Curso de Engenharia de Computação

CONTROLADOR PID DIGITAL DE VELOCIDADE DE UM MOTOR DE CORRENTE CONTÍNUA

Cleber Zorzi

Itatiba – São Paulo – Brasil
Dezembro de 2004

O presente exemplar da monografia CONTROLADOR PID DIGITAL DE VELOCIDADE DE UM MOTOR DE CORRENTE CONTÍNUA contempla as correções sugeridas pela banca examinadora durante a apresentação do Trabalho de Conclusão de Curso.

Itatiba/SP, 07 de Dezembro de 2004.

Prof Ms Paulo Eduardo Silveira

USF – Universidade São Francisco – Itatiba – SP.

Curso de Engenharia de Computação

CONTROLADOR PID DIGITAL DE VELOCIDADE DE UM MOTOR DE CORRENTE CONTÍNUA

Cleber Zorzi

Monografia apresentada à disciplina Trabalho de Conclusão de Curso, do Curso de Engenharia de Computação da Universidade São Francisco, sob a orientação do Prof. Ms. Paulo Eduardo Silveira, como exigência parcial para conclusão do curso de graduação.

Orientador: Prof. Ms. Paulo Eduardo Silveira

Itatiba – São Paulo – Brasil
Dezembro de 2004

Controlador PID Digital de Velocidade de um Motor de Corrente Contínua

Cleber Zorzi

Monografia defendida e aprovada em 01 de dezembro de 2004 pela **Banca Examinadora** assim constituída:

Prof Ms Paulo Eduardo Silveira

USF – Universidade São Francisco – Itatiba – SP.

Prof Ms Alencar de Melo Junior

USF – Universidade São Francisco – Itatiba – SP.

Prof Ms Josemar dos Santos

USF – Universidade São Francisco – Itatiba – SP.

A luta contra o erro tipográfico tem algo de homérico. Durante a revisão os erros se escondem, fazem-se positivamente invisíveis. Mas assim que o livro sai, tornam-se visibilíssimos.

(Monteiro Lobato)

Dedicatória

A meus pais Eliseu e Carmem, sem os quais não chegaria até aqui.

A minha esposa Queli, que me ensinou a fé e o amor.

Agradecimentos

Agradeço primeiramente ao Professor Paulo E. Silveira, meu orientador, que acreditou em mim e incentivou-me para a conclusão deste trabalho, face aos inúmeros percalços do trajeto.

Agradeço também todos os professores que me possibilitaram ter um crescimento acadêmico, profissional e principalmente pessoal.

Alguns experimentos e vários “entendimentos” não teriam sido possíveis sem a colaboração de Caio M. Franco, Thales de T. Cezare, Filipe (empresa T&S), Paulo F. Madiutto, Marcelo e Celso Godoi.

Eu agradeço fraternalmente a todos que me ajudaram.

Sumário

Lista de Siglas ..... **x**

Lista de Figuras ..... **xi**

Lista de Tabelas ..... **xii**

Resumo ..... **xiii**

Abstract ..... **xiii**

1 INTRODUÇÃO **1**

2 DESCRIÇÃO DO SISTEMA **2**
2.1 Microcontrolador 2
2.1.1 PIC16F877A 4
2.2 PWM 4
2.3 Encoder 5
2.3.1 Funcionamento 6
2.3.2 Características 6
2.3.3 Tipos de *encoders* 7
2.4 Controle 7
2.4.1 Controlador 8
2.4.1.1 Proporcional 8
2.4.1.2 Integral 9
2.4.1.3 Derivativo 9
2.4.1.4 PID 9
2.5 Sistema 10
2.5.1 Hardware 10
2.5.2 Software 11

3 RESULTADOS **14**

4 DIFICULDADES ENCONTRADAS **17**

5 PRÓXIMOS TRABALHOS **18**

6 CONCLUSÕES **19**

Referências Bibliográficas **20**

Bibliografia consultada **21**

Anexo 1 – Esquema elétrico da ligação do microcontrolador. **22**

Anexo 2 – Ligação entre o motor e o *encoder*. 23

Anexo 3 – Algoritmo de controle implementado no microcontrolador. 24

Anexo 4 – Algoritmo do supervisório implementado no microcomputador. 26

Lista de Siglas

A/D	*Analogical/Digital*
CCP	*Capture, Compare, PWM*
CPU	*Center Processing Unit*
EEPROM	*Electrically Erasable Programmable Read-only Memory*
GND	*Ground*
Hz	*Hertz*
HTL	*High Threshold Logic*
LCD	*Liquid Crystal Display*
PID	*Proportional, Integral, Derivative*
PWM	*Pulse Width Modulation*
RISC	*Reduced Instruction Set Computer*
TTL	*Transistor-Transistor-Logic*
Vcc	*Voltage Current Continue*

Lista de Figuras

FIGURA 2-1 - DIAGRAMA DE BLOCOS DE UM MICROCONTROLADOR.....3
FIGURA 2-2 - QUANDO LIGA E DESLIGA O INTERRUPTOR, É CONTROLADA A CORRENTE NA CARGA. 4
FIGURA 2-3 - CONTROLANDO A POTÊNCIA DO CICLO ATIVO. 5
FIGURA 2-4 - VISUALIZAÇÃO INTERNA DE UM ENCODER. 5
FIGURA 2-5 - SINAL DO *ENCODER* TIPO INCREMENTAL E TIPO ABSOLUTO. 7
FIGURA 2-6 - DIAGRAMA QUE MOSTRA UM SISTEMA DE CONTROLE DO TIPO MALHA FECHADA. ..7
FIGURA 2-7 - TELAS DO PROGRAMA DESENVOLVIDO PARA O MICROCOMPUTADOR (A) TELA PRINCIPAL, (B) TELA PARA INSERIR NOVOS VALORES DE CONSTANTES E (C) TELA PARA INSERIR A VELOCIDADE DESEJADA. 13
FIGURA 3-1 – TELA DO HYPERTERMINAL TESTANDO A COMUNICAÇÃO ENTRE O MICROCONTROLADOR E O MICROCOMPUTADOR. 16

Lista de Tabelas

TABELA 2-1 - TABELA COM AS CARACTERÍSTICAS DO MICROCONTROLADOR 16F877A. 4
TABELA 3-1 - COMPARAÇÃO COM OS VALORES GERADOS E VALORES MEDIDOS UTILIZANDO O MICROCONTROLADOR. 14

ZORZI, Cleber. Controlador PID Digital de Velocidade de um Motor de Corrente Contínua. 2004. 25f. Monografia (Engenharia de Computação) – Curso de Engenharia de Computação da Universidade São Francisco, Câmpus de Itatiba.

Resumo

O objetivo dessa monografia é o projeto e a implementação de um sistema de aquisição de dados e controle autônomo baseado em um microcontrolador da linha PIC (Microchip) e uma interface de comunicação com um computador. Para o funcionamento desta interface foi desenvolvido um software supervisório que também atua no controle do sistema.O sistema de controle foi desenvolvido utilizando a lógica de controle PID (Proporcional Integral Derivativo).

PALAVRAS-CHAVE: aquisição, controlador, controle, microcontrolador, processo.

Abstract

The objective of this monograph is the project and the implementation of a system of acquisition of data and independent control based in a microcontroller of line PIC (Microchip) and an interface of communication with a computer. For the functioning of this interface supervisory software was developed that also acts in the control of system. The control system was developed using the logic of control PID (Proportional Integral Derivative).

KEY WORDS: acquisition, controller, control, microcontroller, process.

1 INTRODUÇÃO

A aquisição de dados engloba métodos e dispositivos capazes de transformar informações do mundo real, preponderantemente analógicas, para o formato digital, com o qual os computadores trabalham. Um sistema de aquisição de dados é composto por um ou mais dispositivos de entrada gerando dados para um computador (ou uma rede de computadores), capaz de interpretá-los como grandezas físicas, requerendo para isto, o software adequado [SILVA JR.].

