UNIVERSIDADE DO ESTADO DE SANTA CATARINA

CCT – CENTRO DE CIÊNCIAS TECNOLÓGICAS

DEE- DEPARTAMENTO DE ENGENHARIA ELÉTRICA

ANDRÉ RAMOS MARCINICHEN

INTERFACE VIRTUAL LABVIEW PARA COMUNICAÇÃO COM ARDUINO EM EMBARCAÇÕES

JOINVILLE – SC

2016

ANDRÉ RAMOS MARCINICHEN

INTERFACE VIRTUAL LABVIEW PARA COMUNICAÇÃO COM ARDUINO EM EMBARCAÇÕES

Trabalho de conclusão de curso apresentado ao curso de Engenharia Elétrica do Centro de Ciências Tecnológicas, da Universidade do Estado de Santa Catarina, como requisito parcial para a obtenção do grau de Bacharel em Engenharia Elétrica.

Orientador: Me. Joaquim Rangel Codeço.

Coorientador: Me. Marco Shawn Meireles Machado.

JOINVILLE – SC

2016

ANDRÉ RAMOS MARCINICHEN

INTERFACE VIRTUAL LABVIEW PARA COMUNICAÇÃO COM ARDUINO EM EMBARCAÇÕES

Trabalho de Conclusão de Curso, Departamento de Engenharia Elétrica – DEE, Centro de Ciências Tecnológicas – CCT, Universidade do Estado de Santa Catarina – UDESC, Bacharel.

Banca Examinadora

Orientador: _____

Me. Joaquim Rangel Codeço

UDESC (Universidade do Estado de Santa Catarina)

Coorientador: _____

Me. Marco Shawn Meireles Machado.

UDESC (Universidade do Estado de Santa Catarina)

Membro: _____

Me. Fabrício Noveletto

UDESC (Universidade do Estado de Santa Catarina)

Suplente: _____

Me. Ana Teruko Yokomizo Watanabe

UDESC (Universidade do Estado de Santa Catarina)

JOINVILLE – SC

2016

AGRADECIMENTOS

Agradeço a todos os membros do Wololo, pelas dicas. A minha família pelo aprendizado da vida, principalmente ao meu falecido pai. A minha namorada pelo grande apoio. Aos professores Joaquim e Marcos Shawn, pelas sugestões e colaborações com o TCC.

Ao grupo pet por fornecer shield de aprendizado do Arduino, ao Marcelo com conversor e os sensores de tensão e corrente.

“A interface de usuário é como uma piada.

Se você tem que explicar, ela não é tão boa.”

Martin Leblanc

RESUMO

Marcinichen, André R. **Interface virtual Labview para Comunicação com Arduino em embarcações.** 2016. TCC (Bacharelado em Engenharia Elétrica – Área: Eletrônica e Microprocessadores) – Universidade do Estado de Santa Catarina, Joinville, 2016.

Este trabalho apresenta desenvolvimento de uma interface virtual utilizando o Labview, no qual terá a função de demonstrar os dados coletados de uma embarcação. Para isso é necessário fazer um estudo sobre o Arduino, pois através dele será coletado os dados e transmitido serialmente para Labview, realizar também um estudo sobre o Labview. Outro fator importante a ser estudado é o conhecimentos sobre microcontroladores, como o ATMEGA328-PU que está presente na placa do Arduino. O TCC tem como objetivo desenvolver a interface e verificar sua viabilidade ao coletar esses dados e demonstra-los para o usuário. A interface poderá servir como ferramenta de trabalho, por exemplo, em embarcações. Onde usuário terá condições de verificar a situação da embarcação.

Palavras-chave: Labview, Arduino, Embarcação, Sensores

ABSTRACT

This paper presents the development of a virtual interface using Labview, which will function to show the data collected from a vessel. This requires a study on the Arduino, because through it will collect the data and transmitted serially to Labview also conduct a study on the Labview. Another important factor to be studied is the knowledge of microcontrollers, as ATMEGA328 -PU that is present in Arduino plate. TCC aims to develop the interface and verify its feasibility to collect this data and shows them to the user. The interface can serve as a working tool, for example in boats. Where users will be able to check the status of the vessel.

Key-words: Labview, Arduino, Ships, Sensors.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIMBOLOS

USB	*Universal Serial Bus*
DSP	*Digital Signal Processor*
DSC	*Digital Signal Controller*
FPGA	*Field Programmable Gate Array*
RISC	*Reduced Instruction Set Computer*
USART	*Universal Serial Asynchronous Receiver/Transmitter*
DAQ	*Data Acquisition*
NI	*National Instruments*
PWM	*Pulse Width Modulation*
CPU	*Central Processor Unit*
API	*Application Programming Interface*
RTU	*Remote Terminal Unit*
ASCII	*American Standard Code for Information Interchange*
VI	*Virtual Instrument*
LED	*Light Emitting Diode*
VISA	*Virtual Instrument Software Architecture*
NPEE	Núcleo de Processamento de Energia Elétrica
MCU	*Microcontroller unit*

SUMARIO

1 INTRODUÇÃO

A escolha deste tema deve-se a necessidade de interfaces amigáveis e uteis nos dias de hoje. A utilização de um controle com *software* demonstra maior eficiência devido aos benefícios que proporciona. Desta forma, será criado um programa para ser aplicado em embarcações. Com a criação do mesmo, será possível a visualização dados de sensores e outras informações. Um fator importante a ser considerado é a ocupação de tempo de projeto para as várias etapas, como pesquisa, e implementação.

Um ponto decisivo na hora de iniciar o projeto é a escolha da forma de controle que será utilizada. O controle pode ser implementado de várias formas, alguns dos controladores são:

- Microcontroladores. Exemplo: ATMEGA328-PU.
- Processadores Digitais de Sinais (DSP). Exemplo: TMS320F/C24x.
- Controladores Digitais de Sinais (DSC). Exemplo: TMS320F28335.
- FPGA's. Exemplo: Cyclone III.

Alguns aspectos pertinentes na hora de escolher o controlador são as configurações básicas, como por exemplo o tempo de processamento, a frequência máxima de *clock*, o tipo de memória, a quantidade de entradas e saídas disponíveis, o custo, entre outros fatores. Isso irá depender também do tipo de conversor, ou do investimento a ser feito no projeto.

A linguagem de programação pode influenciar no tempo total de elaboração de um projeto. Além disso, poderá afetar a coleta de dados e informações gerais a serem obtidas, ou ainda em adaptações futuras no projeto. Como existem vários tipos de linguagem, serão mostradas algumas vantagens de utilizar a interface visual do Labview.

O Labview é uma ferramenta que permite programação de controladores na forma de blocos, ele abrange recursos extremamente elaborados, de forma que permite maior facilidade na programação e na elaboração de várias sub-rotinas úteis. Também permite uma visualização mais simplificada do conteúdo em utilizações futuras.

Devido a interface visual que apresenta, torna-se possível a realização de alterações em dados, sem a necessidade de mudança na programação interna, consequentemente é extremamente didática para mostrar mudanças de comportamento das variáveis que serão coletadas.

Tem como problemática o fato de que nem toda embarcação possui alta tecnologia aplicada na verificação e analise de condições da mesma.

A hipótese é de que é possível elaborar uma interface de simples utilização que permite a verificação das condições da embarcação.

Uma justificativa é que nem sempre é elaborada uma interface de alta tecnologia para embarcação, assim permitindo a ocorrência de falhas sem o conhecimento do operador da embarcação. Caso seja implementado é possível verificar com maior rapidez a situação da embarcação e assim fazer uma correção em alta velocidade.

Este TCC tem como objetivo realizar uma análise sobre Labview e o Arduino para aplicar em embarcações. Tanto o Labview quanto o Arduino possuem uma enorme quantidade de informações em suas comunidades assim permitindo fazer uma análise.

O objetivo especifico é desenvolver uma interface simples para um operador, onde ele seja capaz de verificar as condições da embarcação. Também sendo capaz de alterar algumas dessas condições. A metodologia aplicada é empírica e experimental.

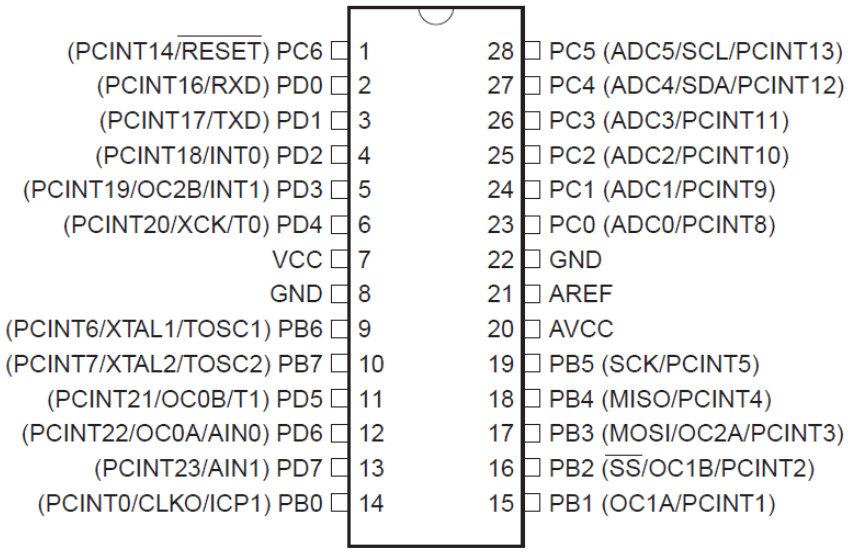
2 ATMEGA328-PU

O microcontrolador é composto por uma combinação de memorias, registradores e outros componentes que permitem o processamento, a recepção e a transmissão de dados.

Existe uma vasta gama de microcontroladores (MCU) com uma quantidade bem variada de características, dependendo muito de cada empresa e projeto escolhido pela mesma. Os MCUS têm uma enorme gama de opções de trabalho, como na automação, iluminação, dispositivos moveis, entre outros.

O ATMEGA328-PU é utilizado na plataforma Arduino. Este microcontrolador possui um custo baixo no mercado, e seu custo é de aproximadamente 10 reais, o que faz com que seja viável para a realização inúmeros projetos. O ATMEGA328-PU será utilizado na leitura dos dados de sensores, que será transmitido para o shield Arduino, que farão comunicação serial em USB. Na Figura 1 é mostrado as pinagens do microcontrolador. Na Figura 2 é mostrado superiormente e conectado ao Arduino.

Figura 1 – Pinos do ATMEGA328-PU



Fonte: ATMEL, 2015

2.1 Características

O ATMEGA328-PU possui:

- Alta performance, de baixa consumo Atmel® AVR® da família de microcontroladores 8-bits.
- Arquitetura RISC avançada:

- 131 instruções poderosas – a maioria de execução em um pulso de clock.
- 32x8 registradores de uso geral.
- Operação totalmente estático (ou seja, em 0Hz).
- Até 20 MIPS através de 20MHz.
- No chip, multiplicação em 2 ciclos.

- Alta permanência não volátil em segmentos de memória.
  - 4/8/16/32KB de Sistema auto programado de programa em memória flash.
  - 256/512/512/1KB EEPROM.
  - 512/1K/1K/2KB interno SRAM
  - Escrever/Apagar ciclos 10.000 Flash e 100.000 EEPROM
  - Retenção de dados por 20 anos a 85º/100 anos a 25º
- Travamento de programa para segurança de software.
- Atmel® QTouch® suporte da biblioteca
- Periféricos:
  - Dois timers de 8 bit com *prescaler* separado e modo comparativo.
  - Um timer de 16 bit com *prescaler* separado, modo comparativo e modo de captura.
  - Contador com tempo real com oscilador separado.
  - Seis canais PWM.
  - Oito canais 10-bit ADC com pacotes TQFP e QFN.
  - Seis canais 10-bit ADC com pacotes PDIP.
  - Comunicação serial USART programável.
  - Mestre/escravo SPI interface serial.
  - *Watchdog* programável.
  - Comparador analógico dentro do chip.
  - Interrupção em troca de pino.
- Recursos especiais
  - Oscilador calibrado internamente
  - Interrupções internas e externas das fontes.
  - Seis modos de *sleep*: inativo, redução de ruído ADC, economia de energia, potência baixa, *standby* e *standby* estendido.

- 23 entradas e saídas programáveis.
- Tensão de operação: 1,8V até 5,5V.
- Temperatura de Funcionamento entre: 40ºC até 85ºC
- Consumo a 1MHz, 1.8V,25ºC
  - Modo ativo: 0.2mA
  - Potência baixa: 0.1µA
  - Modo economia de energia: 0.75 µA
- Grade de Velocidade em faixas de tensão e frequência, indicados na Tabela 1.

Tabela 1 – Tabela de frequência e tensão necessária.

Frequência (MHz)	0 – 4	0-10	0-20
Tensão (V)	1.8-5.5	2.7-5.5	4.5-5.5

Fonte: ATMEL, 2015.

Figura 2 – ATMEGA328-PU

Fonte: Produção do próprio autor

3 ARDUINO

O Arduino é uma plataforma muito utilizada atualmente, devido ao seu baixo custo. Dessa maneira, permite que várias pessoas possam adquirir seu próprio Arduino. Os criadores, ao terem a ideia da plataforma, pensaram em criar uma ferramenta simples que possibilitasse a não engenheiros a criação de projetos digitais. Com um time inicial composto por Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, e David Mellis, foi criada uma comunidade *open-source,* que facilitou a criação de projetos para iniciantes. Existe uma vasta quantidade de informações de projetos e ideias que auxiliam na programação e no aprendizado do Arduino. (Barragan, tradução própria)

3.1 Plataforma

A plataforma é composta por um conjunto de poucos elementos, que permitem a gravação do programa no microcontrolador em conjunto com algumas funções auxiliares. Entre os elementos mais utilizados, que não são simplesmente pinos, temos uma entrada de fonte de alimentação e uma entrada USB, como na Figura 3.

Figura 3 – Entrada USB e alimentação

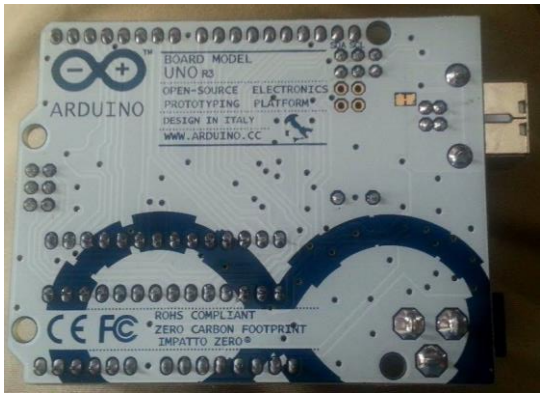
Fonte: Produção do próprio autor

A plataforma abrange 5 entradas analógicas, 13 entradas digitais, sendo que 6 delas são de *Pulse Width Modulation* (PWM). Além disso, 2 portas digitais abrangem a opção de serem utilizadas na comunicação serial. Ele ainda apresenta alguns pinos de tensão, e um *ground* para utilização geral. Como mostrados na Figura 4 e 5.

Figura 4 – Arduino UNO frente

Fonte: Produção do próprio autor

Figura 5 – Arduino UNO verso



Fonte: Produção do próprio autor

Existem vários modelos de *shield* que podem ser acoplados ao Arduino, alguns com *hardware* semelhante ao de um *joystick*, outros para comunicação GSM, por exemplo. Como mostrado na Figura 6.

Figura 6 – *Shield* GSM

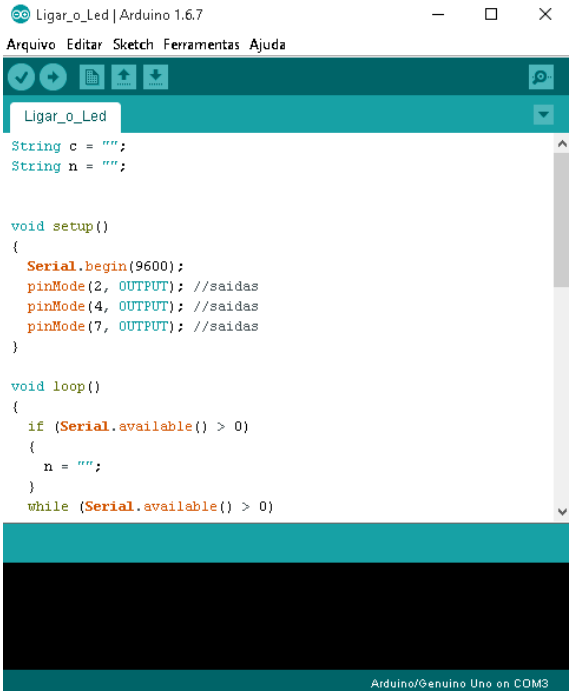
Fonte: http://ubidots.com/docs/_images/GSM_Shield.jpg

3.2 Estrutura básica

3.2.1 Ambiente de desenvolvimento.

O ambiente de programação do Arduino é o mostrado na figura 7. Ao conectar o USB do Arduino no computador, é preciso ir ao menu ferramentas e selecionar a placa referente a plataforma que está sendo utilizado. No decorrer desse TCC será utilizado o "Arduino/Genuino Uno". Posteriormente, é necessário verificar a porta serial utilizada, definida por COM X, onde x varia de acordo com a porta USB utilizada no computador. No TCC será utilizada a COM3.

Figura 7 – Ambiente Arduino



Fonte: Produção do próprio autor

Em geral, primeiramente são definidas as bibliotecas, depois disso, são declaradas as variáveis e as constantes. O código deve ser inicializado no "void setup()", onde são colocados os códigos que serão rodados uma única vez. No "void loop()" é adicionado o código que será rodado em *loops* contínuos no Arduino.

3.2.2 Variáveis

Uma variável é um objeto que serve para guardar e representar um valor, ou em expressão como no caso de *strings.* Elas comumente são classificadas como indicada na Tabela 2.

Tabela 2 – Tabela de variáveis e faixa de bits.

Tipo	Tamanho em Bytes	Faixa Mínima
char	1	-127 a 127
unsigned char	1	0 a 255
signed char	1	-127 a 127
int	4	-2.147.483.648 a 2.147.483.647
unsigned int	4	0 a 4.294.967.295
signed int	4	-2.147.483.648 a 2.147.483.647
short int	2	-32.768 a 32.767
unsigned short int	2	0 a 65.535
signed short int	2	-32.768 a 32.767
long int	4	-2.147.483.648 a 2.147.483.647
signed long int	4	-2.147.483.648 a 2.147.483.647
unsigned long int	4	0 a 4.294.967.295
float	4	Seis dígitos de precisão
double	8	Dez dígitos de precisão
long double	10	Dez dígitos de precisão

Fonte: Cruz, 1997

Uma variável de grande importância nesse TCC é variável do tipo *string*, que é um arranjo de caracteres. A comunicação será feita com a transferência dos caracteres, que serão armazenados em variáveis do tipo *string*, onde serão fragmentados no decorrer do programa.

3.3 Comandos utilizados de C

3.3.1 If()

É o comando mais utilizado, pois torna possível verificar se uma condição é verdadeira ou falsa. Traduzindo *if* seria “se”, simplificando o entendimento.

3.3.2 While()

Serve como um *loop,* fazer evento ocorrer várias vezes, dependendo do que for definido nos parênteses. A tradução de *while* se “enquanto”, ou seja, enquanto tal *flag* não for ativada o evento continuará ocorrendo.

3.3.3 For()

Parecido com a função *while*, a função *for* depende de uma variável que vai crescendo até ficar igual a um valor definido. Traduzindo *for* é “para”,

3.3.4 Break

Esta função serve para quebrar um loop, geralmente é utilizada quando determinada condição se torne verdadeira. Traduzindo é “quebra”.

3.3.5 String(x)

Com esta função é possível transformar a variável “x” que pode ser do tipo char, em uma variável do tipo String.

3.4 Comandos utilizados de Arduino

3.4.1 Delay

Esta função faz com que o programa espere por uma determinada quantidade de tempo. Traduzindo significa “atraso”.

3.4.2 Serial.begin(x)

A leitura de dados através da comunicação serial pode iniciar. O valor x se define o *baud rate* que normalmente é definido em 9600.

3.4.3 Serial.read()

Através desse comando é possível ler os bytes que estão sendo recebidas na porta USB, e também converter para string.

3.4.4 Serial.write()

É enviado para a porta USB em forma de bytes as strings.

3.4.5 X.indexof(“y”)

Através do comando “indexof” é possível procurar por um determinado texto ou conjunto de palavras que é representado pela string “y” dentro da variável “x”. Esse comando retorna uma variável “z”, contendo o a string desejada.

3.4.6 X.charAt(‘z’)

Verifica um caractere específico da string X.

