CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA
UNIDADE DE PÓS-GRADUAÇÃO, EXTENSÃO E PESQUISA
MESTRADO PROFISSIONAL EM GESTÃO E TECNOLOGIA
EM SISTEMAS PRODUTIVOS

CLAUDEMIR SANTOS PINTO

DESENVOLVIMENTO DE UMA PLATAFORMA PARA AQUISIÇÃO E TRATAMENTO DE DADOS EXPERIMENTAIS

São Paulo
Abril/2015

CLAUDEMIR SANTOS PINTO

DESENVOLVIMENTO DE UMA PLATAFORMA PARA AQUISIÇÃO E TRATAMENTO DE DADOS EXPERIMENTAIS

Dissertação apresentada como exigência parcial para a obtenção do título de Mestre em Gestão e Tecnologia em Sistemas Produtivos do Centro Estadual de Educação Tecnológica Paula Souza, no Programa de Mestrado Profissional em Gestão e Tecnologia em Sistemas Produtivos, sob a orientação do Prof. Dr. Francisco Tadeu Degasperi.

São Paulo
Abril/2015

CLAUDEMIR SANTOS PINTO

DESENVOLVIMENTO DE UMA PLATAFORMA PARA AQUISIÇÃO E TRATAMENTO DE DADOS EXPERIMENTAIS

Prof. Dr. Francisco Tadeu Degasperi

Prof. Dr. Humber Furlan

Prof. Dr. Claudemir Stelatti

São Paulo, 07 de abril de 2015

Agradeço a minha família e amigos pelo suporte e incentivo durante a elaboração deste trabalho e ao meu orientador por todo conhecimento transmitido e direcionamento para a conclusão desta dissertação.

AGRADECIMENTOS

À Deus por me dar esta oportunidade e condições de realizar este mestrado.

À Sheila, minha esposa e aos meus filhos Ivo e João por me darem ânimo para continuar nos momentos difíceis.

Aos meus colegas da Fatec Guaratinguetá por todo incentivo dado durante este período.

“Você pode encarar um erro como uma
besteira a ser esquecida ou como um resultado
que aponta uma nova direção”
(Steve Jobs)

RESUMO

PINTO, C. S. **Desenvolvimento de uma plataforma para aquisição e tratamento de dados experimentais** 100 p. Dissertação (Mestrado Profissional em Gestão e Tecnologia em Sistemas Produtivos). Centro Estadual de Educação Tecnológica Paula Souza, São Paulo, 2015.

A utilização de uma interface computacional no processo de aquisição e tratamento de dados experimentais confere maior credibilidade e agilidade na geração de resultados. Existem diversas soluções no mercado para atender a essa demanda, porém, com custo elevado e que não atendem à necessidades específicas de cada laboratório de pesquisas. Sendo assim, o presente trabalho tem por objetivo avaliar por meio de experimentos, a viabilidade do desenvolvimento de um sistema para aquisição e tratamento de dados utilizando tecnologia *Open Source,* e de baixo custo envolvendo a plataforma Arduino e o Ambiente de Desenvolvimento Integrado Lazarus. A metodologia usada consta de pesquisa bibliográfica e de um estudo de caso que tem como evidência experimentos realizados no laboratório de Tecnologia do Vácuo da Fatec São Paulo. Após o desenvolvimento da interface computacional, foram realizados experimentos com aquisição de dados experimentais provenientes de um termopar e um sensor de membrana capacitiva, onde percebeu-se que os resultados obtidos são equivalentes aos coletados por outros dispositivos de exibição de resultados. Quanto à ferramenta de análise de dados, esta possibilita a importação dos dados adquiridos e imediata geração de gráfico com determinação da curvas de ajuste.

Palavras-chave: Aquisição de dados, Tratamento de dados, Arduino, Lazarus

ABSTRACT

PINTO, C. S. **Developing a platform for acquisition and processing of experimental data** 100 p. Dissertation (Professional Master in Management and Technology in Production Systems). Centro Estadual de Educação Tecnológica Paula Souza, São Paulo, 2015.

Using a computational interface in the acquisition and processing of experimental data process gives greater credibility and flexibility in generating results. There are several solutions on the market to meet this demand, but with high cost and do not meet the specific needs of each research laboratory. Therefore, this study aims to evaluate through experiments the feasibility of developing a system for acquisition and processing of data using Open Source technology and low cost involving Arduino Platform and Integrated Development Environment Lazarus. The methodology consists of a literature review and case study evidence whose experiments in the Laboratory Vacuum Technology of Fatec São Paulo. After the development of computer interface experiments with acquisition of experimental data from a thermocouple sensor, and a capacitive membrane which were carried out it was noticed that the results are equivalent to those listed by other devices for displaying results. Regarding data analysis tool that enables the import of acquired and immediate generation graphics with determining the curves adjustment data.

Keywords: Data acquisition, Data processing, Arduino, Lazarus

LISTA DE QUADROS

LISTA DE FIGURAS

LISTA DE SIGLAS

DAS	Data-Acquisition System
DC	Direct Current – corrente contínua
IDE	Integrated Development Environment
LTV	Laboratório de Tecnologia do Vácuo
PWM	Pulse-Width Modulation
SGBD	Sistema Gerenciador de Banco de Dados
SQL	Structured Query Language
TI	Tecnologia da Informação

SUMÁRIO

1 INTRODUÇÃO

Por muito tempo, os laboratórios de pesquisas científicas e tecnológicas de diversas áreas produziram um volume relativamente pequeno de informações devido à grande quantidade de dados gerados em cada experimento e pela dificuldade de adquiri-los de forma confiável e tratá-los manualmente.

Em um ambiente onde se produz um volume grande de dados analógicos através de experimentos em laboratório, faz-se necessária a utilização de ferramentas modernas para a aquisição e manipulação desses dados em formato digital, como a utilização de computadores e software especializado.

Ao longo do tempo, o computador tem se consolidado como ferramenta adequada para apoiar pesquisas de laboratório, devido à sua grande capacidade de armazenamento e processamento, por possuir sistemas de visualização e recursos de conectividades com outros equipamentos o que propicia um maior ganho de produtividade. Em todas as áreas da ciência, seja em Exatas, Humanas ou Biológicas, o computador tem sido um aliado indispensável para o avanço da tecnologia.

Percebe-se também nos dias atuais, uma tendência mundial em se utilizar experimentos de forma virtual, o que significa que o modelo de laboratório experimental também está acompanhando as mudanças de comportamento causadas pela evolução da Tecnologia da Informação. Assim, muitos experimentos produzidos em laboratórios são acompanhados por recursos computacionais, cujos resultados são gerados de forma mais rápida, com maior confiabilidade e disponibilidade. Como exemplo desta tendência, pode-se citar o projeto PhET Interactive Simulations, da Universidade do Colorado (PhET, 2015), que oferece simulações baseadas em ciências e matemática para alunos e professores. As simulações são oferecidas para execução de forma online e também podem ser baixadas e instaladas em computadores pessoais. Todas as simulações são *open source* e o desenvolvimento é custeado por patrocinadores para que os alunos e professores usufruam gratuitamente.

Nota-se a necessidade de prover as instituições de ensino de recursos para o desenvolvimento das pesquisas e construção do conhecimento. A procura por soluções de baixo custo é uma exigência quando se fala em aquisição de equipamentos e software para as empresas e instituições de ensino, especialmente o ensino público. Essa procura torna-se

ainda mais importante quando as soluções disponíveis no mercado são caras e não atendem por completo as necessidades das instituições.

A proposta deste trabalho é demonstrar a partir do desenvolvimento de um sistema de aquisição e tratamento de dados que é possível criar soluções para laboratórios de pesquisa tanto na área de ensino como na indústria sem dispender de muito recurso financeiro. Para esse desenvolvimento foram utilizados os sensores termopar para medir temperatura e o sensor de membrana capacitiva para medir pressão, porém, qualquer tipo de sensor-transdutor que gere como resposta uma tensão elétrica pode ser utilizado. Como interface de aquisição foi utilizado o microcontrolador Arduino UNO e para o desenvolvimento do software, a linguagem de programação Lazarus. O experimento foi realizado no Laboratório de Tecnologia do Vácuo (LTV) da Fatec São Paulo.

O sistema funciona da seguinte forma: uma vez tendo sido elaborado o arranjo para o transdutor, o hardware coleta os sinais elétricos, em geral na faixa de 0 V até 5 V, converte esses sinais em valores digitais, que são traduzidos na grandeza que está sendo medida, e em seguida esses valores são armazenados em arquivo. Posteriormente, a mesma interface de software que coletou os dados realiza o tratamento dos mesmos, gerando gráficos e realizando determinação de curvas de ajustes, necessárias a um bom e completo trabalho experimental dentro das áreas de engenharia, química e física, e ainda, podendo ser usado em medicina e biologia.

Complementando o estudo, foram realizados testes no equipamento desenvolvido permitindo um estudo de suas capacidades e limitações.

1.1 Questão de pesquisa

Como desenvolver uma plataforma de baixo custo para aquisição e tratamento de dados experimentais ?

Além dessa questão, também se pesquisa:

- Como montar uma interface de hardware para a aquisição de dados experimentais a partir de uma placa Arduino ?
- Que recursos a Interface de Desenvolvimento Integrado Lazarus oferece para o desenvolvimento de um sistema de tratamento de dados.

1.2 Objetivo

Objetivo geral:

Este trabalho de pesquisa tem como objetivo principal o projeto e a construção de uma plataforma de baixo custo para aquisição e tratamento de dados experimentais. O circuito eletrônico utilizado é *open source* (livre para ser copiado e utilizado para qualquer propósito), está disponível no mercado e tem custo acessível. A Interface de Desenvolvimento Integrado utilizada para criar o software de tratamento de dados também é *open source* e não tem custo para utilização. O trabalho desenvolvido está em criar uma interface computacional para adquirir dados experimentais, vindos de transdutores, e em seguida o tratamento dos mesmos por meio da geração de gráficos e determinação de curvas de ajustes.

Objetivos específicos:

1. Demonstrar como montar uma interface de *hardware* para capturar informações provenientes de um sensor/transdutor que gera resultados através de sinais elétricos. Atenção especial deve ser dada aos níveis de tensão elétrica para a aquisição de dados experimentais.
2. Desenvolver uma ferramenta de software para aquisição, armazenamento e tratamento de dados de forma automática e simultânea ao experimento.
3. Demonstrar a eficiência, eficácia e exequibilidade do sistema proposto em relação ao processo manual de aquisição e tratamento de dados.
4. Demonstrar as capacidades e analisar as limitações do sistema.

1.3 Estrutura do trabalho

Para melhor entendimento, este trabalho foi dividido em 6 partes principais, cada uma com seu conteúdo, conforme detalhado na figura 1:

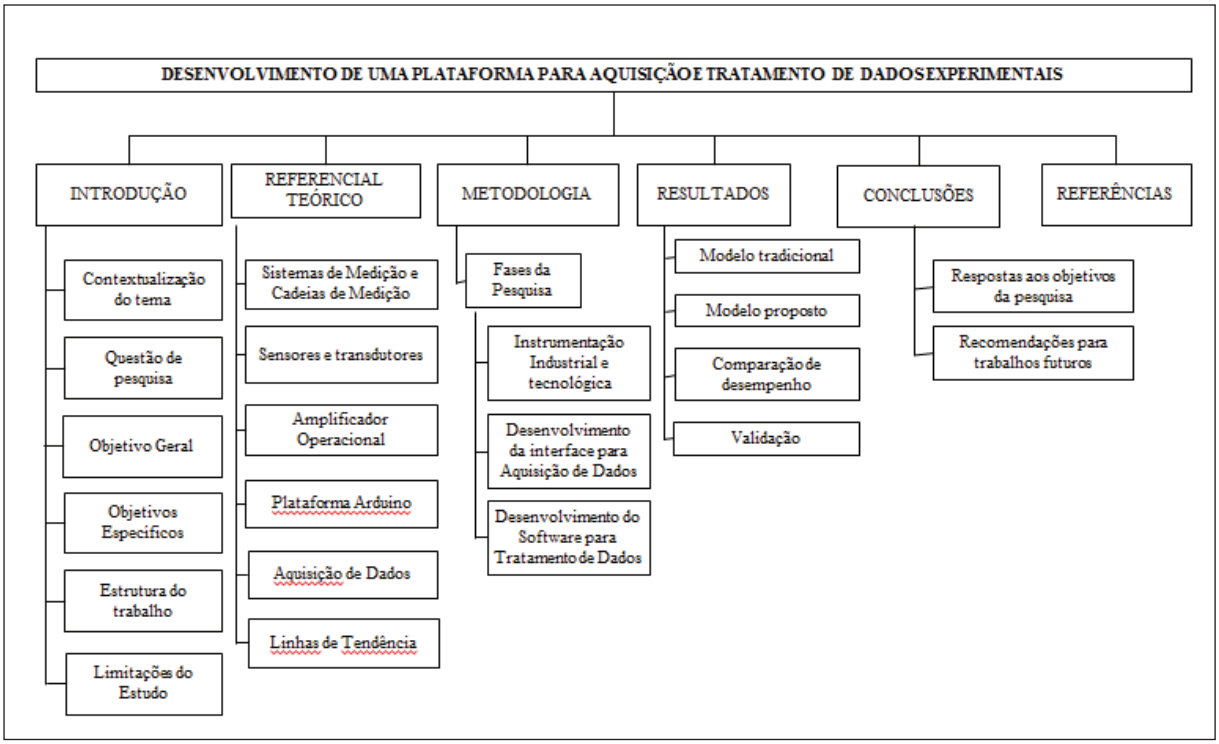


Figura 1: Estrutura do trabalho

1.4 Limitações do estudo

Este trabalho aborda o desenvolvimento de uma ferramenta de aquisição e análise de dados experimentais, oriundos de sensores transdutores que gerem uma tensão elétrica entre 0 e 5 volts. Nesta pesquisa são testados os sensores termopar tipo K e medidor Pirani, cada qual tendo seu arranjo elaborado conforme suas características físicas. Qualquer outro tipo de sensor a ser utilizado com o microcontrolador ATmega328, da plataforma de modelagem Arduino necessita ser estudado para se projetar e montar o arranjo experimental. O software construído para tratamento de dados, após adquirir os dados por importação de arquivo do tipo txt, gera um gráfico de pontos com a possibilidade de exibir a linha de tendência, além de exibir a equação do modelo. Com esta pesquisa, pretende-se demonstrar a viabilidade da construção e utilização desta plataforma.

2 FUNDAMENTAÇÃO TEÓRICA

Para fundamentar o desenvolvimento desta pesquisa, serão abordados os seguintes tópicos:

- Sistemas de Medição e Cadeias de Medição
- Sensores e Transdutores
- Amplificador Operacional
- Plataforma Arduino
- Aquisição e análise de dados
- Linhas de Tendências
- Linguagem de programação Lazarus

2.1 Sistemas de Medição e Cadeias de Medição

Segundo FIGLIOLA e BEASLEY (2007), “uma medição é o ato de atribuir um valor específico à uma variável física”. Muitos dos testes realizados em laboratório resultam de medições de valores e comparações destes com valores de referência. Em outros casos, estes são simplesmente observados, pois descrevem o comportamento do que está sendo analisado.

RÚBIO (2000) afirma que medição é o processo de atribuição de números a propriedades de objetos ou eventos do mundo real e assim descrevê-los, ou ainda, a comparação da quantidade variável desconhecida com um padrão definido para cada tipo de quantidade, implicando então em um tipo de escala. O Quadro 1 descreve 5 tipos de medidas e suas respectivas descrições segundo Rúbio (2000).

De acordo com VUOLO (1992), uma grandeza física experimental pode ser entendida como qualquer grandeza física, cujo valor numérico é determinado a partir de um conjunto de dados experimentais.

Conforme TORREIRA (2002), a medida elétrica é uma das técnicas modernas, com a qual podem ser resolvidos problemas na pesquisa em geral e, principalmente, aqueles referentes ao controle, avaliação e processos industriais.

Quadro 1–Tipos de medidas e suas respectivas descrições

Tipo de Medida	Descrição
Medida Nominal	Quando duas quantidades do mesmo tipo são comparadas para saber se são iguais. (Ex. duas cores, acidez de dois líquidos)
Medida Ordinal	Quando é necessário ter informação a tamanhos relativos. (Ex. Classificação por peso e altura de uma turma)
Medida em Intervalos	Quando se deseja uma informação mais especifica, envolve-se então uma certa escala, sem incluir pontos de referência ou zero. (Ex. no caso anterior usar a escala de metros e quilogramas)
Medidas Normalizadas	Define-se um ponto de referência e realiza-se a razão, dividindo cada medida pelo valor de referência, determinando as magnitudes relativas. (Ex. O maior valor obtido será 1, quando foi escolhido como referência o valor máximo medido).
Medidas Cardinais	O ponto de referência é comparado com um padrão definido. Assim todo parâmetro físico pode ser medido contra uma referência padrão, como o Sistema Internacional de medidas SI.

Fonte: Rubio(2000)

Quanto a sistema de medição e cadeia de medição, para que não haja confusão, é necessário saber que existem definições diferentes. Um sistema de medição é definido de acordo com o método utilizado para se efetuar a medição bem como o tipo de sinais utilizados durante o processo de medição e tratamento, ao passo que uma cadeia de medição relaciona os aspectos funcionais de cada elemento que nela está inserido.

2.1.1 Sistemas de Medição

Classificados de acordo com diversos critérios, os sistemas de medição podem, por exemplo, ser classificados como sistemas em malha aberta ou sistemas em malha fechada.

Num sistema de medição em malha aberta não existe normalmente nenhuma função de controle, limitando-se o sistema de medição apenas a apresentar e/ou gravar as variações do sinal de entrada. Na figura 2 pode-se ver uma representação de um sistema de medição em malha aberta.

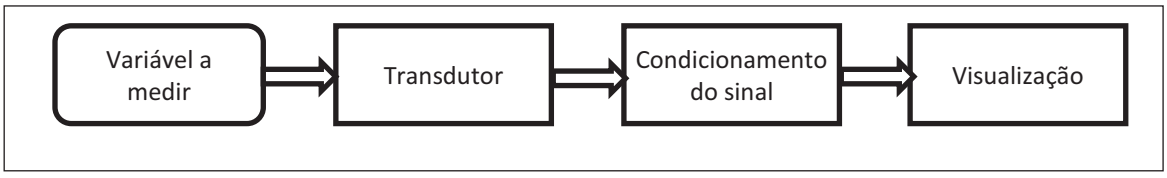


Figura 2: Sistema de medição de malha aberta
Fonte: Campilho (2000)

Já, num sistema de medição em malha fechada as medições efetuadas são utilizadas para controlar todo o sistema. O processo de realimentação, exemplificado na figura 3, é o que diferencia a configuração de um sistema de medição de malha aberta ou fechada.

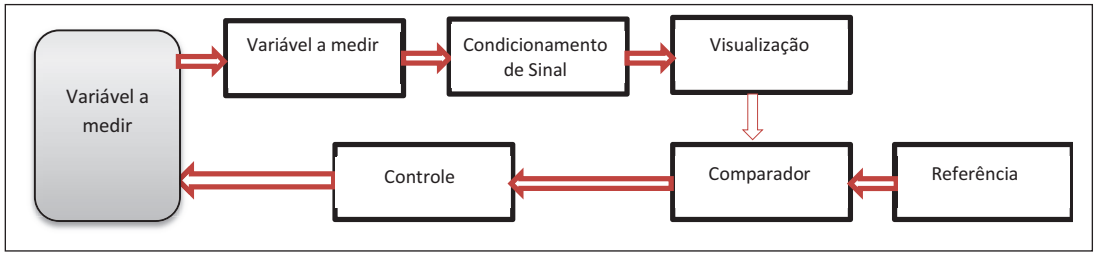


Figura 3: Sistema de medição em malha fechada
Fonte: Campilho (2000)

CAMPILHO (2000) sugere outra forma de classificar os sistemas de medição dividindo-os em sistemas digitais e sistemas analógicos. O avanço tecnológico da TI e da eletrônica tiveram um impacto maior nos sistemas digitais do que nos analógicos, tornando assim os sistemas de medição digitais mais complexos e com capacidade de processamento maior, usando computadores para servir de suporte à aquisição de dados, como exemplificado na figura 4:

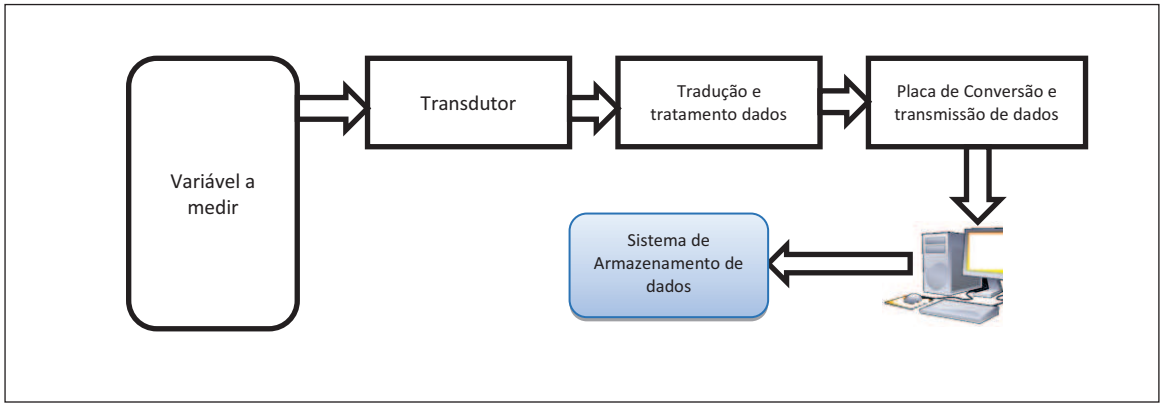


Figura 4: Sistema de medição digital
Fonte: Campilho (2000)

A placa de conversão e transmissão de dados efetua a conversão dos dados analógicos em dados digitais, ao mesmo tempo que os transmite para o sistema computadorizado que irá fazer o tratamento das medições realizadas e o encaminhamento dos dados para um sistema de visualização e armazenamento.

2.1.2 Cadeia de Medição

Segundo CAMPILHO (2000), uma cadeia de medição, na sua forma mais básica, é uma sucessão de elementos de um sistema de medição, que explica o trajeto do sinal de medição desde a sua entrada à sua saída. O sinal de entrada de um sistema de medição (ou de um instrumento de medição) é geralmente analógico, sendo habitualmente convertido para um

sinal de saída digital, devido ao expressivo aumento da utilização de sistemas computacionais nos processos de medição.

2.1.2.1 Organizando uma cadeia de medição

Uma cadeia de medição tem por objetivo a junção de vários componentes para que seja possível medir uma determinada grandeza de entrada. Dessa forma, uma cadeia de medição é geralmente formada por transdutores, circuitos de condicionamento, conversores Analógico/Digital e por sistemas de visualização e armazenamento de dados. Pode-se ver na figura 5 um exemplo de uma cadeia de medição.

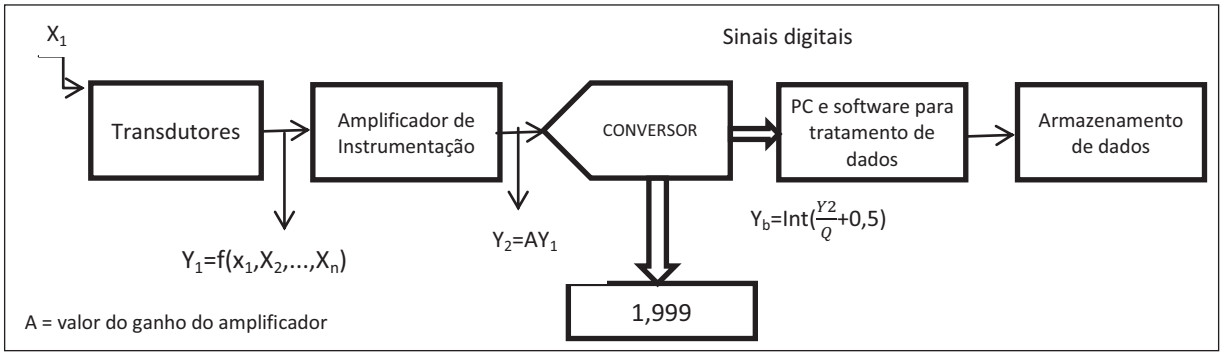


Figura 5: Exemplo da organização de uma cadeia de medição
Fonte: Campilho (2000)

Conforme dito anteriormente e segundo AZEVEDO (1992), uma cadeia de medição típica é composta pelos seguintes elementos:

1. Um transdutor e um circuito de linearização, que têm como principal função a conversão de uma determinada grandeza física, normalmente não elétrica, num determinado padrão elétrico (tal como uma corrente ou uma tensão), de forma a simplificar o tratamento dos dados medidos. Se considerar X como sendo o valor da grandeza a ser medida, a equação de saída é do tipo Y1=f(X1, X2, X3,….,Xn), onde Y é o valor de tensão de saída do circuito de linearização.

2. Um amplificador de instrumentação, que tem como objetivo a amplificação do sinal de saída do circuito de linearização, para valores adequados aos processos seguintes. O sinal de entrada e de saída do amplificador são relacionados da seguinte forma $Y_2=A.Y_1$, onde A é o valor do ganho do amplificador.

3. Um conversor analógico-digital (A/D), que tem por objetivo a conversão dos sinais analógicos de saída do amplificador para sinais digitais, de modo que estes possam ser traduzidos e observados num mostrador digital ou mesmo enviados para um computador.

4. Um sistema formado por computador e software, para analisar e tratar os dados recebidos de acordo com as necessidades/características inerentes a cada sistema de medição. Note-se que a utilização de um sistema informatizado num sistema de medição é atualmente quase uma obrigatoriedade, devido não só ao desenvolvimento da TI, mas também devido às necessidades atuais de resposta cada vez mais rápida, nomeadamente em relação à necessidade de processamento "personalizado" para cada tipo de medição.

5. Outra razão para a utilização de sistemas informatizados de apoio aos sistemas de medição relaciona-se com o processo de aquisição e armazenamento de dados, uma vez que o torna muito mais rápido e cômodo para o usuário do sistema.

Assim, uma cadeia de medição pode ser caracterizada por duas equações. A equação analógica, que traduz as medições analógicas que são realizadas e que são colocadas na entrada do conversor A/D e, uma equação digital que indica as características do conversor utilizado. Conforme Azevedo (1992), essas duas equações serão do tipo:

$$Y_2=A\ f(X_1, X_2, ..., X_n) \qquad \textit{Eq. 1}$$

e

$$Y_b= Int\left(\frac{Y_2}{Q}+0{,}5\right)=Int\left(2^n\frac{Y_2}{V_f}+0{,}5\right) \qquad \textit{Eq. 2}$$

onde Y_b é a representação binária, ou digital, do sinal de saída do conversor A/D, e n, Q e V_f são, respectivamente, o número de bits, o quantum e a gama de tensão de entrada do conversor.