Existe uma gama considerável de opções de sistemas de aquisição de dados. A escolha do sistema adequado depende essencialmente do tipo de grandeza a ser medida e do objetivo da medida. Com base nestes dois parâmetros é possível definir características como: velocidade da medição, número de grandezas distintas, exatidão e a configuração do sistema, determinando os tipos componentes utilizados.

Nos capítulos seguintes, será feita toda a descrição do sistema, resultados obtidos, dificuldades encontradas, sugestões para próximos trabalhos, conclusão e referências bibliográficas.

2 DESCRIÇÃO DO SISTEMA

Este capítulo descreve todos os componentes utilizados no sistema e suas respectivas características e formas de atuação. Também são descritos todos os algoritmos utilizados na programação do microcontrolador e do microcomputador.

O objetivo principal do projeto é a leitura de informações correspondentes à velocidade e a alteração de variáveis de controle que modifiquem o estado do motor de corrente contínua (motor CC). A informação de velocidade é transferida para o microcontrolador através de um dispositivo chamado *encoder* [CAPELLI]. Utilizando essas informações, o microcontrolador atua de forma autônoma controlando o motor. Para o controle foi implementado no microcontrolador um algoritmo do tipo *PID*. No dispositivo do microcontrolado, além do microcontrolador, conta também com um *display LCD 2x16*, quatro teclas e um *driver* para o acionamento do motor de corrente contínua. Através do *display* são visualizadas informações como estado do motor (parado ou em movimento), velocidade atual e velocidade desejada. As teclas servem de forma a atuar no controle desse motor. A variação de velocidade do motor é feita utilizando a saída *PWM* (Modulação de Largura de Pulso) [BRAGA] do microcontrolador e o *driver* de potência.

Foi implementada uma interface de comunicação com um microcomputador utilizando a entrada/saída *RS232*. O software supervisório foi implementado utilizando a linguagem de programação *C++ Builder*. O microcomputador atua diretamente no controle do motor, podendo também alterar as variáveis de controle.

2.1 Microcontrolador

Praticamente todos estamos rodeados de aparelhos eletrônicos que possuem dentro de si um microcontrolador. O microcontrolador é um componente que possui todos os periféricos dos microprocessadores comuns embutidos em uma só pastilha, facilitando assim o desenvolvimento de sistemas pequenos e baratos, embora complexos e sofisticados.

Costumam apresentar em uma única só pastilha memórias de dados (volátil) e de programas (não volátil), canal serial, temporizadores, interfaces para *displays (LCD)*, memória *EEPROM*, módulo *CCP* (Capture, Compare e PWM) e muito mais, dependendo do modelo.

O diagrama de blocos simplificado pode ser visto na Figura 2-1.

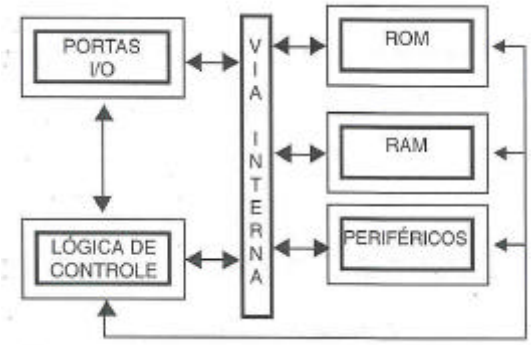


Figura 2-1 - Diagrama de blocos de um microcontrolador.

Os *PIC´s* utilizam uma arquitetura diferente conhecida como *Harvard* [SILVA JR.], que prevê várias vias de comunicação entre a *CPU* e os periféricos, permitindo a realização de várias operações simultaneamente, o que implica em um aumento considerável na velocidade de execução de uma instrução e permite ainda que a memória de dados e de programa tenha tamanhos diferentes.

Esta facilidade permite ainda que em uma única palavra de 14 *bits* tenhamos o código da instrução, onde a mesma vai atuar e o eventual operando ou dado, se houver.

Criou-se então uma terminologia chamada *RISC* (Computador com *Set* de Instruções Reduzido) [SILVA JR.] que faz com que existam poucas instruções (aproximadamente 35 instruções).

Cada instrução possui um tempo de execução de um ciclo do microcontrolador, com exceção da instrução que faz chamada a uma outra instrução (dois ciclos).

Nos microcontroladores *PIC*, o sinal do *clock* é dividido internamente por quatro, gerando as fases conhecidas como Q1, Q2, Q3 e Q4. Cada ciclo de instrução é composto das quatro fases, de forma que cada ciclo demanda então de um tempo.

A seguir é apresentada a Equação 2-1 para se calcular o tempo de cada ciclo:

$t_{cy} = \frac{1}{(F_{OSC} / 4)}$	**Equação 2-1**

onde t_{cy} é o tempo que leva para se completar um ciclo e F_{osc} é a freqüência de oscilação do circuito (*clock*).

Se o *clock* for de 4*MHz*, o período de cada fase será T=250 ηs, então cada ciclo terá a duração de 1μs.

2.1.1 PIC16F877A

O *PIC16F877A* é o principal componente utilizado neste sistema. Através dele foi desenvolvida toda a lógica de aquisição de dados e controle.

Esse microcontrolador em particular possui 40 pinos sendo 33 pinos utilizados com entrada/saída. Possui 35 palavras de instrução, *clock* de até 20*MHz*, 368 *bytes* de memória de dados, 256 *bytes* de memória programa e memória *Flash*. Os pinos de saída são compatíveis com microcontroladores de 28 pinos e 40/44 pinos dos modelos *16CXXX* e *16FXXX*.

Na Tabela 2-1 as principais características do microcontrolador.

Características	PIC16F877A
Frequência de operação	DC - 20MHZ
Resets (e atraso)	POR, BOR (PWRT, OST)
Memória de programa Flash (14 bits de palavra)	8k
Memória de dados (bytes)	368
Memória de dados EEPROM (bytes)	256
Interrupções	15
Portas de entrada/saída	Portas A, B, C, D, E
Temporizadores	3
Módulo Cappture/Compare/PWM	2
Comunicação serial	MSSP, USART
Comunicação paralela	PSP
Módulo 10 bits analógico/digital	8 canais de entrada
Comparador analógico	2
Conjunto de instruções	35 instruções
Tipo pinagem	40 pinos tipo PDIP

Tabela 2-1 - Tabela com as características do microcontrolador 16F877A.

2.2 PWM

Para definir o *PWM*, foi analogamente comparado com um interruptor, ilustrado na Figura 2-2. O interruptor fechado pode definir uma largura de pulso pelo tempo em que ele fica nesta condição, e um intervalo entre pulsos pelo tempo em que ele fica aberto. Os dois tempos juntos definem o período e, portanto, uma freqüência de controle.

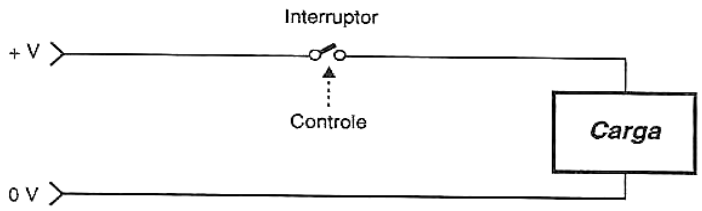


Figura 2-2 - Quando liga e desliga o interruptor, é controlada a corrente na carga.

A relação entre o tempo em que temos o pulso e a duração de um ciclo completo de operação do interruptor nos define o ciclo ativo.

Variando a largura do pulso e também o intervalo de modo a ter ciclos ativos diferentes, pode-se controlar a potência média aplicada a uma carga. Assim, quando a largura do pulso varia de zero até o máximo, a potência também varia na mesma proporção, conforme indicado na Figura 2-3.

