# SISTEMA DE MEDIÇÃO E CONTROLE DE TEMPERATURA APLICADO EM UM TÚNEL DE VENTO

Sandro Santoro Rezende

"PROJETO SUBMETIDO AO CORPO DOCENTE DO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO ELETRICISTA"

Aprovado por:

_____

Prof. José Luiz da Silva Neto, Ph.D.

(Orientador)

_____

Prof. Luís Guilherme Barbosa Rolim, Ph.D.

_____

Prof. Cesar Cosenza de Carvalho, D.Sc.

_____

Juliana Braga Rodrigues Loureiro, M. Sc.

RIO DE JANEIRO, RJ – BRASIL

MARÇO DE 2007

# Resumo

Em aplicações industriais e laboratoriais um fator que às vezes é muito importante é a temperatura, seja para promover reações químicas, medições de outras grandezas, etc... A temperatura tem que ser não apenas conhecida, mas também estável.

O Laboratório de Mecânica de Turbulência da UFRJ adquiriu um equipamento baseado em túnel de vento aquecido e sistema de controle de temperatura, porém este sistema não estava funcionando dentro das exigências do laboratório e não havia muita informação sobre o funcionamento do equipamento. Observou-se experimentalmente que este trabalhava em um regime tipo ligado-desligado, isto é, caso a temperatura estivesse abaixo da especificada ele acionava o aquecimento do ar no túnel. O problema é que devido a constante de tempo térmica ser muito elevada, esse sistema não fornece uma saída estável.

Este projeto se propõe a desenvolver um sistema de controle baseado em um sinal PWM utilizando o controle PI (proporcional-integrador) de forma a obter uma saída de temperatura do túnel de vento controlada, isto é, com baixa oscilação de temperatura e de baixo custo de forma a atingir as exigências do laboratório. O sinal PWM permite um controle eficiente do aquecimento atuando em freqüência maior que a da rede e a técnica PI permite que o sistema se ajuste a cada instante para garantir a temperatura desejada. Este sistema é utilizado em conjunto com um sensor de temperatura eletrônico. As vantagens desse tipo de sensor em relação a outros métodos de medição de temperatura de um escoamento são: fácil implementação e resposta rápida que já pode ser capturada por um dispositivo eletrônico para posterior análise, pois a faixa de tensão de saída utilizada para o projeto varia de 0 a 5 Volts.

# Índice

# Índice de figuras

# 1 – Introdução

## *1.1 - Motivação*

O laboratório de turbulência da UFRJ adquiriu um sistema de controle de temperatura para ser usado em conjunto com o túnel de vento, mas, após diversas tentativas verificou-se que o sistema apresentava uma grande oscilação de temperatura, por exemplo, quando a temperatura selecionada era de 40°C, o sistema apresentava uma saída que variava de 40°C até um valor pouco acima de 60°C. Com isso surgiu a necessidade de se desenvolver um projeto de baixo custo e fácil implantação para substituir o sistema anterior e aproveitando os recursos já disponíveis no laboratório.

## *1.2 - Objetivo*

Desenvolver um sistema baseado em microcontrolador utilizando a técnica PWM e um controle PI (proporcional-integrador) que tem por objetivo garantir uma temperatura definida e estável na saída de um túnel de vento, com o auxilio de um computador para a entrada da temperatura e visualização da temperatura medida, para ser usado como instrumento de calibração.

## *1.3 – Dispositivos utilizados*

O projeto possui quatro conjuntos de equipamentos distintos:

- Túnel de vento
- Computador
- Sistema de controle baseado em microcontrolador
- Relé de Estado Sólido
- Sensor

O diagrama do projeto pode ser visto na figura 1.3.1.

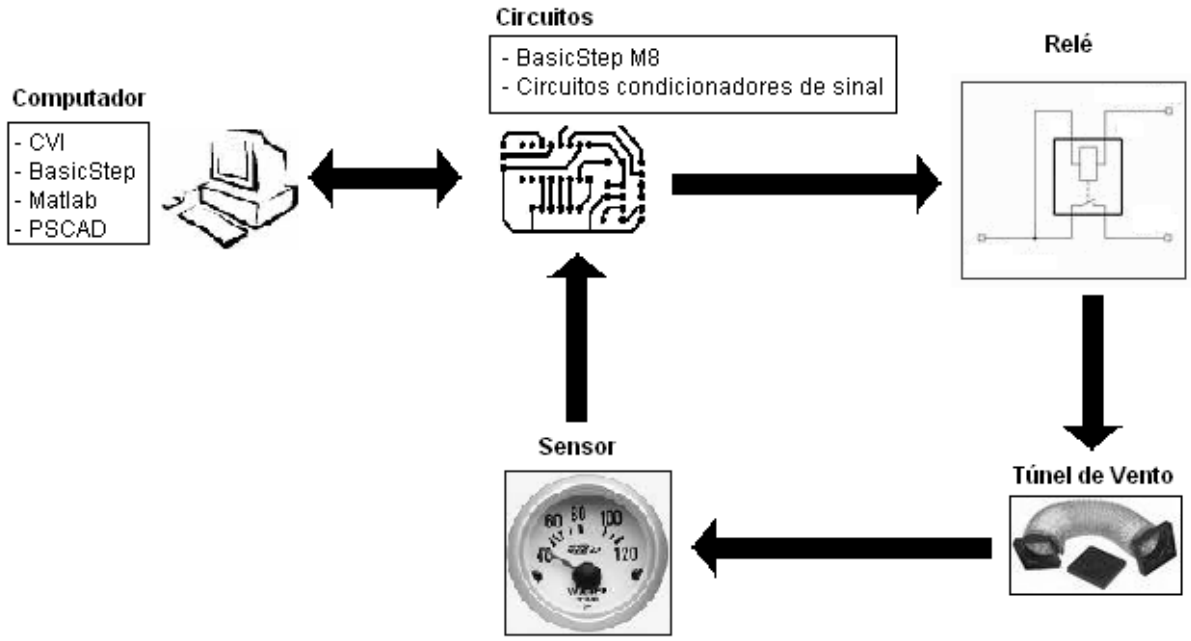



**Figura 1.3.1 – Diagrama do Sistema**

## *1.4 - O túnel de vento*

O túnel de vento utilizado é composto por um ventilador com filtro de ar confeccionado pelo próprio pessoal do laboratório, uma lona de acoplamento entre o ventilador e o tubo, um tubo com uma serpentina de dois filamentos internos para promover o aquecimento do ar e um bocal de saída projetado para garantir uma saída uniforme do jato de ar aquecido.

O ventilador é acionado por um controle de variação de freqüência (inversor) e este por sua vez é ligado à rede (220V).

A serpentina é ligada ao circuito de controle e alimentada pela tensão da rede. A serpentina possui uma resistência de 11Ω, medida experimentalmente.

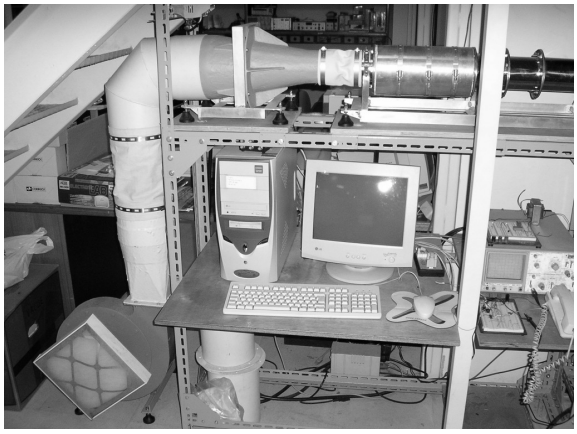
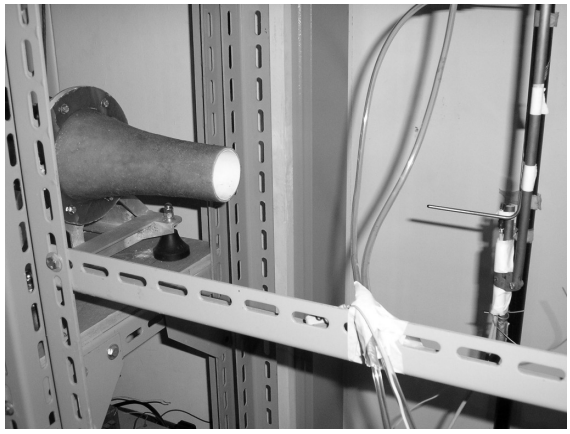



**Figura 1.4.1 – O túnel de vento (esquerda) / Detalhe do bocal (direita)**

## *1.5 - Descrição dos capítulos seguintes*

No capítulo 2 abordamos o microcontrolador, falando sobre suas principais características, recursos relevantes ao projeto e o software utilizado para fazer a gravação do *firmware*.

No capítulo 3 serão abordados os tópicos referentes aos circuitos de apoio ao microcontrolador, são estes: o sensor, o relé de estado sólido, o amplificador de sinal e o regulador de tensão.

No capítulo 4 serão abordados os tópicos referentes a determinação dos parâmetros utilizados no controle e a técnica de controle utilizada.

Os resultados podem ser vistos no capítulo 5, onde também apresentamos o software para uso pelo usuário final e o programa gravado no BasicStep M8 onde é implementado o controle.

No capítulo 6 temos uma avaliação dos resultados do projeto.

Para finalizar temos a bibliografia utilizada para a execução do projeto, onde estão listados os *datasheets* e livros utilizados.

# 2 – O Microcontrolador

## *2.1 - Surgimento dos microcontroladores*

Os microcontroladores surgiram através do desenvolvimento da tecnologia dos circuitos integrados. Este desenvolvimento tornou possível armazenar centenas de milhares de transistores num único circuito integrado. Isso constituiu um pré-requisito para a produção de microprocessadores e, os primeiros computadores foram construídos adicionando periféricos externos tais como memória, linhas de entrada e saída, temporizadores e outros. Um crescente aumento do nível de integração permitiu o aparecimento de circuitos integrados contendo simultaneamente processador e periféricos.

Foi assim que o primeiro *chip* contendo um microcomputador e que mais tarde haveria de ser designado por microcontrolador, apareceu.

## *2.2 - O microcontrolador*

O microcontrolador definido para o projeto foi o BasicStep M8, revendido pela TATO, este microcontrolador é o Atmega8L desenvolvido pela *Atmel Corporation* [7]. O adaptador para a conexão do cabo serial foi desenvolvido pela própria TATO. Este microcontrolador já estava disponível para uso do laboratório e possuía as características necessárias para o desenvolvimento do projeto.

### 2.2.1 - Algumas características do BasicStep M8

O BasicStep M8 usa a tecnologia de baixo consumo CMOS baseada na arquitetura AVR RISC. A figura 2.2.1.1 representa o diagrama de blocos do BasicStep M8.

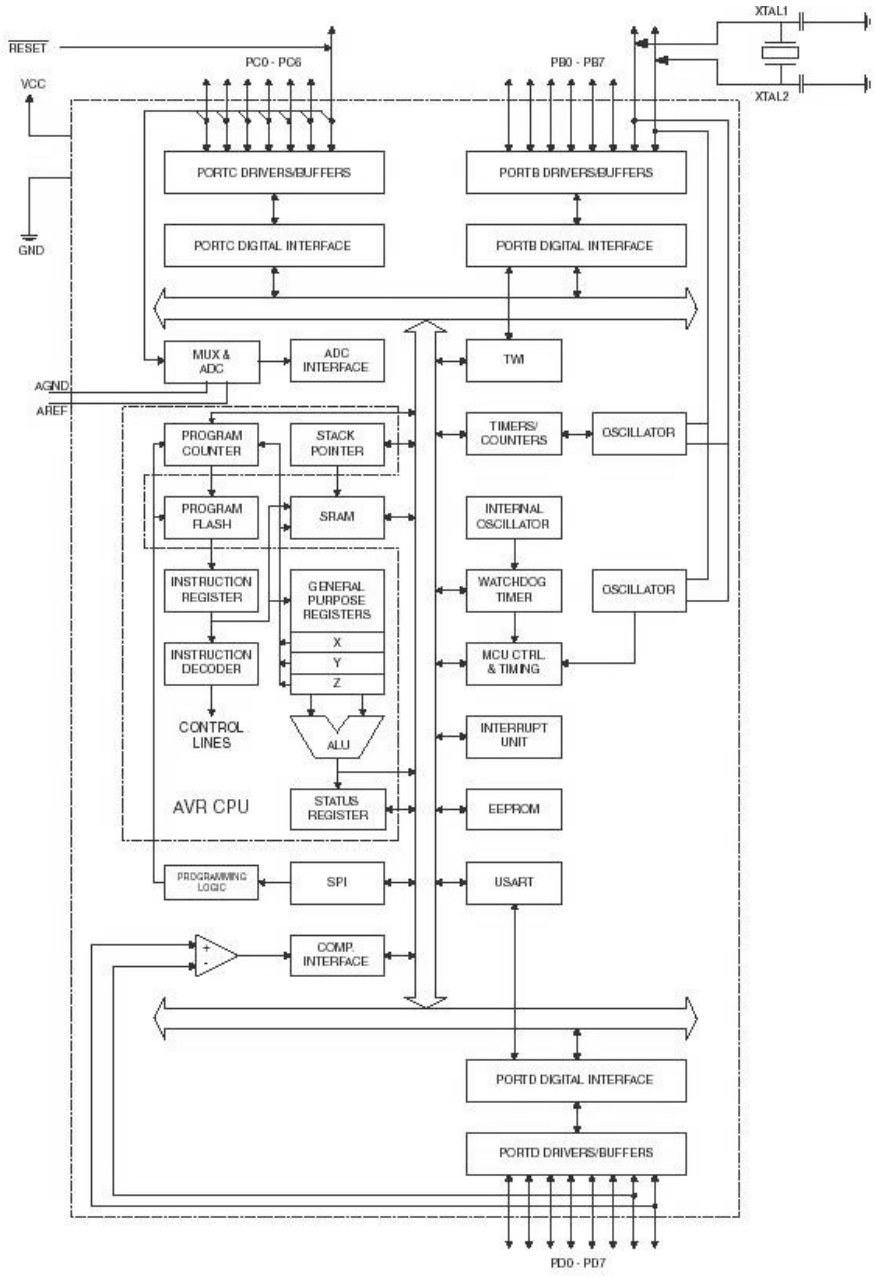



**Figura 2.2.1.1 – Diagrama do BasicStep M8**

O BasicStep M8 é um microcontrolador que possui diversas características que simplificaram bastante o desenvolvimento da parte técnica do projeto.
Sua velocidade é de 2MIPs e 8MHz de freqüência de operação o que gera resultados mais confiáveis e precisos.

O BasicStep M8 possui três tipo de memória. Uma delas é composta por uma série de registros. Esta memória é dividida na memória do programa (onde as instruções de programa são armazenadas) e memória dos dados (onde algumas das variáveis são armazenadas). Quando um programa é carregado no microcontrolador, fica armazenado na memória do programa, e quando o programa está funcionando, usa a memória dos dados

como um espaço para armazenar e manipular suas variáveis. A memória dos dados possui registros especiais onde a informação sobre as funções do *chip* são armazenadas. Cada função do *chip* terá um registro especial da função (ou diversos registros) associado com ela: a porta serial, os conversores analógico digitais, os temporizadores, e talvez mais importante, os pinos de entrada e saída. A outra é uma memória RAM que serve para o armazenamento de certos tipos de variáveis e a terceira, também destinada ao armazenamento de variáveis é uma memória EEPROM.

O microcontrolador dispõe de 8kb de memória interna disponíveis para o programa e as variáveis armazenadas em registradores, 1024 bytes de memória RAM e 512 bytes de memória EEPROM, 23 linhas de E/S, 32 registradores de uso geral, 3 Timers/Contadores, controle de interrupção externa e interna, uma interface serial programável entre outras características.

As variáveis do programa podem ser armazenadas nos três tipos de memória citados anteriormente dependendo do tipo de variável e da escolha feita pelo programador, o tipo de variável é definida pelo último *caracter* em seu nome. As variáveis do tipo BYTE podem ser armazenadas em registradores (memórias de acesso rápido) ou em RAM (memórias de acesso mais lento), as variáveis do tipo INTEGER são do tipo *unsigned integer* e são armazenadas em RAM, utilizando dois bytes, as variáveis do tipo STRING são armazenadas em EEPROM e as variáveis do tipo matriz de BYTES são armazenadas em RAM.

Uma das grandes vantagens dos microcontroladores da série BasicStep é a presença do circuito exclusivo de comunicação serial seguindo os padrões RS-232, que é o mesmo padrão adotado nos computadores, assim, não é necessário nenhum circuito externo para o tratamento do sinal a ser enviado ou recebido pela porta serial. Este conector é usado em conjunto com um cabo de comunicação serial fornecido também pela própria TATO, este cabo garante maior qualidade na comunicação serial do microcontrolador, até o momento este é o único microcontrolador da séria BasicStep que possui essa característica.

Outra característica importante neste BasicStep é a presença de três canais PWM, pois o controle foi desenvolvido aplicando um sinal PWM na resistência do túnel de vento.

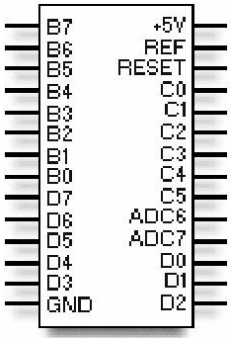



**Figura 2.2.1.2 – Pinagem do BasicStep M8**

Através da figura 2.2.1.2 que mostra os terminais do BasicStep M8 podemos visualizar os pinos descritos como B7, B6, B5, etc.., estes se referem às portas de entrada e saída (E/S). No BasicStep M8, há três portas de E/S, portas B, C e D. Dentro de cada porta, são dados números para os pinos. Para a porta B, os pinos vão de B0 a B7. A porta B e D são portas de E/S bi-direcional de 8bits.