Portanto, percebe-se que antes da criação de um sistema de medição, deve-se caracterizar o problema a ser analisado, estabelecendo todas as dependências funcionais que forem possíveis de identificar, bem como fazer o dimensionamento de todos os componentes da cadeia. Para isso, deve-se logo de início responder às seguintes questões:

1. Qual o tamanho da grandeza de entrada, bem como qual o tamanho da grandeza elétrica de saída correspondente aos blocos iniciais?

2. Qual o ganho dos amplificadores a serem utilizados? Vale observar que o ganho do amplificador deve ser diretamente dependente do valor máximo à entrada do conversor A/D.

Como tal não se deve confundir este valor com o valor de fim de escala do conversor A/D, pois podem não ser coincidentes.

3. Qual o número de bits a utilizar no conversor A/D? Isso está ligado diretamente à resolução esperada.

As respostas a estas questões irão determinar as especificações do sistema de medição, bem como a escolha dos elementos que o constituem. Estas questões, embora muito gerais, podem orientar o desenvolvimento de uma cadeia de medição.

2.1.3 Erros em uma cadeia de medição

Para se determinar uma grandeza física experimental, é necessário realizar medições físicas, porém o valor real da grandeza física é uma quantidade desconhecida, especialmente devido aos erros de medida. Segundo Vuolo (1992), o objetivo final da teoria de erros, consiste em determinar o melhor valor possível para a grandeza a partir dos resultados das medidas, e quanto este valor pode ser diferente do valor verdadeiro ou esperado.

No desenvolvimento de um sistema de medição é importante saber que cada um dos seus componentes pode introduzir um determinado erro, que irá impactar diretamente o erro global da medição. O amplificador operacional é um dos principais causadores de erro em um sistema de medição, por isso é importante este estudo.

Antes de estudar as diversas fontes de erro, deve-se em primeiro lugar entender alguns conceitos básicos que, geralmente, induzem alguma confusão. Segundo Azevedo (1992) uma vez que em qualquer dispositivo, com um ganho superior à unidade, o valor absoluto do erro da entrada (RAE) é inferior ao erro absoluto de saída do dispositivo (RAS), é preciso relacioná-los em função do seu ganho. A equação seguinte ilustra o conceito descrito:

$$\mathrm{RAE} = \frac{RAS}{A} \qquad Eq.\ 3$$

Ainda segundo Azevedo (1992), são diversas as fontes de erro e elas podem ser divididas em dois grupos distintos:

1) Os erros sistemáticos, responsáveis por desvios constantes de tensão ou de corrente na saída. Estes erros são caracterizados por fontes DC de tensão (VOSIN e VOSOUT) e por fontes de corrente (IOS). Os erros de ganho podem também ser divididos em dois grupos, sendo eles os erros devido à tolerância da resistência externa RG e os erros internos do amplificador de instrumentação, descritos normalmente pelo fabricante.

2) Os erros de natureza aleatória, onde está incluído o ruído. O ruído por sua vez também pode ser dividido em dois grupos distintos, denominados de ruído de corrente ($I_{RUÍDO}$) e o ruído de tensão ($V_{RUÍDO}$).

A figura 6 representa o exemplo de uma curva da densidade espectral do ruído de tensão de um amplificador, sendo a curva da densidade de ruído de corrente muito semelhante a essa figura.

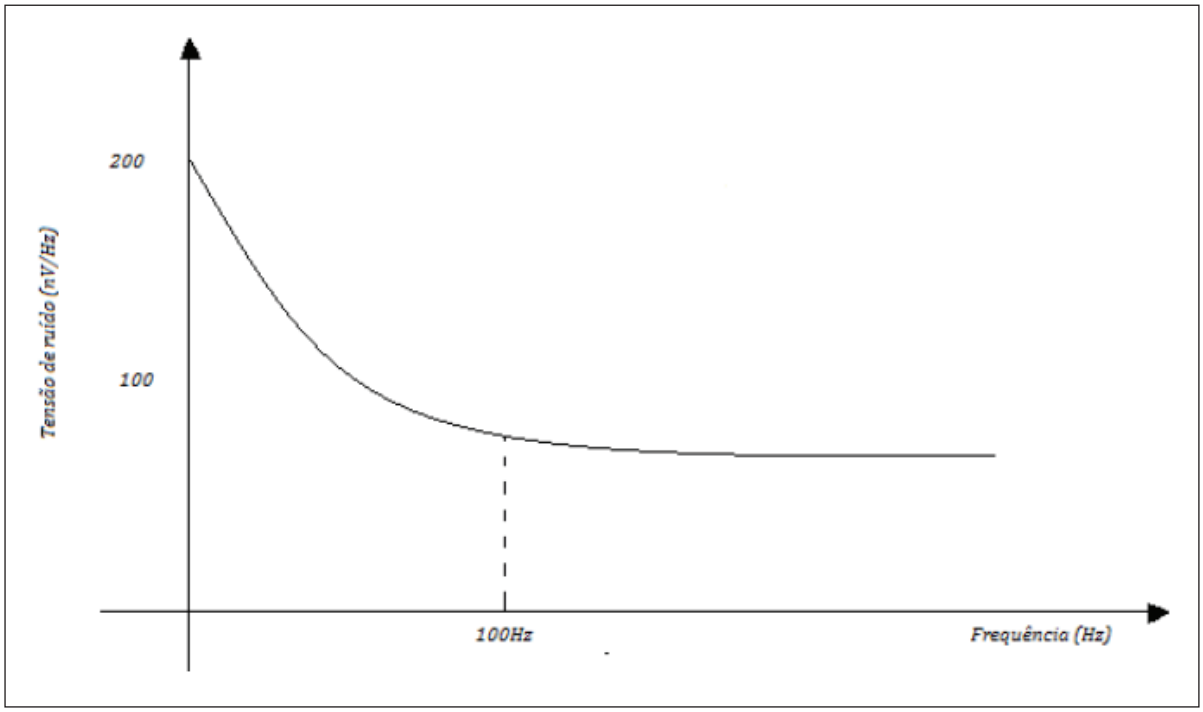


Figura 6: Densidade espectral de ruído de um amplificador de instrumentação
Fonte: Azevedo (1992)

Como se pode verificar pela da figura 6, este tipo de ruído aumenta com a diminuição da frequência, sendo por isso também denominado de ruído 1/f, uma vez que a curva da sua densidade espectral para baixas frequências possui uma evolução inversamente proporcional à da frequência. Este tipo de ruído é normalmente considerado constante para frequências acima de 100Hz, e muitas vezes desprezível, para aplicações que utilizem frequências superiores a 100Hz.

Uma vez que os ruídos de tensão e de corrente não são relacionáveis, o valor total medido, em tensão, é determinado através da seguinte equação:

$$V_{Total} = V_{RUIDO} + R_{fonte} \, . \, I_{RUÍDO} \qquad \text{Eq. 4}$$

$$\Delta V_{Total} = \sqrt{\Delta v^2{}_{RUÍDO} + R^2{}_{fonte} I^2{}_{RUÍDO}} \qquad \text{Eq. 5}$$

onde R_{fonte} é a resistência da fonte de sinal que se encontra em paralelo com a fonte de corrente de ruído.

Outros tipos de ruído têm de ser considerados num sistema de medição, nomeadamente os erros de linearidade, bem como as derivas térmicas correspondentes. Assim o valor total da grandeza medida numa uma cadeia de medição é a soma do valor absoluto dos vários erros parciais de cada componente, ou seja,

$$Y_{total} = Y_{f+T} + Y_{amp} + Y_{AD} \qquad Eq.\ 6$$

Agora, agregando-se estes erros às equações Eq.1 e Eq2, obtém-se a seguinte equação:

$$Y_2 = {}_{A\,.\,f}\,.\,(X_1, X_2, \ldots X_n) + Y_{f+1} + Y_{amp} + Y_{A/D} \qquad Eq.\ 7$$

onde, $Y_{A/D}$ é nulo, ou melhor, não aparece na Eq. 7, pois o sinal Y2 é lido antes do conversor analógico digital. A leitura na saída do conversor é do tipo:

$$Y_b = Int\,(2^n \frac{Y2}{Vf} + 0{,}5) + Y_{A/D} \qquad Eq.\ 8$$

Note-se que na equação *Eq. 7* referem-se aos erros de entrada no conversor, ao passo que os erros da saída do conversor estão representados na equação *Eq. 8*.

2.1.4 Medição de temperatura

A temperatura é uma das variáveis da engenharia mais comumente utilizada e medida. Ela afeta diariamente todos os ambientes, mas sua definição e medição não são coisas simples. De acordo com FIGLIOLA e BEASLEY (2007), a temperatura pode ser definida em termos gerais como a propriedade de um objeto que descreve como mais quente ou mais frio, conceitos que são claramente relativos. Afirma ainda que a Lei Zero da Termodinâmica declara que dois sistemas em equilíbrio térmico com um terceiro sistema estão em equilíbrio entre si. O equilíbrio térmico implica que não ocorre nenhuma transferência de calor entre os sistemas, o que também indica igualdade de temperatura. Embora a Lei Zero da Termodinâmica forneça essencialmente a definição da igualdade de temperatura, ela não proporciona meios para definir uma escala de temperatura.

Uma escala de temperatura sustenta três aspectos essenciais da medição de temperatura: a definição da magnitude, os pontos de referência fixos para o estabelecimento das temperaturas conhecidas e os meios para interpolação entre esses pontos de temperatura fixos.

2.2 Sensores e Transdutores

Um sensor é um dispositivo capaz de detectar estímulos externos e responder em consequência, ou seja, são artefatos que permitem obter informações do meio e interagir com o mesmo. Já o transdutor é um dispositivo que transforma um tipo de energia em outro, utilizando para isso um elemento sensor. Por exemplo, o sensor pode traduzir informação não elétrica (velocidade, posição, temperatura) em informação elétrica (corrente, tensão, resistência), como exemplificado na figura 7.

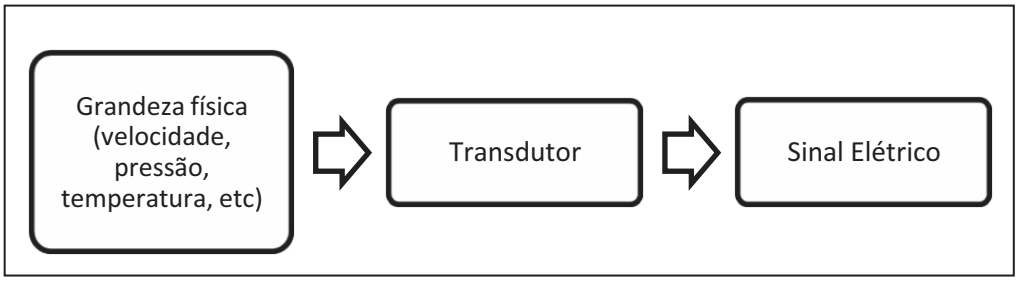


Figura 7: Exemplo de aplicação de transdutor

Segundo GARCIA (2005), um transdutor consiste basicamente de um elemento sensor combinado com um transmissor, conforme indicado na figura 8:

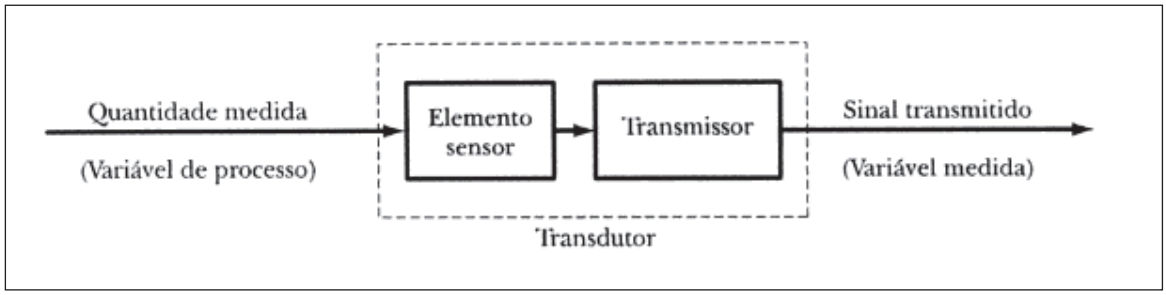


Figura 8: Representação de um Transdutor
Fonte: GARCIA (2005)

Os transdutores podem ser passivos, ou seja, aqueles cuja energia de saída é proveniente da energia de entrada, ou ativos, que dispõem de uma fonte de alimentação de energia. Nestes, a maior parte da energia de saída é provida pela alimentação.

Neste trabalho, são utilizados termopares, que são transdutores que convertem temperatura em sinais elétricos e também os medidores de pressão do tipo Pirani e o de membrana Capacitiva.

2.2.1 Termopares

Segundo BALBINOT E BRUSAMARELLO (2010), o circuito de Seebeck, denominado par termelétrico ou, comumente, termopar, é uma fonte de força eletromotriz – fem (medido em unidades de tensão elétrica) e, normalmente utilizado como um sensor de temperatura. A polaridade e magnitude da tensão dependem da temperatura e do tipo de material que compõe o termopar.

O chamado "efeito Seebeck" foi observado a partir de um experimento em que uma corrente elétrica percorre um circuito fechado composto por dois metais diferentes, quando as junções estão expostas à temperaturas diferentes, conforme figura 9. Neste circuito fechado, surge uma *fem* que depende apenas dos tipos de metais e da temperatura nas junções do termopar.

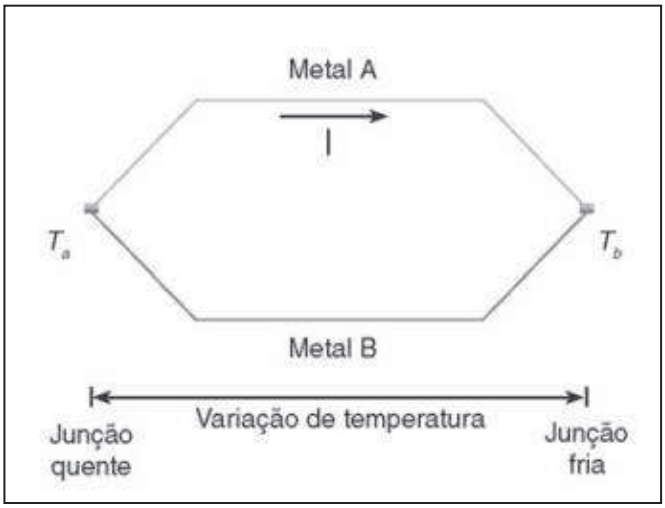


Figura 9: Representação do efeito Seebeck
Fonte: BALBINOT E BRUSAMARELLO (2010)

A relação entre a *fem* e a diferença de temperatura T entre as junções define o coeficiente de Seebeck, definido por:

$$S_{ab} = \frac{d(fem)}{dT} = S_a - S_b$$

sendo que S_a e S_b representam, respectivamente, a potência termoelétrica absoluta entre dois pontos a e b do termopar. Pela definição do coeficiente de Seebeck, percebe-se que ele é inversamente proporcional à T.

Outra importante regra ao se utilizar o termopar é a Lei dos Metais Intermediários, que estabelece que se em algum ponto do circuito for inserido um metal genérico, desde que, a temperatura nas novas junções sejam mantidas iguais, a tensão de Seebeck não se altera – conforme pode-se verificar na figura 10. Este é um aspecto muito importante a se considerar no momento de ligar um sistema de medição ao termopar.

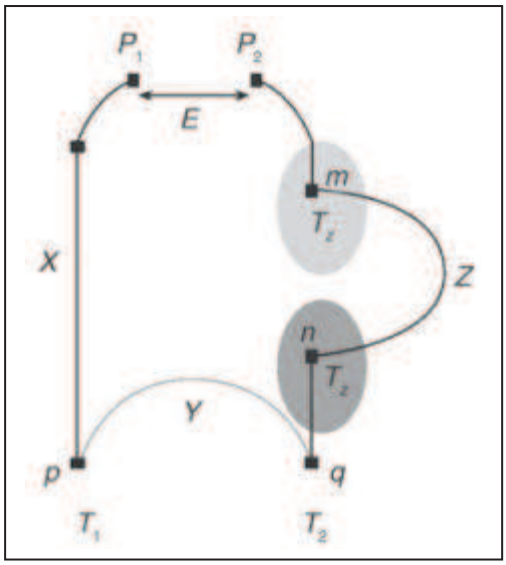


Figura 10: Lei dos metais intermediários: a tensão de Seebeck não se altera em função da inserção do metal intermediário, desde que as junções novas (m, n) sejam mantidas à mesma temperatura (T_z)
Fonte: BALBINOT E BRUSAMARELLO (2010)

Outra lei essencial para a correta utilização dos termopares é a Lei das Temperaturas Sucessivas, que descreve a relação entre a *fem* obtida para diferentes temperaturas de referência ou de junção fria. Essa lei permite compensar ou prever dispositivos que compensem mudanças na temperatura da junta de referência. A relação $V_s(T)$ pode ser obtida graficamente, seguindo a chamada curva de calibração, para um termopar com a junta de referência em T_1, mantida à 0 °C, conforme figura 11.

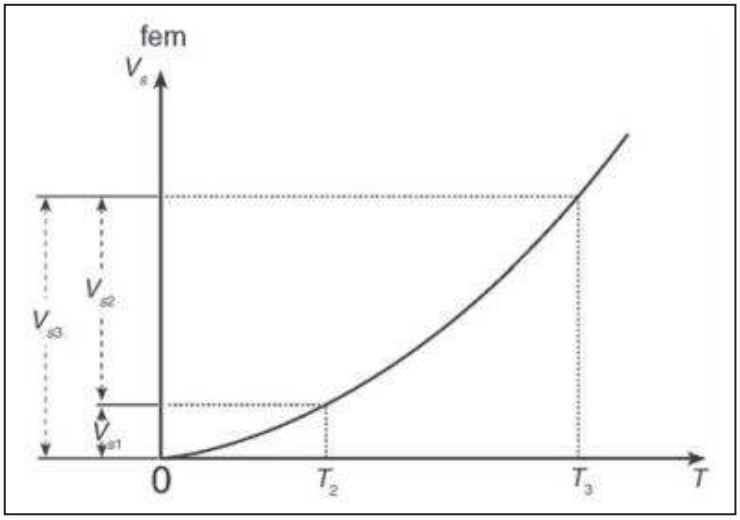


Figura 11: Curva de calibração de um determinado termopar
Fonte: BALBINOT E BRUSAMARELLO (2010)

Segundo BALBINOT E BRUSAMARELLO (2010), com a curva de calibração ou com a função de calibração de um determinado termopar pode-se determinar qualquer outra curva relativa à junta de referência a uma dada temperatura. As curvas de calibração dos termopares geralmente não são lineares, mas para a maioria dos termopares usuais pode-se considerá-las lineares, a depender da faixa de temperatura utilizada e da sensibilidade do medidor de fem. A inclinação da curva V_s x T em um ponto qualquer é denominada potência termoelétrica, que geralmente é pequena e varia com a natureza do termopar.

A dependência da diferença de potencial entre as juntas com a temperatura pode ser aproximada por funções como essa:

$$V_s = a + b \,.\, T + c \,.\, T^2 + \ldots$$

em que *a, b, c* são constantes determinadas pelo experimento em cabe observar que, se a junção de referência está a 0 °C, implicará em a = 0. Portanto, a variação da fem (Δfem) em função da variação da temperatura (ΔT) é a potência termoelétrica:

$$P = \frac{dVs}{dT} = b + 2 \,.\, c \,.\, T + \ldots$$

ou, para um intervalo de temperatura muito grande, $P_m = \frac{\Delta Vs}{\Delta T}$, onde P_m = potência termoelétrica média.

2.2.1.1 Medição de tensão do termopar

Segundo FIGLIOLA e BEASLEY (2007), o melhor método para medição de tensões em termopares é um dispositivo que minimiza o fluxo de corrente. Por muitos anos, o potenciômetro foi o padrão de laboratório para a medição de tensão em circuitos de termopares. Um potenciômetro possui erro de carregamento próximo de zero em uma condição de equilíbrio. Entretanto, equipamentos modernos de medição de tensão, como placas de aquisição de dados digitais, possuem uma impedância de entrada tão alta que podem ser usados com erro mínimo de carregamento.

2.2.1.2 Composição e erro sistemático do termopar

O quadro 2 fornece a composição padrão de materiais de termopares, juntamente com os limites padrões de erro para várias combinações de materiais. Esses limites especificam os erros máximos esperados que resultam dos materiais dos termopares.

Quadro 2: Composição de termopares

Tipo	Fio		Erro Sistemático Esperado
	Positivo	Negativo	
S	Platina	Platina/10% ródio	+- 1,5°C ou 0,25%
R	Platina	Platina/13% ródio	+- 1,5°C
B	Platina/30% ródio	Platina/6% ródio	+- 0,5°C
T	Cobre	Constantan	+- 1,0°C ou 0,75%
J	Ferro	Constantan	+- 2,2°C ou 0,75%
K	Cromel	Alumel	+- 2,2°C ou 0,75%
E	Cromel	Constantan	+- 1,7°C ou 0,5%
Designações de ligas Constantan: 55% cobre com 45% níquel Cromel: 90% níquel com 10% cromo Alumel: 94% níquel com 3% manganês, 2% alumínio e 1% silício			

Fonte: Adaptado de FIGLIOLA e BEASLEY (2007)

2.2.2 Sensor de Membrana Capacitiva

Um medidor de membrana consiste basicamente de uma placa sensora, um dos lados de um capacitor plano, isolado do meio por uma membrana sensível. A variação de pressão exercida pelo meio na membrana, provoca uma deflexão desta em relação à placa sensora. Se for mantido o dielétrico constante, tem-se uma variação da capacitância proporcional à

pressão exercida sobre a membrana. O circuito eletrônico básico usado para a medida da variação da capacitância, é um oscilador "LC" onde um dos capacitores é o próprio elemento sensor. Assim, pela variação da capacitância deste, tem-se uma variação da frequência do oscilador, que é convertida em tensão.

O medidor utilizado neste estudo é o sensor de membrana capacitiva Baratron MKS 626, cujas características são descritas no quadro 3.

Quadro 3: Características do Baratron MKS 626

Característica	Descrição
Precisão	0,25% de leitura padrão para faixas de 1 a 1000 Torr. Para valores menores que 1 Torr, a especificação padrão é de 0,50% da leitura.
Tensão de Entrada	+- 15 VDC
Tensão de Saída	0V a 10V

Fonte: MKS (2014)

O quadro 4 descreve a pinagem do sensor:

Quadro 4: Pinagem do sensor Baratron MKS 626

Pino	Descrição
1	Sem conexão
2	Sinal de saída
3	Sem conexão
4	Sem conexão
5	Power Common
6	-15 VDC
7	+15 VDC
8	Sem conexão
9	Sem conexão
10	Sem conexão
11	Sem conexão
12	Signal Common
13	Sem conexão
14	Sem conexão
15	Aterramento do Chassi

Fonte: MKS (2014)

2.3 Amplificador Operacional (AMPOP)

De acordo com BALBINOT E BRUSAMARELLO (2010) um amplificador operacional é um dispositivo eletrônico composto por resistências, transistores, capacitores, entre outros presentes em projetos de condicionadores de sinais, cuja finalidade principal é a de amplificar sinais de entrada. São diversas as aplicações desse dispositivo. Além de encontrá-lo em sistemas de aquisição de dados, pode estar presente também em sistemas de controle industrial, em instrumentação em geral, em equipamentos de telecomunicação, sistemas de áudio e outros. Por ter tantas aplicações diferentes, pode ser encontrado com características diversas, como por exemplo, pode ser construído e otimizado para ter um baixo consumo de energia ou pode ser otimizado para responder a sinais em uma ampla gama de frequência, tudo depende da aplicação. De forma geral, é indicado para situações onde são necessários altos ganhos, imunidade ao ruído, impedância de entrada alta e impedância de saída baixa, sem distorção e com estabilidade.

Um amplificador operacional ideal apresenta as seguintes características:

- Ganho Infinito
- Impedância de Entrada Infinita
- Largura de Banda Infinita
- Impedância de Saída Zero
- Tensão de Offset e Corrente de Offset Zero

Existem duas regras fundamentais para o funcionamento adequado de um amplificador operacional ideal com realimentação externa:

1) A saída fará todo o possível para fazer com que a diferença de tensão entre os terminais Não Inversor e Inversor seja zero.

2) Nas entradas do amplificador não deve fluir corrente.

Em condições ideais, o amplificador operacional pode ser representado conforme a Figura 12, onde os terminais (+) e (-) correspondem às entradas do amplificador e têm propriedades de entradas não-inversora e inversora.

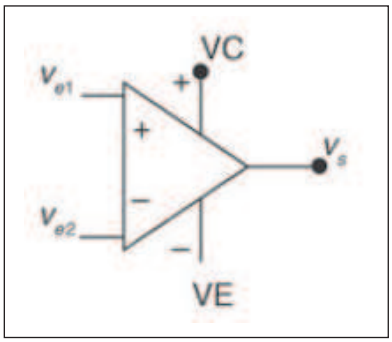


Figura 12: Representação de um Amplificador Operaciona1
Fonte: Balbinot e Bruzamarello (2010)

O amplificador é alimentado simetricamente pelos pinos +VC e –VE, embora alguns tipos de AMPOP não tenham necessidade de ser alimentados com tensão simétrica. O ganho diferencial Ad é dado por:

$$V_s = Ad(V_{c1} - V_{c2})$$

e

$$Ad \rightarrow \infty$$

A impedância de entrada é infinita e a impedância de saída é nula. Se $V_{c1} = V_{c2}$, tem-se Vs = 0, ou seja, o ganho em modo comum é zero. Um amplificador com essas características é chamado de Amplificador Ideal.

