UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

JOÃO PAULO MILANI

INTERFACE DE SENSORES ANALÓGICOS E DIGITAIS PARA PRÁTICAS DE MEDIDAS MECÂNICAS

São Carlos
2017

JOÃO PAULO MILANI

INTERFACE DE SENSORES ANALÓGICOS E DIGITAIS PARA PRÁTICAS DE MEDIDAS MECÂNICAS

Monografia apresentada ao Curso de Engenharia mecatrônica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Mecatrônico.

Orientador: Prof. Dr. Leopoldo Pisanelli Rodrigues de Oliveira

VERSÃO CORRIGIDA

São Carlos

2017

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

M637i

Milani, João Paulo
Interface de sensores analógicos e digitais para práticas de medidas mecânicas / João Paulo Milani; orientador Leopoldo Pisanelli Rodrigues de Oliveira. São Carlos, 2017.

Monografia (Graduação em Engenharia Mecatrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2017.

1. Interface. 2. Medidas. 3. Sensores. I. Título.

FOLHA DE AVALIAÇÃO

Candidato: JOÃO PAULO MILANI

Título: INTERFACE DE SENSORES ANALÓGICOS E DIGITAIS PARA PRÁTICAS DE MEDIDAS MECÂNICAS

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

BANCA EXAMINADORA

Professor Doutor Leopoldo Pisanelli Rodrigues de Oliveira (Orientador)

Nota atribuída: 8,5 (oito e meio) _____ (assinatura)

Professor Doutor Adriano Almeida Gonçalves Siqueira

Nota atribuída: 8,5 (oito e meio) _____ (assinatura)

Engenheiro Luiz Ferreira Alves

Nota atribuída: 8,5 (oito e meio) Luiz Ferreira Alves (assinatura)

Média: 8,5 (oito e meio)

Resultado: APROVADO

Data: 12/12/2017

Este trabalho tem condições de ser hospedado no Portal Digital da Biblioteca da EESC

SIM ☒ NÃO ☐ Visto do orientador _____

EPÍGRAFE

"A importância da Instrumentação poderia ser resumida em uma frase: 'A medição é a base do processo experimental.' Seja em um processo que deve ser controlado, seja em pesquisa ou em uma linha de produção dentro de uma indústria, o processo da medição de grandezas físicas é fundamental."

Alexandre Balbinot e Valner João Brusamarello (2010)

RESUMO

MILANI, J. P. **Interface de sensores analógicos e digitais para práticas de medidas mecânicas**. 2017. 198 f. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2017.

Esta tese visa à análise da aplicação de interfaces gráficas para a leitura de sensores em práticas do laboratório de ensino. Durante este processo, foi-se analisado um sensor de temperatura em conjunto com um microcontrolador Arduino Uno, um potenciômetro linear rotativo em conjunto com a GUI do Matlab® e uma célula de carga também em conjunto com o Arduino Uno. Os resultados se provaram satisfatórios às duas primeiras aplicações, por serem de baixo custo e realização relativamente fácil, porém à última delas foi verificado que as condições ambientais em que as medidas são realizadas podem influenciar negativamente o resultado, e por isso é considerada uma prática que precisaria de uma atenção a mais, caso fosse implantada. A GUI do Matlab®, por outro lado, consiste de uma ferramenta muito útil e de fácil utilização, por isso podendo ser utilizada em conjunto com outros microcontroladores e sensores, na criação de novas interfaces com um grande número de opções, sendo muito abrangente. Assim, concluiu-se que o uso de interfaces gráficas pode fazer das práticas de laboratório um ensino multidisciplinar muito completo, podendo exigir conhecimentos de programação, obtenção e leitura de dados, além de apresentar novos tipos de sensores aos estudantes.

Palavras-chave: Interface. Medidas. Sensores.

ABSTRACT

MILANI, J. P. **Analogic and digital sensor interface for mechanical measurement practices**. 2017. 198 f. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2017.

This thesis aims at the analysis of the application of graphical interfaces for reading of sensors in practices of teaching laboratories. During this process, a temperature sensor was analyzed in conjunction with an Arduino Uno microcontroller, a linear-taper potentiometer in conjunction with the Matlab® GUI and a load cell also in conjunction with the Arduino Uno board. The results proved to be satisfactory for the first two applications, since they are relatively cheap and relatively easy to carry out, but the last of them was tested and it was proved that the environmental conditions in which the measurements are carried out can negatively influence the results, proving though a practice that would need more attention, if it were implemented. The Matlab® GUI, on the other hand, proved to be a very useful and user-friendly tool. So it can be used in conjunction with other microcontrollers and sensors, in creation of new interfaces with a large number of options, being very embracing. Thus, it was concluded that the use of graphical interfaces can make laboratory practices a very complete multidisciplinary teaching, which may require programming skills and data acquisition knowledge. Furthermore present new types of sensors to the students.

Keywords: Interface. Mesurements. Sensors.

LISTA DE ILUSTRAÇÕES

Figura 1 Disposição das portas do Arduino Uno 27
Figura 2 Potenciômetro do modelo B10K 28
Figura 3 Configuração da ponte de Wheatstone 29
Figura 4 Célula de carga com capacidade de 5kg 29
Figura 5 Amplificador de sinal HX 711 30
Figura 6 Circuito do Arduino com o sensor de temperatura 33
Figura 7 Primeira parte do programa para leitura do sensor 34
Figura 8 Segunda parte do programa para leitura de temperatura 35
Figura 9 Última parte do programa para leitura de temperatura 35
Figura 10 Montagem do circuito Arduino-Potenciômetro 36
Figura 11 GUI do Matlab® 37
Figura 12 Interface para leitura de um potenciômetro 38
Figura 13 Código para inicialização 39
Figura 14 Inicialização da interface 40
Figura 15 Codificação do botão "Ler" 41
Figura 16 Código da caixa de texto editável 42
Figura 17 Interface com botão liga/desliga 43
Figura 18 Código do botão liga/desliga 44
Figura 19 Código do botão "Salvar" 45
Figura 20 Caixa de diálogo padrão com textos alterados 45
Figura 21 Montagem do circuito para uso da célula de carga 46
Figura 22 Programa para calibrar e medir utilizando célula de carga 47
Figura 23 Monitor serial com valores de temperaturas medidas 49
Figura 24 Interface gráfica para a leitura de um potenciômetro 50
Figura 25 Experimento dirigido ao programa com botão liga/desliga 52
Figura 26 Caixa de diálogo para salvar os valores 53
Figura 27 Medições e variáveis obtidas com a célula de carga 54

LISTA DE TABELAS

Tabela 1 Vetores de tempo e voltagem medida obtidos através do experimento.....51
Tabela 2 Vetores de tempo e voltagem medida obtidos através deste experimento.....52

LISTA DE ABREVIATURAS E SIGLAS

SEM	–	Departamento de Engenharia Mecânica
EESC	–	Escola de Engenharia de São Carlos
USP	–	Universidade de São Paulo
USB	–	*Universal Serial BUS*
GUI	–	*Graphical User Interface*
GND	–	*Ground*
DT	–	Saída de dados do HX 711
IDE	–	*Integrated Development Environment*
I2C	–	*Inter-Integrated Circuit*

LISTA DE SÍMBOLOS

V	Volt
kB	Quilo bytes
Ω	Ohm
W	Watt
kg	quilograma
ºC	Graus Celsius
s	segundo
g	grama

Sumário

1. INTRODUÇÃO 25
1.1. Objetivo 25
1.2. Motivação 25
1.3. Considerações 25
2. MATERIAIS E MÉTODOS 27
2.1. Microcontrolador 27
2.2. Sensores 28
2.2.1. Potenciômetro 28
2.2.2. Sensor de temperatura 28
2.2.3. Célula de carga 28
2.3. Softwares 30
2.3.1. Arduino 1.8.5 30
2.3.2. Matlab® 30
3. PROCEDIMENTOS EXPERIMENTAIS 33
3.1. Sensor de temperatura com Arduino Uno 33
3.1.1. Montagem do circuito 33
3.1.2. Programação do Arduino 33
3.2. Potenciômetro linear rotativo com Matlab® 36
3.2.1. Montagem do circuito 36
3.2.2. GUI do Matlab® 36
3.2.3. Medida com tempo pré-determinado 38
3.2.4. Medida com botão liga/desliga 42
3.3. Célula de carga 46
3.3.1. Montagem do sistema 46
3.3.2. Programação do Arduino 46
4. RESULTADOS E DISCUSSÕES 49

4.1. Sensor de temperatura 49
4.2. Potenciômetro linear rotativo 50
4.2.1. Medida com tempo pré-determinado 50
4.2.2. Medida com botão liga/desliga 51
4.3. Célula de carga 53
5. CONCLUSÃO 57
REFERÊNCIAS 59
Apêndice A – Prática com sensor de temperatura 61
Apêndice B – Prática com GUI do Matlab® 63

1. INTRODUÇÃO

1.1. Objetivo

Esta tese tem como principal objetivo a elaboração de novas práticas de laboratório de sistemas de medidas mecânicas, bem como testes que comprovem a efetividade das mesmas na aplicação para alunos do programa de graduação, aos quais se aplicam tais conhecimentos técnicos. Para isto, foi-se criada uma interface simpática ao usuário, por meio de uma ferramenta já conhecida pelos graduandos, o Matlab® em conjunto com o Arduino Uno em comunicação com diferentes sensores e a disposição das medições em tempo real na forma de gráficos, facilitando a interpretação das mesmas.

Buscou-se também a elaboração de exemplos práticas que pudessem ser aplicadas no laboratório de ensino. Estas serão apresentadas nos apêndices A e B, em forma de sugestões não formalizadas de práticas a serem utilizadas nos laboratórios de ensino da EESC - USP.

1.2. Motivação

A principal motivação para a elaboração deste estudo é a possibilidade de aumentar os conhecimentos na área de sistemas de medidas mecânicas, assim como auxiliar na elaboração de práticas de laboratório de ensino que possam fazer parte das matérias de graduação, adicionando mais conhecimentos técnicos e práticos aos integrantes da disciplina SEM 0142 – Medidas Mecânicas.

1.3. Considerações

A escolha pelo Matlab® e pelo Arduino Uno, deu-se pela familiaridade dos graduandos com tais ferramentas, as quais são comumente utilizadas durante a graduação em engenharias mecânica e mecatrônica, principalmente, por ter como parte da grade disciplinas de controle e sensoriamento. Sendo assim, há uma proximidade maior com a linguagem de programação utilizada.

Há no mercado ferramentas *Open-source*, que poderiam ser utilizadas também para a implementação destas mesmas práticas, porém, pelos motivos acima citados, seu estudo de viabilidade foi descartado nesta tese.

