclos) + "R");
    Delay(10);                                          // Tempo para envio dos dados
}

//Telas apresentadas no display grafico
void display_ciclos(void)
{                                                                   // Tela principal
    u8g.firstPage();                                  // Seleciona primeira tela
```

```
    do
    {
        u8g.setFont(u8g_font_unifont);                      // Define fonte da escrita
        u8g.drawStr(30, 10, "PALFINGER");                       // Define caracteres
        u8g.setPrintPos(0, 25);                                    // Define posição
        u8g.print(file_num, 10);
        u8g.drawStr(0, 40, "Ciclos: ");                         // Define caracteres
        u8g.setPrintPos(60, 40);                                   // Define posição
        u8g.print(ciclos, 10);                                      // Mostar ciclos
        u8g.drawStr(0, 55, "Fios: ");                           // Define caracteres
        u8g.setPrintPos(60, 55);                                   // Define posicao
        u8g.print(total_fios, 10);                        // Mostra quantidade de fios
    } while (u8g.nextPage());
}
void display_reset(void)
{                                                                   // Tela de reset
    u8g.firstPage();                                    // Seleciona primeira tela
    do
    {
        u8g.setFont(u8g_font_unifont);                      // Define fonte da escrita
        u8g.drawStr(0, 10, "Presione Verde");                   // Define mensagem
        u8g.drawStr(0, 22, "para resetar");                     // Define mensagem
        u8g.drawStr(0, 35, "ou Vermelho ");                     // Define mensagem
        u8g.drawStr(0, 47, "para sair");                        // Define mensagem
    } while (u8g.nextPage());
}
void Delay(int x)
{                                                 // Delay para uso nas interrupcoes
    for (int i = 0; i <= x; i++)
    {
        delayMicroseconds(1000);
    }
}
void AtivarSD()
{                                                 // Subrotina para ativar modulo SD
    digitalWrite(pinEthernet, HIGH);
    digitalWrite(pinSD, LOW);
}
void AtivarEthernet()                          // Subrotina para ativar  modulo Ethernet
{
    digitalWrite(pinSD, HIGH);
    digitalWrite(pinEthernet, LOW);
}
void Ligar()
{                                                                   // Habilita testes
    while (digitalRead(ON) == LOW) ;
    if (Qant == 0)
    {
        digitalWrite(Q2, LOW);                                  // Desiga Solenoide Q2
        digitalWrite(Q1, HIGH);                                   // Liga Solenoide Q1
    }
    if (Qant == 1)
    {
        digitalWrite(Q1, LOW);                                  // Desiga Solenoide Q1
        digitalWrite(Q2, HIGH);                                   // Liga Solenoide Q2
    }
    flag = ' ';                                       // Limpa dados recebidos pela rede
    ft = true;                                                      // Habilita testes
}
void Desligar()
{
    while (digitalRead(OFF) == LOW) ;
```