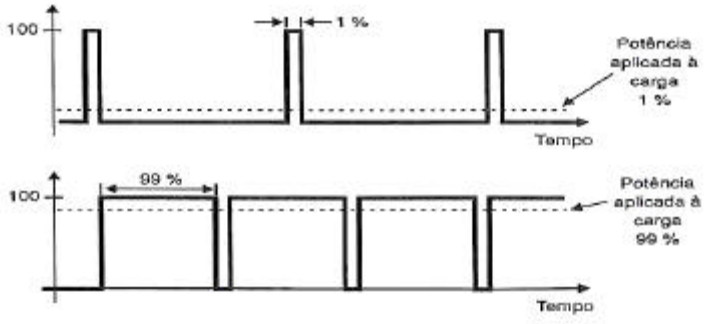


Figura 2-3 - Controlando a potência do ciclo ativo.

Este princípio é usado no controle do *PWM*: é modulada (variada) a largura do pulso de modo a controlar o ciclo do sinal aplicado a uma carga e, com isso, a potência aplicada à mesma.

2.3 Encoder

O *encoder* é um dispositivo eletrônico capaz de medir deslocamentos angulares ou lineares. Na verdade, ele não é apenas eletrônico, pois possui partes móveis mecânicas. Fisicamente se parece muito com um pequeno motor *Vcc.*

O *encoder* funciona através do processo óptico-eletrônico, muito semelhante ao *mouse* do microcomputador. Na Figura 2-4, o aspecto interno do *encoder*.

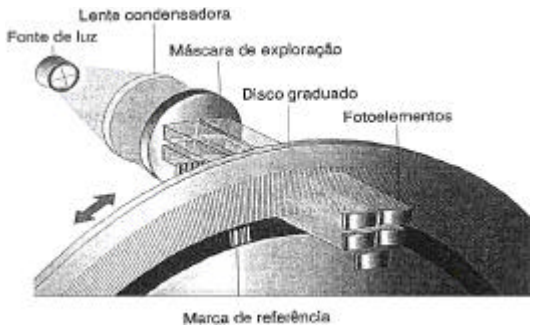


Figura 2-4 - Visualização interna de um encoder.

Ao conectar o eixo do encoder ao eixo de um motor, e submeter o motor a uma rotação, o encoder proporcionará um conjunto de sinais elétricos a cada volta do seu eixo. Esses sinais analógicos serão convertidos em pulsos digitais, podendo ser processados e “traduzidos” em medidas de deslocamento.

2.3.1 Funcionamento

Uma fonte de luz, depois de tratada por lentes, atravessa um disco perfurado. A cada janela do disco, a luz consegue atingir os foto elementos do outro lado. Como a geometria do disco é circular, a exposição dos foto elementos à luz é gradativa, isto é, a luz começa incidindo com pouca intensidade até chegar ao máximo. Neste ponto, começa a diminuir na mesma proporção. Em vista disso, as formas de onda geradas por um *encoder*, logo após os foto elementos, são senoidais. O disco possui uma “marca de referência”, que indica que uma rotação foi concluída. A maioria dos *encoders* já digitalizam os sinais através de circuitos *A/D* internos, e já proporcionam na sua saída, sinais digitais compatíveis com as tecnologias *TTL* e *HTL*. O *encoder* utilizado possui o sinal digitalizado compatível com as tecnologias *TTL* e *HTL*.

2.3.2 Características

O *encoder* [CAPELLI] possui algumas características como: resolução, graduação, precisão, interpretação e classe de precisão.

- ✓ Resolução - é o menor incremento de contagem que o *encoder* pode fornecer. Trata-se do número de pulsos emitidos por rotação. Quanto maior o número de pulsos, maior a resolução, e vice-versa. Os mais encontrados no mercado são *encoders* de 1048 a 5000 pulsos, porém isso é relativo, pois o *encoder* pode realizar o processo de interpolação. A interpolação é a multiplicação dos pulsos do encoder, que pode ser feita dentro do próprio encoder ou pelos circuitos eletrônicos que ele está conectado. A finalidade da interpolação é aumentar a resolução do sistema, visto que ela aumenta as divisões dos sinais;
- ✓ Graduação - é a distância entre as janelas da escala graduada;
- ✓ Precisão - é o erro real do *encoder;*
- ✓ Interpretação – é a contagem das bordas do sinal digitalizado, e está relacionado com a resolução podendo ser simples, dupla ou qrádrupla;
- ✓ Classe de precisão - é a faixa de erro utilizada para classificar o *encoder*.

2.3.3 Tipos de *encoders*

Existem dois tipos de *encoders*: o incremental e o absoluto.

- ✓ *Encoder* Incremental - gera pulsos seriais para a eletrônica subseqüente. Quando ele está com seu eixo parado, não há sinal algum em sua saída, portanto, para que a máquina saiba onde seu eixo está, é necessário que haja a movimentação do *encoder*;
- ✓ *Encoder* Absoluto - gera um conjunto de 6 a 8 bits de uma única vez. Ao contrário do *encoder* incremental, o absoluto (mesmo parado) tem uma "palavra" digital em seus terminais de saída. Sendo assim, uma máquina que funciona com um *encoder* absoluto não precisa movimentar seus eixos para saber onde eles estão.

A Figura 2-5 ilustra os dois tipos de *encoders*.

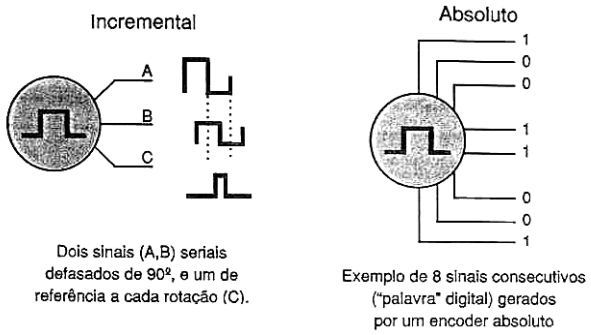


Figura 2-5 - Sinal do *encoder* tipo incremental e tipo absoluto.

2.4 Controle

O dispositivo a ser controlado é do tipo malha fechada. Num sistema de malha fechada, o sinal atuante de erro, que é a diferença entre o sinal de entrada e o sinal de retroação (sinal de saída), excita o controlador de modo a reduzir o erro e trazer o valor do sinal de saída para o valor desejado. Uma característica necessária de um sistema de malha fechada é a realimentação. A Figura 2-6 mostra o sistema de malha fechada.

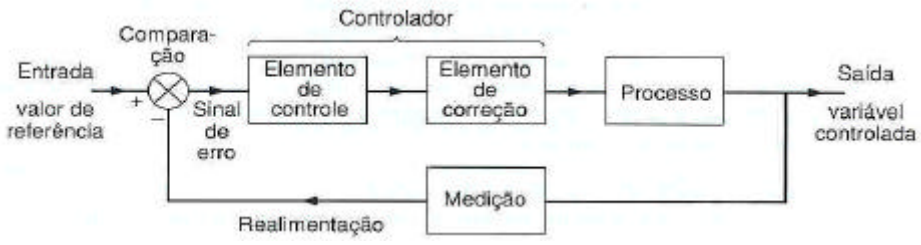


Figura 2-6 - Diagrama que mostra um sistema de controle do tipo malha fechada.

A seguir, os elementos de um sistema de malha fechada:

- ✓ Elemento de comparação: compara o valor desejado, ou de referência, da variável controlada com o valor medido e determina o sinal de erro que indica quanto o valor da saída está desviado do valor desejado;
- ✓ Elemento de correção: é usado para provocar uma mudança no processo de forma a corrigir o erro e é freqüentemente chamado de atuador;
- ✓ Processo: o processo ou planta é o sistema no qual uma variável está sendo controlada;
- ✓ Elemento de medida: gera um sinal relacionado com a condição da variável que está sendo controlada e fornece um sinal de realimentação para o elemento de comparação, para que ele determine se existe um erro.