A porta C pode assumir funções diferentes dependendo da necessidade, possibilitando o acesso dos blocos periféricos ao meio externo. Por exemplo, o conversor analógico digital pode utilizar um dos pinos da Porta C como leitor de entrada de dados a serem convertidos ao invés da função padrão de E/S.

A porta C possui dois pinos exclusivos para a função de conversor analógico digital, ADC6 e ADC7.

O pino de tensão de referência para o conversor analógico digital (VREF) determina o valor máximo que será medido e convertido pelo conversor, podendo ser no máximo de 5V.

O BasicStep M8 segue os padrões TTL de operação, logo, o CI opera com uma tensão de 5V aplicada ao pino +5V e o pino GND deve estar conectado ao terra do sistema.

Um botão tipo *push-pull* faz a conexão entre o pino de *RESET* e o terra do sistema, quando este é pressionado e solto o microcontrolador reinicia a sua operação, perdendo qualquer informação previamente armazenada.

Uma característica interessante observada durante o desenvolvimento do projeto foi que quando o cabo de comunicação serial estava conectado ao computador e este estava ligado, o microcontrolador entrava no estado de RESET, era necessário iniciar o programa BasicStep ou o CVI para fazer com que o microcontrolador operasse. Sem o cabo de comunicação o microcontrolador ficava operando continuamente. Outra característica é que para garantir que o programa fosse gravado com sucesso no BasicStep M8 era necessário definir a velocidade de operação da porta serial do computador para 2400bps.

Uma outra característica que foi importante no aspecto de simplificação da parte técnica do projeto foi que o BasicStep possui um controlador de interrupção interno facilitando bastante a transferência do dado no sentido do computador para o microcontrolador.

O conversor analógico digital pode trabalhar de várias maneiras diferentes, neste projeto foi utilizado o modo IDLE, pois este interrompe o microcontrolador colocando-o em modo SLEEP para que a leitura analógica seja feita e convertida, isso garante que o valor da medida estará livre de ruídos originados pelo próprio microcontrolador. Os outros modos de operação são RUN (habilita o modo continuo), INTERRUPT (gera um sinal de interrupção no programa) e IR (gera um sinal de interrupção e habilita o modo continuo).

## 2.2.2 - Conversor analógico - digital

Os sinais retirados do sistema controlado são diferentes daqueles que o microcontrolador pode entender (zero e um), por isto, estes devem ser convertidos num formato que possa ser compreendido pelo microcontrolador. Esta tarefa é executada por intermédio de um bloco destinado à conversão analógico-digital. Este bloco vai ser responsável pela conversão de uma informação de valor analógico para um número binário.

Além da precisão, outro parâmetro importante é a taxa de amostragem, que é a quantidade de vezes por unidade de tempo em que o sinal analógico é medido e convertido.

Quando um sinal é convertido com uma alta taxa de amostragem maior será a sua semelhança com o sinal original. A figura 2.2.2.1 mostra o sinal analógico original à esquerda e sua representação já convertida a direita. Os pontos indicam as posições em que o sinal foi medido e convertido.

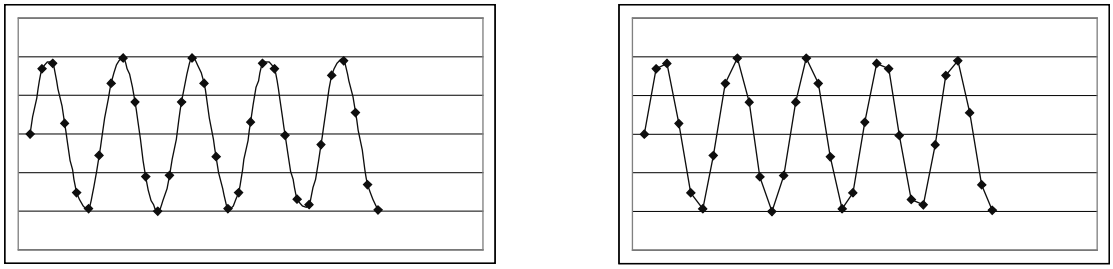


**Figura 2.2.2.1 - Sinal original (esquerda) / Sinal convertido (direita)**

Para poder medir um sinal analógico a taxa de amostragem deve ser pelo menos o dobro da freqüência do sinal medido, caso contrário vales ou picos deste sinal serão perdidos causando uma conversão falha. A figura 2.2.2.2 exemplifica o efeito da perda de

informações do sinal devido à baixa taxa de amostragem, o sinal originalmente senoidal à esquerda e sua reprodução convertida com perda de informação.

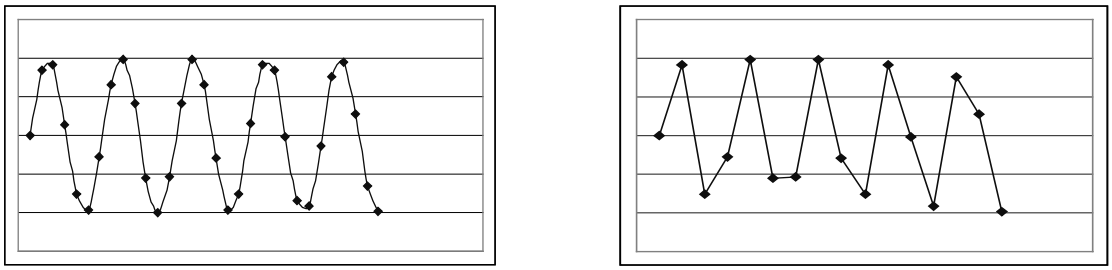


**Figura 2.2.2.2 - Sinal original (esquerda) / Sinal convertido (direita)**

O conversor tem acesso ao meio externo por meio da porta C. Este projeto requer o uso de apenas uma entrada analógica, no caso foi escolhido o pino 5 da porta C. Esta entrada será usada para converter os sinais enviados pelo sensor de temperatura.

Para definirmos a taxa de amostragem deste projeto precisamos levar em consideração algumas características inerentes ao projeto. Como este projeto trabalha com variações de temperatura e esta grandeza é muito mais lenta do que as grandezas elétricas e como a constante do sistema térmico é na ordem de algumas dezenas de segundos, a taxa de amostragem foi definida como sendo 1 segundo. Em função da constante de tempo do sistema ser bastante elevada, o tempo de conversão do sinal analógico para digital é desprezível e também o tempo de execução do próprio programa gravado no microcontrolador e o tempo de transferência dos dados do microcontrolador para o computador através da porta serial não possuem grande influência no controle.

O BasicStep M8 possui um conversor analógico digital com resolução de dez bits, que corresponde à $2^{10}$ (1024) divisões do intervalo medido.

Para calcular a precisão dos valores aferidos pode-se utilizar a seguinte equação (1).

$$\mathrm{Pr} = \frac{\max - \min}{2^{R}} \tag{1}$$

Onde Pr é a precisão máxima atingida pelo conversor analógico digital, *max* é o valor máximo do intervalo de medição, *min* é o valor mínimo do mesmo intervalo e *R* é a resolução do conversor.

Para o caso do BasicStep M8 temos os seguintes parâmetros: *max*=5, *min*=0 e *R*=10, o que nos resulta:

$$\mathrm{Pr} = \frac{5-0}{2^{10}} = \frac{5}{1024} \cong 0{,}005 \tag{2}$$

## *2.3 - Comunicação serial entre o microcontrolador e o computador*

O microcontrolador definido para o projeto possui uma interface serial para a comunicação entre este e o computador, tanto para a transferência de dados como para a programação, sendo assim era importante que o computador utilizado possuísse um conector seria tipo DB-9 e que este estivesse disponível para ser usado pelos softwares utilizados no projeto. O computador utilizado possuí uma placa-mãe que segue os padrões de formato ATX com um conector DB-9 soldado diretamente nela. A conexão serial da placa-mãe foi definida como COM1 com o seguinte protocolo de comunicação serial: 2400bps, 8 bits de dados, Nenhum bit de paridade, 1 bit de parada (*stop bit*), nenhum controle de fluxo, usar buffer de fila (14 para a recepção e 16 para a transmissão) e drivers do próprio sistema operacional.

Importante mencionar que a opção de 2400bps de velocidade de comunicação foi definida baseada na instabilidade que o programa BasicStep apresentou com velocidades mais altas para a gravação do programa no BasicStep M8.

Toda a comunicação é realizada através deste único conector DB-9, quando o programa era gravado e testado no microcontrolador pelo BasicStep e quando o sistema era efetivamente usado, com o programa CVI. Sendo assim, os dois programas, BasicStep e CVI não podiam estar ativos ao mesmo tempo. Mais especificamente, quando o programa em CVI é executado, ele não aciona a porta serial, isto quer dizer que a porta ainda está disponível para uso por outros softwares, mas quando o usuário pressiona a chave “ligar” dentro do programa, está bloqueia o uso da porta permitindo que apenas o CVI a utilize e no caso do BasicStep, ele bloqueia o uso da porta quando os dados estão sendo gravados ou quando o usuário pressiona o botão “liberar reset” no próprio programa.

### 2.3.1 - RS-232

Este padrão foi inicialmente usado para conectar um teletipo (equipamento eletromecânico de comunicação assíncrona que usava código ASCII) a um modem. A terceira revisão deste padrão (chamada de RS-232C) fora publicada em 1969, em parte para adequar-se a características elétricas destes dispositivos. Deste modo, fora utilizado em diversos tipos de comunicação remota, especialmente por modems [4].

Atualmente o RS-232 está sendo gradualmente suprimido pelo USB para comunicação local. Este é mais rápido, tem conectores mais simples de conectar e usar, e tem um melhor suporte por software. Por isso muitas placas-mãe, destinadas ao uso em

escritórios são produzidas sem circuitos RS-232. Mesmo assim, continua sendo utilizado em periféricos para pontos de venda (caixas registradoras, leitores de códigos de barra ou fita magnética) e para a área industrial (dispositivos de controle remoto), então computadores para estes fins continuam sendo produzidos com portas RS-232, tanto 'on-board' ou uma placa PCI ou ISA separada. Como alternativa, existem adaptadores para portas USB, que podem ser utilizados para conectar teclados ou mouses PS/2, uma ou mais portas seriais e uma ou mais portas paralelas.

São usados conectores machos e fêmeas, geralmente os conectores dos cabos são machos e os conectores de dispositivos são fêmeas. Este padrão é recomendado para conexões curtas (quinze metros ou menos). Os sinais variam de 3 a 15 Volts positivos ou negativos, valores próximos de zero não são sinais válidos.

Os dispositivos RS-232 podem ser classificados em DTE e DCE; isto define que fios irão mandar e enviar quais sinais.

O diagrama dos pinos da RS-232 está presente na figura 2.3.1.1.

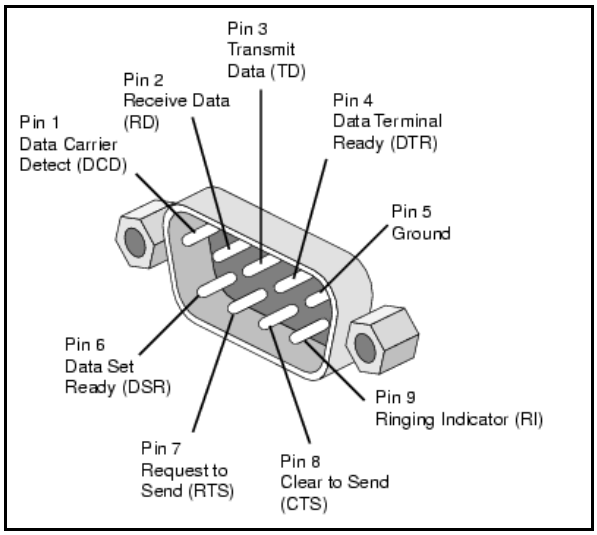



**Figura 2.3.1.1 – Conector DB-9 fêmea**

O BasicStep possui um controlador serial padrão RS-232 que funciona no modo síncrono full-duplex permitindo que dados sejam enviados e recebidos dados simultaneamente. O controlador serial utiliza quatro pinos localizados na parte superior do CI para envio e recebimento de informações. Esses pinos também identificam se o cabo está conectado a um computador controlando o RESET.

Para efetuar uma transmissão ou receber qualquer dado através da RS-232 é preciso configurar o padrão de envio e recebimento bem como a velocidade de comunicação. Para o projeto configuramos o microcontrolador para trabalhar com a velocidade de envio e recepção dos dados de 9600bps, importante mencionar que a

velocidade de envio e recepção devem ser as mesmas. A saída serial do microcontrolador trabalha com a configuração 8 N 1, isto é, 8 bits de dados, nenhum bit de paridade e 1 bit de parada (*stop bit*).

O circuito de comunicação cujos conectores se encontram na parte superior do CI já está nas especificações de tensões exigidas pelo padrão RS-232, sendo assim, não é necessário utilizar nenhum circuito adicional para isso, simplificando bastante o desenvolvimento do projeto.

O segundo circuito de comunicação, pinos D0 e D1, não seguem os padrões RS-232 de tensão e sim os padrões TTL, isto é, ele trabalha com um trem de pulsos de 5V.

## *2.4 - Breve introdução ao software BasicStep (versão para BasicStep M8)*

Software desenvolvido pela TATO (www.tato.ind.br) que acompanha o microcontrolador, neste caso, para o microcontrolador BasicStep M8 também fornecido pela TATO, responsável pela escrita, compilação e gravação do programa para o microcontrolador. Foi utilizada a versão 1.2.2.1, última versão disponível no site até o momento de execução do projeto. O software pode ser baixado através do próprio site da TATO. A linguagem de programação do microcontrolador é a TBASIC que é uma variação da linguagem BASIC desenvolvida especificamente para os microcontroladores da série BasicStep, mas o microcontrolador também pode ser programado usando a linguagem C . O ambiente de desenvolvimento possui uma ajuda e no site se encontram alguns manuais sobre sua utilização em formato PDF. O programa possui uma área de programação, um serviço de terminal para testes e alguns botões de atalho para as suas principais funções.

A figura 2.4.1 apresenta a tela inicial do software onde os pontos importantes estão assinalados em vermelho.

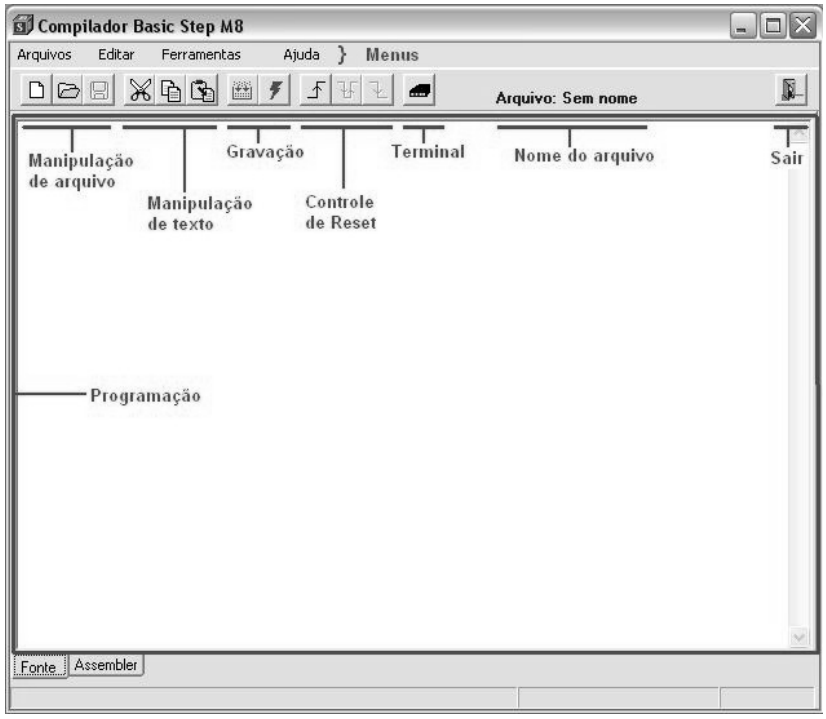



**Figura 2.4.1 – Tela do BasicStep**

A primeira coisa a fazer é definir para qual microcontrolador este programa será utilizado, a versão do BasicStep utilizada pode ser usada para o BasicStep M8, BasicStep M16, BasicStep 1 e BasicStep 1 LT. Para isto basta abrir o menu “Ferramentas” e clicar em “BasicStep M8”, no caso deste projeto. Feito isso definimos a porta serial onde o microcontrolador está conectado ao computador, também no menu “Ferramentas” e selecionando “Porta Serial” e “COM1”. Agora definimos a linguagem de programação a ser usada, também no menu “Ferramentas”, selecionamos “Linguagem BASIC”. Agora o programa está pronto para ser usado.

O menu “Ajuda” possui uma listagem de todos os comandos que podem ser utilizados na programação do microcontrolador explicando como utilizá-los. A ajuda se refere apenas a linguagem TBASIC.

Abaixo dos menus temos os botões de fácil acesso, todas as funções executadas pelos botões também se encontram nos menus.

Manipulação de arquivo:

Esta área possui os botões que permitem abrir uma área em branco para digitar um novo programa, abrir uma janela para procurar e abrir um arquivo já existente no computador e salvar a programação atual em um arquivo, o nome do arquivo é fornecido pelo usuário.

Obs: O programa BasicStep nesta versão possui um “bug”, o arquivo para ser gravado no microcontrolador deve estar no mesmo diretório de instalação do programa.

Manipulação de texto:

Esta área possui os botões que permitem recortar o texto selecionado, copiar o texto selecionado e colar o texto na posição onde se encontra o cursor.