```
    digitalWrite(Q1, LOW);                                          // Desliga Q1
    digitalWrite(Q2, LOW);                                          // Desliga Q2
    Grava_estados();                        // Gravar dados atuais no arquivo de estados
    flag = ' ';                                         // Limpa dados recebidos pela rede
    ft = false;                                                     // Desabilita testes
}
void Resetar()
{                                                                   // Reset atraves dos botoes
    if (digitalRead(OFF) == LOW && digitalRead(ON) == LOW && ft == false)
    {
        Delay(100);                                          // Continua apos soltar botoes
        while (digitalRead(OFF) == LOW || digitalRead(ON) == LOW) ;
        sair_reset = false;                                  // Habilita a tela de reset
        while (sair_reset == false)
        {
            display_reset();                      // Projeta tela de reset (sub-rotina)
            if ((digitalRead(ON) == LOW && digitalRead(OFF) == HIGH))
            {                                                                   // Botao ON
                while (digitalRead(ON) == LOW) ;    // Continua apos soltar o botao ON
                ciclos = 0;                                  // Zera contador de ciclos
                total_fios = 0;                         // Zera contado de fios rompido
                fd = true;                              // Habilita atualização do display
                memset(estado_fios, (int)0, sizeof(int) * tam_vet);      // Zera vetor
                file_num++;                             // Incrementa numero do arquivo
                sair_reset = true;                       // Desabilita tela de reset
            }
            if (digitalRead(OFF) == LOW && digitalRead(ON) == HIGH)
            {                                                                   // Botao off
                while (digitalRead(OFF) == HIGH) ;     // Continua apos soltar o botao
                fd = true;                              // Habilita atualizacao do display
                sair_reset = true;                       // Desabilita tela de reset
            }
        }
    }
}
void ResetarON()
{
    ciclos = 0;                                                 // Zera contador de ciclos
    total_fios = 0;                                       // Zera contado de fios rompido
    memset(estado_fios, (int)0, sizeof(int) * tam_vet);       // Zera vetor de estados
    test_name.toCharArray(filename, sizeof(filename));       //configura o tipo do arq
    Grava_estados();                        // Gravar dados atuais no arquivo de estados
    vldata = 0;                                       // Limpa dados recebidos pela rede
}
void LerDadosPC()
{                                                   // Subrotina para ler dados da rede
    strnum = "";                                 // Limpa vetores de recebimento de numero
    vldata = 0;                                                                 // e chars
    if (client.available() > 0)
    {                                                   // Verifica se possui dados na rede
        while (true)
        {                                                   // Entra em laço de recebimento
            if (client.available() > 0)
            {                                               // Verifica se possui dados na rede
                flag = client.read();                                       // Le os dados
                if (flag == 'F')
                {                       // Verifica se recebeu flag de termino da string
                    break;                                  // Sai do laco de leitura
                }
                if (flag >= 65)
                {                                     // Verifica se recebeu uma letra
                    vldata = flag;                                  // Atribui a chars
```

```
                }
                else
                {
                    strnum += flag;                // Caso contrario, atribui a numeros
                }
            }
        }
    }
}
```

APÊNDICE E - PROGRAMA DO APLICATIVO DO PC

Devido a extensão do programa em C# desenvolvido, é apresentado somente a parte responsável pela comunicação com o Arduino. Os códigos referem-se a classe Comunicar e o formulário de Supervisão de Ensaios.

```
// CLASSE COMUNICAR
using System;                           // Carrega bibliotecas utilizadas nos comandos
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Windows.Forms;

namespace Projeto_TCC
{
    public class Comunicar
    {
        private string dados;                                   // Resposta do servidor
        public Socket Socket;                                   // Canal de comunicação
        private TelaSupervisao tela;                             // Manipular interface
        public bool EstaConectado = false;                          // Verifica conexão
        private static readonly short BUFFER_SIZE = 2048;// Tamanho da string recebida

        public Comunicar(TelaSupervisao t)                                  // Construtor
        {
            tela = t;                                       // Atribui tela de supervisão
        }

        public void Conectar(string ip, int porta)                  //Conectar  canal
        {
            try
            {
                Socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
                                    ProtocolType.Tcp);       // Configura socket em TCP
                var ipe = new IPEndPoint(IPAddress.Parse(ip), Convert.ToInt32(porta));
                                                  // Cria um endereço de comunicação
                this.Socket.Connect(ipe);                             // Efetua a conexão
                this.EstaConectado = true;                 // Indica a conexão efetuada
                tela.conexao = true;                 // Indica a conexão ao formulário
            }
            catch
            {
                MessageBox.Show("Falha na transferência de dados, verifique a
                                conexão", "Erro // Exibe msg caso ocorra erro na conexão
            }
        }

        public void LerDados()                              // Leitura dos dados do canal
        {
            var thread = new Thread(this.LerArduino);     // Cria seguento de execução
            thread.Start();                              // para leitura dos dados na rede
        }

        private void LerArduino()                  // Ler os dados enviado pelo Arduino
        {
            var buffer = new byte[BUFFER_SIZE];                // Recebimento dos bytes
```

```
    var bytesRec = 0;                                       // Número de bytes
    var result = 0L;                                      //  Ciclos recebidos
    string strnum = "";                                //  Dados recebidos sem o flag
    while (this.EstaConectado)                           // Laço enquanto conectado
    {
        try
        {
            bytesRec = Socket.Receive(buffer);      // Recebe os dados do canal
            dados = Encoding.ASCII.GetString(buffer, 0, bytesRec);
                                                 // Converte para string em ASCII
            strnum = dados.Substring(0, dados.Length - 1); // Retira flag
            result = Convert.ToInt64(strnum);
            if (this.Ruptura(dados))        // Se dados correspondem a ruptura
            {
                tela.AtualizarRuptura(result);              //Envia para a tela
            }
            else if (this.TotalRuptura(dados))               //Se ruptura total
            {
                tela.AtualizarTotal(result);              // Envia para a tela
            }
            else if (this.Ciclos(dados))     // Se dados correspondem a ciclos
            {
                tela.AtualizarCiclos(result);             // Envia para a tela