2.4.1 Controlador

O controlador é o elemento no sistema de controle em malha fechada que tem como entrada o sinal de erro e gera uma saída que se torna à entrada para o elemento corretivo. A relação entre a saída e a entrada do controlador é chamada de *lei de controle* [BOLTON]. As formas mais usuais são: proporcional, integral, derivativa e suas combinações (PI, PD, PID). Em alguns sistemas é necessário melhorar o desempenho do controlador, o que é conseguido introduzindo-se elementos adicionais chamados compensadores nos sistemas de controle. Essa alteração é chamada de *compensação* [BOLTON].

2.4.1.1Proporcional

Com o controle proporcional, a saída do controlador é diretamente proporcional a sua entrada, sendo esta o sinal de erro ε, que é uma função do tempo. A saída do controle proporcional e expressa na Equação 2-2.

$Saída = kp * e$	**Equação 2-2**

onde *kp* é uma constante chamada ganho proporcional.

A saída do controlador depende apenas da amplitude do erro no instante de tempo. O controlador é apenas um amplificador com um ganho constante. Um grande erro em algum instante de tempo acarreta um valor alto na saída do controlador nesse instante de tempo. O ganho constante tende a existir somente para uma certa faixa de erros, chamada de *banda proporcional* [BOLTON].

2.4.1.2 Integral

O controle integral, a saída do controlador é proporcional a integral do sinal de erro ε com o tempo. A saída do controle integral é expressa na Equação 2-3.

$Saída = ki\int_0^t e * dt$	**Equação 2-3**

onde *ki* é uma constante chamada de ganho integral.

A integral entre *0* e *t* é de fato a área sob a curva do erro entre *0* e *t.* Assim quando aparece o sinal de erro, a área sob a curva aumenta em uma razão regular e a saída do controlador deve também aumentar em uma razão regular. A saída em qualquer instante de tempo é proporcional ao acúmulo de efeitos do erro em instantes anteriores.

2.4.1.3 Derivativo

O controle derivativo, a saída do controlador é proporcional a taxa de variação do erro ε com o tempo. A saída do controle derivativo é expressa na Equação 2-4.

$Saída = kd * \frac{de}{dt}$	**Equação 2-4**

onde *kd* é uma constante chamada de ganho derivativo.

Com o controle derivativo, tão logo o sinal de erro apareça, a saída do controlador pode tornar-se grande, já que a saída é proporcional a taxa de variação do sinal de erro e não do erro propriamente dito. Isto pode fornecer uma grande ação corretiva antes que um grande sinal de erro ocorra. Entretanto, se o erro é uma constante, então não existe ação corretiva, mesmo que o erro seja grande. O controle derivativo é insensível a sinais de erro constantes ou de variação lenta, e conseqüentemente não é usado sozinho, mas combinado com outras formas de controle.

2.4.1.4 PID

Um controlador proporcional mais integral mais derivativo *(PID),* ou controlador de três termos terá uma saída para uma entrada de um erro ε. A saída do controle proporcional mais integral mais derivativo é expressa na Equação 2-5.

$Saída = (kp * e) + (ki\int_0^t e * dt) + (kd * \frac{de}{dt})$	**Equação 2-5**

A seguir a discretização da Equação 2-5 utilizado no algoritmo do microcontrolador.

```
 watual = conta_pulso; // velocidade atual gerado pelo encoder.
e = wdes - watual;  // erro é igual a veloc. desejada menos a veloc. atual.
prop = (e * kp)/10; // proporcional.
aatual = ((e + eant) / 2) * dt; // soma dos erros multiplicado pelo tempo.
aatual = aatual + aant; // área atual sob a curva.
inte = (aatual * ki)/10; // integral.
derr = (e - eant) / dt.
deri = derr * kd; // derivativo.
calc = prop + inte + deri; // calculo PID.
if(calc > 400) calc = 400; //se for maior que 400 níveis (valor máximo) calc recebe 200.
if(calc < 0) calc = 0; // se for menor que 0(valor mínino) calç recebe 0.
set_pwm2_duty(calc); // função que altera o ciclo de trabalho.
eant = e; // erro anterior recebe erro atual
aant = aatual; // area anterior recebe area atual
conta_pulso = 0; // zera a variável responsável pela veloc. do motor
```

2.5 Sistema

O sistema é dividido em duas partes: o *hardware* e o *software*.

O *hardware* é composto pelo microcontrolador, *driver* controlador do motor, motor e um *encoder*.

O software é dividido em duas lógicas: uma lógica implementada no microcontrolador e uma outra lógica implementada no microcomputador.

2.5.1 Hardware

Basicamente, o microcontrolador *16F877A* está ligado utilizando uma tensão 5*Vcc* através dos pinos 11 e 32 *(VDD)*. O aterramento está ligado nos pinos 12 e 31 *(VSS)*. O *clock* externo está ligado nos pinos 13 e 14 *(OSC1 e OSC2)*. Para gerar o clock externo, foi utilizado um cristal oscilador de 4*MHz*. O *LCD* (display de cristal líquido) está ligado nos pinos 19, 20, 21, 22, 27, 28 e 29 *(RD0, RD1, RD2, RD3, RD4, RD5, RD6)*. As teclas estão ligadas nos pinos 35, 36, 37, 38 *(RB2, RB3, RB4, RB5)*. Os pinos 25 e 26 *(RC6/TX, RC7/RX)* estão ligados no *driver (MAX232)* utilizado na comunicação via porta serial com o microcomputador. O pino 16 *(RC1/CCP2)* está ligado no *driver* para o controle da velocidade do motor. O pino 17 *(RC2/CCP1)* está ligado na saída do *encoder* para aquisição do sinal correspondente a velocidade do motor. O pino 1 *(MCLR barrado)* está ligado uma tensão 5*Vcc*. O restante dos pinos não está sendo utilizado.

O esquema elétrico da ligação do microcontrolador pode ser visualizado no Anexo 1.

O *driver* controlador do motor que liga o pino *(RC1/CCP2)* do microcontrolador com o motor, é responsável pelo chaveamento e o fornecimento suficiente da corrente para que o motor possa funcionar.

O motor a ser controlado é um motor de 2 pólos, tensão *Vcc* e ímã permanente. Na **Erro! A origem da referência não foi encontrada.**

Modelo	Tensão (V)	sem carga		máxima eficiência				Torque
		RPM	A	RPM	A	Torque cf cm	Potencia W	Travado gf cm
M710 - 3K8	12	3642	0,2	3030	0.68	257	7,8	990

O *encoder* tem o formato parecido com um motor *Vcc*. A suas características técnicas está descrito na **Erro! A origem da referência não foi encontrada.**

Modelo	Temperatura	Frequência max.	Consumo (mA)	Corrente saída (mA)	PPR	Tensão
5810-0351-0600	-20 ... +60 graus	100MHz	80	20	600	05 - 28

O encoder possui cinco pinos onde:

- ✓ pino 1: aterramento *(GND)*;
- ✓ pino 2: tensão *Vcc* que pode variar entre 5V e 28V;
- ✓ pino 3: sinal A com uma forma de onda quadrada;
- ✓ pino 4: sinal B com o sinal defasado em 90º em relação ao sinal A;
- ✓ pino 5: sinal O é o ponto zero do *encoder*.

O motor e o *encoder* estão ligados em linha através de um eixo. Esse eixo possui uma polia simulando uma carga. A ligação entre o motor e o *encoder* pode ser visualizada no Anexo 2.

2.5.2 Software

A lógica implementada no microcontrolador foi toda desenvolvida utilizando a linguagem de programação *C*, através do compilador *CCS*. Essa lógica pode ser visualizada através do Anexo 3.