3.4.7 DigitalRead(x)

Com esse comando é possível ler os dados do pino x. Os dados de retorno são digitais logo é 1 ou 0. Normalmente feito por botões, pela configuração de hardware pode ser definido como quer a atuação do botão. Por exemplo o botão pode estar normalmente em 1 então quando pressionado retorna zero para o Arduino

3.4.8 DigitalWrite(x)

Utilizando essa função permite a escrita de dados no pino x. Esses dados são digitais, ou seja, é zero ou um. Possui inúmeras aplicações como ativar ou desativar, um

motor ou habilitar determinado aparelho. A aplicação mais simples pode ser através de um LED, que ao varia o valor escrita irá variar em ligado e desligado.

3.4.9 AnalogRead(x)

Esse comando permite leitura de dados analógicos na porta x, como por exemplo dados vindo de sensores ou potenciômetros. Esses dados vão ser lidos em forma de bits que variam de 0 a 1023 ou seja 12 bits. Assim dessa forma é necessário ajustar os dados para serem mostrados para exibição. Como por exemplo de 0 a 5v. Dessa forma é necessário uso do comando map.

3.4.10 AnalogWrite(x)

Aqui é possível enviar dados analógicos para a porta x, diferentemente da por digital que é escrito zero ou um. Aqui é possível ter uma variação maior nos dígitos de dados a serem transmitidos. Os dados são enviados através de 12 bits. Então possui 4 dígitos de precisão.

3.4.11 Map(t,x,y,z,w)

Esse comando é semelhante a uma regra de 3. Ele coleta o valor da variável t, que possui *range* entre x e y, e muda seu *range* para z a w, e assim retorna a variável com o range desejado.

4 LABVIEW

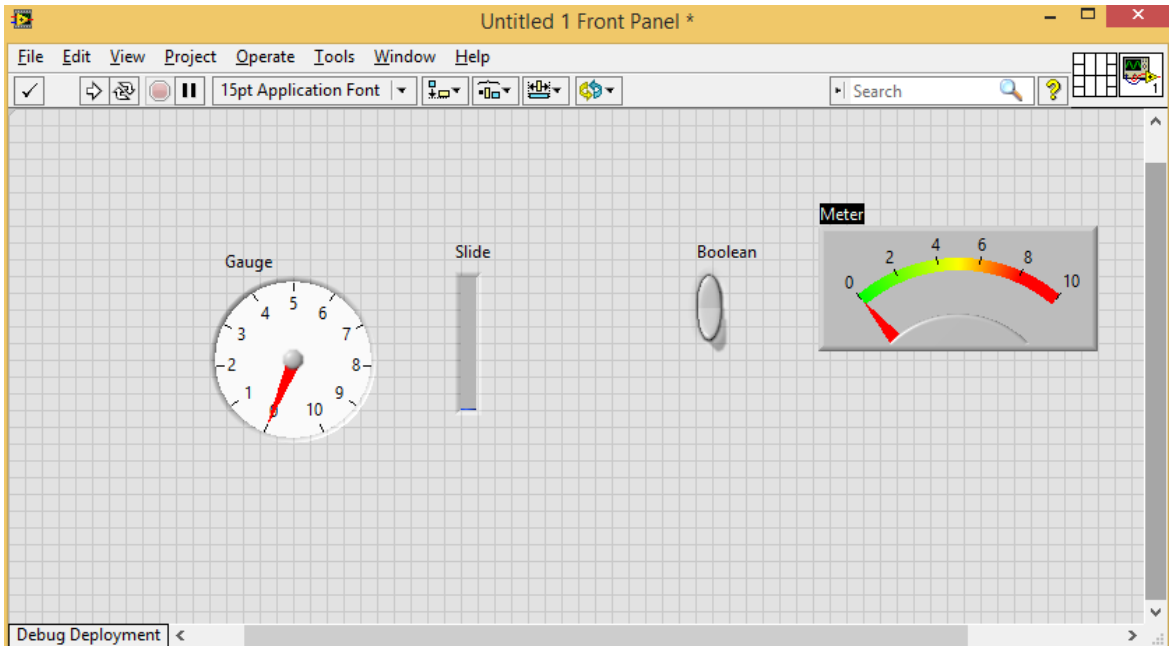
O Labview foi criado pela *National Instruments* e é um ambiente gráfico de desenvolvimento de sistemas. Seu desenvolvimento baseado em plataforma para aplicações em engenharia e ciências, como objetivo de acelerar a produtividade de engenheiros e cientistas, devido a sua sintaxe de programação gráfica.

Utilizando Labview torna simples visualizar, criar e codificar sistemas de engenharia, assim permitindo fazer teste e verificar dados coletados, desta forma possuindo uma interface pratica.

4.1 Interfaces

O Labview possui duas interfaces principais. Este primeiro painel é chamado de painel frontal. Nele é mostrado os dados de uma maneira visual e interativa para o usuário que for utilizar o painel de comandos. Mostrado na Figura 8.

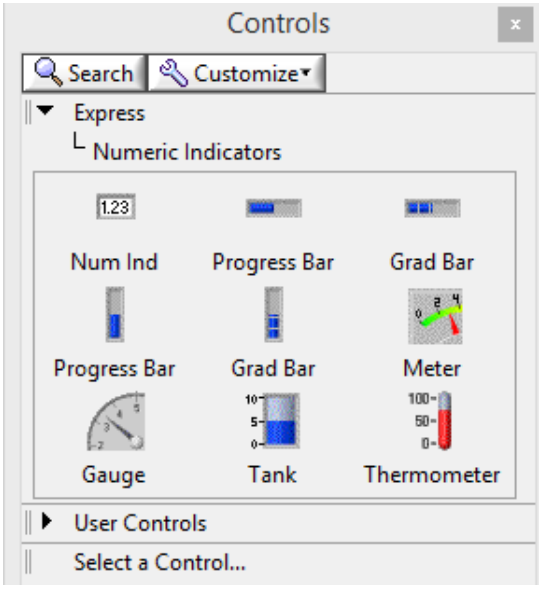
Figura 8 – Tela do Labview



Fonte: Produção do próprio autor

É possível ver na Figura 8, que possui algumas formas de medição, e botões que permitem mudanças, para a coleta de dados. Também é possível o envio de informações com botões, ou outros menus de escolha para que o usuário tenha várias opções na hora de utilizar a interface. Para que seja colocado esses botões é necessário menu de botões mostrados na Figura 9.

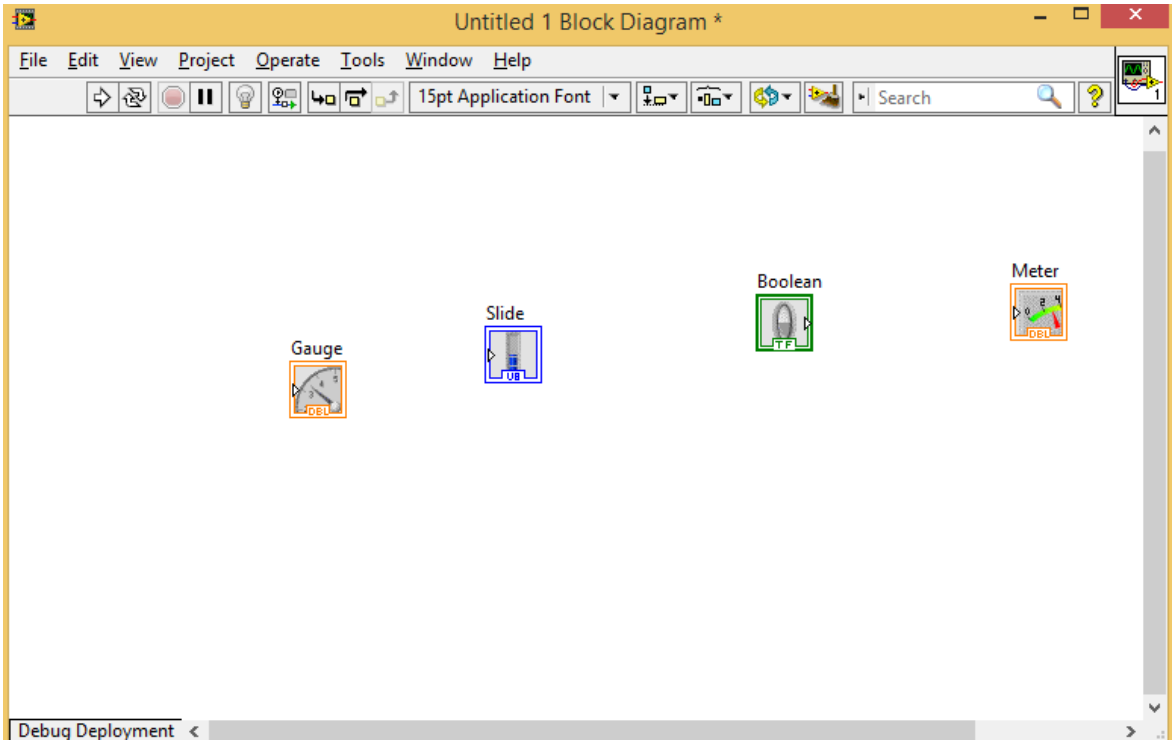
Figura 9 – Menu do Labview



Fonte: Produção do próprio autor

A outra tela de interface é onde a programação em blocos será elaborada. Como na Figura 10.

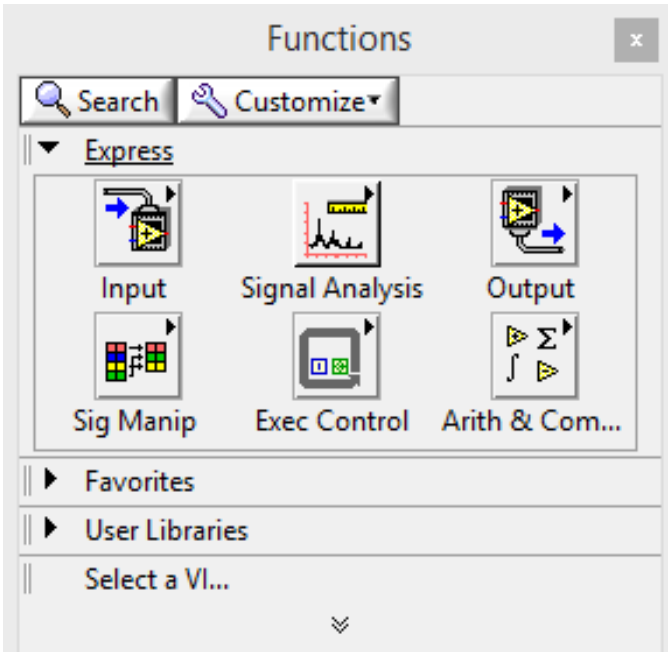
Figura 10 – Tela do Labview no modo blocos



Fonte: Produção do próprio autor

Para poder colocar os blocos, é necessário outro menu. Mostrado na Figura 11.

Figura 11 – Menu do Labview no modo blocos



Fonte: Produção do próprio autor

4.2 VI

Nos menus mostrados nas Figuras 9 e 11, é possível ver os *Virtual Instruments* (VI's) que podem ser utilizados nos programas. Os VI's da Figura 9 são mais didáticos e mais simples de serem analisados como gráficos, botões e valores.

Os VI da Figura 11, que estão no menu são somente blocos de programação. Cada um deles possui suas características próprias, como entradas ou saídas, por exemplo. Existem inúmeros com funções diferentes, como por exemplo, contas matemáticas ou verificações booleanas. Eles podem, inclusive, abranger arquivos que podem ser exportados em forma de texto ou planilhas.

Como existem vários que são próprios do Labview, e tem vários criados na comunidade, será necessário explicar aqueles que serão utilizados no decorrer do TCC.

4.2.1 Dados

Existem alguns tipos de dados que podem ser trabalhados conforme a situação. Nesse tópico serão comentados alguns.

4.2.1.1 Tipo string

Consiste de uma sequência de caracteres. Eles seguem o padrão da ASCII. Esse tipo vai ser muito utilizado na comunicação serial. Esse tipo de dado é mostrado na cor rosa. Como na Figura 12.

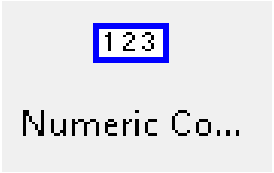
Figura 12 – Constante inteira

Fonte: Produção do próprio autor

4.2.1.2 Tipo numérico

Existem 2 tipos. O primeiro, que está representado em azul, como na Figura 13, é dos números inteiros. Geralmente usado em simples aplicações.

Figura 13 – Dado numérico inteiro



Fonte: Produção do próprio autor

O segundo tipo é representado pela cor laranja, como na Figura 14. Ele pode ser composto por números de ponto flutuante, números de ponto fixo, inteiros, inteiros não sinalizados e números complexos.

Figura 14 – Dado numérico

Fonte: Produção do próprio autor

4.2.1.3 Tipo booleano

Esse tipo de dados consiste de verdadeiro ou falso, 0 ou 1. Utilizado para contas simples de logica. A VI que representa é mostrada na Figura 15, com a cor verde.

Figura 15 – Constantes booleanas

Fonte: Produção do próprio autor

4.2.1.4 Tipo serial

Esses dados serão mostrados em roxo, que na interface de blocos do Labview, serão utilizados pelos VI do visa.

4.2.1.5 Tipo erro

Esses dados são utilizados na verificação de erros em determinados VI's. Eles são demonstrados em amarelo escuro.

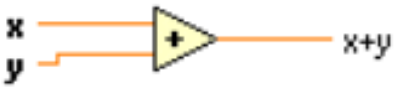
4.2.2 Operações numéricas

Alguns VI aritméticos mais básicos serão demonstrados.

4.2.2.1 Soma

Esse VI serve para fazer a soma da variável x, com a variável y. Ou seja, resulta em x+y. Como mostrado na Figura 16.

Figura 16 – VI da soma.

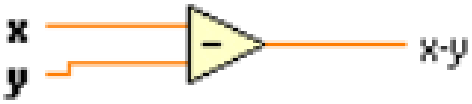


Fonte: Ajuda do Labview

4.2.2.2 Subtração

Seguindo mesmo princípio do VI de soma, neste caso ocorrerá a subtração dos dados de x com os de y. Desta forma sendo visto na Figura 17.

Figura 17 – VI da subtração.



Fonte: Ajuda do Labview

4.2.2.3 Multiplicação

Este VI vai receber os dados de x e y. Retornando assim a multiplicação destes. Como mostrado na Figura 18.

Figura 18 – VI da multiplicação

Fonte: Ajuda do Labview

4.2.2.4 Divisão

Este VI diferentemente dos outros VI mostrados, este pode retornar 2 valores, ao realizar a divisão dos dados x e y, ela retorna quociente e o resto. Como mostrado na Figura 19.

Figura 19 – VI da divisão.

Fonte: Ajuda do Labview

4.2.2.5 Maior

Este VI, como mostrado na Figura 20, compara valor x e verifica se ele é maior que o y. Se isso for verdade, ele retorna o dado booleano *true*. Se não for verdade ele retorna *false*.

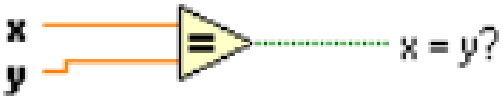
Figura 20 – VI maior.

Fonte: Ajuda do Labview

4.2.2.6 Igual

Este VI pode usado para qualquer comparação de dados, tanto numérico, quanto booleano ou até de strings. Ele retorna os dados booleanos *true* ou *false*. Como mostrado Figura 21.

Figura 21 – VI comparação



Fonte: Ajuda do Labview

4.2.2.7 Inversor

Este VI simplesmente pega o dado e eleva ao expoente -1. Como mostrado Figura 22.

Figura 22 – VI inversor

Fonte: Ajuda do Labview

4.2.2.8 Incremento

Este VI, mostrado na Figura 23, serve para incrementar em 1 o valor anterior. Normalmente é utilizado quando possui algum tipo de loop, ou quando deseja-se mudar a condição inicial zero. Ele torna-se menor em termos de espaço, do que usar VI de soma com mais constante numérica 1.

Figura 23 – VI incremento

Fonte: Ajuda do Labview

4.2.3 Operações booleanas

O Labview abrange algumas funções booleanas internamente, mas no decorrer deste TCC utilizou-se apenas a porta logica *and.* Os outros resultados lógicos obtidos foram usados em estruturas de case, que serão explicados mais à frente do TCC. Como mostrado Figura 24.

Figura 24 – VI porta logica *and*

Fonte: Ajuda do Labview

4.2.4 Estruturas

Em geral as estruturas são usadas para loops, com objetivo de manter determinadas configurações, determinada quantidade de iterações ou até certo requisito se tornar verdadeiro.

4.2.4.1 *While loop*

Essa estrutura só será interrompida quando uma ação *false* for enviada na entrada do botão vermelho mostrado na Figura 25. O "i" em azul envia o número de loops que o *while* está ativo.

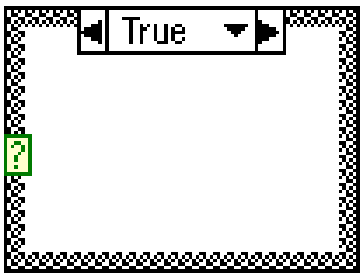
Figura 25 – VI *while loop*

Fonte: Ajuda do Labview

4.2.4.2 Case loop

A estrutura mais utilizada, dentre todas as que o Labview possui, foi a estrutura case. Ela verifica a condição de entrada, que pode ser tanto numérica, quanto booleana ou do tipo String. Como visto na Figura 26, é possível ver uma interrogação onde será imposta a entrada. Onde está escrito *true* é definido os casos. Por exemplo se a entrada for booleana terá 2 casos: *true* ou *false.* Cada um deles rodará funções diferentes.

Figura 26 – VI *case loop*



Fonte: Ajuda do Labview

4.2.4.3 For *loop*

Essa estrutura será responsável por realizar um número finito de iterações, de acordo com o valor definido na entrada, simbolizado por um N na Figura 27. O i informa a iteração atual que está rodando o *loop.*

Figura 27 – VI *for loop*

Fonte: Ajuda do Labview

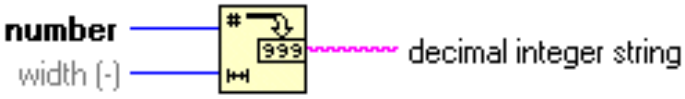
4.2.5 Strings

Nesse tópico serão abordados os VI's utilizados para trabalhar com dados do tipo string.

4.2.5.1 Conversão decimal para string

Como mostrado na Figura 28, é possível visualizar que esse VI possui 2 entradas, Na primeira entrada é inserido o valor decimal que deseja-se converter, já na segunda é definido o tamanho desse número.

Figura 28 – VI para conversão decimal para string

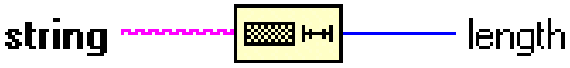


Fonte: Ajuda do Labview

4.2.5.2 Tamanho da string

Essa função conta a quantidade de caracteres dentro dos dados da string. Ela retorna um valor, que é um dado do tipo numérico. O VI é mostrado na Figura 29.

Figura 29 – VI para tamanho da string

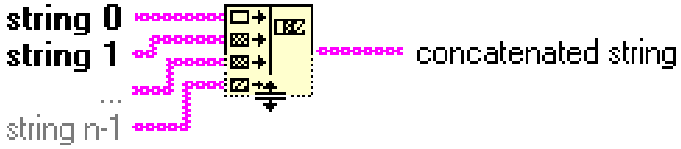


Fonte: Ajuda do Labview

4.2.5.3 Concatenar strings

Com o VI, mostrado na Figura 30, é possível juntar a quantidade de strings necessárias, de forma a fazer uma combinação de strings para enviar serialmente.

Figura 30 – VI para concatenar strings

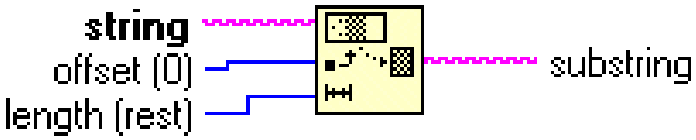


Fonte: Ajuda do Labview

4.2.5.4 Substring

Com esse VI, entra-se com os dados de string. A partir disso é possível retirar uma substring. A localização dessa string é indicada no *offset* e o tamanho dessa substring é definido na entrada *length* Caso seja somente os primeiros dados não é necessário conectar a porta *offset*. Como mostrado na Figura 31.

Figura 31 – VI para gerar subtring



Fonte: Ajuda do Labview

4.2.5.5 String para valor

Com esse VI é possível fazer modificação de uma string conforme for definido na entrada do *format* string, as definições serão mostradas na tabela 3. Como mostrado na Figura 32, tem 2 saídas, a saída superior é do tipo string seguindo conforme definido. Na segunda saída obtém-se o valor do tipo numérico.

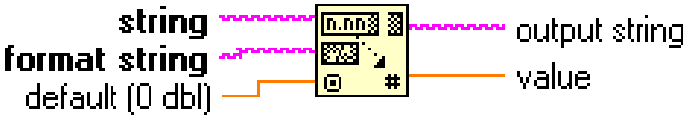
Tabela 3 – Dados para definição de valor.