            }
        }
        catch (Exception)                                          // Exceção
        {
            break;
        }
    }
}

private bool Ruptura(string dados)          // Verifica se sinal é uma ruptura
{
    return dados.EndsWith("R");                   // Verifica se possui flag R
}
private bool TotalRuptura(string dados)  // Verifica se é de total de rupturas
{
    return dados.EndsWith("T");                   // Verifica se possui flag T
}
private bool Ciclos(string dados)             // Verifica se sinal é de ciclos
{
    return dados.EndsWith("C");                   // Verifica se possui flag C
}
public void EnviarComando(string comando)    // Subrotina de envio de comandos
{
    try
    {                             // Codifica a string em uma sequencia de bytes
        var data = Encoding.ASCII.GetBytes(comando+"F");
        Socket.Send(data);         // Envia os dados pelo canal de comunicação
    }
    catch
    {
        MessageBox.Show("Falha na transferência de dados, verifique conexão",
                        "Erro");             //Mensagem caso ocorra algum erro
    }
}
public void Desconectar()             // Subrotina de desconexão com o Arduino
{
    if (EstaConectado)                        // Verifica se comunicação existe
    {
```

```
                EnviarComando("D");  // Envia flag para que o Arduino cesse a comunic.
                Thread.Sleep(10);                          // Tempo para o envio da flag
                Socket.Close();                      // Canal de comunicação é encerrado
                EstaConectado = false;         // informa o encerramento da comunicação
                tela.conexao = false  // Informa o encerramento da conexão para a tela
            }
            MessageBox.Show("Desconectado com sucesso", "Aviso");       // Msg de erro
        }
    }
}


// Formulário de Supervisão de Ensaios
using System;                                          // Carrega bibliotecas utilizadas
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Threading;
using System.Windows.Forms;
using Microsoft.VisualBasic;              // Biblioteca utilizada na classe Interation

namespace Projeto_TCC
{
    public partial class TelaSupervisao : Form
    {
        private BancodeDados banco;                          // Classe de banco de dados
        private Condutor fio;                                        // Classe condutor
        private Teste ensaio;                                         // Classe ensaio
        List<Condutor> listaCond { get; set; }               // Coleção de condutores
        private Comunicar comunicar;                              // Classe comunicar
        public bool conexao = false;                               // Indica conexão
        public delegate void setTempCallback(long dados);
        private bool reset = false;                               // Indica novo teste
        private bool attCampos = true;                          // Indica att de campos
        string[] rede = new string[2];                          // Vetor para porta e IP
        string ip;                                                    // Endereço IP

        public TelaSupervisao()                                              // Construtor
        {
            InitializeComponent();                           // Inicializa o formulário
            ensaio = new Teste();                                 // Instancia ensaio
            fio = new Condutor();                               // Instancia condutor
            banco = new BancodeDados();                   // Instancia banco de dados
            rede = banco.CarregarRede();             // Carrega IP e porta armazenados
            supPorta.Text = rede[0];                          // Mostra Porta na tela
            supIP1.Text = rede[1].Substring(0, 3);               // Mostra IP na tela
            supIP2.Text = rede[1].Substring(4, 3);               // Mostra IP na tela
            supIP3.Text = rede[1].Substring(8, 3);               // Mostra IP na tela
            supIP4.Text = rede[1].Substring(12, 2);              // Mostra IP na tela
        }

        public void AtualizarCiclos(long dados)   // Att ciclos recebidos pelo Arduino
        {
            var dadosStr = Convert.ToString(dados);        // Converte dados em string
            if (dadosStr != supCiclos.Text)          // Se diferente dos dados na tela
            {
                if (InvokeRequired)                              // Tratamendo da Thread
                    this.Invoke(new setTempCallback(AtualizarCiclos), new object[] {
                              dados });
                else
                {
                    supCiclos.Text = dadosStr;           // Atribui  de ciclos na tela
                }
```

