SOCIEDADE EDUCACIONAL DE SANTA CATARINA – SOCIESC
INSTITUTO SUPERIOR TUPY – IST

MATHEUS NORBERTO HAGEMANN
RAFAEL SOARES DE SOUZA

MOTOR DE PASSO PROGRAMÁVEL COM DRIVER E ENCOLDER INTEGRADOS

Joinville-SC
2010

MATHEUS NORBERTO HAGEMANN

RAFAEL SOARES DE SOUZA

MOTOR DE PASSO PROGRAMÁVEL COM DRIVER E ENCOLDER INTEGRADOS

Trabalho de Conclusão de Curso submetido ao Instituto Superior Tupy como requisito parcial para obtenção do título de Engenheiro de Controle e Automação, sob orientação do

PROF. ROBERTO DO AMARAL SALES

Joinville-SC

2010

MATHEUS NORBERTO HAGEMANN
RAFAEL SOARES DE SOUZA

MOTOR DE PASSO PROGRAMÁVEL COM DRIVER E ENCOLDER INTEGRADOS

Este trabalho foi julgado e aprovado em sua forma final, sendo assinado pelos professores da banca examinadora.

Prof. Roberto do Amaral Sales

Prof. Albarello

Prof. Alexandre Werner Arins

Joinville, 14 de dezembro de 2010.

Dedicamos este trabalho a Deus quenos sustentou para alcançarmos esse objetivo, e aos nossos familiares, e amigos.

AGRADECIMENTOS

Agradecemos a Deus por ter dado a força necessária para atingirmos esse objetivo. Aos nossos familiares, pelo carinho e entendimento pelos momentos em que não estivemos presentes e aos nossos amigos que nos ajudaram e nos incentivaram nessa caminhada

RESUMO

A finalidade deste projeto é desenvolver um motor de passo programável via porta serial (RS232 “DB9”) do computador, que comportará um driver junto a um encoder que irão interagir de forma a comandar o deslocamento do motor garantindo o seu posicionamento. Visa-se elaborá-lo de forma que seu dimensional possua pouco volume já que seus componentes serão acoplá-los num mesmo conjunto. Estuda-se a possibilidade de redução de custo na implementação desse componente na automação industrial em geral, e esperasse uma maior facilidade na sua instalação e implementação. O circuito eletrônico será construído focando utilizar um microcontrolador que terá a função lógica do implemento.

Palavras-chave: Motor de Passo. Programável. Driver. Encoder. Microcontrolador

ABSTRACT

The purpose of this project is to develop a programmable stepper motor via serial port (RS232 "DB9") computer, which include a driver with an encoder that will interact to control the displacement of the engine ensuring your position. The aim is to elaborate it so that its volume has little dimensional components since MT will be engaging them in the same set. It studies the possibility of reducing cost in implementing this component in industrial automation in general and expected a greater ease in installation and implementation. The electronic circuit will be built focusing using a microcontroller that has the logical function of the implement.

Keywords: Stepper motor. Driver. Encoder. Microcontroller.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

O motor de passo é encontrado em equipamentos que necessitam de movimentos precisos. Sua utilização na automação industrial é bastante ampla, sendo possível controlar ângulo de rotação, velocidade, posição e sincronismo.

O controle de velocidade irá variar com freqüência que os pulsos são aplicados na entrada. O ângulo de rotação está relacionado com o número de pulsos que é aplicado nas suas bobinas, e quanto a sua direção, está diretamente ligada a seqüência de aplicação dos pulsos. Observa-se que o motor de passo para a indústria abrange uma grande área, principalmente em aplicações mais simples, contudo para essas aplicações sabe-se que há a necessidade de um painel elétrico para incorporar o controle do processo.

Assim sugere-se um produto que possa tanto ser programado ou comandado por um computador, e que de forma simples permita a elaboração de uma solução precisa em controle de ângulo de rotação, velocidade, posição e sincronismo. Focando em baixo custo, precisão (encolder), fácil implementação e programação, tem-se como objetivo desenvolver um driver que interagindo a um encolder serão acoplados a um motor de passo.

1.1 TEMA E DELIMITAÇÃO DO TEMA

Conforme o tema desse trabalho sugere: **“Motor de passo programável com driver e encolder integrados”,** esse trabalho será focado no estudo para o desenvolvimento de uma solução para a indústria, em motores de passo.

1.2 PROBLEMATIZAÇÃO

O motor de passo atualmente utilizado na indústria necessita de um driver que ocupa um bom espaço físico em um painel elétrico de controle, o mesmo não obtém uma rápida instalação, e muitas vezes também requer um controlador “CLP” externo para seu controle que por sua vez acaba necessitando de mão de obra especializada, assim uma aplicação simples acaba somando um custo desnecessário.

Criou-se um cenário para a obtenção de uma melhor comparação entre o que atualmente é oferecido pela automação industrial” VS., e o produto sugerido

neste projeto conforme descrito a seguir: Um equipamento para verificar o comprimento ”X” de barras de ferro através de deslocamento transversal, o mesmo deve acusar “Peças Boas” e “Peças de Rejeito”. A parte mecânica consiste em um batente sobre um fuso que será girado através de um motor de passo, que por sua vez através de lógica de controle de rotação irá monitorar o deslocamento no eixo do fuso). Quantificadas a automação para esse equipamento o resultado será de um painel elétrico completo, sem contar mão de obra, que somaria: CLP + DRIVER + MOTOR DE PASSO + FONTE + UTILITÁRIOS: 3.900,00 R$. Sendo que nessa aplicação descrita, não está prevista a implementação de um encolder que seria necessário para garantir o posicionamento para o equipamento.

No projeto proposto haverá encolder para monitoramento do posicionamento e esse poderá ser programado ou controlado por um computador com o simples requisito de comportar uma porta serial. Quantificando o seu custo para a mesma aplicação: MOTOR DE PASSO + DRIVER + ENCOLDER = 400 R$, sendo que o driver será construído acomodando um microcontrolador PIC 16F887A, que terá a função lógica do implemento.

1.3 OBJETIVO GERAL

A idéia principal para esse estudo é a de oferecer um produto de baixo custo que seja uma solução para a automação industrial e que obtenha a mesma qualidade ou uma qualidade superior aos produtos de mesma linha encontrados no mercado atual.

Será desenvolvido num motor de passo, um controle com driver programável e encolder de forma a acoplá-los, buscando obter o menor volume o possível para esse conjunto, com o objetivo de facilitar o processo de instalação desse produto em máquinas industriais. Desenvolver-se-á um menu de programação, da forma mais simples e objetiva possível para facilitar a sua implementação.

Haverá um dispositivo mecânico para simular o funcionamento de controle do motor de passo, monitorado através do software de programação. A descrição de como se deseja obter esse dispositivo, segue conforme o cenário já descrito anteriormente: (Um equipamento para verificar o comprimento ”X”, de barras de ferro através de deslocamento transversal, o mesmo deve acusar “Peças Boas” e “Peças de Rejeito”, a parte mecânica consiste em um batente sobre um fuso que será girado através de um motor de passo, que por sua vez através de lógica irá monitorar o

deslocamento no eixo do fuso).

1.4 OBJETIVO ESPECÍFICO

A) Desenvolver uma placa com microcontrolador “PIC16F887”, com a função de controlar o motor de passo, sendo esse o driver que irá abranger a rotina pré-programada para o motor de passo.

B) Desenvolver junto a placa driver, uma porta de comunicação RS-232 para programar e executar o programa no PIC.

C) Acoplar um encoder junto ao eixo do motor de passo para garantir a sua rotação. Assim, será formado um circuito em malha fecha, que ao ser implementado a lógica do driver irá informar caso haja escorregamento no motor de passo.

D) Desenvolver e construir um acoplamento de nylon que irá integrar todos os componentes do conjunto, ou seja; motor de passo + placa driver + encolder.

E) Estudar a possibilidade da construção de um encolder com o objetivo de maximizar o volume do conjunto, sendo que, quanto menor o volume obtido melhor será sua aplicabilidade.

F) Construir um dispositivo mecânico conforme descrito no objetivo geral, o mesmo será construído com base em aço carbono 1020, fuso, e guia linar com patins de esfera.

1.5 HIPÓTESES

A principio, a maior dificuldade está na implementação do encoder, pelo fato do motor de passo vibrar de forma a poder interferir na leitura do encoder, sendo assim se o motor de passo interferir na leitura do encoder será inapropriado o uso de um circuito em malha fechada, ou seja, o uso do encolder será descartado. Também não é garantido nem fato de que as placas a serem construídas fiquem no tamanho desejado para a integração no motor de passo, assim poderá ocorrer do conjunto obter um tamanho desproporcional (tamanho esperado: largura, base e altura com no máximo os mesmos dimensionais do motor de passo a ser usado). Contudo, por ser esse projeto um protótipo não é considerado como primordial o seu dimensional.

1.6 JUSTIFICATIVA

A iniciativa para o estudo e implementação desse projeto, se designa à inovação em motores de passo, aplicando soluções de baixo custo com facilidade de adaptação em máquinas para a indústria, eliminando o painel de controle do mesmo.

2 FUNDAMENTAÇÃO TEÓRICA

A partir desse capítulo se explanará as principais teorias de embasamento para desenvolver o trabalho. Focam-se nos conceitos e aplicabilidade dos componentes utilizados com o intuito de facilitar o entendimento do projeto.

2.1 MOTOR DE PASSO

Conforme Messias (2007), motores de passos são dispositivos mecânicos eletro-magnético que podem ser controlados digitalmente através de um hardware específico e/ou através de softwares. O mesmo trata-se de um transdutor que converte energia elétrica em movimento controlado através de pulsos, o que possibilita o deslocamento por passo, onde o passo é o menor deslocamento angular.

2.1.1 Tipos de motores de passo

De acordo com Souza (2007) quanto à estrutura interna, existem basicamente 3 tipos de motores de passo:

- Relutância variável;
- Imã permanente;
- Híbridos.

Relutância variável – consiste de um rotor de aço doce multipolar com os enrolamentos no estator. Quando as bobinas do estator são energizadas, os dentes do rotor tentam se alinhar com os pólos do estator. Ao alternar as bobinas que são alimentadas no estator, o campo do estator muda e, assim, o rotor move-se para uma nova posição. A figura 1 mostra a estrutura interna de um motor de passo relutância variável. (SOUZA, 2007)

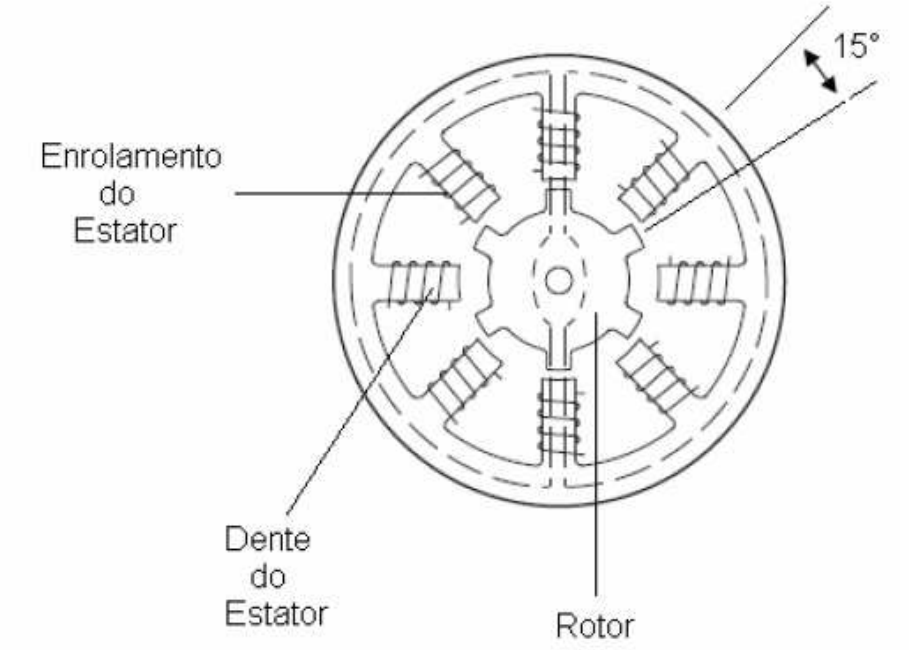

Figura 1 - Estrutura interna de um motor de passo relutância variável

Fonte: Tecnical English – Spanhish Vocabulary, (2010)

Imã permanente – Possui um rotor constituído de um matérial permanentemente magnetizado radialmente. Seu funcionamento é baseado na reação entre o campo magnético fixo do rotor e o campo gerado no enrolamento do estator. (SOUZA, 2007)

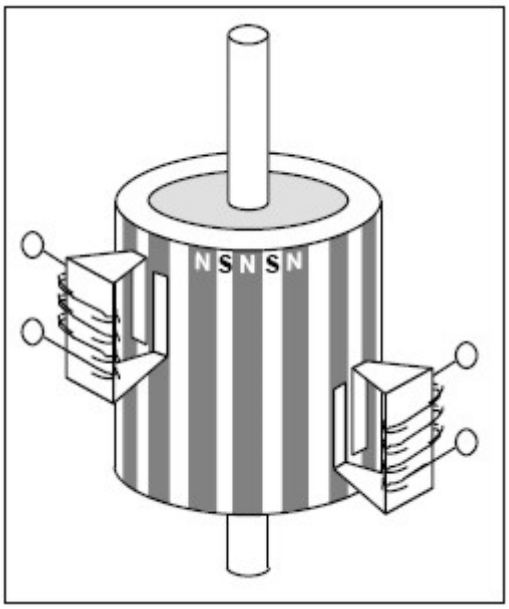

Figura 2 - Rotor do motor de passo imã permanente

Fonte: Tecnical English – Spanhish Vocabulary, (2010)

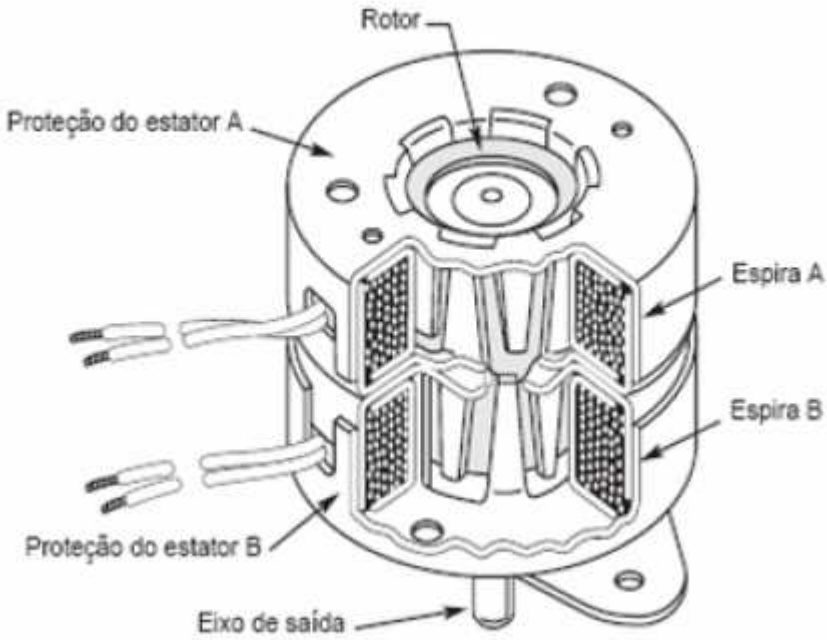

Figura 3 - Vista de corte de um motor de passo imã permanente

Fonte: LYRA (2010, p.3)

Híbridos - É o tipo mais usado em aplicações industriais. Ele combina os princípios operacionais dos outros dois tipos de motores apresentados anteriormente. Sendo um motor de tamanho reduzido apresentando alto torque e pequenos passos. (SOUZA, 2007)

> A estrutura do núcleo do estator de um motor híbridos é essencialmente a mesma que a de um motor relutância variável, a diferença é que no motor relutância variável somente uma das bobinas de uma fase é enrolada em um pólo, enquanto um típico motor hibrido possui as bobinas de duas fases diferentes no mesmo pólo. Estas duas bobinas no mesmo pólo são enroladas em uma configuração conhecida como conexão bifilar. (KENJO, 1986)

A estrutura do motor consiste em duas peças de pólo multi-dentada. Entre estas peças de pólo há um imã permanente magnetizado em paralelo com o eixo do rotor, tornando uma ponta um pólo norte e na outra um pólo sul. A figura 4 mostra a seção transversal de um motor híbrido. (SOUZA, 2007)

> O motor híbrido combina as melhores características dos motores de ímã permanente e motor de relutância variável. O rotor é multi-dentado como no motor de relutância variável e contem um ímã permanente ao redor do seu

> eixo. Os dentes do rotor provém um melhor caminho que ajuda a guiar o fluxo magnético para locais preferidos no GAP de ar. (SOMESSARI, 2010, p.20).