Gravação:

Esta área possui os botões para compilar o programa, se o software detectar algum erro no programa a operação não é finalizada com sucesso e o software informa ao usuário. Caso não sejam detectados erros, o software passa algumas informações do uso da memória pelo programa para o usuário, gravar o programa no microcontrolador, se o programa não tiver sido compilado anteriormente, ele será compilado antes da gravação.

Controle de *Reset:*

Estes botões permitem: liberar o *reset* permitindo que o microcontrolador opere normalmente, “resetar” o microcontrolador momentaneamente, o mesmo que pressionar e soltar o botão de *reset* conectado ao microcontrolador e ativar o *reset* interrompendo a operação do microcontolador até que o *reset* seja liberado novamente, o mesmo que manter pressionado o botão de *reset* conectado ao microcontrolador indefinidamente.

Terminal:

Abre um terminal para envio e recebimento de dados para o microcontrolador, normalmente usado apenas para testes.

Nome do arquivo:

Informa o nome do arquivo aberto ou salvo, “Sem nome” caso o arquivo ainda não tenha sido salvo.

Programação:

Área destinada à escrita do programa que será posteriormente compilado e gravado no microcontrolador.

# 3 – Circuitos condicionadores de sinais

## *3.1 - Circuito regulador de tensão*

A alimentação do circuito do microcontrolador e demais circuitos auxiliares é feita através de um transformador e um regulador de tensão como pode ser observado na figura 3.1.1. A alimentação do microcontrolador é feita pelo pino 2 do BasicStep.

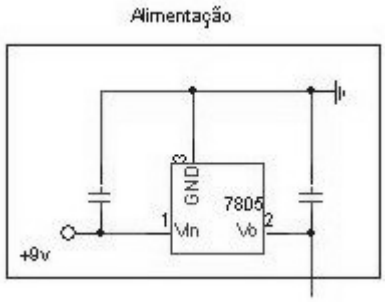



**Figura 3.1.1 – Circuito regulador de tensão**

## *3.2 - Circuito de medição*

Para que o controle atuasse de forma mais eficiente, o sinal medido foi amplificado. O circuito amplificador pode ser visto na figura 3.2.1.

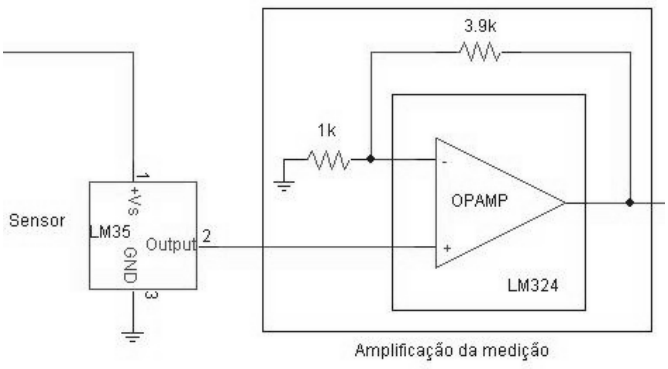



**Figura 3.2.1 – Circuito de aquisição e amplificação do sinal**

O circuito amplificador amplia a saída por um fator de aproximadamente 5 (4,995 medidos experimentalmente). Segue a equação do circuito:

$$\frac{Vo}{Vi} = 1 + \frac{R1}{R2} = 1 + \frac{3{,}9k}{1k} = 4{,}9 \approx 5 \qquad (3)$$

Com isso, a temperatura máxima que pode ser medida é de 100℃, pois isso corresponde a 1V que multiplicado por 5 resulta em 5V que é a tensão máxima que pode ser fornecida ao canal A/D do microcontrolador.

### 3.2.1 - Sensor

O sensor definido para o projeto precisava ser um sensor de boa precisão que trabalhasse a temperaturas maiores que 20℃ e até a pelo menos 100℃, de boa velocidade de resposta e se possível que trabalhasse com tensão nominal de 5V.

O sensor escolhido foi o TMP [6] revendido pela TATO, pois atende a todas as especificações acima. Este sensor possui três pinos, um de alimentação que pode ser de 4V a 30V, atendendo a especificação de tensão exigida, um pino de terra e um pino com o sinal de saída. O TMP é fabricado pela *National Semicondutor* com o nome de LM35. Seu esquema de ligação pode ser visto na figura 3.2.1.1.

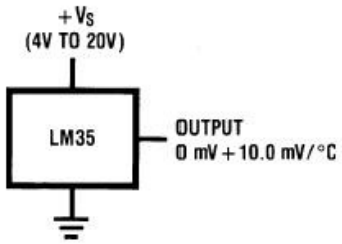



**Figura 3.2.1.1 – Configuração do TMP para temperaturas acima de 2°C sensor (retirada do *datasheet* [6])**

O TMP apresenta uma baixa impedância de saída e uma saída linear e precisa o que facilita bastante a conexão com os circuitos de aquisição de dados.

Este sensor também conta com um baixo consumo de energia, sua corrente de operação é na ordem de 60μA, o que é importante para um medidor de temperatura, pois quanto maior a energia consumida pelo dispositivo, maior seu aquecimento o que implica em uma incerteza maior na medida da temperatura do meio ao qual está imerso. Com essa

corrente baixa, apresenta baixíssimo aquecimento, menos de 0,1℃ no ar. Como a corrente do dispositivo é bem pequena, sua saída pode ser ligada diretamente ao microcontrolador sem risco de danificá-lo.

A sua tensão de saída apresenta um comportamento linear em relação a temperatura medida, outra característica importante e que também contribui para a simplificação do projeto é o fato da tensão de saída ser de 10mV para cada grau Celsius, o que pode ser representado pela equação (4):

$$V = \frac{T}{100} \tag{4}$$

Onde V é a tensão no pino de sinal de saída do sensor em Volts e T é a temperatura medida diretamente em graus Celsius, ou seja, se a temperatura medida for de 25℃, o sensor terá uma saída de 0,25V.

Este sensor trabalha com uma faixa de temperatura de +2℃ até +150℃, mas pode trabalhar em outras faixas fazendo uma pequena modificação no circuito fazendo com que a faixa de temperatura seja de –55℃ até 150℃. O esquema para essa configuração pode ser visto na figura 3.2.1.2.

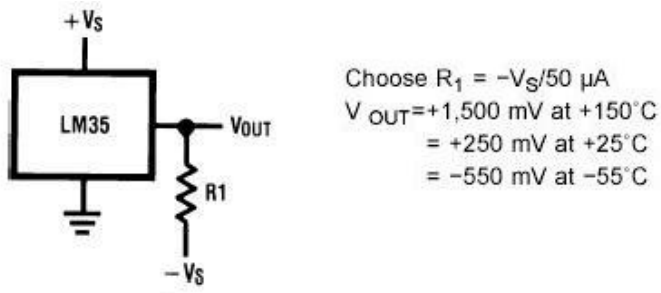



**Figura 3.2.1.2 – Configuração do TMP para temperaturas acima de -55°C sensor (retirada do *datasheet* [6])**

A precisão da medida na temperatura ambiente é de aproximadamente 0,25℃ e para outras faixas de temperatura a precisão é de aproximadamente 0,75℃.

Neste projeto o sensor será utilizado para medir a temperatura da saída de um túnel de vento. O gráfico da figura 3.2.1.3 mostra o tempo de resposta do sensor em função da velocidade do vento, observa-se que quanto maior a velocidade do vento melhor é o tempo de resposta do sensor.

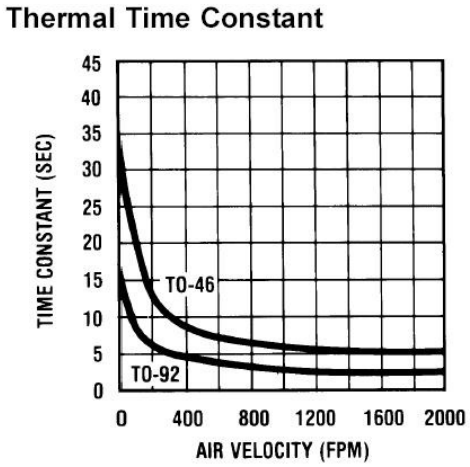



**Figura 3.2.1.3 – Relação entre a velocidade do vento e a constante de tempo do sensor (retirada do *datasheet* [6])**

O encapsulamento do LM35 utilizado é do tipo TO-92, semelhante a um transistor BJC, onde a face plana é a face responsável pela medição da temperatura, pela figura 3.2.1.4 observa-se que o encapsulamento TO-92 apresenta um melhor resultado que o TO-46. Também podendo ser encontrado com outros tipos de encapsulamento. O encapsulamento e a pinagem podem ser vistos na figura 3.2.1.4.


**Figura 3.2.1.4 – Encapsulamento do sensor (retirada do *datasheet* [6])**

O circuito do sensor pode ser visto na figura 3.2.1.5.

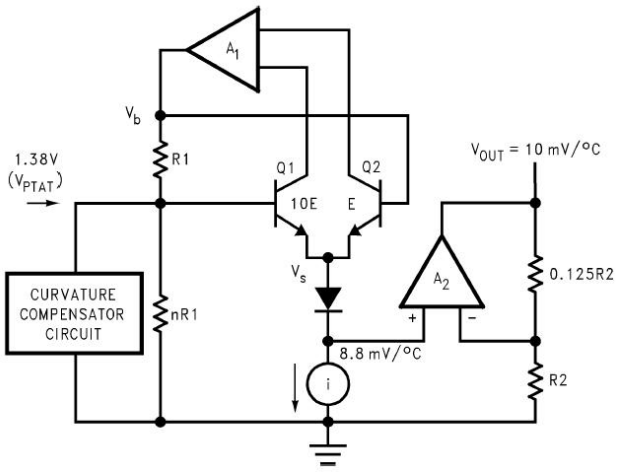



**Figura 3.2.1.5 – Circuito interno do sensor (retirada do *datasheet* [6])**

Maiores informações podem ser encontradas no *datasheet* do LM35 encontrado no site do fabricante (http://www.national.com/pf/LM/LM35.html).

## *3.3 - Circuito de segurança*

Como não havia muitos dados elétricos sobre o microcontrolador (corrente máxima suportada, consumo de energia, etc...) foi desenvolvido um circuito auxiliar de interface com a saída digital para garantir que a corrente entregue ao microcontrolador fosse bem pequena. O circuito está destacado na figura 3.3.1, este circuito é alimentado com uma tensão de 5V e a base do transistor BC327 é conectada a uma das portas de E/S do microcontrolador.

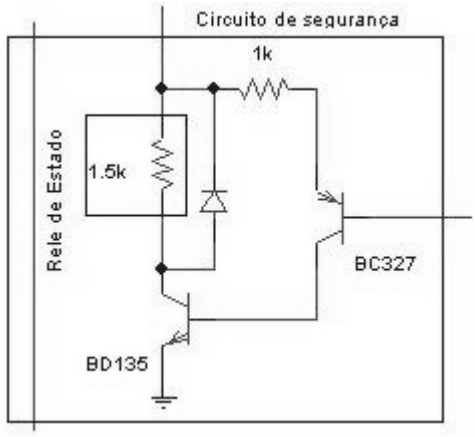



**Figura 3.3.1 – Circuito de segurança**

Observa-se o uso de um transistor PNP (BC 327) cuja função é fornecer a corrente para o microcontrolador. Quando a porta de E/S está em nível lógico 1, sua tensão é de 5V se igualando a tensão do circuito, assim a corrente no circuito é zero e conseqüentemente o relé fica no estado aberto. Quando a porta está em nível lógico 0, ou seja 0V, a corrente flui circuito fazendo com que o relé opere como uma chave fechada, isto é, a resistência no túnel passa a receber corrente. Em resumo, quando a porta se encontra em nível lógico 1, não há potência fornecida a resistência do túnel e quando a porta se encontra em nível lógico 0 é fornecida a potência para resistência. Essa característica se reflete no programa de controle gravado no BasicStep M8, pois quando o programa calcula um ciclo de trabalho alto, isto quer dizer que o circuito deverá fornecer potência por mais tempo e isso significa colocar a saída da porta de E/S em nível lógico 0 por mais tempo.

### 3.3.1 - O Relé de Estado Sólido

O relé é um circuito que funciona como uma chave. Quando uma tensão é aplicada no lado de baixa tensão, o lado de alta tensão passa a conduzir, ou seja, a chave fica fechada.

O relé utilizado havia sido adquirido pelo laboratório, é o mesmo relé utilizado anteriormente, modelo S505-0SJ640-000. Este relé era desenvolvido pela *Continental Industries Inc*.

Este relé não é mais fabricado e não foram encontradas informações sobre ele, assim as informações são todas dados de placa, com exceção de sua resistência interna que foi medida experimentalmente.

O diagrama do relé de estado sólido S505-0SJ640-000 se encontra em anexo (anexo 1).

## *3.4 – Outros circuitos*

O diagrama esquemático do circuito completo está em anexo (anexo 2).

No anexo também pode ser visto um circuito Ligado/Desligado composto por um LED e um resistor. Quando o microcontrolador está operando, este LED fica piscando com uma duração de aproximadamente um segundo. Quando o microcontrolador está em *reset* ou desligado o LED fica apagado. Isso é importante, pois como foi visto na introdução, o fato do circuito estar alimentado não significa que o microcontrolador esteja operando, é

necessário que algum software abra a porta de comunicação serial (COM1) para o microcontrolador entrar em funcionamento ou que o cabo serial esteja desconectado.

O circuito de alta tensão pode ser vista na figura 3.4.1.

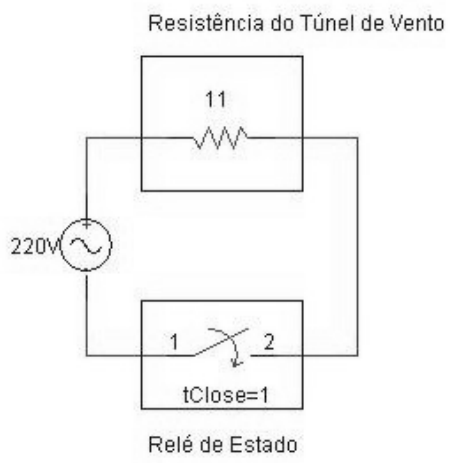



**Figura 3.4.1 – Circuito de alta tensão**

# 4 – O controle

## *4.1 - O Controle PI*

Um controle somente do tipo P (proporcional) tem a desvantagem de sempre apresentar um erro na saída para plantas que não tenham pólos na origem, com isso não é possível o usuário pré-definir uma temperatura de saída e obter esta temperatura.

Esta desvantagem é corrigida pelo PI (proporcional integrador), assim o usuário pode determinar a temperatura desejada do sistema que o controlador irá trabalhar para alcançar este valor e tentar mantê-lo o mais estável possível.

Um outro problema é que o sistema possui muitas variáveis externas, temperatura ambiente não constante, tempo de aquecimento do ar no tubo, etc... O controlador PI também tentará corrigir essas perturbações.

Neste projeto foi implantado um sistema de controle do tipo PI [5]. Esse tipo de controle atua no erro da medida de duas maneiras, possui uma parte que trabalha com um valor proporcional ao erro e outra parte que trabalha com um valor relativo a integral do erro sendo o erro definido como a temperatura desejada menos a temperatura medida na saída do sistema. Essas duas partes são combinadas para gerar um único sinal de controle que é aplicado ao sistema a ser controlado.

Neste projeto o sinal de controle é um sinal tipo PWM aplicado na resistência interna do túnel de vento, dependendo do ciclo de trabalho desse sinal, a potência fornecida para a resistência será proporcional ao valor do ciclo de trabalho definido e conseqüentemente a temperatura do ar passando por está resistência terá valores diferentes. O ar tem contato direto com a resistência no interior do tubo.

A função de transferência do controlador PI está apresentada na equação (5):

$$PI(s) = Kp + {Ki}/{s} \tag{5}$$

Que pode ser arrumada na notação de pólos e zeros ficando da seguinte forma:

$$PI(s) = {[sKp + Ki]}/{s} \tag{6}$$

Sendo que Kp é o ganho relativo à parte proporcional e Ki é o ganho relativo à parte integral.

A função de transferência adiciona um pólo na origem e um zero na posição $-K_i/K_p$ ao sistema. Neste projeto o controle tipo PI é aplicado em um sistema microcontrolado logo ele trabalha em ponto fixo.

A figura 4.1.1 representa um sistema de controle qualquer baseado em um controle tipo PI.

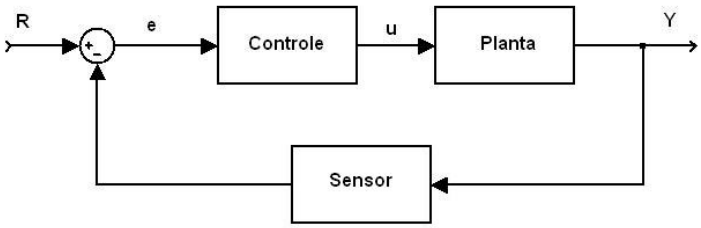



**Figura 4.1.1 – Diagrama em blocos do controle**

A equação de saída de um controlador PI é a seguinte:

$$u = k_p e + k_i \int e dt \tag{7}$$

Eliminando a integral tem-se:

$$\frac{du}{dt} = k_p \frac{de}{dt} + k_i e \tag{8}$$

E passando para ponto fixo:

$$\Delta u_k = k_p \Delta e_k + k_i h e_k \tag{9}$$

Onde *u* é a saída do controlador, *e* é a sua entrada, *h* é o tempo de amostragem, $k_i$ é o ganho da parte integral e $k_p$ é o ganho da parte proporcional do PI.

No projeto *u* representa o ciclo de trabalho do PWM e *e* o erro da medida.