Tipo	Descrição
%g	Modo automático ele vai escolher entre modo cientifico ou ponto flutuante dependo do numero
%d	Define como decimal inteiro.
%f	Define como ponto flutuante
%e	Notação cientifica
%p	Notação cientifica do SI
%x	Hexadecimal
%o	Octal

%b	Binário
%t	Tempo relativo
%T	Tempo absoluto
%s	String
%$	Define a ordem das variáveis

Fonte: Adaptado do *help do Labview*

Figura 32 – VI que muda string para valor



Fonte: Ajuda do Labview

4.2.6 Comunicação

Neste tópico abordarei os VI responsáveis pela comunicação serial, que farão a configuração do Labview com o computador, e em sequência enviarão para a porta serial os dados do tipo string, que serão recebidos no Arduino.

Primeiramente é necessário comentar sobre o NI VISA, que é uma arquitetura de software que permite a comunicação serial. O NI VISA pode ser usado para maioria dos *buses* USB, serial, Ethernet. (National Instruments,2009)

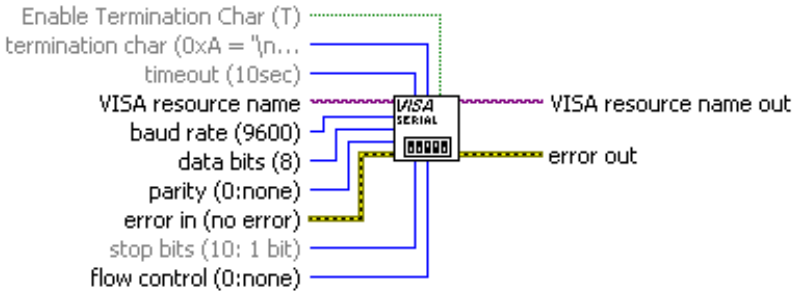
NI-VISA não vem em conjunto com Labview, então é necessário fazer download no site para que os VI's a seguir venham a funcionar corretamente.

4.2.6.1 Configuração de porta serial

Entre todos os VI abordados durante o TCC, este provavelmente é o de maior importância, pois através dele é possível fazer a comunicação serial se ele for devidamente configurado. Na Figura 33, é possível ver as configurações padrões do VI, quando não estão conectados. De todas as entradas mostradas, a única que necessita de um controlador com o painel frontal é a entrada VISA *resouce name*. Nas outras é possível adotar o valor padrão.

Este VI tem 2 saídas. Na primeira, identificada em roxo, estão localizados os dados configurados, que serão enviados para os outros VISA. A segunda, em amarelo escuro indica a ocorrência de algum tipo de erro.

Figura 33 – VI de configuração serial



Fonte: Ajuda do Labview

4.2.6.2 Escrita de dados

Esse VI possui 3 entradas. A primeira, que irá receber as configurações seriais, caracterizada pela cor roxa. A segunda, que recebe os dados de string localizados no padrão ASCII, e que serão comunicados com as configurações definidas. Por fim, a terceira entrada, que recebe e transmite os dados de erro. O VI ainda abrange 3 saídas. Na primeira estão os dados configurados anteriormente para o próximo VI de comunicação serial. A segunda irá indicar a quantidade de caracteres que foram enviados através da comunicação serial. Finalmente, a 3ª saída apresenta o erro, como mostrado na Figura 34.

Figura 34 – VI que envia os dados na porta serial

Fonte: Ajuda do Labview

4.2.6.3 Leitura de dados

Esse VI é semelhante ao anterior, mas ao invés de ter função de enviar dados, ele possui a função de receber dados. Ele conta com uma entrada de dados do tipo numérica, onde é especificada a quantidade de bytes a serem lidos, desta forma, a saída *read buffer* terá a quantidade de caracteres lidos no formato ASCII, como na Figura 35.

Figura 35 – VI que recebe os dados da porta serial

Fonte: Ajuda do Labview

4.2.6.4 Limpar o buffer.

Com esse VI, mostrado na Figura 36, é possível fazer uma limpeza no buffer serial permitindo uma releitura.

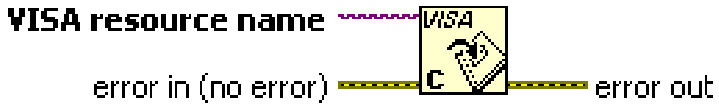
Figura 36 – VI que limpa o buffer

Fonte: Ajuda do Labview

4.2.6.5 Fechar a porta serial

Por fim, este VI encerra comunicação serial. Como mostrado Figura 37.

Figura 37 – VI finaliza comunicação serial



Fonte: Ajuda do Labview

4.2.7 Entradas e saídas

Nesse tópico será abordado os VI que estarão presentes nas duas telas do Labview, e que possuem características de entrada ou saída de informação entre as telas. Para facilitar o entendimento, as figuras deste tópico estarão ordenadas primeiro pela figura a ser mostrada no painel frontal, e em sequência pelo VI que representa no diagrama de blocos.

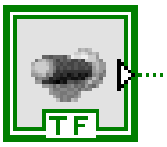
4.2.7.1 Botão de comutação

Esse VI, no painel frontal tem a aparência de um botão de comutação, e passa uma informação booleana de *true* ou *false* para painel de blocos. Suas aparências são mostradas nas Figuras 38 e 39.

Figura 38 – Botão de comutação

Fonte: Produção do próprio autor

Figura 39 – VI de *true* e *fals*e do botão

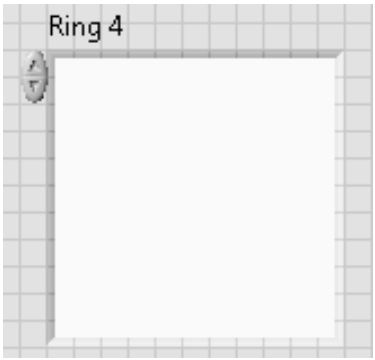


Fonte: Produção do próprio autor

4.2.7.2 Anel de imagem

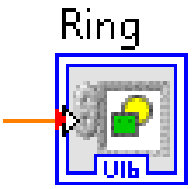
Este VI possibilita a configuração como indicador ou controlador. Nele, é possível colocar uma série de imagens, onde cada uma delas representa um valor numérico. O VI é indicado nas Figuras 40 e 41.

Figura 40 – Anel de imagem



Fonte: Produção do próprio autor

Figura 41 – Anel de imagem

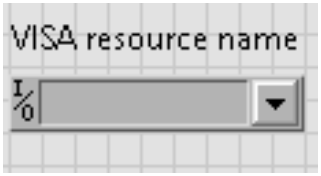


Fonte: Produção do próprio autor

4.2.7.3 VISA *resource name*

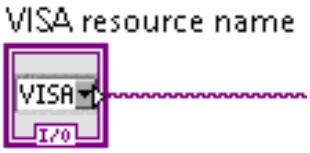
Este VI é o responsável por indicar em qual porta serial está conectado o dispositivo, no qual ocorrerá a comunicação. Na primeira utilização o nome a ser escolhido é COMX, onde x é a porta USB do computador em que o dispositivo está conectado. Caso possua outro programa que utilize a mesma porta serial para comunicar pode ocorrer um conflito. Desta forma é necessário fazer um *refresh.* Assim é possível que ele mude de nome para ASRLX::INSTR, onde x é número variável. Conforme as Figuras 42 e 43.

Figura 42 – VISA *resouce name*



Fonte: Produção do próprio autor

Figura 43 – VISA *resouce name*

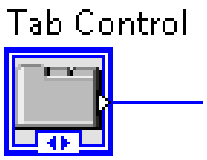


Fonte: Produção do próprio autor

4.2.7.4 Controle de abas

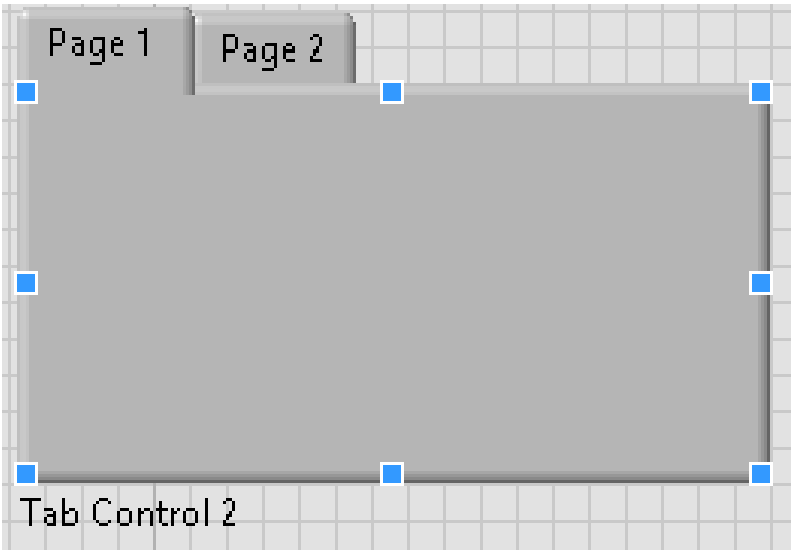
Com esse VI é possível ter várias telas diferentes mostrando dados, gráficos, etc. De forma a ser possível ter mais dados, e também deixar de forma mais visível ou também mais organizado conforme o necessário. Como mostrado nas Figuras 44 e 45.

Figura 44 – *Tab control*



Fonte: Produção do próprio autor

Figura 45 – *Tab control*



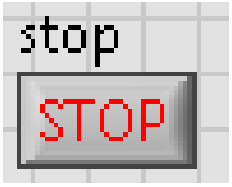
Fonte: Produção do próprio autor

4.2.7.5 *Stop*

O stop funciona como um botão de apertar, ou como um pulso na hora em que é pressionado. Este botão enviará false para o que for determinado. O interessante neste VI,

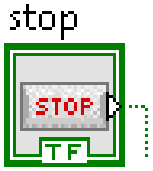
mostrado na figura 46, é que ele chama mais atenção em relação a um botão comum. Assim, é melhor para ser utilizado em processos onde é necessária uma maior atenção. A versão para o diagrama de blocos é mostrada na Figura 47.

Figura 46 – *Stop*



Fonte: Produção do próprio autor

Figura 47 – *Stop*

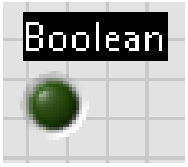


Fonte: Produção do próprio autor

4.2.7.6 LED

Este VI irá indicar um estado booleano. Caso receba verdadeiro estará representado por um verde claro, já se indicar falso será representado por um verde escuro, como mostrado na Figura 48. Para o diagrama de blocos está representado na Figura 49.

Figura 48 – LED



Fonte: Produção do próprio autor

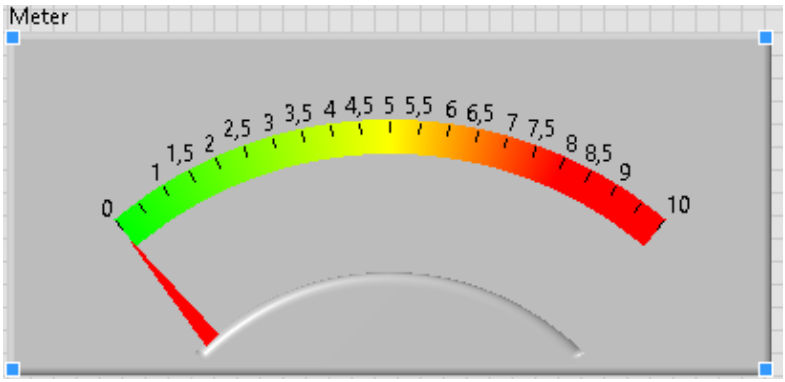
Figura 49 – LED

Fonte: Produção do próprio autor

4.2.7.7 Meter

Através deste VI é possível ver os dados obtidos, de uma forma diferenciada, com aparência analógica. Como representado nas Figuras 50 e 51.

Figura 50 – Meter



Fonte: Produção do próprio autor

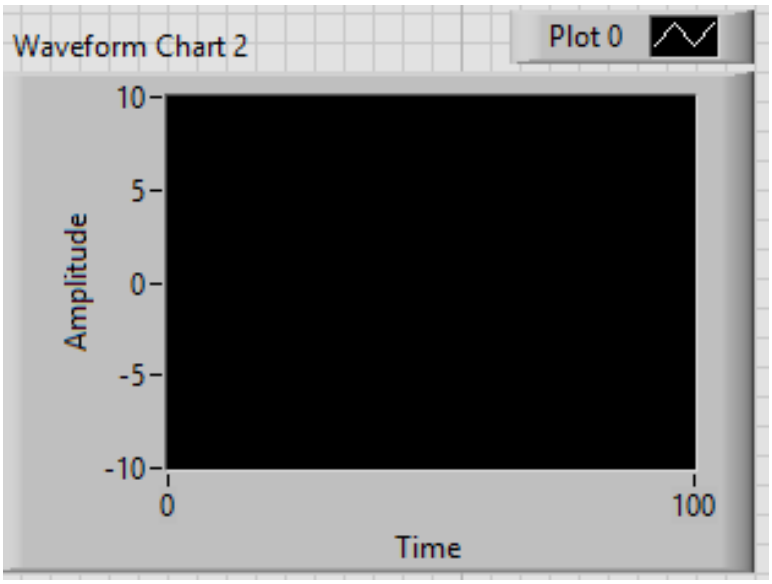
Figura 51 – Meter

Fonte: Produção do próprio autor

4.2.7.8 Gráfico

Com esse VI é possível visualizar o comportamento de dados ao decorrer do tempo de atuação dos dados. Também é possível exportar os dados para um programa, como o excel, por exemplo. Como representado nas Figuras 52 e 53.

Figura 52 – Gráfico



Fonte: Produção do próprio autor

Figura 53 – Gráfico

Fonte: Produção do próprio autor

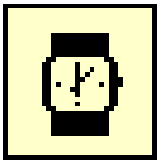
4.2.8 Outros

Neste tópico será abordado alguns VI's utilizados, e que não se encaixam nos outros tópicos.

4.2.8.1 Wait

Este VI, como o próprio nome diz em inglês, tem a função de esperar o programa ou função. Ele requer uma entrada numérica e também possui uma saída numérica, caso seja necessário utilizar. Como mostrado na Figura 54.

Figura 54 – Wait

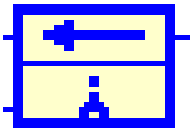


Fonte: Produção do próprio autor

4.2.8.2 Feedback Node

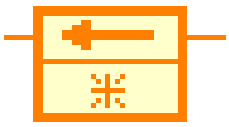
Este VI é mais elaborado, pois ele tem a função de z^{-1}, ou seja, ele guarda um dado de iteração anterior para ser utilizado na próxima. Pode ser utilizado com números inteiros (Figura 55), número de ponto flutuante (Figura 56), condições de verdadeiro ou falso (Figura 57), e também com strings (Figura 58).

Figura 55 – Feedback Node para números inteiros



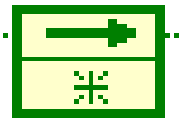
Fonte: Produção do próprio autor

Figura 56 – Feedback Node para números de ponto flutuante



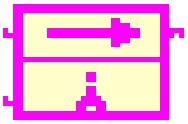
Fonte: Produção do próprio autor

Figura 57 – Feedback Node para condições



Fonte: Produção do próprio autor

Figura 58 – Feedback Node para strings



Fonte: Produção do próprio autor

4.2.8.3 Simples imagens

É possível realizar a inserção de imagens, visando uma melhora no visual do programa. Dessa maneira, o entendimento dos VI's utilizados é facilitado. Para realizar esse procedimento é necessário abrir a figura em um programa de edição de imagens, como o Paint, por exemplo. Dentro desse programa, a figura deverá ser recortada. Posteriormente, já utilizando o Labview, deve-se pressionar ctrl+v para que a figura seja "colada" na tela. Ainda é possível ajustar onde figura irá aparecer.

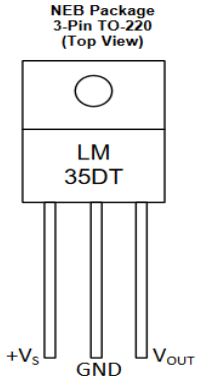
5 SENSORES

Ao todo serão utilizados três sensores para a realização da coleta de dados e para a verificação das condições do equipamento na embarcação.

5.1 Sensor de temperatura

O Primeiro sensor a ser analisado é o LM35. Ele é um sensor de temperatura simples, barato, e possui aparência semelhante a um transistor. Está mostrado na Figura 59.

Figura 59 – LM35



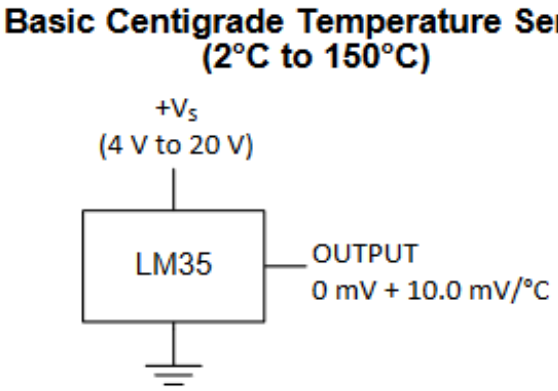
Fonte: Texas Instruments, 1999

Este sensor comporta várias características. As mais relevantes são:

- Calibrado em Celsius
- Linear +10mV/ºC
- Precisão de 0,5 a 25º
- Range de –55ºC até 150ºC
- Opera de 4 a 30V
- Menos de 60µ de corrente de dreno.

A partir das características listadas acima, será utilizado o modelo mais simples, que funciona de 0 a +10mv/ºC. Nesse modelo, como mostrado na Figura 60, a variação de temperatura é de 2 a 150ºC. Sabendo disso, a tensão máxima a ser trabalhada será de 1,5V no canal do sinal. Como será trabalhado com Arduino a tensão de entrada será 5V.

Figura 60 – Funcionamento do sensor

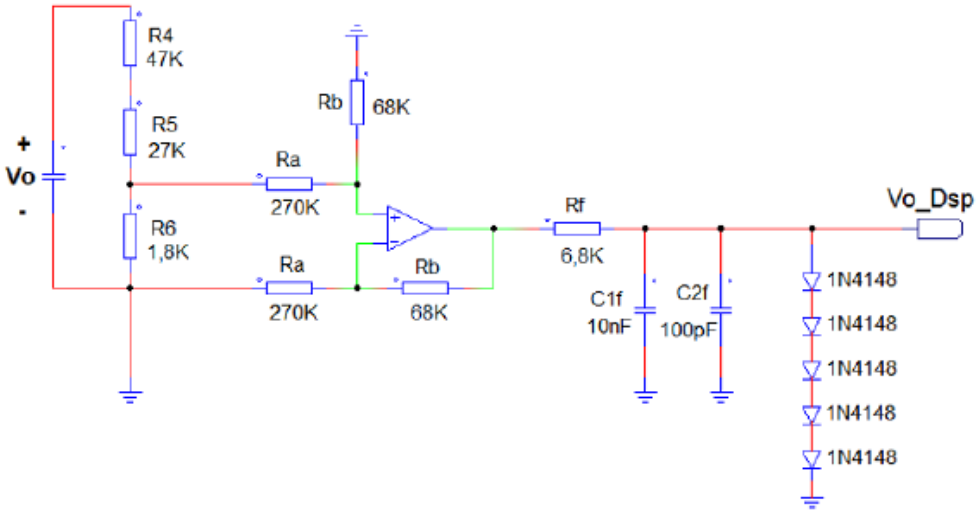


Fonte: Texas Instruments, 1999

5.2 Sensor de tensão

Existem 2 sensores de tensão que são utilizados, resumidamente eles são divisores resistivos. A tensão de entrada esperada é de aproximadamente 200V. Este primeiro sensor com o seu arranjo possui um ganho de $5{,}929 * 10^{-3}$. Como mostrado na Figura 61. Na figura o Vo é a entrada, Vo_Dsp é saída do sensor. Os diodos tem a função de evitar uma sobretensão, o limite de tensão foi definido para no máximo 3V para o sinal de saída. (Souto,2015)

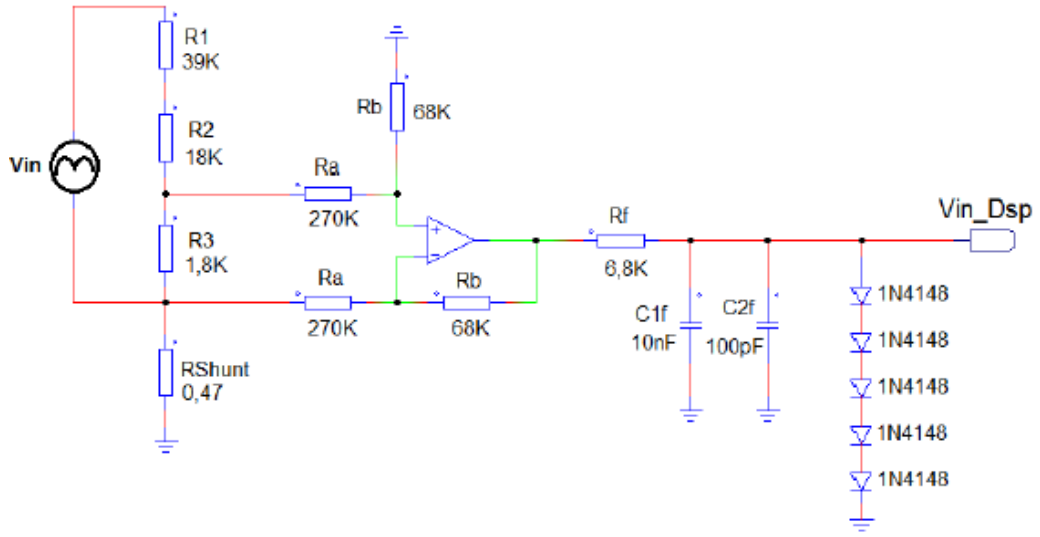
Figura 61 –Sensor de tensão de entrada do conversor



Fonte: Souto, 2015

Este segundo sensor, com o seu arranjo possui um ganho de $7{,}65 * 10^{-3}$, como mostrado na Figura 62.