```
        }
    }
    public void AtualizarRuptura(long dados) // Att ruptura recebidos pelo Arduino
    {
        var dadosStr = Convert.ToString(dados);        // Converte dados em string
        if (InvokeRequired)                                // Tratamento da Thread
            this.Invoke(new setTempCallback(AtualizarRuptura), new object[] {
                        dados });
        else
        {
            string nomeBox = dadosStr.Substring(0, 2);       // Recebe a ID do fio
            int numBox = Convert.ToInt32(nomeBox);           // Converte ID em int
            ((TextBox)this.Controls.Find("supCi" + nomeBox,
               true).FirstOrDefault()).Text = dadosStr.Substring(2,
               dadosStr.Length - 2);    // Encontra textbox referente ao ID do fio
            if (numBox <= 11)           // Tratamento do vetor que indica ruptura
            {
                ensaio.vetRup.Remove(numBox - 10, 1);          // Se inferior a 11
                ensaio.vetRup.Insert(numBox - 10, '1');
            }
            else
            {
                ensaio.vetRup.Remove(numBox - 1, 1);              // Superior a 11
                ensaio.vetRup.Insert(numBox - 1, '1');
            }
        }
    }
    public void AtualizarTotal(long dados)           // Att total ruptura recebidos
    {
        var dadosStr = Convert.ToString(dados);              // Converte em string
        if (dadosStr != supCiclos.Text)              // Se dados diferente da tela
        {
            if (InvokeRequired)                              // Tratamento da Thread
                this.Invoke(new setTempCallback(AtualizarTotal), new object[] {
                            dados });
            else
            {
                supCondutores.Text = dadosStr;    // Mostra total rupturas na tela
            }
        }
    }
    private void btSupIniciar_Click(object sender, EventArgs e)   // Botão Iniciar
    {
        try
        {
            comunicar.EnviarComando("L");        // Envia flag que habilita testes
            btSupIniciar.Enabled = false;              // Desabilita botao Iniciar
            btSupParar.Enabled = true;                    // Habilita botao Parar
            btSupDesconect.Enabled = false;        // Desabilita botao Desconectar
        }
        catch
        {
            MessageBox.Show("Falha na transferência de dados, verifique a
                            conexão", "Erro");  // Aviso de erro de comunicação
        }
    }
    private void btSupConect_Click(object sender, EventArgs e)   // Botão Conectar
    {
            ip = supIP1.Text + "." + supIP2.Text + "." + supIP3.Text + "." +
                 supIP4.Text;                                          // Atribui IP
            banco.GravarRede(supPorta.Text, ip);         // Armazena dados da rede
            if (comunicar == null) comunicar = new Comunicar(this);    //Instancia
```