Para uma medida em um instante k temos:

$$\Delta e_k = e_k - e_{k-1} \tag{10}$$

$$e_k = T - T_k \tag{11}$$

Onde $T$ é a temperatura desejada e $T_k$ é a temperatura medida no instante $k$.

E para finalizar:

$$u_k = u_{k-1} + \Delta u_k \tag{12}$$

## *4.2 - PWM*

PWM (Pulse Width Modulation) ou Modulação por Largura de Pulso [1] é uma técnica bastante utilizada atualmente para controle de dispositivos analógicos através de sistemas digitais. Com ele é possível controlar a potência fornecida para o dispositivo analógico com apenas um canal digital, diferentemente de um sistema de controle baseada em conversão D/A (digital/analógica) que precisa de um circuito mais complexo de operação e de maior interferência a ruídos.

Controlando os circuitos digitalmente, o custo do equipamento e o consumo de energia podem ser bastante reduzidos.

Circuitos digitais só produzem dois números: "0" e "1". Já circuitos analógicos podem ter uma infinidade de variações. Por exemplo, em um circuito digital só podemos ligar ("1") ou desligar ("0") um motor ou uma lâmpada, enquanto que em um circuito analógico podemos controlar em infinitos valores o brilho da lâmpada desde o seu estado total de apagamento até o seu brilho máximo. Com um motor acontece o mesmo, podemos controlar em infinitos gradientes sua velocidade, desde o seu estado de repouso até a sua velocidade máxima.

A conversão A/D usa uma quantidade de bits proporcionais à quantidade de gradientes (brilhos, velocidades, etc) que pretendemos ter. Por exemplo, se forem usados 4 bits, temos uma possibilidade de 16 (2^4) gradientes de brilho/rotação, de 0000 a 1111.

O problema desta técnica é que quanto mais gradientes você quiser, mais bits são necessários e mais complexo é o circuito de conversão A/D.

Já a técnica PWM utiliza apenas um bit. Nela é gerada uma forma de onda quadrada onde o ciclo de trabalho (tempo em que a forma de onda permanece em "1") define a velocidade/brilho do sistema analógico.

Por exemplo, supondo uma forma de onda perfeitamente quadrada, onde 50% do tempo ela está em "0" e 50% do tempo ela está em "1", o resultado final será que a lâmpada terá 50% do seu brilho e um motor 50% de sua velocidade.

Se configurarmos esta forma de onda para ficar 10% do seu tempo em "1" e 90% do seu tempo em "0", o resultado será um brilho/velocidade de 10% de sua capacidade total.

A figura 4.2.1 apresenta algumas configurações de ciclos de trabalho PWM.

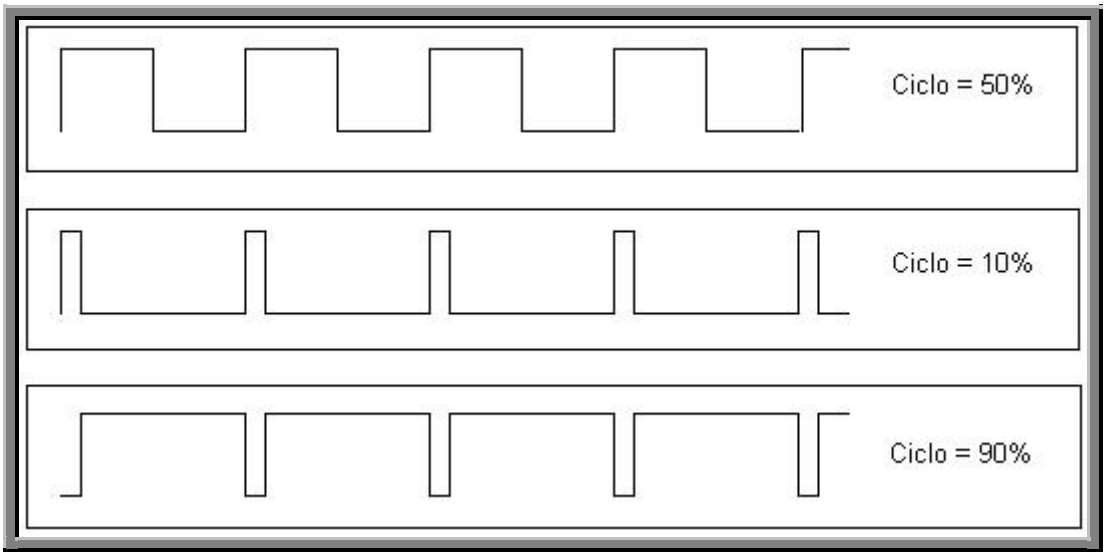



**Figura 4.2.1 – Ciclos de trabalho PWM**

O ciclo de trabalho é definido como:

$$D = \frac{t_{on}}{T} \tag{13}$$

onde:

$$T' = t_{on} + t_{off} \tag{14}$$

Sendo T' o período total da onda, $t_{on}$ o período em que a onda se encontra no estado "1" e $t_{off}$ o período no estado "0" e D o ciclo de trabalho. Esses tempos são limitados pelas características do dispositivo digital utilizado.

## *4.3 - Determinação dos parâmetros do sistema*

### 4.3.1 – MATLAB

Este *software* é desenvolvido pela The MathWorks Inc, permite fazer cálculos matemáticos e a simulação de sistemas de controle.

O MATLAB permite identificar se um sistema é estável ou instável e para quais condições isso ocorre. Para isso determina-se a função de transferência em malha aberta do sistema. O gráfico de Root-Locus é criado usando os pólos é zeros dessa função de transferência. As figuras 4.3.1.1 e 4.3.1.2 representam duas possíveis formas do Root-Locus para o sistema desenvolvido no projeto.

No primeiro gráfico verifica-se que o sistema apresenta um pólo que não tem muita influência no sistema (pólo mais a esquerda) e que o sistema fica instável para ganhos elevados (valores a direita da origem).

No segundo gráfico verifica-se que o sistema é sempre estável para qualquer ganho. Também aparece o pólo de pouca influência no sistema.

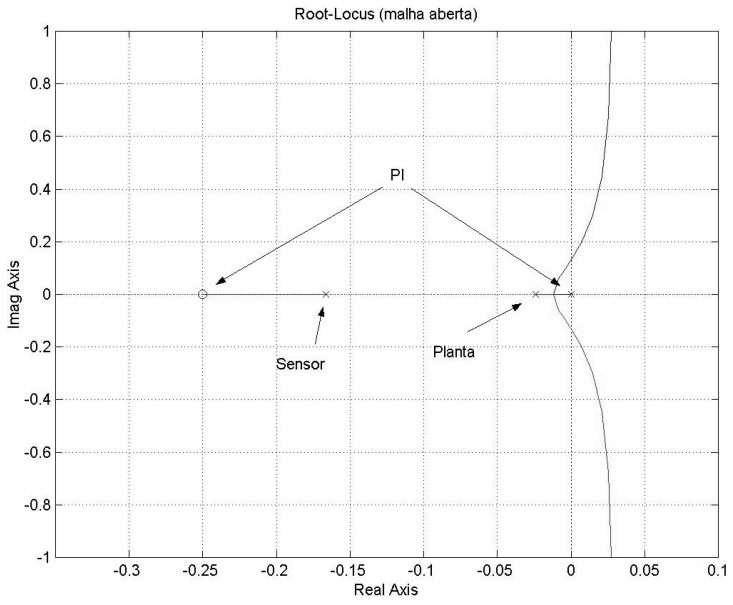



**Figura 4.3.1.1 – Root-Locus**

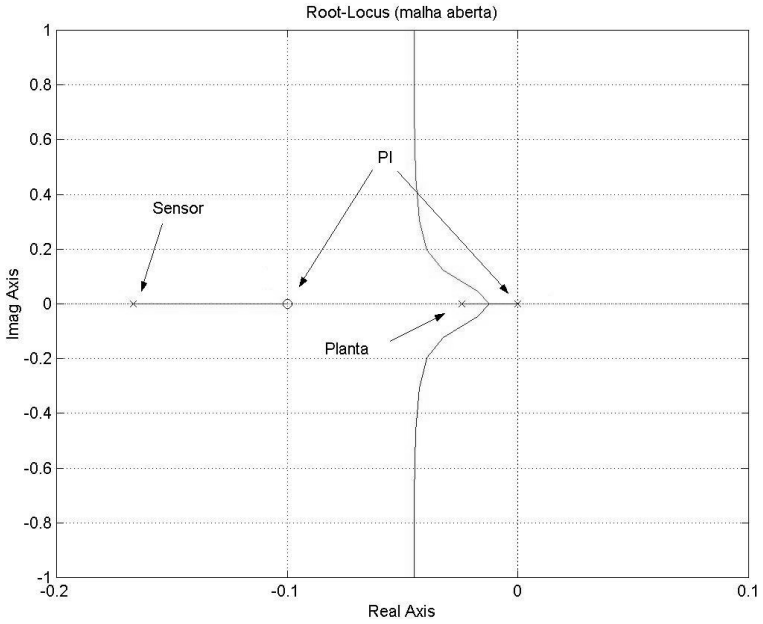



**Figura 4.3.1.2 – Root-Locus**

Verifica-se que o elemento determinante é a posição do zero determinado pelas constantes do PI, relação –Ki/Kp. Em ambos os casos, a resposta do sistema não é muito rápida.

A modelagem acima é feita com base no diagrama do circuito de controle (figura 4.1.1), utilizando as equações de seus elementos.

Abaixo são apresentas as equações dos elementos e a equação final utilizada na modelagem do sistema.

Equação do controle:

$$\frac{kp * s + ki}{s} \tag{15}$$

Equação da planta (circuito do túnel de vento):

$$\frac{k}{\tau_P s + 1} \tag{16}$$

Onde $\tau_P$ é a constante de tempo da planta, 41,33 segundos, e *k* é a relação entre o ciclo de trabalho PWM e a temperatura de saída, 0,16.

Equação do sensor:

$$\frac{1}{\tau_S s + 1} \tag{17}$$

Onde $\tau_s$ é a constante de tempo do sensor e equivale a 6 segundos.

Função de Transferência de Malha Aberta do sistema:

$$\frac{k * kp * s + k * ki}{s(\tau_P s + 1)(\tau_s s + 1)} \tag{18}$$

### 4.3.2 - PSCAD

Uma das vantagens do software PSCAD é que possibilita a analise tanto do comportamento dos dispositivos (componentes analógicos) como do controle desenvolvido em software (componentes digitais).

Este software é desenvolvido pela Manitoba HVDC Research Center Inc [8].

Nesta simulação foi considerada a temperatura desejada na saída do túnel de 50°C que após as conversões devido ao A/D e ao circuito de amplificação da medida corresponde ao valor 511, este valor tem que ser inteiro, pois o microcontrolador só trabalha com números inteiros.

Processo de transformação de °C para palavra inteira (exemplo para 50°C):

50°C => 0,50V (tensão correspondente a uma medição de 50°C) * 5 (circuito de amplificação) * 204,6 (fator de conversão do A/D, 10 bits e tensão máxima de 5V) = 0,50 * 1023 = 511,5 => 511 (valor inteiro correspondente).

Esta simulação tem por objetivo validar o sistema de controle, ou seja, verificar se os comportamentos dos parâmetros estão dentro do esperado. Os valores das constantes do PI são determinados experimentalmente devido ao fato do circuito da simulação não representar todos os aspectos do ambiente simulado como, por exemplo, a temperatura da sala.

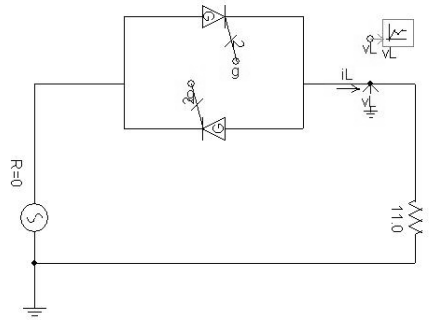



**Figura 4.3.2.1 – Circuito de alta tensão do PSCAD**

A figura 4.3.2.1 representa a parte do circuito de alta tensão (220V), isto é, a fonte de alimentação, o relé de estado sólido e a resistência no túnel. Nesta figura o relé está sendo representado por dois GTO, apesar dele ser composto por um TRIAC, o que não invalida a simulação. Um GTO nada mais é que um tiristor com um canal de controle, quando é aplicada uma corrente reversa neste canal, o GTO é desligado. São necessários dois GTOs para permitir que a potência seja fornecida à resistência tanto durante o ciclo positivo como no ciclo negativo da tensão da rede.

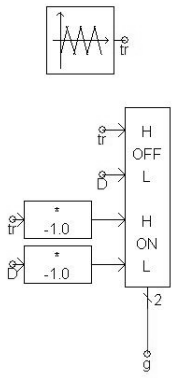



**Figura 4.3.2.2 – Circuito de disparo dos GTO**

A figura 4.3.2.2 representa o circuito responsável pelo controle dos GTOs, isto é, a geração da onda PWM que irá ser aplicada ao canal de controle dos GTOs. Está onda é originada em função de uma onda triangular com uma freqüência de 500Hz e o ciclo de trabalho resultante do circuito de controle. O circuito também garante que quando um GTO estiver ligado o outro estará desligado.

Como o circuito tem um componente de chaveamento, o relé, este chaveamento gera potência reativa no circuito, sendo assim, a potência dissipada na resistência não pode ser apenas a corrente através dela multiplicada pela sua queda de tensão. O bloco

apresentado na figura 4.3.2.3 representa esse cálculo da potência considerando a potência reativa do circuito.

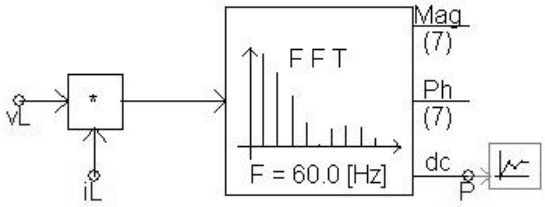



**Figura 4.3.2.3 – Circuito do cálculo da potência da resistência**

O bloco da figura 4.3.2.4 transforma essa potência em temperatura já fazendo a medição desta que é a temperatura de saída do túnel.

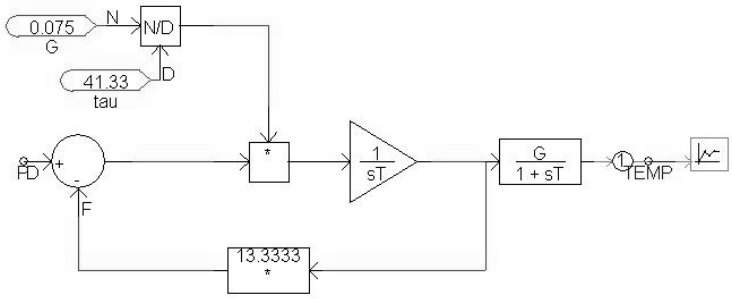



**Figura 4.3.2.4 – Bloco de transformação da potência em temperatura (modelo do sistema junto com o sensor)**

O bloco de controle está apresentado na figura 4.3.2.5. Nela, é feita a comparação das medidas que resulta no erro que é usado no PI. A saída do PI passa por um limitador por causa das limitações do sistema digital. O circuito menor é apenas para preparar o sinal do erro para a exibição no gráfico.

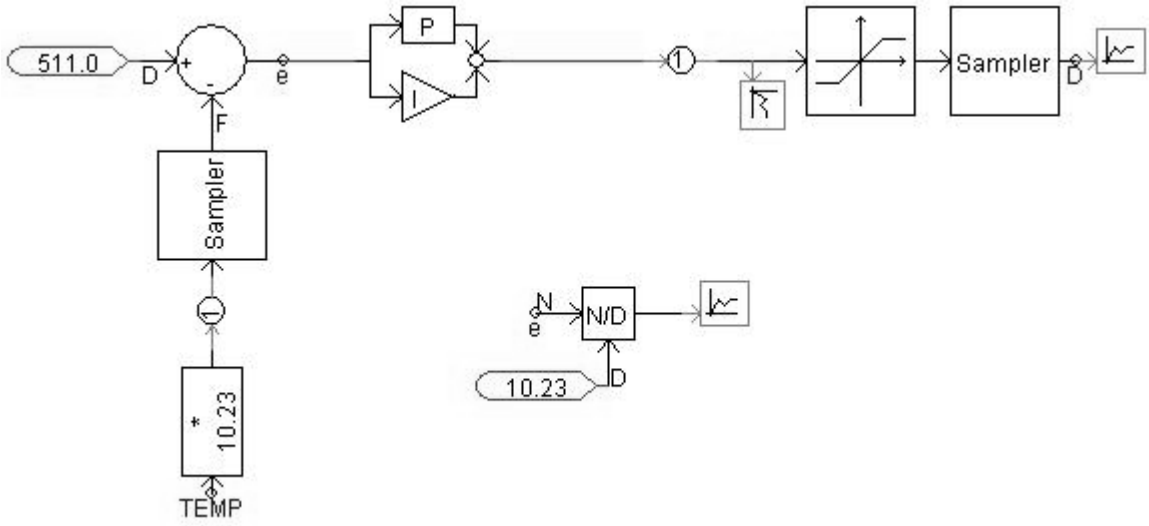



**Figura 4.3.2.5 – Bloco de controle**

A simulação foi efetuada por 200 segundos, os gráficos abaixo demonstram o comportamento do circuito.

A figura 4.3.2.6 apresenta o gráfico da tensão que chega na resistência, ele expressa claramente o efeito que tem um chaveamento em PWM, a distorção harmônica e os estados on/off do PWM.