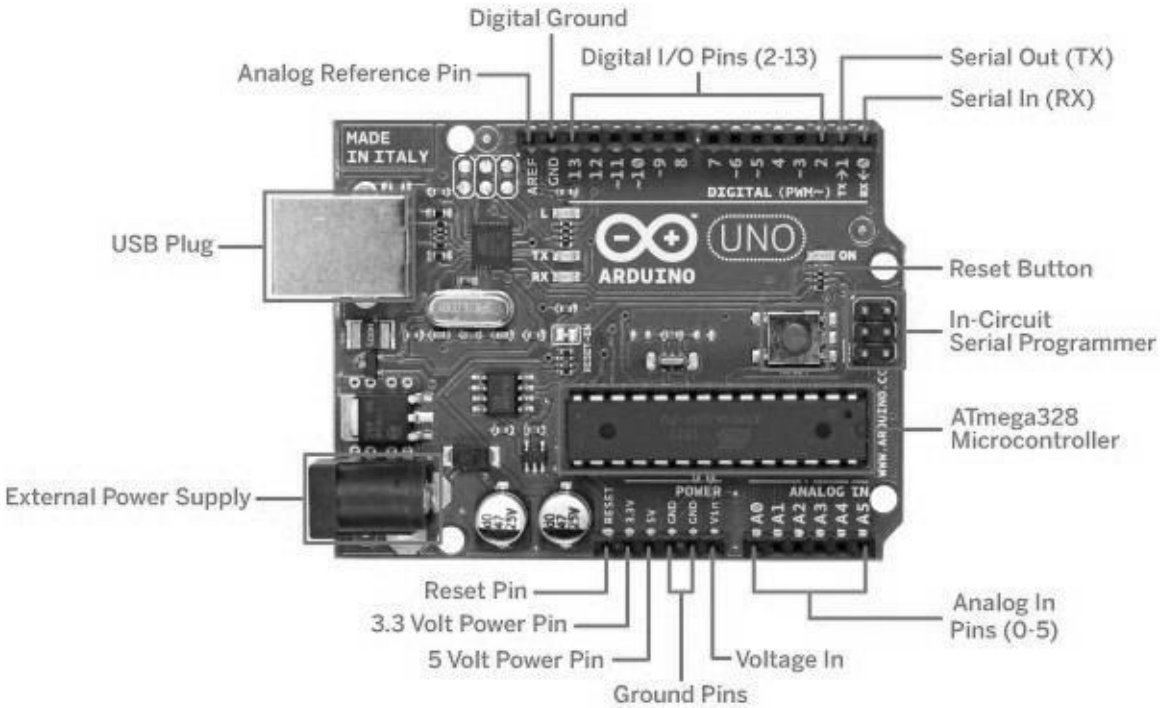
Como citado na seção 1.1, o estudo em questão se direciona a uma disciplina específica do curso de engenharia mecânica, que não se aprofunda muito na parte de eletrônica, que inclui aquisição de dados por meio de sensores, portanto, a elaboração das práticas visa simplificar a leitura dos dados para estes graduandos especialmente, por isso o tempo de amostragem não é tratado de forma crucial, por exemplo; algumas imprecisões também são desconsideradas, já que para o objetivo central desta práticas tornam-se irrelevantes.

2. MATERIAIS E MÉTODOS

2.1. Microcontrolador

Para a elaboração dos testes, foi-se utilizado o Arduino Uno, com as portas discriminadas segundo a figura 1, para a aquisição e processamento de dados. Sua escolha foi baseada na sua capacidade de armazenamento de dados de 32KB, sendo que 2KB são designados para a programação, sendo esta quantidade mais que suficiente para as aplicações desejadas. Além disso, as portas de entrada e saída disponíveis são analógicas e digitais, o que engloba todos os sensores que poderiam ser utilizados em uma prática de laboratório de ensino. Há uma porta USB, que faz a conexão do Arduino com o computador, além da alimentação da placa, dispensando o uso de uma fonte externa. A programação é feita em uma linguagem fácil e intuitiva, o que favorece também a escolha deste hardware. Algo também determinante é o preço acessível.

Figura 1 Disposição das portas do Arduino Uno



Fonte – arduinoarts.com (2011)

2.2. Sensores

2.2.1. Potenciômetro

O potenciômetro abordado nesta tese é do tipo linear rotativo B10k, ilustrado na figura 2, com resistência máxima de 10000Ω e potência máxima de 0,2W. A escolha deste sensor será explicada melhor nos tópicos a seguir.

Figura 2 Potenciômetro do modelo B10K

Fonte – google.com (2017)

2.2.2. Sensor de temperatura

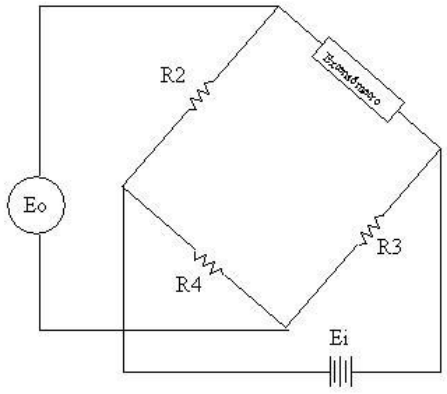
Um sensor de temperatura do modelo DS18B20, que funciona com alimentação de 3,0V a 5,5V, em corrente contínua, e tem precisão de +/- 0,5ºC em leituras entre -10ºC e 85ºC, porém podendo ser utilizados para leituras entre -55ºC e 125ºC, apesar da queda de precisão. A comunicação com o Arduino é realizada através de uma porta digital, e a resolução do sensor é de 9 ou 12 bits.

Este sensor tem custo acessível e pode ser utilizado para práticas muito didáticas, devido ao seu fácil funcionamento, no sentido de montagem do circuito e programação do Arduino. Estes tópicos serão abordados na sessão 3 deste documento.

2.2.3. Célula de carga

Células de carga são utilizadas para criar um sinal elétrico a partir de uma entrada física (força). As células comumente utilizadas funcionam através de quatros extensômetros, que são dispostos formando uma ponte de Wheatstone, como apresentado na figura 3.

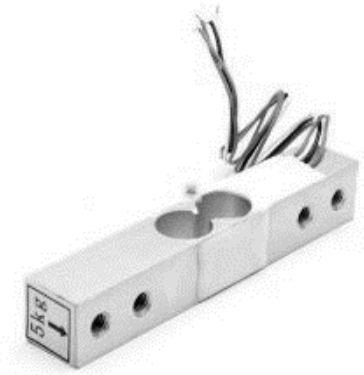
Figura 3 Configuração da ponte de Wheatstone



Fonte – Unicamp (2017)

Quando estes extensômetros são expostos a uma força, eles se deformam e geram uma diferença em sua resistência, o que gera um sinal diferente na saída, que é proporcional à força aplicada. O corpo da célula de carga, onde a ponte de Wheatstone se encontra fixada, é de metal e é também totalmente solidária à sua deformação. Este é projetado e produzido com meticuloso cuidado para garantir a proporcionalidade e repetibilidade das medições. Com isso, a célula de carga é projetada no formato exposto na figura 4.

Figura 4 Célula de carga com capacidade de 5kg

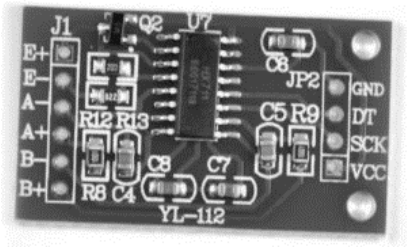


Fonte – google.com (2017)

O modelo utilizado possui quatro fios ligados a ele, dois são responsáveis pela alimentação da célula de carga, e dois são responsáveis pela leitura do sinal de saída, sendo um o *clock*, que libera a transmissão de dados, e o outro, os dados propriamente ditos.

O uso de um amplificador de sinal é imperativo para as aplicações desejadas, já que o sinal de saída do sensor utilizado está em milivolts por volt (mV/V), o que impede uma leitura direta pelo Arduino Uno. O mais comumente utilizado, e também escolhido, foi o HX 711, disposto na figura 5, um conversor analógico-digital com precisão de 24 bits. Ele é utilizado para medidas em escala de peso e para aplicações industriais.

Figura 5 Amplificador de sinal HX 711



Fonte – circuits4you.com (2016)

2.3. Softwares

2.3.1. Arduino 1.8.5

O Arduino Uno pode ser programado a partir do software *open-source* homônimo, que está em sua versão 1.8.5. Por meio deste, também se faz possível o acompanhamento das leituras de todas as portas, assim como o envio de sinais para estas. A linguagem de programação é intuitiva e de fácil aprendizagem. A interface é simples e compacta. Por estes motivos é um software perfeito para qualquer nível de usuário.

Por ser um *open-source*, o desenvolvimento de novos programas e bibliotecas é frequente, o que auxilia muito na criação de novos projetos. Nestes experimentos, fez-se uso de uma biblioteca para comunicação do Arduino com o Matlab® e uma biblioteca para a leitura do amplificador anteriormente citado, o HX 711, ambas disponibilizadas no próprio site do Arduino, já com instruções de uso.

2.3.2. Matlab®

O Matlab® é uma ferramenta comumente utilizada nos cursos de engenharia pela sua versatilidade, por possuir métodos de simulação e criação de gráficos em 2D ou até mesmo em 3D, por possuir também uma linguagem de programação de fácil aprendizado, se

assemelhando ao C++ e, como será abordado posteriormente nesta tese, por possuir uma ferramenta para o desenvolvimento de uma interface gráfica simpática ao usuário, com possibilidade de criação de gráficos em tempo real.

O Matlab® possui um banco de dados com programas e ajudas, dispostos através do site *mathworks.com*, que auxilia muito o usuário, por ser muito completo, possuindo, além das explicações, diversos exemplos, que demonstram todo o potencial de tudo o que havia sido anteriormente esclarecido.

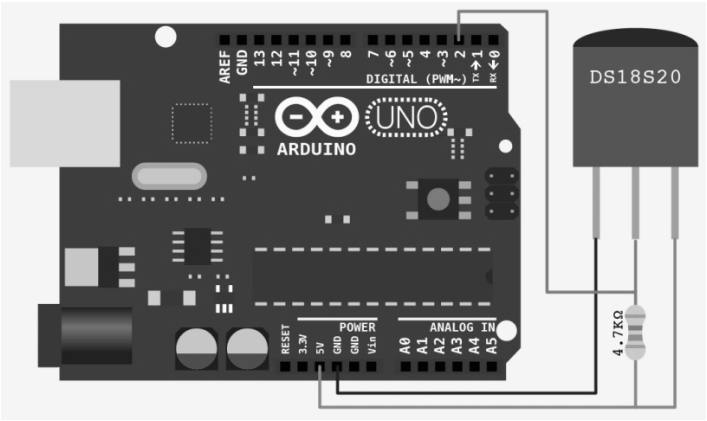
3. PROCEDIMENTOS EXPERIMENTAIS

3.1. Sensor de temperatura com Arduino Uno

3.1.1. Montagem do circuito

A montagem do circuito para a leitura do sensor de temperatura do modelo DS18B20 necessita apenas de um resistor de 4.7kΩ, fazendo a conexão do fio de leitura do sensor com o fio ligado aos 5V do Arduino. O último dos fios é aterrado. A montagem pode ser observada na figura 6, que apresenta outro modelo de sensor, o DS18S20, porém que segue a mesma configuração.