Figura 62 –Sensor de tensão de saída do conversor

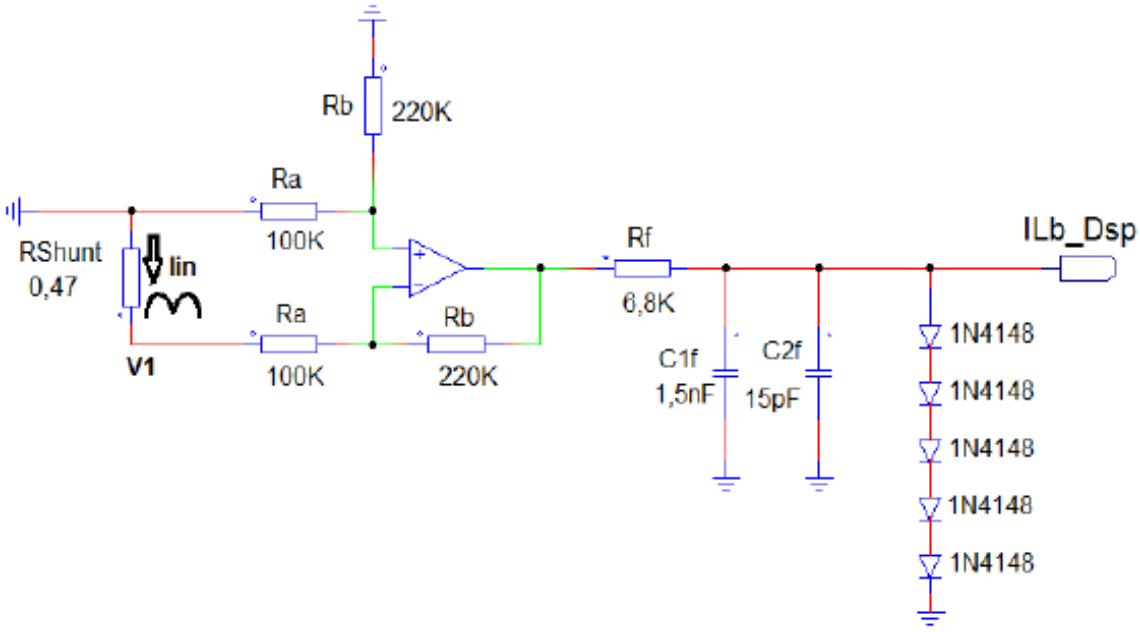


Fonte: Souto, 2015

5.3 Sensor de corrente

É um sensor semelhante ao de tensão, mas possui como diferencial o resistor Shunt. Estes sensor possui um ganho de 1,078, como mostrado na Figura 63.

Figura 63 –Sensor de corrente de entrada do conversor



Fonte: Souto, 2015

6 APLICAÇÕES

Para a utilização do Arduino com o Labview é necessário instalar o programa NI-VISA, que é o responsável pela configuração do computador. Dessa forma é possível realizar a configuração da comunicação serial.

Inicialmente foram realizados testes de comunicação serial, já que através dela ocorrerá a transmissão de todos os dados. Em seguida foi utilizada uma placa, que possui LEDs, botões e potenciômetro para testes. Por fim, foi conectado o Arduino em sensores de corrente e tensão que estão ligados em um conversor, para verificar a aplicação de forma mais próxima da real.

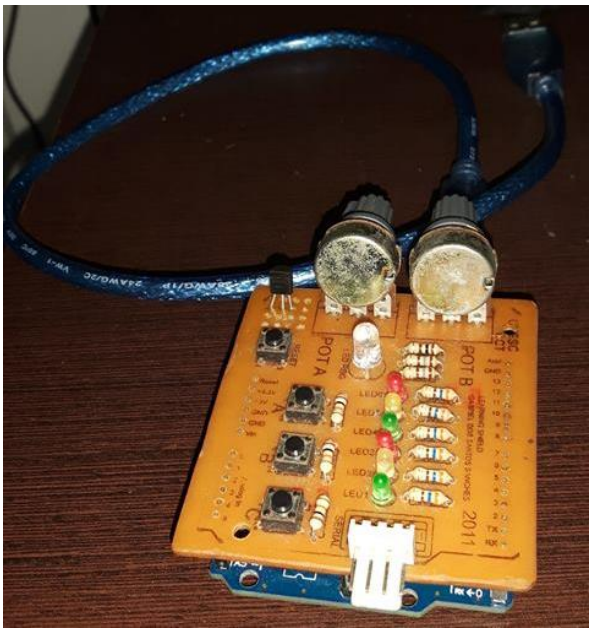
6.1 Primeiros testes

O primeiro teste realizado foi o de comunicação serial. Ele foi de grande dificuldade, tendo em vista que os exemplos disponibilizados na internet não funcionavam. O primeiro fator trabalhado foi a definição dos parâmetros de comunicação serial, como a taxa de transmissão de 9600, por exemplo. O shield do Arduíno abrange LEDs de TX e RX, que são os canais de comunicação. Quando os dados eram enviados do Arduíno para o Labwiew, o TX piscava. Quando ocorria o contrário, ou seja, o Arduíno recebia os dados do Labview, o RX piscava. Depois de verificar os LEDs piscando foi iniciado o primeiro teste de Transmissão de uma string.

O primeiro teste consistia no recebimento de um sinal, e na retransmissão do mesmo. Durante o procedimento ocorria o surgimento de caracteres indesejados na comunicação, como eles seguem o padrão ASCII pude notar que o problema com o tempo da transmissão, ou com a quantidade de dados, normalmente eram valores altos da tabela, concluí então que eram consequência de um overflow de dados. Dessa maneira, foi definido um caractere com a função de finalizar a transmissão ou o recebimento de dados. O caractere definido foi o dois pontos.

Em sequência, foi acoplada uma placa no Arduino, ela foi fornecida pelo grupo PET e está a mostra na Figura 64. A placa abrange 6 LEDS comuns, 1 LED RGB, 3 botões de utilização geral, 1 botão de reset, 2 potenciômetros, 1 saída serial, e um sensor LM35, que foi soldado em espaços disponíveis na placa.

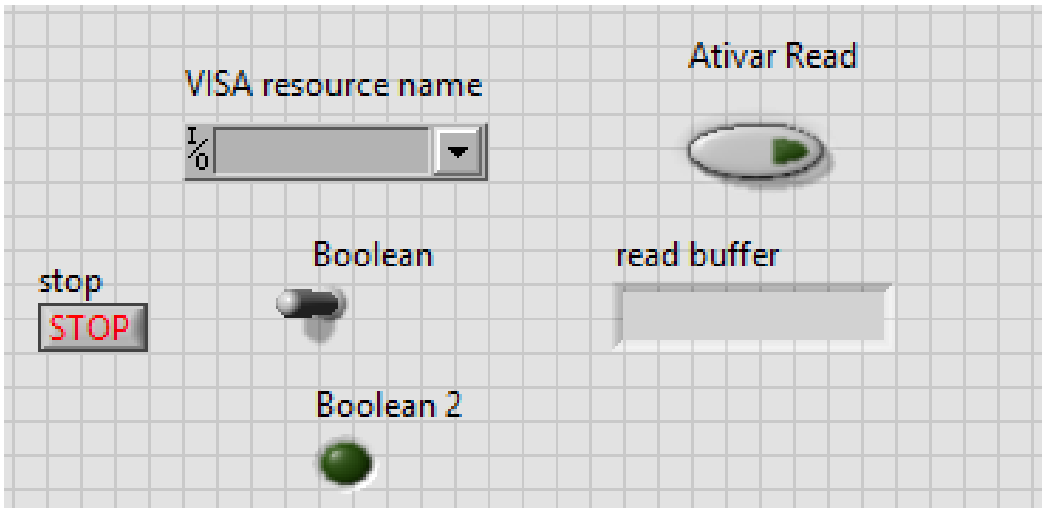
Figura 64 – Placa do PET



Fonte: Produção do próprio autor

O próximo passo era fazer um dos LEDS, na placa, ligar ou desligar utilizando o Labview. Esse programa tinha como objetivo a criação de um protocolo para que os dois dispositivos operassem em harmonia, e fosse possível realizar os procedimentos necessários. Foi definido o padrão de LX, onde o LED seria ligado, e X seria um número variável de 1 a 6. Para desligar o LED foi usado o padrão DX. A programação está descrita nos apêndices A para o Arduino, e B para o Labview, no diagrama de blocos. A Interface com os elementos mais básicos está mostrada na Figura 65.

Figura 65 – Interface inicial



Fonte: Produção do próprio autor

6.2 Aplicação na placa

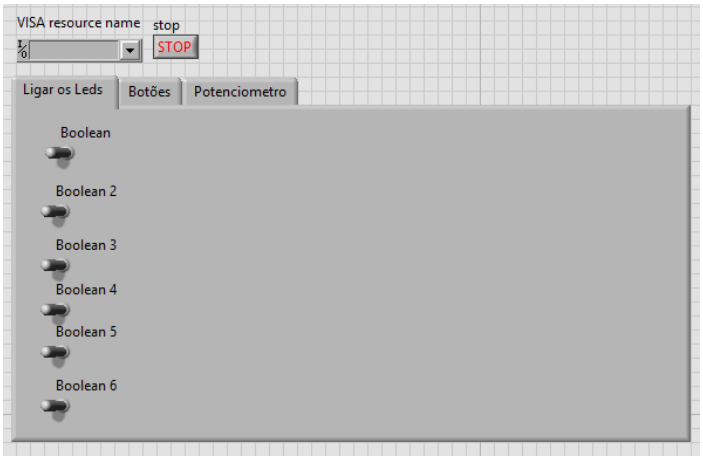
Em sequência, foi aprimorado o programa. O número de LEDs que seriam ligados na placa aumentou, por consequência foi necessário aprimorar o sistema das strings na programação do Arduino, já que cada LED necessitaria de uma string correspondente, como mostrado no apêndice C.

Já no Labview a programação ficou mais complexa. Devido a existência de 6 LEDs, foi necessário fazer um loop com 6 iterações para cada LED. A parte de lógica mais complicada foi o concatenamento de dados, pois a string final deveria ter os caracteres L e D, e os números dos 6 LEDs. Ainda foi necessário criar um sistema de verificação da string, para que nenhum dado ou iteração incoerente fosse enviado para o Arduino.

Após finalizado o sistema dos LEDs, foram colocadas abas, já que com adição de vários elementos, a string acabaria ficando muito complexa. Dessa forma, cada String de aba possui uma caraterística própria de início, consequentemente a quantidade de dados relevantes a serem trabalhados é menor.

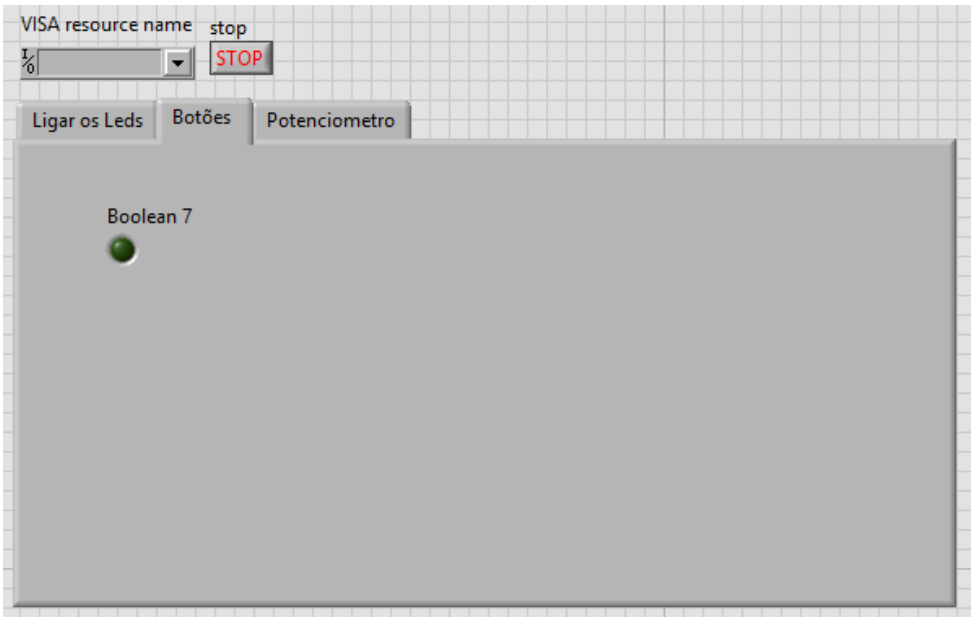
Com sistema de abas criado, foi desenvolvida a aba botão, onde um LED, localizado no Labview, envia do Labview para o Arduino, a informação de que aba botão está aberta. Em sequência o Labview recebe a informação da condição do botão, se ele está pressionado ou não. Assim, ele liga o VI de LED no Labview. Finalmente, para garantir que a transmissão foi realizada com sucesso, o primeiro LED da placa é aceso. Como mostrado nos apêndices C e D das Figuras 66 e 67, mostra a interface no momento, com a aba de LEDS e a aba botões.

Figura 66 – Interface LEDs



Fonte: Produção do próprio autor

Figura 67 – Interface botões



Fonte: Produção do próprio autor

Como mostrado na Figura 67, a aba potenciômetro já estava criada, mas não continha a função. Dando continuidade à criação, foi adicionado ao programa Arduino a leitura analógica de dados. Esses dados são adicionados a uma string, que é enviada ao Labview, e quando disponibilizada num gráfico permite ver as condições de tensão do potenciômetro.

Os dados do potenciômetro variam de 0 a 5V, ou seja, são adquiridos dentro do Arduino, de 0 a 1023. No Arduino é programada a conversão de 0 a 500. Como esse valor varia de 1 a 3 dígitos, ocorreu uma interferência na comunicação serial, sendo necessário fazer uma adição com o número 100 para manter 3 dígitos ao enviar para o Labview. Ao chegar no Labview é necessário interpretar a string, Do valor adquirido é preciso subtrair em 100, como mostrado nos apêndices E e F.

Ao realizar o envio das strings, surgiram erros na transmissão quando ocorriam trocas de abas. Algumas vezes, durante uma transmissão, apareciam dados indesejados armazenados no buffer. Então foi necessário fazer uma verificação dos dados, e caso fossem indesejados, era feita uma limpeza no buffer de dados, que continha as strings. A placa contava com 2 potenciômetros no Labview para a interface. Os 2 dados são mostrados de forma numérica, mas possuem um VI de botão que define qual deles será mostrado no gráfico. Na Figura 68 é mostrada a interface para o potenciômetro.

Figura 68 – Interface potenciômetro.

Fonte: Produção do próprio autor

Na placa foi soldado um sensor de temperatura, o LM35, que varia de 0 a 1,5V, que equivale a 0 a 307. Nessa variação de tensão ele varia de 2 a 150ºC. Como ocorreu o mesmo problema na quantidade de dígitos que o procedimento anterior, para ser enviado teve um incremento em 100, antes de fazer a comunicação serial. Ao chegar no Labview foi decrementado e enviado para um Meter. Como mostrado no apêndice K.

6.3 Coletar dados em um conversor.

Em um sistema de embarcação é necessário analisar os dados fornecidos pelos sensores, para verificar as condições do conversor. Foram utilizados sensores de tensão e corrente.

Os sensores foram trabalhados de forma muito semelhante a do potenciômetro, já que os dados analógicos eram coletados e posteriormente convertidos. Os dados convertidos foram enviados ao Labview, reajustados, e por fim aplicou-se o ganho dos sensores com a finalidade de obter o valor real dos mesmos.

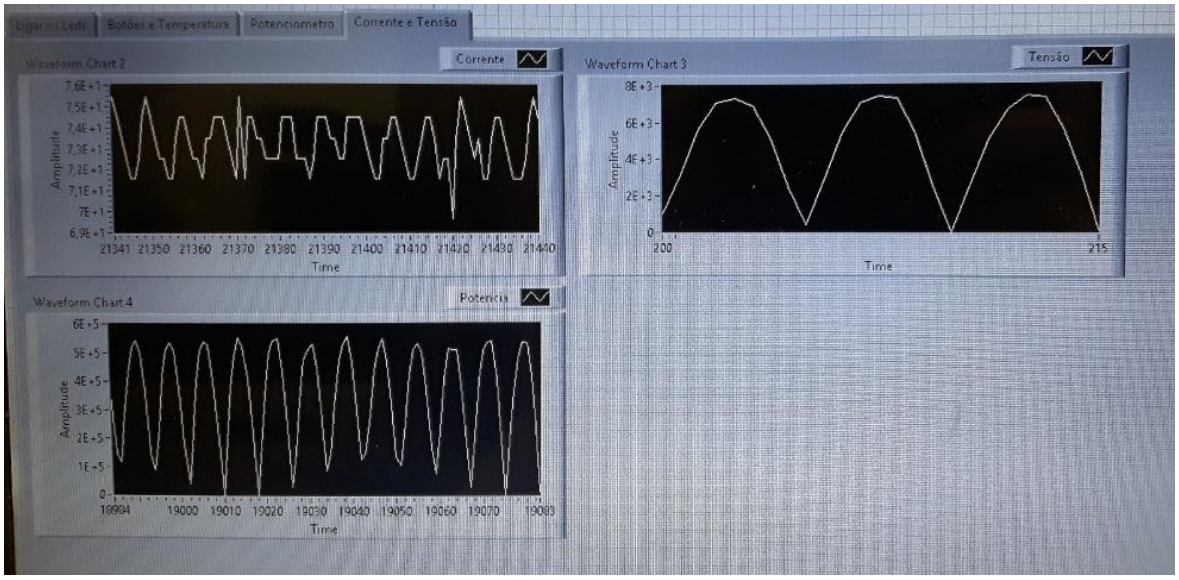
Ao todo foram dois dias de teste, realizados no NPEE. No primeiro dia foi utilizado um conversor *boost*, que converte uma tensão de entrada de 200V para 400V. Infelizmente, não foi possível realizar o procedimento com um conversor extremamente

semelhante ao utilizado em uma embarcação, mas foi possível coletar os dados para fazer o programa. Como o conversor necessitava de muita carga para operar com potência total e não havia carga suficiente disponível, foi utilizada uma tensão de entrada de 100V e uma tensão de 200V de saída. Os dados de entrada e saída são senoidais, mas como não estavam na tensão esperada, ocorreram erros devido a ruídos nos sensores de tensão e corrente.

Para gerar a tensão de entrada utilizou-se um Varivolt. Foi necessário utilizar um gerador de sinais para alimentar o PWM das chaves do conversor, com *dutty cycle* de 50%. Ainda, foi utilizada uma fonte de alimentação de 15 volts DC para alimentação do CI's, 4 resistores de 20 ohms como carga, e o osciloscópio, para verificar o sinal coletado pelos sensores e comparar com sinais mostrados no computador.

Ao realizar os primeiros testes, o programa ainda não estava completamente correto, pois os ganhos não estavam devidamente ajustados, e os cabos estavam com problema de mal contato. Também foi necessário arrumar o problema do buffer. Outro problema era a incerteza da existência dos 3 sensores, de forma que foram coletados dados de apenas 2 deles. Ainda, a quantidade de dados por tempo estava muito alta, então foi necessário diminuir na análise do gráfico para que fosse possível analisar, como mostrado na Figura 69. As fotos do teste estão disponíveis no apêndice G. É importante destacar que nesse dia de teste não foi possível utilizar a placa usada anteriormente, pois ela utilizava muitas portas analógicas.