```
            comunicar.Conectar(ip, int.Parse(supPorta.Text));   // Conecta Arduino
            if (conexao)                                        // Se conexão efetuada
            {
                if (reset)                                  // Se indicado novo teste
                {
                    comunicar.EnviarComando("I" + supNum.Text); //Envia num teste
                    comunicar.EnviarComando("R" + ensaio.vetRup.ToString());
                                                          // Envia vetor de rupturas
                    comunicar.EnviarComando("C" + ensaio.ciclos);//Envia qt ciclos
                    reset = false;                          // Desativa novo teste
                }
                comunicar.EnviarComando("M" + supCicloMax.Text);    // Envia qt máx
                Thread.Sleep(1000);                 // Tempo para o envio dos dados
                comunicar.LerDados();           // Habilia leitura de dados da rede
                btSupIniciar.Enabled = true;             // Habilita botao Iniciar
                btSupConect.Enabled = false;          // Desabilita botao Conectar
                btSupDesconect.Enabled = true;       // Habilita botao Desconectar
                btSupReset.Enabled = false;     // Desabilita botao Carregar Teste
                supPorta.ReadOnly = true;               // Desabilita edição porta
                supIP1.ReadOnly = true;                    // Desabilita edição IP
                supIP2.ReadOnly = true;                    // Desabilita edição IP
                supIP3.ReadOnly = true;                    // Desabilita edição IP
                supIP4.ReadOnly = true;                    // Desabilita edição IP
        }
    }
    private void btSupParar_Click(object sender, EventArgs e)       // Botao Parar
    {
        comunicar.EnviarComando("P");    // Envia flag de Desabilitação dos testes
        btSupIniciar.Enabled = true;                    // Habilita botão Iniciar
        btSupParar.Enabled = false;                     // Desabilita botão Parar
        btSupDesconect.Enabled = true;               // Habilita botão Desconectar
        GravarDados();                        // Grava os dados obtidos dos ensaios
    }
    private void FecharTela(object sender, FormClosingEventArgs e)  // Subrotina
                                                          para fechamento da tela
    {
        if (conexao) {                                      // Se comunicação ativa
            if (MessageBox.Show("Deseja desconectar e sair?", "Aviso",
                MessageBoxButtons.YesNo, MessageBoxIcon.Question)
                == DialogResult.Yes)             // Tela para verificação da saída
            {
                comunicar.Desconectar(); // Encerramento aceito, desconecta a rede
            }
            else
            {
                e.Cancel = true;                    // Caso não, permanece na tela
            }
        }
    }
    private void btSupDesconect_Click(object sender, EventArgs e // Botão Descone.
    {
        comunicar.Desconectar();                              // Desconecta a rede
        if (comunicar.Socket.Connected == false)           // Se rede desconectada
        {
            btSupParar.Enabled = false;                     // Desabilita botão Parar
            btSupIniciar.Enabled = false;                 // Desabilita botão Iniciar
            btSupDesconect.Enabled = false;            // Desabilita botão Desconectar
            btSupConect.Enabled = true;                    // Habilita botão Conectar
            btSupReset.Enabled = true;                  // Habilita botão Novo teste
            supPorta.ReadOnly = false;                   // Habilita edição da porta
            supIP1.ReadOnly = false;                       // Habilita edição de IP
            supIP2.ReadOnly = false;                       // Habilita edição de IP
```

```
        supIP3.ReadOnly = false;                        // Habilita edição de IP
        supIP4.ReadOnly = false;                        // Habilita edição de IP
    }
}
private void btSupReset_Click(object sender, EventArgs e)  // Botão Novo teste
{
    try
    {
       String input = Interaction.InputBox("Informe o número do teste", "Novo
                      teste", "", 500, 300);      // Tela para do num do teste
        if (input != "")                                    // Se numero válido
        {
            attCampos = true;                       // Habilita att de campos
            supNum.Text = input;                 // Insere num informado na tela
            reset = true;            // Habilita carregamento de um novo teste
        }
    }
    catch
    {
            MessageBox.Show("Valor incorreto", "Erro"); // Msg se num inválido
  }
}
private void IndicarRuptura(object sender, EventArgs e)      // Cor da ruptura
{
    TextBox cicloBox = new TextBox();                       // Atribui TextBox
    cicloBox = sender as TextBox;        // Recebe textbox atribuida no evento
    string numBox = cicloBox.Name.Substring(5, 2);                // ID do fio
    TextBox secBox = (TextBox)this.Controls.Find("supSe" + numBox,
                      true).FirstOrDefault();       // Recebe Textbox da Secção
    TextBox idBox = (TextBox)this.Controls.Find("supID" + numBox,
                     true).FirstOrDefault();          // Recebe textbox do fio
    if (!String.IsNullOrEmpty(cicloBox.Text) && Convert.ToInt32(cicloBox.Text)
        > 0)                                // Se ruptura identificada (ciclos>0)
    {
        secBox.BackColor =  Color.LightCoral;    // Modifica cor para vermelho
        idBox.BackColor = Color.LightCoral;
        cicloBox.BackColor = Color.LightCoral;
    }
    else                                          // Caso nao modifica para verde
    {
        secBox.BackColor = Color.LightGreen;
        idBox.BackColor = Color.LightGreen;
        cicloBox.BackColor = Color.LightGreen;
    }
}
public void ReceberValoresTeste(string num)   // Receber val do banco de dados
{
    supNum.Text = num;                      // Recebe valor do numero do teste
    supCicloMax.Text = ensaio.maximoCiclos.ToString();    // recebe ciclos max
    supPorta.Text = rede[0];                          // Recebe numero da porta
    supIP1.Text = rede[1].Substring(0, 3);                          // Recebe IP
    supIP2.Text = rede[1].Substring(4, 3);                           // Recebe IP
    supIP3.Text = rede[1].Substring(8, 3);                          // Recebe IP
    supIP4.Text = rede[1].Substring(12, 2);                         // Recebe IP
    supCondutores.Text = ensaio.quantRupturas.ToString(); /   / Recebe tot rup
    supCiclos.Text = ensaio.ciclos.ToString();       // Recebe total de ciclos
    for (int i = 0; i < listaCond.Count; i++)             // Recebe condutores
     {
         ((TextBox)this.Controls.Find("supID" + listaCond[i].idmaq.ToString(),
          true).FirstOrDefault()).Text = listaCond[i].id.ToString();
                                                    // Recebe ID do condutor
```