Figura 6 Circuito do Arduino com o sensor de temperatura



Fonte – bildr.org (2011)

O resistor anteriormente citado funciona como *pull-up*, ou seja, evita flutuações nas portas configuradas como entradas digitais, mantendo seus valores em 0 ou em 1, dependendo da situação. Muitos microcontroladores já possuem *pull-ups* internos, porém não é o caso do Arduino Uno, portanto o circuito apresentado na figura 6 se faz imperativo.

3.1.2. Programação do Arduino

Para a programação do Arduino em conjunto com este termômetro, é necessário o download do protocolo *OneWire*; com este protocolo torna-se possível a leitura de mais de um sensor ligado a um mesmo barramento, sendo que o número máximo de sensores que podem ser lidos ao mesmo tempo chega a 127. Isto acontece pela determinação de diferentes

números seriais para cada termômetro e controle do barramento, para que não haja envios simultâneos de respostas. Este protocolo deve ser usado em conjunto com a biblioteca *Dallas Temperature* para o correto funcionamento dos sensores dos modelos DS18B20, DS18S20, DS1822 e DS1820.

Na programação utilizada, exposta nas figuras 7, 8 e 9, faz-se uso também da biblioteca *LiquidCrystal* já presente nas bibliotecas padrão do Arduino. Ela foi usada para que um monitor de cristal líquido pudesse ser acoplado ao sistema, para a verificação da temperatura lida, porém, pode-se dispensar o uso do mesmo e acompanhar as leituras pelo monitor serial do Arduino mesmo, pelo computador. A opção pela programação deste foi por dar uma possibilidade a mais para o usuário.

Figura 7 Primeira parte do programa para leitura do sensor

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

// Porta do pino de sinal do DS18B20
#define ONE_WIRE_BUS 3

// Define uma instancia do oneWire para comunicacao com o sensor
OneWire oneWire(ONE_WIRE_BUS);

// Armazena temperaturas minima e maxima
float tempMin = 999;
float tempMax = 0;

DallasTemperature sensors(&oneWire);
DeviceAddress sensor1;

// Inicializa o LCD
LiquidCrystal lcd(12, 11, 7, 6, 5, 4);

void setup(void)
{
  Serial.begin(9600);
  sensors.begin();
  // Localiza e mostra enderecos dos sensores
  Serial.println("Localizando sensores DS18B20...");
  Serial.print("Foram encontrados ");
  Serial.print(sensors.getDeviceCount(), DEC);
  Serial.println(" sensores.");
  if (!sensors.getAddress(sensor1, 0))
     Serial.println("Sensores nao encontrados !");
  // Mostra o endereco do sensor encontrado no barramento
  Serial.print("Endereco sensor: ");
  mostra_endereco_sensor(sensor1);
  Serial.println();
  Serial.println();
  lcd.begin(16, 2);

}
```

Fonte – Arduino (2017)

Figura 8 Segunda parte do programa para leitura de temperatura

```
void mostra_endereco_sensor(DeviceAddress deviceAddress)
{
  for (uint8_t i = 0; i < 8; i++)
  {
    // Adiciona zeros se necessário
    if (deviceAddress[i] < 16) Serial.print("0");
    Serial.print(deviceAddress[i], HEX);
  }
}

void loop()
{
  // Le a informacao do sensor
  sensors.requestTemperatures();
  float tempC = sensors.getTempC(sensor1);
  // Atualiza temperaturas minima e maxima
  if (tempC < tempMin)
  {
    tempMin = tempC;
  }
  if (tempC > tempMax)
  {
    tempMax = tempC;
  }
  // Mostra dados no monitor serial
  Serial.print("Temp C: ");
  Serial.print(tempC);
  Serial.print(" Min : ");
  Serial.print(tempMin);
  Serial.print(" Max : ");
  Serial.println(tempMax);
```

Fonte – Arduino (2017)

Figura 9 Última parte do programa para leitura de temperatura

```
  // Mostra dados no LCD
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Temp.:        ");
  //Simbolo grau
  lcd.write(223);
  lcd.print("C");
  lcd.setCursor(7,0);
  lcd.print(tempC);
  lcd.setCursor(0,1);
  lcd.print("L: ");
  lcd.setCursor(3,1);
  lcd.print(tempMin,1);
  lcd.setCursor(8,1);
  lcd.print("H: ");
  lcd.setCursor(11,1);
  lcd.print(tempMax,1);
  delay(500);
}
```

Fonte – Arduino (2017)

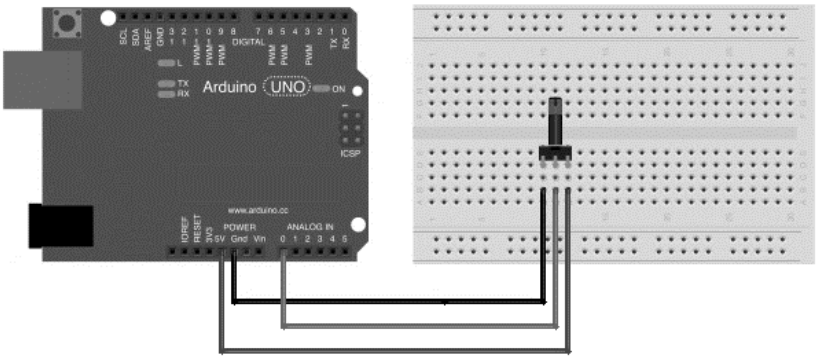
Os códigos apresentados possuem comentários explicando o uso das funções, facilitando o entendimento dos mesmos. As temperaturas são dadas já em ºC.

3.2. Potenciômetro linear rotativo com Matlab®

3.2.1. Montagem do circuito

Para o experimento em questão, não é necessário o uso de adicionais. Simplesmente liga-se a alimentação e o aterramento nos pinos das extremidades do sensor, a inversão das ligações altera apenas a polaridade, e o pino do meio é utilizado para leitura de dados, devendo, portanto, ser ligado a uma porta analógica do Arduino Uno. Esta montagem pode ser observada na figura 10.

Figura 10 Montagem do circuito Arduino-Potenciômetro

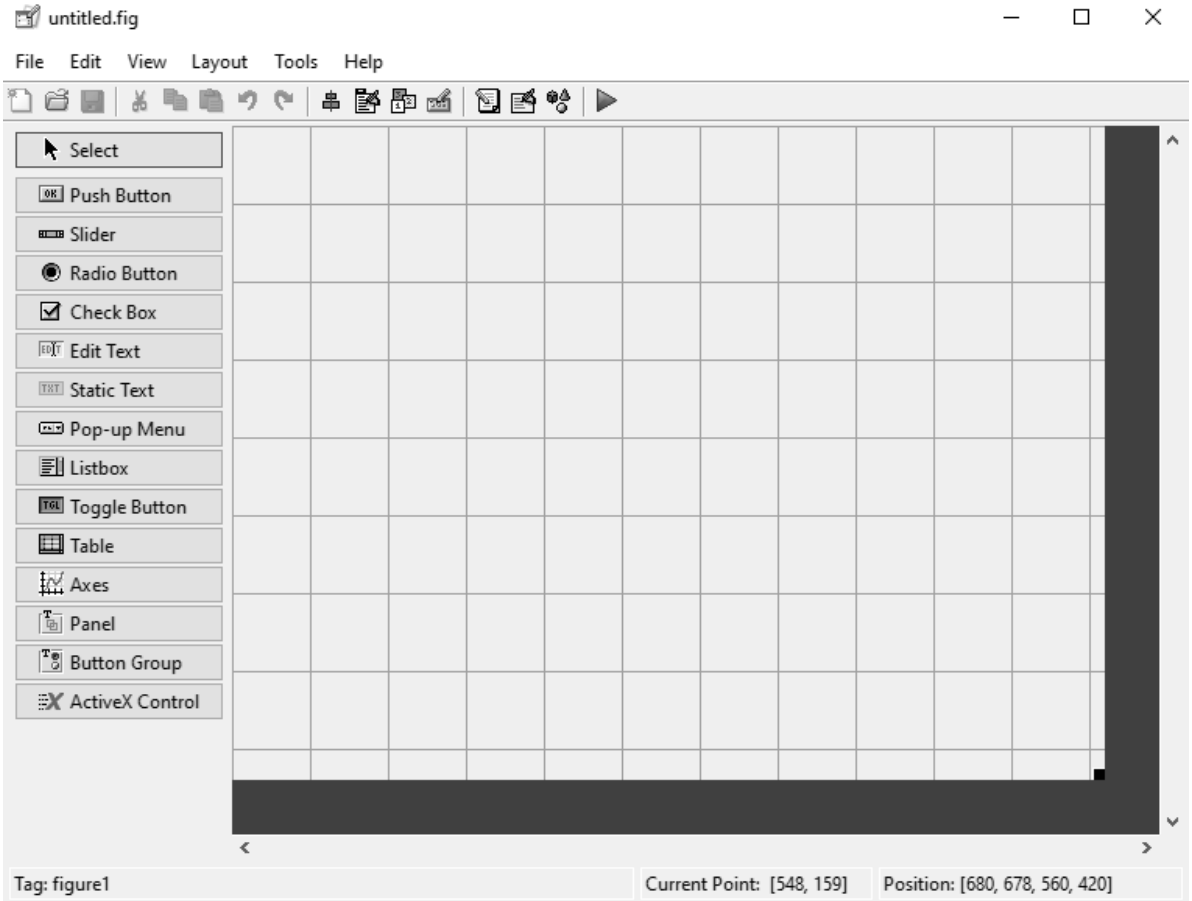


Fonte – saigesp.es (2015)

3.2.2. GUI do Matlab®

O Matlab® possui uma ferramenta para criação de interface gráfica para os usuários, chamada GUI, localizada no menu New→App→GUIDE, do Matlab® 2016. Esta interface com o usuário consiste em componentes que podem ser ativados pelo usuário, a fim de interagir com a interface e ter uma resposta previamente esperada, mesmo desconhecendo as funções por trás da programação. Estes componentes podem ser botões, gráficos, ferramentas de múltipla escolha e assim por diante, como pode ser observado na figura 11, que é a representação da área de criação de tais interfaces.

Figura 11 GUI do Matlab®



Fonte – Matlab® (2016)

Esta é uma ferramenta de fácil utilização, por ter grande suporte Online para eventuais dúvidas, além de uma apresentação autoexplicativa, como pode ser observado na figura 11, onde se veem as etiquetações de cada botão, apresentadas em forma de texto.

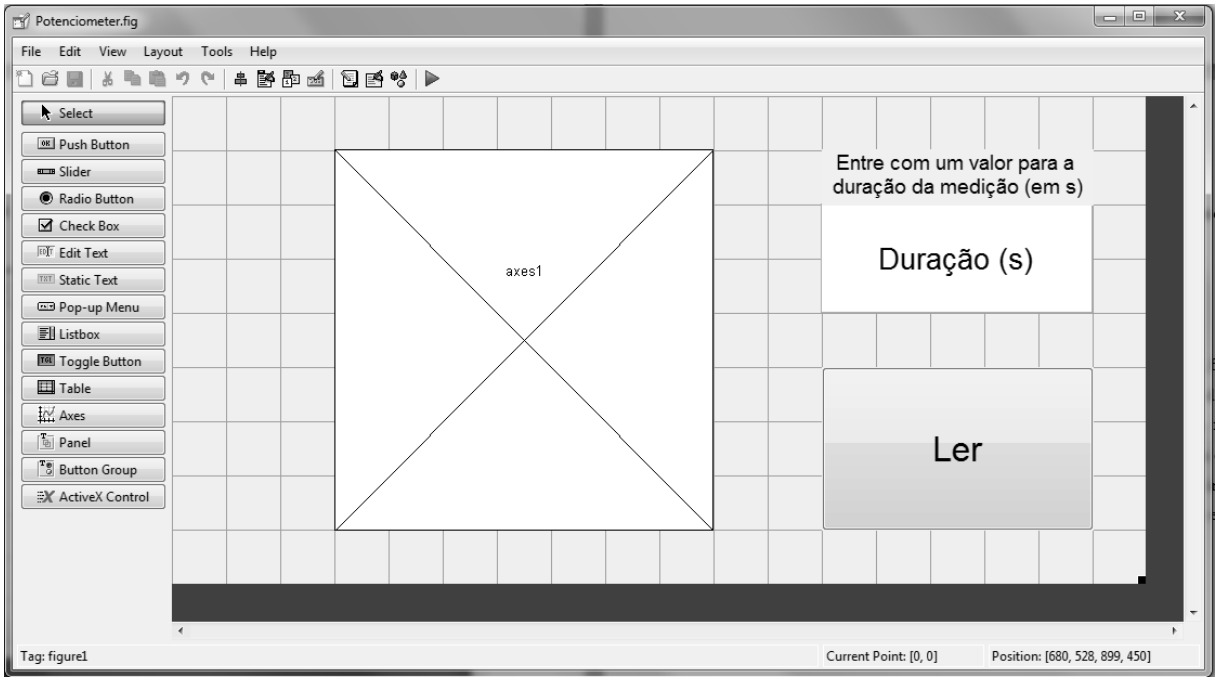
Ao final da criação desta figura, o próprio GUI cria a codificação básica de cada componente adicionado, restando ao programador apenas a codificação de cada componente em si. Por exemplo, no caso de um botão que desenhe um gráfico qualquer. O programador inclui um componente “Axes” e um “Push Button”. No código gerado pela GUI, em extensão .m (arquivo do Matlab®), aparecerão as funções básicas para o funcionamento da interface e os espaços editáveis, onde o programador implementa as funções do botão, sendo que estas serão executadas quando o mesmo for acionado, o que, neste caso, inclui apenas a resolução de uma equação (y = x, por exemplo, onde x foi previamente definido) e a impressão dos resultados obtidos no gráfico desenhado na figura, isto será melhor visualizado no desenvolvimento da tese. Para o usuário, a interface terá um gráfico, em branco no primeiro momento, e um botão acionável. Naturalmente, este é um exemplo simples, apenas para dar

uma noção do uso da ferramenta. Podem-se desenvolver figuras complexas, com uma gama de opções extensa, dependendo da dedicação e do conhecimento do usuário.

3.2.3. Medida com tempo pré-determinado

Dando início ao desenvolvimento de um trabalho com a GUI, foi-se estudado um meio simples de ler os sinais enviados por um potenciômetro através do Arduino Uno para o Matlab®, inserindo uma contagem pré-determinada de tempo. A figura desenvolvida foi denominada *Potenciometro*, e incluiu uma caixa de texto fixo, para dar instruções, uma caixa de texto editável, que é utilizada para indicar o tempo de operação, um botão para o início da contagem e um gráfico, que é desenhado em tempo real, acompanhando os dados enviados pelo sensor. A disposição dos componentes pode ser observada na figura 12.

Figura 12 Interface para leitura de um potenciômetro



Fonte – Matlab® (2016)

O código, originado através da figura salva anteriormente, pode ser observado nas figuras 13, 14, 15 e 16, dispostas a seguir. As edições realizadas foram devidamente comentadas para melhor compreensão, os comentários em inglês são instruções de uso fornecidas pela própria GUI, o que também facilita a utilização por usuários iniciantes.

Figura 13 Código para inicialização