2.4 Plataforma Arduino

O projeto Arduino foi desenvolvido na Itália em 2005 objetivando oferecer uma plataforma de prototipagem eletrônica de baixo custo e de fácil manuseio por qualquer pessoa interessada em criar projetos com objetos e ambientes interativos (ARDUINO, 2011). A plataforma Arduino é composta de uma placa eletrônica (hardware) e de um ambiente de desenvolvimento (software) para criação dos projetos pelos usuários. É também uma plataforma de desenvolvimento *open source hardware* (*OSHW),* também chamado de hardware livre*,* baseado em uma placa de microcontrolador simples, geralmente controladores da marca ATmega que pode ser associada ao conceito de *physical computing*, ou seja, ao conceito da criação de sistemas físicos, através de hardware e software que interagem e respondem às entradas (*inputs*) do mundo real. Segundo Mellis (2009):

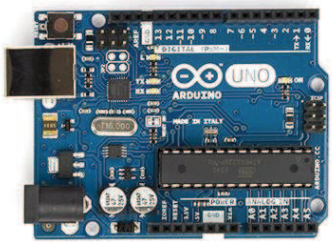
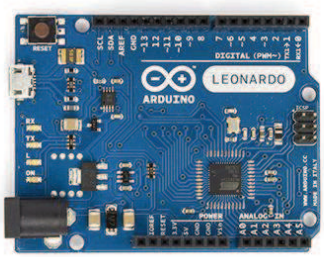
> Nós queríamos que outras pessoas estendessem a plataforma para adequá-la às suas necessidades. Para isso, elas deveriam ter acesso ao código-fonte do

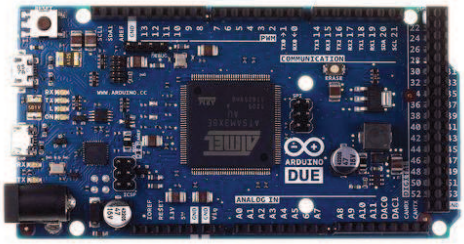
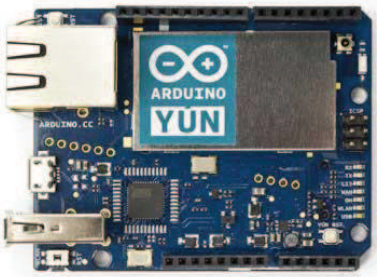
> software e ao projeto do hardware. Além disso, como era uma plataforma nova, ser de código aberto deu confiança às pessoas. Elas sabiam que poderiam continuar expandindo a plataforma mesmo que o desenvolvedor original desistisse dela.

A definição para o termo hardware livre ou hardware aberto tem sido discutida e aperfeiçoada nos fóruns do Open Hardware Sumitt (OHS, 2014), estando atualmente em sua versão 1.1. A definição de OSHW está disponível na comunidade Freedom Defined. Segundo esta definição, Open Source Hardware (OSHW) é um termo para artefatos tangíveis - máquinas, dispositivos ou outros objetos físicos - cujo design foi disponibilizado ao público de modo que qualquer um pode construir, modificar, distribuir e utilizar estes artefatos. (FREEDOM DEFINED, 2014). Outros critérios que validam uma distribuição sob o conceito Open Source Hardware (OSHW) está disponível no ANEXO A.

O quadro 5 exibe alguns dos modelos da placa Arduino encontrados no site oficial (ARDUINO, 2015).

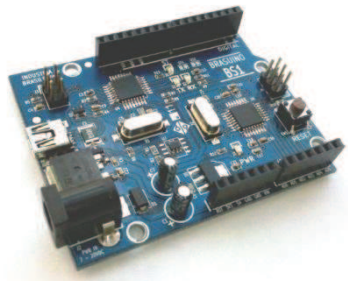
Quadro 5: Modelos de Placas Arduino

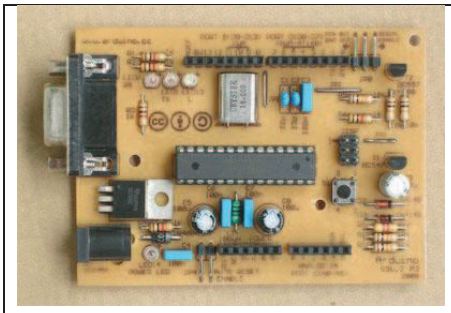
Placa	Características
 ARDUINO UNO	✓ Microcontrolador ATmega328 ✓ 14 pinos de I/O digitais ✓ 6 pinos analógicos ✓ Memória Flash de 32 KB ✓ Velocidade do clock de 16 MHZ
 ARDUINO LEONARDO	✓ Microcontrolador ATmega32u4 ✓ 20 pinos de I/O digitais ✓ 12 pinos analógicos ✓ Memória Flash de 32 KB ✓ Velocidade do clock de 16 MHZ
	✓ Microcontrolador ATmega32u4 ✓ 20 pinos de I/O digitais ✓ 12 pinos analógicos

 ARDUINO DUE	✓ Memória Flash de 32 KB ✓ Velocidade do clock de 16 MHZ
 ARDUINO YUN	✓ Microcontrolador ATmega32u4 ✓ 20 pinos de I/O digitais ✓ 12 pinos analógicos ✓ Memória Flash de 32 KB ✓ Velocidade do clock de 16 MHZ

Sob a filosofia *Open Hardware*, a placa eletrônica Arduino possui toda documentação no site do projeto para aqueles que desejarem montar sua própria placa a partir do modelo, o que proporcionou que, além dos modelos oficiais lançados pela equipe do projeto Arduino, diversos outros modelos de hardware surgissem em várias partes do mundo. Em função desta característica *open source,* muitos outros modelos da placa eletrônica surgiram, desenvolvidos pela comunidade Arduino em todas as partes do mundo. Esses modelos não oficiais também estão listados no site do projeto, inclusive as versões brasileiras Severino e Brasuino, que são destacadas no quadro 6.

Quadro 6: Versões brasileiras do Arduino

Placa	Modelo
 BRASUINO	✓ Microcontrolador ATmega328 ✓ 14 pinos de I/O digitais ✓ 6 pinos analógicos ✓ Memória Flash de 32 KB ✓ Velocidade do clock de 16 MHZ • Compatível com o Arduino UNO

 SEVERINO	✓ Microcontrolador ATmega168 ✓ 8 pinos de I/O digitais ✓ 6 pinos analógicos ✓ Memória Flash de 16 KB ✓ Velocidade do clock de 16 MHZ

Pensando em aumentar as funcionalidades da placa Arduino, diversas empresas de hardware desenvolveram placas eletrônicas que podem ser conectadas aos terminais do Arduino. Estas placas, chamadas de Shields, possibilitam que Arduino tenha novas funções, desde um maior controle sobre motores até conexão com redes sem fio. O site oficial do Arduino mostra uma lista de Shields disponíveis e também uma lista de fabricantes de Shields (SHIELDLIST.ORG, 2014). A figura 13 mostra um exemplo de Shield para Arduino.

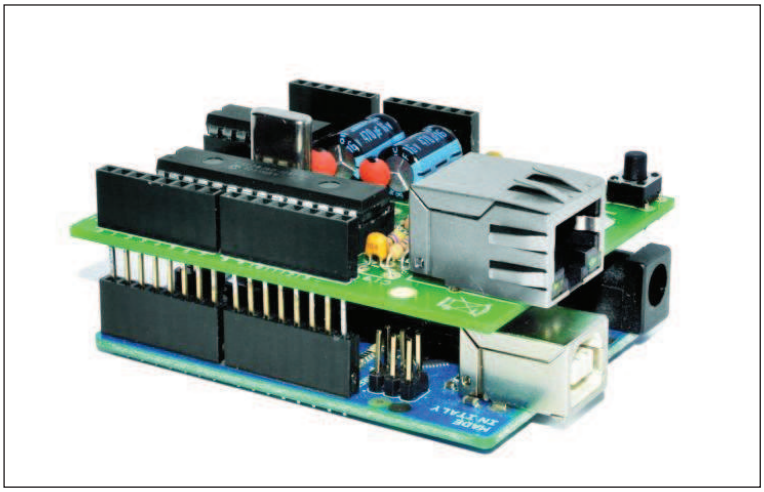


Figura 13: Exemplo de Shield com conexão de rede acoplada ao Arduino

O Arduino possui seu próprio ambiente de desenvolvimento (IDE – Interface Development Environment) para o desenvolvimento dos programas (sketches), que pode ser conseguido gratuitamente do site do projeto. Essa IDE serve para escrever software, transmiti-lo para a placa e visualizar os resultados gerados por meio do monitor da porta serial.

Ela pode ser utilizada para desenvolver interfaces de interação, possuindo entradas provenientes de uma variedade de sensores (temperatura, luz, som, etc.) e controle de uma variedade de saídas, como luzes, motores e outras. Projetos do Arduino podem ser *stand-*

alone, ou seja, executar independente do computador ou eles podem se comunicar com software rodando em um computador.

Neste projeto se utilizou a IDE Arduino versão 1.0.5. Esta versão oferece diversos exemplos de *sketches* (como são chamados os programas do Arduino), prontas para serem utilizadas, o que favorece o entendimento da linguagem por parte de um usuário iniciante e também, ganho de tempo no desenvolvimento ou adaptação de um programa para uma nova aplicação.

Conforme mostrado na figura 14, a IDE Arduino 1.0.5 apresenta menus suspensos para diversas funções (1), botões de controle (2), uma área de texto para digitação do código do programa (3), além de uma área de mensagens para comunicação com o usuário (4).

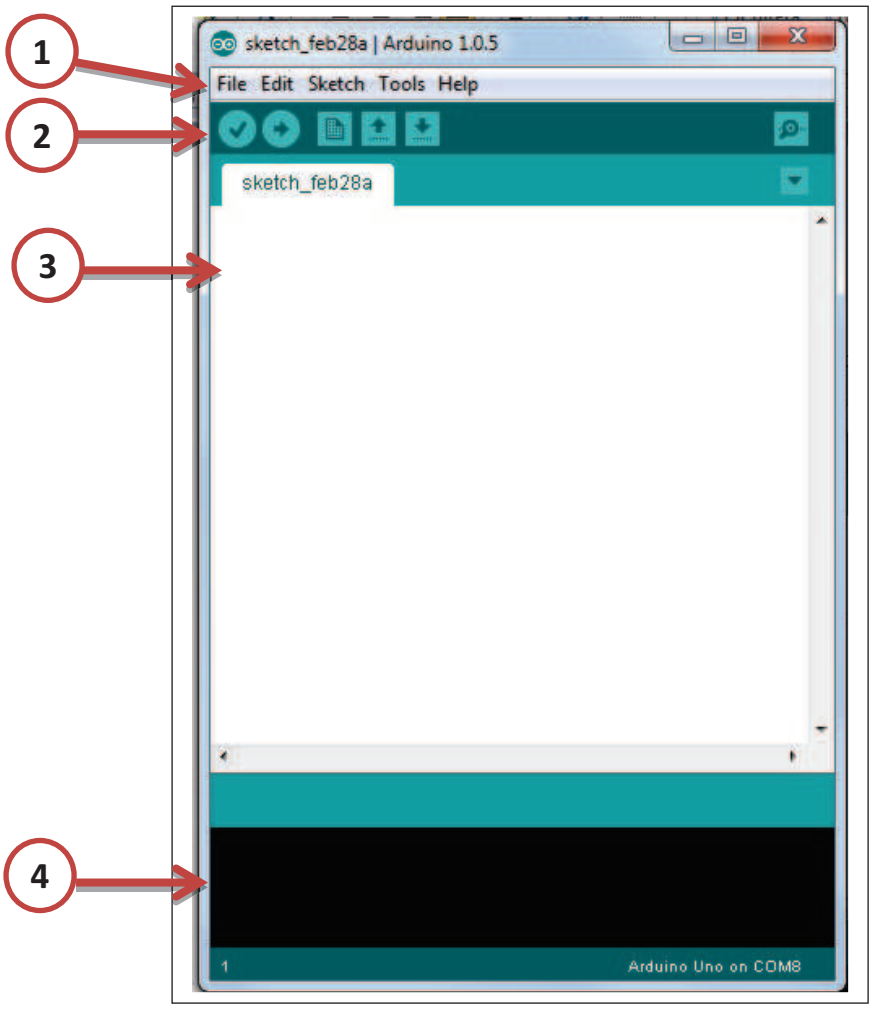


Figura 14: IDE – Ambiente de Desenvolvimento Integrado

As funcionalidades principais da IDE podem ser acessadas pelos botões abaixo do Menu de Opções e são descritas a seguir:

 - **Verify**: checa se há algum erro no código

 - **Upload**: compila o código e envia para o *bootloader* da placa

 - **New**: cria uma nova sketch

 - **Open**: abre uma sketch já existente

 - **Save**: salva as alterações realizadas

 - **Serial Monitor**: Abre o monitor de porta serial

Para obter a IDE, basta acessar o site oficial do projeto, na seção Download e selecionar a opção adequada ao sistema operacional a ser utilizado, podendo ser Windows, Linux ou Mac OS X. No Anexo D pode ser encontrada a apostila de programação do Arduíno que serviu de referência para este trabalho.

2.4.1 A interface Arduino UNO

A versão demonstrada na figura 15 é o modelo Arduino Uno, escolhida para este projeto, que apresenta 6 portas analógicas de entrada e saída que permitem a medida de tensões externas, sendo possível receber informações de uma série de sensores como medidores de temperatura, pressão, umidade, distância, sensores de gases, fototransistores, etc. Existem, também, 14 portas digitais, onde é possível ler e escrever dois estados, 0/1 ou HIGH/LOW, permitindo, por exemplo, manter um LED ligado ou desligado. Quando necessário algumas dessas portas digitais podem ser configuradas para atuarem como portas de saída analógicas, através de modulação por largura de pulso ou *Pulse-Width Modulation* (PWM). Possui ainda um cristal oscilador de 16 MHz, uma conexão USB, uma entrada para alimentação, um cabeçalho ICSP e um botão de reset. O esquema elétrico da placa Arduino UNO pode ser encontrado no ANEXO B.

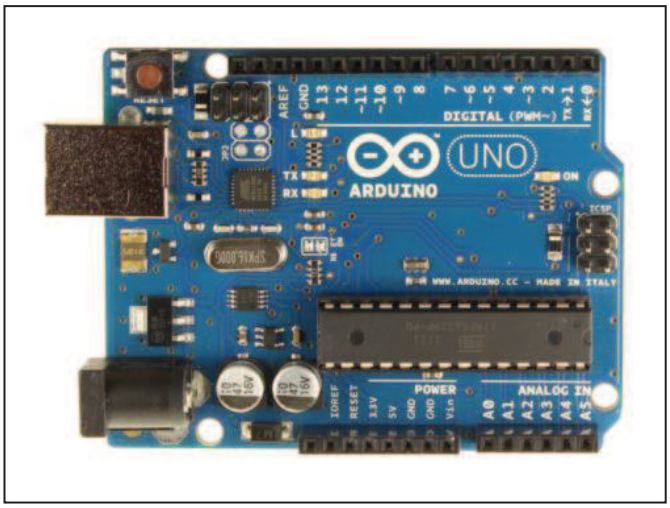


Figura 15: Placa Arduino
Fonte: Arduino (2012)

2.4.2 Características do Arduino UNO

O quadro 7 mostra um resumo das características do Arduino UNO e outros aspectos como alimentação, memória, comunicação, programação e reset são descritos em seguida.

Quadro 7 - Resumo das características do Arduino

Componente	Características
Microcontrolador	ATmega328
Voltagem de operação	5V
Voltagem de entrada (recomendada)	7-12V
Voltagem de entrada (limites)	6-20V
Entradas/Saídas Digitais	14
Entradas Analógicas	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Memória Flash	32 KB (ATmega328) dos quais 2 KB são usados pelo *bootloader*
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

Fonte: Arduino (2012)

2.4.2.1 Alimentação:

O Arduino UNO pode ser alimentado pela conexão USB ou por qualquer fonte de alimentação externa. A fonte de alimentação é selecionada automaticamente. Alimentação externa (não-USB) pode ser tanto de uma fonte ou de uma bateria. A fonte pode ser conectada

com um plug de 2,1mm (centro positivo) no conector de alimentação. Cabos vindos de uma bateria podem ser inseridos nos pinos Gnd (terra) e Vin (entrada de voltagem) do conector de alimentação. A placa pode operar com uma alimentação externa de 6 a 20 volts. Entretanto, se a alimentação for inferior a 7 volts o pino 5V pode fornecer menos de 5 volts e a placa pode ficar instável. Se a alimentação for superior a 12 volts o regulador de voltagem pode super-aquecer e avariar a placa. A alimentação recomendada é de 7 a 12 volts.

Os pinos de alimentação são:

VIN. Entrada de alimentação para a placa Arduino quando uma fonte externa for utilizada. Você pode fornecer alimentação por este pino ou, se usar o conector de alimentação, acessar a alimentação por este pino.

5V. A fonte de alimentação utilizada para o microcontrolador e para outros componentes da placa. Pode ser proveniente do pino Vin através de um regulador on-board ou ser fornecida pelo USB ou outra fonte de 5 volts.

3V3. Alimentação de 3,3 volts fornecida pelo chip FTDI. A corrente máxima é de 50 mA.

GND. Pino terra.

2.4.2.2 Memória:

O ATmega328P tem 32 KB de memória flash para armazenar código (dos quais 2 KB são utilizados pelo bootloader), além de 2 KB de SRAM e 1 KB de EEPROM.

2.4.2.3 Entrada e Saída:

Cada um dos 14 pinos digitais do Arduino UNO pode ser usado como entrada ou saída usando as funções de pinMode(), digitalWrite(), e digitalRead(). Eles operam com 5 volts. Cada pino pode fornecer ou receber um máximo de 40 mA e tem um resistor pull-up interno (desconectado por padrão) de 20-50 kΩ. Além disso, alguns pinos têm funções especializadas:

- Serial: 0 (RX) e 1 (TX). Usados para receber (RX) e transmitir (TX) dados seriais TTL. Estes pinos são conectados aos pinos correspondentes do chip serial FTDI USB-to-TTL.
- External Interrupts: 2 and 3. Estes pinos podem ser configurados para disparar uma interrupção por um baixo valor, uma elevação ou falling edge ou uma mudança de valor. Veja a função attachInterrupt() para mais detalhes.

- PWM: 3, 5, 6, 9, 10, e 11. Fornecem uma saída analógica PWM de 8-bit com a função analogWrite().
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Estes pinos suportam comunicação SPI, que embora compatível com o hardware, não está incluída na linguagem do Arduino.
- LED: 13. Há um LED já montado e conectado ao pino digital 13. Quando o pino está no valor HIGH, o LED acende; quando o valor está em LOW, ele apaga.

O Arduino UNO tem 6 entradas analógicas e cada uma delas tem uma resolução de 10 bits (equivale a 1024 valores diferentes). Por padrão, elas medem de 0 a 5 volts, embora seja possível mudar o limite superior usando o pino AREF e um pouco de código de baixo nível.

2.4.2.4 Comunicação:

Com o Arduino UNO a comunicação com um computador, com outro Arduino ou com outros microcontroladores é muito simplificada. O ATmega328P permite comunicação serial no padrão UART TTL (5V), que está disponível nos pinos digitais 0 (RX) e 1 (TX).Um chip FTDI FT232RL na placa encaminha esta comunicação serial através do USB e os drives FTDI fornece uma porta COM virtual para o software no computador. O software Arduino inclui um monitor serial que permite que dados simples de texto sejam enviados à placa Arduino. Os LEDs RX e TX da placa piscam quando os dados estão sendo transferidos ao computador pelo chip FTDI e a conexão USB.

A biblioteca SoftwareSerial permite comunicação serial por quaisquer dos pinos digitais do Arduino UNO.

2.4.2.5 Programação:

O Arduino UNO pode ser programado com o software Arduino. O ATmega328P no Arduino UNO vem pré-gravado com um bootloader que permite enviar novos programas sem o uso de um programador de hardware externo. Ele se comunica utilizando o protocolo original STK500.

2.5.2.6 Proteção contra sobrecorrente USB:

O Arduino UNO tem um polifusível reinicializável que protege a porta USB do seu computador contra curto-circuito e sobrecorrente. Apesar da maioria dos computadores possuírem proteção interna própria, o fusível proporciona uma proteção extra. Se mais de 500mA foram aplicados na porta USB, o fusível irá automaticamente interromper a conexão até que o curto ou a sobrecarga seja removida.

2.5 Aquisição e Análise de Dados

O computador tem sido frequentemente utilizado como uma ferramenta que agiliza processos através da gravação de dados e recuperação dos mesmos de forma rápida e confiável. Em vários ambientes de trabalho, o computador tornou-se indispensável devido ao grande volume de dados que são gerados atualmente. Em laboratórios de pesquisa científica, essa conduta não é diferente.

Conforme BOLTON (2009), o termo Aquisição de Dados tende a ser usado frequentemente para sistemas nos quais as entradas dos sensores são convertidas do formato analógico para o formato digital para processamento, análise e apresentação de resultados por meio de um computador, conforme demonstrado na figura 16.

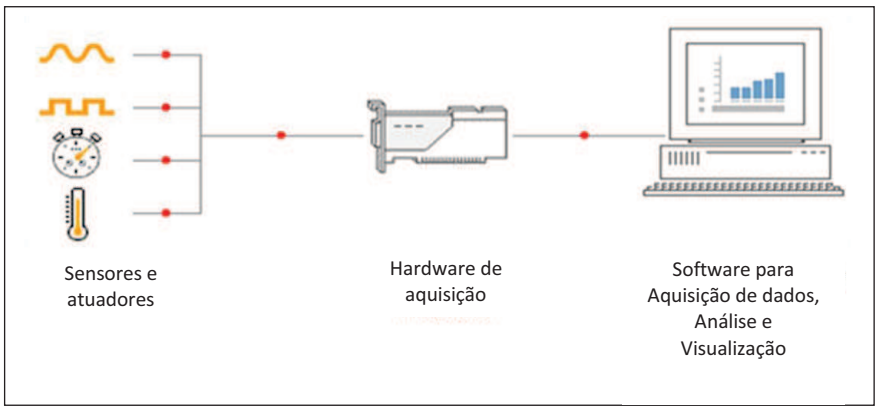


Figura 16: Representação de aquisição de dados para tratamento em computador
Fonte: Adaptado de National Instruments (2013)

Para FIGLIOLA e BEASLEY (2007), um sistema de aquisição de dados é a porção de um sistema de medição que quantifica e armazena dados. Exemplificando, um pesquisador que lê o indicador de um transdutor, associa um número à uma posição do indicador e registra a informação em um livro de registros realiza todas as tarefas pertinentes à um sistema de aquisição de dados. Porém, este trabalho trata da aquisição de dados por meio de um sistema

baseado em microprocessador, que faz todo o trabalho de forma automatizada. A figura 17 mostra como um sistema de aquisição de dados (DAS – data-acquisition system) pode ser encaixado no esquema geral de medição, da medição real à subsequente redução de dados.

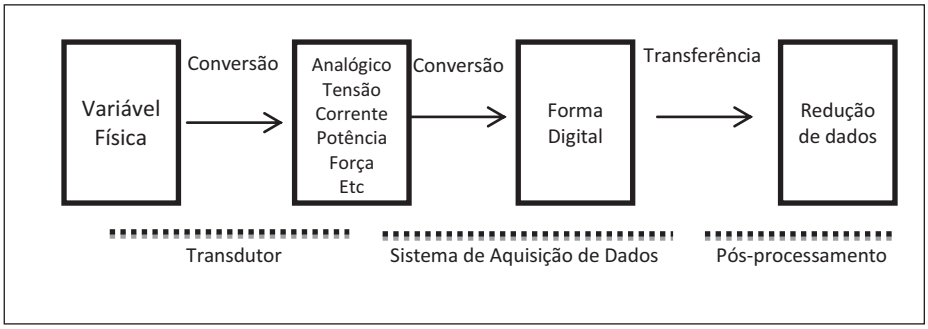


Figura 17: Sinal e esquema de medição típicos
Fonte: FIGLIOLA e BEASLEY (2007)

Sistemas de microprocessador dedicado podem realizar continuamente instruções programadas para medir, armazenar, interpretar e controlar processos sem nenhuma intervenção. Esses microprocessadores possuem portas de entrada/saída (I/O) com interface com outros equipamentos para medir e gerar instruções. A programação permite operações tais como escolher qual sensor fará uma medição, quando e com que frequência. Permite ainda tomada de decisão e realimentação para controle de variáveis de processo.

Os sistemas de aquisição de dados baseados em computadores, conforme demonstrado na figura 16, são considerados híbridos porque combinam um pacote de aquisição de dados com o microprocessador e a interface entre o homem e o computador pessoal. A interface entre instrumentos externos e o PC é feita utilizando placas de entrada/saída (I/O), que se conectam a uma porta adequada existente no computador ou a uma porta de comunicação externa, o que dá acesso direto ao barramento do computador, o principal caminho utilizado para todas as operações de computador.

A figura 18 mostra um diagrama de fluxo típico de um sinal com múltiplos sinais de entrada para um único controlador DAS baseado em microprocessador.

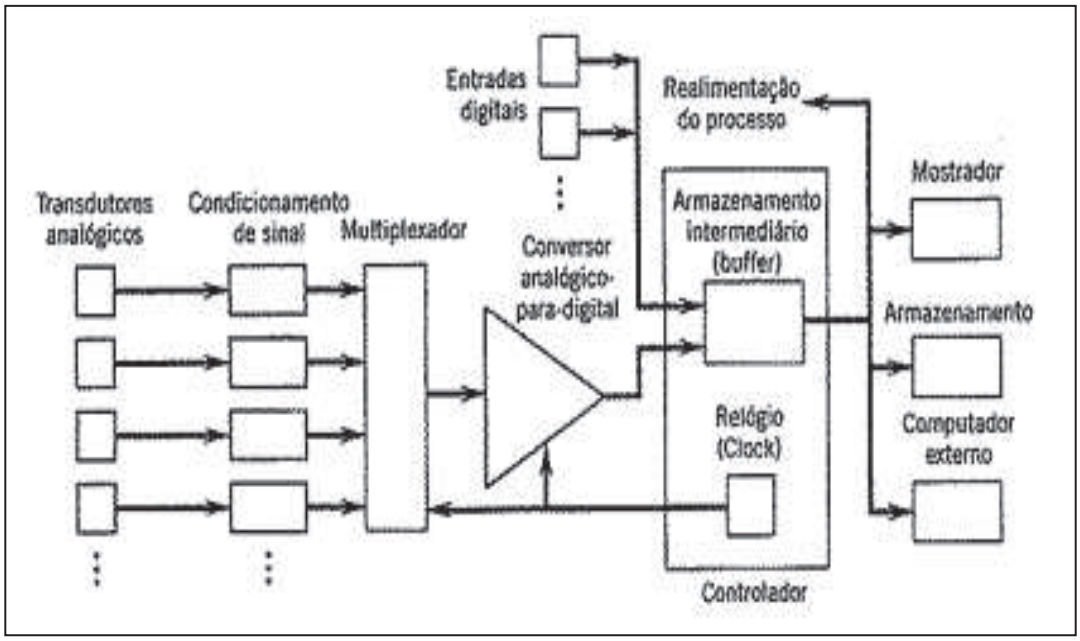


Figura 18: Esquema de fluxo de sinal para um DAS automatizado
Fonte: FIGLIOLA e BEASLEY (2007)

2.6 Linhas de Tendência

2.6.1 – Ajuste Linear

Supondo que a relação linear entre as variáveis Y e X é satisfatória, podemos estimar a linha de regressão e resolver alguns problemas de inferência. O problema de estimar os parâmetros β_0 e β_1 é o mesmo que ajustar a melhor reta em um gráfico de dispersão. O Método dos Mínimos Quadrados é uma eficiente estratégia para estimar os parâmetros da regressão e sua aplicação não é limitada apenas às relações lineares.