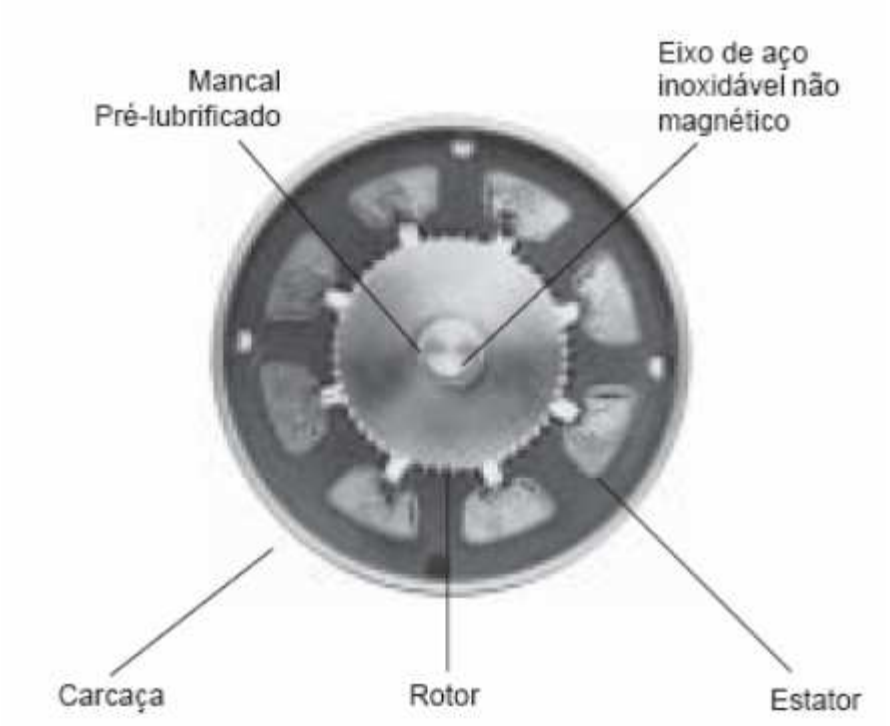

Figura 4 - Secção transversal de um motor híbrido

Fonte: (SOUZA, 2007)

2.1.2 Tipos de alimentação dos enrolamentos de um motor de passo

Alimentação unipolar – De acordo com Marco Antonio de Souza (2007) Possui uma derivação central em cada um dos enrolamentos, onde o número de fases é o dobro do número de bobinas. A figura 5 abaixo representa um motor de passo unipolar de 4 fases. (SOUZA, 2007)

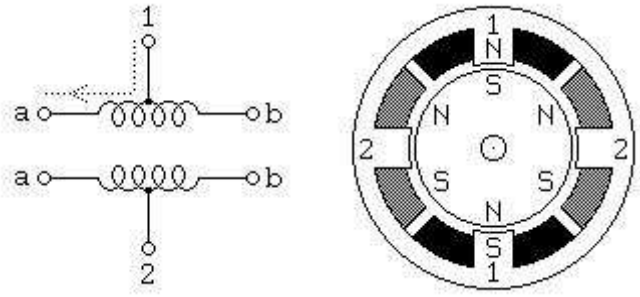

Figura 5 - Motor de passo unipolar

Fonte: SOUZA (2007)

Alimentação bipolar – Possui um único enrolamento por fase, que devem ser alimentadas em ambas as direções para permitir o avanço de um passo. Para isso deve-se inverter a polaridade da tensão na bobina e de forma seqüencial. A figura 6 abaixo representa um motor de passo bipolar.

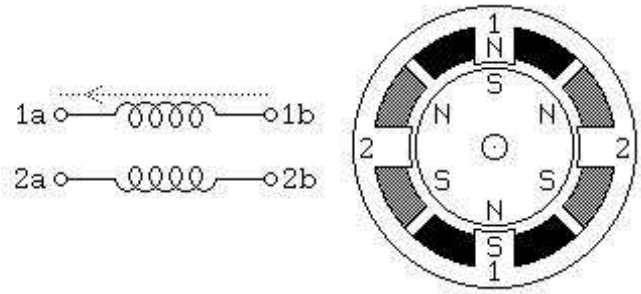

Figura 6 - Motor de passo bipolar

Fonte: SOUZA (2007)

2.1.3 Modos de operação do motor de passo

> A energizaçãoo de uma e somente uma bobina de cada vez produz um pequeno desloca-mento no rotor. Este deslocamento ocorre simplesmente pelo fato de o rotor ser magneticamente ativo e a energizaçãoo das bobinas criar um campo magnético intenso que atua no sentido de se alinhar com os dentes do rotor. Assim, polarizando de forma adequada as bobinas, podemos movimentar o rotor entre as bobinas (meio passo ou “half-step”) ou alinhadas com as mesmas (passo completo ou “full-step”). (BRITES & SANTOS, 2008, p.5)

Existem três modos de operação para um motor de passo:

Passo completo 1 (Full-step 1):

Segundo Brites & Santos (2008), somente uma bobina é energizada a cada passo, onde o consumo de energia e o torque são baixos com uma maior velocidade. A figura 7 e 8 mostram respectivamente a energização de uma bobina produzindo um pequeno deslocamento no rotor de motores unipolar e bipolar de passo completo 1 (Full-step).

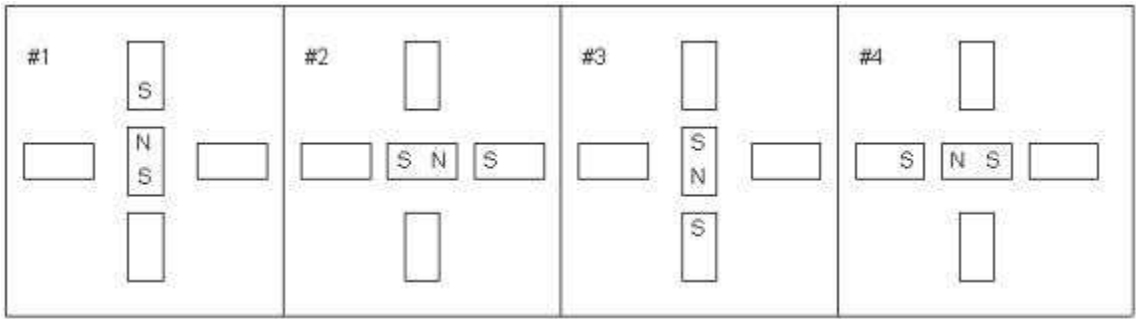

Figura 7 - Motor Unipolar de passo inteiro

Fonte: Brites & Santos (2008)

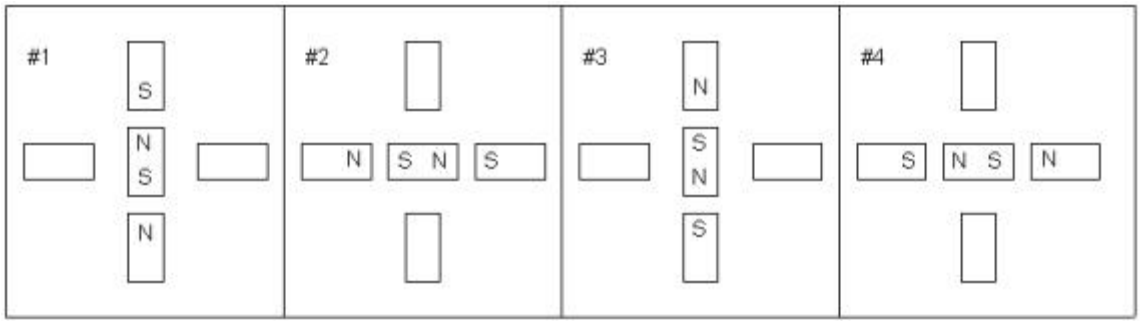

Figura 8 - Motor bipolar de passo inteiro

Fonte: Brites & Santos (2008)

Passo completo 2 (Full-step 2)

Duas bobinas são energizadas a cada passo onde o torque e o consumo de energia são maiores do que o passo completo 1 porem velocidade é a mesma. A figura 9 e 10 mostram respectivamente a energização de duas bobina produzindo um pequeno deslocamento no rotor de motores unipolar e bipolar de passo completo 2 (Full-step). (MESSIAS, 2006)

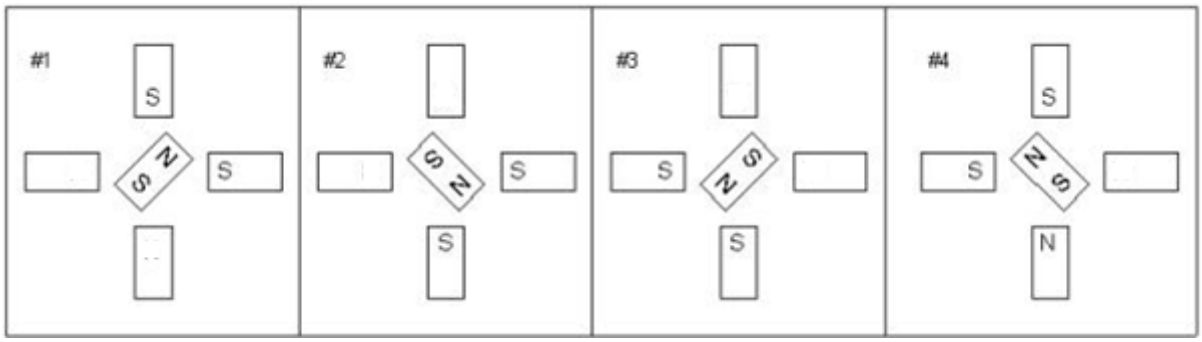

Figura 9 - Motor unipolar de passo inteiro

Fonte: www2.eletronica.org/artigos/outros/estudo-do-motor-de-passo-e-seu-controle-digital

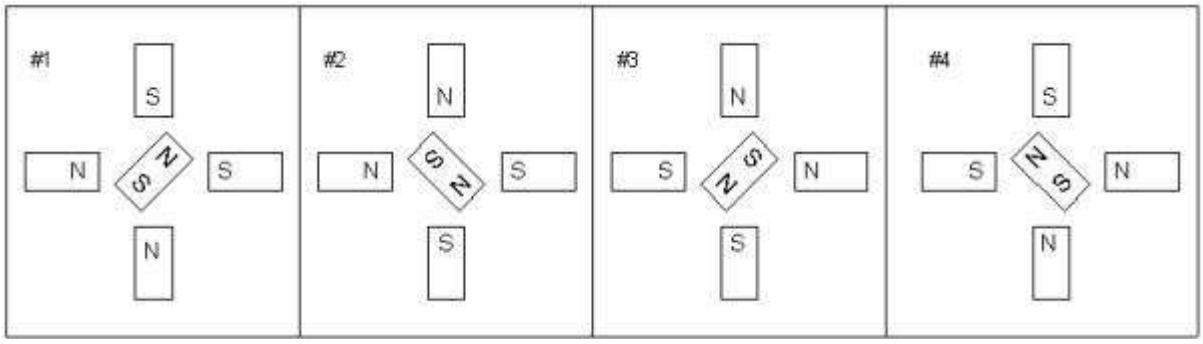

Figura 10 - Motor bipolar de passo inteiro

Fonte: www2.eletronica.org/artigos/outros/estudo-do-motor-de-passo-e-seu-controle-digital

Meio passo (Half-step)

A combinação do passo completo 1 e do passo completo 2 que gera um efeito de meio passo de revolução. Este modo consome mais energia e é mais preciso, porem a sua velocidade é menor que do Passo completo 1 e 2, e o torque é próximo ao do Passo completo 2.

A figura 11 e 12 mostram respectivamente a seqüência de energização das bobinas produzindo um pequeno deslocamento no rotor de motores unipolar e bipolar de meio passo (Half-step). (MESSIAS, 2006)

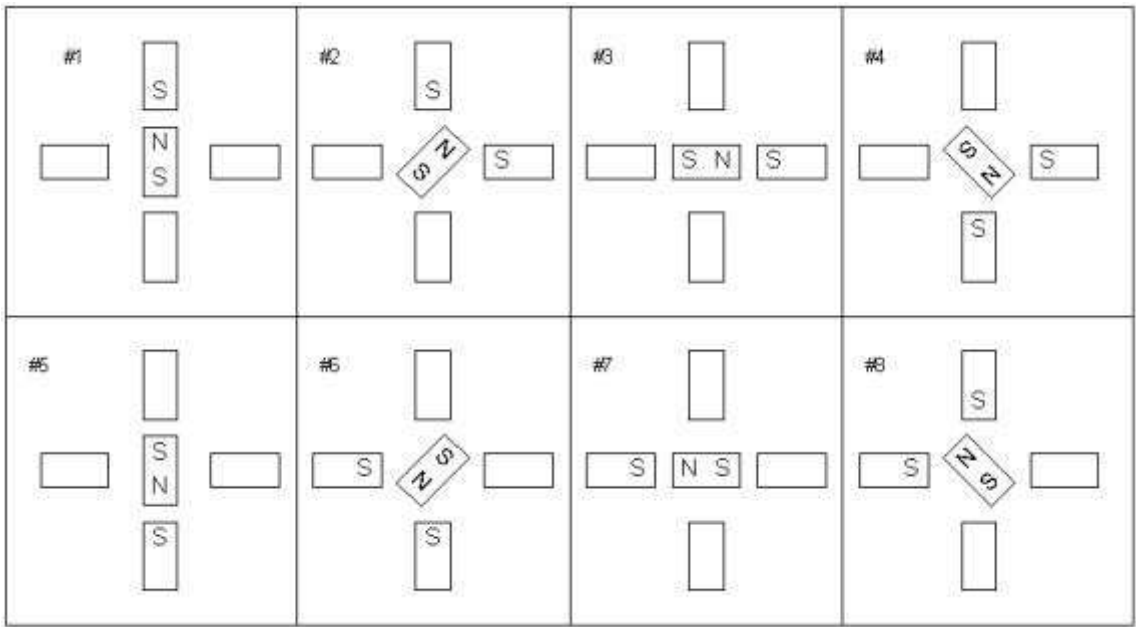

Figura 11 - Motor unipolar de meio passo

Fonte: www2.eletronica.org/artigos/outros/estudo-do-motor-de-passo-e-seu-controle-digital

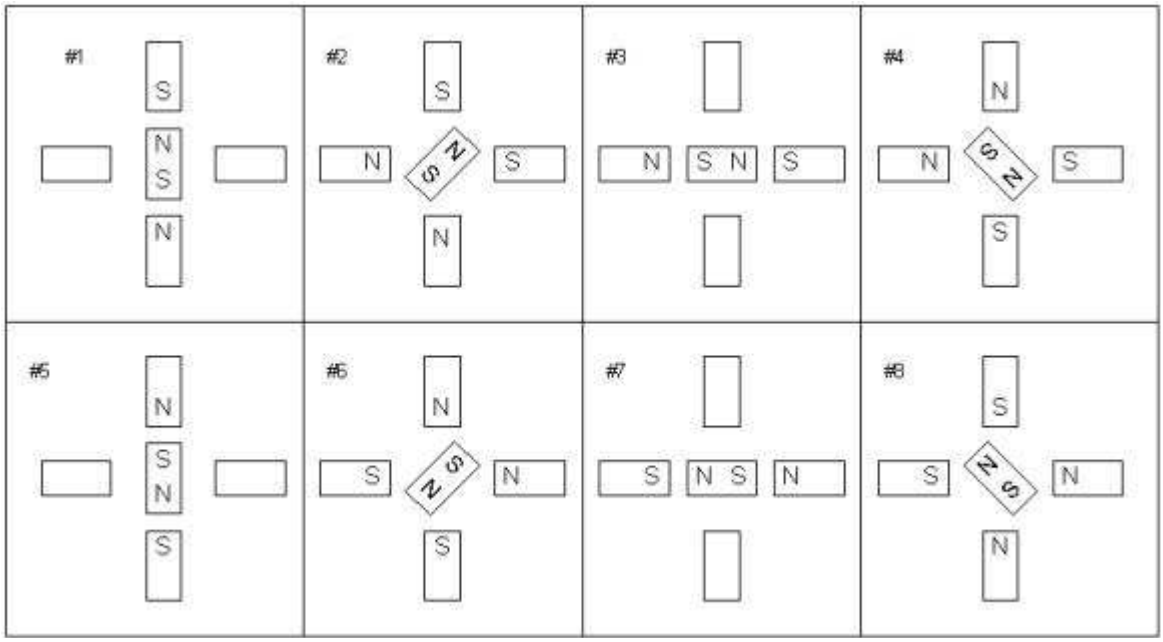

Figura 12 - Motor Bipolar de meio passo

Fonte: www2.eletronica.org/artigos/outros/estudo-do-motor-de-passo-e-seu-controle-digital

2.1.4 Modos de controle do motor de passo

Para o controle desses motores de passo segue abaixo as figuras com as seqüências de acionamento:

Número do Passo	Bobina 3	Bobina 2	Bobina 1	Bobina 0	Decimal
1	1	0	0	0	8
2	0	1	0	0	4
3	0	0	1	0	2
4	0	0	0	1	1

Figura 13 - Modo de operação passo completo 1 (Full-step)

Fonte: (MESSIAS, 2008)

Número do Passo	Bobina 3	Bobina 2	Bobina 1	Bobina 0	Decimal
1	1	1	0	0	12
2	0	1	1	0	6
3	0	0	1	1	3
4	1	0	0	1	9

Figura 14 - Modo de operação passo completo 2 (Full-step)

Fonte: (MESSIAS, 2008)

Número do Passo	Bobina 3	Bobina 2	Bobina 1	Bobina 0	Decimal
1	1	0	0	0	8
2	1	1	0	0	12
3	0	1	0	0	4
4	0	1	1	0	6
5	0	0	1	0	2
6	0	0	1	1	3
7	0	0	0	1	1
8	1	0	0	1	9

Figura 15 - Modo de operação meio passo (Half-step)

Fonte: Fonte: (MESSIAS, 2008)

2.1.5 Velocidade, posicionamento e direção

A velocidade do motor de passo conforme Messias (2008) é controlada quando se envia uma seqüência de pulsos digitais num determinado intervalo. Esse

intervalo não deve ser menor que 10ms entre cada passo, pois o motor perderá o torque e não irá girar. O tamanho do ângulo rotacional é diretamente relacionado com o número de passos aplicados.

A direção do motor de passo é controlada pela seqüência da alimentação da suas bobinas, que por sua vez dão movimento para o motor. As figuras 16 e 17 mostram as seqüências que devem são usadas para movimentar o motor de passo no sentido horário e no sentido anti horário.