```
function varargout = Potenciometro(varargin)
% POTENCIOMETRO MATLAB code for Potenciometro.fig
%      POTENCIOMETRO, by itself, creates a new POTENCIOMETRO or raises the existing
%      singleton*.
%
%      H = POTENCIOMETRO returns the handle to a new POTENCIOMETRO or the handle to
%      the existing singleton*.
%
%      POTENCIOMETRO('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in POTENCIOMETRO.M with the given input arguments.
%
%      POTENCIOMETRO('Property','Value',...) creates a new POTENCIOMETRO or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Potenciometro_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Potenciometro_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Potenciometro

% Last Modified by GUIDE v2.5 28-Nov-2017 17:36:22

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Potenciometro_OpeningFcn, ...
                   'gui_OutputFcn',  @Potenciometro_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

Fonte – Matlab® (2016)

Esta primeira parte do código é a criação de uma função, que é chamada ao inicializar a figura. Com exceção do nome, no caso *Potenciometro*, o restante do código é o mesmo para qualquer criação realizada através da GUI e, por isso, será omitida para as codificações futuras.

Figura 14 Inicialização da interface

```
47      % --- Executes just before Potenciometro is made visible.
48      function Potenciometro_OpeningFcn(hObject, eventdata, handles, varargin)
49      % This function has no output args, see OutputFcn.
50      % hObject    handle to figure
51      % eventdata  reserved - to be defined in a future version of MATLAB
52      % handles    structure with handles and user data (see GUIDATA)
53      % varargin   command line arguments to Potenciometro (see VARARGIN)
54
55      % Choose default command line output for Potenciometro
56 -    handles.output = hObject;
57
58      % Update handles structure
59 -    guidata(hObject, handles);
60
61      % UIWAIT makes Potenciometro wait for user response (see UIRESUME)
62      % uiwait(handles.figure1);
63
64      %Limpa-se a porta 'COM3', para evitar erros
65 -    delete(instrfind({'Port'}, {'COM3'}));
66      %Limpa-se a variável a
67 -    clear a;
68      %Inicia-se o Arduino na variável global 'a'
69 -    global a;
70 -    a = arduino('COM3');
71
72
73      % --- Outputs from this function are returned to the command line.
74      function varargout = Potenciometer_OutputFcn(hObject, eventdata, handles)
75      % varargout  cell array for returning output args (see VARARGOUT);
76      % hObject    handle to figure
77      % eventdata  reserved - to be defined in a future version of MATLAB
78      % handles    structure with handles and user data (see GUIDATA)
79
80      % Get default command line output from handles structure
81 -    varargout{1} = handles.output;
```

Fonte – Matlab® (2016)

Na parte do código exposta na figura 14, fez-se apenas o acréscimo das linhas 64 a 70, que consiste na inicialização do Arduino Uno e sua leitura realizada através da variável global 'a'. O restante do código é gerado automaticamente, assim como o da figura 13 e, por esse motivo, também será omitido futuramente.

A leitura do Arduino através do Matlab® só é possível com o auxilio de uma biblioteca, formalmente conhecida como "*Arduino IO Package*", mas também encontrada com o nome "*Legacy MATLAB and Simulink Support for Arduino*", baixada gratuitamente do site da MathWorks®, citado nas referências ao final deste documento.

Figura 15 Codificação do botão "Ler"

```
84      % --- Executes on button press in read_button.
85      function read_button_Callback(hObject, eventdata, handles)
86      % hObject    handle to read_button (see GCBO)
87      % eventdata  reserved - to be defined in a future version of MATLAB
88      % handles    structure with handles and user data (see GUIDATA)
89
90      %Declaração das variáveis
91 -    global a;
92 -    t = 0;
93 -    x = 0;
94 -    d = 0;
95 -    n = 0;
96 -    f = 0;
97
98 -    while (d <= (handles.xSamples))
99
100 -       tic %Inicia-se a contegem do tempo
101
102         %Armazena a leitura da porta A0 em uma variável, aplicando a conversão de valores
103 -       Pot = a.analogRead(0)*5/1024;
104 -       x = [x, Pot]; %Cria um vetor para a leitura
105 -       t = [t, d];   %Cria um vetor de tempo
106 -       plot (t(1:n), x(1:n), 'LineWidth', 1.5); %Plotagem
107 -       grid on;
108 -       axis ([0 handles.xSamples 0 5]);
109
110         %Apresentação da tabela
111 -       xlabel ('Tempo (s)');
112 -       ylabel ('Voltagem (V)');
113 -       title ('Experimento com Potenciometro');
114
115 -       pause(0.01); %Pausa necessária para leitura e processamento no Arduino
116
117 -       n = n + 1; %Incrementa em 1 a variável responsável pale definição do valor a ser plotado
118 -       d = d + f; %Incrementa em f a variável de duração total decorrida
119 -       f = toc; %Para-se a contagem do tempo
120
121 -   end
122
123 -   save ('Teste.mat', 't', 'x', '-append'); %Salva as leituras e o tempo em arquivo .mat
124
125     %display (d); %Para verificar o tempo total decorrido, tirar o comentário desta função
```

Fonte – Matlab® (2016)

Na figura 15, pode-se observar o que ocorre ao se pressionar o botão, intitulado "Ler". As funções *tic* e *toc* foram necessárias, porque o programa é normalmente contado em pulsos, ou seja, o usuário deveria entrar com um número de pulsos para a contagem, e o gráfico seria dado em Pulsos x Leitura do Potenciômetro, o que fornece dados pouco palpáveis, portanto, foi-se implementada a contagem do tempo com as funções citadas e com variáveis auxiliares, no caso, "n", para auxiliar a plotagem, "d", "t" e "f", para a contagem do tempo.