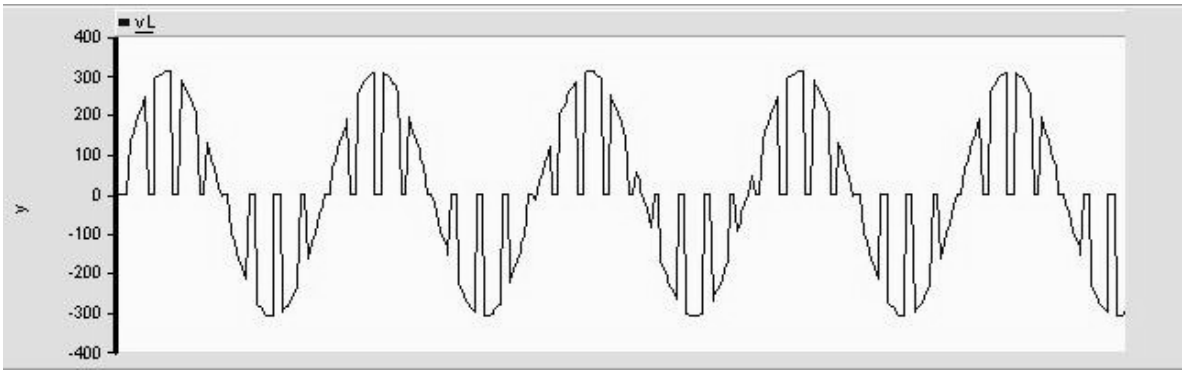



**Figura 4.3.2.6 – Forma de onda da resistência aplicando o sinal PWM**

A potência fornecida para a resistência pode ser vista na figura 4.3.2.7. Observa-se que a potência é máxima no início, isto ocorre porque a temperatura medida está bem abaixo da temperatura desejada e da alta constante de tempo do sistema.

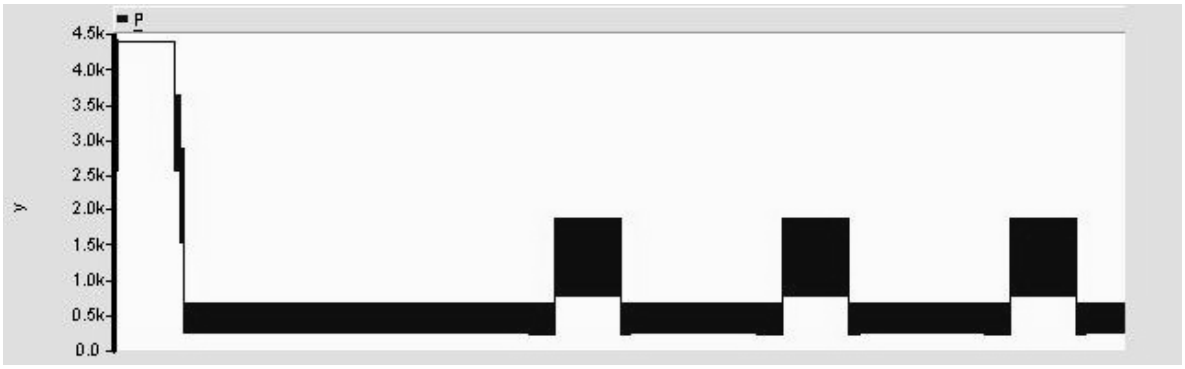



**Figura 4.3.2.7 – Potência dissipada na resistência**

A temperatura medida é vista na figura 4.3.2.8. Em função da alta constante de tempo do sistema, a temperatura sobe bastante no começo, mas depois o sistema consegue controlar a saída. O gráfico se encontra em regime permanente a partir de mais ou menos 80 segundos.

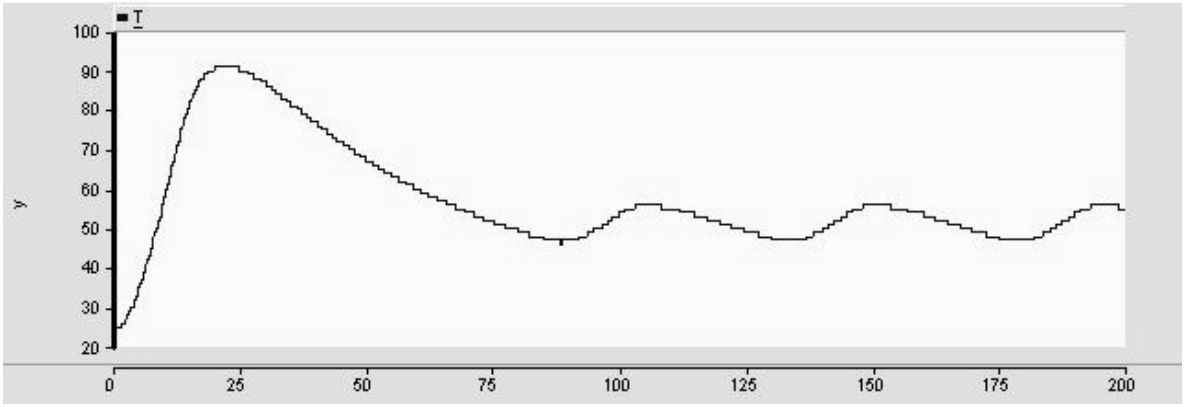



**Figura 4.3.2.8 – Temperatura na saída do sistema**

A saída do controlador PI já limitada para o valor 1023 pode ser vista na figura 4.3.2.9. Nota-se que o ciclo de trabalho cai rapidamente chegando a zero quando a temperatura está acima da temperatura desejada.

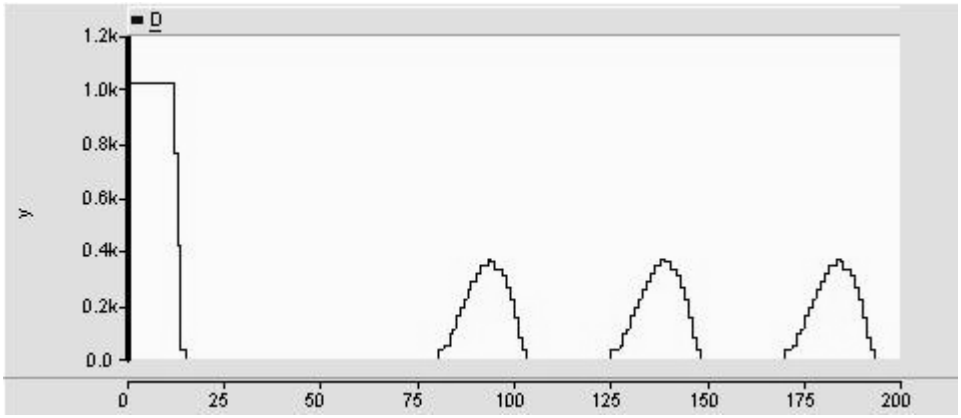



**Figura 4.3.2.9 – Saída do controle (valor do ciclo de trabalho PWM)**

O erro da medida, diferença entre a temperatura desejada e a medida, pode ser visto no gráfico da figura 4.3.2.10.

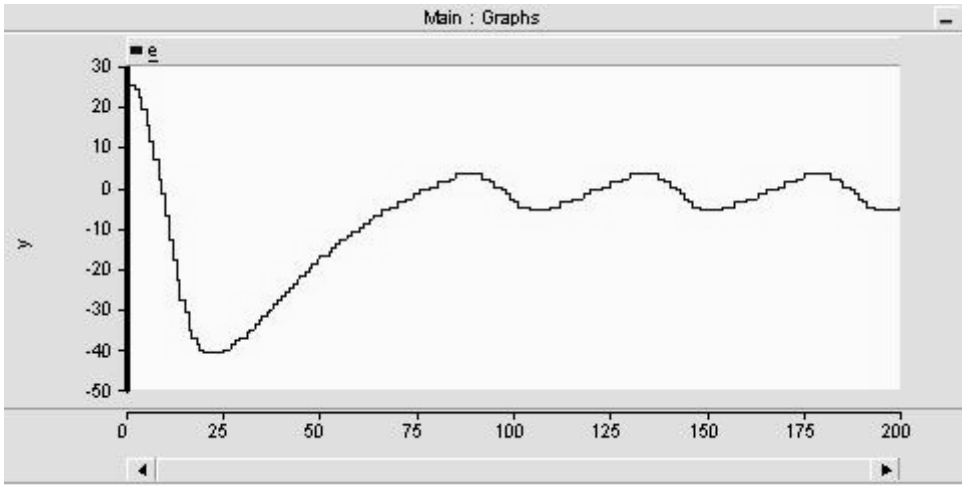



**Figura 4.3.2.10 – Erro da medida observada na saída do sistema**

Com exceção do primeiro gráfico (gráfico da tensão na resistência), os demais estão apresentados totalmente, isto é, usando todo o intervalo de tempo da simulação. No gráfico da tensão, a faixa de tempo é de apenas alguns segundos, isso é necessário para poder visualizar a onda que é de 60Hz.

### 4.3.2.1 - Determinação de K e PT

A constante K é a relação entre o ciclo de trabalho e a temperatura de saída, ela é medida aplicando um ciclo de trabalho constante no circuito e mede-se a temperatura em regime permanente de saída. K é a temperatura em graus Celsius dividida pelo ciclo de

trabalho. Os valores de K para diferentes ciclos de trabalho estão apresentados na tabela 4.3.2.1.

| Ciclo de trabalho | Temperatura (°C) | K |
|---|---|---|
| 123 | 44 | 0,358 |
| 223 | 46 | 0,206 |
| 323 | 54 | 0,167 |
| 423 | 66 | 0,156 |
| 523 | 69 | 0,132 |

**Tabela 4.3.2.1.1 – Determinação da constante K**

Verifica-se que a relação entre o ciclo de trabalho e a temperatura não é linear, logo, na simulação e no desenvolvimento do projeto foi usado o valor médio de K, que é 0.20.

PT é a relação entre a potência fornecida à resistência e a temperatura obtida na saída. Este parâmetro é determinado na simulação. Aplica-se um ciclo de trabalho constante e ajusta-se essa constante até obter a temperatura de saída igual à temperatura de saída verificada experimentalmente para o mesmo ciclo de trabalho.

Na simulação foi usado o valor 323 para o ciclo de trabalho, logo, PT foi ajustado para o sistema ter uma saída de 54°C. Para esta temperatura, PT é igual a 0.04133.

### 4.3.2.2 - Determinação da constante de tempo da resistência

Para a determinação da constante de tempo, aplica-se um ciclo de trabalho constante e espera o sistema entrar em regime permanente. A constante de tempo é determinada graficamente e corresponde ao tempo que o sistema leva para variar de 62,3% de seu valor final em condições iniciais nulas, conforme a equação (19).

$$T_{\tau} = (T_f - T_i) * 0{,}632 \qquad (19)$$

Onde $T_\tau$ é a temperatura no tempo igual a constante de tempo do sistema, $T_f$ é a temperatura final (em regime permanente) e $T_i$ é a temperatura inicial do sistema. A constante de tempo do sistema é de 41,33s.

O gráfico usado para a determinação da constante de tempo pode ser visto na figura 4.3.2.2.1. A temperatura final é de 85°C. e a temperatura inicial é de 25°C.

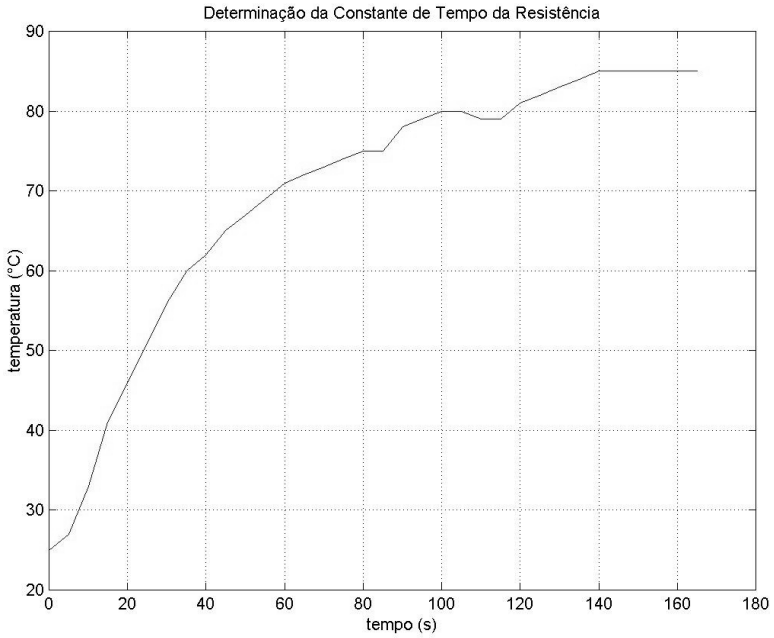



**Figura 4.3.2.2.1 – Forma de onda gerada para um ciclo de trabalho constante**

Esse cálculo apresenta um erro porque ele não leva em consideração a constante de tempo do sensor, mas como essa é bem menor isso não apresenta problemas.

Como este sistema é um sistema térmico então ele é um sistema de primeira ordem.

### 4.3.2.3 - Determinação da constante de tempo do sensor

O *datasheet* do sensor apresenta a curva de constante de tempo em função da velocidade do vento ao qual o sensor está imerso.

A tabela 4.3.2.3.1 apresenta a relação entre a velocidade do vento e a freqüência definida no inversor.

| f (freqüência do inversor) | Velocidade (m/s) | Velocidade em FPM (pés por minuto) | τ (s) |
|---|---|---|---|
| 30 | 5,8 | 105 | 8 |
| 40 | 10,1 | 184 | 6 |
| 50 | 14,2 | 259 | 5,5 |
| 60 | 17,6 | 320 | 5 |

**Tabela 4.3.2.3.1 – Relação entre a freqüência do inversor e a velocidade do vento**

Com base na tabela 4.3.2.3.1 determinou-se a constante de tempo do sensor de aproximadamente 6s, considerando que o desenvolvimento do projeto foi realizado com uma freqüência no inversor de 40Hz. O sistema de controle pode ser usado para velocidades maiores o que resulta em uma melhor resposta do controle, melhorando a qualidade do sinal de saída.

# 5 - Resultados experimentais

O projeto foi desenvolvido para ser aplicado em uma sala com controle ambiental precário, isto é, a temperatura da sala não é constante. Nos experimentos verificou-se que quanto maior a temperatura ambiente maior a oscilação da temperatura de saída do túnel. Isso se deve ao fato de não haver um controle de resfriamento no sistema, o sistema de controle desenvolvido atua apenas no aquecimento, assim, quando a sala estava mais fria evitava que a temperatura se elevasse muito acima da desejada. Outra característica do projeto é que a medição da temperatura não é feita no interior do túnel, mas em sua saída, logo existe um tempo entre o aquecimento do túnel e esse ar aquecido chegar ao sensor e isto também aumenta a interferência do meio na temperatura do ar aquecido.

Os cabos que ligam o sensor ao circuito são isolados para evitar que o ar aquecido os esquente o que resultaria em erro de medição do sensor.

## *5.1 - Programação do BasicStep M8*

O programa gravado no BasicStep se encontra em anexo (anexo 3).

Inicio do programa:

```
XMIT INIT 9600
RECV INIT 9600
RECV INTERRUPT ON
MAKEOUT B,2
MAKEOUT B,7
```

Esta seção define a velocidade de comunicação serial (a velocidade de envio e recepção deve ser a mesma), ativa o suporte a interrupção e define as portas de E/S para serem usadas, neste caso B,7 é o pino da porta B destinado a indicar o estado de operação do controle (ligado ou desligado) e o pino B,2 é destinado a saída PWM. O BasicStep M8 possui 3 saídas PWM mas no manual não informa quais são os pinos usados por ela. O pino B,2 foi identificado observando o sinal de saída em um osciloscópio.

Definição das variáveis:

```
u0% = 0
u1% = 2
e1% = 0
e0% = 0
se0 = 0
TEMP% = 511
ki% = 2
kp% = 8
y% = 1023
```

Esta seção define os estados iniciais de todas as variáveis utilizadas no programa.
e0% é o erro do estado anterior do PI, necessário definir ele aqui já que ele não é gerado na primeira passagem do programa.
se0 é o sinal do erro e0%.
TEMP% é a temperatura desejada inicial, serve apenas para o microcontrolador ter uma condição inicial de trabalho e o programa funcionar corretamente, o valor 511 representa uma temperatura de 50°C.
ki% e kp% são os ganhos do controle PI (obtidos experimentalmente).
y% é o ciclo de trabalho inicial, 1023 significa que inicialmente a resistência não recebe potência.

PWM:

```
PWMB INIT 8, 10
PWMB THRESHOLD y%
```

Esta seção define qual dos canais PWM será usado e suas características (200 significa uma freqüência de 500Hz e 10 significa 10bits) e inicia o PWM com o valor de y% definido anteriormente.

Conversor A/D:

```
A2D x%, 5, IDLE
```

Esta linha define o modo de operação do conversor A/D, o pino a ser utilizado e já faz a leitura do canal. Neste caso IDLE significa que o microcontrolador é colocado em estado de espera para fazer a leitura sem ruídos, x% é a variável que irá receber o valor convertido e 5 significa que será usado o pino C,5 para fazer a conversão.