Número do Passo	Bobina 3	Bobina 2	Bobina 1	Bobina 0	Decimal
1	1	0	0	0	8
2	0	1	0	0	4
3	0	0	1	0	2
4	0	0	0	1	1

Figura 16 - Passo completo 1 (direita)

Fonte: Fonte: (MESSIAS, 2008)

Número do Passo	Bobina 3	Bobina 2	Bobina 1	Bobina 0	Decimal
1	0	0	0	1	1
2	0	0	1	0	2
3	0	1	0	0	4
4	1	0	0	0	8

Figura 17 - Passo completo 2 (esquerda)

Fonte: Fonte: (MESSIAS, 2008)

2.2 MICROCONTROLADOR (PIC 16F628A)

Expressa Pereira (2007) que microcontrolador é um circuito integrado que faz uso de todos os componentes necessários para o controle de um processo. Os microcontroladores possuem uma Unidade Lógica Aritmética (ULA) que faz o processamento de toda parte de operações lógicas e matemáticas, tudo isso compactado em um único CI(Circuito Integrado).

A figura 18 exemplifica o diagrama em blocos interno de um microcontrolador.

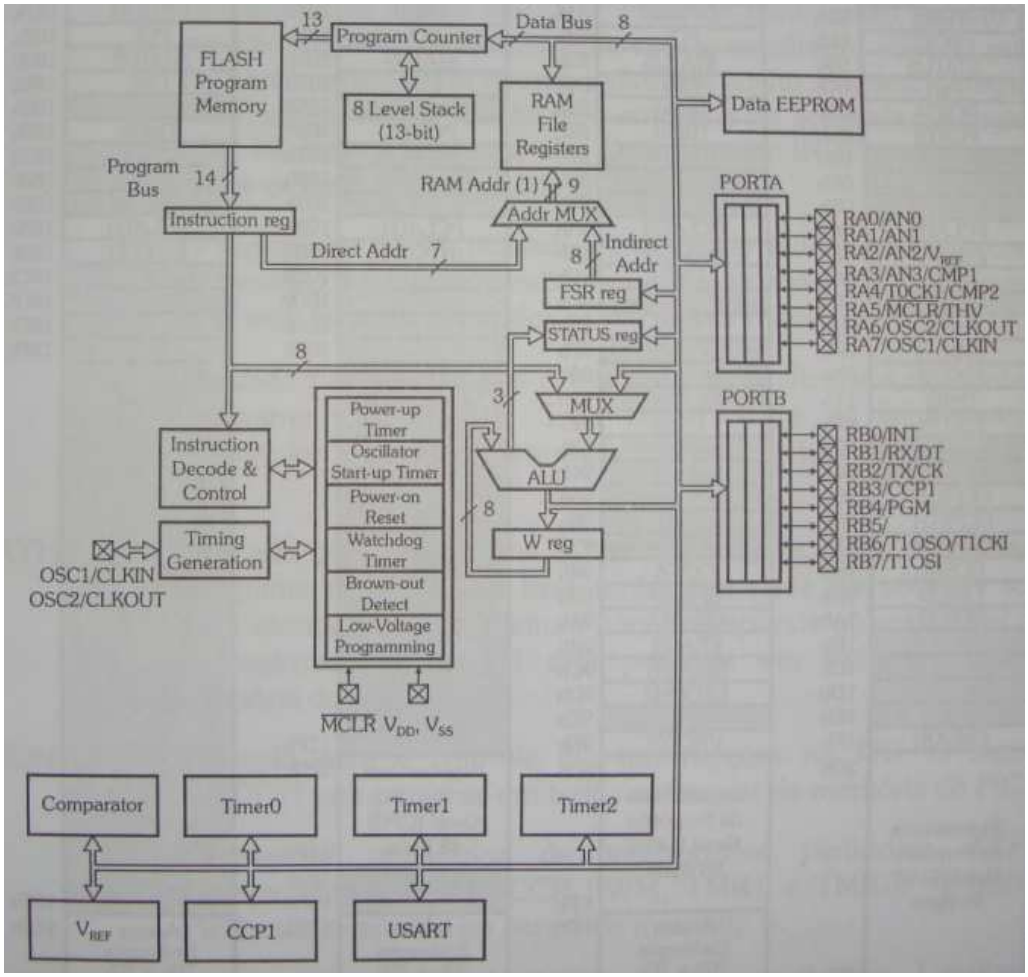

Figura 18 - Estrutura interna do PIC 16F628

Fonte: Pereira (2007, p.101)

O microcontrolador escolhido foi o PIC 16F627 do fabricante Microchip, por ser um controlador intermediário “Mid-rande” que supre técnicamente as necessidades desse projeto.

Conforme Pereira (2007, p.35):

> Os microcontroladores PIC são uma família de dispositivos fabricados pela MIcrochop. Utilizando uma arquitetura RISC, com freqüências de clock de até 40Mhz, até 2048K Word de memória de programa e até 3968 bytes de memória RAM. Além disso, podem ser encontrados com diversos periféricos internos, como: até quatro temporizadores/contadores, memória EEPROM interna, gerador/comparador/amostradpr PWM, conversores A/D de até 12 bits, interface de barramento CAN, 12C, SPI, entre outros.

Segue as suas principais características, segundo Pereira (2007, p.99):

- 1024x14bits de memória FLASH (2048x14bits para o 16F628)
- 224x8bits de memória SRAM disponíveis para o usuário

- 128x8bits de memória EEPROM interna
- Pilha com 8 níveis
- 15pinos de I/O (entrada ou saída)
- (1x) Pino de entrada
- (1x) Timer / contador de 8 bits
- (1x) Timer / contador de 16 bits
- (1x) Timer de 8 bits
- (1x) Canal PWM com captura e amostragem (CCP)
- (1x) Canal de comunicação USART serial
- (2x) Comparadores analógicos com referência interna programável de tensão
- (1x) Timer watchdog
- (10x) Fontes de interrupção independentes
- Capacitores de corrente de 25mA por pino de I/O
- 35 instruções
- Freqüência deoperação desde DC (0 Hz) até 20 Mhz
- Oscilador 4Mhz/37Khz interno
- Tensão de operação entre 3.0 a 5.5V
- Compatível pino a pino com 16F84 e outros PICs de 18 pinos

De acordo Pereira (2007) existem fundamentalmente três família de PICs diferenciais pelo tamanho da palavra da memória de programa 12, 14 e 16 Bits. Todos estes dispositivos possuem um barramento interno de dados de oito bits.

2.2.1 Características físicas e pinagem do PIC 16F628A

Segue estrutura da pinagem do microcontrolador PIC 16F628A conforme figura 19, extraído do Livro de David J. de Souza (2007).

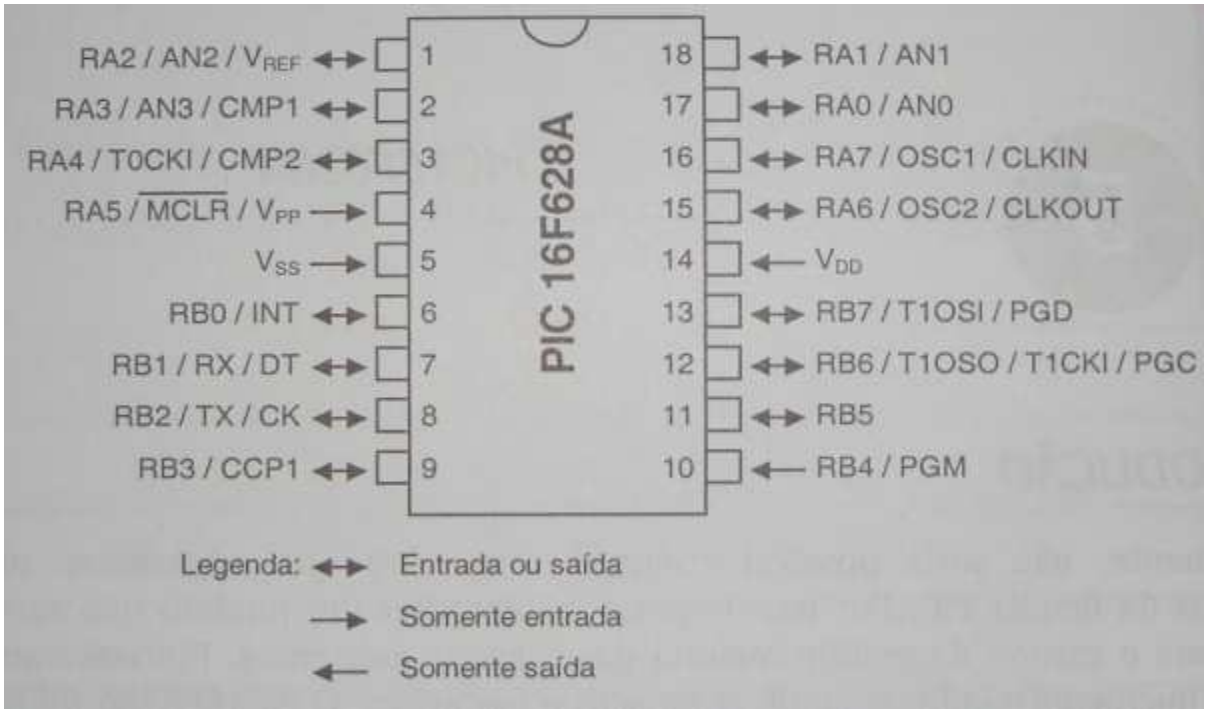

Figura 19 - Pinagem PIC 16F628A

Fonte: Souza (2007, p.36)

O Quadro a seguir (figura 20) descreve os detalhes da nomenclatura de cada pino:

Número de Pino	Função	Tipo de Entrada	Tipo de Saída	Descrição
17	RA0	ST	CMOS	I/O digital biodirecional
	AN0	AN	-	Entrada analógica para os comparadores
18	RA1	ST	CMOS	I/O digital biodirecional
	AN1	AN	-	Entrada analógica para os comparadores
1	RA2	ST	CMOS	I/O digital biodirecional
	AN2	AN	-	Entrada analógica para os comparadores
	Vref	-	AN	Saída da tensão de referência programável
2	RA3	ST	CMOS	I/O digital biodirecional
	AN3	AN	-	Entrada analógica para os comparadores
	CMP1	-	CMOS	Saída do comparador 1
3	RA4	ST	OD	I/O digital biodirecional
	T0CKI	ST	-	Entrada externa do contador TMR0
	CMP2	-	OD	Saída do comparador 2
4	RA5	ST	-	Entrada digital
	MCLR	ST	-	Master Cler (reset) externa. O PIC só funciona quando este pino encontra-se em nível alto
	Vpp		-	Entrada para tensão de programação (13V)

15	RA6	ST	CMOS	I/O digital bioderecional
	OSC2	-	XTAL	Saída para cristal externo
	CLKOUT	-	CMOS	Saída com onda quadrada em 1/4 da frequência imposta em OSC1 quando em modo RC. Essa frequência equivale aos ciclos de máquina internos
16	RA7	ST	CMOS	I/O digital bidirecional
	OSC1	XTAL	-	Entrada para cristal externo
	CLKIN	ST	-	Entrada para osciladores externos (híbridos ou RC)
6	RB0	TTL	CMOS	I/O digital bidirecional com pull-up interno
	INT	ST	-	Entrada para interrupção externa
7	RB1	TTL	CMOS	I/O digital bidirecional com pull-up interno
	RX	ST	-	Recepção para comunicação USART assíncrona
	DT	ST	CMOS	Via de dados para comunicação USART assíncrona
8	RB2	TTL	CMOS	I/O digital bidirecional com pull-up interno
	TX	-	CMOS	Transmissão para comunicação USART assíncrona
	CK	ST	CMOS	Via de clock para comunicação USART síncrona
9	RB3	TTL	CMOS	I/O digital bidirecional com pull-up interno
	CCP1	ST	CMOS	I/O para o Capture, Compare e PWM
10	RB4	TTL	CMOS	I/O direcional bidirecional com pull-up interno. Interrupção por mudança de estado.
	PGM	ST	-	Entrada para programação em baixa tensão (5V)
11	RB5	TTL	CMOS	I/O digital bidirecional com pull-up interno. Interrupção por mudança de estado.
12	RB6	TTL	CMOS	I/O digital bidirecional com pull-up interno. Interrupção por mudança de estado.
	T1 OSO	-	XTAL	Saída para cristal externo para TMR1
	T1 CKI	ST	-	Entrada externa do contador TMR1
	PGC	ST	-	Clock da programação serial (ICSP)
13	RB7	TTL	CMOS	I/O digital bidirecional com pull-up interno. Interrupção por mudança de estado.
	T1 OSI	XTAL	-	Entrada para cristal externo para TMR1
	PGD	ST	CMOS	Data da programação serial (ICSP)
5	Vss	P	-	GND
14	Vdd	P	-	Alimentação positiva

Figura 20 - Nomenclatura da Pinagem PIC 16F628A

Fonte: Souza (2007 p.37-38)

Legenda:

P = *Power* (alimentação)
- = Não-utilizado
TTL = Entrada tipo TTL
ST = Entrada tipo *Schmitt Trigger*
CMOS = Saída do tipo CMOS
OD = Saída tipo Dreno Aberto (Open Drain)
NA = Entrada / Saída analógica (SOUZA, 2007, p.37-38)

2.3 PROGRAMAÇÃO MICROCONTROLADOR

O microcontrolado PIC 16F628A será programado pelo software MPLAB da empresa MICROCHIP “figura 21”, na linguagem de programação C++.

Figura 21 - Logo Empresa MICROCHIP
Fonte: Microchip Tecnology (2010, p.1)

2.3.1 MPLAB

O MPLAB é um sofware que estabelecer a lógica de programação do microcontrolador.

2.4 LINGUAGEM C++

Conforme Wikipédia (2010), o C++ é uma linguagem de programação multiparadigma e de uso geral. A linguagem pode ser considerada de médio nível, visto combinar as características de linguagens de alto e baixo níveis.

> Em C++, a biblioteca padrão é uma coleção de classes, funções e variáveis escritas na própria linguagem para facilitar o desenvolvimento de aplicações. Também incorpora a biblioteca padrão do C, e todas suas funcionalidades estão declaradas no espaço de nomes” “A biblioteca padrão do C++ fornece vários containers genéricos, funções que utilizam e manipulam tais containers, funções-objeto, cadeias de caracteres e *streams* genéricos, suporte para algumas facilidades da linguagem e funções de uso geral, como funções matemáticas. (WIKIPÉDIA, 2010, p.1).

Linguagem C++ oferece maior facilidade à programação por ter instruções mais especificas de definidas. Tem uma ampla biblioteca para uma melhor aplicação nas mais particulares rotinas. (WIKIPÉDIA, 2010).

Segue a seguir a biblioteca padrão do C++, Conforme Wikipédia (2010, p.2):

Lista de cabeçalhos

Containers

<bitset> - manipulação de arranjo de bits, algo parecido com vector<bool> (que é uma construção não recomendada[1])

<deque> - manipulação de lista duplamente ligada

<list> - manipulação de lista simplesmente ligada

<map> - manipulação de conjunto associativo ordenado (associação: chave → valor)

<queue> - manipulação de lista FIFO

<set> - manipulação de conjunto

<stack> - manipulação de lista LIFO

<vector> - manipulação de arranjo

Uso geral

<algorithm> - algoritmos genéricos

<functional> - funções-objeto

<iterator> - declaração de iterador

<locale> - manipulação de diversas convenções culturais do utilizador, como a representação de números, moeda e datas, para efeitos de internacionalização

<memory> - funções para o gerenciamento de memória

<stdexcept> - especialização de <exception>, fornece relatórios sobre exceções

<utility> - operações com pares de elementos (comparação e construção)

Cadeias de caracteres

<string> - manipulação de cadeia de caracteres

Streams *e entrada/saída*

<fstream> - manipulação de fluxo de dados em arquivo

<ios> - declaração mais geral de fluxo de dados

<iostream> - manipulação de fluxo de dados padrão do sistema (entrada padrão, saída padrão e saída de erros padrão)

<iosfwd> - declaração dos fluxos de dados presentes na linguagem

<iomanip> - manipulação da apresentação e do processamento de fluxos de dados

<istream> - manipulação de entrada de dados

<ostream> - manipulação de saída de dados

<sstream> - manipulação de fluxo de dados em cadeias de caracteres

<streambuf> - manipulação de buffers de fluxo de dados

[editar] Funcionalidades numéricas

<complex> - manipulação de número complexo

<numeric> - operações com conjuntos numéricos

<valarray> - arranjo de valores mutáveis

Suporte à linguagem C++

<exception> - manipulação de exceção

<limits> - manipulação de limites numéricos dos tipos embutidos na linguagem
<new> - manipulação de alocação e desalocação de memória
<typeinfo> - auxílio para o RTTI do C++
Biblioteca padrão do C
<cassert> - adequação de <assert.h>
<cctype> - adequação de <ctype.h>
<cerrno> - adequação de <errno.h>
<cfloat> - adequação de <float.h>
<climits> - adequação de <limits.h>
<cmath> - adequação de <math.h>
<csetjmp> - adequação de <setjmp.h>
<csignal> - adequação de <signal.h>
<cstdlib> - adequação de <stdlib.h>
<cstddef> - adequação de <stddef.h>
<cstdarg> - adequação de <stdarg.h>
<ctime> - adequação de <time.h>
<cstdio> - adequação de <stdio.h>
<cstring> - adequação de <string.h>
<cwchar> - adequação de <wchar.h>
<cwctype> - adequação de <wctype.h>

Descrições detalhadas

Este cabeçalho de acordo com a Wikipédia (2010),fornece diversos algoritmos genéricos proveitoso para busca, ordenação e transformação de containers (estruturas de dados), entre outros. Podem ser invocados para diferentes containers por meio da interface comum dos iteradores, e de operadores específicos que cada algoritmo requisita da estrutura de dado usada. Os algoritmos comumente são especificados através da posição de começo e de fim da estrutura de dados, e o iterador no final da estrutura deve ter acessibilidade a partir do iterador no começo da mesma estrutura através de contínuos incrementos no iterador. Alguns algoritmos promovem uma condição especial de entrada; por exemplo os algoritmos de busca binária, que demandam uma estrutura de dados já ordenada previamente. Notar entretanto que esse requerimento é implícito e não detectável em tempo de compilação, cabendo a competência de atender aos requisitos ao desenvolvedor. (WIKIPÉDIA, 2010).