2.6.1.1 Método dos Mínimos Quadrados

O primeiro passo na análise de regressão é obter as estimativas β_0 eβ_1 dos parâmetros do modelo. Os valores dessas estimativas serão obtidos a partir de uma amostra de n pares de valores (xi,Yi), i=1,...,n que correspondem a n pontos em um gráfico, conforme na Figura 19.

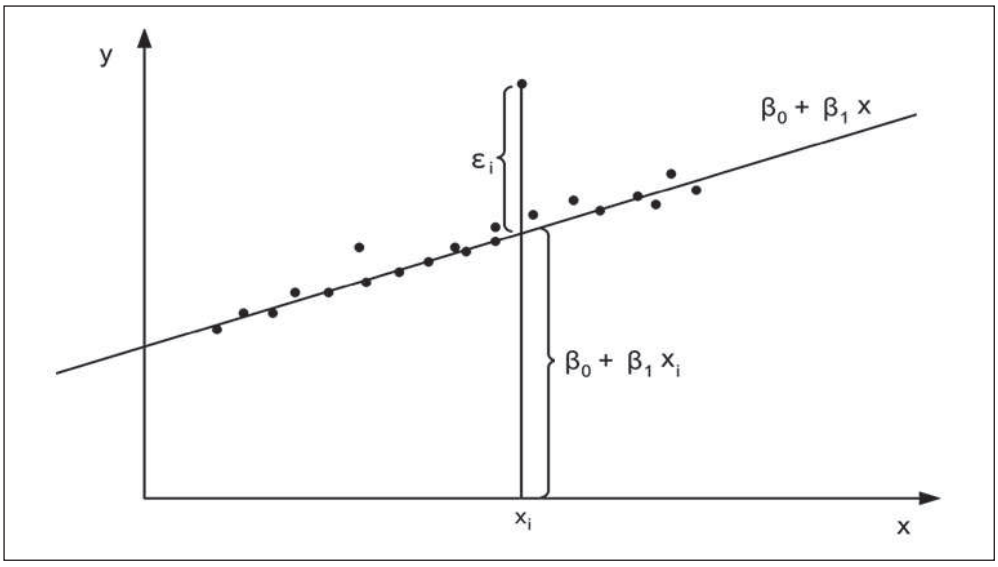


Figura 19: Representação da Reta de Regressão

No método de Mínimos Quadrados, não é necessário conhecer a forma da distribuição dos erros.

Supondo-se que é traçada uma reta arbitrária Y= β_0 + β_1.x passando por esses pontos. No valor x_i da variável independente, o valor previsto por esta reta é Yi=β_0 + β_1 .x_i , enquanto que o valor observado é y_i. Os desvios entre estes dois valores é $\varepsilon_i = y_i - [\beta_0 + \beta_1 . x_i]$, que corresponde a distância vertical do ponto à reta arbitrária.

O objetivo é estimar os parâmetros β_0 e β_1 de modo que os desvios (ε_i) entre os valores observados e estimados sejam mínimos. Isso equivale a minimizar o comprimento do vetor de erros, $\varepsilon_i = (\varepsilon_1, \varepsilon_2, \ldots\ldots \varepsilon_n)^x$.

Uma forma de obter essas estimativas é o Método de Mínimos Quadrados. Este método consiste em minimizar a soma dos quadrados dos desvios L, como na equação abaixo:

$$L = \sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2 \qquad \text{Eq. 01}$$

Para encontrar estimativas para os parâmetros, deve-se minimizar a equação 01 em relação aos parâmetros β_0 e β_1. Para isto, deriva-se em relação aos parâmetros β_0 e β_1. Assim,

$$\frac{\partial L(\beta_0, \beta_1)}{\partial \beta_0} = -2 \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i) \qquad \text{Eq. 02}$$

e,

Eq. 03

$$\frac{\partial L(\beta_0, \beta_1)}{\partial \beta_0} = -2 \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i) x_i$$

Substituindo β_0 e β_1 por $\widehat{\beta}_0$ e $\widehat{\beta}_1$, para indicar valores particulares dos parâmetros que minimizam L, e igualando as derivadas parciais a zero, obtém-se:

$$-2 \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) = 0 \qquad \text{Eq. 04}$$

e,

$$-2 \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) x_i = 0 \qquad \text{Eq. 05}$$

Simplificando, obtém-se as equações denominadas Equações Normais de Mínimos Quadrados.

$$\begin{cases} n\hat{\beta}_0 + \hat{\beta} \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i & \text{Eq. 06} \\ \hat{\beta}_0 \sum_{i=1}^{n} x_i + \hat{\beta}_1 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} x_i y & \text{Eq. 07} \end{cases}$$

Para encontrar os valores de $\widehat{\beta}_0$ e $\widehat{\beta}_1$ que minimizam L, resolve-se o sistema composto pelas equações 06 e 07. Da equação 06, temos:

$$\hat{\beta}_0 = \frac{\sum_{i=1}^{n} y_i}{n} - \frac{\hat{\beta}_1 \sum_{i=1}^{n} x_i}{n} \qquad \text{Eq. 08}$$

ou seja,

$$\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1 \bar{x} \qquad \text{Eq. 09}$$

em que $\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \quad \text{e} \quad \bar{Y} = \frac{1}{n} \sum_{i=1}^{n} Y_i$ são as médias das variáveis x e Y, respectivamente.

Desta forma, substituindo a equação 09 em 06, teremos:

$$\hat{\beta}_1 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} x_i y_i - \hat{\beta}_0 \sum_{i=1}^{n} x_i$$

$$= \sum_{i=1}^{n} x_i y_i - (\bar{y} - \hat{\beta}_1 \bar{x}) \sum_{i=1}^{n} x_i$$

$$= \sum_{i=1}^{n} x_i y_i - \bar{y} \sum_{i=1}^{n} x_i - \hat{\beta}_1 \bar{x} \sum_{i=1}^{n} x_i$$

$$= \sum_{i=1}^{n} x_i y_i - n \bar{x} \bar{y} + n \hat{\beta}_1 \bar{x}^2$$

$$\hat{\beta}_1 \left(\sum_{i=1}^{n} x_i^2 - n\bar{x}^2 \right) = \sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y}$$

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n} x_i^2 - n\bar{x}\bar{y}}{\sum_{i=1}^{n} x_i^2 - n\bar{x}^2} \qquad \text{Eq. 10}$$

Logo, substituindo 10 em 09, temos:

$$\hat{\beta}_0 = \bar{y} - \frac{\sum x_i y_i - n\bar{x}\bar{y}}{\sum x_1^2 - n\bar{x}^2}$$

$$\hat{\beta}_0 = \bar{y} - \frac{\bar{x} \sum x_i y_i - n\bar{x}^2 \bar{y}}{\sum x_1^2 - n\bar{x}^2} \qquad \text{Eq. 11}$$

A equação da reta de ajuste dos pontos experimentais será:

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 x \qquad \text{Eq. 12}$$

$$\hat{\beta}_0 = \bar{y} - \frac{\bar{x} \sum x_i y_i - n\bar{x}^2 \bar{y}}{\sum x_1^2 - n\bar{x}^2} \qquad \text{Eq. 13}$$

$$\hat{\beta}_1 = \frac{\sum x_i y_i - n\bar{x}\bar{y}}{\sum x_1^2 - n\bar{x}^2} \qquad \text{Eq. 14}$$

Os valores de $\widehat{\beta_0}$ e $\widehat{\beta_1}$ assim determinados são chamados Estimadores de Mínimos Quadrados (EMQ).

A equação da reta de ajuste dos pontos experimentais será:

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 x \qquad \text{Eq. 15}$$

sendo que $\widehat{Y}$ é um estimador pontual da média da variável Y para um valor de x, ou seja,

$$\widehat{E(Y|x_i)} = \widehat{\beta}_0 + \widehat{\beta}_1 x_i, \quad i = 1, \ldots, n.$$

Notação:

Considerando n pares de valores observados $(x_1,y_1),...,(x_n,y_n)$, podemos reescrever a equação 10 como sendo:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i,$$

$$S_{xx} = \sum_{i=1}^{n}(x_i-\bar{x})^2 = \sum_{i=1}^{n}(x_i-\bar{x})x_i = \sum_{i=1}^{n} x_i^2 - n\bar{x}^2 = \sum_{i=1}^{n} x_i^2 - \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n}, \qquad \text{Eq. 16}$$

$$S_{yy} = \sum_{i=1}^{n}(y_i-\bar{y})^2 = \sum_{i=1}^{n}(y_i-\bar{y})y_i = \sum_{i=1}^{n} y_i^2 - n\bar{y}^2 = \sum_{i=1}^{n} y_i^2 - \frac{\left(\sum_{i=1}^{n} y_i\right)^2}{n}$$

$$\text{e } S_{xy} = \sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y}) = \sum_{i=1}^{n}(x_i-\bar{x})y_i = \sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y}. \qquad \text{Eq. 17}$$

As quantidades $\bar{x}$ e $\bar{y}$ são as médias amostrais de x e y. Já as quantidades S_{xx} e S_{yy} são as somas dos quadrados dos desvios das médias e S_{xy} é a soma dos produtos cruzados dos desvios de x e y.

Desta forma, as estimativa de β_1 pelo método dos mínimos quadrados será:

$$\widehat{\beta}_1 = \frac{S_{xy}}{S_{xx}} \quad \text{e} \quad \widehat{\beta}_0 = \bar{Y} - \widehat{\beta}_1\bar{x}. \qquad \text{Eq. 18}$$

3 METODOLOGIA

A metodologia utilizada neste trabalho consta de pesquisa bibliográfica e de um estudo de caso que tem como evidência os experimentos realizados no laboratório de Tecnologia do Vácuo da Fatec São Paulo.

O estudo de caso foi escolhido como método de obtenção de dados, tendo em vista que as questões envolvidas nesta pesquisa requerem o entendimento de como são processados os experimentos em laboratórios de pesquisa.

A pesquisa bibliográfica dá o embasamento teórico necessário aos assuntos pesquisados, além de permitir a escolha das variáveis a serem consideradas no estudo de caso.

Para um melhor entendimento dos procedimentos adotados, esta seção está organizada da seguinte forma: Instrumentação Industrial e Tecnológica, Aquisição de Dados, Arduino, Termopar, Medidor Pirani e Desenvolvimento do Sistema de Tratamento de Dados.

3.1 Instrumentação Industrial e Tecnológica

Esta pesquisa foi desenvolvida utilizando os recursos cedidos pelo Laboratório de Tecnologia do Vácuo da Fatec São Paulo.

3.1.1 Tipos de instrumentos

Para a realização da pesquisa, foram utilizados os seguintes sensores: termopar tipo K (para medição de temperatura entre -200 ºC e 1200 ºC) e uma membrana capacitiva (para a medição de pressão em sistemas de vácuo entre 10^{3} mbar e 10^{-2} mbar). Para verificar a qualidade de aquisição de dados, fez-se uma comparação direta entre valores adquiridos eletronicamente e com aqueles exibidos no visor do equipamento convencional de leitura dos transdutores (sensores).

A figura 20 representa o sistema onde foi utilizado o sensor de Membrana Capacitiva nesta pesquisa.

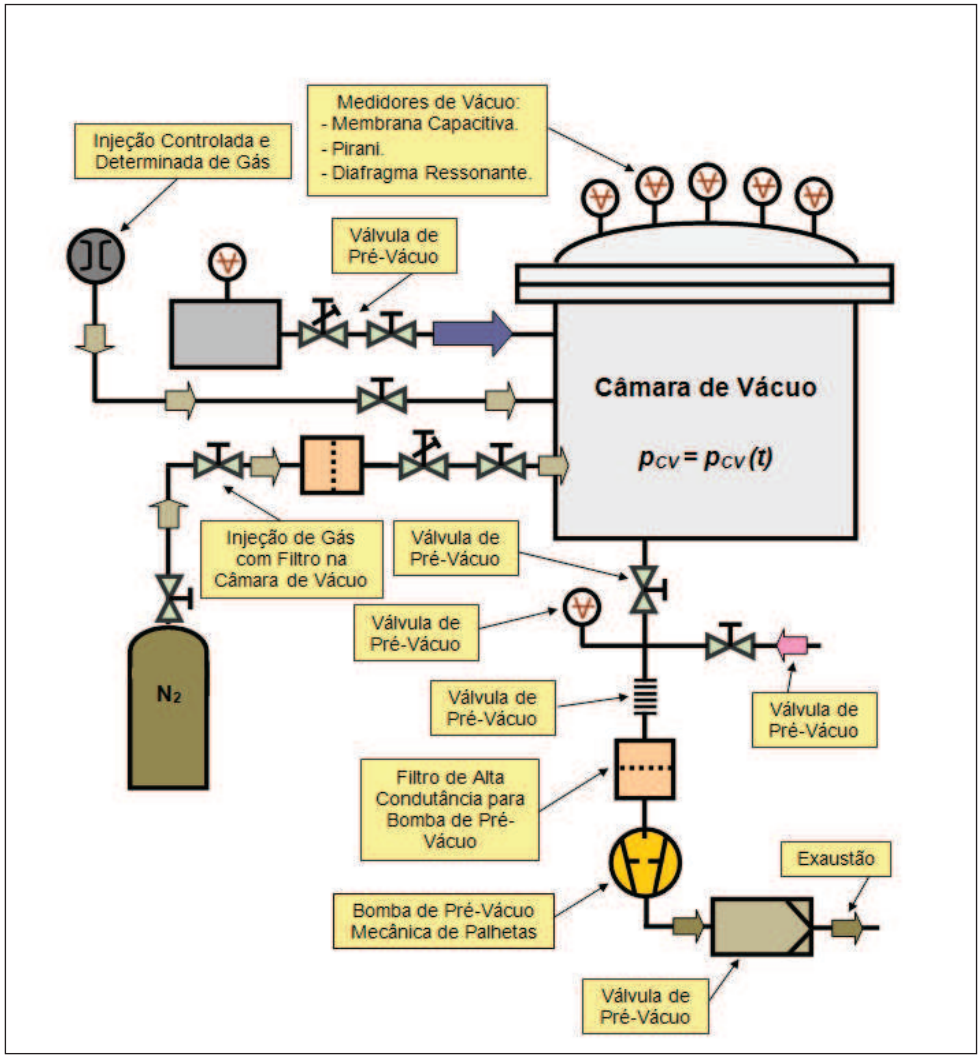


Figura 20: Sistema de vácuo para calibração de medidores
Fonte: Laboratório de Tecnologia do Vácuo (LTV) – Fatec São Paulo

3.1.2 Tipos de laboratórios

Uma vez que o trabalho de pesquisa em mestrado profissionalizante é bastante amplo em suas aplicações, dentro das áreas de pesquisa cuja aquisição de dados experimentais se faz necessária, temos que os tipos de laboratórios são todos que efetuam medições com transdutores com saídas analógicas de 0 volt até aproximadamente 10 volts.

3.2 Desenvolvimento de interface para aquisição de dados:

Numa primeira etapa, foram levantados requisitos necessários para aquisição e tratamento de dados provenientes de sensores que produzem uma tensão elétrica.

Considerando que o sistema inicialmente foi desenvolvido para um termopar do tipo K, cuja tensão gerada varia entre -6,458 mV e 48,838 mV com uma sensibilidade de

aproximadamente 0,47 μV/°C, fez-se necessário pesquisar por uma interface que oferecesse recursos com a maior precisão possível.

Sendo assim, foram identificados dispositivos disponíveis para a aquisição de dados. Existem no mercado diversas soluções prontas para aquisição dos dados dessa natureza, como os da empresa National Instruments. Porém, são tecnologias proprietárias, com custo relativamente alto. Buscando uma solução de baixo custo, optou-se pelo desenvolvimento da interface para a captura de dados a partir da placa microcontroladora Arduino UNO combinada com um amplificador operacional MAX31855.

A escolha do Arduino se deu devido à facilidade de acesso ao equipamento, pois trata-se de tecnologia *open source* e de baixo custo, além de oferecer relativa simplicidade no desenvolvimento da aplicação. Já o amplificador operacional MAX31855, embora seja uma solução proprietária, atende às necessidades do projeto a um custo relativamente baixo.

3.2.1 Arduino

Uma vez que o sinal de entrada do Arduino seja garantida, é necessário um software para converter o valor da tensão identificada na grandeza a ser medida. Foi então usada a IDE (Ambiente de Desenvolvimento Integrado) que é disponibilizada no site do fabricante. Por meio dessa IDE, foi desenvolvida uma aplicação em linguagem C, para a conversão dos valores de tensão na grandeza medida. Esta aplicação, uma vez tendo sido desenvolvida no computador, é transmitida por meio da porta USB ao *bootloader* do Arduino, onde é gravada e de onde é executada quando o mesmo é ativado. Uma vez tendo realizado esse procedimento, para a aquisição de dados basta conectar o PC ao Arduino por meio da porta USB, que além de transmitir os dados, também alimenta o Arduino.

A placa Arduino UNO possui 6 entradas analógicas que podem ser utilizadas para aquisição de sinais provenientes de sensores que geram uma tensão elétrica. A leitura analógica permite que o Arduino traduza a tensão em valores que variam de 0 até 1023. Isso é realizado por um circuito interno, chamado Conversor Analógico Digital (CAD). O CAD faz a conversão da tensão e seleciona 10 bits de acordo com a tensão, ou seja, resulta em um valor com 10 bits de resolução.

$$2^{10} = 1024 \text{ estados}$$

Para realizar a leitura, deve-se utilizar a função “analogRead(pino)”, onde “pino” deve ser o número do pino onde se deseja fazer a leitura. Essa função retorna um valor inteiro de 0 a 1023.

Para converter esse valor em voltagem, foi feito o seguinte cálculo:

$$V = V_a \,.\, (5 / 1023) \qquad \text{Eq. 19}$$

Onde V_a = leitura analógica, 5 é a tensão máxima e 1023 o valor máximo de leitura analógica.

O código foi escrito tendo como referência o termopar tipo K, cujas temperaturas de medição variam entre -200° e 1200° C. Considerando que o Arduino reconhece sinais entre 0 e 5 volts, têm-se como referência que a temperatura mínima é 0 e a máxima 5 volts. Dentro deste espectro é calculada a temperatura, de acordo com a voltagem recebida.

A figura 21 apresenta a correlação entre a *fem* produzida e a temperatura correspondente nos termopares do tipo E, J, K e T:

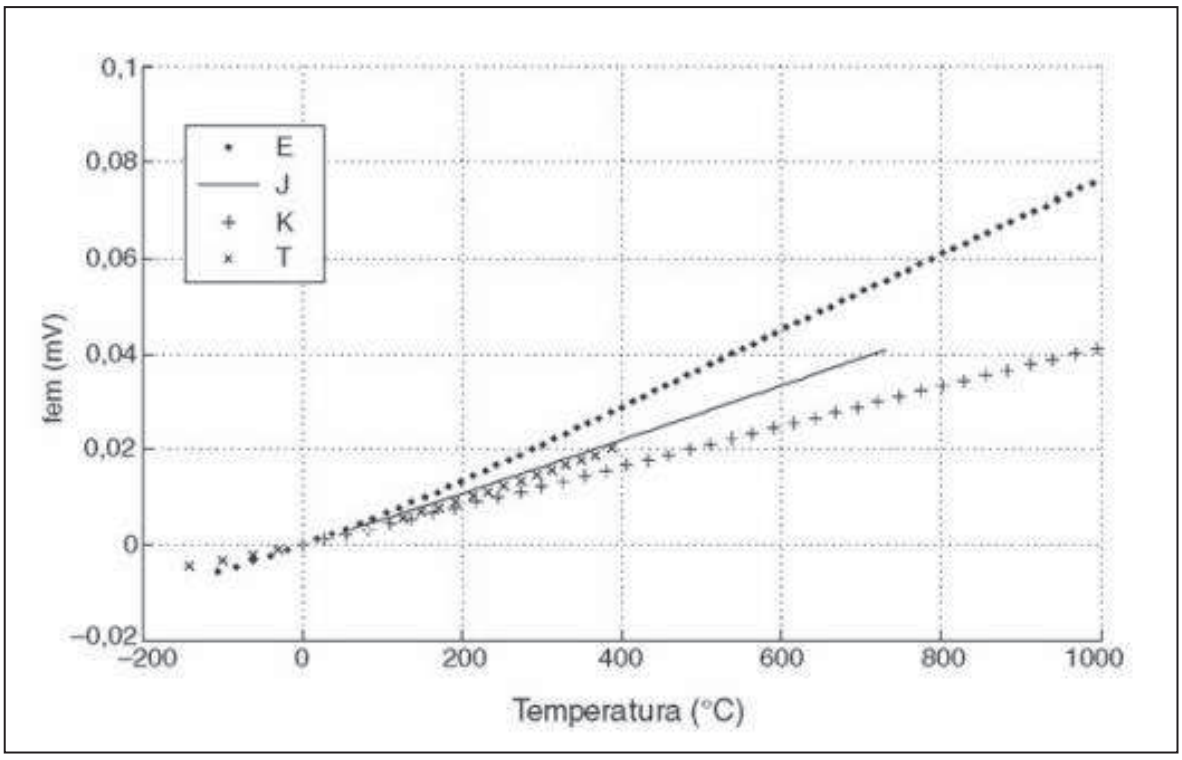


Figura 21: Características de alguns tipos de termopares com junção de referência a 0°C
Fonte: BALBINOT E BRUSAMARELLO (2010, p. 295)

3.2.2 Conversão de Tensão para Temperatura

Como a relação entre a *fem* e a Temperatura não é linear, é preciso de algum modo linearizar o sinal gerado pelo sensor. A equação matemática que descreve a curva do sensor é:

$$T = A0 + A1.X^1 + A2.X^2 + A3.X^3 + A4.X^4 \,.... + An.X^n \qquad \text{Eq. 20}$$

onde: T: temperatura

A: o coeficiente de cada termo do polinômio

X: a tensão elétrica gerada

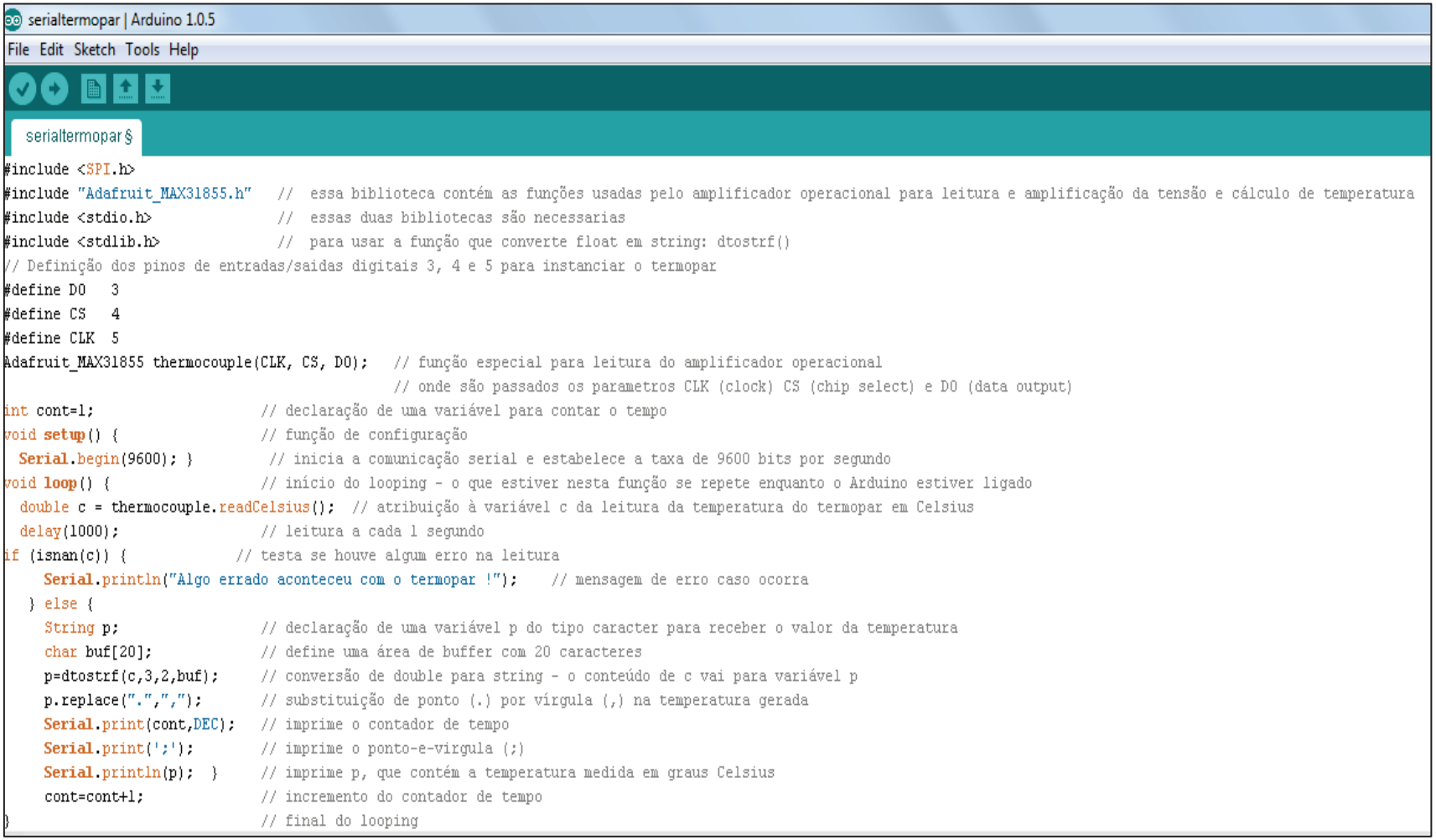
n: a ordem do polinômio

Para a aplicação da fórmula, utilizam-se os coeficientes demonstrados no Quadro 8:

Quadro 8: Coeficientes para os termopares E, J, K e T

	Tipo E	Tipo J	Tipo K	Tipo T
	−100 a 1000 °C	**0 a 760 °C**	**0 a 1370 °C**	**−160 a 400 °C**
a_0	0,104967248	−0,048868252	0,226584602	0,100860910
a_1	17189, 45282	19873, 14503	24152, 10900	25727, 94369
a_2	−282639, 0850	−218614, 5353	67233, 4248	−767345, 8295
a_3	12695339, 5	11569199, 78	2210340, 682	78025595, 81
a_4	−448703084, 6	−264917531, 4	−860963914, 9	−9247486589
a_5	$1{,}10866 \times 10^{+10}$	2018441314	$4{,}83506 \times 10^{+10}$	$6{,}97688 \times 10^{+11}$
a_6	$-1{,}76807 \times 10^{+11}$		$-1{,}18452 \times 10^{+12}$	$-2{,}66192 \times 10^{+13}$
a_7	$1{,}71842 \times 10^{+12}$		$1{,}38690 \times 10^{+13}$	$3{,}94078 \times 10^{+14}$
a_8	$-9{,}19278 \times 10^{+12}$		$-6{,}33708 \times 10^{+13}$	
a_9	$2{,}06132 \times 10^{+13}$			