A "conversão de valores" citada nos comentários do código faz referência à constante de proporcionalidade que multiplica o valor lido pela porta A0 (comando a.analogRead(0)), que deve estar presente para a leitura em Volts do sensor, já que, por se tratar de uma porta analógica, o resultado da leitura sai em bits, ou seja, em valores entre 0 e 1023, que é a resolução da porta. Para a conversão, multiplicou-se o valor da tensão de alimentação (5V) e em seguida dividiu-se pela resolução da porta (1024).

No laço de repetição, fez-se uso da variável "handles.xSamples", a qual é determinada na função da edição de texto, que pode ser verificada na figura 16.

Após o término da execução do programa, os dados obtidos são salvos em um arquivo criado anteriormente em .mat (formato de dados do Matlab®), no caso, o arquivo foi denominado "Teste.mat" e está guardando as variáveis "t" e "x", que representam o tempo decorrido e as medições realizadas, respectivamente.

Figura 16 Código da caixa de texto editável

```
127     function edit_text_samples_Callback(hObject, eventdata, handles)
128     % hObject    handle to edit_text_samples (see GCBO)
129     % eventdata  reserved - to be defined in a future version of MATLAB
130     % handles    structure with handles and user data (see GUIDATA)
131
132     % Hints: get(hObject,'String') returns contents of edit_text_samples as text
133     %        str2double(get(hObject,'String')) returns contents of edit_text_samples as a double
134
135     %Recebe o INPUT do usuário para a duração
136 -   handles.duracao = get(hObject, 'String');
137
138     %Transforma o dado de String para Double
139 -   handles.xSamples = str2double(handles.duracao);
140
141     %Atualiza as variáveis do programa
142 -   guidata(hObject, handles);
143
144     % --- Executes during object creation, after setting all properties.
145     function edit_text_samples_CreateFcn(hObject, eventdata, handles)
146     % hObject    handle to edit_text_samples (see GCBO)
147     % eventdata  reserved - to be defined in a future version of MATLAB
148     % handles    empty - handles not created until after all CreateFcns called
149
150     % Hint: edit controls usually have a white background on Windows.
151     %       See ISPC and COMPUTER.
152 -   if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
153 -       set(hObject,'BackgroundColor','white');
154 -   end
```

Fonte – Matlab® (2016)

O código exposto acima é simples e está devidamente comentado de forma a não gerar dúvidas. A única observação a ser feita é o uso da função implementada entre as linhas 144 e 154, que é executada durante a criação do objeto e depois de passar todas as configurações. Ela é usada na configuração das cores de fundo da caixa de texto, sendo que "ispc" verifica se o sistema operacional é Windows e "isequal" compara a cor definida para o fundo da caixa de texto e a cor padrão definida pelo programador.

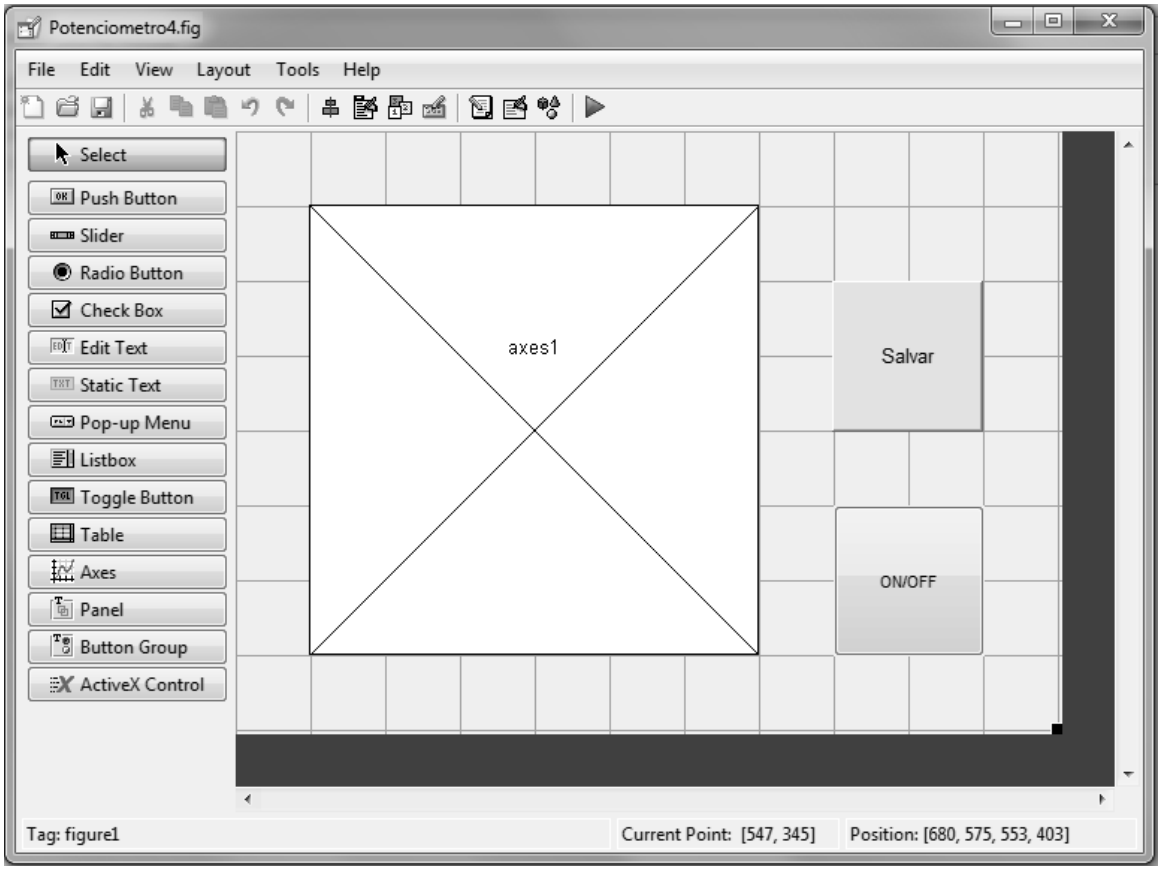
3.2.4. Medida com botão liga/desliga

Uma nova interface foi criada com a finalidade de dar mais liberdade ao usuário quanto ao tempo de medição, principalmente quando não há certeza no tempo exato, durante

o qual os dados devem ser tomados. Neste ponto, foi-se abordado a confecção de um botão liga/desliga para a obtenção dos dados, assim como um botão que permita o salvamento dos mesmos, evitando a perda de medidas previamente realizadas sem o devido consentimento do usuário, já que a função "save" do modo em que foi utilizada sobrescreve os dados, por salvar todas as medidas em um mesmo arquivo.

A nova interface pode ser observada na figura 17.

Figura 17 Interface com botão liga/desliga



Fonte – Matlab® (2016)

O código para o botão liga/desliga está exposto na figura 18, sendo que antes deste, como já citado, fez-se uso dos códigos mostrados nas figuras 13 e 14, que são comuns a todas as aplicações abordadas na sessão 3.2 deste documento.

Para este caso, foi-se utilizada uma verificação do estado do botão, para o início das medições e, após a mudança para o estado "ligado", no caso da codificação: "Max", inicia-se um laço de repetição que será interrompido assim que o botão for lido com um valor diferente de "Max". O estado do botão pode também ser averiguado através do próprio Matlab®, já que este é citado como "ON" e "OFF" na tela principal.

Figura 18 Código do botão liga/desliga

```
83      % --- Executes on button press in On_Off.
84      function On_Off_Callback(hObject, eventdata, handles)
85      % hObject    handle to On_Off (see GCBO)
86      % eventdata  reserved - to be defined in a future version of MATLAB
87      % handles    structure with handles and user data (see GUIDATA)
88
89      % Hint: get(hObject,'Value') returns toggle state of On_Off
90 -    global a x t n p;
91
92      %Leitura do Estado do botão ON/OFF
93 -    button_On_Off = get(hObject, 'Value');
94
95
96 -    if  (button_On_Off == get(hObject, 'Max'))
97
98 -        display('ON'); %Avisa sobre a ativação do botão ON/OFF
99
100 -       while(button_On_Off == get(hObject, 'Max')) %Repetição para leitura e montagem do gráfico
101
102 -               tic;                                %Função para iniciar contagem do tempo
103
104 -               Pot = a.analogRead(0)*5/1024;       %Leitura da porta A0 do arduino
105 -               x = [x, Pot];                       %Montagem do vetor de leitura
106
107                 %Plotagem
108 -               plot (t(1:n), x(1:n), 'LineWidth', 2);
109
110                 %Título, grid e legendas
111 -               grid on;
112 -               xlabel ('Tempo (s)');
113 -               ylabel ('Voltagem (V)');
114 -               title ('Experimento com Potenciômetro');
115
116                 %Releitura do botão ON/OFF
117 -               button_On_Off = get(hObject, 'Value');
118
119 -               t = [t, p];                          %Montagem do vetor de tempo
120 -               n = n + 1;  %Faz a contagem das colunas dos vetores t e x, onde t é o tempo e x a leitura
121 -               pause(0.01);                         %Pausa necessária para o bom funcionamento do Arduino
122 -               p = p + toc;                         %Função para finalizar contagem do tempo
123 -       end
124 -       elseif (button_On_Off == get(hObject, 'Min'))
125 -           display('OFF');                          %Avisa sobre o desligamento do botão ON/OFF
126 -           display(p);                              %Mostra o tempo decorrido ao pausar o programa
127 -   end
```

Fonte – Matlab® (2016)

O botão "Salvar" foi implementado de forma que uma segunda resposta do usuário é pedida através de uma caixa de diálogo. Isto foi feito simplesmente para evitar que os dados antigos sejam substituídos no caso do acionamento errôneo do botão, por descuido do usuário. O código segue na figura19.