Os algoritmos podem ser classificados dessa biblioteca em dois grandes grupos, os realizam ou ocorrem mutação no containers e os que não realizam mutação. Algoritmos de busca, comparação e contagem de elementos não realizam mutação, apenas lêem o container e retroceder como saída referências para elementos da estrutura. Em compensação, algoritmos de ordenação, de cópia, de

transformação e de adição ou remoção de elementos realizam mutação. (WIKIPÉDIA, 2010).

Para algoritmos que realizam mutação, contam com o sufixo especial copy, que recomenda que o algoritmo mantém intacto o container passado como parâmetro, retornando na saída um novo container que corresponde ao container original mais o processamento realizado. Para todos os algoritmos, modificável ou não, existe o sufixo especial _if, usado em algoritmos que envolvem comparação de elementos. A indicação é que uma função de comparação está sendo passada por parâmetro para o algoritmo, favorecendo a utilização dos operadores padrão de comparação que são fornecidos pelo tipo de dado do container. (WIKIPÉDIA, 2010).

Com os recursos de informações disponíveis na Wikipédia (2010), valeu-se dos mesmos para o se descreve a seguir:

1) <fstream>

std::fstream é um transmissor de fluxos de dados de arquivos de computador designado para o tipo de dado nativo char. Permite ler e escrever em modo de texto (deve-se utilizar os operadores de deslocamento de bits, << e >>) ou binário (utiliza-se os métodos read e write para buffers de dado). A biblioteca padrão também fornece classes para casos de uso de std::fstream somente para leitura (std::ifstream) ou somente para escrita (std::ofstream). (WIKIPÉDIA, 2010).

A implementação de std::fstream segue o padrão RAII. O gerenciamento do arquivo aberto (um recurso do sistema) é de responsabilidade da classe. Isso implica que ao inicializar std::fstream (através de informações como o nome do arquivo e o modo de abertura) o recurso é adquirido, e na destruição de std::fstream o recurso é liberado ao sistema automaticamente. Apesar da biblioteca padrão fornecer o método close para a liberação manual do arquivo ao sistema, o RAII permite que isso seja feito automaticamente quando a instância de std::fstream sai de escopo no programa.(WIKIPÉDIA, 2010).

2) <functional>

Este cabeçalho fornece suporte para funções-objeto, classes que encapsulam funções de forma que a instância da classe possa ser invocada tal qual uma função qualquer. Por exemplo, as funções-objeto são usadas na STL para a passagem de predicados aos algoritmos genéricos de <algorithm>. São disponibilizadas funções-objeto unárias (que demandam um argumento) e binárias (que demandam dois argumentos), adaptadores que permitem converter ponteiros de funções em

funções-objeto e adaptadores que permitem converter funções-objeto binárias em unárias ao associar um valor a um dos argumentos. (WIKIPÉDIA, 2010).

O cabeçalho ainda define algumas funções-objeto de uso geral tais como operações aritméticas e lógicas. Um exemplo é equal_to, uma função-objeto binária que testa se dois valores são iguais. Nada mais é que uma função de comparação genérica encapsulada em uma classe. (WIKIPÉDIA, 2010).

3) <iostream>

Este cabeçalho tem sua responsabilidade em manipular o fluxo de dados padrão do sistema (entrada padrão, saída padrão e saída de erros padrão) e tem representatividade de simulação de evolução do cabeçalho <stdio.h> da linguagem C. São apresentados os objetos cin, cout, cerr e clog para o envio e recebimento de dados dos fluxos de entrada, saída, erro sem *buffer* e erro com buffer, ao mesmo tempo; para isso usa-se os operadores de deslocamento de bits (<< e >>).(WIKIPÉDIA, 2010).

Também são fornecidos métodos para a formatação do fluxo de dados, como width, que define uma largura para a saída, fill, que define um caractere específico para ser impresso caso o fluxo é menor que o mínimo esperado, e precision, que define a quantidade de dígitos significativos de números de ponto flutuante. (WIKIPÉDIA, 2010).

Alguns compiladores são incapazes de remover código desnecessário ao produzir executáveis que incluem esta biblioteca através de ligação estática. Por exemplo, um programa Olá Mundo usando-se a implementação GNU da biblioteca padrão produz um executável maior que o equivalente utilizando-se <cstdio> devido parcialmente a deficiências do ligador. (WIKIPÉDIA, 2010).

4) <locale>

Este cabeçalho manipula diversas convenções culturais do utilizador, como a representação de números, moeda e datas, para efeitos de internacionalização. A biblioteca faz uso da faceta, uma interface para um serviço dum locale específico. Cada locale possui um conjunto de facetas. O construtor padrão da classe std::locale define uma cópia do locale da máquina executando o programa, com as convenções atuais do utilizador. (WIKIPÉDIA, 2010).

5) <map>

O container std::map<Key, Data, Compare, Alloc> é um conjunto associativo ordenado que mapeia um objetos do tipo Key (a chave) em objetos do tipo Data (o

valor). As chaves são únicas: se um objeto é inserido com uma chave já existente, o valor presente é substituído pelo valor inserido. (WIKIPÉDIA, 2010).

O tempo requerido para acesso aleatório a cada elemento é $O(\log(n))$, e os iteradores atribuídos não são invalidados após as operações de inserção e remoção. Portanto, a implementação mais usada para o container é a árvore de busca binária auto-balanceada (ainda que qualquer outra estrutura de dados que respeite as restrições de complexidade computacional pode ser usada, como uma *skiplist*).

Internamente, os elementos do mapa são ordenados através das chaves.

Uma variação do container é o std::multimap, que permite chaves repetidas. (WIKIPÉDIA, 2010).

6) <set>

O container std::set<Key, Compare, Alloc> é um conjunto associativo que permite acesso aleatório rápido aos dados. Difere do container std::map pois os valores dos elementos também são suas chaves. Por esse motivo, cada valor (e, portanto, sua chave) é único, não pode repetir. O container pode ser acessado de forma bidirecional, a partir do começo ou do fim. (WIKIPÉDIA, 2010).

A implementação interna do container geralmente é uma árvore de busca binária. Uma variação do container é o std::multiset, um multiconjunto associativo, que permite valores repetidos. (WIKIPÉDIA, 2010).

7) <sstream>

std::stringstream é um manipulador de fluxos de dados de cadeias de caracteres especializado para o tipo de dado nativo char. Ele permite ler e escrever em modo de texto (utiliza-se os operadores de deslocamento de bits, << e >>) ou binário (utiliza-se os métodos read e write para buffers de dado).

A biblioteca padrão também fornece classes para casos de uso de std::stringstream somente para leitura (std::istringstream) ou somente para escrita (std::ostringstream). (WIKIPÉDIA, 2010).

8) <string>

O container std::string é uma cadeia de caracteres especializada para o tipo de dado nativo char. Ele remove vários dos problemas introduzidos pela linguagem C ao confiar no programador no gerenciamento de cadeias de caractere, encapsulando internamente rotinas e considerações que o programador não precisa tomar conhecimento. Ele também permite conversão de e para cadeias de texto do C (const char*). (WIKIPÉDIA, 2010).

Distinto de uma cadeia de caracteres em C, que compreende um ponteiro para uma região de memória contendo a cadeia, o conteúdo de std::string é armazenada por valor. Por esse motivo, a operação de cópia possui tempo *O*(*n*). Para evitar cópias desnecessárias, a passagem de cadeias de texto como parâmetro de funções é geralmente feita por referência constante. Isso tem uma segunda vantagem visto permitir que a mesma função possa receber implicitamente uma cadeia de caracteres do C, sem necessitar sobrecarga para o tipo de dado const char*.

A especialização para cadeias de char nem sempre é desejada. Por exemplo, uma aplicação que implemente UTF-32 (o mapeamento do padrão Unicode para 32 bits, suficiente para representar todo o padrão) deve reservar quatro bytes para cada caractere, enquanto um char armazena somente um byte. Por isso, o conceito genérico de cadeia de texto é implementado no container std::basic_string, que pode ser especializado para qualquer tipo de dado. No caso de UTF-32, pode-se usar uma especialização do container genérico para um tipo de dado que armazene pelo menos quatro bytes. Como na maioria das vezes um char é suficiente, a especialização é também definida no padrão por conveniência. (WIKIPÉDIA, 2010).

9) <vector>

O container std::vector é um arranjo e generaliza o conceito de um vector em C. O acesso pode ser através de índices para o dados assim como em C (através de uma sobrecarga do operador adequado) e sua memória está colocada de forma contígua. Entretanto, diferente de um vetor em C, o tamanho do container é dinâmico que gerencia-se automaticamente e há uma flexibilidade maior para adicionar elementos. Além de conhecer seu tamanho atual, uma instância de std::vector também pode se conhecer quantos elementos ainda pode alocar antes de precisar de redimensionamento: a alocação é feita em blocos e não para cada elemento, e é possível definir manualmente qual o tamanho do bloco. Diferente dum vetor em C, std::vector fornece o método at para acessar elementos pelo índice, mas com tratamento de exceções para o caso de índices inválidos (valores negativos ou acima do tamanho atual do container).

Para inserções no meio do container std::list é mais eficiente. Enquanto std::vector fornece inserção eficiente no final do container, std::deque fornece inserção eficiente tanto no começo quanto no final do container. Além de inserção

eficiente no final, pontos positivos do std::vector incluem acesso em tempo constante a qualquer elemento através do índice e iteração em tempo linear.[8]

Sendo um container genérico, pode ser especializado para diferentes tipos de dado. Entretanto, desaconselha-se especializá-lo para o tipo booleano nativo, bool, cuja especialização já consta na biblioteca padrão. (WIKIPÉDIA, 2010).

2.5 COMUNICAÇÃO SERIAL

Ao que expressa Canzian (s/d) a distância que um dado sinal percorre em um computador tem variação de alguns milímetros, como no caso de conexões de um simples CI, até vários centímetros quando a conexão de sinais engloba, por exemplo, uma placa mãe com conectores destinado a diversos circuitos. O dado digital para estas distâncias, pode ser transmitido diretamente. Exceto em computadores muito rápidos, os projetistas não se preocupam com o formato e espessura dos condutores, ou com as características analógicas dos sinais de transmissão.

2.5.1 Porta de Comunicação RS 232

Conforme Braga (2009, p.1) o protocolo RS-232 se expressa da seguinte forma:

> O protocolo RS-232 foi estabelecido em 1962 e era chamado EIA232. Ele foi criado para ser usado com as antigas máquinas de teletipo, mas devido a sua simplicidade e versatilidade tornou-se um padrão que até hoje é adotado em aplicações de curta distância, principalmente as que envolvem a comunicação entre as máquinas (sensores e efetores) e os computadores que as controlam.

A comunicação serial RS232 é um protocolo padronizado de fácil acesso e baixo custo, trabalha com taxas de transmissão padronizadas menores que 20kbps (4.8, 9.6 e 19.2 kbps) e com níveis de tensão referenciados ao pino *TERRA* (Ground - pino 9 no DB-9), sendo que quando o nível de tensão é maior que +3V o sinal do nível lógico é zero (0), e quando o nível de tensão é menor que -3V então o nível lógico será um (1). Por mais que seja uma comunicação extremamente utilizada, o padrão (RS232) obtém sérias limitações em relação a interferências eletromagnéticas. (PEREIRA & LAGES, S/D)

2.5.2 Conexões

Na figura 22 segundo site IDEV, (s/d) tem-se a pinagem de um conector DB-9, definida pela norma EIA-574. Este é encontrada nas portas de comunicação do PC e é usado na comunicação assíncrona de dados (RS-232/V.24).

1 - Detecta portadora (CD)
2 - Recebe dados (RD)
3 - Transmite dados (TD)
4 - Terminal pronto para enviar (DTR)
5 - Terra (SG)
6 - Pronto para comunicar (DSR)
7 - Pronto para enviar (RTS)
8 - Pronto para transmitir (CTS)
9 - Indicador de tom (RI)

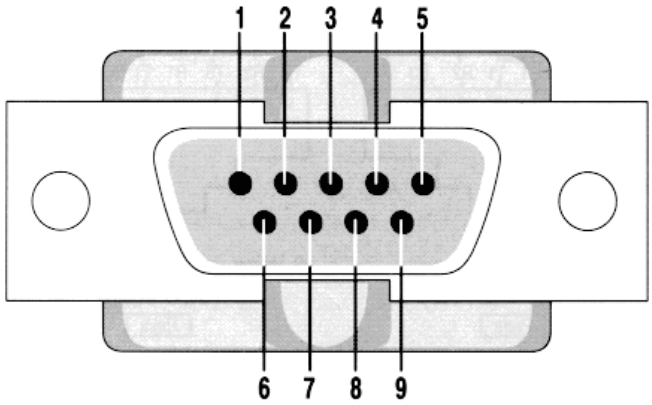

Figura 22 - Pinagem do conector DB-9

Fonte: (Site IDEV (S/D, p.1)

2.5.3 Transmissão serial

O modo mais comum de transmissão de sinais, conforme expressa (Souza (s/d), é o assíncrono. Neste modo não há necessidade do transmissor estar sincronizado com o receptor, pois ele é informado quando cada "pacote de dados" começa e termina, através de bits chamados de *start bit* e *stop bit.* Assim o sinal é

formado por bits individuais que são enviados (um a um) em um "pacote" de tamanho definido no formato ASCII.

Cada pacote é composto de 10 a 11 bits, dos quais 8 bits constituem a porção do pacote referente à parte da mensagem enviada. Estes são enviados depois que um sinal de *start bit* é reconhecido. Para tal a linha (Pino), que está normalmente no ´nível lógico 1`, passa para o ´nível lógico 0`. No final do pacote de bits de dados, pode ser enviado um bit de paridade que serve para verificar se a informação foi recebida com sucesso. Em seguida é enviado um *stop bit,* indicando que o envio deste pacote foi concluído. Se este foi o ultimo pacote, o nível da linha se mantém alto (1), mas se um novo pacote deve ser enviado, tem-se a nova transição do nível 1 para o 0, o que é reconhecido como *start bit*, e o processo se repete. (SOUZA, S/D)

2.5.4 Driver (CI) Para Transformação De Níveis TTL Para RS232

Para realizar uma comunicação entre um equipamento digital e uma interface RS232 é necessário transformar níveis TTL (0 a 5 Volts) em RS232.

O CI MAX232 é dotado de um circuito de charge pump (elevação de carga), capaz de gerar tensões de +10 e -10 Volts a partir de uma fonte de +5 Volts, bastando para isso alguns capacitores externos. Este CI também tem 2 receivers e 2 drivers no mesmo encapsulamento. A figura 23 mostra a pinagem e configuração desse CI. (SOUZA, S/D)

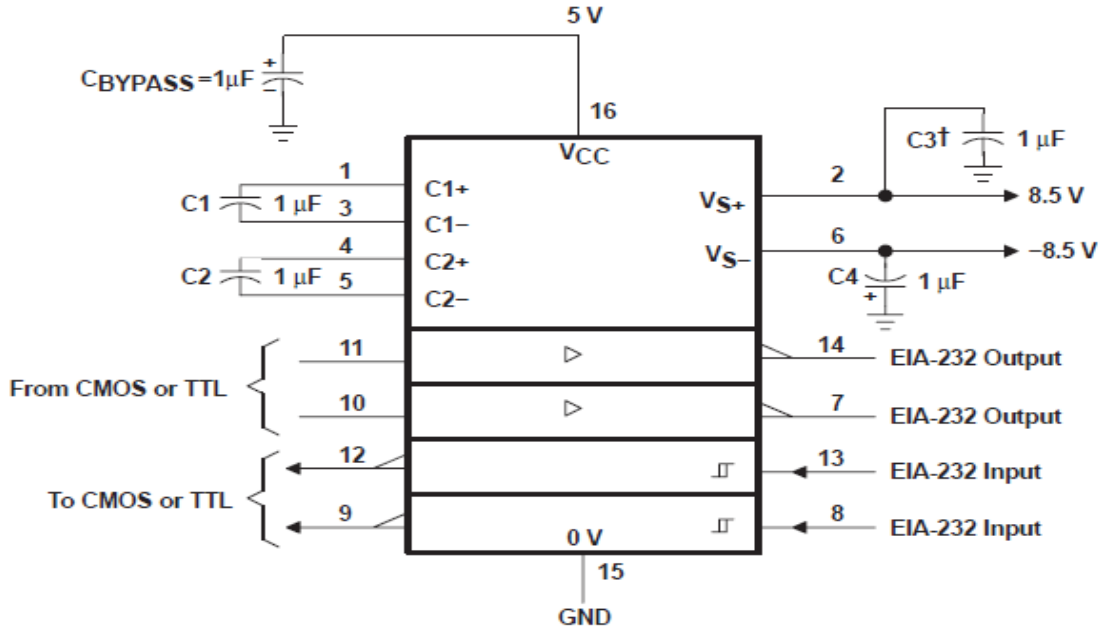

Figura 23 - Ci - Pinagem e configuração do Max 232

Fonte: datasheet Max 232 texas instruments

2.6 DISPOSITIVOS DE REALIMENTAÇÃO – ENCODERS

Com a contribuição de Matias (2008, p.1) tem-se que os encolder's são transdutores que convertem um movimento angular ou linear em uma série de pulsos elétricos. Quando acoplados aos motores, permitem saber suas posições e velocidades possibilitando seu controle. No caso da utilização do encoder em motores de passo proporciona a verificação de passos executados garantindo assim a precisão de seus movimentos. O sistema de leitura dos encoder's é feito através de um sensor óptico "emissor" e um receptor onde entre eles existe um disco rotativo formado por pequenas janelas radiais. Quando esse disco se movimenta as janelas radiais existentes no disco rotativo permitem a passagem da luz infravermelha emitido pelo sensor emissor até o receptor convertendo-a em pulsos elétricos através de uma placa eletrônica. A figura 24 mostra o funcionamento de um encoder rotativo. (MATIAS, 2008, p.1)

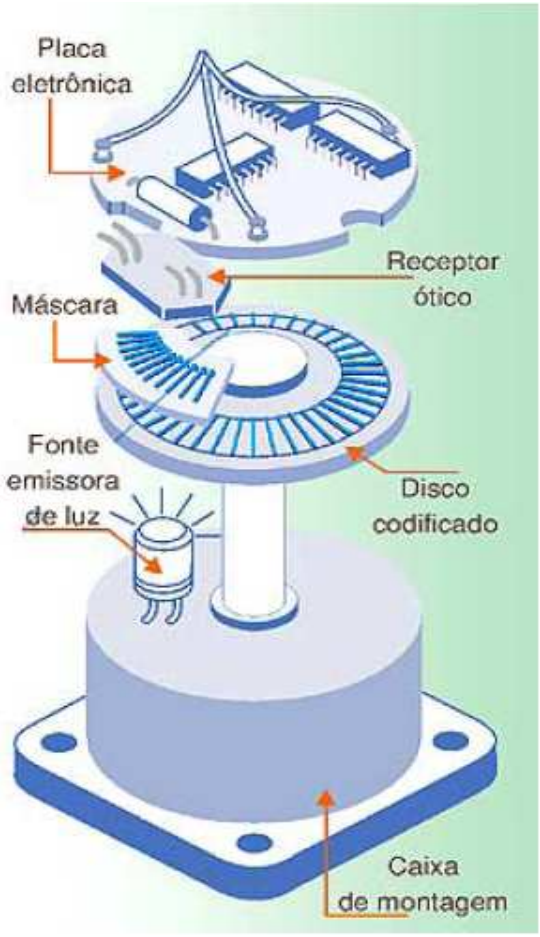

Figura 24 - Funcionamento do encoder rotativo

Fonte: Matias (2008, p.1)

3 DESENVOLVIMENTO DO PROJETO

O projeto foi desenvolvido com a finalidade de compactar e facilitar o controle de um motor de passo, através de periféricos acoplado a ele, onde a sua programação é feita através da comunicação serial e sua configuração é escolhida por um menu desenvolvido na programação em “C++” do microcontrolador.