Basicamente, o programa utiliza algumas *includes* disponibilizadas pelo próprio compilador como, por exemplo, para a comunicação com a entrada/saída serial e a saída para o *LCD*. O programa possui algumas funções como:

- ✓ void ccp1() – essa função é ativada através da interrupção pelo pino *RC2*. A cada interrupção, incrementa de dois na variável responsável pela velocidade atual do motor;
- ✓ void timer1() – essa função é ativada através do timer 1. A cada período de 200ms, essa função executa a lógica do controlador *PID*, zera a variável incrementada pela função *void ccp1()* e controla a saída do *PWM*. O tempo é controlado através do timer 1;

- ✓ void rs232() – essa função é ativada através da interrupção *RDA*. A função irá receber valores como velocidade desejada, constante proporcional, integral e derivativa, que atuarão diretamente no controle do motor;
- ✓ void timer0() – essa função é ativa através do timer 0. A cada período de 1s, o *display* é atualizado e também é enviada para o microcomputador a velocidade atual, velocidade desejada e os valores das constantes;
- ✓ void inicia_lcd() – essa função é responsável em inicializar o *display*;
- ✓ void ccp_int() – essa função é responsável em inicializar os módulos CCP, os timers, as interrupções e definir quais pinos utilizados são de entrada e quais pinos são de saída;
- ✓ void pwm() – essa função é responsável em configurar o sinal *PWM* gerado no microcontrolador;
- ✓ int varre_botão() – essa função verifica qual tecla foi pressionada, retornando um valor correspondente à tecla;
- ✓ void botao() – essa função faz o *debounce* da tecla pressionada e depois incrementa ou decrementa a variável responsável pela velocidade desejada;
- ✓ void main() – função principal, carrega as funções *void inicia_lcd(), void ccp_int(), void pwm() e void botao().*Essa função é responsável em manter o *software* funcionando através de um *loop* infinito.

A lógica implementada no microcomputador foi toda desenvolvida também utilizando a linguagem de programação *C*, através do *C++ Builder*. Essa lógica pode ser visualizada através do Anexo 4.

O programa possui uma tela gráfica onde pode ser visualizada a velocidade atual do motor. Pode também atuar diretamente no controle desse motor através da variável que controla a velocidade desejada, fazendo com que o motor aumente ou diminua a velocidade e alterar as constantes de controle (kp, ki, kd), responsáveis pela sincronização do sistema. Na Figura 2-7 pode ser visualizada as telas do programa.

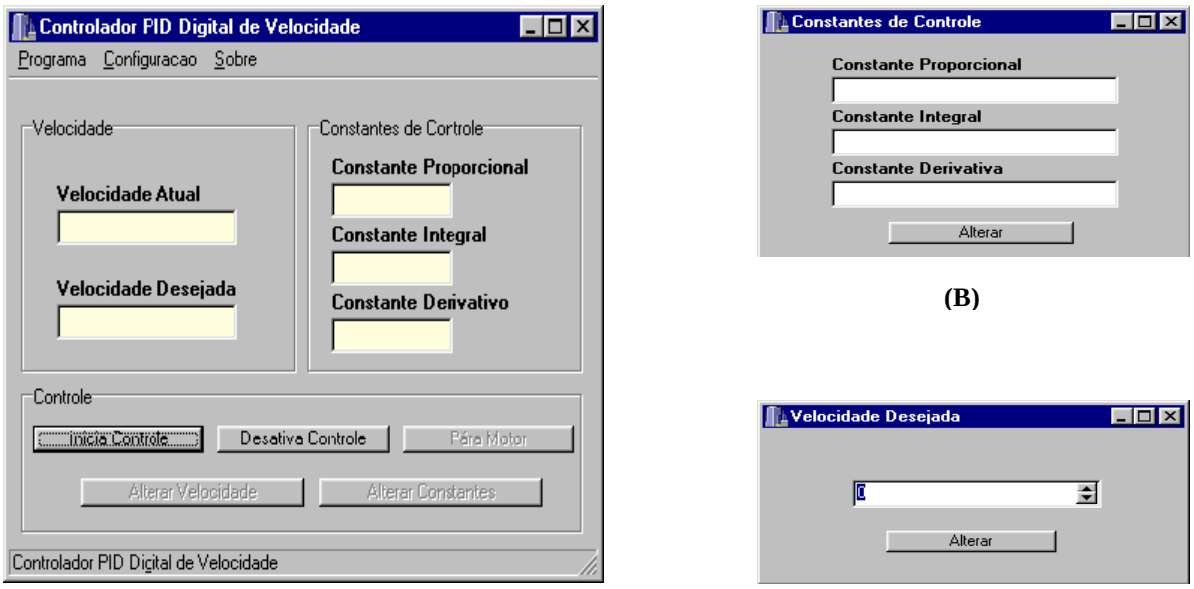


Figura 2-7 - Telas do programa desenvolvido para o microcomputador (A) tela principal, (B) tela para inserir novos valores de constantes e (C) tela para inserir a velocidade desejada.

3 RESULTADOS

Para se obter alguns resultados, primeiro foram testados alguns pontos importantes da lógica implementada no microcontrolador.

Primeiro foi configurado e testando a entrada que recebe o sinal do *encoder*. Através da função *setup_ccp1(CCP_CAPTURE_RE)*, foi configurado a entrada *(RC2/CCP1)* para que a cada quatro pulsos contados pela borda de subida do sinal fosse gerada uma interrupção incrementando de dois na variável que controla a velocidade do motor. Essa função foi testada utilizando um gerador de funções. Esse gerador foi configurado para que gerasse uma onda parecida com a onda gerada pelo o *encoder*. Na Tabela 3-1, o resultado da comparação feita com a frequência configurada no gerador de funções com a frequência capturada pelo microcontrolador.

Frequência Gerador (Hz)	Frequência LCD (Hz)	Erro (%)
100	100	0
200	200	0
300	300	0
400	400	0
500	500	0
1000	999/1000	0,1
1300	1299	0,07
1500	1498	0,13
2000	1997	0,15

Tabela 3-1 - Comparação com os valores gerados e valores medidos utilizando o microcontrolador.

Neste caso, a frequência e diretamente proporcional a velocidade, a cada 1Hz equivale a 1 rpm (rotações por minuto).

Em seguida foi testada a lógica para gerar o pulso do *PWM*. Para gerar o pulso foram utilizadas três funções: *setup_timer_2(T2_DIV_BY_1, 99, 1)*, *setup_ccp2(ccp_pwm)* e *set_pwm2_duty(0)*. A função *setup_timer_2(T2_DIV_BY_1, 99, 1)* está configurada para gerar uma frequência de 10*KHz*, a função *setup_ccp2(ccp_pwm)* está configurando a saída *CCP2* como *PWM* e a função *set_pwm2_duty(valor)* configura o ciclo de trabalho do *PWM*. A frequência gerada pelo *PWM* foi configurada utilizando a Equação 3-1 e a Equação 3-2.

$P = (1/clock) * 4 * tipo * per + 1$	**Equação 3-1**
$F = \frac{1}{P}$	**Equação 3-2**

onde *P* é o período, *clock* é a frequência usada pelo circuito, *tipo* é o valor do *prescaler* (esse valor pode ser 1, 4 ou 16), *per* é o período (esse valor varia de 0 a 255) e *F* é a frequência.

O valor máximo correspondente ao ciclo de trabalho utilizado na função *set_pwm2_duty(valor)* foi calculado utilizando a Equação 3-3.

$Valor = P / tipo * (1 / clock)$	**Equação 3-3**

onde *valor* é o valor máximo do ciclo de trabalho correspondente a 100%, *P* é o período calculado na Equação 3-1, *tipo* é o valor do *prescaler* (esse valor pode ser 1, 4 ou 16) e o *clock* é a frequência usada pelo circuito.

Depois de configurado o *PWM,* foi testado variando o valor do ciclo de trabalho através das teclas do dispositivo. O resultado foi comparado utilizando um osciloscópio. A cada incremento de 100 níveis, a forma de onda gerada tinha um ciclo de trabalho de 25% e uma frequência de 10*KHz*.