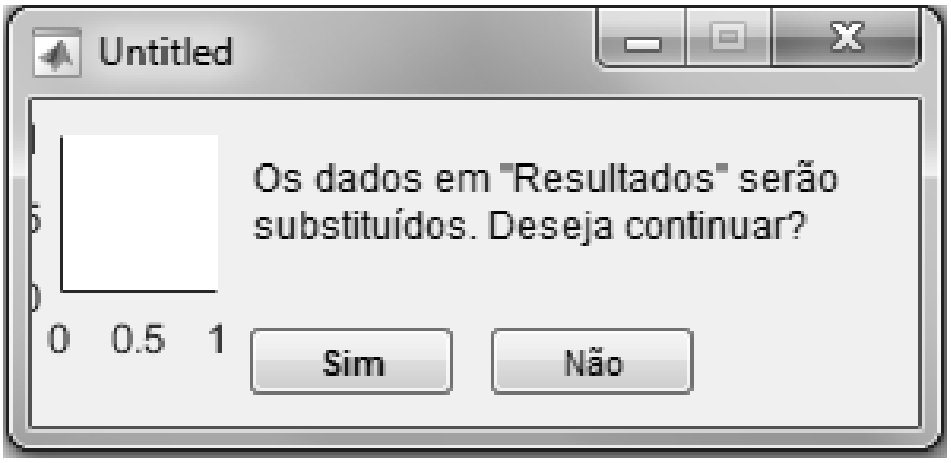
Figura 19 Código do botão "Salvar"

```
130     % --- Executes on button press in Save.
131     function Save_Callback(hObject, eventdata, handles)
132     % hObject    handle to Save (see GCBO)
133     % eventdata  reserved - to be defined in a future version of MATLAB
134     % handles    structure with handles and user data (see GUIDATA)
135 -   global x t p;
136     % Get the current position from the handles structure
137     % to pass to the modal dialog.
138 -   pos_size = get(handles.figure1,'Position');
139     % Call modaldlg with the argument 'Position'.
140 -   user_response = Dialog_box('Title','Salvando...');
141 -   switch user_response
142 -   case 'No'
143 -   case 'Yes'
144 -       save ('Resultados.mat', 't', 'x', 'p', '-append'); %Salva os resultados em arquivo .mat
145 -   end
```

Fonte – Matlab® (2016)

A caixa de diálogo é criada a partir da posição do objeto criado anteriormente, esta foi denominada "pos_size", e é chamada pelo nome salvo, no caso "Dialog_box". O título da caixa é passado como parâmetro, neste experimento fez-se o uso do nome "Salvando...", assim que a escolha é feita, a resposta é passada para o programa e há a execução do caso escolhido. A GUI tem caixas de diálogos já prontas, restando ao programador apenas a customização da mesma. O exemplo do que foi citado se encontra na figura 20.

Figura 20 Caixa de diálogo padrão com textos alterados



Fonte – Matlab® (2016)

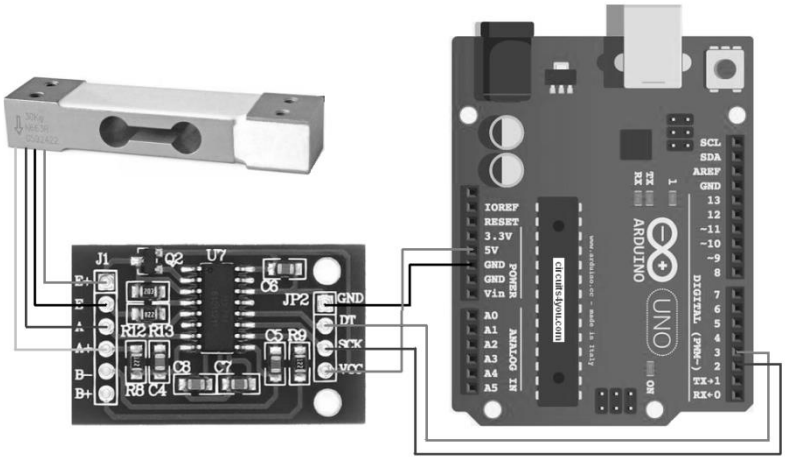
O nome foi exibido como "Untitled", porque a figura mostrada não foi chamada através do código, onde é intitulada. Aqui se mostrou que a caixa pode ser bem genérica e pode ser reutilizada em vários programas, alterando apenas o título.

3.3. Célula de carga

3.3.1. Montagem do sistema

Como tratado na sessão 2.2.3 deste documento, a montagem do circuito foi realizada com a célula de carga, o Arduino Uno e um amplificador de sinal do modelo HX 711. A configuração final do circuito foi representada pela figura 21.

Figura 21 Montagem do circuito para uso da célula de carga



Fonte – circuits4you.com (2016)

Quanto ao cabeamento, à esquerda do amplificador, tomando como referência a figura 21, têm-se as entradas E+ e E-, que fazem parte da alimentação do sensor, e os canais de comunicação A- e A+, que são as portas de dados, as quais possuem um ganho programável de 128 ou 64. Pelo sinal de entrada ser muito pequeno, foi preferível usar um ganho mais elevado, ao invés do ganho fixo de 32, fornecido pelo canal B.

Ao lado direito do amplificador, tem-se o GND, que representa o aterramento, o VCC, que representa a tensão de alimentação, no caso 5V fornecidos pelo próprio Arduino, DT e SCK fazem a comunicação serial entre o amplificador e o microcontrolador.

3.3.2. Programação do Arduino

Para a leitura através do HX 711, a biblioteca "hx711.h" foi necessária. Esta pode ser baixada gratuitamente, pesquisando por este mesmo nome na internet. O local de download se encontra também nas referências deste documento.

O amplificador pode ser conectado ao Arduino por comunicação serial, a qual foi utilizada, sendo DT a porta 3 e SCK a porta 2 da placa. Com isso, pode-se calibrar o sensor de forma simples com o programa da figura 22.

Figura 22 Programa para calibrar e medir utilizando célula de carga

```
1  #include <HX711.h>
2
3  HX711 cell (3, 2); // O primeiro pino é o DT e o segundo, o SCK
4
5  void setup() {
6    Serial.begin(9600);
7  }
8
9  long val = 0;
10 float contador = 0;
11 float tara = 0;
12 float massa_definida = 1;
13 float medida_massa = 1;
14
15 void loop() {
16   //Utilizado para calibrar a medida, fazendo a média das medições
17   contador = contador + 1;
18   val = ((contador - 1)/contador) * val + (1/contador) * cell.read();
19
20   //Colocar o valor obtido pelo Arduino em medição sem carga
21   //tara = ;
22
23   //Pegar uma massa conhecida para calibrar a célula de carga
24   //massa_definida = ;
25
26   //Colocar o valor obtido pelo Arduino em medição com carga
27   //medida_massa = ;
28
29   //Medida real, utilizada somente após a calibragem
30   //val = cell.read();
31
32   Serial.println((val - tara) * massa_definida / medida_massa);
33 }
```

Fonte – Arduino (2017)

O programa mostrado funciona seguindo um passo-a-passo simples. Primeiramente, o programa foi compilado da forma em que se encontra, com isso, após algumas iterações, obteve-se o valor a ser colocado na variável "tara", sendo este a medição realizada sem aplicação de carga no sensor. Com o preenchimento deste campo, pode-se repetir o processo, retirando o comentário da linha 21 e 30, comentando-se as linhas 17 e 18. Assim o usuário pode ver o valor real da medição após esta parte do balanceamento.

Seguindo com a calibragem, voltou-se a comentar a linha 30, usada apenas para verificação do valor real, e retiraram-se os comentários das linhas 17 e 18. Isto serviu para dar à célula de carga seu zero relativo, porém falta ainda dar a escala da carga aplicada. Para isso, colocou-se sobre o sensor uma massa conhecida e, após algumas iterações, anotou-se o novo valor de medição, sendo que estas informações foram gravadas nas variáveis "massa_definida" e "medida"massa", respectivamente.

Com isso a calibragem está completa. As linhas 17 e 18 devem ser comentadas, ou apagadas e as linhas 24, 27 e 30 devem ter seus comentários retirados, após o devido preenchimento das variáveis. Assim, o valor mostrado no monitor serial da IDE do Arduino condiz com o aplicado.

4. RESULTADOS E DISCUSSÕES

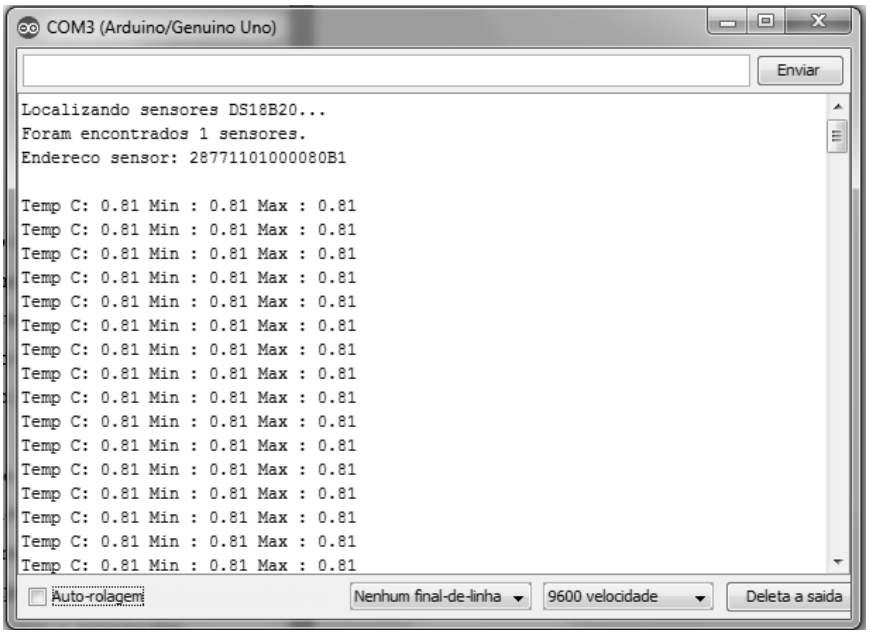
4.1. Sensor de temperatura

Os experimentos realizados para a certificação do funcionamento do programa e do próprio sensor foram os mais básicos. Primeiramente, colocou-se o sensor imerso em um copo com água e gelo. Após o equilíbrio térmico, notou-se que a temperatura mostrada era de 0.50ºC, isso se deve à pressão da cidade de São Carlos, onde se decorreu o experimento. Com isso, comprovamos a eficácia da medição tanto pelo sensor, quanto pelo programa, que foi devidamente calibrado.

Em seguida, ferveu-se água e mediu-se a temperatura, chegando ao valor de 87.25ºC, o que está bem abaixo do esperado, que seria 97ºC, para a cidade, mas se levando em conta as condições do experimento, o valor está razoável. O experimento decorreu-se em uma sala com poucos aparatos que garantissem a integridade das medições, principalmente no que se refere à manutenção da temperatura durante o tempo de equilíbrio, já que o termômetro utilizado não entra em equilíbrio com o meio imediatamente, sendo necessário certo tempo para que isso ocorra. Durante esse período a temperatura da água caiu vertiginosamente, tentando entrar em equilíbrio térmico com o ambiente.

O formato dos dados no monitor serial do Arduino pode ser observado na figura 23.