Os periféricos acoplados ao motor de passo são: um encoder, um driver de potência e o controlador do driver, permitindo a conexão com um computador, possibilitando as configurações necessárias para cada tipo de aplicação.

A figura 25 mostra a estrutura onde serão acoplados os periféricos ao motor de passo.

Figura 25 - Montagem Motor de Passo e Cabine p/ Driver's

Fonte: Os autores

3.1 MENU DE CONFIGURAÇÃO

Ao conectar o cabo de comunicação serial ao computador irá aparecer um menu de configuração do motor de passo em **C++**. Após aberto o menu escolha o parâmetro que se deseja configurar conforme figura 26.

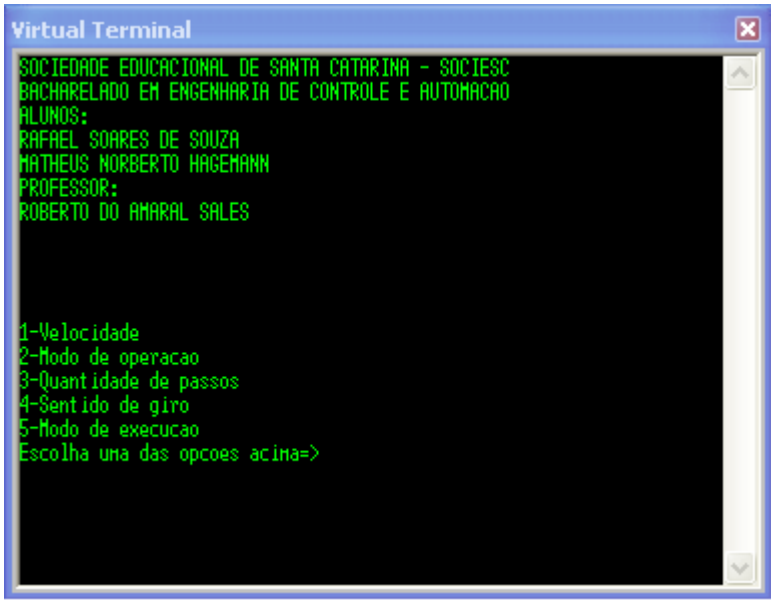

Figura 26 - Tela Principal

Fonte: Os autores

Velocidade:

Sendo optado proceder com a configuração de velocidade aperte o número um (1) do Teclado, conforme mostrado na figura 27.

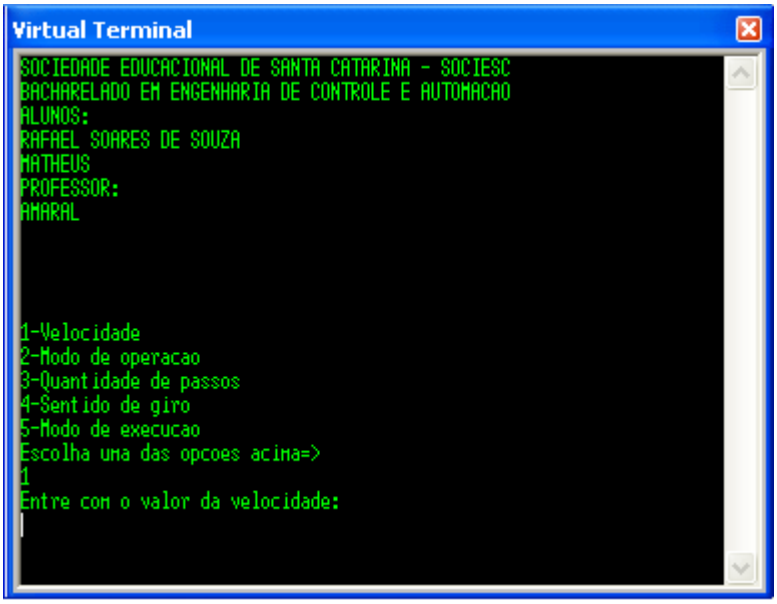

Figura 27 - Tela Parâmetro Velocidade

Fonte: Os autores

Determinar o valor desejado conforme exemplo na figura 28. Nesse caso foi posto o valor mil milissegundos “1000ms” de velocidade.

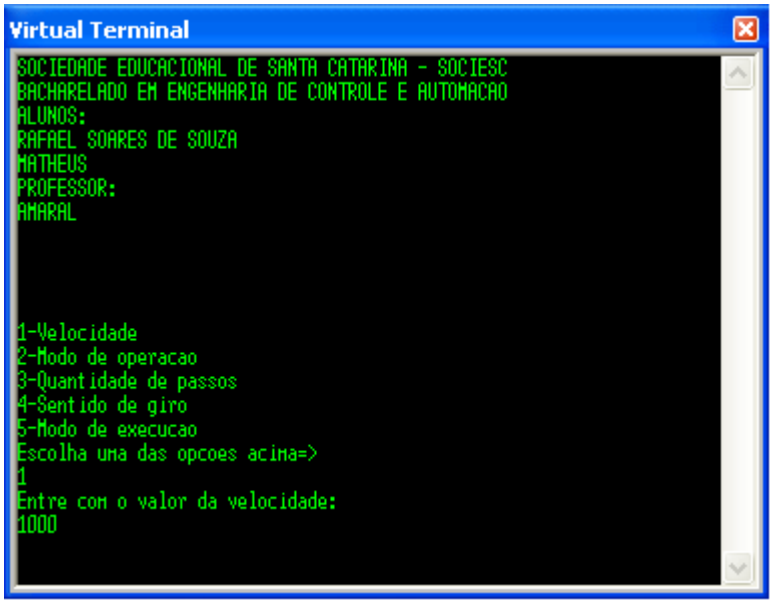

Figura 28 - Tela Exemplo de Parâmetro Velocidade

Fonte: Os autores

Após feito isso pressionar enter conforme figura 29 para carregar o parâmetro configurado, e voltar para a tela principal.

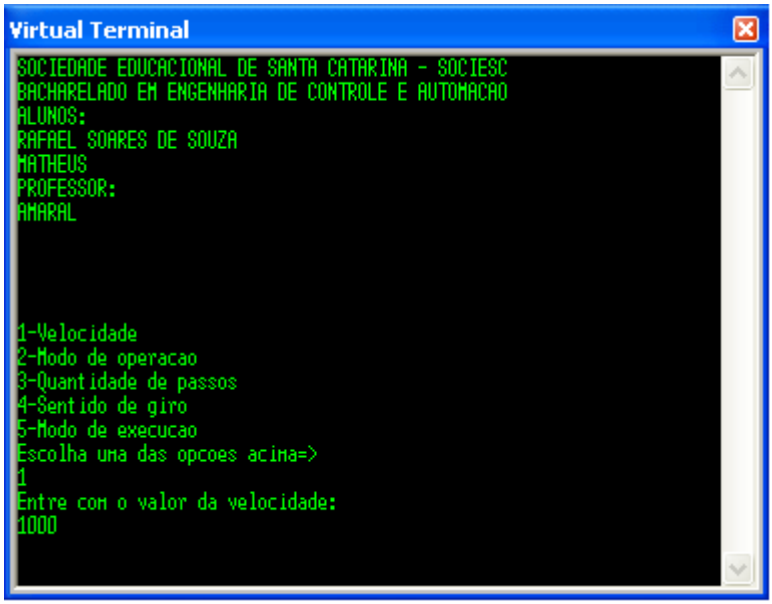

Figura 29 - Tela Carregar Parâmetro Velocidade

Fonte: Os autores

Modo de operação:

No modo de operação pode ser parametrizado o motor de passo para trabalhar nos modos:

•Half-step: Que energiza uma bobina por passo

•Full-step I – Que energiza duas bobinas por passo

•Full-step II – É a combinação Half-step e Full-step, energiza tanto uma como duas bobinas.

Sendo optado proceder com a configuração do modo de operação digite o número dois (2) do Teclado, conforme mostrado na figura 30.

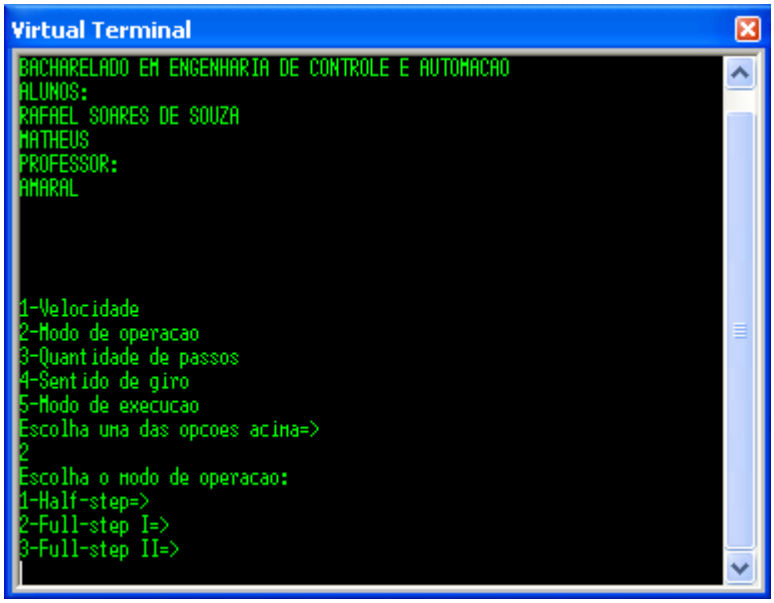

Figura 30 - Tela Modo de Operação

Fonte: Os autores

Para escolher um dos três modos de operação digite, um (1) para Half-Step, dois (2) para Full-Step ou três (3) para Full-Step II. No exemplo abaixo foi optado por trabalhar com o modo de operação Half-Step, conforme mostra a figura 31.

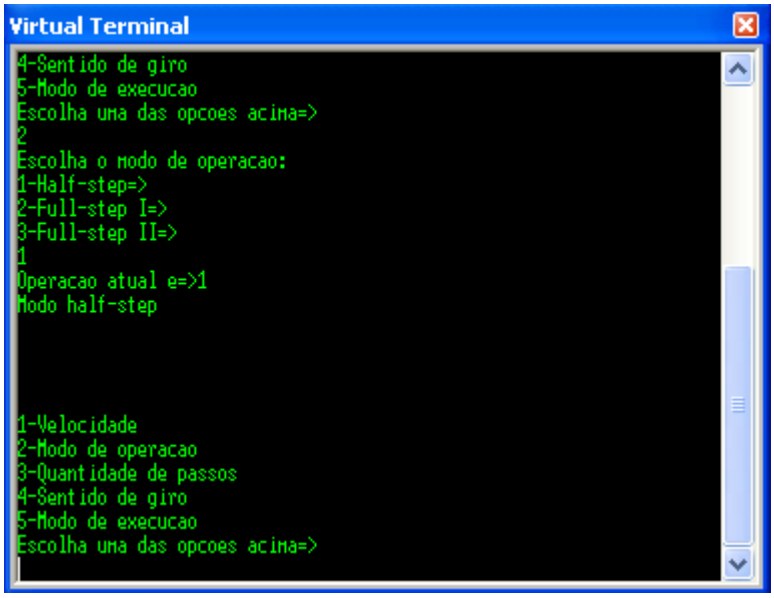

Figura 31 - Tela Modo de Operação Half-Step

Fonte: Os autores

Quantidade de Passos:

Determina o número de passos que se deseja obter na operação.

O número de pulsos do motor de passo depende do ângulo de passo, assim quanto menor é esse ângulo maior é o número de passos que o motor de passo pode dar em uma volta de 360º graus.

Esse parâmetro possibilita escolher a quantidade de passos desejada operar.

Exemplo:

Para saber a quantidade de passos necessários para que o motor dê uma volta completa de 360º, execute os seguintes cálculos:

- Passos por volta = 360º **/** 1.8º
- Passos por volta = 200.

A quantidade de passos para o motor completar uma volta, sabendo-se que sua precisão é de 1.8º, será de 200 passos.

Ao optar em proceder com a configuração de quantidade de passos digite o número três (3) para executar a configuração conforme figura 32.

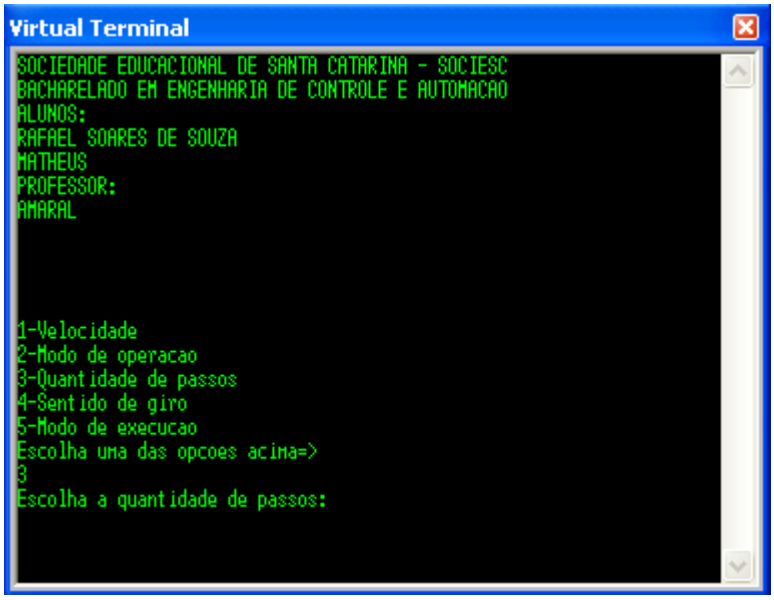

Figura 32 - Tela Quantidade de Passos

Fonte: Os autores

Após dar entrada na parametrização determine o valor do número de passo que se deseja obter na operação. No exemplo conforme figura 33 foi posto o valor 200 para o motor completar uma volta ou seja 360 graus.

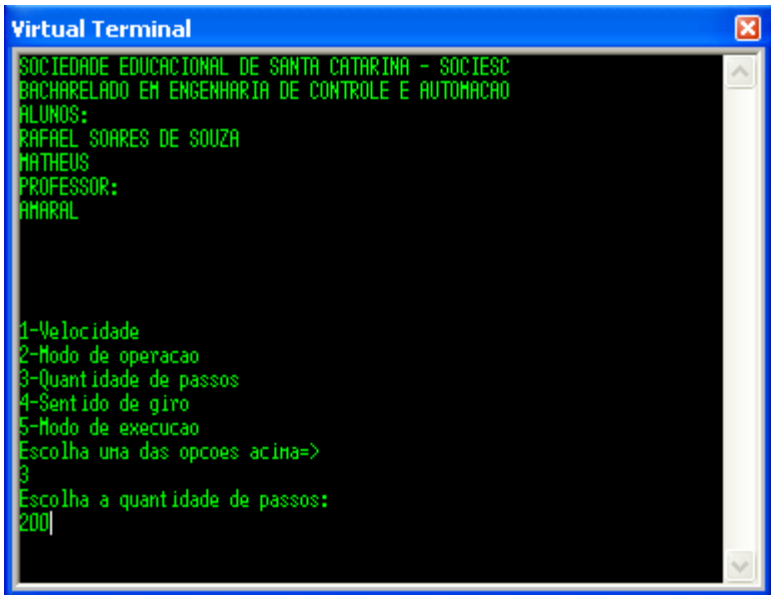

Figura 33 - Tela Exemplo de Quantidade de Passos

Fonte: Os autores

Após feito isso pressionar enter conforme figura 34 para carregar o parâmetro configurado, e voltar para a tela principal.