```
                ((TextBox)this.Controls.Find("supCi" + listaCond[i].idmaq.ToString(),
                 true).FirstOrDefault()).Text = listaCond[i].ciclos.ToString();
                                                        // Recebe ciclos de ruptura
                ((TextBox)this.Controls.Find("supSe" + listaCond[i].idmaq.ToString(),
                 true).FirstOrDefault()).Text = listaCond[i].seccao;
                                                                  // Recebe secção
                ((CheckBox)this.Controls.Find("supFita" +
                 listaCond[i].idmaq.ToString(),  true).FirstOrDefault()).Checked =
                 Convert.ToBoolean(listaCond[i].fita);                 // Recebe Fita
            }
        }

        private void supNum_TextChanged(object sender, EventArgs e)  // Alterado o cód
        {
            if (attCampos)                    // Verifica se att dos campos está ativa
            {
                attCampos = false;    // Desativa att, impedindo ativar essa subrotina
                string num = supNum.Text;             // Atribui o código atual a var.
                LimparCaixasDeTexto(this.Controls);  // Limpa todas as caixas de texto
                ensaio = banco.CarregarEnsaio(num);// Carrega ensaio do banco de dados
                listaCond = banco.CarregarCondTestes(num);   // Carrega fios  do banco
                rede = banco.CarregarRede();                     // Carrega IP e porta
                ReceberValoresTeste(num);          // Rotina para mostrar valores tela
                attCampos = true;                             // Ativa a att dos campos
                btSupConect.Enabled = true;                  // Ativa o botão Conectar
            }
        }
        private void GravarDados()              // Grava informações no banco de dados
        {
            List<Condutor> listAux = new List<Condutor>();     // Cria coleção de fios
            ensaio.id = Convert.ToUInt32(ensaio.TestarCampo(supNum.Text));       // id
            ensaio.ciclos = Convert.ToUInt32(ensaio.TestarCampo(supCiclos.Text));
                                                        // Recebe numero de ciclos
            ensaio.quantRupturas =
Convert.ToUInt32(ensaio.TestarCampo(supCondutores.Text));  // Recebe total de rupturas
            if (listaCond != null)                  // Se lista de condutores não nula
            {
                for (int i = 0; i < listaCond.Count; i++)
                {                    // Laço de leitura dos condutores rompidos no teste
                    uint ciclosFio = Convert.ToUInt32(fio.TestarCampo(((TextBox)
                    this.Controls.Find ("supCi" + (10 + i).ToString(),
                    true).FirstOrDefault()).Text));           // Verifica ciclo do cabo

                    if (listaCond[i].ciclos != ciclosFio)            // Se novos dados
                    {
                        listAux.Add(new Condutor       // lista auxiliar de condutores
                        {
                            idmaq = listaCond[i].idmaq,  // Armazena id do fio rompido
                            ciclos = ciclosFio,         // Ciclod e ruptura atualizado
                        });
                    }
                }
            }
            banco.CadastrarSupervisao(ensaio, listAux);  // Armazena do banco de dados
        }
        private void LimparCaixasDeTexto(Control.ControlCollection cc)
        {                                                   // Limpa caixa de textos
            foreach (Control ctrl in cc)                // Laço para todos as textBox
            {
                TextBox tb = ctrl as TextBox;
                if (tb != null)                             // Se texbox contem dados
                    tb.Text = "";                                       // Limpa TexBox
```

```
                else
                    LimparCaixasDeTexto(ctrl.Controls);
            }
        }
    }
}
```