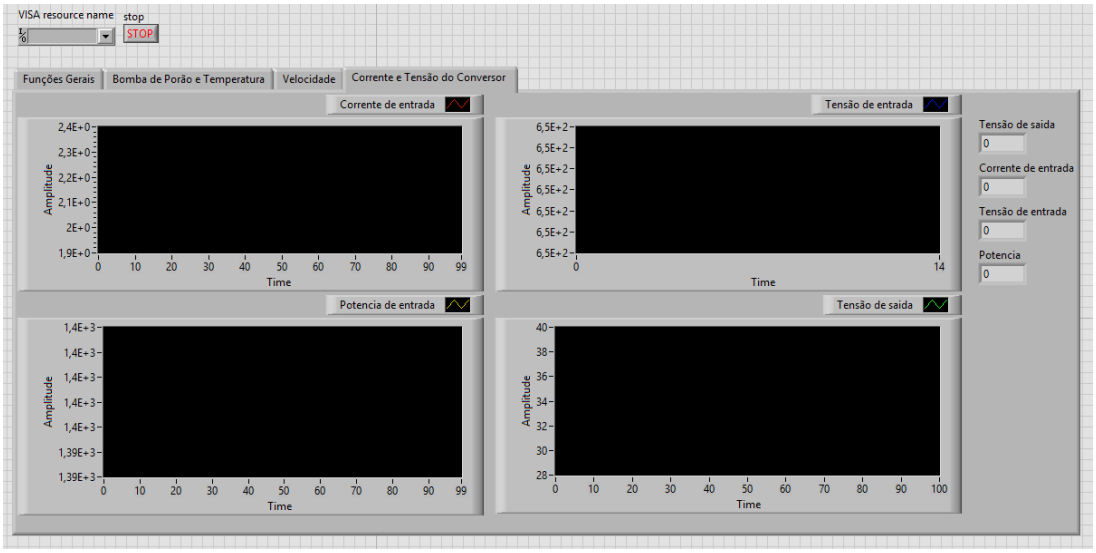
Figura 69 – Interface com dados no primeiro dia



Fonte: Produção do próprio autor

Realizando as correções dos erros constados no parágrafo anterior. O programa, tanto no Labview, quanto no Arduino estavam mais estruturados e prontos para receber os dados, como mostrado na Figura 70. No processo de remontagem do hardware foram utilizados cabos de melhor qualidade, visando diminuir os erros provenientes do mal contato, como mostrado no apêndice H.

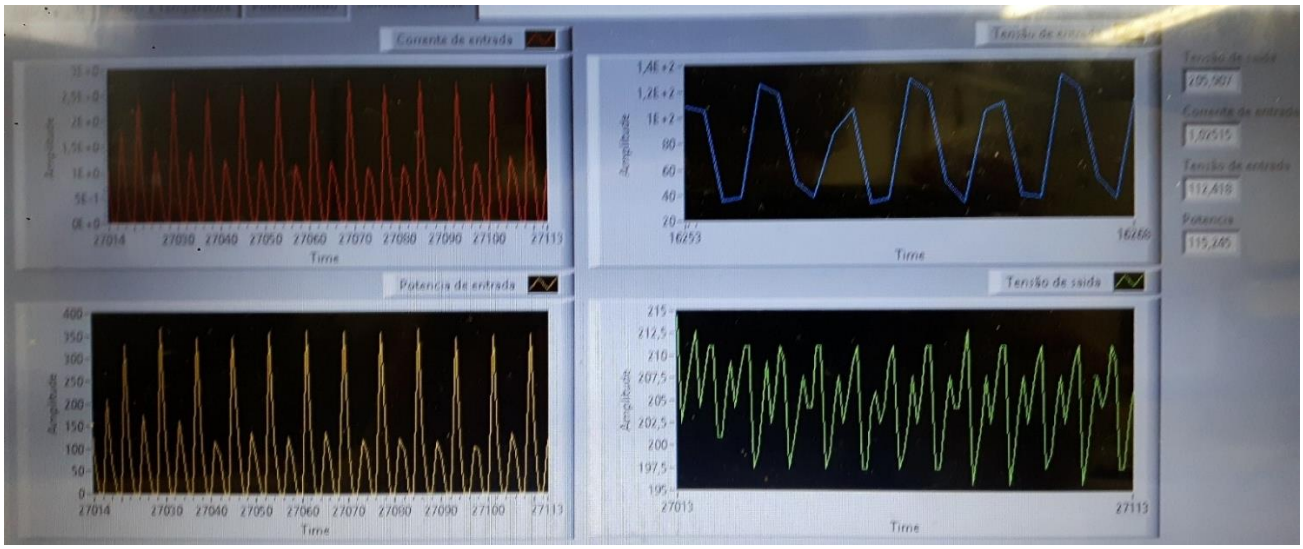
Figura 70 – Interface no segundo dia



Fonte: Produção do próprio autor

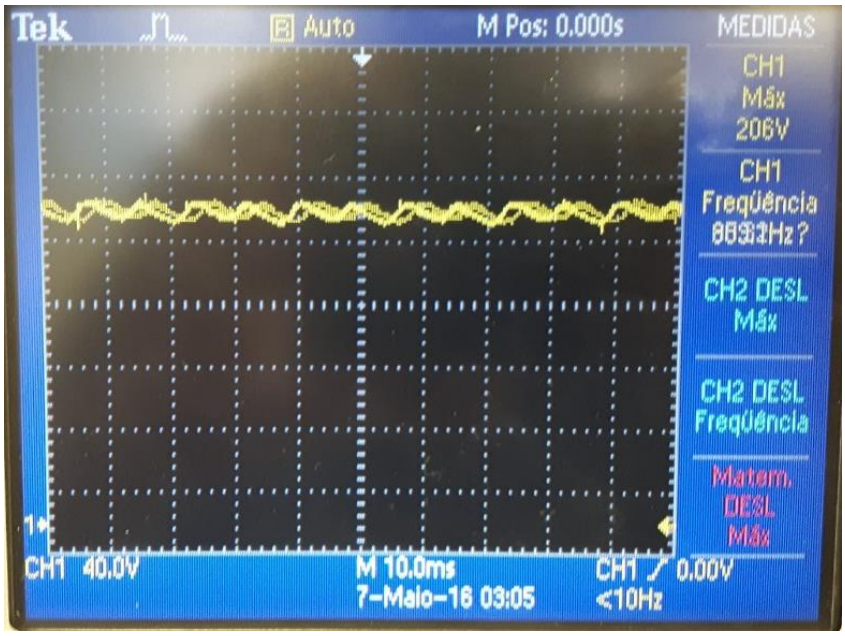
Dessa maneira, o programa foi implementado e os dados foram coletados e em seguida exportados. Os dados estão disponíveis no apêndice I e também são mostrados na Figura 71, através da interface do Labview. A Figura 72 mostra o sinal no osciloscópio quando a ponteira dele está conectada na saída do conversor.

Figura 71 – Interface com dados no segundo dia



Fonte: Produção do próprio autor

Figura 72 – Osciloscópio com sinal de saída no segundo dia



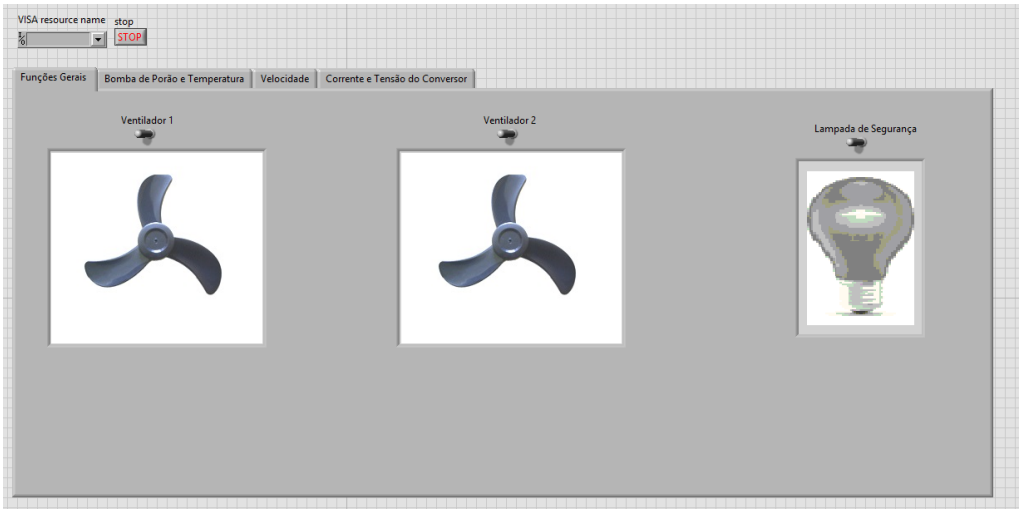
Fonte: Produção do próprio autor

6.4 Versão final

Na última versão do programa, a interface foi aprimorada, com o objetivo de ser amigável e de fácil entendimento ao usuário que for utiliza-la. Nela foram adicionados elementos de imagens complementar os elementos já existentes. Uma ideia geral do programa final é visto no Fluxograma no apêndice N.

A primeira aba foi nomeada de “Funções Gerais”, finalizada com 3 elementos, sendo 2 ventiladores para as baterias e 1 lâmpada para área do conversor, onde são ativadas por VI de botões, como mostrado na Figura 73.

Figura 73 – Versão final da aba “Funções Gerais”.



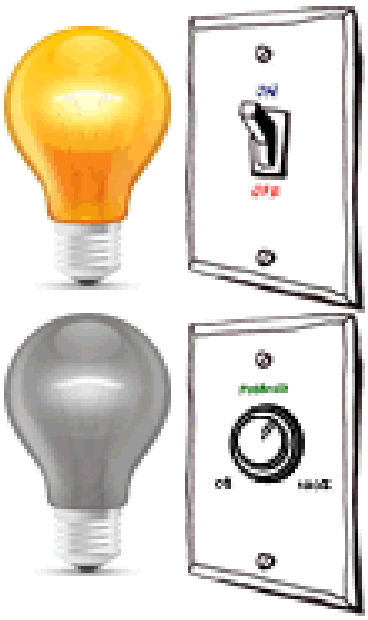
Fonte: Produção do próprio autor

AS figuras de ventilador e de luz foram editadas, para fazer as animações. As figuras originais são as Figuras 74 e 75.

Figura 74 – Ventilador.

Fonte: https://www.dompepeud.com.br/loja/images/Helice%20Ventilador%20Houston%2050%20cm.gif

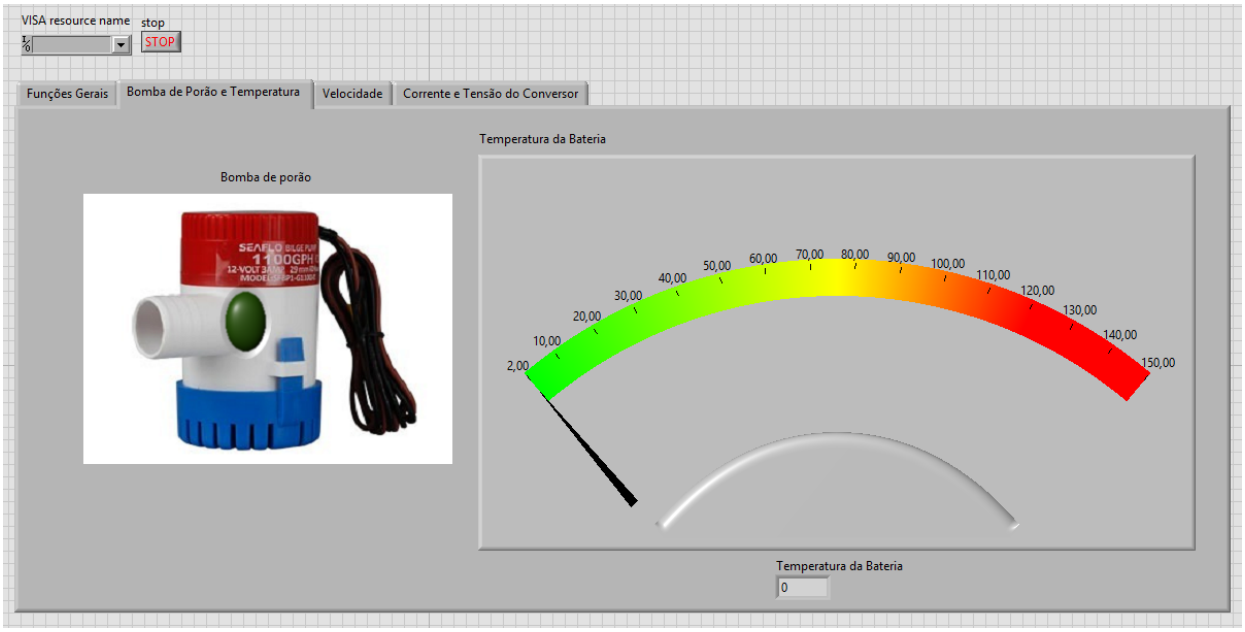
Figura 75 – Lâmpada



Fonte: http://www.arcondicionadoemnatal.com.br/inverter/gif.gif

A aba “botão” ela tornou-se aba “Bomba de Porão e Temperatura”, caso as bombas de porão sejam ativadas, será demonstrado por um VI de LED. Esta aba ainda mostra o VI meter, que vai mostrar os dados do sensor LM35. Mostrado na Figura 76. A imagem da bomba de porão utilizada é igual a Figura 77.

Figura 76 – Versão final da aba “Bomba de Porão e Temperatura”.



Fonte: Produção do próprio autor

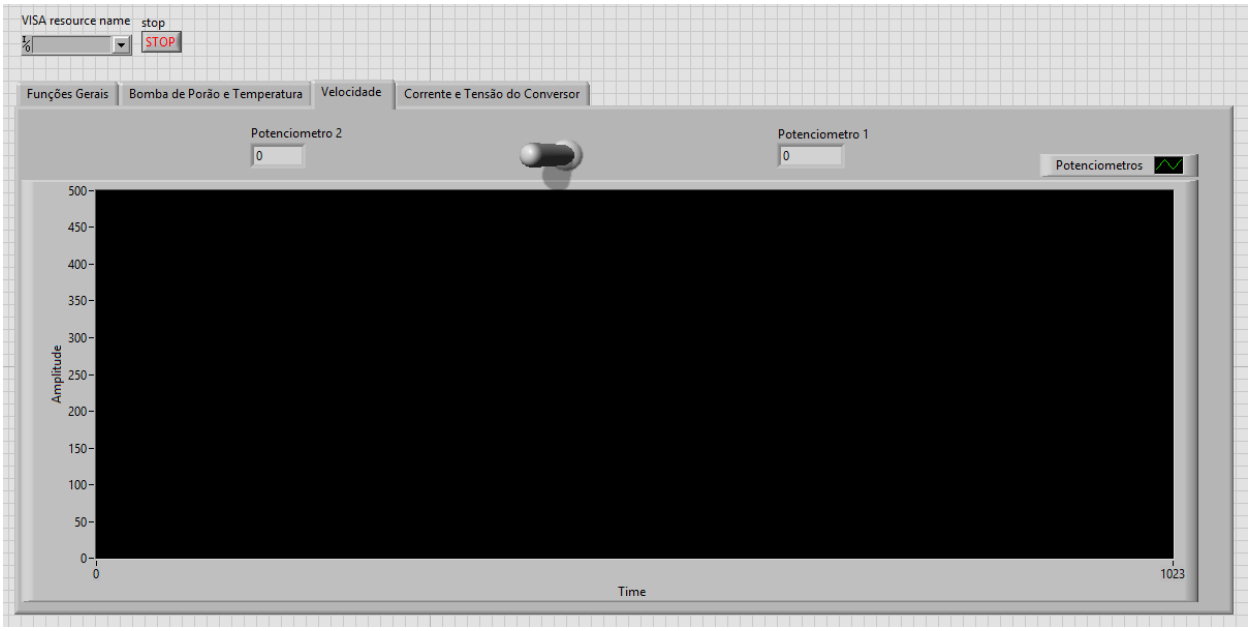
Figura 77 – Bomba de porão.

Fonte: http://www.seaflo.com/en/productDetail_285.html

A terceira aba foi nomeada de “Velocidade”. Ela está localizada onde anteriormente ficava a aba dos potenciômetros. Abrange um botão, que quando

pressionado mostra no gráfico o valor do potenciômetro escolhido e pode ser equivalente a potência fornecida a um determinado motor, como mostrado na Figura 78.

Figura 78 – Versão final da aba “Velocidade”.



Fonte: Produção do próprio autor

Por fim, a última aba é “Corrente e Tensão do Conversor”, onde estão localizados os gráficos dos dados recebidos, e que não foi alterada, então é mesma da Figura 70.

7 CONCLUSÃO

O tema de interfaces é de extrema importância. Isso se deve ao fato de que a interface permite que o usuário realize o controle do equipamento que lhe foi atribuído. Além disso, uma interface prática pode contribuir significativamente no ganho de tempo.

Criou-se um programa no Labview para ser aplicado em embarcações. A criação do mesmo possibilitou a visualização de vários dados relevantes, como os fornecidos por sensores de tensão, corrente e temperatura. Um exemplo de melhoria na programação seria uma filtragem para os dados recebidos, por exemplo.

Também, através desse programa foi possível ativar e desativar ventiladores, e a lâmpada da área do conversor. Ainda possibilitou verificar a condição da bomba de porão, ou seja, o objetivo do Labview foi realizado com sucesso.

Um dos fatores importantíssimos, que permitiu a realização de todos os procedimentos listados acima, foi a comunicação serial. No decorrer do TCC ela foi realizada exclusivamente através do cabo USB, porém, poderia ser implementada no futuro através de uma transferência de dados via WIFI ou GSM. Outra coisa interessante a ser pesquisada em mais detalhes seria algum tipo de padrão internacional na forma de envio de dados.

O Arduino mostrou-se bastante válido e útil nos serviços requisitados, além de ser um dispositivo de alto custo-benefício, abrange uma infinidade de material disponível para ajudar na evolução do projeto realizado. Assim cumprindo o seu objetivo.

Visto que o objetivo maior do TCC era elaborar um programa, onde seria coletado dados, armazenados no Arduino, para serem transmitidos para o Labview e por fim serem demonstrados para os usuários, foi realizado com um ótimo desempenho.

Esse TCC poderia no futuro ser implementado por exemplo no projeto de extensão da UDESC Barco Solar. Já que se demonstrou valido em uma placa que simula os componentes reais.

8 REFERÊNCIAS

[1] ATMEL. **ATmega-Datasheet.** San Jose, CA. Tradução própria.2015. Disponível em: <http://www.atmel.com/images/atmel-8271-8-bit-avr-microcontroller-atmega48a-48pa-88a-88pa-168a-168pa-328-328p_datasheet_complete.pdf>. Acesso em: 2 de fevereiro de 2016.

[2] Barragan, Hernando. **The Untold History of Arduino.** Tradução própria. Disponível em: <http://arduinohistory.github.io/>. Acesso em: 10 de março de 2016.

[3] Cruz, Adriano Joaquim de Oliveira. **Tipos de Dados, Constantes e Variáveis.** 1997. Disponível em:< http://equipe.nce.ufrj.br/adriano/c/apostila/tipos.htm > Acesso em: 30 de março de 2016.

[4] Nacional Instruments. **NI-VISA Overview.** Tradução própria. Fevereiro de 2009. Disponível em: <http://www.ni.com/tutorial/3702/en/>. Acesso em: 10 de abril de 2016.

[5] Texas Instruments. **LM35 Precision Centigrade Temperature Sensors.** Tradução própria. 1999. Disponível em: <http://www.ti.com/lit/ds/symlink/lm35.pdf >. Acesso em: 30 de maio de 2016.