O algoritmo do *PID* implementado utilizando as constantes de controle com os seguintes valores: kp=0.1, ki=0.1 e kd=0. Através de vários experimentos chegou-se conclusão de manter a constante kd com valor zero, diminuindo portanto o ruído no sistema. A cada período de 200ms, o algoritmo é executado tendo como resultado o valor do ciclo de trabalho. O valor da velocidade desejada e o valor da velocidade atual podem ser visualizados no *display*, notando-se que as duas velocidades se mantêm à medida que a carga do motor é aumentada ou diminuída, o erro em regime permanente é nulo. Para ter certeza de que a velocidade atual é realmente verdadeira, foi utilizado um *osciloscópio* para medir o período do pulso gerado pelo *encoder* e conseqüentemente medir a sua frequência. Através desses valores, pode-se comprovar que o valor de velocidade atual visualizado no *display* está correta. Como pode ser verificado também na tabela 3.1.

A comunicação entre o microcontrolador e o microcomputador através da entrada/saída *RS232* foi testada utilizando o *software HyperTerminal* do sistema operacional *Windows* instalado no microcomputador. Através do *software HyperTerminal* foi possível visualizar a *string* gerada pelo microcontrolador contendo os valores da velocidade desejada, velocidade atual e os valores das constantes de controle (kp, ki, kd). Esses valores são separados por espaço e são atualizados no período de um segundo. A tela do *HyperTerminal* pode ser visualizada na Figura 3-1.

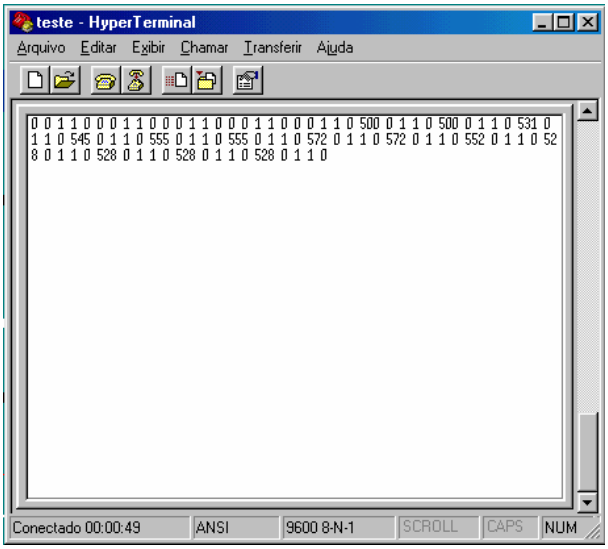


Figura 3-1 – Tela do HyperTerminal testando a comunicação entre o microcontrolador e o microcomputador.

4 DIFICULDADES ENCONTRADAS

Ao longo do projeto algumas dificuldades foram encontras, tais como: dificuldades como escrever no display, configurar o microcontrolador para medir a velocidade e corrigir a forma de onda gerada pelo encoder para a aquisição da velocidade.

As maiores dificuldades encontradas e resolvidas foram:

- ✓ o algoritmo utilizado para escrever no *display* estava com uma *flag* na configuração de inicialização invertida. Essa *flag* é a responsável em habilitar a escrita no *display*. Em vez de ser configurada com o valor zero (número 0), estava com o valor um (número 1), e quando era executada uma função para escrever no *display*, nada acontecia. O erro foi localizado comparando a lógica utilizada com outras lógicas parecidas encontras em livros e internet.
- ✓ para medir a velocidade do motor através do *encoder* ocorreram dificuldades para configurar a entrada CCP. Após vários estudos notou-se que, para utilizar a entrada CCP era preciso configurar o tipo de interrupção em função do ciclo da frequência e habilitar a interrupção global. Depois de configurado, a leitura da frequência gerada pelo encoder começou a funcionar.
- ✓ eliminar os ruídos do sinal gerado do *encoder* foi à tarefa mais difícil. Mesmo com o motor parado ou configurado para girar, por exemplo, em 10 rpm, o valor lido correspondente à velocidade atual atingia 600 rpm. Utilizando o osciloscópio para visualizar a forma de onda gerada pelo *encoder*, podia-se notar que havia um ruído junto ao sinal. Esse ruído era considerado pelo microcontrolador como se o *encoder* estivesse em movimento. Foram realizados vários testes, chegando-se à conclusão de que o chaveamento do *driver* era o responsável pelo ruído. Mudando a alimentação do motor utilizando uma fonte de alimentação externa, separando os aterramentos do motor e do microcontrolador e montando um filtro para o sinal do *encoder*, o ruído foi eliminado.

5 PRÓXIMOS TRABALHOS

O estudo e utilização de métodos para calcular os valores das constantes de controle (kp, ki, kd) para se obter um melhor sinal de acomodação no controle do motor.

Desenvolver um sistema mais customizado, onde possa ser utilizado para o controle de plantas com as mesmas características, mas com dispositivos de controle com outras configurações.

6 CONCLUSÕES

Utilizando o controle tipo *PID* e por meio de experimentos com a planta [OGATA] configurando a sintonia do controlador através das constantes de controle, pode-se comprovar que o sistema realmente funciona. Configurando a velocidade desejada do motor e aumentando a carga para que o mesmo diminua a velocidade, pode-se notar que o controlador corresponde à variação do sistema atuando de forma a aumentar a potência do motor corrigindo a velocidade. Da mesma forma ocorre quando diminui a carga, a potência do motor também diminui. Estes experimentos foram realizados de forma qualitativa.

A escolha da linguagem é um fator muito importante. Se a lógica desenvolvida precisar de um algoritmo rápido e simples, é recomendado o uso da linguagem assembly, mas se a lógica tiver funções complexas e exigir menos tempo para desenvolve-las, a linguagem C é recomendada.

Para desenvolver o algoritmo utilizando os compiladores *CCS* e *C++Builder*, foi preciso ter conhecimentos de programação, como por exemplo, declaração de variáveis, criação de funções, conhecer os recursos disponíveis nos dois compiladores e conhecer a arquitetura do microcontrador.

Embora o microcontrolador e o microcomputador possuírem arquiteturas iguais, a sua interação é possível somente quando utilizado os recursos do compilador.

Referências Bibliográficas

BRAGA, Newton C. **O que é PWM**. Revista Saber Eletrônica nº 336, pg 48-52, Janeiro 2001.

BOLTON, W. **Engenharia de Controle**. Makron Books, capítulos 1 e 10, 1995.

CAPELLI, Alexandre. **Encoder**. Revista Saber Eletrônica nº 329, pg 4-6, Junho 2000.

Características do encoder modelo 58xx-xxxx-xxxx. URL: http://www.hohner.com/Brasil/Data-Sheets/Incremental/Data-58.htm. Recuperado em 05/11/2004.

Características do motor modelo M710. URL: http://www.motron.com.br/m710.htm. Recuperado em 05/11/2004.

Data Sheet, **PIC16F877A 28/40/44 – Pin Enhanced Flash Microcontrolers**. Microchip Technology, 2003. URL: http://www.microchip.com. Recuperado em 05/11/2004.

OGATA, Katsuhiko. **Engenharia de Controle Moderno**. Prentice Hall, 4ª edição, capítulos 1, 3 e 10, 2003.

SILVA Jr., Vidal Pereira. **Microcontroladores PIC – teoria e prática**. 1998.

Bibliografia consultada

C Compiler Reference Manual, 2003. URL: http://www.ccsinfo.com. Recuperado em 05/11/2004.

DataSheet, **BC327/328**. FairChild Semiconductor. URL: http://www.ti.com. Recuperado em 05/11/2004.