Para a leitura desse sensor, o software do Arduino foi configurado conforme demonstrado na figura 22, a seguir.




```
#include <SPI.h>
#include "Adafruit_MAX31855.h"   //  essa biblioteca contém as funções usadas pelo amplificador operacional para leitura e amplificação da tensão e cálculo de temperatura
#include <stdio.h>               //  essas duas bibliotecas são necessarias
#include <stdlib.h>              //  para usar a função que converte float em string: dtostrf()
// Definição dos pinos de entradas/saidas digitais 3, 4 e 5 para instanciar o termopar
#define DO   3
#define CS   4
#define CLK  5
Adafruit_MAX31855 thermocouple(CLK, CS, DO);   // função especial para leitura do amplificador operacional
                                               // onde são passados os parametros CLK (clock) CS (chip select) e DO (data output)
int cont=1;                    // declaração de uma variável para contar o tempo
void setup() {                 // função de configuração
  Serial.begin(9600); }         // inicia a comunicação serial e estabelece a taxa de 9600 bits por segundo
void loop() {                  // início do looping - o que estiver nesta função se repete enquanto o Arduino estiver ligado
  double c = thermocouple.readCelsius();  // atribuição à variável c da leitura da temperatura do termopar em Celsius
  delay(1000);                 // leitura a cada 1 segundo
if (isnan(c)) {             // testa se houve algum erro na leitura
    Serial.println("Algo errado aconteceu com o termopar !");    // mensagem de erro caso ocorra
   } else {
    String p;                  // declaração de uma variável p do tipo caracter para receber o valor da temperatura
    char buf[20];              // define uma área de buffer com 20 caracteres
    p=dtostrf(c,3,2,buf);      // conversão de double para string - o conteúdo de c vai para variável p
    p.replace(".",",");        // substituição de ponto (.) por virgula (,) na temperatura gerada
    Serial.print(cont,DEC);    // imprime o contador de tempo
    Serial.print(';');         // imprime o ponto-e-virgula (;)
    Serial.println(p);  }      // imprime p, que contém a temperatura medida em graus Celsius
    cont=cont+1;               // incremento do contador de tempo
}                              // final do looping
```

Figura 22: Código fonte do Arduino para aquisição de dados do termopar escrito em linguagem C

O resultado gerado pode ser observado diretamente no monitor de porta serial da IDE conforme visto na figura 23:

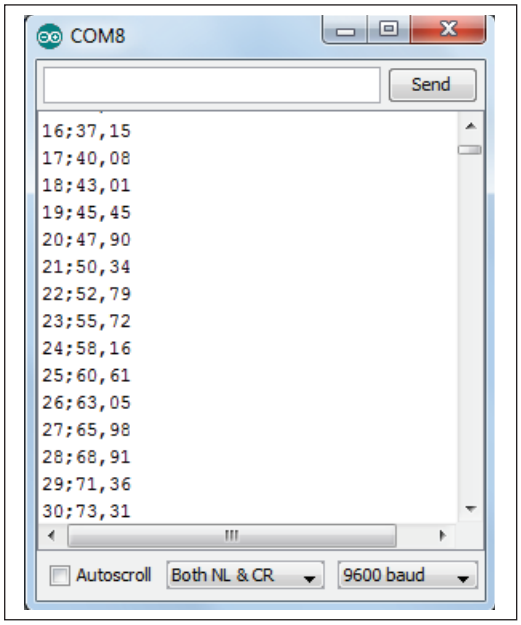


Figura 23: Resultado da aquisição de dados observado pelo monitor serial do Arduino

3.2.3 Termopar

No caso do experimento com o sensor termopar, foi necessária a utilização de um amplificador operacional para que o sinal chegasse até a entrada analógica de forma adequada, ou seja, dentro de uma faixa de 0 a 5 volts e sem interferência de ruídos externos. Existem algumas opções comerciais de amplificadores operacionais desenvolvidos para utilização de termopares no controlador Arduino e o amplificador utilizado nesse arranjo foi o MAX 31855, cujas características são:

- Compensação de junta fria
- Faixa de temperatura de 200°C até 1350°C Resolução de 14 bits ou 0,25° C
- Sensor de temperatura interno de 40°C até 125°C
- Comunicação SPI com entrada de %V ou 3,3V
- Filtro na entrada do Termopar
- Específico para Termopares do tipo K
- Consumo máximo de corrente de aproximadamente 1,5 mA

A figura 24 demonstra a comunicação entre o amplificador operacional e o microcontrolador, neste caso o Arduino UNO:

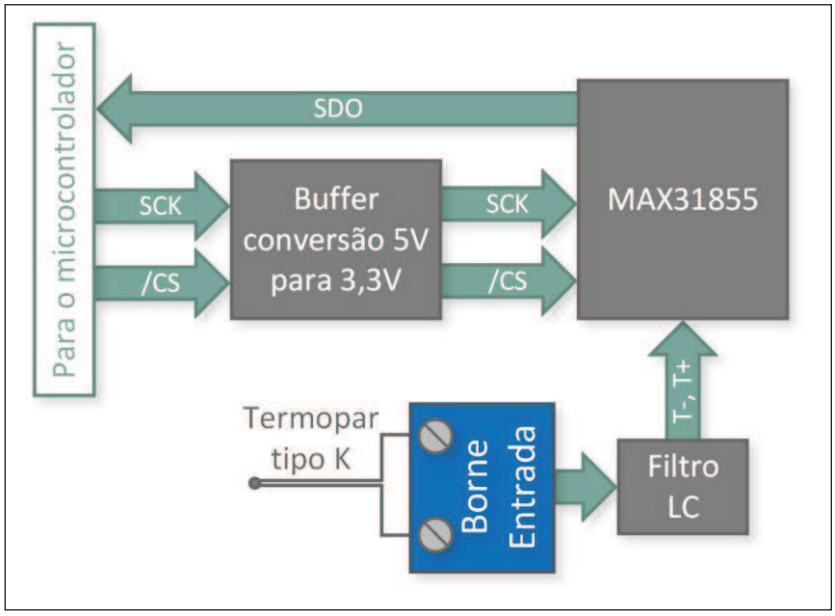


Figura 24: Diagrama de blocos do amplificador operacional MAX31855
Fonte: CIRCUITAR(2014)

Onde temos:

- /CS – Chip Select barramento SPI
- SCK – Linha de clock barramento SPI
- SDO – Linha de dados Barramento SPI

A alimentação é feita pelo pino 3V3 com intervalo de 3V até 3,6V e os pinos de entrada /CS e SCK funcionam com tensões de 5,0V ou 3,3V. O pino de saída SDO tem nível lógico de 3V3 e é 100% compatível com os níveis de tensão aceitos pelo Arduino. O pino GND serve como tensão de referência. A representação esquemática do MAX31855 pode ser encontrada no ANEXO C. A figura 25 apresenta como ficou a conexão entre o amplificador operacional MAX31855 e o Arduino:

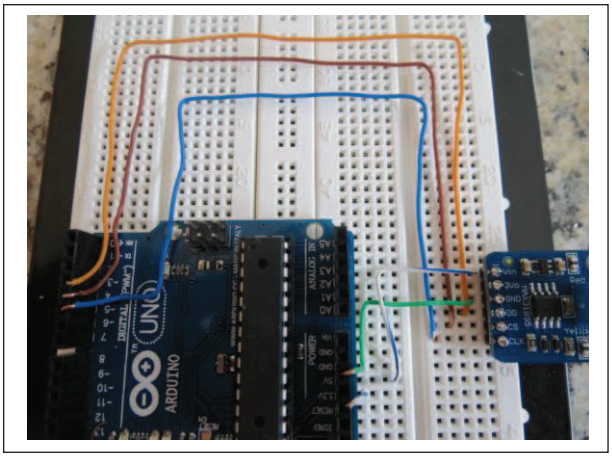


Figura 25: Conexão do MAX31855 ao Arduino

Após a conexão foram realizados testes de medição de temperatura utilizando um Termopar do tipo K. O termopar foi aquecido e resfriado ao longo de 1 minuto e nesse intervalo os dados foram coletados para observação a cada 1 segundo. A programação realizada no Arduino precisou da biblioteca de funções “Adafruit_MAX31855.h” fornecida pelo fabricante do MAX31855, para gerar o valor da temperatura em graus centígrados. Como resultado foi exibido um contador de tempo (s) e o valor da temperatura (T) em graus Celsius separados por um ponto e vírgula. Esta informação é enviada à porta serial e fica disponível para aquisição. A figura 26 apresenta o código fonte do Arduino contendo esta programação com seus respectivos comentários.

```
/**************************************************
  Código do Arduino para leitura de temperatura através do temopar tipo K
  e exibição do resultado na porta serial
 **************************************************/

#include <SPI.h>                      //  biblioteca que permite a conexão do microcontrolador com o PC
#include "Adafruit_MAX31855.h"        //  biblioteca que faz os cálculos necessários para a obtenção da temperatura
#include <stdio.h>                    //  essas duas bibliotecas são necessarias para ...
#include <stdlib.h>                   //  ... usar a função que converte float em string: dtostrf()


// Criando uma instância do termopar com o software SPI nos pinos digitais 3, 4 e 5
#define DO   3
#define CS   4
#define CLK  5
Adafruit_MAX31855 thermocouple(CLK, CS, DO);

void setup() {                        // função de configuração
  Serial.begin(9600);                 // iniciando a comunicação serial
  int s=0;                            // inicializando o contador de tempo em 0
            }

void loop() {                         // função de loop

  double c = thermocouple.readCelsius();// definindo que a variável c irá ser do tipo double e armazenará a temperatura do termopar em cels
  delay(1000);                        // leitura a cada 1 segundo
    int s=s+1;                        // incrementando o contador de tempo em 1
   if (isnan(c)) {                    // função que checa se a variável c está vazia
     Serial.println("Há algo errado com o termopar!");    // mensagem de erro caso variável c esteja vazia
   } else {
     String t;                        // cria uma variável t do tipo string
     char buf[20];                    // define o tamanho do buffer em 20 caracteres
     p=dtostrf(c,3,2,buf);            // converte o valor do buffer em string e armazena em t
     p.replace(".",",");              // substitui . por ,
     Serial.print(s,DEC);             // imprime o contador de tempo
     Serial.print(';');               // imprime o ;
     Serial.println(t);               // imprime o valor de t
   }
```

Figura 26: Código fonte para o Arduino ler e exibir a temperatura do termopar escrito na linguagem C

3.2.4 Sensor de Membrana Capacitiva

Para o experimento de aquisição de dados do sensor de Membrana Capacitiva BARATRON modelo MKS 626, utilizou-se o mesmo sistema demonstrado na figura 23. A medição e aquisição de dados utilizando o sistema proposto durou cerca de 120 segundos e a pressão observada oscilou entre 0 e 301,56 torr.

3.3 Desenvolvimento do software para tratamento de dados

Para o desenvolvimento do software de tratamento dos dados, foi escolhida a IDE *open source* Lazarus, que utiliza o compilador Free Pascal e permite criar aplicações para Windows, Linux, MacOS, FreeBSD e outros sistemas operacionais. Essa ferramenta foi desenvolvida para compilar código e gerar executáveis para diferentes plataformas a partir de um mesmo código fonte, desde que utilizado o compilador adequado. Neste trabalho foi utilizada a versão 1.1.2 para o sistema operacional Windows 64 bits.

O software desenvolvido recebeu o nome de SISTRADA – Sistema de Tratamento de Dados e tem como principais funcionalidades a importação de arquivo contendo os dados adquiridos, a geração de gráficos a partir do arquivo importado e a geração de relatório após a análise dos dados. O código fonte do software SISTRADA pode ser visto no Apêndice B.

As etapas de desenvolvimento foram: identificação dos requisitos do software, modelagem do sistema, codificação, testes e implantação.

3.3.1 – Identificação dos requisitos do software

Os requisitos do software podem ser definidos como as necessidades que precisam ser supridas pelo sistema a ser desenvolvido e foram levantadas por meio de entrevista com pesquisadores, observação de experimentos e análise de sistemas já existentes. Os requisitos foram identificados e descritos conforme quadro 9.

Quadro 9: Descrição dos requisitos para o desenvolvimento do software

Requisito	Descrição
1	O software deve possuir opção para selecionar o tipo de experimento
2	O software deve permitir a configuração da porta serial a ser lida
3	O software deve permitir a leitura da porta serial durante a execução do experimento e gravação dos dados em arquivo.
4	O software deve possibilitar a importação de arquivo com dados adquiridos de experimentos já realizados.
5	O software deve permitir a criação de gráfico demonstrando a evolução da grandeza medida em função do tempo
6	O software deve permitir a edição dos campos: Título do Gráfico, Título do Eixo X, Título do Eixo Y, Observações
7	O software deve permitir o melhor ajuste de curvas do gráfico

3.3.2 Modelagem do Sistema

Modelagem de software é uma representação simplificada de algo real. Ao modelarmos um software identificamos o que ele deverá fazer no futuro, além de tratar as questões funcionais e seus fluxos de dados. (CASTILHO, 2008). A ferramenta utilizada para a modelagem foi a UML 2.0 (Unified Modeling Language), que possui diversos tipos de diagramas que representam visões diferentes do projeto. Neste trabalho utilizou-se o Diagrama de Caso de Uso, que mostra as funcionalidades a serem implementadas no sistema, do ponto de vista do usuário final.

A figura 27 apresenta o Diagrama de Caso de Uso Geral, contemplando a visão mais abrangente do sistema.

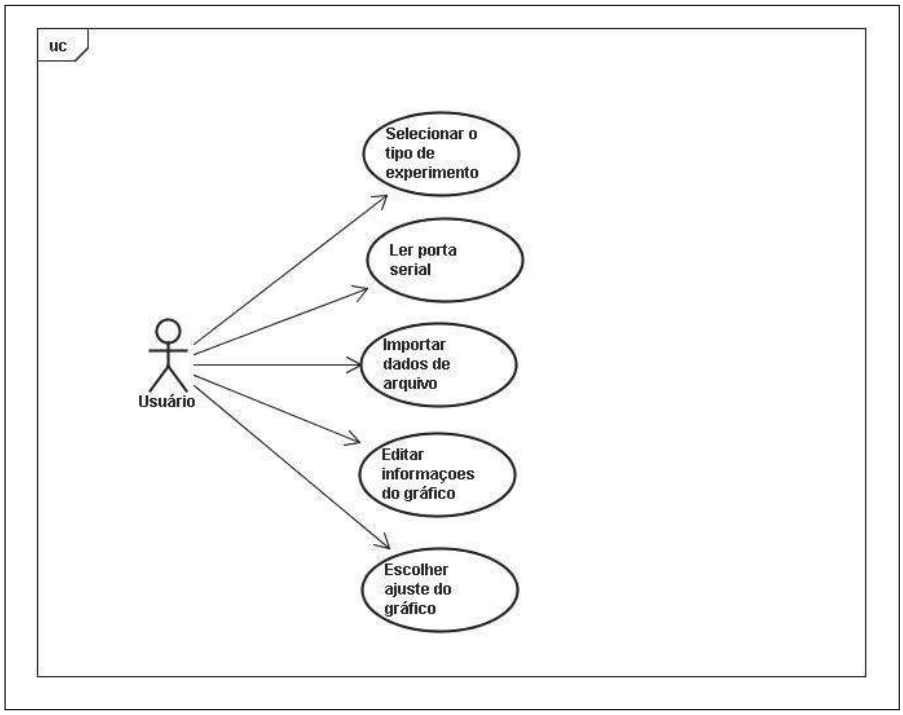


Figura 27: Diagrama de Caso de Uso Geral do sistema Sistrada

O layout do sistema foi disposto em forma de abas onde podem ser acessados todos os recursos do sistema. O primeiro aspecto a ser desenvolvido foi o de conexão com a porta serial. Para isso, foi instalado no Lazarus o componente **5dpo (**http://sourceforge.net/projects/sdpo-cl/files/**)**, que é responsável permitir a comunicação com o dispositivo externo e mantém os seguintes parâmetros de configuração da porta serial: Device (porta), BaudRate, Parity, StopBits, DataBits e FlowControl. Esses parâmetros são configuráveis pelo usuário e podem variar conforme o equipamento a ser conectado. Para saber a configuração adequada é necessário consultar o manual do dispositivo fornecido pelo

fabricante. A figura 28 apresenta a tela de configuração já com dados sugeridos como padrão, mas que podem ser modificados pelo usuário.

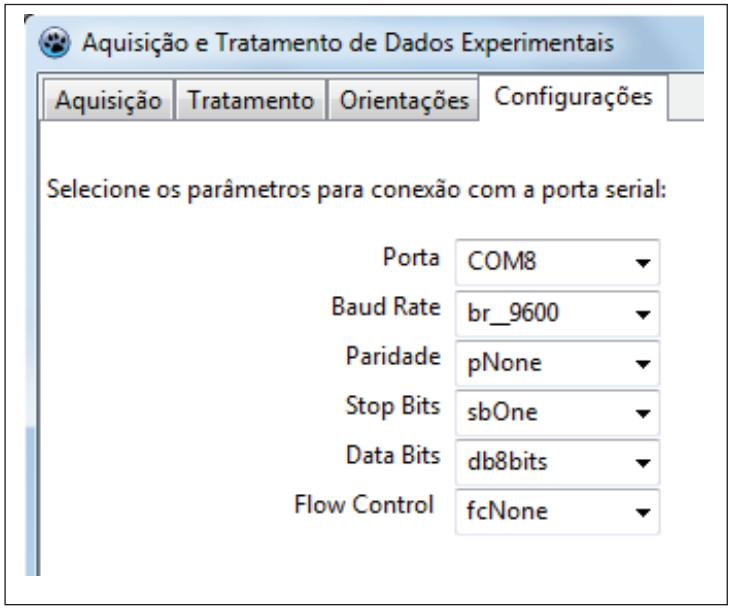


Figura 28: Tela de configuração de parâmetros para conexão com a porta serial

Uma vez estabelecida a conexão com a porta serial, inicia-se o processo de Aquisição de Dados pelo botão Conectar, onde os dados capturados irão aparecer em uma caixa na tela e ao final do experimento, ao clicar em Desconectar, será solicitado um nome para salvar o arquivo. Caso seja escolhido um nome de arquivo existente, o mesmo será sobrescrito. O arquivo contém os mesmos dados que aparecem na tela durante a aquisição. A figura 29 exemplifica o processo de aquisição.

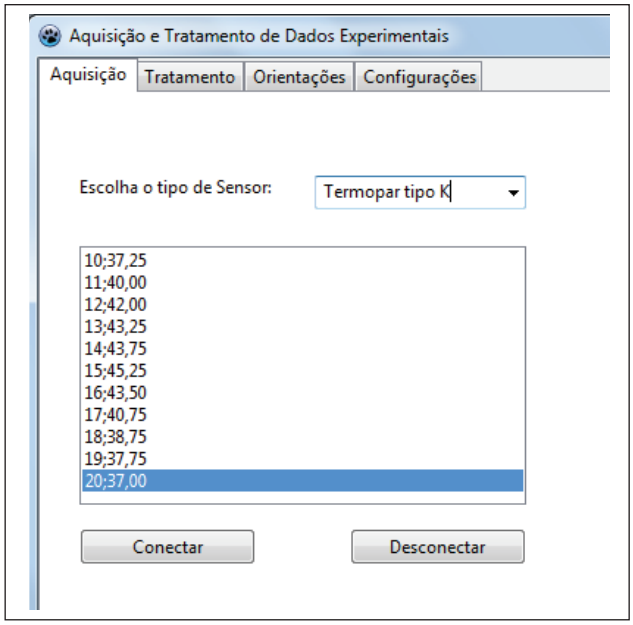


Figura 29: Processo de Aquisição de Dados

Para o tratamento de dados, a interface oferece os seguintes campos para preenchimento conforme o experimento: Título do gráfico, Título do Eixo X, Título do Eixo Y e Observações. O botão Importar permite que o usuário escolha o arquivo de dados que irá gerar o gráfico. Também é possível utilizar uma ferramenta que amplia a área do gráfico e ainda selecionar a caixa exibir os valores do gráfico.

O arquivo a ser importado deve ser no formato .txt e os dados devem estar dispostos da seguinte forma: tempo e grandeza medida separados por ponto e vírgula.

As figuras 30 e 31 exemplificam, respectivamente, o modelo de arquivo .txt a ser importado e a tela de tratamento de dados e geração do gráfico.

```
1;0,00
2;0,98
3;3,42
4;5,87
5;8,80
```

Figura 30: Formato dos dados armazenados em arquivo .txt

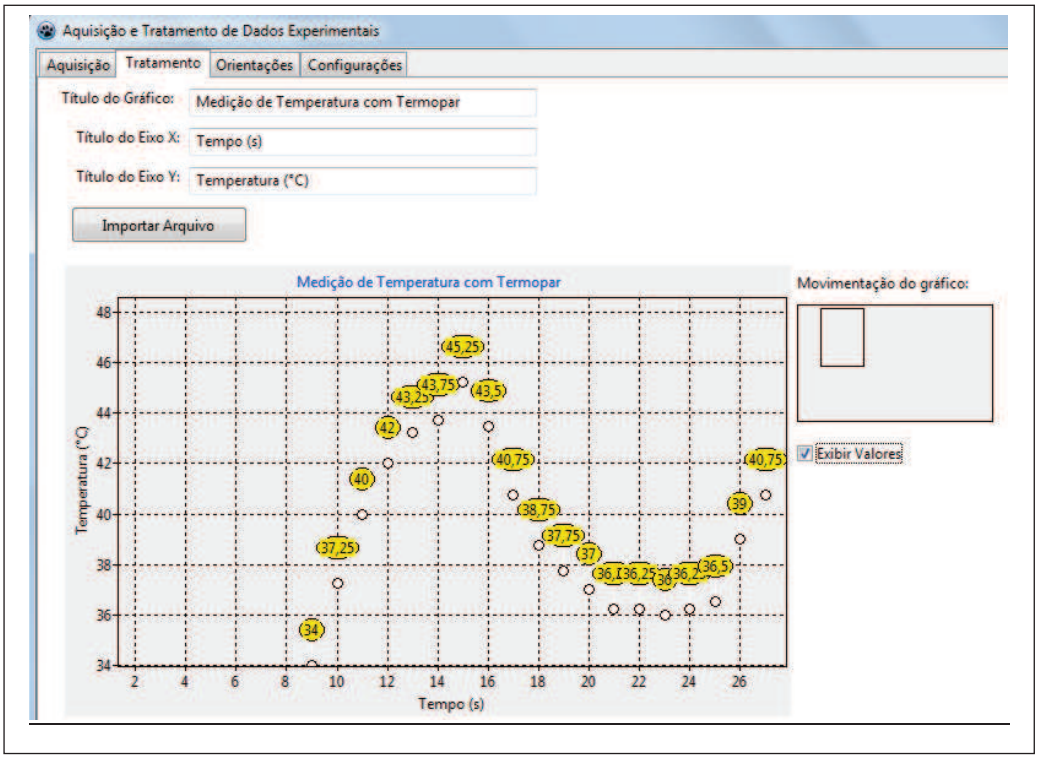


Figura 31: Tela de Tratamento dos dados – geração de gráfico

Após o gráfico ter sido gerado, é possível escolher a linha de tendência mais adequada, ou seja, a melhor reta de ajuste dos dados. Tendo selecionado uma das opções, uma linha de

tendência aparece sobre o gráfico além da função gerada. Um exemplo de gráfico com a utilização de linhas de tendência linear é mostrada na figura 32.

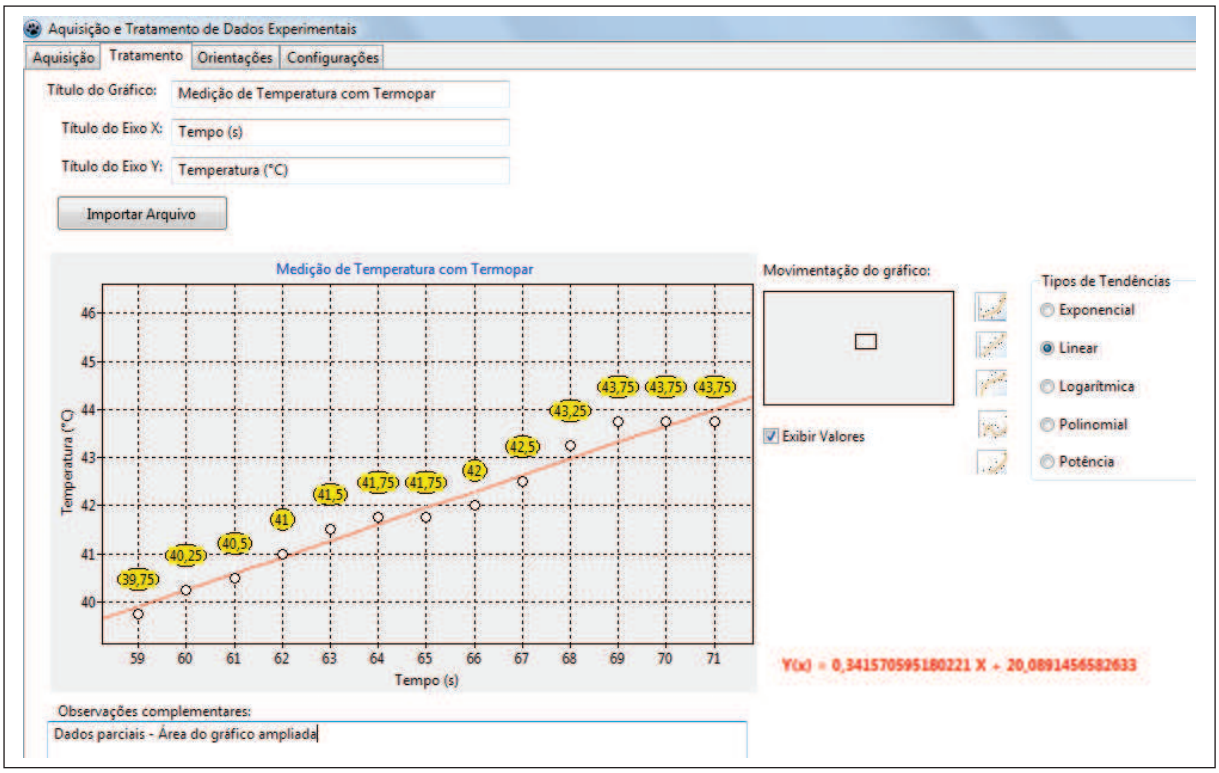


Figura 32: Tela de tratamento dos dados: Linha de Tendência – área do gráfico ampliada

O cálculo de cada curva envolve novamente a leitura do arquivo de dados, portanto, para evitar uma demora no cálculo, optou-se por não salvar os dados no formato de banco de dados, mas sim em arquivo texto.