[6] SOUTO, Marcelo Christian Lopes. **Conversor boost PFC com controle digital.** 2015. 122 f. TCC (Graduação)-Universidade do Estado de Santa Catarina, Curso de Engenharia Elétrica, Joinville, 2015

APÊNDICES

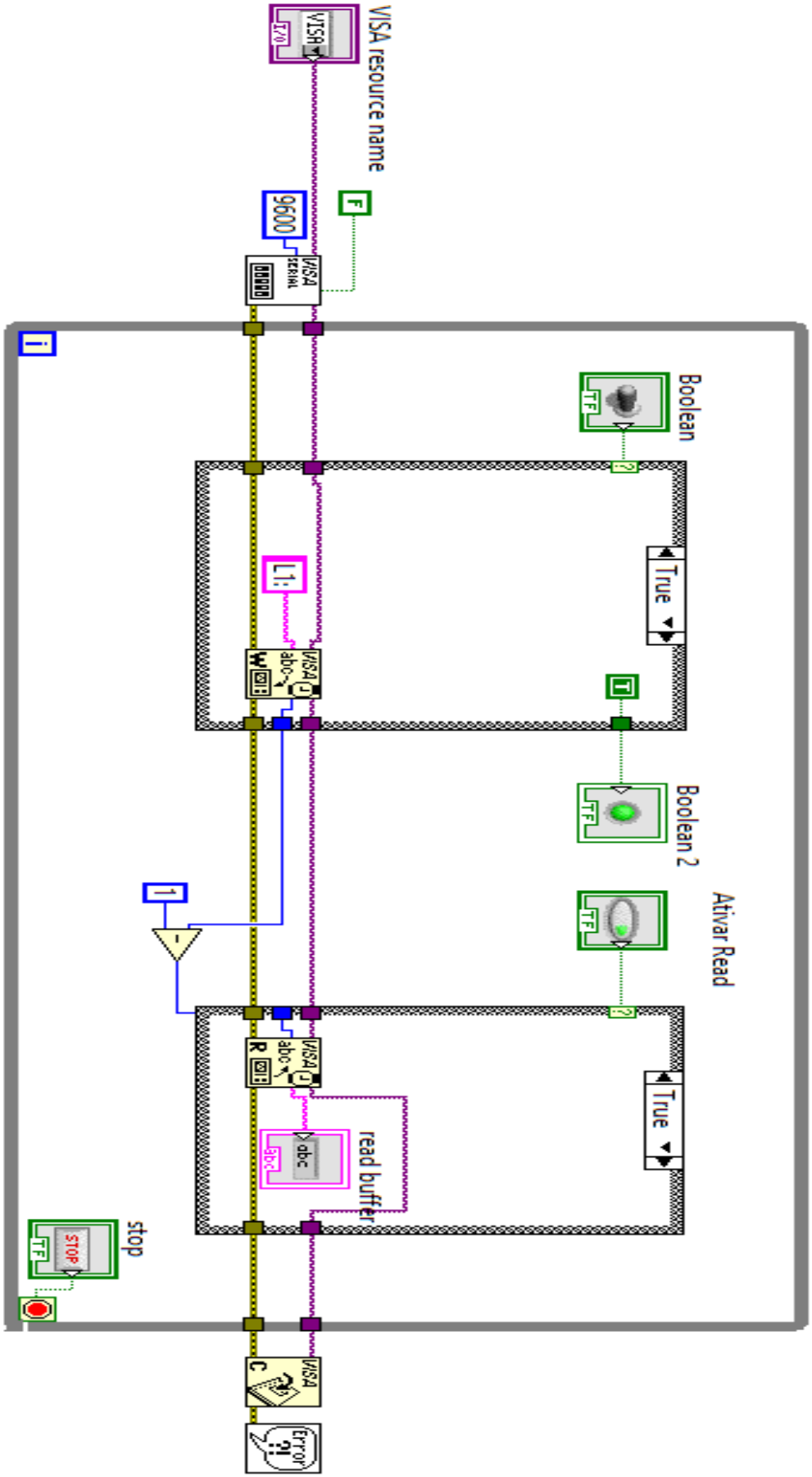
APÊNDICE A – Código da primeira implementação no Arduino.

```
// Variaveis declaradas
char CARACTER = 0;
String STRING = "";
int TERMO;
char Leds[] = {2};

void setup()
{
  //Configuração Serial
  Serial.begin(9600);
  //Saida
  pinMode(Leds[0], OUTPUT);
}


void loop()
{
   //Coleta de dados de string na variável STRING através da comunicação serial. Os dois pontos é um termo criado para indicar finalização da comunicação serial.
  while (Serial.available() > 0)
  {
    CARACTER = (byte)Serial.read();
    STRING = STRING + CARACTER;
    TERMO = STRING.indexOf(":");
    if (STRING.charAt(TERMO) == ':')
    {
      break;
      CARACTER = 0;
```

```
  }
}
//Baseado na string recebida ele vai verificar se deve ligar o LED
TERMO = STRING.indexOf("L1");
if (STRING.charAt(TERMO) == 'L')
{
  digitalWrite(Leds[0], HIGH);
  // aqui é enviada a string para ligar o LED
  String TESTE="L1";
  for (int i = 0; i < TESTE.length(); i++)
  {
    Serial.write(TESTE[i]);
  }
  delay(10);
}
//Baseado na string recebida ele vai verificar se deve desligar o LED

TERMO = STRING.indexOf("D1");
if (STRING.charAt(TERMO) == 'D')
{
  digitalWrite(Leds[0], LOW);
    // aqui é enviada a string para desligar o LED
  String TESTE="D1";
  for (int i = 0; i < TESTE.length(); i++)
  {
    Serial.write(TESTE[i]);
  }
  delay(10);
}
```

```
  //Limpa a variável STRING ao verificar o termo :
  TERMO = STRING.indexOf(":");
  if (STRING.charAt(TERMO) == ':')
  {
    STRING = "";
  }

}
```

APÊNDICE B – Diagrama de blocos dos primeiros testes no Labview



APÊNDICE C – Programa no Arduino para os 6 LEDs e botão

```
// Variáveis declaradas

char CARACTER = 0;

String STRING = "";

String a = "0";

int TERMO;

// Vetores das portas de saídas e entradas

char Leds[] = {2, 4, 7, 8, 12, 13};

char botao[] = {17, 18, 19};


void setup()

{

  //Configuração Serial

  Serial.begin(9600);


  //Definindo os LEDs como saídas

  pinMode(Leds[0], OUTPUT);

  pinMode(Leds[1], OUTPUT);

  pinMode(Leds[2], OUTPUT);

  pinMode(Leds[3], OUTPUT);

  pinMode(Leds[4], OUTPUT);

  pinMode(Leds[5], OUTPUT);

  //Definindo os botões como entrada
```

```
  pinMode(botao[0], INPUT);
  pinMode(botao[1], INPUT);
}

void loop()
{
  //Coleta de dados de string na variável STRING através da comunicação serial. Os dois pontos é um termo criado para indicar finalização da comunicação serial.
  while (Serial.available() > 0)
  {
    CARACTER = (byte)Serial.read();
    STRING = STRING + CARACTER;
    TERMO = STRING.indexOf(":");
    if (STRING.charAt(TERMO) == ':')
    {
      break;
      CARACTER = 0;
    }
  }
   //Baseado na string recebida ele vai verificar se deve ligar ou desligar os LEDs

  TERMO = STRING.indexOf("L1");
  if (STRING.charAt(TERMO) == 'L')
```

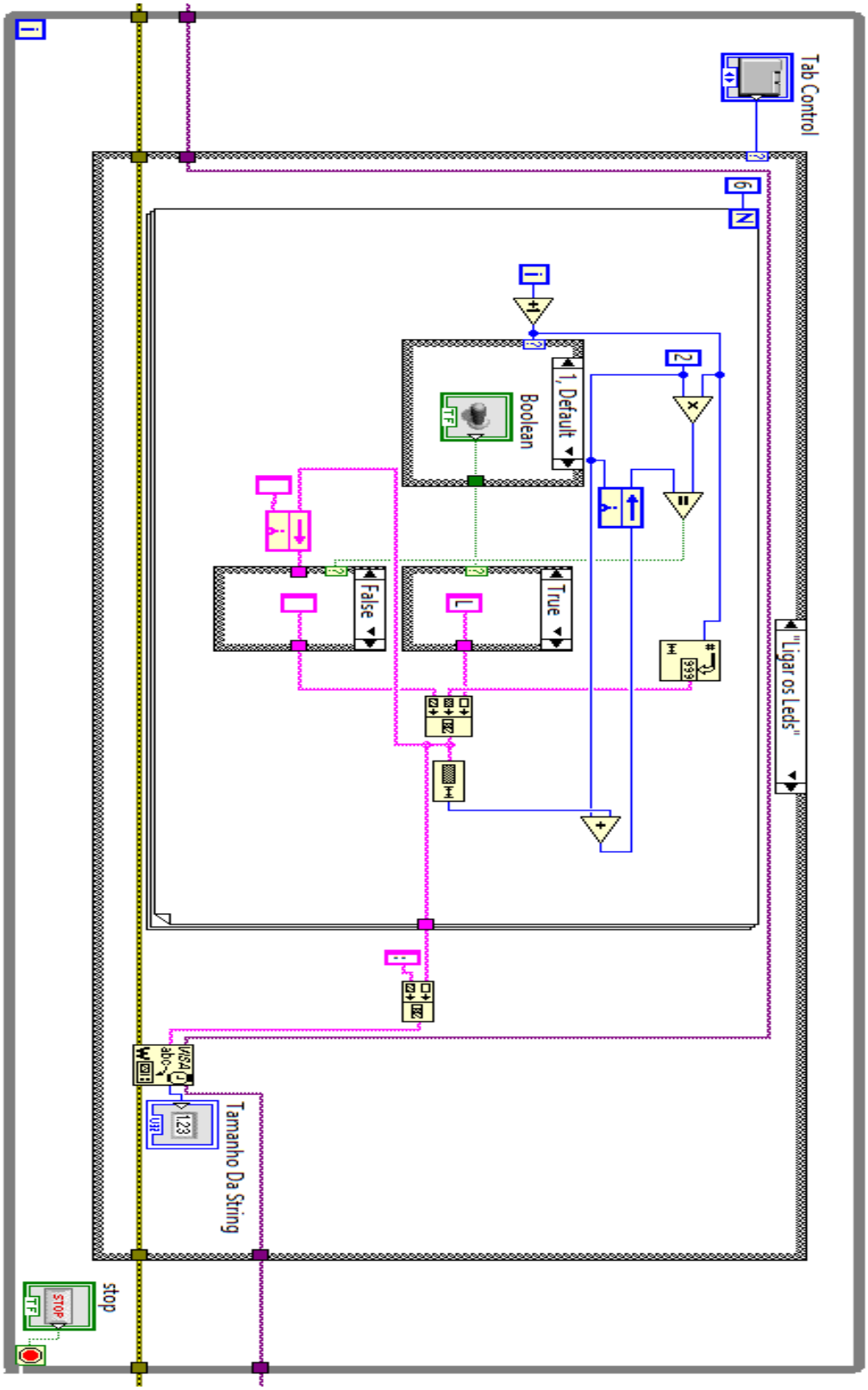
```
{
  digitalWrite(Leds[0], HIGH);
}
TERMO = STRING.indexOf("L2");
if (STRING.charAt(TERMO) == 'L')
{
  digitalWrite(Leds[1], HIGH);
}
TERMO = STRING.indexOf("L3");
if (STRING.charAt(TERMO) == 'L')
{
  digitalWrite(Leds[2], HIGH);
}
TERMO = STRING.indexOf("L4");
if (STRING.charAt(TERMO) == 'L')
{
  digitalWrite(Leds[3], HIGH);
}
TERMO = STRING.indexOf("L5");
if (STRING.charAt(TERMO) == 'L')
{
  digitalWrite(Leds[4], HIGH);
}
```

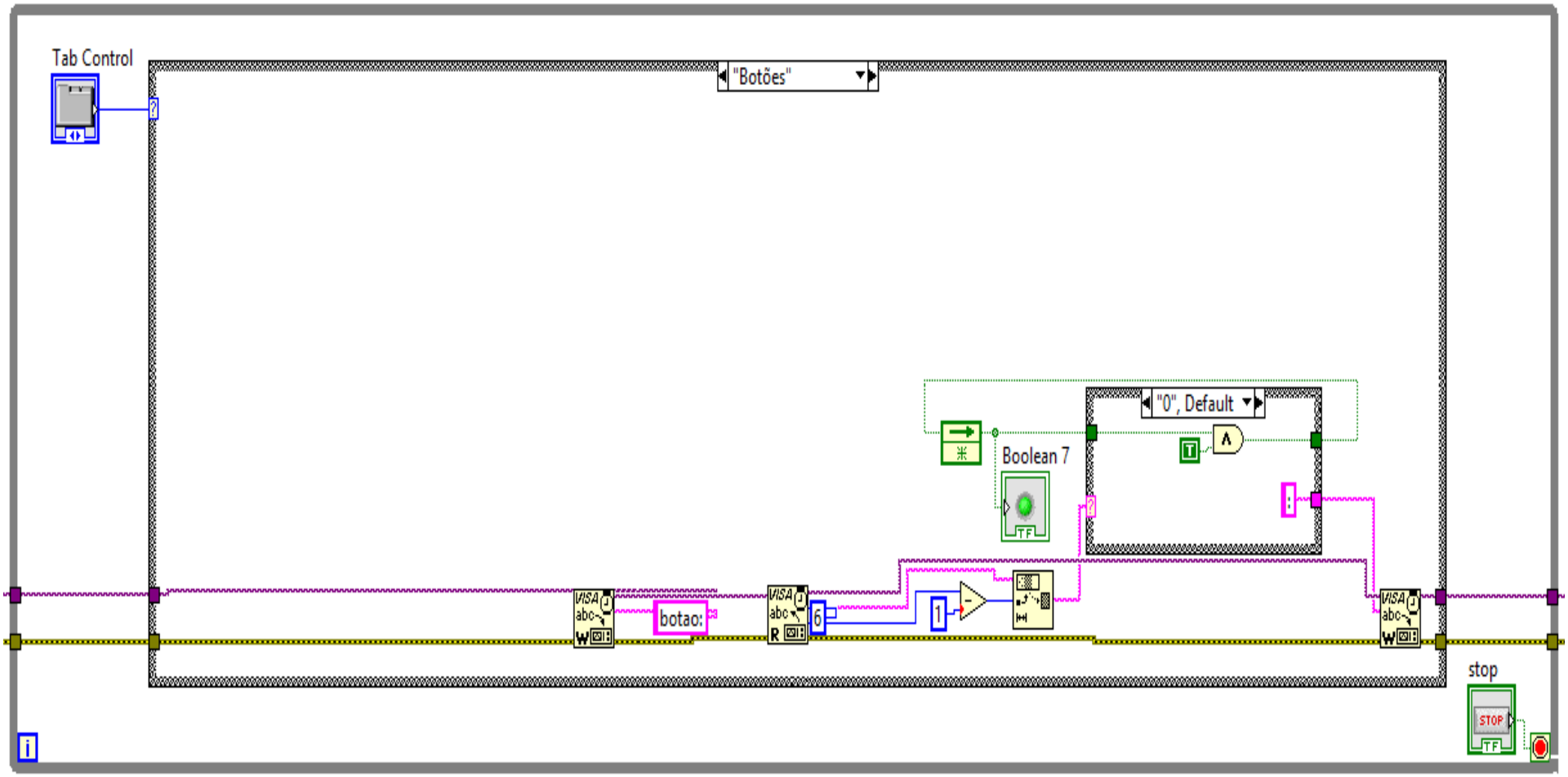
```
TERMO = STRING.indexOf("L6");
if (STRING.charAt(TERMO) == 'L')
{
  digitalWrite(Leds[5], HIGH);
}
TERMO = STRING.indexOf("D1");
if (STRING.charAt(TERMO) == 'D')
{
  digitalWrite(Leds[0], LOW);
}
TERMO = STRING.indexOf("D2");
if (STRING.charAt(TERMO) == 'D')
{
  digitalWrite(Leds[1], LOW);
}
TERMO = STRING.indexOf("D3");
if (STRING.charAt(TERMO) == 'D')
{
  digitalWrite(Leds[2], LOW);
}
TERMO = STRING.indexOf("D4");
if (STRING.charAt(TERMO) == 'D')
{
```

```
  digitalWrite(Leds[3], LOW);
}
TERMO = STRING.indexOf("D5");
if (STRING.charAt(TERMO) == 'D')
{
  digitalWrite(Leds[4], LOW);
}
TERMO = STRING.indexOf("D6");
if (STRING.charAt(TERMO) == 'D')
{
  digitalWrite(Leds[5], LOW);
}
//Aqui vai ser verificado se variável STRING está escrito botão.

TERMO = STRING.indexOf("botao");
if (STRING.charAt(TERMO) == 'b')
{
  a = "0";
  if (digitalRead(botao[0]) == 0)
  {
    a = "1";
  }
  if (digitalRead(botao[1]) == 0)
```

```
    {
      a = "2";
    }
    String teste = "botao" + a;
    for (int i = 0; i < teste.length(); i++)
    {
      Serial.write(teste[i]);   //aqui vai ser comunicado ou botao1 ou botao2 para o
labview
    }
    delay(10);
  }
//Limpa a variável STRING ao verificar o termo :
  TERMO = STRING.indexOf(":");
  if (STRING.charAt(TERMO) == ':')
  {
    STRING = "";
  }

}
```

APÊNDICE D – Diagrama de blocos no Labivew para os 6 LEDs e botão




Tab Control
"Botões"
"0", Default
Boolean 7
botao:
6
1
stop
STOP
TF

APÊNDICE E – Programa no Arduino com potenciômetro.

```
// Variáveis declaradas
char CARACTER = 0;
String STRING = "";
String a = "0";
int TERMO;
// Vetores das portas de saídas e entradas
char Leds[] = {2, 4, 7, 8, 12, 13};
char botao[] = {17, 18};
char pot[] = {A0, A1};

void setup()
{
  //Configuração Serial
  Serial.begin(9600);

  //Definindo os LEDs como saídas
  pinMode(Leds[0], OUTPUT);
  pinMode(Leds[1], OUTPUT);
  pinMode(Leds[2], OUTPUT);
  pinMode(Leds[3], OUTPUT);
  pinMode(Leds[4], OUTPUT);
  pinMode(Leds[5], OUTPUT);
  //Definindo os botões como entrada
  pinMode(botao[0], INPUT);
```

```
  pinMode(botao[1], INPUT);

  //Definindo os Potenciômetros como entrada

  pinMode(pot[0], INPUT);

  pinMode(pot[1], INPUT);

}


void loop()

{

  //Coleta de dados de string na variável STRING através da comunicação serial. Os dois pontos é um termo criado para indicar finalização da comunicação serial.

  while (Serial.available() > 0)

  {

    CARACTER = (byte)Serial.read();

    STRING = STRING + CARACTER;

    TERMO = STRING.indexOf(":");

    if (STRING.charAt(TERMO) == ':')

    {

      break;

      CARACTER = 0;

    }

  }

  //Baseado na string recebida ele vai verificar se deve ligar ou desligar os LEDs


  TERMO = STRING.indexOf("L1");

  if (STRING.charAt(TERMO) == 'L')

  {
```

```
  digitalWrite(Leds[0], HIGH);
}
TERMO = STRING.indexOf("L2");
if (STRING.charAt(TERMO) == 'L')
{
  digitalWrite(Leds[1], HIGH);
}
TERMO = STRING.indexOf("L3");
if (STRING.charAt(TERMO) == 'L')
{
  digitalWrite(Leds[2], HIGH);
}
TERMO = STRING.indexOf("L4");
if (STRING.charAt(TERMO) == 'L')
{
  digitalWrite(Leds[3], HIGH);
}
TERMO = STRING.indexOf("L5");
if (STRING.charAt(TERMO) == 'L')
{
  digitalWrite(Leds[4], HIGH);
}
TERMO = STRING.indexOf("L6");
if (STRING.charAt(TERMO) == 'L')
{
```