Figura 23 Monitor serial com valores de temperaturas medidas



Arduino (2017)

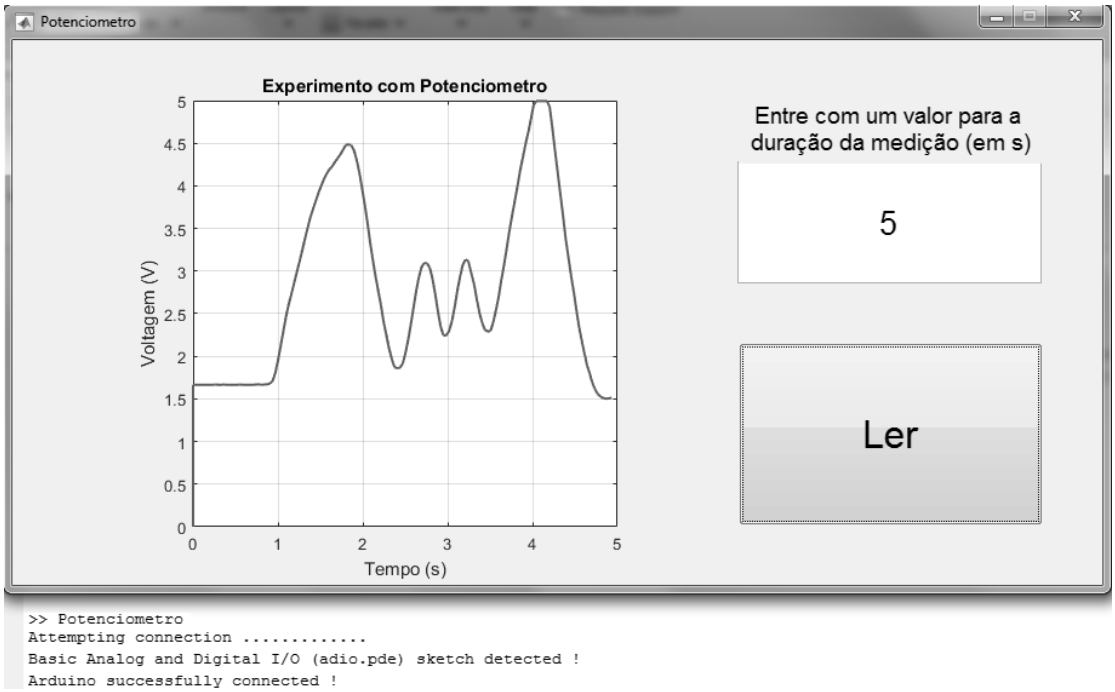
A interface do programa realizado mostra ao usuário a quantidade de sensores ligados em paralelo e seus respectivos endereços. Logo em seguida, imprime a temperatura atual, a mínima e a máxima medidas desde o início do programa. No caso da figura 23, o experimento com o copo com gelo tinha sido reiniciado para que se pudesse mostrar a interface.

4.2. Potenciômetro linear rotativo

4.2.1. Medida com tempo pré-determinado

O experimento com o potenciômetro linear rotativo em conjunto com a GUI do Matlab® tem o resultado exposto na figura 24.

Figura 24 Interface gráfica para a leitura de um potenciômetro



Fonte – Matlab® (2016)

A janela aberta com o nome "Potenciometro" é a figura gerada pela GUI do Matlab®, enquanto que, abaixo dela, tem-se a janela de comando do próprio Matlab® com a chamada do programa, a tentativa de conexão com o Arduino, a verificação da biblioteca de comunicação, anteriormente baixada e carregada para o Arduino através de sua IDE, e a verificação de conexão bem sucedida.

O gráfico é gerado em tempo real, podendo ser verificado durante a rodagem do programa. Várias leituras sucessivas podem ser feitas, sem reiniciar o programa, o que poupa tempo, já que, todas as vezes que o programa é iniciado, o processo de conexão com o Arduino é refeito. O que também ajuda é que pequenas mudanças no código original podem ser feitas sem comprometer o funcionamento da interface, não sendo necessário que esta seja reiniciada a cada constante de proporcionalidade alterada.

Por ser contado originalmente por pulsos, há um erro na casa dos 0.01s na conversão através das funções *tic* e *toc*, para segundos, porém, no formato e objetivos em que os experimentos são baseados, pode ser considerado desprezível, não comprometendo os dados e, por conseguinte, as práticas laboratoriais em que serão utilizados.

Para o tempo de realização do experimento igual a 5s, os vetores gerados para "t" e "x", tempo e voltagem medida, respectivamente, tiveram dimensão 159, ou seja, não pode ser exposta totalmente neste documento, porém, na tabela 1, inserida abaixo, tem-se as medições e seus devidos momentos no tempo, além de sua posição no vetor (pulso em que ocorre) para os pontos próximos aos 3s, onde a variação é mais visível.

Tabela 1 Vetores de tempo e voltagem medida obtidos através do experimento

Pulso	90	91	92	93	94	95	96	97	98
t	2.7884	2.8203	2.8523	2.8833	2.9158	2.9473	2.9801	3.0165	3.0523
x	3.0420	2.9150	2.7344	2.5391	2.3535	2.2510	2.2412	2.2949	2.4219

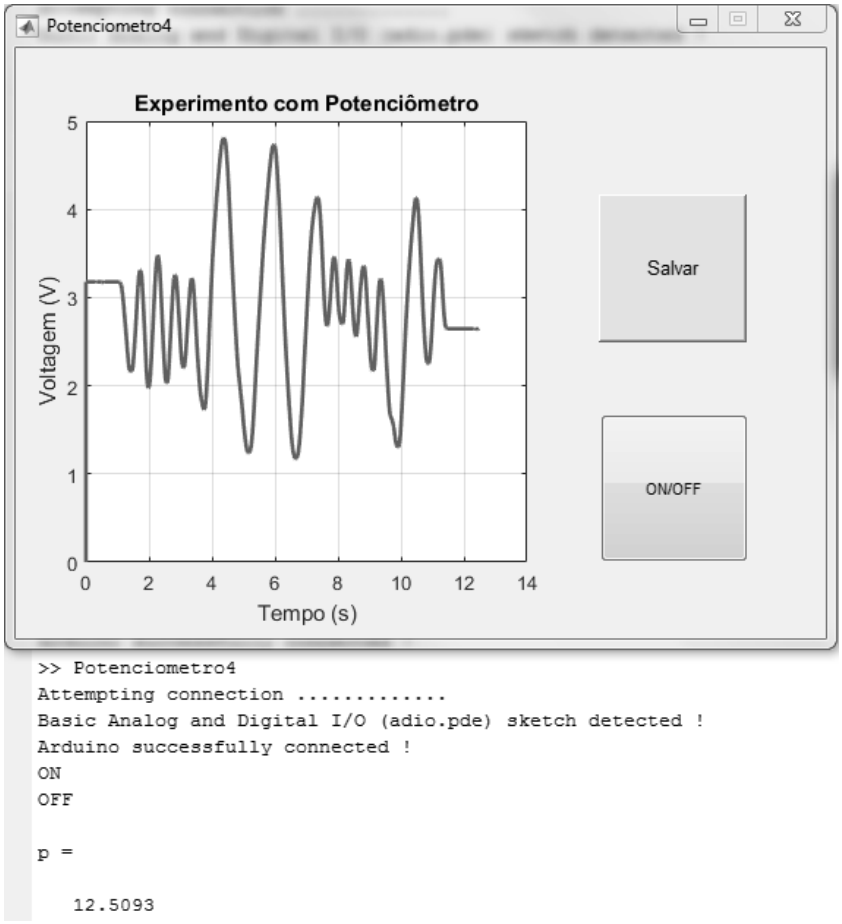
Fonte – Matlab® (2016)

Devido à forma como foi programada, a reutilização sem reinício é possível, porém haverá a perda dos dados anteriormente salvou. Uma opção válida para resolver este problema seria uma caixa de diálogo, como a criada na sessão 3.2.4, para informar ao usuário sobre esta possível perda. Outra solução seria a criação do botão "Salvar", exatamente com foi feito também na sessão 2.3.4, com isso evita-se o salvamento desnecessário de dados ou a perda destes.

4.2.2. Medida com botão liga/desliga

O experimento com o novo programa, desta vez com o botão liga/desliga, seguiu os moldes do anterior, sendo que os resultados podem ser vistos na figura 25.

Figura 25 Experimento dirigido ao programa com botão liga/desliga



Fonte – Matlab® (2016)

Desta vez aplicou-se uma série de mudanças bruscas no sentido de rotação do potenciômetro, com isso os resultados expostos na tabela 2 apresentam uma variação muito maior que a mostrada na tabela 1, levando em conta as mesmas variáveis. Sob a janela "Potenciometro4", segue parte da janela de comandos do Matlab®, contendo o mesmo processo de conexão descrito na sessão 4.2.2, seguido do "ON", indicando o início das medições e o "OFF", indicando seu término. A variável "p" impressa mostra o tempo decorrido durante o experimento, sendo que este é mostrado em segundos.

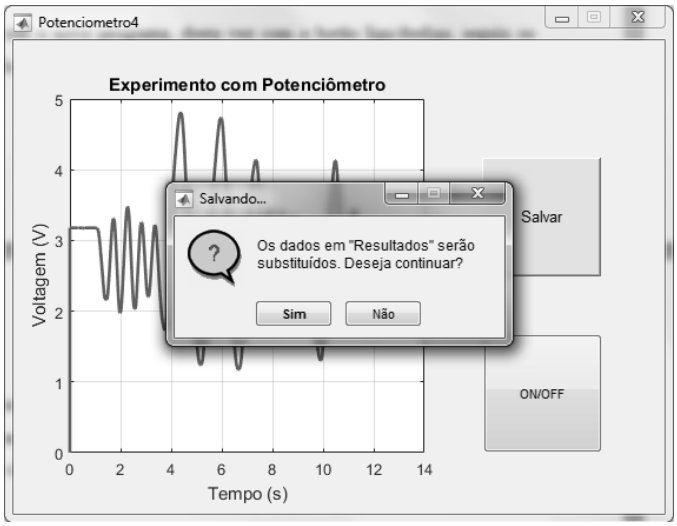
Tabela 2 Vetores de tempo e voltagem medida obtidos através deste experimento

Pulso	250	251	252	253	254	255	256	257	258
t	7.5826	7.6134	7.6422	7.6751	7.7039	7.7348	7.7637	7.7946	7.8235
x	2.9932	3.1738	3.3447	3.5107	3.6279	3.7109	3.7354	3.7354	3.7305

Fonte – Matlab® (2016)

Os dados da tabela 2 foram salvos no arquivo "Resultados.mat", após o requerimento da ação através do acionamento do botão "Salvar" e sua posterior confirmação na caixa de diálogo chamada dentro do programa e representada pela figura 26.

Figura 26 Caixa de diálogo para salvar os valores



Fonte – Matlab® (2016)

A gravação dos dados pode ocorrer durante a execução das medições, porém quando a caixa de diálogo é chamada, a leitura para de ser realizada até que uma ação seja escolhida. Ao final, a leitura é retomada normalmente, do ponto em que parou. Não há espaçamentos no gráfico, nem falhas nas variáveis.