4 RESULTADOS

Durante a pesquisa, pôde-se observar um método manual de aquisição e tratamento de dados experimentais realizado com medição de temperatura por termopares ligados à medidores digitais, onde os valores mostrados nos medidores eram filmados e posteriormente visualizados e digitados em uma planilha eletrônica, consumindo um tempo extra após a realização do experimento. Este método, além de ser mais demorado, apresenta maior possibilidade de erro no momento de digitar os valores observados.

Por meio do modelo proposto, alguns experimentos foram realizados com o uso de um sensor termopar tipo K conectado ao Arduino e, utilizando o amplificador operacional MAX31855, medindo a temperatura da água aquecida por uma resistência durante o período de aproximadamente 120 segundos. Para comparar a medição, foi montado um arranjo utilizando um medidor digital Minipa MT-405, também acoplado à um termopar tipo K, conforme figura 33.

Figura 33: Arranjo para comparação de valores

Simultaneamente ao experimento, foi feita a coleta e armazenamento dos dados em arquivo para posterior tratamento. O tempo de duração do experimento foi o tempo necessário para a aquisição e armazenamento dos dados, sem que houvesse nenhum tipo de interferência manual. O experimento durou cerca de 120 segundos e a temperatura oscilou entre 19º e 60,25ºC durante esse intervalo. A coleta dos dados ocorreu a cada 1 segundo.

Uma vez tendo realizado a aquisição dos dados no sistema proposto e, simultaneamente, em igualdade de condições com outro medidor digital, pode-se observar que

os valores são equivalentes. Deve-se ressaltar que no sistema proposto é fácil verificar a mudança da temperatura a cada 0,25º C ou, com sensibilidade de 14 bits do termopar.

Verifica-se então que, os valores apurados pelos dois métodos foram equivalentes, observando-se apenas que os resultados do sistema de medição proposto apresentam duas casas decimais e o medidor digital apresenta apenas o valor inteiro, conforme figura 34.

Figura 34: Comparativo entre valores apurados

Após a aquisição, foi gerado um gráfico com os dados importados do arquivo contendo como eixo X o tempo, medido em segundos e, como eixo Y a temperatura medida em graus centígrados, conforme demonstra a figura 35.

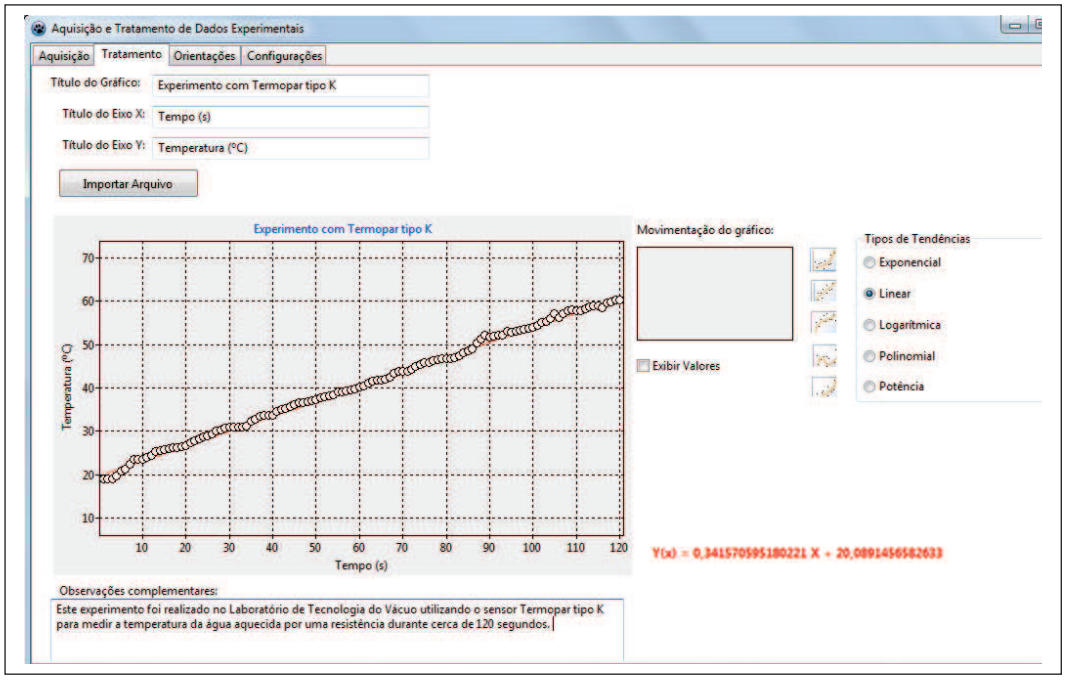


Figura 35: Gráfico do experimento com termopar tipo K

Na figura 36 pode-se observar uma área ampliada do gráfico com os valores dos pontos, que podem ser exibidos conforme necessidade, além da linha de tendência linear e também a função Y(x) relativa aos dados adquiridos.

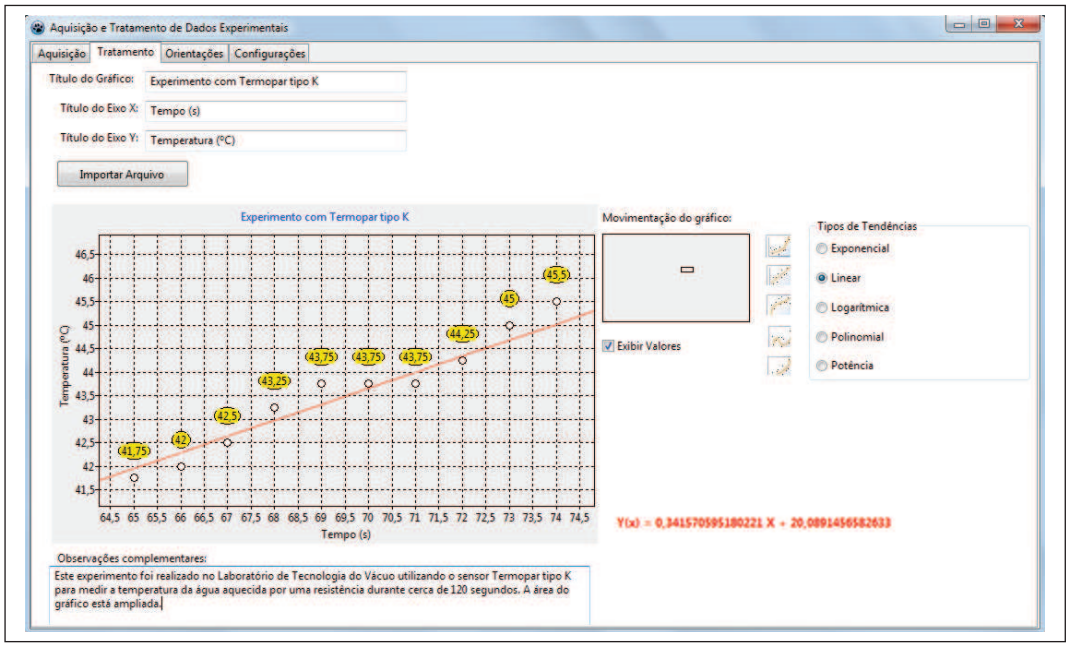


Figura 36: Área ampliada do gráfico com sua linha de tendência e a função Y(x)

Foram também realizados experimentos com o sensor de Membrana Capacitiva, onde o arranjo foi montado a partir da saída do sensor, conectando-se os fios diretamente nas portas analógica A1 e terra GND do Arduino, conforme figura 37. Os dados adquiridos resultam da conversão da tensão de saída em pressão, medida em torr.

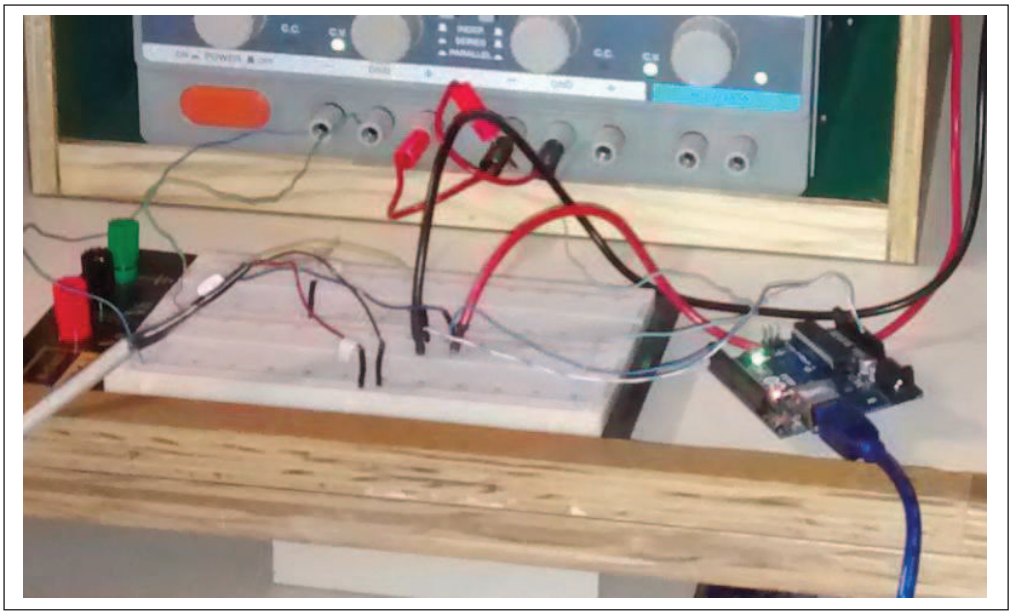


Figura 37: Arranjo para conexão com sensor de membrana capacitiva

Os dados foram adquiridos durante cerca de 150 segundos e os resultados foram comparados com os dados do multímetro Minipa MDM – 8165. O cálculo da pressão seguiu a fórmula “Pressão = voltagem * 100”. Conforme a figura 38, as leituras tanto pelo multímetro quanto pelo sistema automático de aquisição de dados são equivalentes.

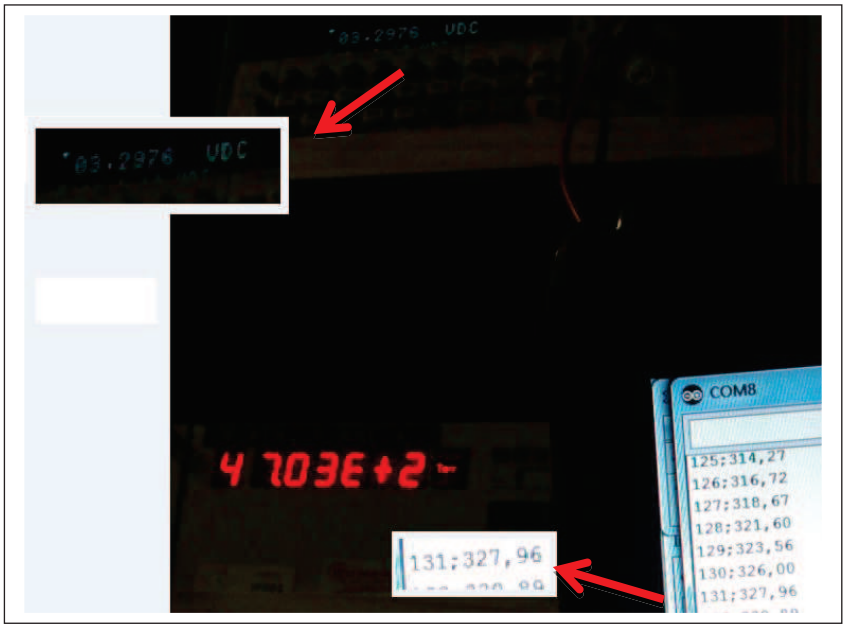


Figura 38: Comparação dos dados adquiridos pelo sistema automático de dados e pelo multímetro

A partir dos dados adquiridos, foi gerado o gráfico demonstrado na figura 39.

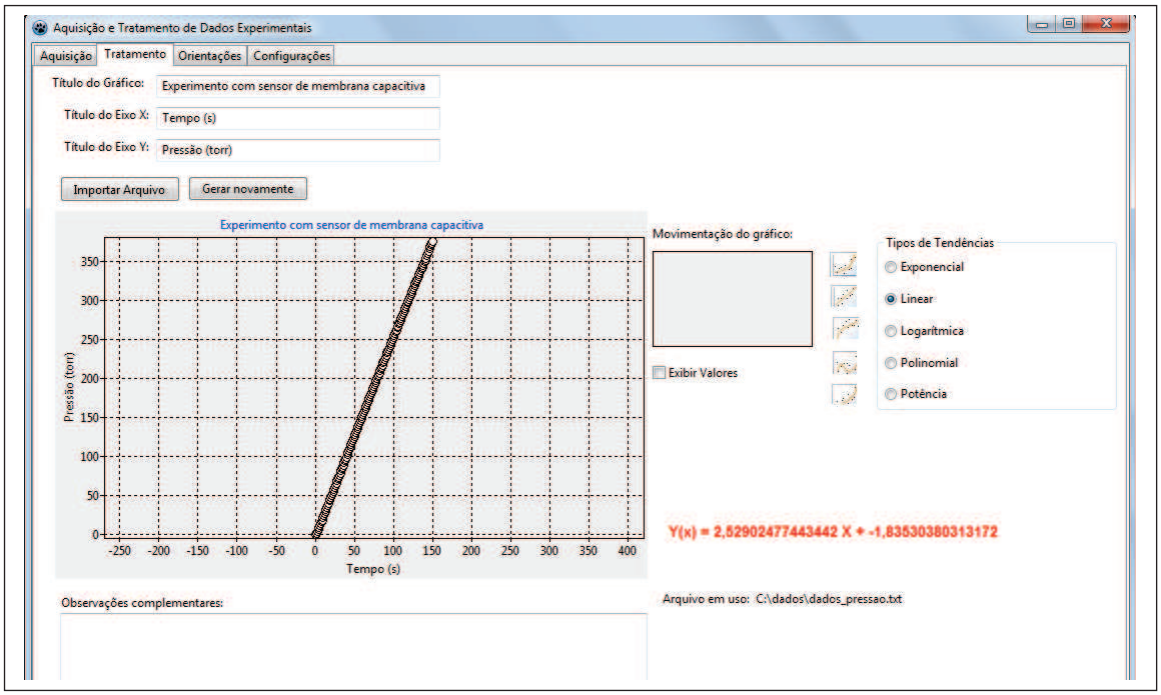


Figura 39: Gráfico com experimento do medidor de membrana capacitiva

5 CONCLUSÕES

Uma vez tendo realizado comparações entre os resultados obtidos pelo sistema de aquisição proposto e, outros dispositivos de medição, percebeu-se que os resultados são equivalentes, o que permite concluir que é viável utilizar o microcontrolador Arduino UNO como interface para aquisição de dados desde que os sensores gerem como resultado um sinal elétrico entre 0 e 5 Volts e os arranjos sejam devidamente elaborados de acordo com cada tipo de experimento. Cada experimento deve ter seu arranjo montado de forma a atender às características de cada sensor/transdutor, ou seja, adequar a leitura dos dados à tensão de saída dos sensores seja amplificando ou reduzindo a tensão, portanto, um conhecimento básico de eletrônica é necessário. No experimento realizado, foi montado um arranjo para leitura de um único termopar, porém, o Arduino UNO comporta a utilização de até 4 amplificadores operacionais do tipo MAX31855, ou seja, é possível fazer a leitura de até 4 termopares simultaneamente. A linguagem de programação do Arduino contempla intervalo de no mínimo 1 milissegundo, portanto esse é o intervalo mínimo entre a leitura de cada porta serial ou analógica.

A linguagem utilizada na IDE para o desenvolvimento da programação do Arduino é baseada na linguagem C, que é conhecida e relativamente simples, portanto fácil de ser implementada. Outro fator que merece destaque é que pelo fato de ser tecnologia *open source,* existem vários programas prontos à disposição, inclusive na própria IDE de desenvolvimento, no link “exemplos”. Há inúmeros com exemplos de aplicações já desenvolvidas utilizando o Arduino disponíveis na Internet, além de uma grande comunidade de desenvolvedores participantes de fóruns de discussão. O custo do desenvolvimento da interface de hardware neste experimento foi muito abaixo do valor de uma interface comercial, conforme demonstrado no Apêndice A e no Anexo E.

Para o desenvolvimento do software para aquisição e tratamento de dados existem soluções no mercado como a linguagem LabVIEW, que é uma linguagem de programação proprietária muito conhecida e utilizada no desenvolvimento de aplicações relacionadas à eletrônica, cujo custo da licença de uso é bem significativo, cerca de R$ 3.300,00 por um ano.

O software SISTRADA, ferramenta desenvolvida para o tratamento de dados, possibilitou a importação e tratamento dos dados de forma rápida e confiável. A IDE Lazarus na qual essa ferramenta foi desenvolvida, oferece recursos para ampliar as funcionalidades da ferramenta e torná-la ainda mais eficiente na análise dos experimentos e comparação com os resultados esperados. Outra característica importante da linguagem de programação usada

pela IDE Lazarus é que ela permite que a partir de um código fonte criado, o programa seja compilado pelo compilador *freePascal* para os três principais sistemas operacionais usados em microinformática: Windows, Linux e MacOS. Cabe ressaltar que a IDE Lazarus é gratuita.

Para implementações futuras no software SISTRADA, faço as seguintes sugestões:

- Seleção de trecho específico de dados para geração de gráfico.
- Conclusão da programação das linhas de tendência logarítmica, polinomial e de potência.
- Cálculo e exibição do R quadrado para as linhas de tendência.
- Geração de gráfico com diversas linhas representando dados provenientes de portas diferentes do Arduino (sensores diferentes ou diversos sensores do mesmo tipo). Em outras palavras, para tratamentos de dados dependentes, onde o fator de proporcionalidade representará uma grandeza física de interesse no experimento.

6 REFERÊNCIAS

ARDUINO Homepage. Disponivel em: <http://www.arduino.cc/>. Consultado em: 10 de Outubro de 2013

AZEVEDO, A. **Análise e Dimensionamento de um sistema de medida baseado numa célula de carga**, Porto,1992.

BOLTON, W. **Mecatrônica uma abordagem multidisciplinar**, 4ª Edição, São Paulo, Bookman, 2008

CAMPILHO, A. **Instrumentação Electónica. Métodos e Técnicas de Medição**, 1ª Edição Ed. Porto, Porto, 2000

CASTILHO, M. **Modelagem de Software - A Importância da Modelagem no Planejamento de Desenvolvimento de Software.** Campinas, Dimensão Tech, 2013.

CIRCUITAR Homepage. Disponível em < https://www.circuitar.com.br/nanoshields/modulos/thermocouple/>. Consultado em: 10 de outubro de 2013

Eletronicos.com.br. Disponível em: <http://eletronicos.etc.br/como-sao-feitos-os-transdutores-resistivos/>. Consultado em 10 de novembro de 2013

FIGLIOLA, R.S.; BEASLEY, D.E. **Teoria e projeto para medições mecânicas**, 4ª Edição, Rio de Janeiro: LTC, 2007

FREEDOM DEFINED - Definição de open source hardware (OSHW) 1.0. Disponível em: <http://freedomdefined.org/OSHW/translations/portuguese>. Consultado em 10 de janeiro de 2015.

GARCIA, C. **Modelagem e Simulação de Processos Industriais e de Sistemas Eletromecânicos,** Vol. 1, São Paulo, EdUSP, 2005.

MELLIS, D. **O hardware em código aberto.** Entrevista para revista Info Exame, março, 2009. Disponível em: <http://info.abril.com.br/professional/tendencias/hardware-livre-leve-e-solto.shtml>. Consultado em 15 de janeiro de 2015.

MKS Technology for Productivity. Disponível em <http://www.mksinst.com/docs/UR/pin626a.aspx>. Consultado em 8 de setembro de 2014.

NATIONAL INSTRUMENTS. Disponível em: <http://www.ni.com/white-paper/8534/pt/>. Consultado em 10 de novembro de 2013.

OHS – Open hardware summit. Disponível em: <http://www.openhardwaresummit.org/>. Consultado em 10 de janeiro 2015.

RUBIO, M. G. Apostila: **Curso de Introdução à Instrumentação em Engenharia,** Instituto de Pesquisas Tecnológicas do Estado de São Paulo – IPT, São Paulo, 2000.

SHIELDLIST.ORG, **Lista de fabricantes de Shields.** Disponível em < http://shieldlist.org/>. Consultado em 10 de dezembro de 2015

TORREIRA, R.P. **Instrumentos de Medição Elétrica**, HEMUS, Curitiba, 2002

VUOLO, J.H. **Fundamentos da Teoria de Erros**, Edgard Blucher Ltda, São Paulo, 1992

YIN, R.K. **Estudo de caso: planejamento e métodos**, Bookman, Porto Alegre, 2005

ANEXO A - CRITÉRIOS PARA DISTRIBUIÇÃO SOB O CONCEITO OSHW

Critérios para distribuição sob o conceito OSHW (Open Source Hardware)

1. Documentação - o hardware deve ser distribuído com documentação, incluindo arquivos de design, e deve permitir a modificação e a distribuição destes arquivos. Quando a documentação não acompanhar o produto físico deve haver um meio devidamente publicado de como se obter esta documentação por não mais do que um custo razoável de reprodução, preferencialmente através do download na internet sem cobrança alguma. A documentação deve incluir arquivos de design em formato preferencial para alteração, por exemplo, o formato nativo de um programa CAD. Formatos de arquivo deliberadamente ilegíveis não são permitidos. Formatos intermediários análogos a códigos de programação compilados -- como o formato de impressão de um arquivo CAD -- não são substitutos válidos. A licença deve requerer que os arquivos de design sejam fornecidos em formatos totalmente documentados e de código aberto.

2. Escopo - a documentação do hardware deve especificar claramente quais partes do design, se não forem todas, são distribuídas sob a licença.

3. Software necessário - caso o design licenciado necessitar de software para operar corretamente e preencher suas funções essenciais, então a licença deve requisitar que uma das seguintes condições seja cumprida:

a) Que as interfaces sejam suficientemente documentadas de modo descomplicado para a criação de software open source que permita ao dispositivo operar corretamente e preencher suas funções essenciais. Por exemplo, a inclusão de diagramas detalhados com os tempos de sinais ou pseudocódigo que ilustre claramente a interface em operação.

b) Que o software necessário seja distrubuído sob uma licensa open source do tipo OSI.

4. Produtos Derivados - a licença deverá permitir modificações e produtos derivados, e permitir que sejam distribuídos sob os mesmos termos da licença do produto original. A licença deverá permitir a manufatura, venda, distribuiçãoe uso de produtos criados a partir dos arquivos de design, os próprios arquivos, e derivados.

5. Livre redistribuição - a licença não deverá restringir a nenhuma das partes a venda ou a livre distribuição da documentação do projeto. A licença não requisitará a cobrança de direitos autorais nem de nenhuma outra taxa para esta venda. A licença não requisitará a cobrança de direitos autorais nem de nenhuma outra taxa para a venda de produtos derivados.

6. Atribuição - a licença poderá requisitar que documentos derivados e notas de direitos autorais associados aos dispositivos, disponibilizem uma atribuição ao licenciador na distribuição de arquivos de design, produtos manufaturados, e derivados. A licença pode requisitar que esta informação seja acessível ao usuário final, mas não deverá especificar um formato de visualização. A licença pode requisitar que produtos derivados tenham um nome ou número de versão diferente do design original.

7. Não discriminação de pessoas ou grupos - a licença não deve fazer nenhuma discriminação contra nenhuma pessoa ou grupo de pessoas.

8. Não discriminação de campos de utilização - a licença não deve restringir o uso do produto (incluindo manufaturas) em nenhum campo de utilização. Por exemplo, não deve restringir o

uso em negócios ou em pesquisa nuclear.

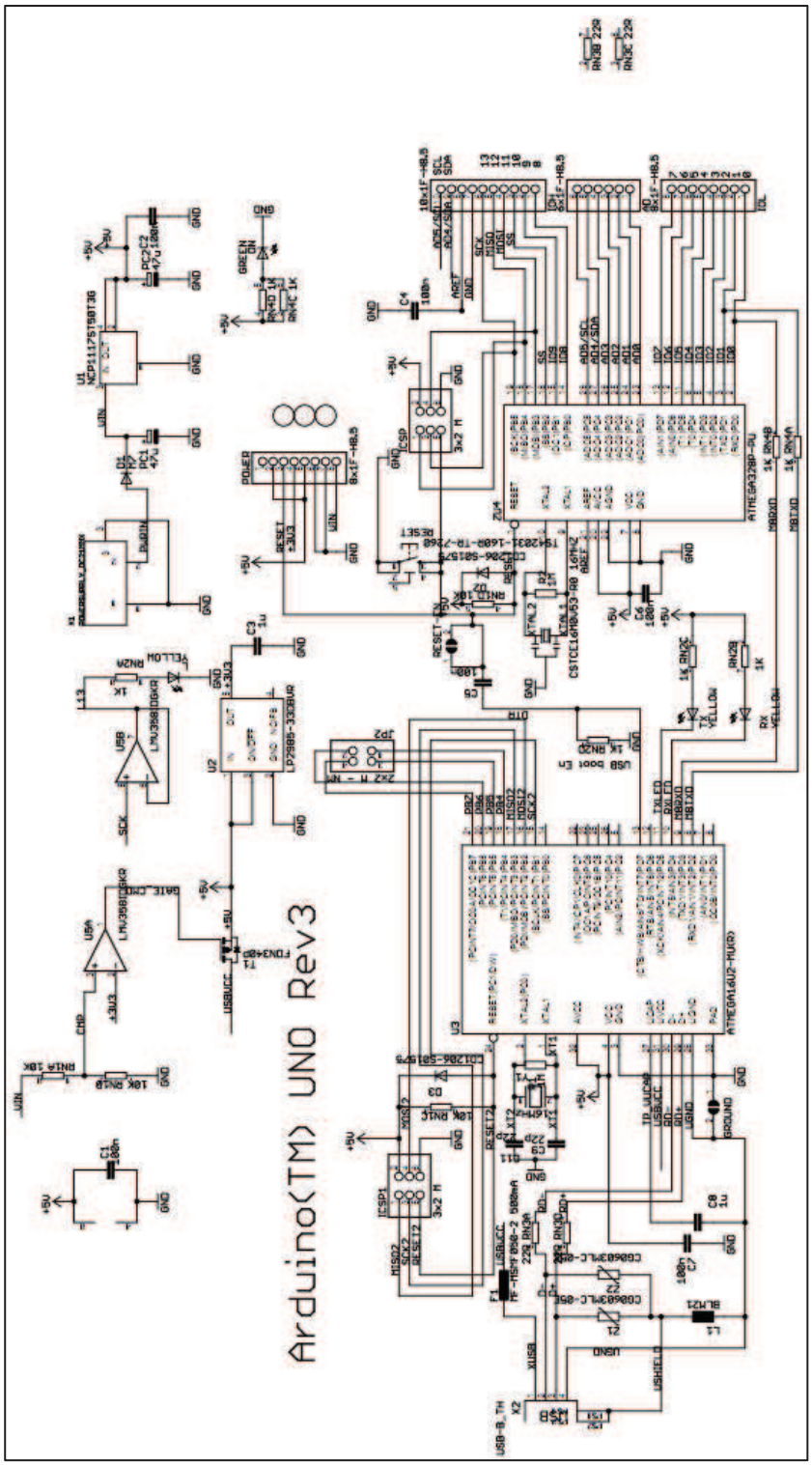
9. Distribuição da licença - os direitos garantidos pela licença devem ser aplicados a todos que tiverem produtos redistribuídos sem a necessidade de execução de nenhuma licença adicional.