O controle PI:

```
IF x% = TEMP% THEN
        IF se0 = 0 THEN
                de% = e0%
                dutemp1% = kp% * de%
                IF u0% > dutemp1% THEN
                        u1% = u0% - dutemp1%
                ELSE
                        SHIFT u0%, 1, RIGHT
                        u1% = u0%
                END IF
        END IF
        IF se0 = 1 THEN
                de% = e0%
                dutemp1% = kp% * de%
                u1% = u0% + dutemp1%
        END IF
    END IF
    IF x% > TEMP% THEN
            se1 = 1
            e1% = x% - TEMP%
            IF se0 = 1 THEN
                    IF e1% = e0% THEN
                            dutemp2% = ki% * e1%
                            IF u0% > dutemp2% THEN
                                    u1% = u0% - dutemp2%
                            ELSE
                                    SHIFT u0%, 1, RIGHT
                                    u1% = u0%
                            END IF
```

```
                END IF
                IF e1% > e0% THEN
                        de% = e1% - e0%
                        dutemp1% = kp% * de%
                        dutemp2% = ki% * e1%
                        IF u0% > du% THEN
                                u1% = u0% - du%
                        ELSE
                                SHIFT u0%, 1, RIGHT
                                u1% = u0%
                        END IF
                END IF
                IF e0% > e1% THEN
                        de% = e0% - e1%
                        dutemp1% = kp% * de%
                        dutemp2% = ki% * e1%
                        IF dutemp1% > dutemp2% THEN
                                du% = dutemp1% - dutemp2%
                                u1% = u0% + du%
                        END IF
                        IF dutemp2% > dutemp1% THEN
                                du% = dutemp2% - dutemp1%
                                IF u0% > du% THEN
                                        u1% = u0% - du%
                                ELSE
                                        SHIFT u0%, 1, RIGHT
                                        u1% = u0%
                                END IF
                        END IF
                        IF dutemp1% = dutemp2% THEN
                                u1% = u0%
                        END IF
                END IF
        END IF
        IF se0 = 0 THEN
                de% = e1% - e0%
```

```
                dutemp1% = kp% * de%
                dutemp2% = ki% * e1%
                du% = dutemp1% + dutemp2%
                IF u0% > du% THEN
                        u1% = u0% - du%
                ELSE
                        SHIFT u0%, 1, RIGHT
                        u1% = u0%
                END IF
        END IF
END IF
IF TEMP% > x% THEN
        se1 = 0
        e1% = TEMP% - x%
        IF se0 = 0 THEN
                IF e1% = e0% THEN
                        dutemp2% = ki% * e1%
                        u1% = u0% + dutemp2%
                END IF
                IF e1% > e0% THEN
                        de% = e1% - e0%
                        dutemp1% = kp% * de%
                        dutemp2% = ki% * e1%
                        du% = dutemp1% + dutemp2%
                        u1% = u0% + du%
                END IF
                IF e0% > e1% THEN
                        de% = e0% - e1%
                        dutemp1% = kp% * de%
                        dutemp2% = ki% * e1%
                        IF dutemp2% > dutemp1% THEN
                                du% = dutemp2% - dutemp1%
                                u1% = u0% + du%
                        END IF
                        IF dutemp1% > dutemp2% THEN
                                du% = dutemp1% - dutemp2%
```

```
                                IF u0% > du% THEN
                                        u1% = u0% - du%
                                ELSE
                                        SHIFT u0%, 1, RIGHT
                                        u1% = u0%
                                END IF
                        END IF
                        IF dutemp1% = dutemp2% THEN
                                u1% = u0%
                        END IF
                END IF
        END IF
        IF se0 = 1 THEN
                de% = e1% + e0%
                dutemp1% = kp% * de%
                dutemp2% = ki% * e1%
                du% = dutemp1% + dutemp2%
                u1% = u0% + du%
        END IF
END IF
```

Esta seção é a responsável pela geração do novo ciclo de trabalho da onda PWM (u1%), ela é muito extensa por causa das limitações do microcontrolador, pois este só trabalha com valores inteiros e sem sinal.

Limitação do PWM:

```
IF u1% < 2 THEN
        u1% = 2
END IF
IF u1% > 800 THEN
        u1% = 800
END IF
```

Esta seção tem a finalidade de limitar os possíveis valores do ciclo PWM entre 2 e 800. Valores acima de 800 podem causar superaquecimento do túnel danificando-o se a temperatura subir demais (algo acima de 80℃).

Implementação do PWM:

```
y% = 1024 - u1%
PWMB THRESHOLD y%
```

Esta seção inverte o sinal PWM e coloca efetivamente a onda no pino C,5. Essa inversão é necessária porque quando o pino C,5 se encontra em nível lógico alto isto significa que a resistência não recebe potência.

Preparação para um novo ciclo do programa:

```
e0% = e1%
se0 = se1
u0% = u1%
```

Nesta seção são armazenadas as variáveis modificadas durante o programa para serem usadas novamente.

Comunicação para o PC:

```
SHIFT x%, 2, RIGHT
temperatura = x%
XMIT OUT temperatura
```

Nesta seção o programa prepara a medida proveniente do sensor e a envia para o computador. A linha “SHIFT x%, 2, RIGHT” tem a finalidade de reduzir o tamanho da informação da medida de 10bits para 8bits deslocando o valor 2bits para a direita, o que significa dividir o valor da medida por 4, com isso a medida pode ser enviada através de um único byte. A linha “temperatura=x%” grava o valor de x% que é uma variável do tipo inteira em uma variável do tipo byte, os 8bits menos significativos são gravados. E a última linha envia o dado pela saída serial para o computador.

Estado do circuito:

```
TOGGLE B,7
```

Esta linha é para acionar o LED para indicar se o circuito está em operação (LED piscando) ou não (LED apagado).

Amostragem:

```
PAUSE 1000
```

Está linha indica que o tempo de amostragem e operação do programa é de aproximadamente 1 segundo. O tempo é um pouco maior devido ao tempo de execução do programa, tempo de conversão do A/D e tempo de envio da medida pela saída serial. Esse tempo foi definido baseado na constante de tempo do sistema que é na ordem de 40 segundos. Como o tempo é de um segundo, o parâmetro *h* do PI é 1 e não precisa ser escrito no programa.

A interrupção:

```
IF rflag = 1 THEN
        TEMP% = rbyte * 10.23
        rflag = 0
END IF
```

```
INTERRUPT RECV
        PUSHFLAGS
        PUSHREG
        rflag = 1
        RECV IN rbyte, errflag
        POPREG
        POPFLAGS
END INTERRUPT
```

Estas duas seções são responsáveis pelo tratamento da interrupção, onde o programa salva seu estado (PUSHFLAGS e PUSHREG), indica ao programa que foi gerada uma interrupção (rflag=1) permitindo que o programa entre na rotina de interrupção, também armazena o dado do recurso que causou a interrupção e restaura o estado do programa (POPREG e POPFLAGS) terminando a interrupção. Neste programa, a interrupção é gerada quando o CVI envia a temperatura desejada pela porta serial. Neste caso, o valor é

armazenado na variável rbyte e após passar pela rotina de interrupção, este valor é armazenado na variável TEMP% para ser utilizado no programa.

A rotina de interrupção é responsável por zerar o pedido de interrupção (rflag=0).

## *5.2 - Programação no PC (Ambiente LabWindows/CVI)*

Este software é desenvolvido pela National Instruments e é destinado principalmente para aplicações gráficas baseadas em linguagem C [9] para ambientes de medição e instrumentação [2].

O programa desenvolvido em CVI tem duas finalidades, permitir que o usuário informe ao microcontrolador a temperatura de saída desejada e obter a leitura da temperatura de saída.

O programa consiste em quatro arquivos, são eles:

- M8TMP.c – É onde se encontra o programa desenvolvido, parte escrita, pode ser editado através do CVI ou por um editor de texto puro qualquer.
- M8TMP.h – Esse arquivo é criado pelo próprio CVI, é o *header* do programa.
- M8TMP.prj – É o arquivo de projeto, também criado automaticamente pelo CVI, neste se encontram informações sobre os demais arquivos e parâmetros do CVI.
- M8TMP.uir – Este arquivo contem a interface gráfica do programa, deve ser editado somente através do CVI.

Os arquivos se encontram em anexo (anexos 4.1 a 4.3), nota-se que o arquivo M8TMP.uir não é um arquivo editável.

A construção do arquivo M8TMP.c utiliza a linguagem C.

```
#include <utility.h>
#include <ansi_c.h>
#include <rs232.h>
#include <formatio.h>
#include <string.h>
#include <userint.h>
#include "M8TMP.h"
```

Nesta seção são incluídas todas as bibliotecas e *headers* utilizados pelo programa, o CVI incluí automaticamente as informações necessárias conforme o programa é desenvolvido.

```
short int n2,
        ler_bytes;
int panel_handle,
        erro,
        flag = 0,
        porta = 0,
        step = 0,
        i,
        resetbox = 0,
        datafile,
        logtime,
        log_count = 0,
        log_flag = 0,
        log_A = 0,
        inqlen;
static int panelHandle;
char escrita[80],
        porta_serial[30],
        passo[10],
        entrada[30],
        transfer[30],
        resultstr[500],
        logfilename[100];
double fractpart,
        n,
        intpart,
        fractpartB,
        intpartB;
```

Esta seção define as variáveis do programa.

```
int main (int argc, char *argv[])
{
        if (InitCVIRTE (0, argv, 0) == 0)       /* Needed if linking in external compiler; harmless otherwise */
                return -1;      /* out of memory */
        if ((panel_handle = LoadPanel (0, "M8TMP.uir", PANEL)) < 0)
                return -1;
        DisplayPanel (panel_handle);
        RunUserInterface ();
        return 0;
}
```

Esta seção é criada automaticamente pelo CVI, nela está indicado o arquivo que contém a interface gráfica do programa.

```
int CVICALLBACK StartCallBack (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        int ligado;
        char setupstr[200], dirname[MAX_PATHNAME_LEN];
        switch (event) {
                case EVENT_COMMIT:
                        GetCtrlVal (panel_handle, PANEL_LIGADO, &ligado);
                        if (ligado) {
                                GetCtrlVal (panel_handle, PANEL_LOG, &logtime);
                                GetProjectDir (dirname);
                                sprintf (logfilename,"%s\\results.log", dirname);
                                                datafile = OpenFile (logfilename,VAL_WRITE_ONLY,VAL_TRUNCATE, VAL_ASCII);
                                        sprintf (setupstr,"Arquivo de log do programa de Monitoracao de Temperatura \n----------\n Criado em: Data: %s Hora: %s \n\n", DateStr(), TimeStr());
                                WriteFile (datafile, setupstr, StringLength (setupstr));
                                        SetCtrlAttribute (panel_handle, PANEL_OK, ATTR_DIMMED, 0);
                                strcpy(&porta_serial[0], "COM1");
```

```
                                erro = OpenComConfig (1, porta_serial, 9600, 0, 8, 1, 512,
512);
                                porta = 1;
                                }
                        else   {
                                        SetCtrlAttribute     (panel_handle,     PANEL_OK,
ATTR_DIMMED, 1);
                                erro = CloseCom (1);
                                step = 0;
                                flag=0;
                                }
                        break;
        }
        return 0;
}
```

O programa executa esta seção quando o usuário clica na chave “Ligado” no programa, também é definida a variável ligado, essa variável é local essa seção então afeta demais áreas do programa. Aqui a comunicação serial é inicializada e é criado o arquivo de log. Após o evento da chave “Ligar” o programa disponibiliza o botão para a aquisição dos dados ao usuário, isso é importante para evitar que o usuário tente ler um dado da porta serial antes desta ter sido iniciada.

```
int CVICALLBACK QuitCallBack (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event) {
                case EVENT_COMMIT:
                        if (porta == 0)
                          erro = CloseCom (1);
                        QuitUserInterface (0);
                        break;
        }
        return 0;
}
```

Esta seção é de encerramento do programa, o procedimento é executado quando o usuário clica no botão “Saída” do programa. Aqui a comunicação com a porta serial é encerrada liberando-a para uso pelo sistema operacional. O programa também é fechado.

```
int CVICALLBACK EscreveCallBack (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event) {
                case EVENT_COMMIT:  {
                        flag=1;
                        }
                        break;
        }
        return 0;
}
```

Esta seção apenas informa ao programa para iniciar os procedimentos de leitura da medida e gravação do arquivo de log.

```
void UpdateTextBoxA (char* event_gen)
{
   char display_str[30];

   Fmt(display_str, "%s<%s\n", event_gen);
        SetCtrlVal (panel_handle, PANEL_TEXTBOXA, display_str);

}
```

Está seção apenas formata o dado nos padrões do CVI e escreve o dado na caixa de exibição. É nessa caixa de exibição que o usuário observa o valor medido, este valor já se encontra em graus Celsius.

```
int CVICALLBACK TimerCallBack (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event) {
```

```
            case EVENT_TIMER_TICK:
                    if (flag == 1) {
                    ler_bytes = ComRdByte (1);
                    sprintf(transfer,"%u",ler_bytes);
                    n = atof(transfer);
                    n = n/2.5;
                    fractpart = modf (n , &intpart);
                    fractpartB = fractpart * 10;
                    fractpartB = modf (fractpartB , &intpartB);
                    if (fractpartB < 0.5 ) {
      n = intpart + (intpartB / 10);
     }
    else {
      n = intpart + ((intpartB + 1) / 10);
     }
                    sprintf (transfer, "%0.1f", n);
                    sprintf (passo, "%0.1u", step);
                    step = step + 1;
                    UpdateTextBoxA(transfer);
                    log_count = log_count + 1;
                            if (log_count >= logtime) {
                                    sprintf (resultstr, "%s          %s\n", passo, transfer);
                                    WriteFile (datafile, resultstr, StringLength(resultstr));
                                    log_count = 0;
                                    }
                    resetbox = resetbox + 1;
                    if (resetbox == 20) {
                            ResetTextBox (panel_handle, PANEL_TEXTBOXA, "");
                            resetbox = 0;
                            }
                    inqlen = GetInQLen (1);
                    SetCtrlVal (panel_handle, PANEL_Buffer, inqlen);
                    if (inqlen > 10)
                            FlushInQ (1);
                            }
    }
```

```
	return 0;
}
```

Esta seção faz o tratamento do dado recebido pela porta serial, o dado recebido é um inteiro sem sinal, aqui este valor é convertido e transformado em um valor de ponto flutuante para ser exibido para o usuário. A conversão é necessária porque, como já citado anteriormente, quando o microcontrolador envia o valor 511, por exemplo, este valor corresponde à temperatura de 50ºC.

Nesta seção também é feita a gravação da informação no arquivo de log baseado no tempo determinado pelo usuário.

Para garantir que o dado apresentado na tela fosse atual, o programa verifica o tamanho do buffer de entrada da porta serial, se for maior que 10, ele zera esse buffer, assim o dão apresentado está atrasado em no máximo 10 segundos. Na interface gráfica existe um campo que exibe o tamanho desse buffer. Durante o experimento verificou-se que o tamanho, após algum tempo de execução, caía para 0, ou seja, o dado exibido era o dado mais atual medido.

As caixas de exibição do CVI têm um limite de quantidade de informações que elas podem receber, quando o limite é atingido, os novos valores não são exibidos. Para contornar esse problema, a caixa de exibição é apagada a cada 20 valores exibidos.

```
int CVICALLBACK ViewDataFile (int panel, int control, int event,
		void *callbackData, int eventData1, int eventData2)
{
	char tempstr[100];

	switch (event) {
		case EVENT_COMMIT:
			CloseFile (datafile);
			sprintf (tempstr,"Notepad.exe %s", logfilename);
			LaunchExecutable (tempstr);
			datafile = OpenFile
(logfilename,VAL_WRITE_ONLY,VAL_APPEND,VAL_ASCII);
	}
	return 0;
}
```

Esta seção exibe o arquivo de log abrindo-o no “Bloco de Notas”. A exibição pode ser feita com o programa em execução, não é necessário parar a medição.

```
int CVICALLBACK LOGTIME (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event) {
        case EVENT_COMMIT:
            GetCtrlVal (panel_handle, PANEL_LOG, &logtime);
            break;
    }
    return 0;
}
```

Esta seção informa ao programa o tempo em que as medidas devem ser gravadas no arquivo de log. O usuário seleciona esse tempo através de uma barra de rolagem na interface gráfica.

```
int CVICALLBACK SerialCallBack (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event) {
        case EVENT_COMMIT:
        GetCtrlVal (panel_handle, PANEL_Janela_2, &n2);
        ComWrtByte (1,n2);
        break;
    }
    return 0;
}
```

Nesta última seção é feito o procedimento de envio da medida informada pelo usuário como medida desejada de temperatura na saída do túnel. O valor enviado é em graus Celsius.

A figura 5.2.1 apresenta a interface gráfica, é nesta interface que o usuário interage com o programa. O usuário pode alterar a temperatura de 5℃ para a faixa de 25℃ até 95ºC.

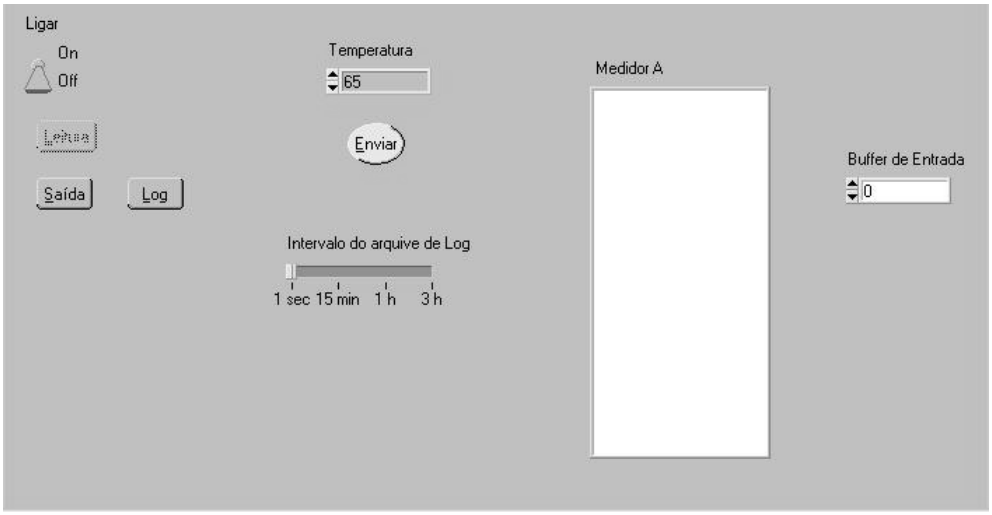



**Figura 5.2.1 – Tela do programa CVI para o usuário final**

Nota-se que o botão de leitura está desabilitado, para habilitá-lo, o usuário deve pressionar a chave “Ligar”.