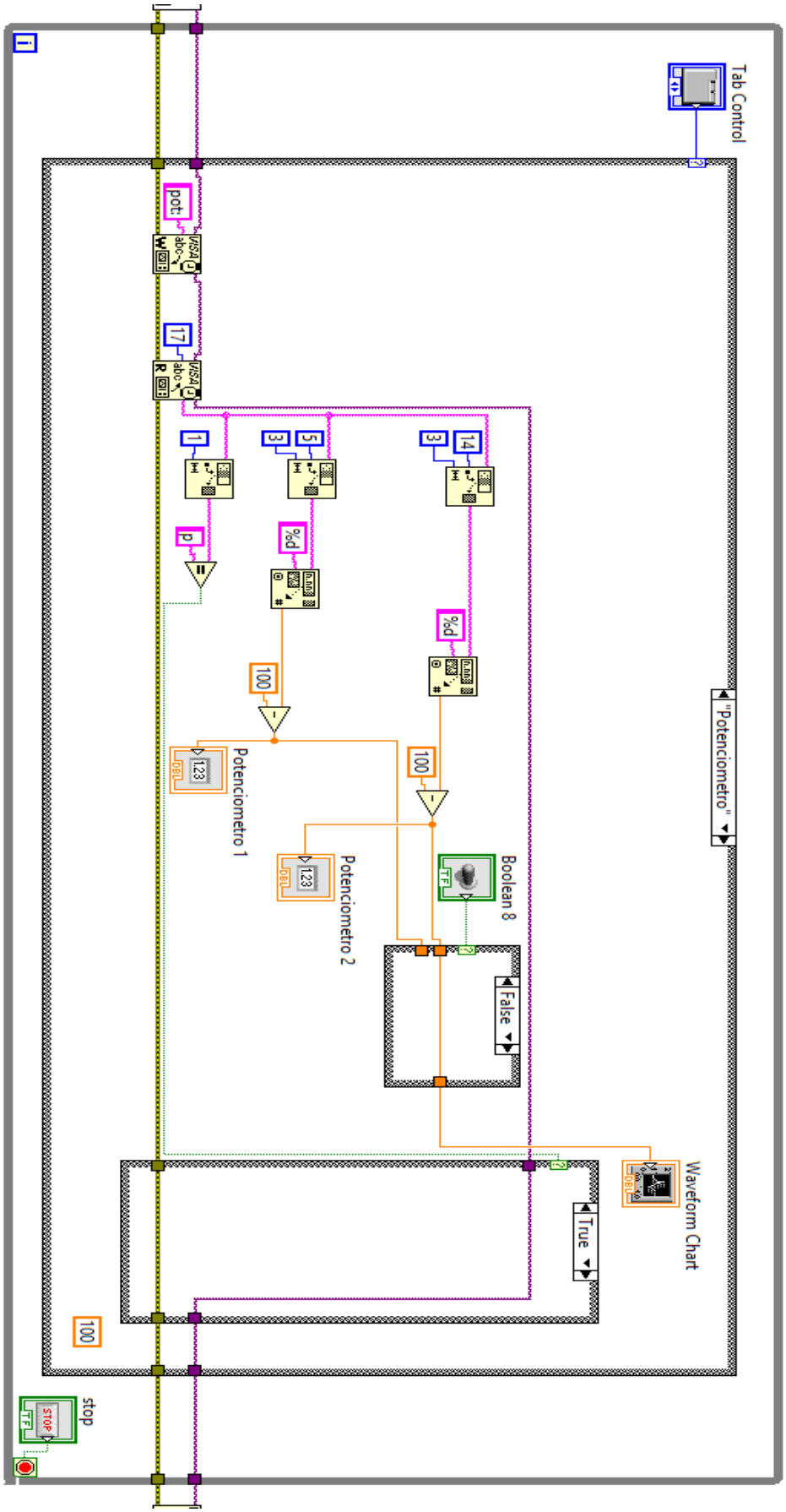
```
  digitalWrite(Leds[5], HIGH);
}
TERMO = STRING.indexOf("D1");
if (STRING.charAt(TERMO) == 'D')
{
  digitalWrite(Leds[0], LOW);
}
TERMO = STRING.indexOf("D2");
if (STRING.charAt(TERMO) == 'D')
{
  digitalWrite(Leds[1], LOW);
}
TERMO = STRING.indexOf("D3");
if (STRING.charAt(TERMO) == 'D')
{
  digitalWrite(Leds[2], LOW);
}
TERMO = STRING.indexOf("D4");
if (STRING.charAt(TERMO) == 'D')
{
  digitalWrite(Leds[3], LOW);
}
TERMO = STRING.indexOf("D5");
if (STRING.charAt(TERMO) == 'D')
{
```

```
  digitalWrite(Leds[4], LOW);
}
TERMO = STRING.indexOf("D6");
if (STRING.charAt(TERMO) == 'D')
{
  digitalWrite(Leds[5], LOW);
}
//Aqui vai ser verificado se variável STRING está escrito botão

TERMO = STRING.indexOf("botao");
if (STRING.charAt(TERMO) == 'b')
{
  a = "0";
  if (digitalRead(botao[0]) == 0)
  {
    a = "1";
  }
  if (digitalRead(botao[1]) == 0)
  {
    a = "2";
  }
   String BOTAO = "botao" + String(a);
  for (int i = 0; i < BOTAO.length(); i++)
  {
    Serial.write(BOTAO[i]);
```

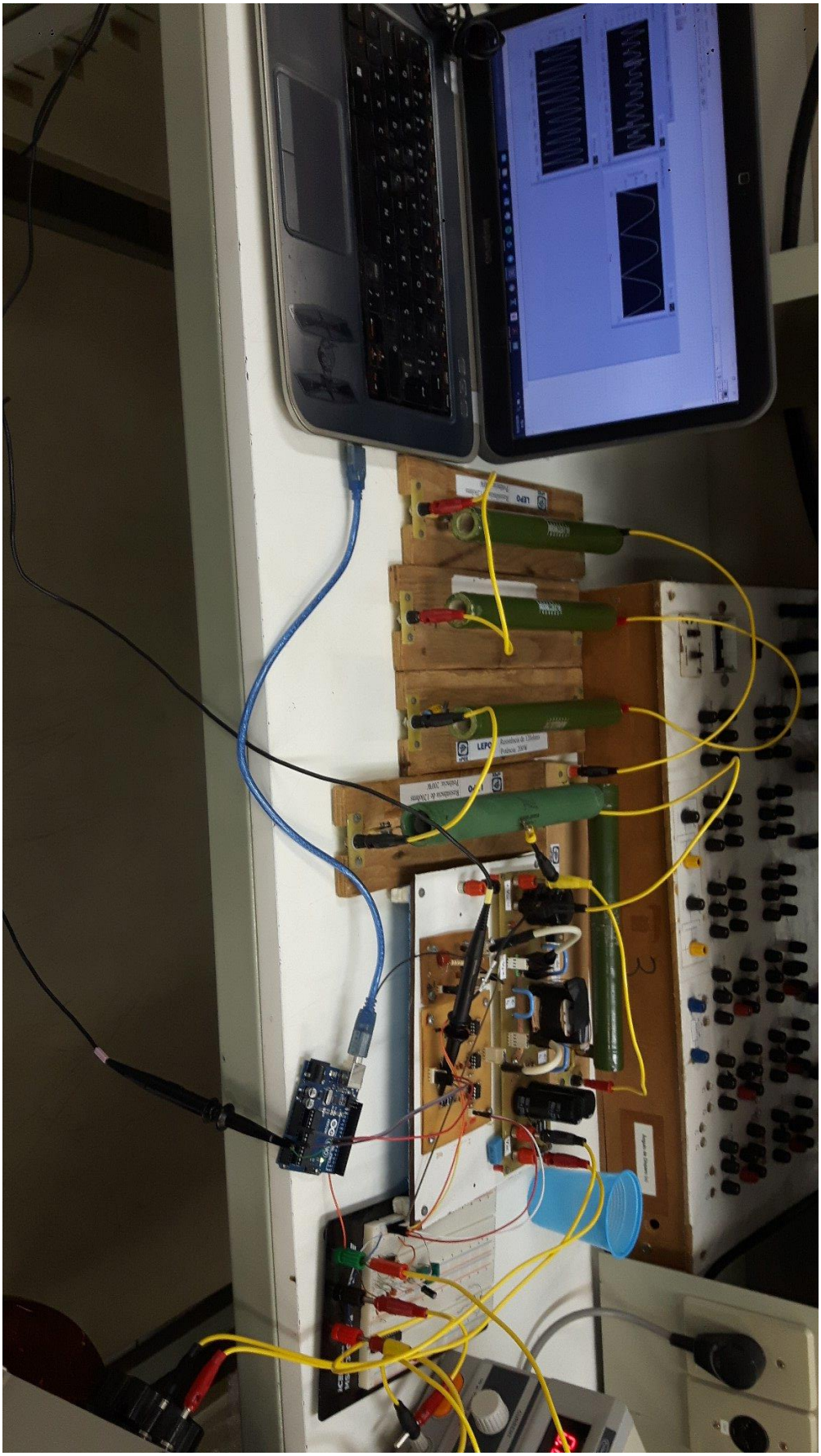
```
  }
  delay(10);
}

//Inicialmente reconhece que aba do potenciômetro foi aberta
TERMO = STRING.indexOf("pot");
if (STRING.charAt(TERMO) == 'p')
{
  //São adquirido os dados analógicos.
  int b = analogRead(pot[0]);
  int c = analogRead(pot[1]);
  //Dados são convertidos de bits para tensão com adicional de 100
  b = map(b, 0, 1023, 100, 600);
  c = map(c, 0, 1023, 100, 600);
  //São enviados as strings.
  String POTENCIOMETRO = "pot1-" + String(b) + " pot2-" + String(c) ;
  for (int i = 0; i < POTENCIOMETRO.length(); i++)
  {
    Serial.write(POTENCIOMETRO[i]);
  }
}
//Limpa a variável STRING ao verificar o termo :
TERMO = STRING.indexOf(":");
if (STRING.charAt(TERMO) == ':')
{
```

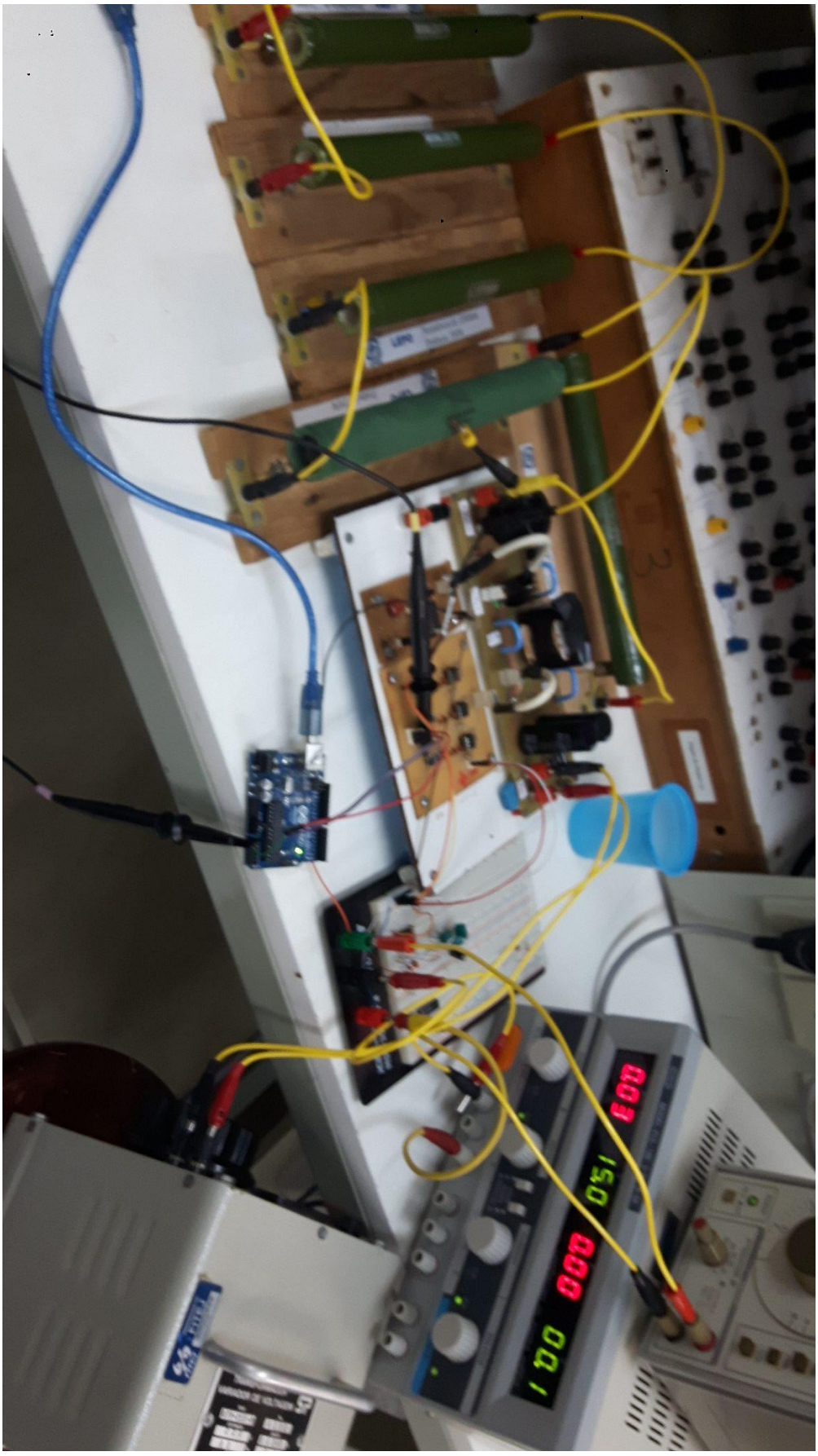
```
   STRING = "";
 }
}
```

APÊNDICE F – Aba do potenciômetro

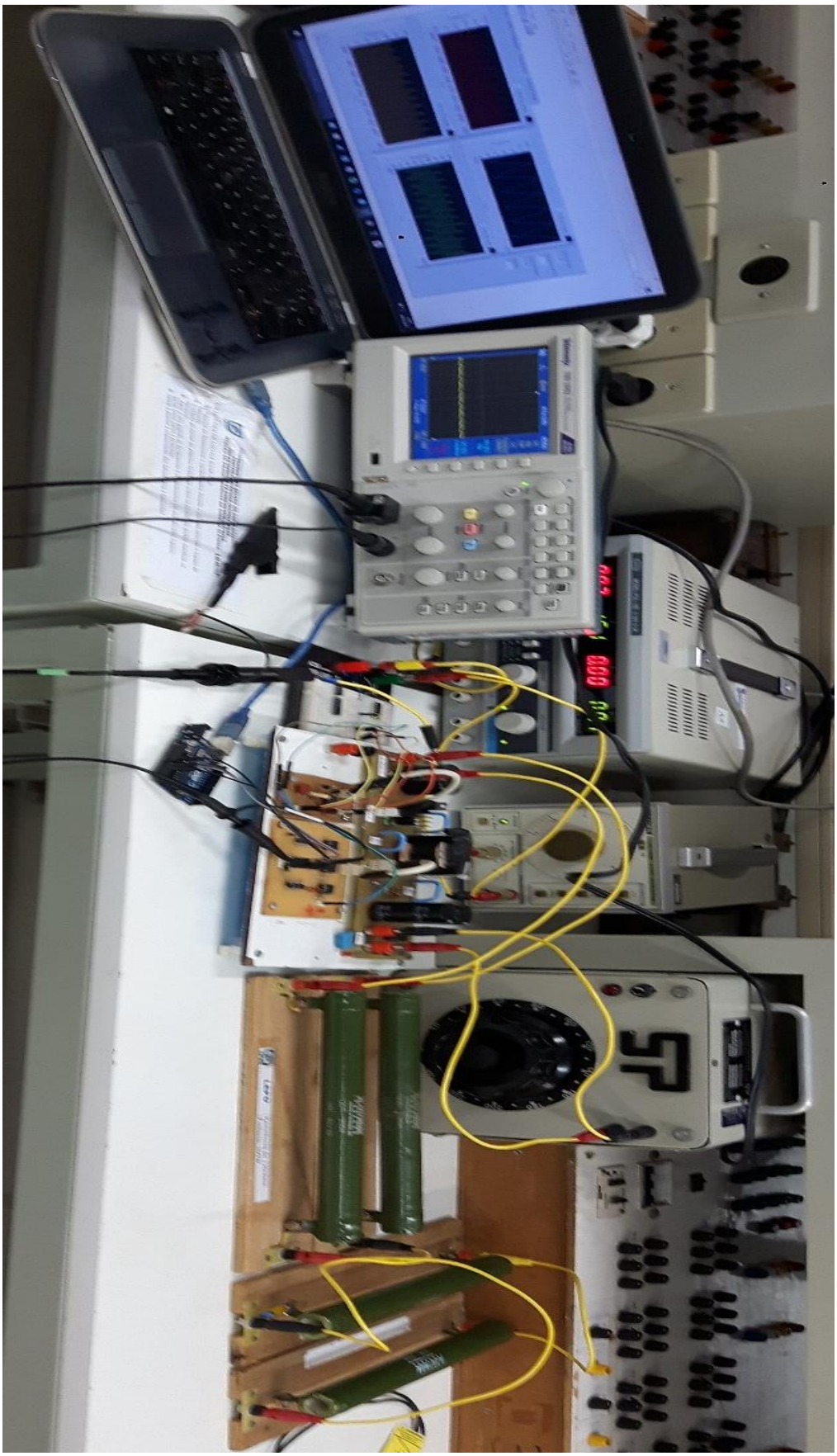


APÊNDICE G – Primeira implementação no conversor





APÊNDICE H – Segunda implementação no conversor.



APÊNDICE I – Dados obtidos dos sensores de tensão e corrente.

Time - Tensão de entrada	Amplitude - Tensão de entrada
17119	93
17119	18
17120	61
17120	1,10E+02
17121	69
17122	13
17122	81
17123	1,20E+02
17123	93
17124	22
17125	65
17125	1,20E+02
17126	67
17126	16
17127	81
17128	1,20E+02
17128	94
17129	22
17129	59
17130	1,20E+02
17131	77
17131	9,2
17132	78
17132	1,20E+02

Time - Tensão de saída	Amplitude - Tensão de saída
28456	197,468
28457	202,532
28458	194,093
28459	192,405
28460	195,781
28461	199,156
28462	199,156
28463	194,093
28464	199,156
28465	202,532
28466	192,405
28467	192,405
28468	197,468
28469	200,844
28470	197,468

28471	195,781
28472	199,156
28473	202,532
28474	194,093
28475	192,405
28476	195,781
28477	200,844
28478	197,468
28479	194,093
28480	199,156
28481	202,532
28482	194,093
28483	192,405
28484	195,781
28485	200,844
28486	199,156
28487	195,781
28488	199,156
28489	202,532
28490	194,093
28491	192,405
28492	195,781
28493	200,844
28494	199,156
28495	194,093
28496	199,156
28497	202,532
28498	194,093
28499	192,405
28500	195,781
28501	200,844
28502	199,156
28503	195,781
28504	199,156
28505	202,532
28506	192,405
28507	190,717
28508	195,781
28509	200,844
28510	199,156
28511	195,781
28512	199,156
28513	202,532
28514	194,093
28515	192,405

28516	195,781
28517	200,844
28518	199,156
28519	194,093
28520	199,156
28521	204,219
28522	194,093
28523	192,405
28524	195,781
28525	200,844
28526	199,156
28527	194,093
28528	199,156
28529	202,532
28530	194,093
28531	190,717
28532	195,781
28533	199,156
28534	199,156
28535	194,093
28536	197,468
28537	202,532
28538	194,093
28539	190,717
28540	194,093
28541	200,844
28542	199,156
28543	194,093
28544	197,468
28545	200,844
28546	194,093
28547	190,717
28548	195,781
28549	199,156
28550	200,844
28551	194,093
28552	199,156
28553	202,532
28554	194,093
28555	192,405

Time - Corrente de entrada	Amplitude - Corrente de entrada
28457	1,77E+00
28458	1,93E+00

28459	0,00E+00
28460	0,00E+00
28461	1,04E+00
28462	1,79E+00
28463	0,00E+00
28464	0,00E+00
28465	1,73E+00
28466	1,94E+00
28467	1,93E-02
28468	0,00E+00
28469	1,08E+00
28470	1,85E+00
28471	0,00E+00
28472	0,00E+00
28473	1,69E+00
28474	2,04E+00
28475	6,77E-02
28476	0,00E+00
28477	9,67E-01
28478	1,84E+00
28479	5,80E-02
28480	0,00E+00
28481	1,62E+00
28482	2,02E+00
28483	0,00E+00
28484	0,00E+00
28485	9,38E-01
28486	1,91E+00
28487	7,74E-02
28488	0,00E+00
28489	1,62E+00
28490	2,11E+00
28491	1,93E-02
28492	0,00E+00
28493	8,12E-01
28494	1,85E+00
28495	0,00E+00
28496	0,00E+00
28497	1,51E+00
28498	2,13E+00
28499	0,00E+00
28500	0,00E+00
28501	7,93E-01
28502	1,85E+00
28503	1,93E-02

28504	0,00E+00
28505	1,51E+00
28506	2,20E+00
28507	0,00E+00
28508	0,00E+00
28509	7,54E-01
28510	1,86E+00
28511	6,77E-02
28512	0,00E+00
28513	1,36E+00
28514	2,13E+00
28515	0,00E+00
28516	0,00E+00
28517	6,67E-01
28518	1,89E+00
28519	0,00E+00
28520	0,00E+00
28521	1,38E+00
28522	2,26E+00
28523	7,74E-02
28524	0,00E+00
28525	6,96E-01
28526	1,89E+00
28527	0,00E+00
28528	2,90E-02
28529	1,36E+00
28530	2,28E+00
28531	3,87E-02
28532	0,00E+00
28533	5,51E-01
28534	1,90E+00
28535	0,00E+00
28536	0,00E+00
28537	1,21E+00
28538	2,29E+00
28539	0,00E+00
28540	0,00E+00
28541	5,42E-01
28542	1,95E+00
28543	0,00E+00
28544	0,00E+00
28545	1,13E+00
28546	2,28E+00
28547	9,67E-03
28548	0,00E+00

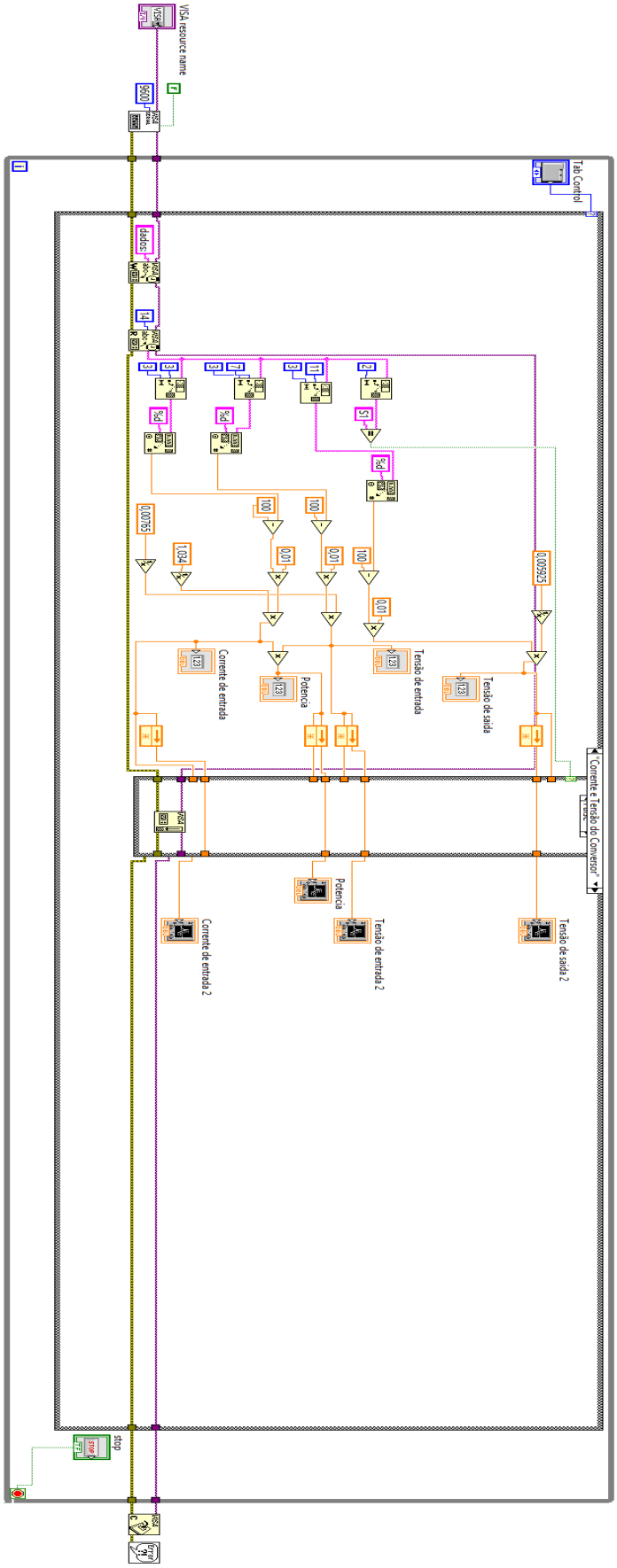
28549	4,64E-01
28550	1,85E+00
28551	6,77E-02
28552	0,00E+00
28553	1,05E+00
28554	2,30E+00
28555	0,00E+00

APÊNDICE J – Programa para os sensores no Arduino.