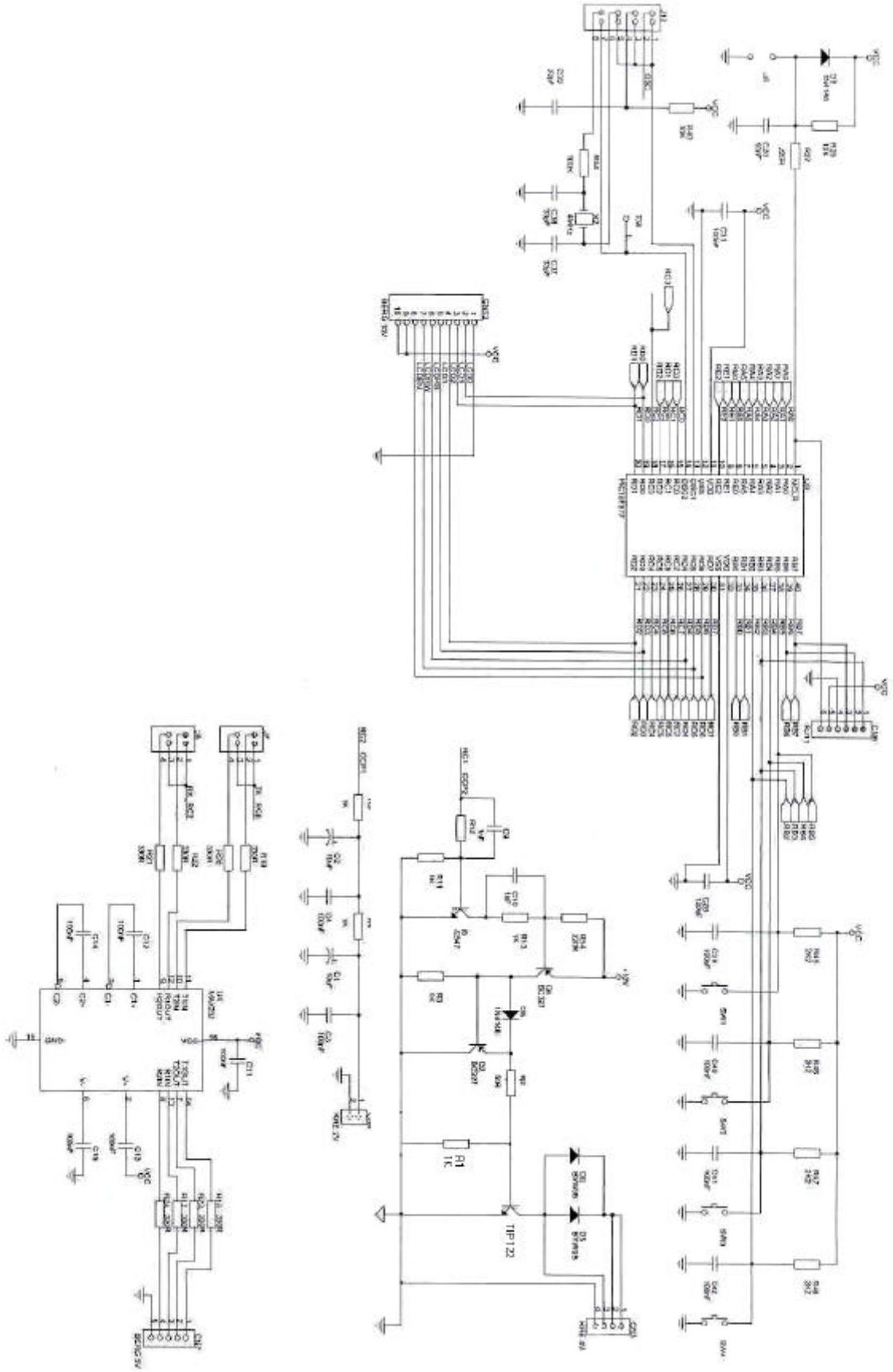
DataSheet, **BC546/547/548/549/550**. FairChild Semiconductor. URL: http://www.ti.com. Recuperado em 05/11/2004.

DataSheet, **TIP 120/121/122**. FairChild Semiconductor. URL: http://www.ti.com. Recuperado em 05/11/2004.

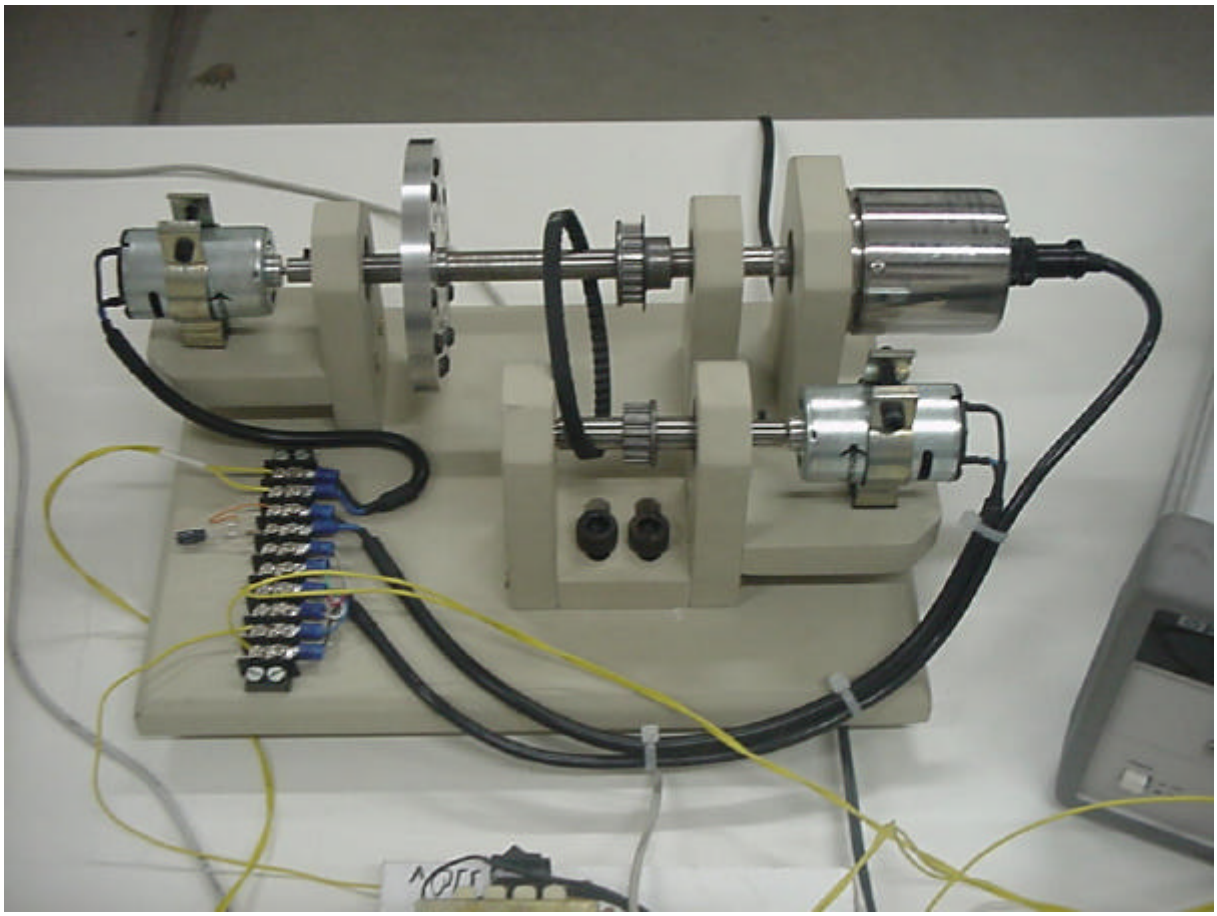
Dutton, Ken.THOMPSON, Steve.BARRACLOUGHT, Bill. **The Artof Control Engineering**. Harlow: Addison-Wesley, 1997.

PEREIRA, Fábio. **Microcontroladores PIC – Programação em C**. Érica, 2ª edição, 2003.