## *5.3 - O sistema em funcionamento*

Após a conclusão da montagem, ajustes das constantes do PI e programação, foram realizadas algumas medidas para verificar o funcionamento do sistema.

Foi verificado experimentalmente que as constantes do PI tinham que ser bem pequenas, senão o sinal de erro apresentava uma grande amplitude. O problema era que com valores maiores e em função da lenta resposta do sistema, o ciclo PWM atingia rapidamente seus valores extremos, 1023 e 2.

A figura 5.3.1 apresenta a temperatura medida para uma temperatura desejada de 50°C. Para esta medição foi utilizada uma freqüência de 60Hz no inversor, o que fornece uma velocidade do vento de aproximadamente 17,6m/s ou 320 FPM (pés por minuto). Esta figura apresenta a medida, em vermelho, e o erro, em azul. Pode-se verificar que o sistema atinge a temperatura desejada em aproximadamente 55 segundos. A constante de tempo é 26 segundos.

A figura 5.3.2 apresenta uma situação em que a temperatura inicial definida é de 50°C e após 600 segundos, o usuário define a temperatura desejada de 60°C.

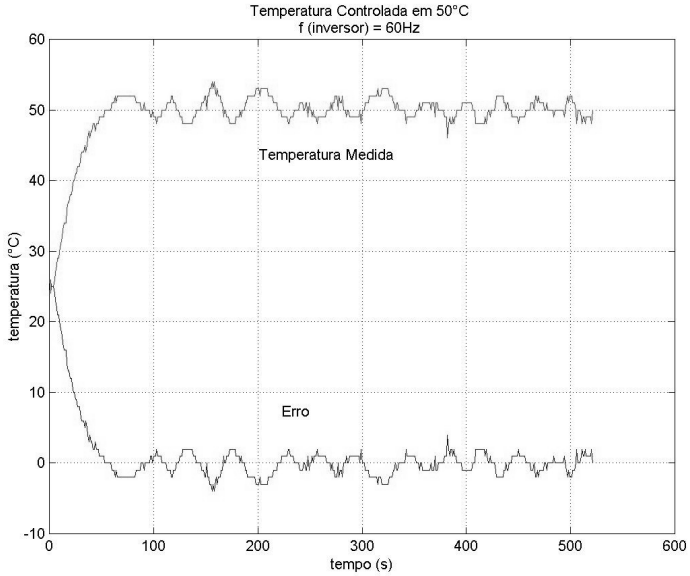



**Figura 5.3.1 – Curvas da temperatura e erro observadas para uma temperatura definida de 50°C**

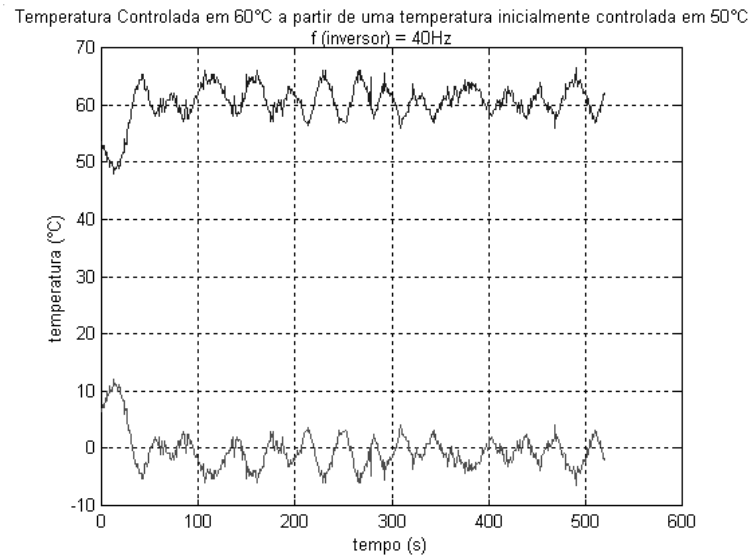



**Figura 5.3.2 – Curvas da temperatura e erro observadas para uma temperatura definida de 50°C e posteriormente alterada para 60°C**

O relé utilizado possui um circuito tipo detetor de zero, isto é, ele é acionado quando a onda senoidal da rede passa pela tensão de 0V. Assim foi utilizado um sinal PWM de baixa freqüência. Com o sinal em nível baixo, o controle permite a passagem da tensão da rede normalmente para a resistência do túnel e com o sinal em nível alto, não há tensão sendo aplicada na resistência. Os sinais de tensão da rede aplicada ao túnel de vento e do PWM podem ser vistos na figura 5.3.3.

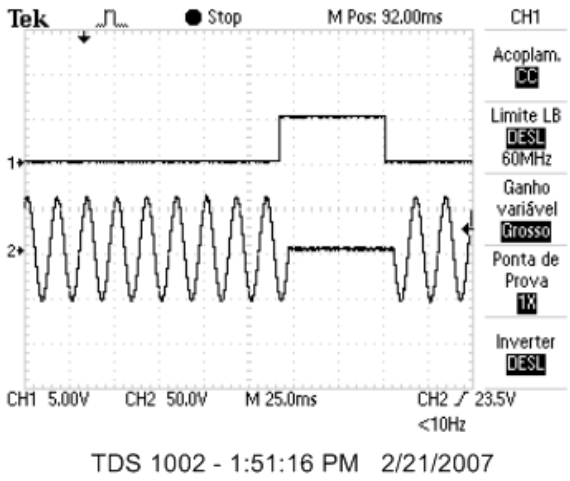



**Figura 5.3.3 – Forma de onde PWM (1) e tensão da rede aplicada a resistência (2)**

# 6 - Conclusão

O sistema de controle de temperatura desenvolvido apresentou um resultado bastante satisfatório. Um cuidado que se deve ter é que ele sofre influência da temperatura externa. Os erros observados para uma temperatura ambiente de 21°C variam entre –3°C e +3°C aproximadamente e para temperatura ambiente de 25°C a variação do erro é aproximadamente de –5°C até +6°C.

A temperatura ambiente influencia o *overshoot* do sistema. Com a sala mais quente observa-se um *overshoot* maior.

A constante de tempo do circuito controlado é bem menor significando uma resposta mais rápida do sistema. No sistema não controlado a constante de tempo é de 41,33 segundos e no sistema controlado é de 26 segundos.

As medidas observadas demonstraram que a freqüência do inversor não tem influência sobre o controle, isto ocorre provavelmente porque os valores utilizados (30Hz a 60Hz) se encontram próximos ao *joelho* da curva da constante de tempo do sensor. Se a velocidade do vento fosse significativamente menor ou maior, o sistema apresentaria respostas diferentes às observadas.

Os circuitos utilizados no projeto são circuitos de baixo custo e facilmente encontrados.

Como este projeto utiliza um microcontrolador para fazer a aquisição dos dados e controle da saída, ele é facilmente adaptado para outras funções, já que o microcontrolador é um componente que possibilita diversos usos e ainda dispõe de comunicação direta com o computador.

As linguagens utilizadas na elaboração dos programas (linguagem C e Basic) são linguagens já bem estabelecidas atualmente, o que facilita sua utilização pois existem muitas fontes de informações sobre elas.

A interação do usuário com o sistema é bem simples e direta, o que facilita bastante seu uso.

Para melhorar ainda mais o controle do sistema poderia futuramente desenvolver um sistema com dois sensores, um mais próximo da resistência, o que reduziria o tempo de resposta e conseqüentemente melhorava o funcionamento do controle. Também poderia ser utilizado um controle tipo PID (proporcional – controlador - derivativo) que é um pouco mais complexo, mas deve apresentar uma resposta melhor.

O projeto se encontra, até a presente data, disponível para uso pelo pessoal do laboratório de turbulência do Centro de Tecnologia da UFRJ, sala I - 138.

# Bibliografia


[1] Mohan, N.; Undeland, T.M.; Robins, W.P.; *Power Electronics – Converters, Applications, and Design*, John Wiley & Sons, New York, 1995, ISBN: 0-471-58408-8.

[2] National Instruments; *LabWindows/CVI – Getting Started with Labwindows/CVI*, September 2004 Edition.

[3] Boylestad, R.; Nashelsky, L.; Dispositivos Eletrônicos e Teoria de Circuitos, Prentice-Hall do Brasil, Rio de Janeiro, 1984,

[4] Torres, Gabriel; *Hardware – Curso Completo*, Axcell Books do Brasil, Rio de Janeiro, 1998, ISBN: 85-7323-087-8.

[5] Dorf, Richard C.; Bishop, Robert H.; *Sistemas de Controle Modernos*, 8° Edição, ED. Ltc, Rio de Janeiro 1998.

[6] NATIONAL. LM35 – Precision Centigrade Temperature Sensor Datasheet. National Semiconductor Inc, 1986.

[7] ATMEL. Atmega8/Atmega8L Datasheet. Atmel Corporation, 2006

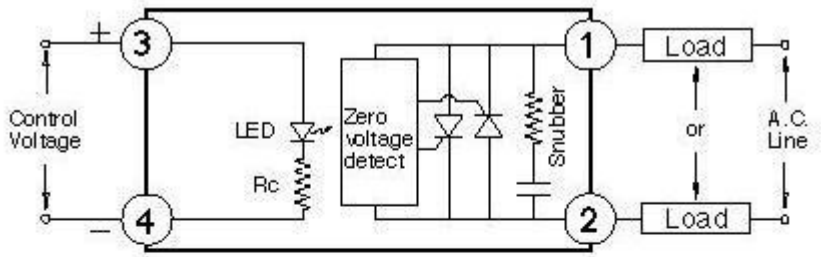
[8] Manitoba HVDC Research Center. *PSCAD on-line help*.

[9] Schildt, Herbert; *C – Completo e Total*, Makron Books, 3° Edição, São Paulo, 1996, ISBN: 85-346-0595-5.

# Anexos

## 1. Dados do Relé

Diagrama do relé modelo S505-0SJ640-000 desenvolvido pela *Continental Industries Inc*.




**Impedância (Rc) = 1500Ω**

**Turn_on = 3Vdc - 32Vdc**

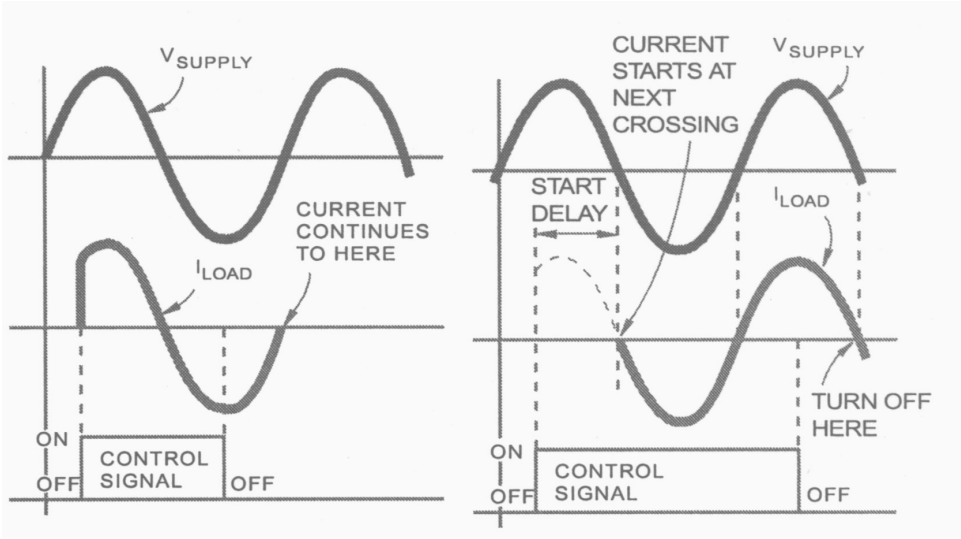
**Turn_off = 1Vdc (max)**

**Vout = 24Vac - 330Vac**

**Operação em modo síncrono**

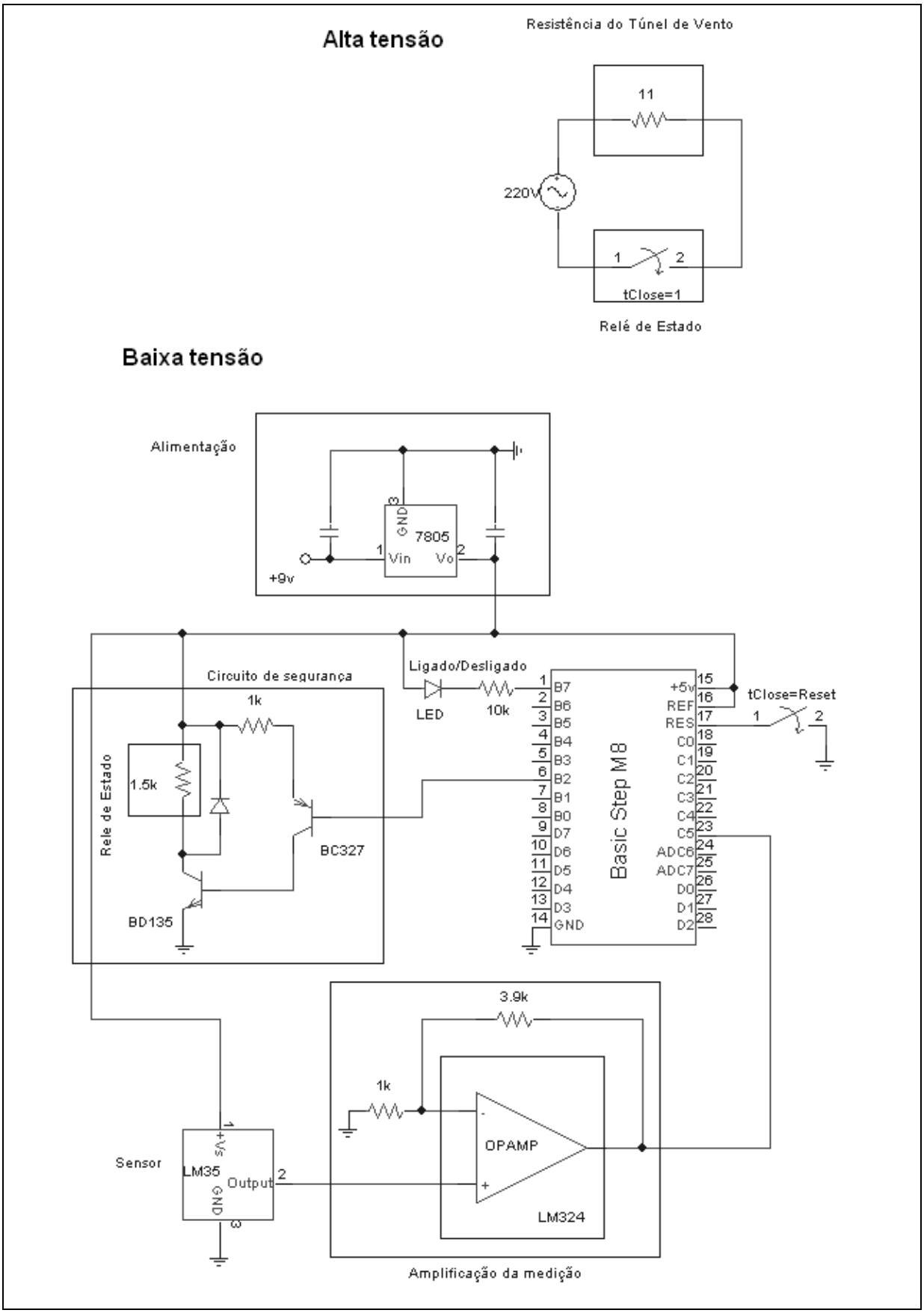
**(http://www.ciicontrols.com/products/s505.htm)**

Exemplos de modos de operação de um relé de estado sólido: (a) assíncrono, (b) síncrono