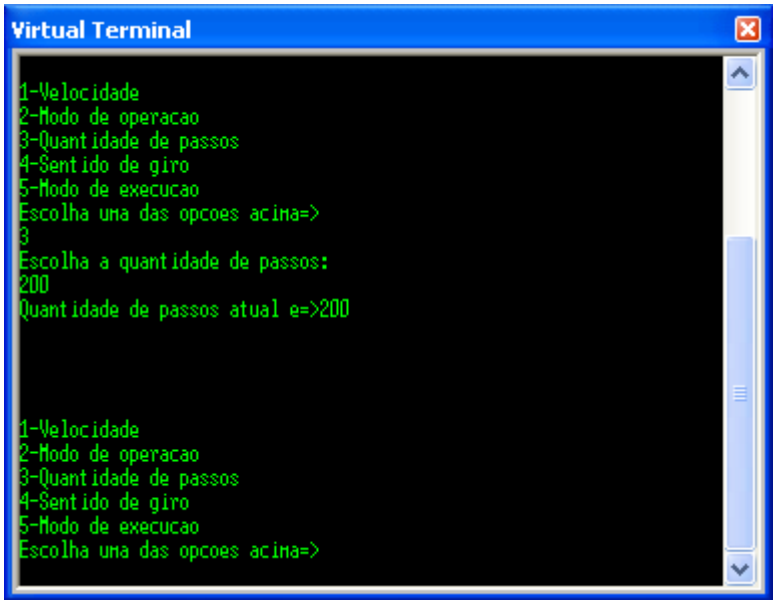

Figura 34 - Tela Carregar Parâmetros de Quantidade de Passos

Fonte: Os autores

Sentido de Giro:

Determina o sentido que se deseja rotacionar o motor de passo.

Para entrar na tela de configuração do sentido de giro digite o numero quatro (4) conforme demonstra a figura 35.

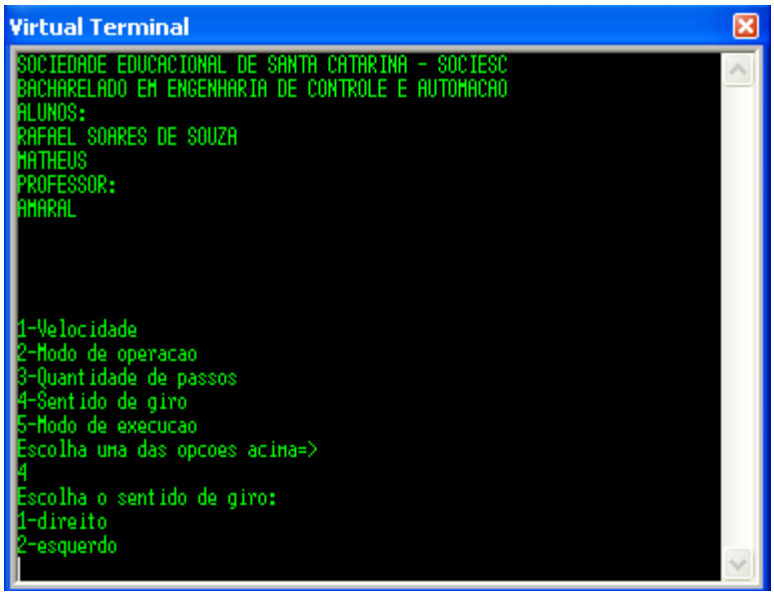

Figura 35 - Tela Sentido de Giro

Fonte: Os autores

Após feito isso digite um (1) para sentido direito (horário) ou dois (2) para sentido esquerdo (anti-horário), nesse exemplo foi escolhido a opção um (1) para o motor de passo operar rotacionando no sentido direito (horário), conforme figura 36

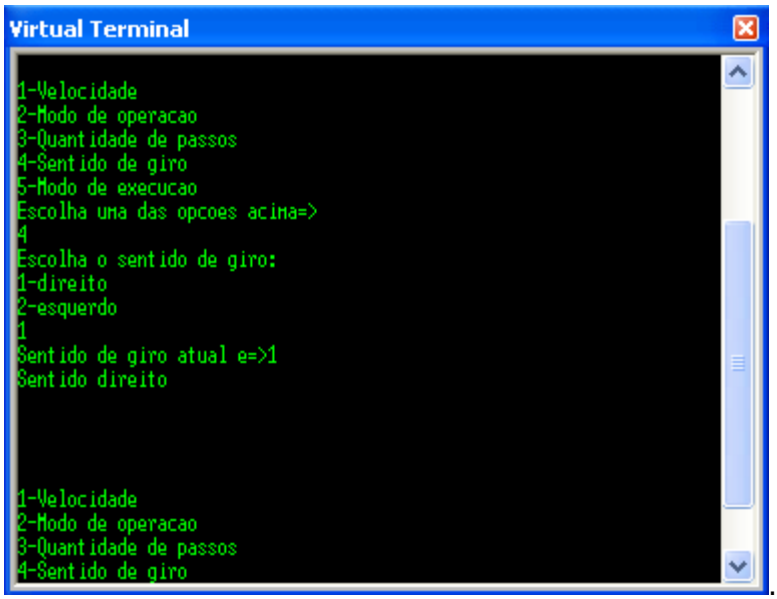

Figura 36 - Tela Exemplo Sentido de Giro

Fonte: Os autores

Modo de Execução:

È o start remoto da operação configurada. Esse inicia o ciclo de execução do programa. Após a escolha de todos os parâmetros desejados, digite cinco (5) para executar o programa e mostrar as configurações escolhidas.

A figura 37 demonstra a tela das configurações que serão sendo executadas conforme a parametrização vista anteriormente.

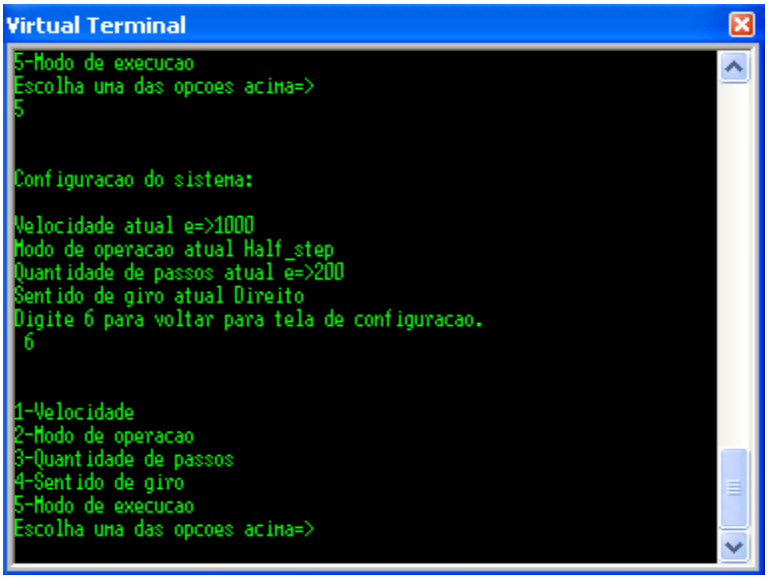

Figura 37 - Modo de Execução

Fonte: Os autores

3.2 PERIFÉRICOS ACOPLADOS AO MOTOR DE PASSO

3.2.1 Encoder

O encoder será responsável pela monitoração de cada passo, garantindo uma precisão no posicionamento. Se acontecer de o motor de passo escorregar, uma mensagem irá aparecer indicando um erro.

Sua montagem será desenvolvida através de um disco de acrílico e dois fotodiodos, onde o código binário gerado será impresso nesse disco, de tal forma que irá controlar a posição e sentido de giro do motor de passo.

O encoder será de 400 pulsos por rotação (incremental), 2 bits de precisão, raio interno de 200 mm, largura de banda 100mm, com mascara para fotodiodo, seu forma construtiva foi dimensionada para esse devido projeto, através do site Optical Encoder wheel generator (s/d), que gerou código binário conforme a figura 38. Disponível em:<*http://www.bushytails.net/~randyg/encoder/encoderwheel.html*>

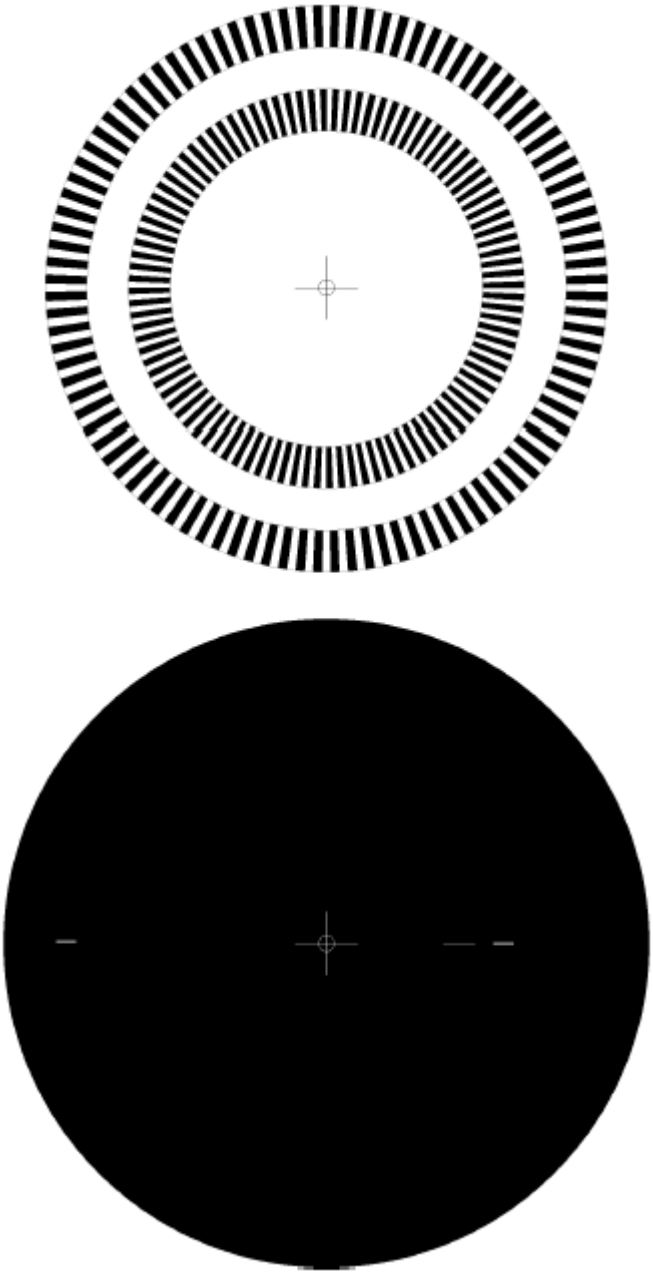

Figura 38 - Mascara Encoder

Fonte: http://www.bushytails.net/~randyg/encoder/encoderwheel.html

3.2.2 Driver

O driver de potencia será responsável pelo acionamento do motor de passo. Essa placa tem como objetivo chavear os transistores de potência “NPN” conforme os pulsos recebidos da placa controladora para poder acionar o motor de passo de tal forma de que suporte a corrente nominal do motor de passo, que é três (3) amperes.

Será utilizado os seguis componentes:

- 4 fotodiodos NPN

-8 resistores de 330ohms

-4 diodos de roda livre

- 4 transistores do tipo “tip 122” suportando uma corrente de até 8 amperes.

-Placa de circuito impresso com dimensiona 50xmm.

A figura 39 mostra a montagem do driver de potencia para o motor de passo.

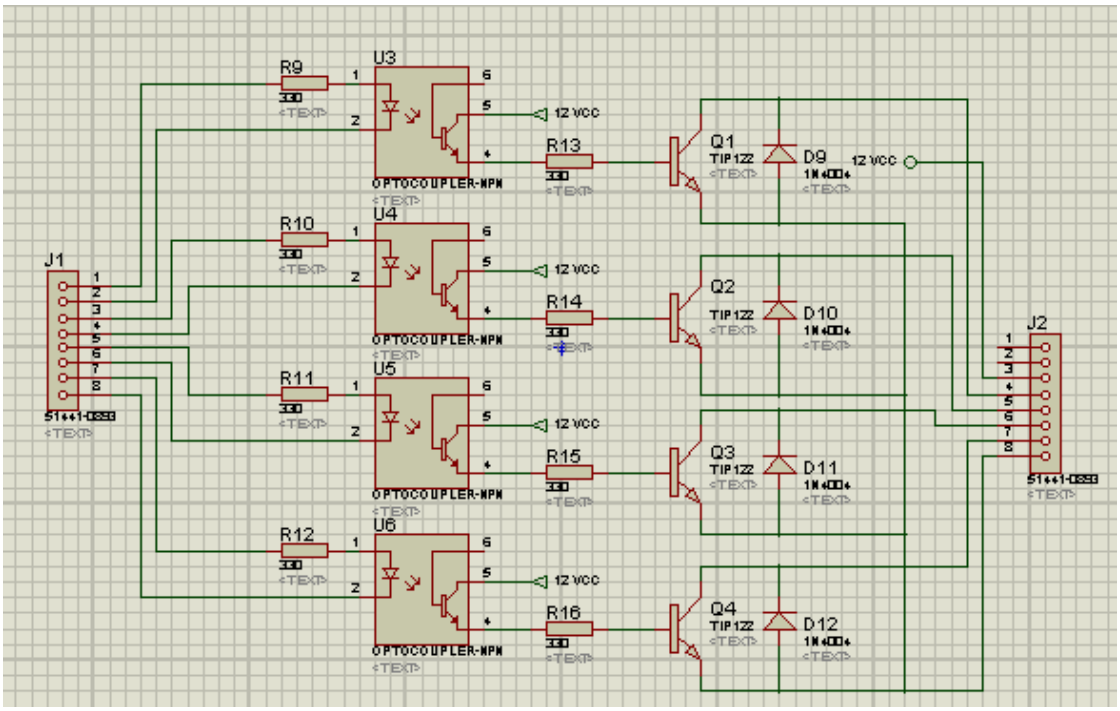

Figura 39 - Projeto Placa Driver

Fonte: Os autores

A figura 40 mostra o esquema dos trilhos da placa do driver de potencia para o motor de passo.

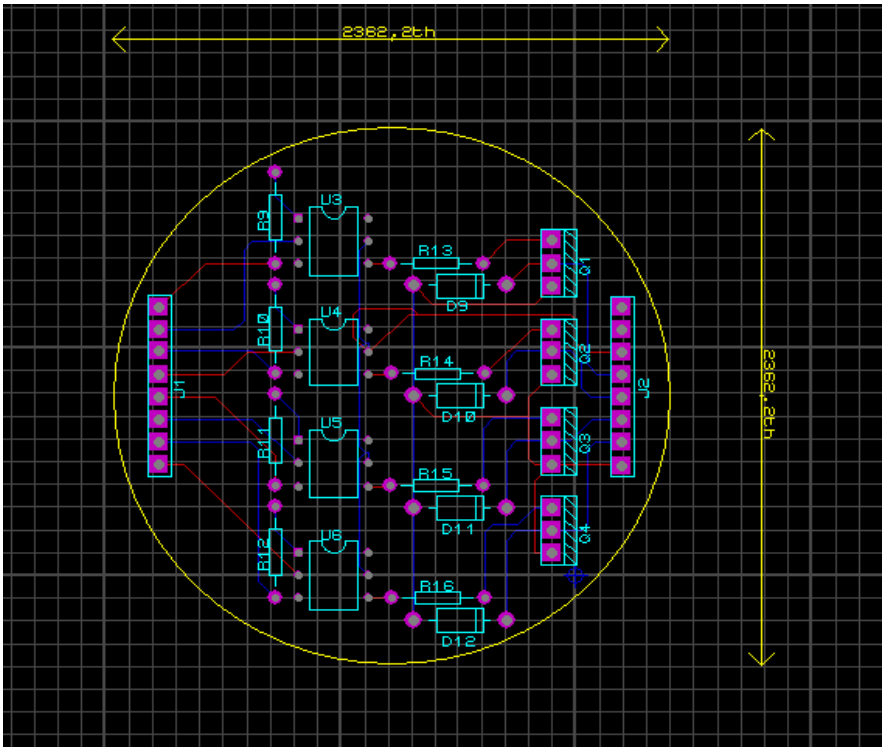

Figura 40 - Esquema da Placa Driver

Fonte: Os autores

A figura 41 demonstra o layout do driver de potência para o motor de passo.

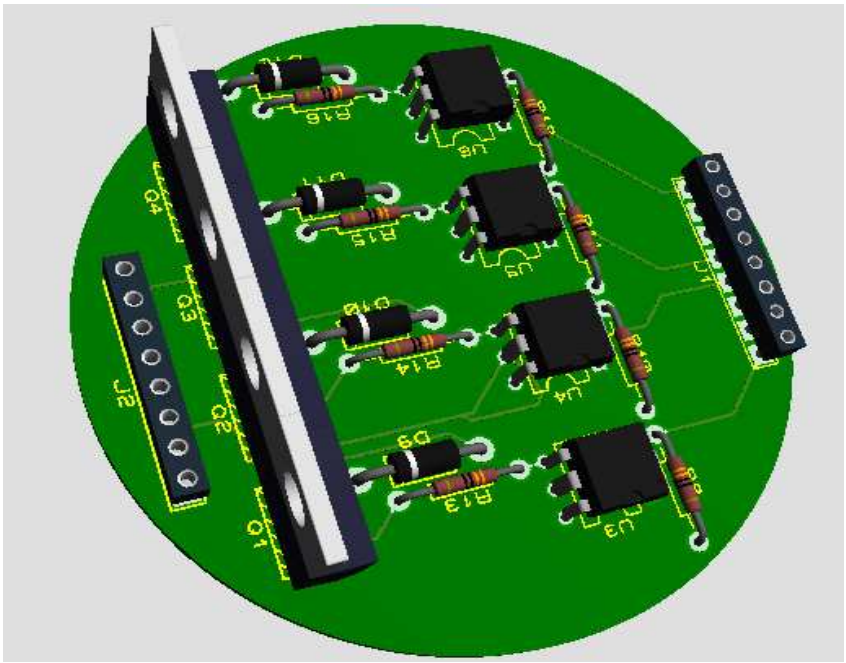

Figura 41 - Layout Placa Driver

Fonte: Os autores

3.2.3 Controlador

Esse é o que abrange toda a lógica do movimento do motor de passo, ele será o responsável por executar na placa Driver os parâmetros: sentido de rotação: forma de trabalho, velocidade do motor, quantidade de passo. Conforme parametrizado no menu de programação.