Anexo 1 – Esquema elétrico da ligação do microcontrolador.



Anexo 2 – Ligação entre o motor e o *encoder*.



Anexo 3 – Algoritmo de controle implementado no microcontrolador.

```
#include <16f877a.h>
#use delay(clock=4000000)
#fuses HS,NOWDT,PUT,NOLVP
#use rs232(baud=9600, xmit=PIN_c6, rcv=PIN_c7)
#include <lcd3.c>
#include <rs232.c>
#define btn1    pin_b5
#define btn2    pin_b4
#define btn3    pin_b3
#define btn4    pin_b2
#priority timer1, ccp1, timer0

signed int16 conta_pulso=0, aatual=0, aant=0, e=0, eant=0;
int kp=1, ki=1, kd=0, dt=1;
signed int16 prop=0, inte=0, deri=0, derr=0, wdes=0, watual=0, calc=0;

void ccp1();
void timer1();
void timer0();
void rs232();
void inicia_lcd();
void ccp_int();
void pwm();
void botao();
int varre_botao();

#int_ccp1
void ccp1() {
  conta_pulso = conta_pulso + 2;
}
#int_timer1
void timer1() {
  static int c;
  set_timer1(53036 - get_timer1());
  c++;
  if(c==2) {
    watual = conta_pulso;
    e = wdes - watual;
    prop = (e * kp)/10; // proporcional
    aatual = ((e + eant) / 2) * dt; // soma dos erros multiplicado pelo tempo.
    aatual = aatual + aant;
    inte = (aatual * ki)/10; // integral
    derr = (e - eant) / dt;
    deri = derr * kd; // derivativo
    calc = prop + inte + deri; // calculo PID
    if(calc > 400) calc = 400;
    if(calc < 0) calc = 0;
    set_pwm2_duty(calc);
    eant = e;
    aant = aatual;
    conta_pulso = 0;
    c = 0;
  }
}
#int_rda
void rs232() {
  char valor;
  valor = rs232_recebe();
  wdes = (int16)valor;
}
#int_timer0
void timer0() {
  static int conta;
  set_timer0(131 - get_timer0());
  conta++;
  if(conta == 125) {
    lcd_pos_xy(1,1);
    printf(lcd_escreve, "%ld c  ", calc);
    lcd_pos_xy(1,2);
    printf(lcd_escreve, "%ld wd  ", wdes);
    lcd_pos_xy(10,2);
    printf(lcd_escreve, "%ld wa  ", watual);
    printf("%ld", wdes);
    printf("%ld", watual);
```

```
      printf("%d", kp);
      printf("%d", ki);
      printf("%d", kd);
    }
}
void inicia_lcd() {
   lcd_ini();
   limpa_display();
   lcd_escreve("Start System");
   delay_ms(800);
   limpa_display();
}
void ccp_int() {
   set_tris_c(0xbd);
   set_tris_b(0xff);
   setup_ccp1(CCP_CAPTURE_DIV_4);
   setup_timer_1(T1_INTERNAL | T1_DIV_BY_8);
   setup_timer_0(RTCC_INTERNAL | RTCC_DIV_64);
   set_timer1(53036);
   set_timer0(131);
   enable_interrupts(GLOBAL);
   enable_interrupts(INT_CCP1);
   enable_interrupts(int_timer1);
   enable_interrupts(int_timer0);
   enable_interrupts(int_rda);
}
void pwm() {
   setup_timer_2(T2_DIV_BY_1, 99, 1);
   setup_ccp2(ccp_pwm);
   set_pwm2_duty(0);
}
int varre_botao() {
   if(!input(btn1)) return(1);
   if(!input(btn2)) return(2);
   if(!input(btn3)) return(3);
   if(!input(btn4)) return(4);
   return(0);
}
void botao() {
   int b;
   b = varre_botao();
   if(b) {
      delay_ms(50);
      if(varre_botao() == b) {
         if(b == 1) {
            wdes = wdes + 1;
            if(wdes >= 1000) wdes = 1000;
         }
         if(b == 2) {
            wdes = wdes - 1;
            if(wdes <= 0) wdes = 0;
         }
         if(b == 3) {
            wdes = 500;
        }
         if(b == 4) {
            wdes = 1000;
         }
      }
   }
}
void main() {
   inicia_lcd();
   ccp_int();
   pwm();
   while(true) {
      botao();
  }
}
```

Anexo 4 – Algoritmo do supervisório implementado no microcomputador.

```
//---------------------------------------------------------------------------
#include <vcl.h>
#include <string.h>
#pragma hdrstop

#include "Unit1.h"
#include "Unit2.h"
#include "Unit3.h"
#include "Unit4.h"
#include <string.h>
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "VaClasses"
#pragma link "VaComm"
#pragma link "CSPIN"
#pragma resource "*.dfm"
TPrincipal *Principal;

AnsiString in, out, wdes, watual, kp, ki, kd;
int tam, tam2, pos1, pos2, pos3, pos4, i;
//---------------------------------------------------------------------------
void __fastcall TPrincipal::Sair1Click(TObject *Sender)
{
      VaComm1->Close();
      Close();
}
//---------------------------------------------------------------------------
void __fastcall TPrincipal::Projeto1Click(TObject *Sender)
{
      ShowMessage(AnsiString("Projeto Desenvolvido como Trabalho de Conclusão de Curso\nCurso: Engenharia de Computação\nOrientador: Prof. Ms. Paulo Eduardo Silveira\nOrientado: Cleber Zorzi\nVersão: 1.0v\nCopyright 2004"));
}
//---------------------------------------------------------------------------
void __fastcall TPrincipal::Programa2Click(TObject *Sender)
{
      ShowMessage(AnsiString("Sistema desenvolvido para o monitoramento e controle da\nvelocidade de um motor de corrente contínua."));
}
//---------------------------------------------------------------------------
void __fastcall TPrincipal::BtnIniciaClick(TObject *Sender)
{
      VaComm1->Active();
      VaComm1->Open();
      BtnPara->Enabled = true;
      BtnWdes->Enabled = true;
      BtnConst->Enabled = true;
}
//---------------------------------------------------------------------------
void __fastcall TPrincipal::Comunicao1Click(TObject *Sender)
{
      Comunicacao->Show();
}
//---------------------------------------------------------------------------
void __fastcall TPrincipal::VaComm1RxChar(TObject *Sender, int Count)
{
     in = VaComm1->ReadText();
     tam2 = StrLen(in.c_str());
     while ((tam = in.Pos(' ')) > 0 ) {
            in[tam] = '0';
            i++;
            if(i==1) pos1=tam;
            if(i==2) pos2=tam;
            if(i==3) pos3=tam;
            if(i==4) pos4=tam;
     }
     wdes = in.SubString(1,(pos1-1));
     watual = in.SubString((pos1+1),(pos2-1)-pos1);
     kp = in.SubString((pos2+1),(pos3-1)-pos2);
     ki = in.SubString((pos3+1),(pos4-1)-pos3);
     kd = in.SubString((pos4+1),(tam2)-pos4);
     EdWdes->Text = wdes;
     EdVeloc->Text = watual;
     EdKp->Text = kp;
     EdKi->Text = ki;
```

```
      EdKd->Text = kd;
}
//---------------------------------------------------------------------------
void __fastcall TPrincipal::BtnDesativaClick(TObject *Sender)
{
      VaComm1->Close();
      BtnPara->Enabled = false;
      BtnWdes->Enabled = false;
      BtnConst->Enabled = false;

}
//---------------------------------------------------------------------------
void __fastcall TPrincipal::BtnConstClick(TObject *Sender)
{
      Constante->Show();
}
//---------------------------------------------------------------------------
void __fastcall TPrincipal::BtnWdesClick(TObject *Sender)
{
      Veloci->Show();
}
//---------------------------------------------------------------------------
```