**(a) Assíncrono** **(b) Síncrono**

## 2. Diagrama do circuito



# 3. Programa gravado no microcontrolador BasicStep M8

```
XMIT INIT 9600
RECV INIT 9600
RECV INTERRUPT ON
MAKEOUT B,2
MAKEOUT B,7
u0% = 0
u1% = 2
e1% = 0
e0% = 0
se0 = 0
TEMP% = 511
ki% = 2
kp% = 8
y% = 1023
PWMB INIT 8, 10
PWMB THRESHOLD y%

DO
        A2D x%, 5, IDLE
        IF x% = TEMP% THEN
                IF se0 = 0 THEN
                        de% = e0%
                        dutemp1% = kp% * de%
                        IF u0% > dutemp1% THEN
                                u1% = u0% - dutemp1%
                        ELSE
                                SHIFT u0%, 1, RIGHT
                                u1% = u0%
                        END IF
                END IF
                IF se0 = 1 THEN
                        de% = e0%
                        dutemp1% = kp% * de%
                        u1% = u0% + dutemp1%
                END IF
        END IF
        IF x% > TEMP% THEN
                se1 = 1
                e1% = x% - TEMP%
                IF se0 = 1 THEN
                        IF e1% = e0% THEN
```

```
                    dutemp2% = ki% * e1%
                    IF u0% > dutemp2% THEN
                        u1% = u0% - dutemp2%
                    ELSE
                        SHIFT u0%, 1, RIGHT
                        u1% = u0%
                    END IF
                END IF
                IF e1% > e0% THEN
                    de% = e1% - e0%
                    dutemp1% = kp% * de%
                    dutemp2% = ki% * e1%
                    IF u0% > du% THEN
                        u1% = u0% - du%
                    ELSE
                        SHIFT u0%, 1, RIGHT
                        u1% = u0%
                    END IF
                END IF
                IF e0% > e1% THEN
                    de% = e0% - e1%
                    dutemp1% = kp% * de%
                    dutemp2% = ki% * e1%
                    IF dutemp1% > dutemp2% THEN
                        du% = dutemp1% - dutemp2%
                        u1% = u0% + du%
                    END IF
                    IF dutemp2% > dutemp1% THEN
                        du% = dutemp2% - dutemp1%
                        IF u0% > du% THEN
                            u1% = u0% - du%
                        ELSE
                            SHIFT u0%, 1, RIGHT
                            u1% = u0%
                        END IF
                    END IF
                    IF dutemp1% = dutemp2% THEN
                        u1% = u0%
                    END IF
                END IF
            END IF
            IF se0 = 0 THEN
                de% = e1% - e0%
                dutemp1% = kp% * de%
```

```
                    dutemp2% = ki% * e1%
                    du% = dutemp1% + dutemp2%
                    IF u0% > du% THEN
                            u1% = u0% - du%
                    ELSE
                            SHIFT u0%, 1, RIGHT
                            u1% = u0%
                    END IF
            END IF
    END IF
    IF TEMP% > x% THEN
            se1 = 0
            e1% = TEMP% - x%
            IF se0 = 0 THEN
                    IF e1% = e0% THEN
                            dutemp2% = ki% * e1%
                            u1% = u0% + dutemp2%
                    END IF
                    IF e1% > e0% THEN
                            de% = e1% - e0%
                            dutemp1% = kp% * de%
                            dutemp2% = ki% * e1%
                            du% = dutemp1% + dutemp2%
                            u1% = u0% + du%
                    END IF
                    IF e0% > e1% THEN
                            de% = e0% - e1%
                            dutemp1% = kp% * de%
                            dutemp2% = ki% * e1%
                            IF dutemp2% > dutemp1% THEN
                                    du% = dutemp2% - dutemp1%
                                    u1% = u0% + du%
                            END IF
                            IF dutemp1% > dutemp2% THEN
                                    du% = dutemp1% - dutemp2%
                                    IF u0% > du% THEN
                                            u1% = u0% - du%
                                    ELSE
                                            SHIFT u0%, 1, RIGHT
                                            u1% = u0%
                                    END IF
                            END IF
                            IF dutemp1% = dutemp2% THEN
                                    u1% = u0%
```

```
                                    END IF
                            END IF
                    END IF
                    IF se0 = 1 THEN
                            de% = e1% + e0%
                            dutemp1% = kp% * de%
                            dutemp2% = ki% * e1%
                            du% = dutemp1% + dutemp2%
                            u1% = u0% + du%
                    END IF
            END IF
            IF u1% < 2 THEN
                    u1% = 2
            END IF
            IF u1% > 800 THEN
                    u1% = 800
            END IF
            y% = 1024 - u1%
            PWMB THRESHOLD y%
            e0% = e1%
            se0 = se1
            u0% = u1%
            SHIFT x%, 2, RIGHT
            temperatura = x%
            XMIT OUT temperatura
            TOGGLE B,7
            IF rflag = 1 THEN
                    TEMP% = rbyte * 10.08
                    rflag = 0
            END IF
            PAUSE 1000
LOOP

INTERRUPT RECV
            PUSHFLAGS
            PUSHREG
            rflag = 1
            RECV IN rbyte, errflag
            POPREG
            POPFLAGS
END INTERRUPT
```

# 4. Programas desenvolvidos em CVI

## *4.1 M8TMP.c*

```
#include <utility.h>
#include <ansi_c.h>
#include <rs232.h>
#include <formatio.h>
#include <string.h>
#include <userint.h>
#include "M8TMP.h"

short int n2,
          ler_bytes;
int panel_handle,
          erro,
          flag = 0,
          porta = 0,
          step = 0,
          i,
          resetbox = 0,
          datafile,
          logtime,
          log_count = 0,
          log_flag = 0,
          log_A = 0,
          inqlen;
static int panelHandle;
char escrita[80],
          porta_serial[30],
          passo[10],
          entrada[30],
          transfer[30],
          resultstr[500],
          logfilename[100];
double fractpart,
          n,
          intpart,
          fractpartB,
          intpartB;
```

```
int main (int argc, char *argv[])
{
	if (InitCVIRTE (0, argv, 0) == 0)	/* Needed if linking in external compiler; harmless otherwise */
		return -1;		/* out of memory */
	if ((panel_handle = LoadPanel (0, "M8TMP.uir", PANEL)) < 0)
		return -1;
	DisplayPanel (panel_handle);
	RunUserInterface ();
	return 0;
}

int CVICALLBACK StartCallBack (int panel, int control, int event,
		void *callbackData, int eventData1, int eventData2)
{
	int ligado;
	char setupstr[200], dirname[MAX_PATHNAME_LEN];
	switch (event) {
		case EVENT_COMMIT:
			GetCtrlVal (panel_handle, PANEL_LIGADO, &ligado);
			if (ligado) {
				GetCtrlVal (panel_handle, PANEL_LOG, &logtime);
				GetProjectDir (dirname);
				sprintf (logfilename,"%s\\results.log", dirname);
				  datafile = OpenFile (logfilename,VAL_WRITE_ONLY,VAL_TRUNCATE,
VAL_ASCII);
				   sprintf (setupstr,"Arquivo de log do programa de Monitoracao de
Temperatura \n----------\n Criado em:	Data: %s Hora: %s \n\n", DateStr(),
TimeStr());
				WriteFile (datafile, setupstr, StringLength (setupstr));
				SetCtrlAttribute (panel_handle, PANEL_OK, ATTR_DIMMED, 0);
				strcpy(&porta_serial[0], "COM1");
				erro = OpenComConfig (1, porta_serial, 9600, 0, 8, 1, 512, 512);
				porta = 1;
				}
			else  {
				SetCtrlAttribute (panel_handle, PANEL_OK, ATTR_DIMMED, 1);
				erro = CloseCom (1);
				step = 0;
				flag=0;
				}
			break;
	}
	return 0;
```

```
}

int CVICALLBACK QuitCallBack (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
          switch (event) {
                    case EVENT_COMMIT:
                              if (porta == 0)
                                erro = CloseCom (1);
                              QuitUserInterface (0);
                              break;
          }
          return 0;
}

int CVICALLBACK EscreveCallBack (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
          switch (event) {
                    case EVENT_COMMIT:  {
                              flag=1;
                              }
                              break;
          }
          return 0;
}



          void UpdateTextBoxA (char* event_gen)
{
   char display_str[30];

   Fmt(display_str, "%s<%s\n", event_gen);
          SetCtrlVal (panel_handle, PANEL_TEXTBOXA, display_str);

}


int CVICALLBACK TimerCallBack (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
          switch (event) {
                    case EVENT_TIMER_TICK:
                              if (flag == 1) {
                              ler_bytes = ComRdByte (1);
```

```
                        sprintf(transfer,"%u",ler_bytes);
                        n = atof(transfer);
                        n = n/2.5;
                        fractpart = modf (n , &intpart);
                        fractpartB = fractpart * 10;
                        fractpartB = modf (fractpartB , &intpartB);
                        if (fractpartB < 0.5 ) {
          n = intpart + (intpartB / 10);
            }
          else {
          n = intpart + ((intpartB + 1) / 10);
          }
                        sprintf (transfer, "%0.1f", n);
                        sprintf (passo, "%0.1u", step);
                        step = step + 1;
                        UpdateTextBoxA(transfer);
                        log_count = log_count + 1;
                                if (log_count >= logtime) {
                                        sprintf (resultstr, "%s          %s\n", passo, transfer);
                                        WriteFile (datafile, resultstr, StringLength(resultstr));
                                        log_count = 0;
                                        }
                        resetbox = resetbox + 1;
                        if (resetbox == 20) {
                                ResetTextBox (panel_handle, PANEL_TEXTBOXA, "");
                                resetbox = 0;
                                }
                        inqlen = GetInQLen (1);
                        SetCtrlVal (panel_handle, PANEL_Buffer, inqlen);
                        if (inqlen > 10)
                                FlushInQ (1);
                                }
          }
          return 0;
}

int CVICALLBACK ViewDataFile (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
          char tempstr[100];

          switch (event) {
                case EVENT_COMMIT:
                        CloseFile (datafile);
                        sprintf (tempstr,"Notepad.exe %s", logfilename);
```

```
                    LaunchExecutable (tempstr);
                    datafile = OpenFile (logfilename,VAL_WRITE_ONLY,VAL_APPEND,VAL_ASCII);
        }
        return 0;
}

int CVICALLBACK LOGTIME (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
        switch (event) {
                    case EVENT_COMMIT:
                                GetCtrlVal (panel_handle, PANEL_LOG, &logtime);
                                break;
        }
        return 0;
}

int CVICALLBACK SerialCallBack (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
        switch (event) {
                    case EVENT_COMMIT:
                    GetCtrlVal (panel_handle, PANEL_Janela_2, &n2);
                    ComWrtByte (1,n2);
                    break;
        }
        return 0;
}
```

## *4.2 M8TMP.h (headrers, gerado automáticamente)*

```
/*****************************************************************************/
/* LabWindows/CVI User Interface Resource (UIR) Include File                 */
/* Copyright (c) National Instruments 2006. All Rights Reserved.             */
/*                                                                           */
/* WARNING: Do not add to, delete from, or otherwise modify the contents     */
/*          of this include file.                                            */
/*****************************************************************************/

#include <userint.h>

#ifdef __cplusplus
```

```
   extern "C" {
#endif

    /* Panels and Controls: */

#define  PANEL                    1
#define  PANEL_LIGADO             2       /* callback function: StartCallBack */
#define  PANEL_OK                 3       /* callback function: EscreveCallBack */
#define  PANEL_QUIT               4       /* callback function: QuitCallBack */
#define  PANEL_TEXTBOXA           5
#define  PANEL_Log                6       /* callback function: ViewDataFile */
#define  PANEL_LOG                7       /* callback function: LOGTIME */
#define  PANEL_Buffer             8
#define  PANEL_Janela_2           9
#define  PANEL_COMMANDBUTTON      10      /* callback function: SerialCallBack */
#define  PANEL_Tempo              11      /* callback function: TimerCallBack */


    /* Menu Bars, Menus, and Menu Items: */

        /* (no menu bars in the resource file) */


    /* Callback Prototypes: */

int  CVICALLBACK EscreveCallBack(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int  CVICALLBACK LOGTIME(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int  CVICALLBACK QuitCallBack(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int  CVICALLBACK SerialCallBack(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int  CVICALLBACK StartCallBack(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int  CVICALLBACK TimerCallBack(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int  CVICALLBACK ViewDataFile(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);


#ifdef __cplusplus
   }
#endif
```

## *4.3 M8TMP.prj (arquivo gerado automaticamente)*

[Project Header]
Version = 401
Platform Code = 4
Pathname = "/C/sandro/Projeto/M8TMP/M8TMP.prj"
CVI Dir = "/c/sandro/projeto/cvi401"
VXIplug&play Framework Dir = "/c/vxipnp/winnt"
Number of Files = 3
Sort Type = "No Sort"
Target Type = "Executable"
Flags = 16
Drag Bar Left = 136
Window Top = 213
Window Left = 300
Window Bottom = 357
Window Right = 844

[File 0001]
File Type = "CSource"
Path = "/C/sandro/Projeto/M8TMP/M8TMP.c"
Res Id = 1
Exclude = False
Disk Date = 3247750248
Project Flags = 0
Window Top = 30
Window Left = 0
Window Height = 0
Window Width = 0
Source Window State = "1,200,207,200,0,-1,0,0,0,124,0,3,0,3,0,37,161,0,197,0,"
Header Dependencies = "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,"

[File 0002]
File Type = "Include"
Path = "/C/sandro/Projeto/M8TMP/M8TMP.h"
Res Id = 2
Exclude = False
Disk Date = 3244024826
Project Flags = 0
Window Top = 152
Window Left = 70
Window Height = 0
Window Width = 0
Source Window State = "1,0,0,0,0,0,0,0,0,80,0,0,0,0,0,25,0,0,7,64,"

[File 0003]
File Type = "User Interface Resource"
Path = "/C/sandro/Projeto/M8TMP/M8TMP.uir"
Res Id = 3
Exclude = False
Disk Date = 3248358078
Project Flags = 0
Window Top = 86
Window Left = 63
Window Height = 384
Window Width = 760

[Compiler Options]
Default Calling Convention = "cdecl"
Max Number Of Errors = 10
Require Prototypes = True
Require Return Values = True
Enable Pointer Mismatch Warning = False
Enable Unreachable Code Warning = False
Track Include File Dependencies = True
Prompt For Missing Includes = True
Stop On First Error File = False
Bring Up Err Win For Warnings = True
Show Build Dialog = False

[Run Options]
Stack Size = 100000
Debugging Level = "None"
Save Changes Before Running = "Ask"
Break On Library Errors = True
Hide Windows = False
Unload DLLs After Each Run = True
Check Disk Dates Before Each Run = True
Break At First Statement = False

[Compiler Defines]
Compiler Defines = "/DWIN32_LEAN_AND_MEAN"

[Command Line Args]
Command Line Args = ""

[Included Headers]
Header 0023 = "/C/sandro/Projeto/M8TMP/M8TMP.h"
Header 0001 = "/c/sandro/projeto/cvi401/include/utility.h"
Header 0002 = "/c/sandro/projeto/cvi401/include/cvidef.h"

Header 0003 = "/c/sandro/projeto/cvi401/include/cvirte.h"
Header 0004 = "/c/sandro/projeto/cvi401/include/ansi_c.h"
Header 0005 = "/c/sandro/projeto/cvi401/include/ansi/assert.h"
Header 0006 = "/c/sandro/projeto/cvi401/include/ansi/ctype.h"
Header 0007 = "/c/sandro/projeto/cvi401/include/ansi/errno.h"
Header 0008 = "/c/sandro/projeto/cvi401/include/ansi/float.h"
Header 0009 = "/c/sandro/projeto/cvi401/include/ansi/limits.h"
Header 0010 = "/c/sandro/projeto/cvi401/include/ansi/locale.h"
Header 0011 = "/c/sandro/projeto/cvi401/include/ansi/math.h"
Header 0012 = "/c/sandro/projeto/cvi401/include/ansi/setjmp.h"
Header 0013 = "/c/sandro/projeto/cvi401/include/ansi/signal.h"
Header 0014 = "/c/sandro/projeto/cvi401/include/ansi/stdarg.h"
Header 0015 = "/c/sandro/projeto/cvi401/include/ansi/stddef.h"
Header 0016 = "/c/sandro/projeto/cvi401/include/ansi/stdio.h"
Header 0017 = "/c/sandro/projeto/cvi401/include/ansi/stdlib.h"
Header 0018 = "/c/sandro/projeto/cvi401/include/ansi/string.h"
Header 0019 = "/c/sandro/projeto/cvi401/include/ansi/time.h"
Header 0020 = "/c/sandro/projeto/cvi401/include/rs232.h"
Header 0021 = "/c/sandro/projeto/cvi401/include/formatio.h"
Header 0022 = "/c/sandro/projeto/cvi401/include/userint.h"
Max Header Number = 23

[Create Executable]
Executable File = "/C/sandro/Projeto/M8TMP/temperatura.exe"
Icon File = ""
Application Title = ""
DLL Exports = "Include File Symbols"
DLL Import Library Choice = "Gen Lib For Current Mode"
Use VXIPNP Subdirectories for Import Libraries = False
Use Dflt Import Lib Base Name = True
Add Type Lib To DLL = False
Include Type Lib Help Links = False
Type Lib FP File = ""
Instrument Driver Support Only = False

[External Compiler Support]
Create UIR Callbacks File = False
Using LoadExternalModule = False
Create Project Symbols File = True
UIR Callbacks Obj File = ""
Project Symbols H File = ""
Project Symbols Obj File = ""

[Distribution Kit]
Installation Directory = "M8TMP"

Install Run-Time Engine = True
Install Low-Level Support Driver = True
Media Size = 3
kBytes Reserved on First Disk = 0
Target Path = "/C/sandro/Projeto/M8TMP"
Language = "English"
Core Group Index = 1
DLL Group Index = -1
Project File 0001 = "/C/sandro/Projeto/M8TMP/M8TMP.uir"
Project File 0002 = "/C/sandro/Projeto/M8TMP/temperatura.exe"
Use Custom Script = False
Run Executable After Setup Group Index = -1
Run Executable After Setup File Index = -1
Use Default Program Group Name = True
Use Default Installation Name = True

[Distribution Kit File Group 001]
Group Name = "temperatura Files"
Destination Directory = "Application"
Use Relative Path = False
Install Icons = False
Distribute Multiple Objects = False
Replace Mode = "Ask"
File 0001 = "/C/sandro/Projeto/M8TMP/M8TMP.uir"
File 0002 = "/C/sandro/Projeto/M8TMP/temperatura.exe"

[Modules Forced Into Executable]
Module 0001 = "analysis.lib"
Module 0002 = "easyio.lib"
Module 0003 = "nivxi.lib"
Module 0004 = "gpib.lib"
Module 0005 = "visa.lib"
Module 0006 = "gdi32.lib"
Module 0007 = "kernel32.lib"
Module 0008 = "user32.lib"