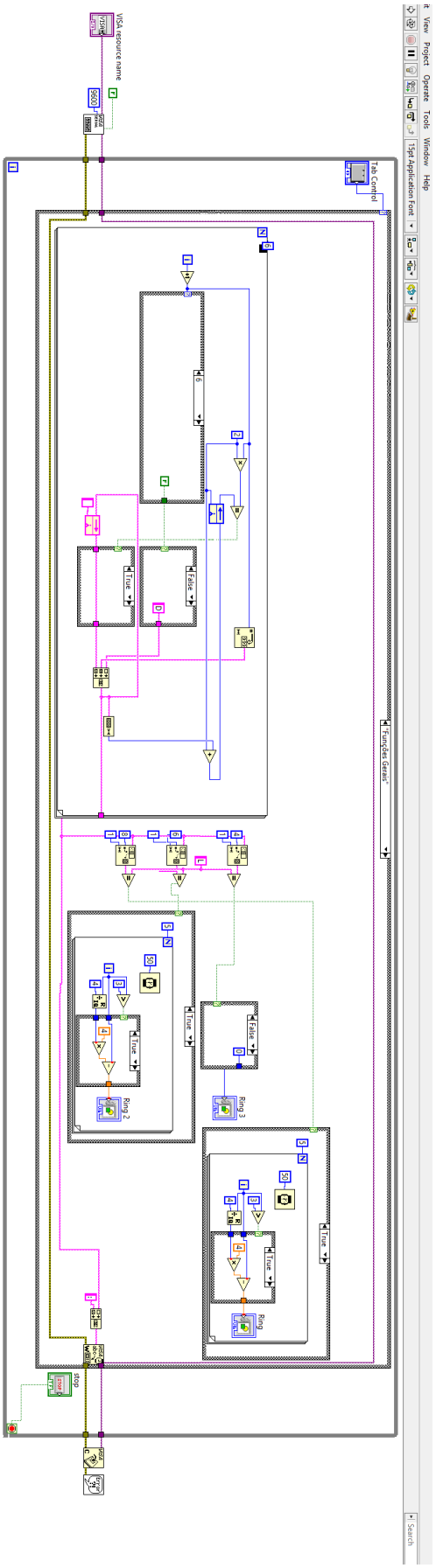
```
// Variáveis declaradas
char CARACTER = 0;
String STRING = "";
String a = "0";
int TERMO;
// Vetores das portas de saídas e entradas
char pot[] = {A0, A1,A2};


void setup()
{
  //Configuração Serial
  Serial.begin(9600);
  pinMode(pot[0], INPUT);
  pinMode(pot[1], INPUT);
  pinMode(pot[2], INPUT);
}


void loop()
{
  //Coleta de dados de string na variável STRING através da comunicação serial. Os dois pontos é um termo criado para indicar finalização da comunicação serial.
  while (Serial.available() > 0)
  {
    CARACTER = (byte)Serial.read();
    STRING = STRING + CARACTER;
    TERMO = STRING.indexOf(":");
    if (STRING.charAt(TERMO) == ':')
    {
```

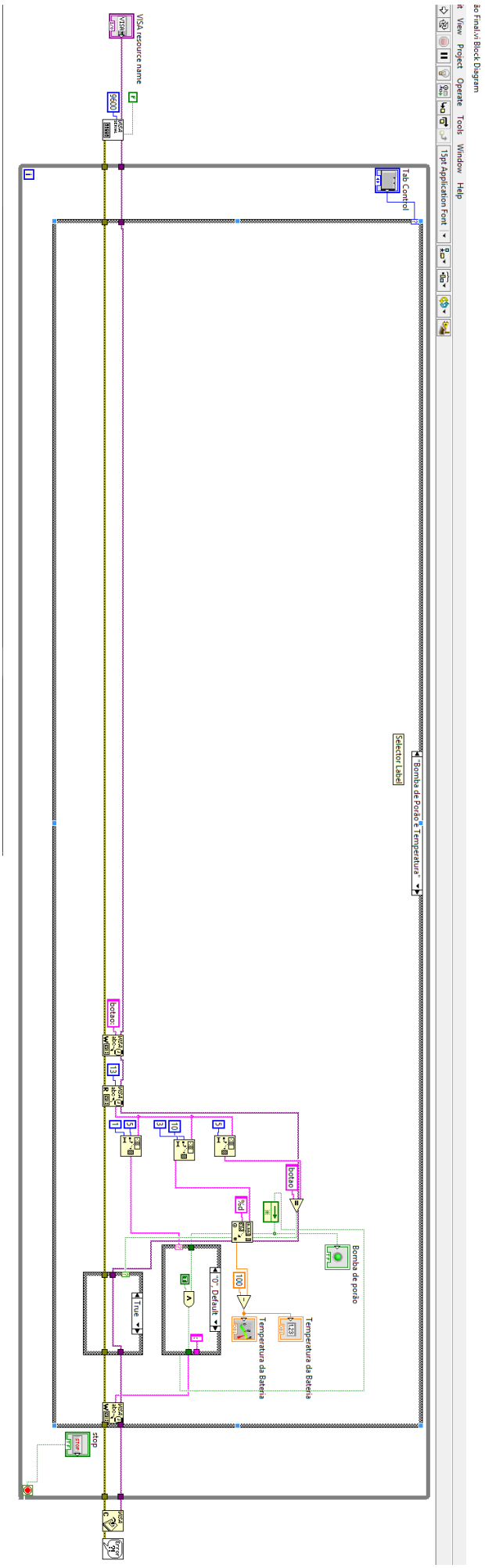
```
      break;
      CARACTER = 0;
    }
  }
// Coleta de dados dos sensores convertidos de 0 -5V e enviados para Labview
  TERMO = STRING.indexOf("dados");
  if (STRING.charAt(TERMO) == 'd')
  {
    int e = analogRead(pot[0]);
    int f = analogRead(pot[1]);
    int g = analogRead(pot[2]);
    e = map(e, 0, 1023, 100, 600);
    f = map(f, 0, 1023, 100, 600);
    g = map(g, 0, 1023, 100, 600);
    String DADOS = "S1-" + String(e) + "-" + String(f) + "-" + String(g) ;
    for (int i = 0; i < DADOS.length(); i++)
    {
      Serial.write(DADOS[i]);
    }
  }
  //Limpa a variável STRING ao verificar o termo :
  TERMO = STRING.indexOf(":");
  if (STRING.charAt(TERMO) == ':')
  {
    STRING = "";
  }
}
```

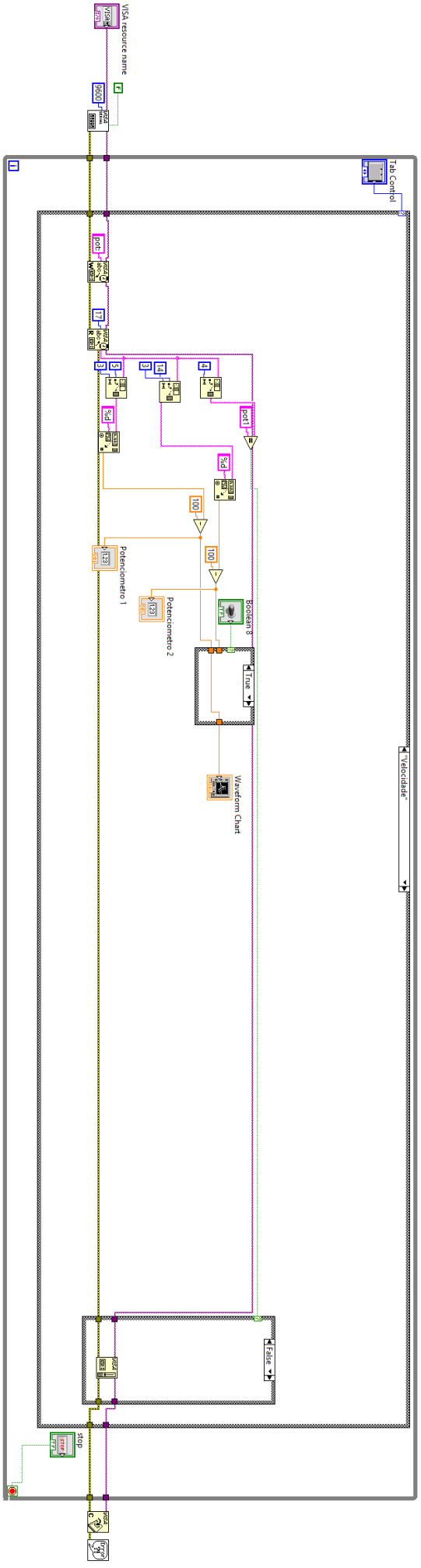
APÊNDICE K – Aba dos sensores

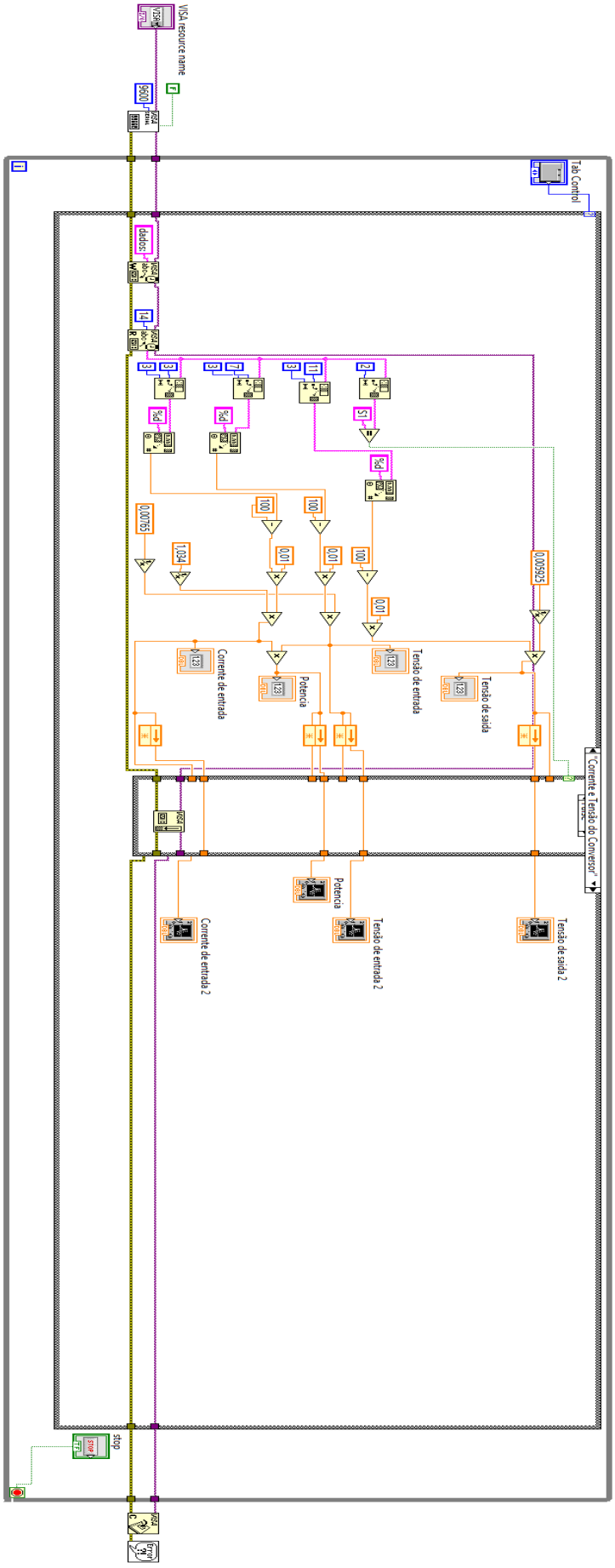


APÊNDICE L – Versão final dos diagramas de blocos




ão Final.vi Block Diagram
it View Project Operate Tools Window Help
15pt Application Font
VISA resource name
9600
Tab Control
Selector Label
"Bomba de Poço" "Temperatura"
botao
13
5
10
2
Default
True
100
Bomba de poço
Temperatura da Bateria
Temperatura da Bateria
stop


VISA resource name
9600
Tab Control
pot1
%d
100
Potenciometro 1
Potenciometro 2
Boolean 8
True
False
Waveform Chart
"Velocidade"
stop


VISA resource name
9600
Tab Control
dados
14
0,00785
1,034
100
0,01
0,009925
Corrente de entrada
Potencia
Tensão de entrada
Tensão de saída
Corrente e Tensão do Conversor
Corrente de entrada 2
Potencia
Tensão de entrada 2
Tensão de saída 2
stop

APÊNDICE M – Versão final do programa no Arduino

```
// Variáveis declaradas

char CARACTER = 0;

String STRING = "";

String a = "0";

int TERMO;

// Vetores das portas de saídas e entradas

char Leds[] = {2, 4, 7, 8, 12, 13};

char botao[] = {17, 18, 19};

char pot[] = {A0, A1, A2, A3};


void setup()

{

  //Configuração Serial

  Serial.begin(9600);


  //Definindo os LEDs como saídas

  pinMode(Leds[0], OUTPUT);

  pinMode(Leds[1], OUTPUT);

  pinMode(Leds[2], OUTPUT);

  pinMode(Leds[3], OUTPUT);

  pinMode(Leds[4], OUTPUT);

  pinMode(Leds[5], OUTPUT);
```

```
  //Definindo os botões como entrada
  pinMode(botao[0], INPUT);
  pinMode(botao[1], INPUT);
  pinMode(botao[2], INPUT);
  //Definindo os Potenciômetros como entrada
  pinMode(pot[0], INPUT);
  pinMode(pot[1], INPUT);
  pinMode(pot[2], INPUT);
}

void loop()
{
  //Coleta de dados de string na variável STRING através da comunicação serial. Os dois pontos é um termo criado para indicar finalização da comunicação serial.
  while (Serial.available() > 0)
  {
    CARACTER = (byte)Serial.read();
    STRING = STRING + CARACTER;
    TERMO = STRING.indexOf(":");
    if (STRING.charAt(TERMO) == ':')
    {
      break;
      CARACTER = 0;
```

```
  }
}
//Baseado na string recebida ele vai verificar se deve ligar ou desligar os LEDs


TERMO = STRING.indexOf("L1");
if (STRING.charAt(TERMO) == 'L')
{
  digitalWrite(Leds[0], HIGH);
}
TERMO = STRING.indexOf("L2");
if (STRING.charAt(TERMO) == 'L')
{
  digitalWrite(Leds[1], HIGH);
}
TERMO = STRING.indexOf("L3");
if (STRING.charAt(TERMO) == 'L')
{
  digitalWrite(Leds[2], HIGH);
}
TERMO = STRING.indexOf("L4");
if (STRING.charAt(TERMO) == 'L')
{
  digitalWrite(Leds[3], HIGH);
```

```
}
TERMO = STRING.indexOf("L5");
if (STRING.charAt(TERMO) == 'L')
{
  digitalWrite(Leds[4], HIGH);
}
TERMO = STRING.indexOf("L6");
if (STRING.charAt(TERMO) == 'L')
{
  digitalWrite(Leds[5], HIGH);
}
TERMO = STRING.indexOf("D1");
if (STRING.charAt(TERMO) == 'D')
{
  digitalWrite(Leds[0], LOW);
}
TERMO = STRING.indexOf("D2");
if (STRING.charAt(TERMO) == 'D')
{
  digitalWrite(Leds[1], LOW);
}
TERMO = STRING.indexOf("D3");
if (STRING.charAt(TERMO) == 'D')
```

```
{
  digitalWrite(Leds[2], LOW);
}
TERMO = STRING.indexOf("D4");
if (STRING.charAt(TERMO) == 'D')
{
  digitalWrite(Leds[3], LOW);
}
TERMO = STRING.indexOf("D5");
if (STRING.charAt(TERMO) == 'D')
{
  digitalWrite(Leds[4], LOW);
}
TERMO = STRING.indexOf("D6");
if (STRING.charAt(TERMO) == 'D')
{
  digitalWrite(Leds[5], LOW);
}
//Aqui vai ser verificado se variável STRING está escrito botão


TERMO = STRING.indexOf("botao");
if (STRING.charAt(TERMO) == 'b')
```

```
  {
    a = "0";
    if (digitalRead(botao[0]) == 0)
    {
      a = "1";
    }
    if (digitalRead(botao[1]) == 0)
    {
      a = "2";
    }
    int Temperatura = analogRead(pot[2]);//como ele varia até 1,5 volts então vai até 307 bits
    Temperatura = map(Temperatura, 0, 307, 102, 250);
    String BOTAO = "botao" + String(a) + "Temp" + String(Temperatura);
    for (int i = 0; i < BOTAO.length(); i++)
    {
      Serial.write(BOTAO[i]);
    }
    delay(10);
  }

  //Inicialmente reconhece que aba do potenciômetro foi aberta
  TERMO = STRING.indexOf("pot");
```

```
if (STRING.charAt(TERMO) == 'p')
{
  //São adquirido os dados analógicos.
  int b = analogRead(pot[0]);
  int c = analogRead(pot[1]);
  //Dados são convertidos de bits para tensão com adicional de 100
  b = map(b, 0, 1023, 100, 600);
  c = map(c, 0, 1023, 100, 600);
  String POTENCIOMETRO = "pot1-" + String(b) + " pot2-" + String(c) ;
  for (int i = 0; i < POTENCIOMETRO.length(); i++)
  {
    Serial.write(POTENCIOMETRO[i]);
  }
}
// Coleta de dados dos sensores convertidos de 0 -5V e enviados para Labview
TERMO = STRING.indexOf("dados");
if (STRING.charAt(TERMO) == 'd')
{
  int e = analogRead(pot[0]);
  int f = analogRead(pot[1]);
  int g = analogRead(pot[2]);
  e = map(e, 0, 1023, 100, 600);
  f = map(f, 0, 1023, 100, 600);
```

```
  g = map(g, 0, 1023, 100, 600);
  String DADOS = "S1-" + String(e) + "-" + String(f) + "-" + String(g) ;
  for (int i = 0; i < DADOS.length(); i++)
  {
    Serial.write(DADOS[i]);
  }
}

//Limpa a variável STRING ao verificar o termo :
TERMO = STRING.indexOf(":");
if (STRING.charAt(TERMO) == ':')
{
  STRING = "";
}
}
```