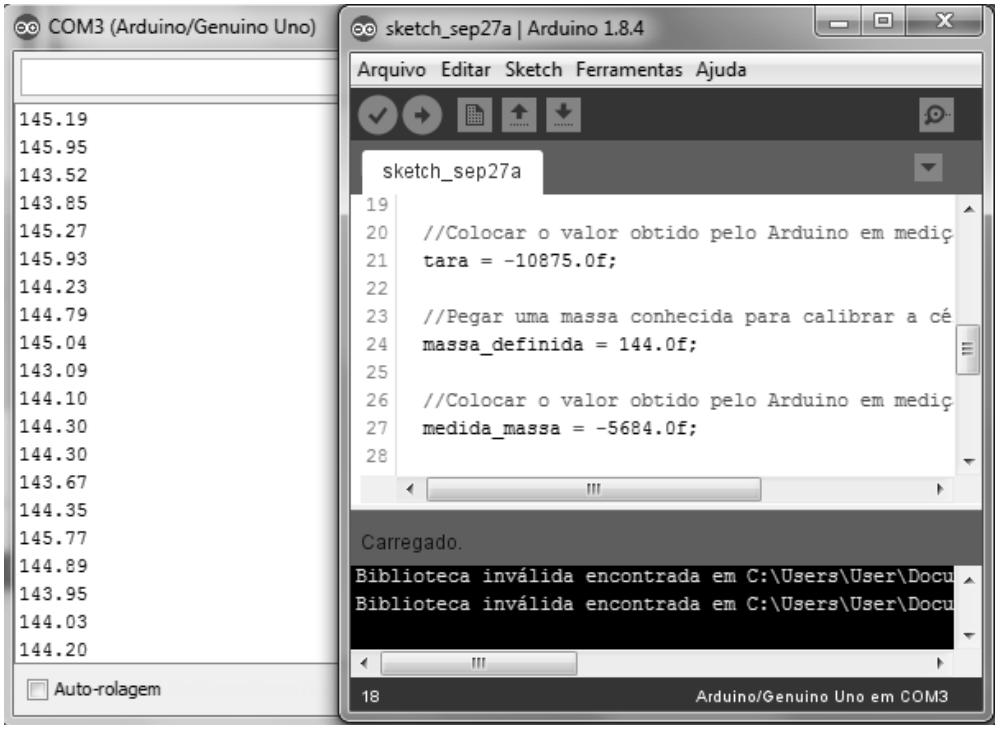
Nas tabelas 1 e 2, quando há a verificação do intervalo dos tempos da amostra exposta, pode-se notar uma irregularidade, isto se deve pelo fato das funções *tic* e *toc* dependerem do processamento tanto do Arduino, quanto do computador que executa o Matlab®, porém, para o nível de precisão das práticas e questão, esta irregularidade de intervalos não altera os resultados esperados nas práticas de laboratório.

4.3. Célula de carga

O programa utilizado foi simples e muito eficaz. Após a calibragem, os valores obtidos ainda flutuam muito, como pode ser observado na figura 27, a qual também mostra os dados utilizados, dentre eles a massa padrão de 144g, além dos valores obtidos pelas iterações.

As condições do experimento não eram as ideais. A célula de carga não tinha apoio fixo, o que dificultava até mesmo a colocação da massa sobre ela, além disso, a falta de equilíbrio causa alterações na medição. Isto pode ser uma das causas da flutuação dos valores obtidos.

Figura 27 Medições e variáveis obtidas com a célula de carga



Fonte – Arduino (2017)

Este experimento está sujeito a alterações devido às condições climáticas. Pelo sensor ser muito sensível e o amplificador gerar um grande número de bits, os dados tendem flutuar um pouco, dependendo da temperatura, por exemplo.

Durante a inserção das variáveis, fez-se uso da expressão ".0f", que indica que a constante inserida é do tipo *float*, caso este recurso não seja utilizado, o programa trata os números de forma diferente, provocando também erros nas medidas.

Durante o tempo em que se trabalhou com a célula de carga, houve diversas tentativas frustradas para a implementação da sua leitura em tempo real, como realizada com o potenciômetro, através do Matlab®, com uma interface simpática ao usuário e que pudesse dar mais liberdade nas formas de obter os dados e gravá-los. Porém as frustrações tiveram início já na comunicação do Arduino. O amplificador não manda os dados diretamente, ele possui um *clock*, para que o mesmo pudesse ser operado, fez-se uso do protocolo I2C, como indicado pela MathWorks®, porém o Matlab® mesmo assim não pôde se comunicar com o

amplificador por meio do Arduino e, por conseguinte, não pôde localizar o sensor, impossibilitando assim sua utilização através deste método.

5. CONCLUSÃO

Com isso, pôde-se concluir que a aplicação de aulas práticas com interfaces gráficas podem fornecer conhecimentos, tanto na área de programação, quanto na área de obtenção de dados e conhecimento de sensores.

A experiência obtida durante a graduação pode ser um diferencial para a conquista de um emprego, em meio a tantos recém-graduados. Não se pode mais manter-se apenas com a teoria e os livros, as práticas de laboratório podem ser de grande ajuda para os estudantes. Como observado nesta tese, a implantação destas práticas tem um preço acessível, exige conhecimentos que, quase em sua totalidade, já são assimilados no programa atual de graduação e podem fornecer grande experiência na área de controle, que, com a implantação das indústrias 4.0, tem se tornado um setor muito atrativo.

Através dos experimentos realizados, pôde-se concluir também que a GUI do Matlab®, juntamente com um potenciômetro linear rotativo pode ter muitas aplicações nos laboratórios de medidas, por ser um sistema resistente e por não sofrer grande interferência do ambiente.

Os experimentos envolvendo os sensores de temperatura também são facilmente aplicáveis. A calibragem pode ser feita através da água em ebulição e o gelo, atentando-se sempre à altitude em que o experimento é realizado, para diminuir os erros nas medidas, ou com os programas já expostos nesta tese, os quais calibram os sensores diretamente, podendo fazer uso dos mesmos para experimentos mais desafiadores.

Dentre os elementos escolhidos para esta tese, a célula de carga é o que teria mais complicações para aplicações em laboratório de ensino, por exigir uma preparação um pouco melhor do ambiente, além do cuidado com o sensor, que é muito sensível.

REFERÊNCIAS

BALBINOT, A.; BRUSAMARELLO, V., J. **Instrumentação e fundamentos de medidas, volume 1.** Rio de Janeiro: LTC, 2010.

BALBINOT, A.; BRUSAMARELLO, V., J. **Instrumentação e fundamentos de medidas, volume 2.** Rio de Janeiro: LTC, 2007.

CAMPA, G. **Legacy MATLAB and Simulink Support for Arduino.** Abril, 2016. Disponível em <https://www.mathworks.com/matlabcentral/fileexchange/32374-legacy-matlab-and-simulink-support-for-arduino>. Acesso em: 20 Set. 2017.

THAKUR, M., R. **HX711 load cell amplifier**: weighing scale design. Nov. 2016. Disponível em: < http://circuits4you.com/2016/11/25/hx711-arduino-load-cell/>. Acesso em 25 Out. 2017.

THE MATHWORKS INC. **Matlab®: Creating Graphical User Interfaces.** Estados Unidos: The MathWorks Inc., 2015.

Apêndice A – Prática com sensor de temperatura

Para a prática em questão, faz-se uso de:

- 2 Sensores de temperatura do modelo DS18B20;
- Arduino Uno;
- Um resistor de 4,7 kΩ;
- Recipiente metálico;
- Resistência elétrica.

Para iniciar a prática, sugere-se realizar uma familiarização dos alunos com o programa utilizado, com o sensor e com a velocidade de resposta do sensor. Para isso, pode-se aplicar inicialmente a montagem do sistema, que segue a figura 6 desta tese, e, posteriormente, a medida de temperaturas conhecidas, como água em ebulição e gelo, para verificar se o sensor está calibrado corretamente, assim como observar o tempo que demora para que o sensor entre em equilíbrio térmico com o ambiente em questão.

Em seguida, pode-se aplicar uma prática para que se verifique a possibilidade do uso de dois sensores, para aumento da precisão, em conjunto com um controle de temperatura propriamente dito. Isto pode ser realizado com a montagem de um sistema composto por um recipiente que contenha um banho a ser fiscalizado e uma resistência, para o aquecimento. O controle pode ser realizado através do Arduino Uno, recebendo o sinal dado pelos sensores e mandando uma resposta de controle para que a resistência seja ligada ou desligada, dependendo da temperatura a ser mantida, que será definida pelo usuário. A precisão dos sensores pode garantir que a temperatura seja mantida com um erro muito pequeno.

Apêndice B – Prática com GUI do Matlab®

Para a prática em questão, faz-se uso de:

- Arduino Uno;
- 1 Potenciômetro linear rotativo;
- 1 Motor;
- Sensores de posição variados.

Por se tratar de uma ferramenta pouco conhecida no meio acadêmico atualmente, sugere-se iniciar esta prática com uma introdução à GUI e com um pequeno tutorial para uso das funções básicas e familiarização com o código gerado.

Em seguida, pode-se começar uma apresentação básica sobre aplicações. Para isso, basta utilizar a montagem apresentada na figura 10. Os programas desenvolvidos durante os experimentos apresentados nesta tese também podem servir como uma boa introdução, pela sua simplicidade.

Para aumentar a complexidade, o motor pode ser acoplado ao circuito, de forma a ser controlado pelas portas PWM do Arduino Uno, fazendo uso das leituras do próprio potenciômetro, mas de forma que o movimento possa ser iniciado apenas através da escolha do usuário. Para isso, a função "*check box"* pode ser utilizada, possibilitando desabilitar o motor quando pertinente. A verificação através do gráfico, porém, pode ser mantida sempre, para controle do aluno. O uso de *check boxes* pode servir até mesmo para acionamento de LEDs, indicando quais dispositivos estão ligados.

Com este software, o controle de posição também pode ser possível. Basta a montagem de um sistema composto por um motor que desloca uma superfície rígida que, por sua vez, será usada de suporte para diferentes sensores, controlados individualmente. Um anteparo cambiável, para que diferentes materiais sejam testados com diferentes sensores, mantido em frente à superfície móvel e um Arduino Uno para o controle. A GUI pode fazer o papel de criação de cenários de teste, possibilitando a escolha do sensor utilizado e a distância a ser mantida, além de mostrar um gráfico tempo x posicionamento da placa com relação ao anteparo.

Com o desenvolvimento de programas mais complexos e, por conseguinte, aprimoramento dos conhecimentos sobre o software, pretende-se chegar à construção de um

simulador de corrida, com possibilidade dos alunos acompanharem os movimentos de volante e pedais em tempo real, através de gráficos obtidos pela GUI do Matlab®. Para isso ser possível, a junção de células de carga para a leitura da força aplicada nos pedais e um potenciômetro para a leitura da posição do volante fazem-se necessárias.