10. A Licença não deve ser específica de um produto - os direitos garantidos pela licença não devem depender de que produtos licenciados sejam partes de um produto particular. Se uma parte for extraída de um produto, dentro dos termos da licença, todas as partes a quem este produto seja redistribuído devem ter os mesmos direitos que foram garantidos ao produto original.

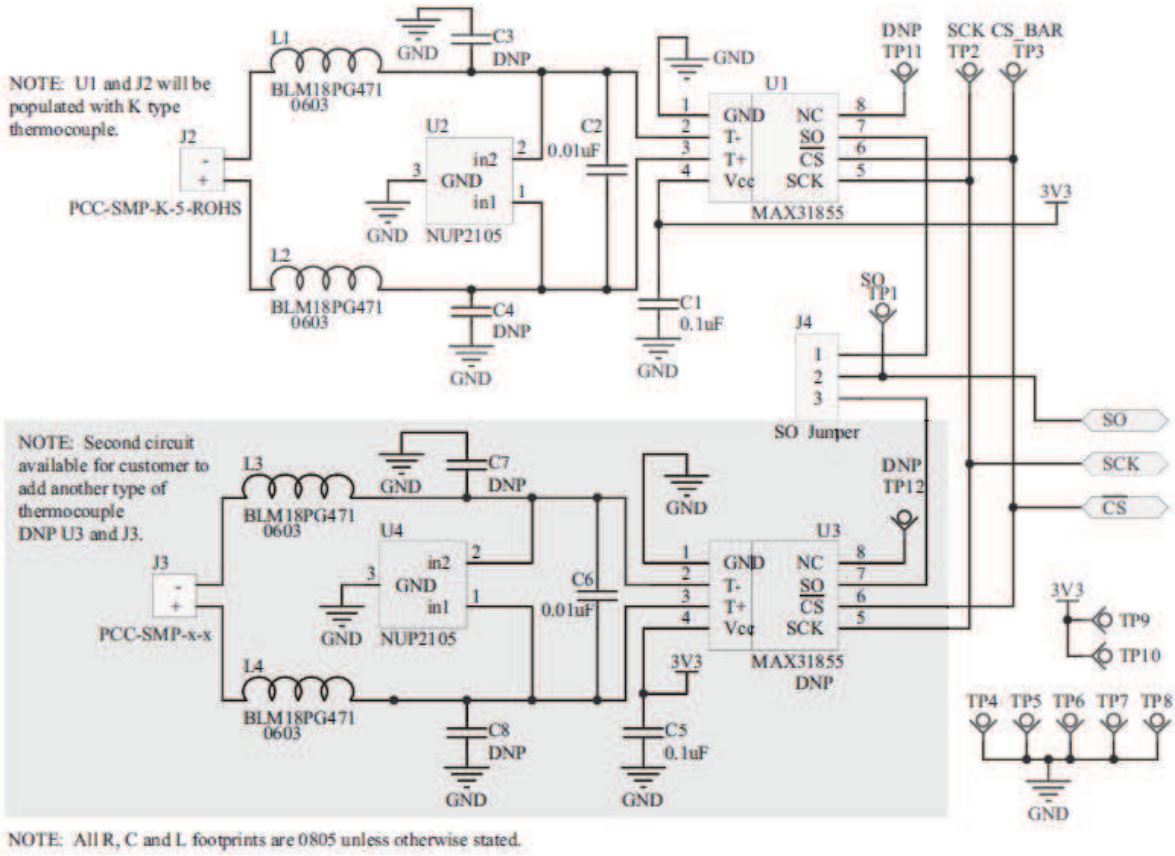
11. A Licença não deve restringir outros softwares ou hardwares - a licença não deve fazer restrições a que outros itens que sejam agregados ao produto licenciado mas não derivados dele. Por exemplo, a licença não deve insistir que todos os outros hardwares vendidos com o item licenciado sejam open source, nem que apenas software open source seja utilizado com o produto.

12. A Licença deve ser tecnologicamente neutra - nenhuma parte da licença pode ser aplicada a uma tecnologia particular, componente, material ou estilo de interface.

ANEXO B - ESQUEMA ELÉTRICO DO ARDUINO UNO



ANEXO C - REPRESENTAÇÃO ESQUEMÁTICA DO MAX31855



ANEXO D – APOSTILA DA IDE ARDUINO

Programação para Arduino - Primeiros Passos

Conceitos iniciais de programação para Arduino

Autor: Luís Fernando Chavier

Neste tutorial vamos apresentar os conceitos básicos de programação necessários para começar a utilizar o Arduino, e também outros tipos de sistemas embarcados semelhantes. Para conhecer o que é possível construir com esses sistemas, veja a nossa seção de projetos. Se quiser aprender mais sobre Arduino e sistemas embarcados, explore a nossa seção de tutoriais.

Aqui nós vamos explicar os conceitos de programação desde o início, e você não precisa saber nada sobre Arduino ou programação de computadores para começar. Se você já tem experiência prévia com programação, este tutorial talvez não acrescente muito ao seu conhecimento.

Nós vamos aprender como funciona um programa simples, fazendo nele algumas modificações ao longo do tutorial. Se você tiver acesso a um Arduino, você pode usá-lo ao longo do tutorial para praticar os conceitos aprendidos, tornando a experiência muito mais legal. Você só precisa de um Arduino, original ou compatível, e mais nada. Então vamos lá.

Introdução

O objetivo deste tutorial é apresentar, de uma forma simples e rápida, o básico de programação para que você possa começar a utilizar o Arduino em seus projetos, sem ter que ler muitos livros ou artigos sobre programação. O tema "desenvolvimento de software" como um todo é muito abrangente, então vamos focar apenas nos conceitos que são importantes para Arduino e sistemas embarcados em geral.

Existem muitas outras coisas para se aprender na parte de software que não vamos abordar aqui. No final do artigo nós colocamos links que você pode seguir para aprender conceitos mais avançados ou conceitos de software que não são muito utilizados na programação de sistemas embarcados.

Vamos começar explicando como funciona um computador (lembre-se que o Arduino é, no fundo, um computador).

Computador

Um **computador** é, de forma simplificada, uma máquina que processa instruções. Essas instruções são processadas no "cérebro" do computador, que se chama **microprocessador**. Todo computador possui pelo menos um microprocessador. O Arduino, por exemplo, nada mais é do que um computador muito pequeno, e ele utiliza um microprocessador do modelo **ATmega**. Alguns microprocessadores, como o ATmega, também são chamados de **microcontroladores**.

Programa de Computador

Um **programa de computador**, ou **software**, é uma sequência de instruções que são enviadas para o computador. Cada tipo de microprocessador (cérebro) entende um **conjunto de instruções** diferente, ou seja, o seu próprio "idioma". Também chamamos esse idioma de **linguagem de máquina**.

As linguagens de máquina são, no fundo, as únicas linguagens que os computadores conseguem entender, só que elas são muito difíceis para os seres humanos entenderem. É por isso nós usamos uma coisa chamada **linguagem de programação**.

No caso de sistemas como o Arduino (os chamados sistemas embarcados), o software que roda no microprocessador é também chamado de **firmware**.

Linguagem de Programação

Nós seres humanos precisamos converter as nossas idéias para uma forma que os computadores consigam processar, ou seja, a linguagem de máquina. Os computadores de hoje (ainda) não conseguem entender a linguagem natural que nós usamos no dia a dia, então precisamos de um outro "idioma" especial para instruir o computador a fazer as tarefas que desejamos. Esse "idioma" é uma linguagem de programação, e na verdade existem muitas delas.

Essas linguagens de programação também são chamadas de **linguagens de programação de alto nível**. A linguagem de programação utilizada no Arduino é a linguagem **C++** (com pequenas modificações), que é uma linguagem muito tradicional e conhecida. Essa é a linguagem que utilizaremos ao longo deste tutorial.

Para converter um programa escrito em uma linguagem de alto nível para linguagem de máquina, nós utilizamos uma coisa chamada **compilador**. A ação de converter um programa para linguagem de máquina é chamada **compilar**. Para compilar um programa, normalmente se utiliza um **ambiente de desenvolvimento** (ou IDE, do inglês ***Integrated Development Environment***), que é um aplicativo de computador que possui um compilador integrado, onde você pode escrever o seu programa e compilá-lo. No caso do Arduino, esse ambiente de desenvolvimento é o Arduino IDE.

O texto contendo o programa em uma linguagem de programação de alto nível também é conhecido como o **código fonte** do programa.

Algoritmo (Programa)

Um **algoritmo**, ou simplesmente **programa**, é uma forma de dizer para um computador o que ele deve fazer, de uma forma que nós humanos conseguimos entender facilmente. Os algoritmos normalmente são escritos em linguagens de programação de alto nível. Isso se aplica a praticamente qualquer computador, inclusive o Arduino, onde um algoritmo também é conhecido como **sketch**. Para simplificar, a partir de agora nós vamos nos referir aos algoritmos, programas ou sketches simplesmente como "programas".

Um programa é composto de uma sequência de comandos, normalmente escritos em um arquivo de texto. Para este tutorial, vamos usar como base os comandos do programa mais simples do Arduino, o **Blink**, que simplesmente acende e apaga um LED, e vamos destrinchá-lo ao longo do tutorial. Veja abaixo o código fonte do Blink:

```
int led = 13;
void setup() {
  pinMode(led, OUTPUT);
}
void loop() {
  digitalWrite(led, HIGH);
```

```
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

Variável

Uma **variável** é um recurso utilizado para armazenar dados em um programa de computador. Todo computador possui algum tipo de **memória**, e uma variável representa uma região da memória usada para armazenar uma determinada informação. Essa informação pode ser, por exemplo, um número, um caractere ou uma sequência de texto. Para podermos usar uma variável em um programa Arduino, nós precisamos fazer uma **declaração de variável**, como por exemplo:

```
int led;
```

Nesse caso estamos declarando uma variável do tipo `int` chamada `led`. Em seguida nós falaremos mais sobre o tipo de dado de uma variável.

Tipo de Dado

O **tipo de dado** de uma variável significa, como o próprio nome diz, o tipo de informação que se pode armazenar naquela variável. Em muitas linguagens de programação, como C++, é obrigatório definir o tipo de dado no momento da declaração da variável, como vimos na declaração da variável `led` acima. No caso dos módulos Arduino que usam processador ATmega, os tipos mais comuns de dados que utilizamos são:

- `boolean`: valor verdadeiro (`true`) ou falso (`false`)
- `char`: um caractere
- `byte`: um byte, ou sequência de 8 bits
- `int`: número inteiro de 16 bits com sinal (-32768 a 32767)
- `unsigned int`: número inteiro de 16 bits sem sinal (0 a 65535)
- `long`: número inteiro de 16 bits com sinal (-2147483648 a 2147483647)
- `unsigned long`: número inteiro de 16 bits sem sinal (0 a 4294967295)
- `float`: número real de precisão simples (ponto flutuante)
- `double`: número real de precisão dupla (ponto flutuante)
- `string`: sequência de caracteres
- `void`: tipo vazio (não tem tipo)

Para conhecer todos os tipos de dado suportados pelo Arduino, veja a seção "Data Types" nessa página.

Atribuição

Atribuir um valor a uma variável significa armazenar o valor nela para usar posteriormente. O comando de atribuição em C++ é o `=`. Para atribuírmos o valor `13` à variável `led` que criamos acima, fazemos assim:

```
led = 13;
```

Quando se armazena um valor em uma variável logo na sua inicialização, chamamos isso de **inicialização de variável**. Assim, no nosso programa de exemplo temos:

```
int led = 13;
```

O objetivo dessa linha de código é dizer que o pino 13 do Arduino será utilizado para acender o LED, e armazenar essa informação para usar depois ao longo do programa.

Os valores fixos usados no programa, como o valor `13` acima, são chamados de **constantes**, pois, diferentemente das variáveis, o seu valor não muda.

Operador

Um **operador** é um conjunto de um ou mais caracteres que serve para operar sobre uma ou mais variáveis ou constantes. Um exemplo muito simples de operador é o operador de adição, o `+`. Digamos que queremos somar dois números e atribuir a uma variável `x`. Para isso, fazemos o seguinte:

```
x = 2 + 3;
```

Após executar o comando acima, a variável `x` irá conter o valor `5`.

Cada linguagem de programação possui um conjunto de operadores diferente. Alguns dos operadores mais comuns na linguagem C++ são:

- Operadores aritméticos:
  - `+`: adição ("mais")
  - `-`: subtração ("menos")
  - `*`: multiplicação ("vezes")
  - `/`: divisão ("dividido por")
- Operadores lógicos:
  - `&&`: conjunção ("e")
  - `||`: disjunção ("ou")
  - `==`: igualdade ("igual a")
  - `!=`: desigualdade ("diferente de")
  - `!`: negação ("não")
  - `>`: "maior que"
  - `<`: "menor que"
  - `>=`: "maior ou igual a"
  - `<=`: "menor ou igual a"
- Operadores de atribuição:
  - `=`: atribui um valor a uma variável, como vimos acima.

Ao longo do desenvolvimento dos seus projetos, aos poucos você se familiarizará com todos esses operadores. Para uma lista completa, veja essa página da Wikipedia.

Função

Uma **função** é, em linhas gerais, uma sequência de comandos que pode ser reutilizada várias vezes ao longo de um programa. Para criar uma função e dizer o que ela faz, nós precisamos fazer uma **declaração de função**. Veja como uma função é declarada no nosso programa de exemplo:

```
void setup() {
  pinMode(led, OUTPUT);
}
```

Aqui estamos declarando uma função com o nome `setup()`. O que ela faz é executar os comandos de uma outra função `pinMode()`. A ação de executar os comandos de função previamente declarada é denominada **chamada de função**. Nós não precisamos declarar a função `pinMode()` porque ela já é declarada automaticamente no caso do Arduino.

Chamada de Função

Chamar uma função significa executar os comandos que foram definidos na sua declaração. Uma vez declarada, uma função pode ser chamada várias vezes no mesmo programa para que seus comandos sejam executados novamente. Para chamarmos a nossa função `setup()`, por exemplo, nós usaríamos o seguinte comando:

```
setup();
```

No entanto, no caso do Arduino, nós não precisamos chamar a função `setup()`, porque ela é chamada automaticamente. Quando compilamos um programa no Arduino IDE, ele chama a função `setup()` uma vez e depois chama a função `loop()` repetidamente até que o Arduino seja desligado ou reiniciado.

Valor de Retorno

A palavra chave que vem antes do nome da função na declaração define o tipo do **valor de retorno** da função. Toda vez que uma função é chamada, ela é executada e devolve ou **retorna** um determinado valor - esse é o valor de retorno, ou simplesmente **retorno** da função. O valor de retorno precisa ter um tipo, que pode ser qualquer um dos tipos de dados citados anteriormente. No caso da nossa função `setup()`, o tipo de retorno é `void`, o que significa que a função não retorna nada.

Para exemplificar, vamos criar uma função que retorna alguma coisa, por exemplo um número inteiro. Para retornar um valor, nós utilizamos o comando `return`:

```
int f() {
  return 1;
}
```

Quando chamada, a função `f()` acima retorna sempre o valor `1`. Você pode usar o valor de retorno de uma função para atribuí-lo a uma variável. Por exemplo:

```
x = f();
```

Após declarar a função `f()` e chamar o comando de atribuição acima, a variável `x` irá conter o valor `1`.

Parâmetros

Um outro recurso importante de uma função são os **parâmetros**. Eles servem para enviar algum dado para a função quando ela é chamada. Vamos criar por exemplo uma função que soma dois números:

```
int soma(int a, int b) {
  return a + b;
}
```

Aqui acabamos definir uma função chamada `soma()`, que aceita dois números inteiros como parâmetros. Nós precisamos dar um nome para esses parâmetros, e nesse caso escolhemos `a` e `b`. Esses parâmetros funcionam como variável que você pode usar dentro da função. Sempre que chamarmos a função `soma()`, precisamos fornecer esses dois números. O comando `return a + b;` simplesmente retorna a função com a soma dos dois números. Vamos então somar 2 + 3 e atribuir o resultado para uma variável `x`:

```
x = soma(2, 3);
```

Após a chamada acima, a variável `x` irá conter o valor `5`.

Comentários

Um **comentário** é um trecho de texto no seu programa que serve apenas para explicar (documentar) o código, sem executar nenhum tipo de comando no programa. Muitas vezes, os comentários são usados também para desabilitar comandos no código. Nesse caso, dizemos que o código foi **comentado**.

Na linguagem C++, um comentário pode ser escrito de duas formas:

- Comentário de linha: inicia-se com os caracteres `//`, tornando todo o resto da linha atual um comentário.
- Comentário de bloco: inicia-se com os caracteres `/*` e termina com os caracteres `*/`. Todo o texto entre o início e o término se torna um comentário, podendo ser composto de várias linhas.

Para facilitar a visualização, os ambientes de desenvolvimento geralmente mostram os comentários em uma cor diferente. No caso do Arduino IDE, por exemplo, os comentários são exibidos na cor cinza. Vamos então explicar o que o programa de exemplo faz, inserindo nele vários comentários explicativos:

```
/*
Programação para Arduino - Primeiros Passos
Programa de exemplo: Blink
*/

/*
Declaração da variável "led"
Indica que o LED está conectado no pino digital 13 do Arduino (D13).
*/
```

```
int led = 13;

/*
Declaração da função setup()
Esta função é chamada apenas uma vez, quando o Arduino é ligado ou reinicia
do.
*/
void setup() {
  // Chama a função pinMode() que configura um pino como entrada ou saída
  pinMode(led, OUTPUT); // Configura o pino do LED como saída
}

/*
Declaração da função loop()
Após a função setup() ser chamada, a função loop() é chamada repetidamente
até
o Arduino ser desligado.
*/
void loop() {
  // Todas as linhas a seguir são chamadas de função com passagem de
parâmetros
  // As funções são executadas em sequência para fazer o LED acender e
apagar
  digitalWrite(led, HIGH); // Atribui nível lógico alto ao pino do LED,
acendendo-o
  delay(1000);             // Espera 1000 milissegundos (um segundo)
  digitalWrite(led, LOW);  // Atribui nível lógico baixo ao pino do LED,
apagando-o
  delay(1000);             // Espera 1000 milissegundos (um segundo)

  // Após terminar a função loop(), ela é executada novamente repetidas
vezes,
  // e assim o LED continua piscando.
}
```

Estruturas de Controle

Estruturas de controle são blocos de instruções que alteram o fluxo de execução do código de um programa. Com elas é possível fazer coisas como executar comandos diferentes de acordo com uma condição ou repetir uma série de comandos várias vezes, por exemplo.

A seguir nós veremos algumas das estruturas de controle mais comuns usadas nas linguagens de programação em geral. Vamos também modificar o nosso programa de teste para exemplificar melhor como essas estruturas funcionam.

While

O **while** é uma estrutura que executa um conjunto de comandos repetidas vezes enquanto uma determinada condição for verdadeira. ***While*** em inglês quer dizer "enquanto", e pronuncia-se "uái-ou". Ele segue o seguinte formato:

```
while(condição) {
    ...
}
```

Vamos então fazer uma modificação no nosso programa para exemplificar melhor como o `while` funciona. O nosso objetivo agora é fazer o LED piscar três vezes, depois esperar cinco segundos, piscar mais três vezes e assim por diante. Nós vamos mudar o conteúdo da função `loop()` para o seguinte:

```
  // Variável para contar o número de vezes que o LED piscou
  int i = 0;

  // Pisca o LED três vezes
  while(i < 3) {
    digitalWrite(led, HIGH); // Atribui nível lógico alto ao pino do LED, acendendo-o
    delay(1000);             // Espera 1000 milissegundos (um segundo)
    digitalWrite(led, LOW);  // Atribui nível lógico baixo ao pino do LED, apagando-o
    delay(1000);             // Espera 1000 milissegundos (um segundo)
    i = i + 1;               // Aumenta o número de vezes que o LED piscou
  }

  delay(5000);               // Espera 5 segundos para piscar o LED de novo
```

Primeiro nós declaramos uma variável `i`. Essa variável vai contar quantas vezes o LED já piscou desde o início do programa ou desde a última pausa de cinco segundos. Nós vamos inicializar essa variável com zero porque no início da função `loop()` o LED ainda não piscou nenhuma vez sob essas condições.

Em seguida nós inserimos o comando `while`, que deve ser seguido de uma **condição** definida entre parênteses. Enquanto essa condição for verdadeira, todo o bloco de comandos entre os caracteres `{` e `}` é executado repetidamente. No caso do nosso programa, enquanto o número de "piscadas" do LED (representado pela variável `i`) for menor do que três, nós continuamos a executar os comandos que fazem o LED piscar. Isso é representado pela expressão `i < 3` dentro dos parênteses.

Entre os caracteres `{` e `}` nós colocamos o código que faz o LED piscar, como anteriormente, mas não podemos nos esquecer de somar `1` à variável que conta o número de "piscadas". Isso é feito na seguinte linha de código:

```
    i = i + 1;                  // Aumenta o número de vezes que o LED piscou
```

Veja que após executar todos os comandos entre `{` e `}`, sempre teremos na variável `i` o número de vezes que o LED piscou desde o início da função `loop()`. Vamos percorrer a sequência de passos executada cada vez que a função `loop()` é chamada:

1. Atribuímos `0` à variável `i`: o LED ainda não piscou nenhuma vez.
2. Comparamos se `i < 3`: como `0` é menor do que `3`, executamos os comandos entre `{` e `}`:
    1. Executamos os comandos para acender e apagar o LED.
    2. Somamos `1` à variável `i`, tornando-a `1`: sabemos que o LED piscou uma vez.
3. Voltamos ao início do `while` e comparamos se `i < 3`: como `1` é menor do que `3`, executamos os comandos entre `{` e `}` novamente:
    1. Executamos os comandos para acender e apagar o LED.
    2. Somamos `1` à variável `i`, tornando-a `2`: sabemos que o LED piscou duas vezes.
4. Voltamos ao início do `while` e comparamos se `i < 3`: como `2` é menor do que `3`, executamos os comandos entre `{` e `}` novamente:
    1. Executamos os comandos para acender e apagar o LED.
    2. Somamos `1` à variável `i`, tornando-a `3`: sabemos que o LED piscou três vezes.
5. Voltamos ao início do `while` e comparamos se `i < 3`: como `3` **não** é menor do que `3`, não executamos mais os comandos entre `{` e `}` e prosseguimos à próxima instrução.
6. Esperamos cinco segundos por meio da chamada `delay(5000)`.

Após esses passos, chegamos ao final da função `loop()`, e como já sabemos, ela é chamada novamente pelo sistema do Arduino. Isso reinicia o ciclo, executando os passos acima indefinidamente.

Rode o programa modificado com as instruções acima no seu Arduino e tente variar o número de "piscadas" e o número

For

Agora que nós já aprendemos o comando while, fica muito fácil aprender o comando **for**, pois ele é quase a mesma coisa. Vamos modificar o conteúdo da função `loop()` como fizemos acima, porém usando o `for` no lugar do `while`:

```
  // Variável para contar o número de vezes que o LED piscou
  int i;

  // Pisca o LED três vezes
  for(i = 0; i < 3; i++) {
    digitalWrite(led, HIGH); // Atribui nível lógico alto ao pino do LED,
acendendo-o
    delay(1000);             // Espera 1000 milissegundos (um segundo)
```

```
    digitalWrite(led, LOW);  // Atribui nível lógico baixo ao pino do LED,
apagando-o
    delay(1000);             // Espera 1000 milissegundos (um segundo)
  }


  delay(5000);               // Espera 5 segundos para piscar o LED de novo
```

A primeira modificação que fizemos foi declarar a variável `i` sem inicializá-la com o valor `0`. Nós podemos fazer isso porque o comando `for` fará isso para a gente. Ele segue o seguinte formato:

```
for(inicialização; condição; finalização) {
    ...
}
```

Vamos descrever cada item separadamente:

- **Condição**: é uma expressão verificada repetidamente, de forma idêntica à condição entre parênteses do `while`. Enquanto ela for verdadeira, os comandos entre `{` e `}` continuam sendo executados.
- **Inicialização**: é um comando executado **apenas uma vez** no início do comando `for`.
- **Finalização**: é um comando executado **repetidas vezes** ao final de cada execução dos comandos entre `{` e `}`.

Podemos então verificar que o `for` nada mais é do que um `while` acrescido de um comando de inicialização e um comando de finalização. Para o nosso programa de teste, esses comandos são, respectivamente:

- `i = 0`: inicializa a contagem do número de "piscadas".
- `i++`: soma `1` à variável `i` ao final da execução dos comandos entre `{` e `}`; nesse caso ele é equivalente ao comando `i = i + 1`. O operador `++` é chamado de operador de **incremento**, e é muito usado na linguagem C++.

Se executarmos o programa acima no Arduino, veremos que o resultado é o mesmo que obtivemos com o programa que fizemos anteriormente utilizando o `while`.

If

O **if** é uma das estruturas mais básicas de programação em geral. ***If*** significa "se" em inglês, e é exatamente isso que ele faz: ele verifica uma expressão e, apenas **se** ela for **verdadeira**, executa um conjunto de comandos. Em linguagem natural, ele executa uma lógica do tipo: "**se** isso for verdadeiro, então faça aquilo"

Para ilustrar, vamos modificar o nosso programa de exemplo para que ele faça a mesma coisa que fizemos com owhile e o for acima, porém vamos fazer isso usando um `if`, que segue o seguinte formato:

```
if(condição) {
    ...
}
```

A lógica é muito simples: sempre que a condição for verdadeira, os comandos entre `{` e `}` são executados, caso contrário o programa prossegue sem executá-los. Vamos ver então como fica a função `loop()`:

```
// Variável para contar o número de vezes que o LED piscou
int i = 0;

void loop() {
  digitalWrite(led, HIGH); // Atribui nível lógico alto ao pino do LED,
acendendo-o
  delay(1000);             // Espera 1000 milissegundos (um segundo)
  digitalWrite(led, LOW);  // Atribui nível lógico baixo ao pino do LED,
apagando-o
  delay(1000);             // Espera 1000 milissegundos (um segundo)

  i++;                     // Incrementa o número de "piscadas"
  if(i == 3) {
    delay(5000);           // Espera 5 segundos para piscar o LED de novo
    i = 0;                 // Reinicia o contador de número de "piscadas"
  }
}
```

Aqui a lógica é um pouco diferente: nós vamos manter a função `loop()` piscando o LED como no programa original, porém vamos inserir uma espera adicional de 5 segundos após cada 3 piscadas. Para isso, criamos uma variável `i` fora da função `loop()`; ela precisa ser declarada de fora da função para poder reter o seu valor entre cada execução da função `loop()`. Chamamos isso de **variável global**. Quando a variável é declarada dentro do corpo da função, ela não retém o valor etnre cada execução, sendo reiniciada a cada vez que a função é re-executada. Chamamos isso de **variável local**.

Nós usaremos então essa variável global `i` para contar, novamente, o número de vezes que o LED acendeu e apagou. Na declaração da variável, nós a inicializamos com o valor `0` para indicar que o LED não acendeu nenhuma vez ainda. A função `loop()` então começa a ser executada, acendendo e apagando o LED. Para contar o número de vezes que o LED piscou, nós adicionamos a seguinte linha de código:

```
  i++;                     // Incrementa o número de "piscadas"
```

Em seguida utilizamos o `if` para verificar se acabamos de acender o LED pela terceira vez. Para isso, usamos a expressão `i == 3` na condição do `if`. Se essa expressão for verdadeira, isso que dizer que o LED já acendeu 3 vezes, então inserimos uma pausa adicional de 5 segundos com a chamada `delay(5000)` e reiniciamos a contagem do número de "piscadas" novamente com o seguinte comando:

```
    i = 0;                 // Reinicia o contador de número de "piscadas"
```

A partir daí a função `loop()` continua sendo chamada e o ciclo se inicia novamente.

If-Else

O **if-else**, também conhecido como **if-then-else**, pode ser visto como uma extensão do comando `if`. ***Else*** em inglês significa "caso contrário", e ele faz exatamente o que o nome diz: "**se** isso for verdadeiro, então faça aquilo,**caso contrário**, faça outra coisa". Ele segue o seguinte formato:

```
if(condição) {
   ...
} else {
   ...
}
```

Para exemplificar, vamos usar o programa do for que mostramos acima, mas vamos dessa vez fazer o LED acender e apagar quatro vezes antes de dar uma pausa de cinco segundos. Depois vamos fazer com que na terceira de cada uma dessas quatro "piscadas", o LED acenda por um período mais curto. Dentro da função `loop()`, teremos o seguinte:

```
  // Variável para contar o número de vezes que o LED piscou
  int i;

  // Pisca o LED três vezes
  for(i = 0; i < 3; i++) {
    if(i == 2) {
      digitalWrite(led, HIGH); // Atribui nível lógico alto ao pino do LED,
acendendo-o
      delay(200);              // Espera 200 milissegundos (um segundo)
      digitalWrite(led, LOW);  // Atribui nível lógico baixo ao pino do
LED, apagando-o
      delay(1800);             // Espera 1800 milissegundos (um segundo)
    } else {
      digitalWrite(led, HIGH); // Atribui nível lógico alto ao pino do LED,
acendendo-o
      delay(1000);             // Espera 1000 milissegundos (um segundo)
      digitalWrite(led, LOW);  // Atribui nível lógico baixo ao pino do
LED, apagando-o
      delay(1000);             // Espera 1000 milissegundos (um segundo)
    }
  }

  delay(5000);               // Espera 5 segundos para piscar o LED de novo
```

Aqui o que fazemos é, toda vez que vamos acender o LED, verificar se é a terceira vez que isso acontece, por meio do comando `if` com a condição `i == 2`. Se essa expressão for verdadeira, isso quer dizer que já acendemos o LED duas vezes e estamos prestes a acendê-lo pela terceira vez;

nesse caso mudamos o tempo que o LED fica acaso para um valor menor, de 0,2 segundo (uma redução de 0,8 segundo) e o tempo que ele fica apagado para um valor maior, de 1,8 segundos (aumento de 0,8 segundo).

Mas e se essa não for a terceira vez que o LED está sendo acionado? É aí que entra o `else`: se a condição do `if` for verdadeira, o bloco de comandos entre `{` e `}` logo após o `if` é executado, **caso contrário**, o bloco entre `{` e `}` após o `else` é executado. Isso quer dizer que para a primeira, segunda e quarta "piscadas" será usado o tempo padrão de um segundo.

Bibliotecas

As coisas que aprendemos nas seções anteriores são importantes para implementar a lógica do seu programa no Arduino, mas normalmente você vai querer fazer mais coisas além de apenas acender um LED. Quando se faz tarefas mais complexas ou se utiliza algum outro circuito conectado ao seu Arduino, um recurso muito importante são as **bibliotecas**.

Uma biblioteca é basicamente composta de código fonte adicional que você adiciona ao seu projeto por meio do comando **include**. Vejamos como adicionar, por exemplo, uma biblioteca para controle de um display de cristal liquido (LCD):

```
#include <LiquidCrystal.h>
```

Uma biblioteca do Arduino se apresenta normalmente como uma ou mais **classes** que possuem funções, os**métodos**, para acionar dispositivos, configurá-los ou executar alguma outra tarefa. Continuando com o exemplo do display de cristal líquido, para usá-lo no seu programa, primeiro é preciso inicializá-lo. O que fazemos nesse caso é criar um **objeto** para acessar o LCD (tecnicamente isso se chama **instanciar** um objeto). Vejamos como isso é feito:

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

Quando fazemos isso, `lcd` se torna um objeto da classe `LiquidCrystal`. Isso é o equivalente a criar uma variáveldo tipo `LiquidCrystal`. Os parâmetros que são passados entre parênteses servem para inicializar a configuração desse objeto, e nesse caso correspondem aos números dos pinos que foram utilizados para conectar o LCD ao Arduino.

Quase sempre as bibliotecas de Arduino possuem um método `begin()`, que serve para fazer a configuração inicial do dispositivo que está sendo controlado. Para chamar a função `begin()` do objeto `lcd` que criamos, fazemos o seguinte:

```
  lcd.begin(16, 2);
```

Normalmente esse método `begin()` é chamado de dentro da função `setup()`, ou seja, durante a inicialização do programa. Os parâmetros do método `begin()`, nesse caso, correspondem ao número de colunas e o número de linhas do LCD, respectivamente.

Feitos esses passos, já podemos escrever texto no LCD. Fazemos isso usando o método `print()` do objeto `lcd`, sempre que precisarmos ao longo do programa:

```
  lcd.print("Oi!");
```

O método `print()` é apenas um dos vários métodos disponíveis na blblioteca `LiquidCrystal`. Para saber todos os métodos fornecidos por uma determinada biblioteca, é preciso consultar a documentação fornecida com ela.

Este é o processo básico de utilização de bibliotecas no Arduino. Para mais informações, leia a documentação fornecida com a biblioteca que você está usando.

Classes, objetos e métodos são conceitos de **programação orientada a objetos**. Não vamos explicar tudo em detalhes aqui, mas se você quiser aprender mais sobre isso, siga os links relacionados no final da página.

ANEXO E - PREÇO DA LINGUAGEM DE PROGRAMAÇÃO LABVIEW

NATIONAL INSTRUMENTS

Technical Sales
Brasil
(11) 3149-3149
ni.brazil@ni.com

LabVIEW Base Development System for Windows

- Software totalmente integrado de projeto gráfico de sistemas
- Suporte a uma ampla variedade de hardware de medição, E/S e barramentos
- Interfaces de usuário personalizadas, orientadas a evento, para medição e controle
- Compilador avançado, que garante alto desempenho na execução e otimização de código
- Inclui SSP, que oferece suporte técnico profissional, treinamento on-line e upgrades de software

Visão geral

O LabVIEW é um ambiente de programação gráfica de uso consagrado na indústria, desenvolvido para engenheiros e cientistas que criam aplicações de teste, medição e controle. Com o LabVIEW, você pode adquirir sinais do mundo real, realizar análises para identificar dados significativos e transmitir ou armazenar resultados de diversas maneiras, com rapidez e facilidade.
Você pode adquirir o LabVIEW juntamente com software add-on para aplicações específicas a preços promocionais, adquirindo o Developer Suite. Se você tiver uma versão anterior do LabVIEW, poderá fazer o upgrade do produto pelo Advisor para upgrades de software.
Com o LabVIEW Base Development System, você tem direito ao programa padrão de serviços (SSP), que irá ajudá-lo a obter o máximo de seu investimento em software. Com sua participação no programa SSP, você tem acesso aos mais recentes aprimoramentos tecnológicos, através de atualizações automáticas de software e releases de manutenção. Além disso, você poderá também acelerar o desenvolvimento de suas aplicações, tendo acesso direto por telefone ou e-mail a suporte técnico fornecido por engenheiros de aplicações da NI. Você também terá acesso a módulos on-line especiais de treinamento em software, nos quais poderá conhecer melhor os recursos, aplicações e melhores práticas de desenvolvimento.

Preços

LabVIEW Base Development System for Windows - 776671-35 Qty

Serviço
1 year SSP
2 years SSP
3 years SSP

Add-ons de software relacionados

LabVIEW Application Builder for Windows - 776675-35 Qty

LabVIEW Report Generation Toolkit for Windows - 778406-35 Qty

Estimated Shipping Days: 12 - 17

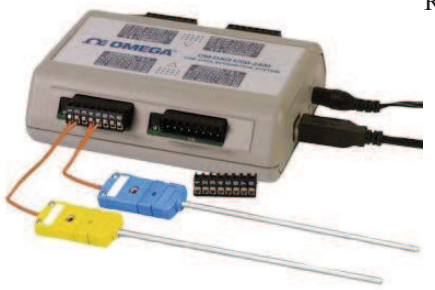
Package Price: $ 1.100

APENDICE A – CUSTO DOS EQUIPAMENTOS USADOS NESTE ESTUDO

Preços dos materiais utilizados na elaboração do arranjo para leitura de um termopar.

Material	**Qtd.**	**Preço**	**Fornecedor**
Placa Arduino Uno	01	R$ 68,00	http://www.webtronico.com/arduino-uno-r3.html
Protoboard 640 furos	01	R$ 14,00	http://www.equibancada.com.br/produto/protoboard-epb-0050-640-PINOS%2C-1-BARRA-DE-DISTRIBUI%C7%C3O-DE-100-PINOS%2C-SEM-BASE.html
Amplificador Operacional MAX31855	01	R$ 44,37	http://pt.aliexpress.com/store/product/200-to-1350C-MAX31855-Module-K-Type-Thermocouple-Thermocouple-temp-Sensor/503657_32229405772.html
TOTAL		**R$ 126,37**	

Dispositivos de Aquisição de Dados comerciais para leitura de termopar.

Modelo	Imagem	Preço	Fornecedor
OM-DAQ-USB-2401		R$ 2.455,00	OMEGA: http://br.omega.com/pptst/OM-DAQ-USB-2400.html
NI 9211		R$ 2.201,00	National Instruments: http://sine.ni.com/nips/cds/view/p/lang/pt/nid/209887

APENDICE B - CÓDIGO FONTE DO SISTEMA SISTRADA.

```
unit unitSerial;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, TAGraph, TASources, TASeries, TAStyles,
  TALegendPanel, TANavigation, TAIntervalSources, TAChartListbox,
  TAChartExtentLink, TAChartImageList, TAFuncSeries, Forms, Controls, Graphics,
  Dialogs, StdCtrls, ComCtrls, ExtCtrls, ColorBox, ValEdit, Buttons, Grids,
  SdpoSerial, Math, types, TAChartUtils;

type

  { TForm1 }

  TForm1 = class(TForm)
    Btn_importar: TBitBtn;
    Btn_conecta: TButton;
    Btn_desconecta: TButton;
    btn_gerar: TButton;
    btn_ok: TButton;
    Chart1: TChart;
    Chart1LineSeries1: TLineSeries;
    Chart1LineSeries2: TLineSeries;
    Chart1LineSeries3: TLineSeries;
    ChartExtentLink1: TChartExtentLink;
    ChartImageList1: TChartImageList;
    ChartListbox1: TChartListbox;
    ChartNavPanel1: TChartNavPanel;
    ChartStyles1: TChartStyles;
    CheckBox_exibe: TCheckBox;
    ComboBox_baudrate: TComboBox;
    ComboBox_databits: TComboBox;
    ComboBox_flowcontrol: TComboBox;
    ComboBox_parite: TComboBox;
    ComboBox_port: TComboBox;
    ComboBox_sensor: TComboBox;
    ComboBox_stopbits: TComboBox;
    edt_inicial: TEdit;
    Edt_final: TEdit;
    edit_ordem_poli: TEdit;
    Image1: TImage;
    Image2: TImage;
    Image3: TImage;
    Image4: TImage;
```

```
Image5: TImage;
IntervalChartSource1: TIntervalChartSource;
Label1: TLabel;
Label10: TLabel;
Label11: TLabel;
Label12: TLabel;
Label13: TLabel;
Label14: TLabel;
Label15: TLabel;
Label16: TLabel;
Label17: TLabel;
Label_arquivo: TLabel;
label_funcaoE: TLabel;
varvalor: TLabel;
varMediaX: TLabel;
varB: TLabel;
lbl_preencher: TLabel;
lbl_inicial: TLabel;
lbl_final: TLabel;
Label_do_zoom: TLabel;
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
Label6: TLabel;
Label_zoom: TLabel;
label_funcaoL: TLabel;
Label7: TLabel;
Label8: TLabel;
Label9: TLabel;
ListBox1: TListBox;
ListChartSource3: TListChartSource;
Memo1: TMemo;
Memo_ajuda: TMemo;
oB1: TLabeledEdit;
oB0: TLabeledEdit;
oSXY: TLabeledEdit;
oSYY: TLabeledEdit;
oSXX: TLabeledEdit;
ListChartSource1: TListChartSource;
ListChartSource2: TListChartSource;
Memo_terminal: TMemo;
OpenDialog1: TOpenDialog;
PageControl1: TPageControl;
RadioGroup1: TRadioGroup;
SdpoSerial1: TSdpoSerial;
TabSheet1: TTabSheet;
TabSheet2: TTabSheet;
TabSheet3: TTabSheet;
TabSheet4: TTabSheet;
```

```
    UpDown1: TUpDown;
    procedure Btn_importarClick(Sender: TObject);
    procedure btn_gerarClick(Sender: TObject);
    procedure btn_okClick(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Btn_conectaClick(Sender: TObject);
    procedure Btn_desconectaClick(Sender: TObject);
    procedure Chart1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure Chart1DrawReticule(ASender: TChart; ASeriesIndex,
      AIndex: Integer; const AData: TDoublePoint);
    procedure Chart1MouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer
      );
    procedure Chart1MouseUp(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure Chart1StartDrag(Sender: TObject; var DragObject: TDragObject);
    procedure ChartListbox1Click(Sender: TObject);
    procedure CheckBox_exibeChange(Sender: TObject);
    procedure ComboBox_sensorChange(Sender: TObject);
    procedure edit_ordem_poliChange(Sender: TObject);
    procedure label_funcaoEClick(Sender: TObject);
    procedure label_funcaoLClick(Sender: TObject);
    procedure ListBox1Click(Sender: TObject);
    procedure RadioGroup1Click(Sender: TObject);
    procedure SdpoSerial1RxData(Sender: TObject);
    procedure TabSheet2ContextPopup(Sender: TObject; MousePos: TPoint;
      var Handled: Boolean);
    procedure UpDown1Click(Sender: TObject; Button: TUDBtnType);
  private
    { private declarations }
  public
    { public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.lfm}

{ TForm1 }

procedure TForm1.Button1Click(Sender: TObject);
begin
//  poSerial1.WriteData(Edit1.text);
end;

procedure TForm1.Btn_importarClick(Sender: TObject);
```

```
  var arquivo: textfile;
  sizeline,pos_i: integer;
  x,y,k,w: array[1..2000] of string;
  linha: String;
  cont, l:integer;
  nomearq: string;
  TB,numero, somaXLOGN, XLOGN, LOGN, somaLOGN,
mediaLOGN,A,B,NX,NY,oY,oX,XY,X2,Y2,somaX,somaX2,somaY,somaY2,somaXY,medi
aX,mediaY,SXX,SYY,SXY,B1,B0: real;
  YM: double;
begin
  if OpenDialog1.execute() then
  begin
    nomearq:=opendialog1.FileName;
  end;
ListChartSource1.Clear;
AssignFile(arquivo,nomearq);
Reset (arquivo);
pos_i:=0;
chart1.visible:=true;
chart1.BottomAxis.Title.caption:=edit2.text;
chart1.LeftAxis.Title.caption:=edit3.text;
chart1.Title.Text.Text:=edit1.text;
cont:=1;      // contador de elementos
//T1:=1;
l:=1;         // contador de elementos da matriz
NX:=0;        // o X do gráfico
NY:=0;        // o Y do gráfico
oY:=0;        // o Y da reta (quando for linear)
oX:=0;        // o X da reta (quando for linear)
X2:=0;        // X ^ 2
Y2:=0;        // Y ^ 2
XY:=0;        // X * Y
TB:=0;
somaX:=0;     // somatória de X
somaX2:=0;    // somatória de X ^ 2
somaY:=0;     // somatória de Y
somaY2:=0;    // somatória de Y ^ 2
somaXY:=0;    // somatória de X * Y
mediaX:=0;    // média de X
mediaY:=0;    // média de Y
mediaLOGN:=0;       // Logaritmo neperiano
somaLOGN:=0;
somaXLOGN:=0;
btn_gerar.visible:=true;
numero:=0;    // recebe o numero para calcular o logn
while not Eof (arquivo) do
     begin
     ReadLn (arquivo,linha);
     sizeline:=length(linha);
```

```
      pos_i:=0;
      x[l]:=copy(linha,pos_i,pos(';',linha)-1);
      NX:=strtofloat(copy(linha,pos_i,pos(';',linha)-1));
      pos_i:=pos(';',linha)+1;
      linha:=copy(linha,pos_i,(sizeline-pos_i+1));
      y[l]:=linha;
      NY:=strtofloat(linha);
      ListChartSource1.Add(strtofloat(x[l]),strtofloat(y[l]));
      numero:=strtofloat(y[l]);
      cont:=cont+1;
      X2:=NX*NX;
      Y2:=NY*NY;
      somaX:=somaX+NX;
      somaX2:=somaX2+ X2;
      somaY:=somaY+NY;
      somaY2:=somaY2+Y2;
      XY:=NX*NY;
      somaXY:=somaXY+XY;
      LOGN:=ln(numero);          // Logaritmo Neperiano de Numero
      somaLOGN:=somaLOGN+LOGN;
      XLOGN:=strtofloat(x[l])*LOGN;
      somaXLOGN:=somaXLOGN+XLOGN;
      l:=l+1;
      end;

cont:=cont-1;
mediaX:=somaX/(cont);
mediaY:=somaY/(cont);
mediaLOGN:=somaLOGN/(cont);

SXX:=somaX2-cont*(mediaX*mediaX);
SYY:=somaY2-cont*(mediaY*mediaY);
SXY:=somaXY-cont*mediaX*mediaY;
B1:=SXY/SXX;
B0:=mediaY-B1*mediaX;
label_arquivo.caption:='Arquivo em uso:  '+nomearq;

//////////////////////////
// linhas de tendencia //
//////////////////////////

// linear e exponencial

label_funcaoL.caption:='Y(x) = '+floattostr(B1)+ ' X + '+floattostr(B0);
ListChartSource2.Clear;
ListChartSource3.Clear;
AssignFile(arquivo,nomearq);
Reset (arquivo);
pos_i:=0;
l:=1;
```

```
while not Eof (arquivo) do
     begin
     ReadLn (arquivo,linha);
     sizeline:=length(linha);
     pos_i:=0;
     x[l]:=copy(linha,pos_i,pos(';',linha)-1);
     oX:=strtofloat(copy(linha,pos_i,pos(';',linha)-1));
     pos_i:=pos(';',linha)+1;
     l:=l+1;
     oY:=B1*oX+B0;
     ListChartSource2.Add(strtofloat(x[l]),oY);   // linear
     B:=((cont*somaXlogn-somaX*somaLOGN))/(cont*somaX2-(somaX*somaX));
     A:=mediaLOGN - B * mediaX;
     TB:=B*strtofloat(x[l]);
     YM:=exp(A)*exp(TB);
     ListChartSource3.Add(strtofloat(x[l]),(YM));  // exponencial
     label_funcaoE.caption:='Y(x) = '+floattostr(exp(A))+ ' * exp ('+floattostr(B) + ' * x)';

     end;

end;

procedure TForm1.btn_gerarClick(Sender: TObject);
begin
  lbl_inicial.visible:=true;
  edt_inicial.visible:=true;
  lbl_final.visible:=true;
  edt_final.visible:=true;
  btn_ok.visible:=true;
end;

procedure TForm1.btn_okClick(Sender: TObject);
begin
end;



procedure TForm1.Btn_conectaClick(Sender: TObject);
begin
  Btn_desconecta.visible:=true;
  Memo_terminal.Clear;
  SdpoSerial1.Device:=ComboBox_port.text;
  SdpoSerial1.BaudRate:=TBaudrate(ComboBox_baudrate.ItemIndex);
  SdpoSerial1.DataBits:=TDataBits(ComboBox_databits.ItemIndex);
  SdpoSerial1.FlowControl:=TFlowControl(ComboBox_flowcontrol.ItemIndex);
  SdpoSerial1.Parity:=TParity(ComboBox_parite.ItemIndex);
  SdpoSerial1.StopBits:=TStopBits(ComboBox_stopbits.ItemIndex);
  SdpoSerial1.Active:=True;
```

```
end;

procedure TForm1.Btn_desconectaClick(Sender: TObject);
begin
    SdpoSerial1.Active:=false;
end;

procedure TForm1.Chart1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  label_do_zoom.visible:=true;
end;

procedure TForm1.Chart1DrawReticule(ASender: TChart; ASeriesIndex,
  AIndex: Integer; const AData: TDoublePoint);
begin
  label_do_zoom.visible:=true;
end;

procedure TForm1.Chart1MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
  label_do_zoom.visible:=true;
end;

procedure TForm1.Chart1MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  label_do_zoom.visible:=true;
end;

procedure TForm1.CheckBox_exibeChange(Sender: TObject);
begin
if checkbox_exibe.checked=true then
//     chart1.Series.Items[1].
//chart1lineseries1.showpoints:=true;
Chart1Lineseries1.Marks.visible:=true
else
Chart1Lineseries1.Marks.visible:=false;

end;

procedure TForm1.ComboBox_sensorChange(Sender: TObject);
begin
  if combobox_sensor.ItemIndex<>-1 then
  btn_conecta.visible:=true;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
  var arquivo: textfile;
```

```
begin
  if radiogroup1.ItemIndex=0 then   // exponencial
   begin
    label12.visible:=false;
    edit_ordem_poli.visible:=false;
    updown1.visible:=false;
    chart1.Series.Items[1].Active:=false;
    chart1.Series.Items[2].Active:=true;
    label_funcaoL.visible:=false;
    label_funcaoE.visible:=true;
   end;
  if radiogroup1.ItemIndex=1 then   // linear
   begin
    label12.visible:=false;
    edit_ordem_poli.visible:=false;
    updown1.visible:=false;
    chart1.Series.Items[1].Active:=true;
    chart1.Series.Items[2].Active:=false;
    label_funcaoL.visible:=true;
    label_funcaoE.visible:=false;
   end;
  if radiogroup1.ItemIndex=2 then   // logaritmica
   begin
    label12.visible:=false;
    edit_ordem_poli.visible:=false;
    updown1.visible:=false;
    chart1.Series.Items[1].Active:=false;
    chart1.Series.Items[2].Active:=false;
    label_funcaoL.visible:=false;
    label_funcaoE.visible:=false;
   end;
  if radiogroup1.ItemIndex=3 then   // polinomial
   begin
     label12.visible:=true;
     edit_ordem_poli.visible:=true;
     updown1.visible:=true;
     chart1.Series.Items[1].Active:=false;
     chart1.Series.Items[2].Active:=false;
     label_funcaoL.visible:=false;
     label_funcaoE.visible:=false;
   end;
  if radiogroup1.ItemIndex=4 then   // potencia
   begin
    label12.visible:=false;
    edit_ordem_poli.visible:=false;
    updown1.visible:=false;
    chart1.Series.Items[1].Active:=false;
    chart1.Series.Items[2].Active:=false;
    label_funcaoL.visible:=false;
    label_funcaoE.visible:=false;
```

```
   end;

end;

procedure TForm1.SdpoSerial1RxData(Sender: TObject);
var rec:string;
  texto:TStringList;
  volt,volt_str:string;
  temp, voltage: real;

begin
 rec:=SdpoSerial1.ReadData;          // verificar o que ta chegando aqui
Memo_terminal.append(rec);
texto:=TStringlist.create;
texto.add(Memo_Terminal.lines.text);
texto.savetofile('C:\dados\ArquivoDeTeste.txt');
freeandnil(texto);

    ListBox1.Items.Add(rec);
    ListBox1.Items.SaveToFile('C:\dados\arquivo.csv');

  volt:=trim(Memo_terminal.Lines.text);
  volt_str:=StringReplace(volt, '.', ',',[rfReplaceAll, rfIgnoreCase]);
  volt_str:=trim(Memo_terminal.Lines.text);

//  if combobox_sensor.ItemIndex=0 then     // caso seja termopar tipo K
//  begin
//     temp := ((0.226584602) + (24152.10900 * voltage) + (67233.4248 * power(voltage,2)) +
(2210340.682 * power(voltage,3)) + (-860963914.9 * power(voltage,4)) + (4.83506E+10 *
power(voltage,5))+(-1.18452E+12 * power(voltage, 6))+( 1.38690E+13 * power(voltage,7))
+ (-6.33708E+13 * power(voltage, 8)));  // TERMOPAR TIPO K

end;

procedure TForm1.UpDown1Click(Sender: TObject; Button: TUDBtnType);
begin
  if strtofloat(edit_ordem_poli.text) < 2 then
     edit_ordem_poli.text:='2';
end;

Procedure FindReplace (const Enc, subs: String; Var Texto: TMemo);
Var
i, Posicao: Integer;
Linha: string;
Begin
For i:= 0 to Texto.Lines.count - 1 do
begin
Linha := Texto. Lines[i];
Repeat
Posicao:=Pos(Enc,Linha);
```

```
If Posicao > 0 then
Begin
Delete(Linha,Posicao,Length(Enc));
Insert(Subs,Linha,Posicao);
Texto.Lines[i]:=Linha;
end;
until Posicao = 0;
end;
end;
end.
```