O controlador utilizado será o pic 16F6248A com a finalidade de gravar e executar a programação desejada. A figura 42 mostra o projeto da placa controladora.

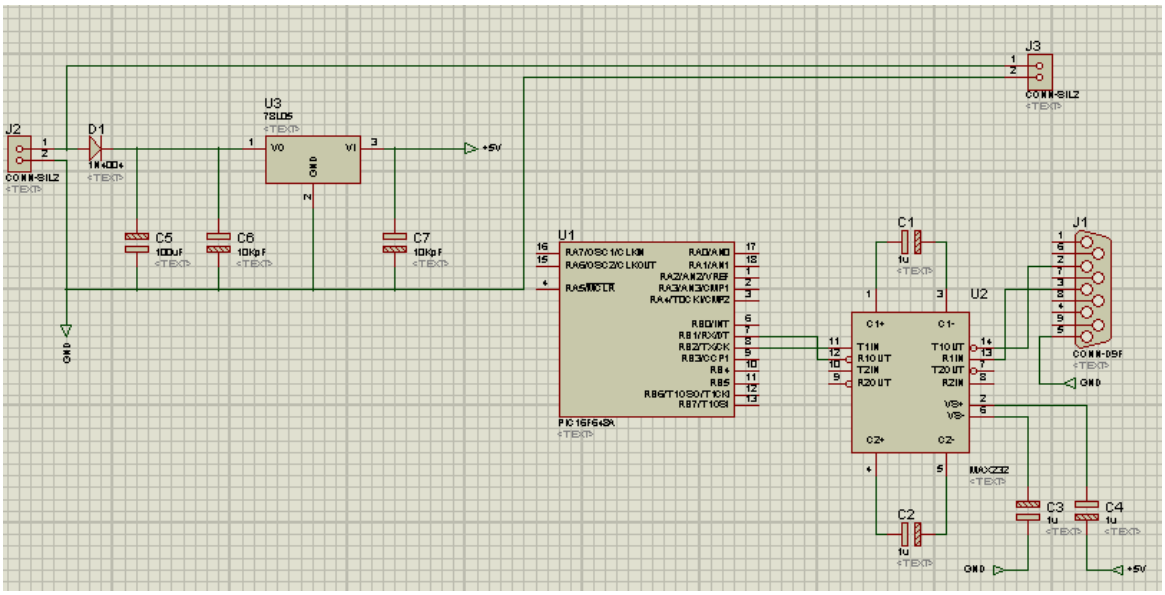

Figura 42 - Projeto Placa Controlador

Fonte: Os autores

A figura 43 mostra o esquema dos trilhos da placa do controlador do motor de passo.

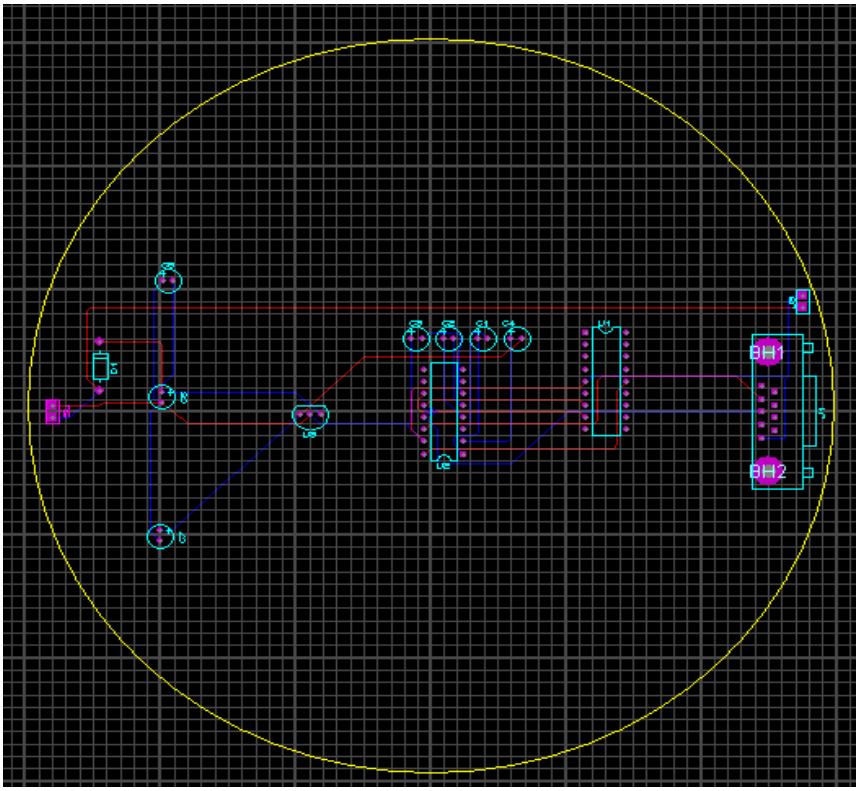

Figura 43 - Esquema Placa Controlador

Fonte: Os autores

A figura 44 demonstra o layout do controlador do motor de passo.

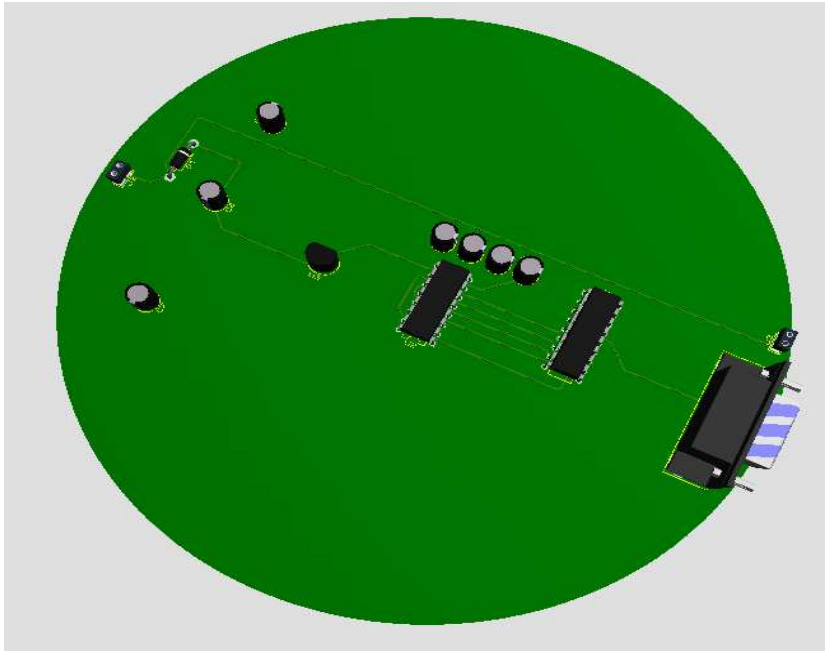

Figura 44 - Layout Placa Controlador

Fonte: Os autores

3.3 DIAGRAMA DE BLOCOS

Em serão apresentados os diagramas de blocos que demonstram as seqüencias lógica obtidas no projeto “MOTOR DE PASSO PROGRAMÁVEL COM DRIVER E ENCOLDER INTEGRADOS”.

A figura 45 demonstra a seqüência de parametrização do Menu Principal 1:

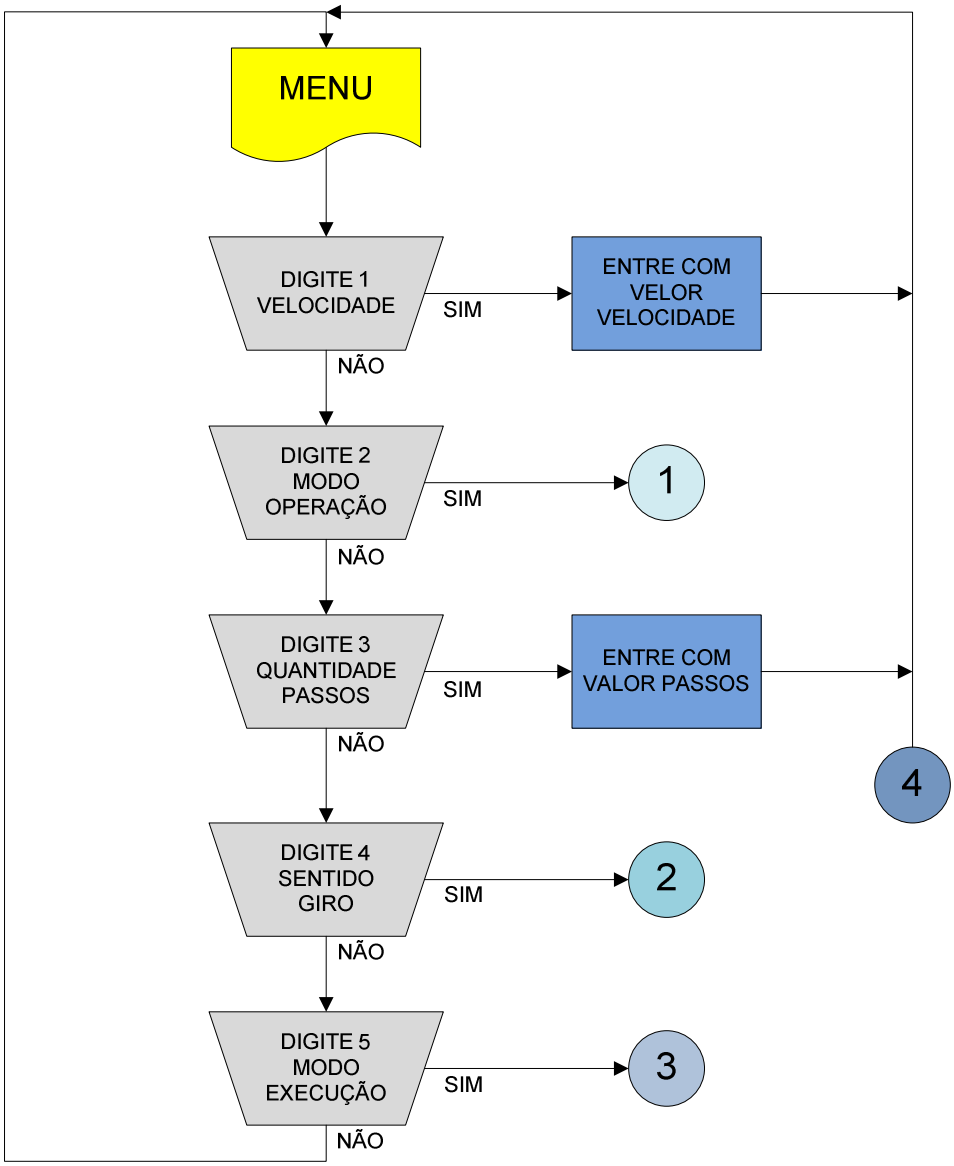

Figura 45 - Diagrama de Blocos Menu Principal 1

Fonte: Os autores

O Menu Principal 2 conforme figura 46, mostra a lógica obtida da programação para a execução dos movimentos do motor de passo:

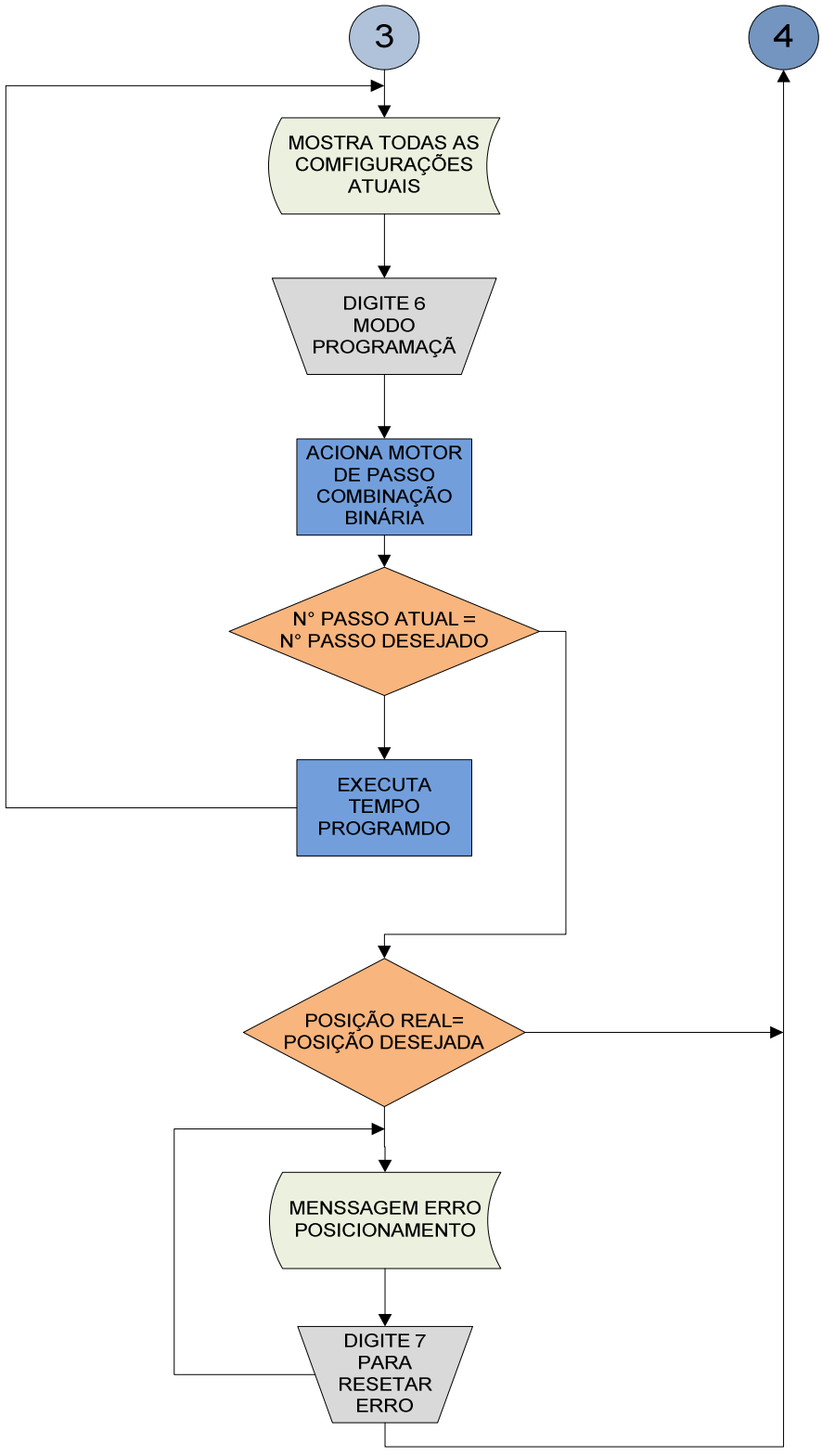

Figura 46 - Diagrama de Blocos Menu Principal 2

Fonte: Os autores

A figura 47 demonstra a seqüência de parametrização do Menu Principal 3, que define o modo de operação que se deseja trabalha no motor de passo, HALL STEP, FULL STEP 1 ou FULL STEP 2:

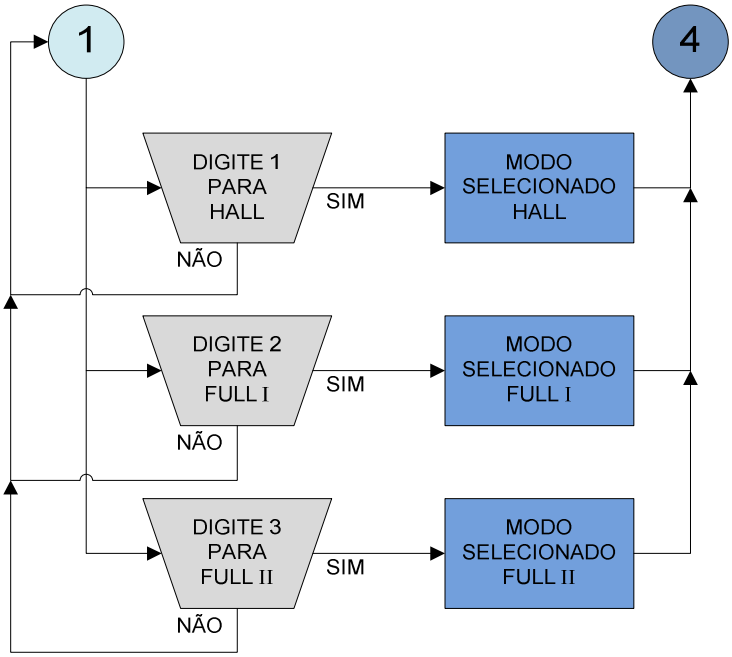

Figura 47 - Diagrama de Blocos Menu Principal 3

Fonte: Os autores

A figura 48 demonstra a seqüência de parametrização do Menu Principal 4, que define o sentido de giro do motor de passo, direita ou esquerda:

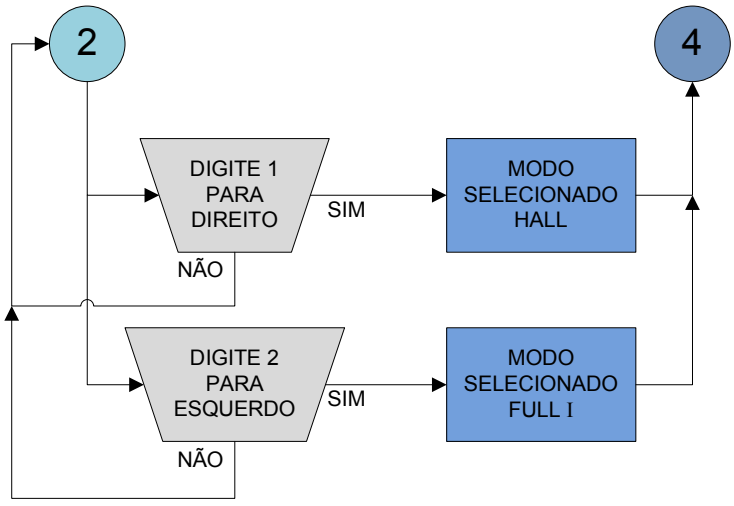

Figura 48 - Diagrama de Blocos Menu Principal 4

Fonte: Os autores

Legenda do diagrama de blocos conforme figura 49:

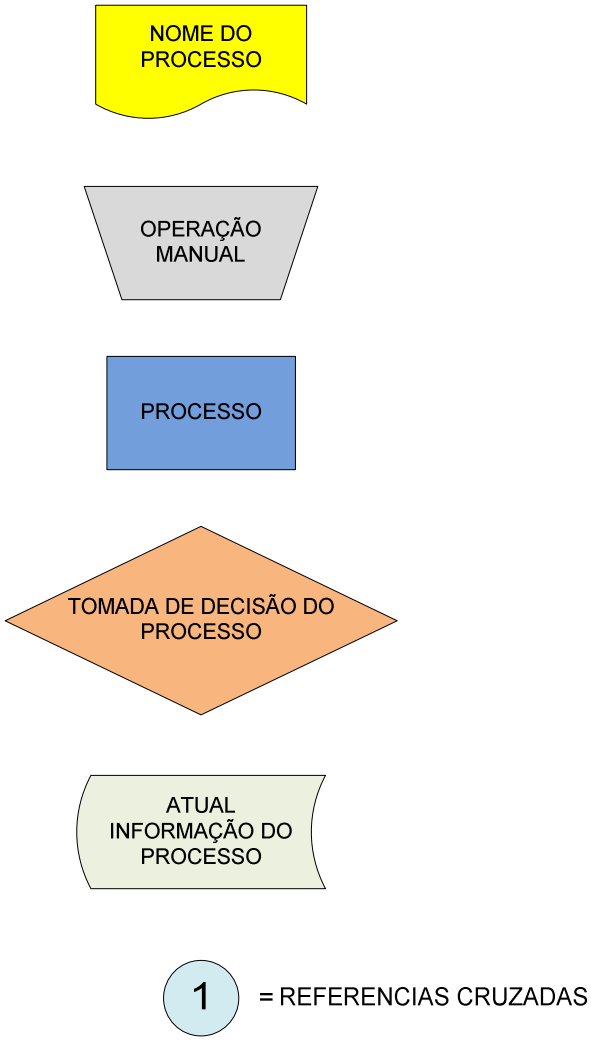

Figura 49 - Legenda do Diagrama de Blocos

Fonte: Os autores

3.4 PROGRAMAÇÃO

Segue o codigo do programa desenvolvido em C++:

```
#include "Motor_Passo_Serial.h"
#include <stdlib.h>
#FUSES NOWDT                    //No Watch Dog Timer
#FUSES XT                       //Crystal osc <= 4mhz
#FUSES NOPUT                    //No Power Up Timer
#FUSES NOPROTECT                //Code not protected from reading
#FUSES NOBROWNOUT               //No brownout reset
#FUSES NOLVP                    //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NOCPD                    //No EE protection
#use delay(clock=4000000)
#use rs232(baud=9600,parity=N,xmit=PIN_B2,rcv=PIN_B1,bits=8)
int passo[8]={0b00000001,
              0b00000011,
              0b00000010,
              0b00000110,
              0b00000100,
              0b00001100,
              0b00001000,
              0b00001001};

int passo1[4]={0b00000001,
               0b00000010,
               0b00000100,
               0b00001000};

int passo2[4]={0b00000011,
               0b00000110,
               0b00001100,
               0b00001001};
```

```
signed int pos;
unsigned int16 w_num,c_num,cont;
int y,z,d,v,h,x;
char c[5],w[13],modo_1[13],modo_2[13],modo_3[13],sentido_1[13],sentido_2[13],
sentido[13],modo[13];
int letra=0;
void apresentacao(void);
void menu(void);
void principal(void);
void main()
{
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);
  setup_ccp1(CCP_OFF);
  setup_comparator(NC_NC_NC_NC);
  setup_vref(FALSE);
  pos=0;
  output_a(passo[pos]);
x=0;
apresentacao();
while (true)
{
if (x==0)
{
cont=c_num;
menu();
}
else
{
principal();
}
if (kbhit())
{
```

```
letra=getc();
if (letra==0x36)
{
x=0;
}
}
}
}
////////////////////////////////////////////////////////////////////////////
void apresentacao(void)
{
printf("SOCIEDADE EDUCACIONAL DE SANTA CATARINA - SOCIESC\n\r");
printf("BACHARELADO EM ENGENHARIA DE CONTROLE E AUTOMACAO\n\r");
printf("ALUNOS:\n\r");
printf("RAFAEL SOARES DE SOUZA\n\r");
printf("MATHEUS NORBERTO HAGEMANN\n\r");
printf("PROFESSOR:\n\r");
printf("ROBERTO DO AMARAL SALES \n\r");
printf("\n\r");
delay_ms(5000);
}
////////////////////////////////////////////////////////////////////////////
void menu(void)
{
do
{
printf("\n\r");
printf("\n\r");
printf("\n\r");
printf("1-Velocidade\n\r");
printf("2-Modo de operacao\n\r");

printf("3-Quantidade de passos\n\r");
```

```
printf("4-Sentido de giro\n\r");
printf("5-Modo de execucao\n\r");
printf("Escolha uma das opcoes acima=>\n\r");
y=getchar();
}
while ((y<'1')||(y>'5'));
   switch(y)
   {
 case '1':
 printf("\n\r");
 printf("Entre com o valor da velocidade:\n\r");
 gets(w);
 w_num=atol(w);
 printf("Velocidade atual e=>");
 puts(w);
 printf("\n\r");
 break;
  case '2':
 printf("\n\r");
 printf("Escolha o modo de operacao:\n\r");
 printf("1-Half-step=>\n\r");
 printf("2-Full-step I=>\n\r");
 printf("3-Full-step II=>\n\r");
 z=getchar();
 printf("\n\r");
 printf("Operacao atual e=>");
 putchar(z);
  switch(z)
  {
  case '1':
  printf("\n\r");
  printf("Modo half-step\n\r");
  printf("\n\r");
  h=1;
```

```
 modo_1="Half_step";
 strcpy (modo,modo_1);
 break;
 case '2':
 printf("\n\r");
 printf("Modo full-step I\n\r");
 printf("\n\r");
 h=2;
 modo_2="Full_step_I";
 strcpy (modo,modo_2);
 break;
 case '3':
 printf("\n\r");
 printf("Modo full-step II\n\r");
 printf("\n\r");
 h=3;
 modo_3="Full_step_II";
 strcpy (modo,modo_3);
 break;
 }
 break;
   case '3':
printf("\n\r");
printf("Escolha a quantidade de passos:\n\r");
gets(c);
c_num=atol(c);
printf("Quantidade de passos atual e=>");
puts(c);
printf("\n\r");
break;
  case '4':
printf("\n\r");
printf("Escolha o sentido de giro:\n\r");
printf("1-direito\n\r");
```

```
printf("2-esquerdo\n\r");
d=getchar();
printf("\n\r");
printf("Sentido de giro atual e=>");
putchar(d);
printf("\n\r");
  switch(d)
 {
 case '1':
 printf("Sentido direito\n\r");
 printf("\n\r");
 v=1;
 sentido_1="Direito";
 strcpy (sentido,sentido_1);
 break;
 case '2':
 printf("Sentido esquerdo\n\r");
 printf("\n\r");
 sentido_2="Esquerdo";
 strcpy (sentido,sentido_2);
 v=2;
 break;
 }
break;
 case '5':
printf("\n\r");
printf("\n\r");
printf("\n\r");
printf("Configuracao do sistema:\n\r");
printf("\n\r");
  printf("Velocidade atual e=>");
puts(w);

printf("Modo de operacao atual %s",modo);
```

```
  printf("\n\r");
  printf("Quantidade de passos atual e=>");
  puts(c);
  printf("Sentido de giro atual %s",sentido);
  printf("\n\r");
  printf("Digite 6 para voltar para tela de configuracao.\n\r ");
  x=1;
  break;
  }
}

////////////////////////////////////////////////////////////////////////////
void principal(void)
{
while (h==1)
    {
        if (v==1)pos++,cont--;
    if (v==2)pos--,cont--;
    if (pos > 7) pos = 0;
    if (pos < 0) pos = 7;
    output_a(passo[pos]);
    if (cont==0)
    {
    x=0;
    }
    delay_ms(w_num);
         break;
}
while (h==2)
{
    if (v==1)pos++,cont--;
    if (v==2)pos--,cont--;
    if (pos > 3) pos = 0;
    if (pos < 0) pos = 3;
```

```
    output_a(passo1[pos]);
    if (cont==0)
    {
    x=0;
    }
    delay_ms(w_num);
    break;
}
    while (h==3)
{
    if (v==1)pos++,cont--;
    if (v==2)pos--,cont--;
    if (pos > 3) pos = 0;
    if (pos < 0) pos = 3;
    output_a(passo2[pos]);
    if (cont==0)
    {
    x=0;
    }
    delay_ms(w_num);
    break;
}

}
```

4 CRONOGRAMA

O cronograma abaixo (figura 50) demonstra a seqüência para o desenvolvimento do projeto

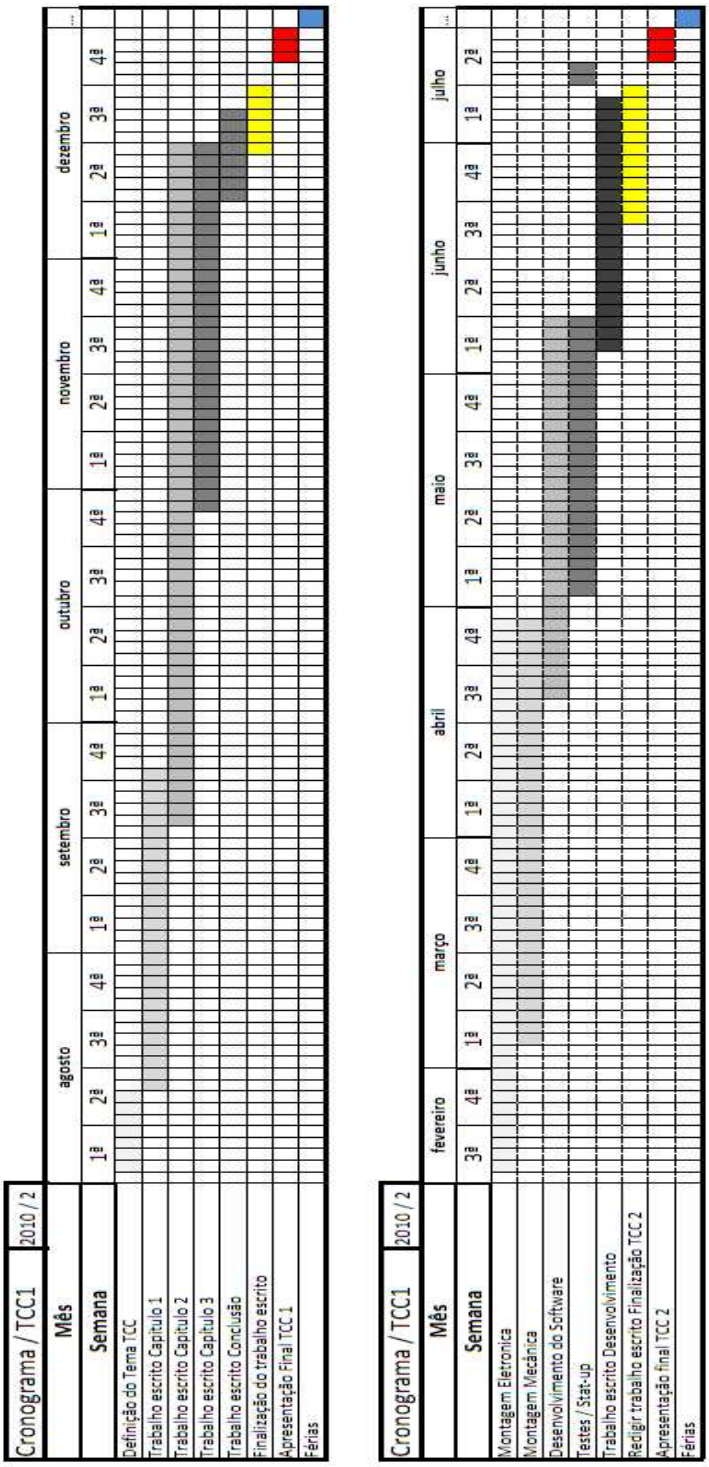

Figura 50 - Cronograma

Fonte: Os autores

5 CONCLUSÃO

O estudo para elaboração desse projeto permitiu adquirir um amplo conhecimento sobre o funcionamento de motores de passo. E o aumento do entendimento e aplicabilidade de microprocessadores utilizando a programação em C++.

Os motores de passo, ainda que sendo uma solução mais barata para industria, ainda são pouco utilizados em aplicações de precisão se forem comparados aos servomotores. Porem observa-se que a sua aplicabilidade em malha fechada tornasse bastante confiável e que ainda é possível reduzir bastante os custos num produto que ofereça precisão em deslocamento em motores de passo.

A muitos produtos oferecidos no mercado industrial em geral podem e devem ser explorados, maximizados para abrangerem o maior nível de aplicações o possível, desse forma, facilita-se a sua implementação e abaixa-se o custo.

REFERÊNCIAS

BRAGA, Newton C. **Comunicação serial na Indústria usando o protocolo RS-232.** Disponível em: <http://www.mecatronicaatual.com.br/secoes/leitura/333> - Acesso em: 18 set. 2010.

BRITES, Felipe G. e SANTOS, Vinicius P. A. **Motor de passo** - Universidade Federal Fluminense. Disponível em: <http://www.telecom.uff.br/pet/petws/downloads/tutoriais/stepmotor/stepmotor2k81119.pdf > - Acesso em: 20 set. 2010.

CANZIAN, Edmur. **Comunicação serial - RS232.** Disponível em: <http://www.professores.aedb.br/arlei/AEDB/Arquivos/rs232.pdf> - Acesso em: 20 Set. 2010.

CATÁLAGO. Datasheet Max 232 texas instruments. s/d

IDEV. **RS 232**. Disponível em: http://ldev.wordpress.com/2009/04/18/rs-232/ - acesso em: 20 set. 2010.

KENJO, T. **Stepping motors and their microprocessor controls**. 1. ed. Oxford University Press, 1986.

LYRA, Pablo Vinicius Apolinário. **Desenvolvimento de uma máquina fresadora CNC didática**. (2010). Disponível em: <http://alvarestech.com/temp/cnc/Fresadora%20CNC%20Did%E1tica.pdf> – Acesso em: 20 out 2010.

MATIAS, Juliano. **Encoders**. Disponível em: <http://www.mecatronicaatual.com.br/secoes/leitura/281> - Acesso em: 06 out 2010.

MESSIAS, Antonio Rogério. **Controle de motor de passo através da porta paralela**. Disponível em: <http://www.rogercom.com/pparalela/IntroMotorPasso.htm> - Acesso em: 02 out 2010.

MICROCHIP **Tecnology. Microchip.** Disponível em: http://www.microchip.com/ - Acesso em 20 set 2010.

NEWTON C. Braga. **Automação de redes**. Disponível em: <http://www.mecatronicaatual.com.br/secoes/leitura/333 / Automação - Redes> - Acesso em 20 out 2010.

OPTICAL Encoder Wheel Generator. Disponível em: <http://www.bushytails.net/~randyg/encoder/encoderwheel.html> - Acesso em: 06 out. 2010.

PEREIRA, Fábio. **Microcontroladores pic:** técnicas avançadas. 5. ed. São Paulo: Érica Ltda, 2007.

PEREIRA, Eduardo Pereira e LAGES, Walter Fetter. **Integração de sinais e dados**. Disponível em: <http://alvarestech.com/temp/simprebal/Relatorios_Tecnicos-Publicacoes-Dissertacoes/docs/cursos/Aula2-integracaosinais-apostila-cap3.pdf> - Acesso em: 20 set. 2010.

SITE. Disponível em: <www2. eletronica.org/artigos/outros/estudo-do-motor-de-passo-e-seu-controle-digital> - Acesso em: 06 out. 2010.

SOMESSARI, Samir Luiz. **Automação do processo de soldagem a laser (Nd:YAG) para confecção das sementes de iodo-125 utilizadas em braquiterapia**. Disponível em: <http://pelicano.ipen.br/PosG30/TextoCompleto/Samir%20Luiz%20Somessari_M.pdf> Acesso em: 20 out. 2010.

SOUZA, David, J. DE. **Desbravando o PIC** – Amplificado e atualizado para pic 16f628a. São Paulo: Editora Érica Ltda., 2007.

SOUZA, Marco Antonio. **Implementação de sistemas controladora de motor de passo em malha fechada utilizando tecnologia baseada em controlador digital de sinais.** Trabalho de conclusão. São Paulo: Escola de Engenharia São Carlos, 2007.

SOUZA, Vitor Amadeu. **Comunicação RS232 e RS485**. Disponível em: <http://www.cerne-tec.com.br/Comunica%E7%E3o232485.pdf> - Acesso em: 20 out. 2010.

TECNICAL English. **Stepper motor basics**. Disponível em: <www.sapiensman.com/ESDictionary/docs/d6.htm> - Acesso em: 06 out. 2010.

WIKIPEDIA. **Biblioteca padrão do C++.** Disponível em: <http://pt.wikipedia.org/wiki/Biblioteca_padr%C3%A3o_do_C%2B%2B> – Acesso em 18 set. 2010.