UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

ISRAEL BURIGO DALMOLIN

CONTROLE DE UM SISTEMA XY COM MOTORES DE PASSO POR MEIO DO ALGORITIMO DE BRESENHAM

CRICIUMA

2014

ISRAEL BURIGO DALMOLIN

CONTROLE DE UM SISTEMA XY COM MOTORES DE PASSO POR MEIO DO ALGORITIMO DE BRESENHAM

Trabalho de Conclusão de Curso, apresentado para obtenção do grau de Bacharel no curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC.

Orientador: Prof. Esp. Sérgio Coral

CRICIUMA

2014

ISRAEL BURIGO DALMOLIN

CONTROLE DE UM SISTEMA XY COM MOTORES DE PASSO POR MEIO DO ALGORITIMO DE BRESENHAM

Trabalho de Conclusão de Curso aprovado pela Banca Examinadora para obtenção do Grau de Bacharel, no Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC, com Linha de Pesquisa em Automação Industrial.

Criciúma, 24 de junho de 2014.

BANCA EXAMINADORA

Prof. Esp. Sérgio Coral - (UNESC) - Orientador

Prof. Esp. Ênio José Peruchi - (UNESC)

Prof. MEng. Evânio Ramos Nicoleit - (Instituição)

Dedico este trabalho aos colegas e professores, que foram essenciais para a conclusão do curso.

AGRADECIMENTOS

Agradeço primeiramente a minha família, pelo incentivo a educação, pelo apoio nas horas difíceis, bem como, o auxilio e a perseverança nos estudos.

Estendo o agradecimento aos professores que contribuíram com seus conhecimentos ao longo da formação do curso. Enfatizando o Prof. Sérgio Coral, pela orientação na produção do trabalho de conclusão de curso.

Aos colegas de classe, que durante o curso partilharam, das dúvidas e conhecimentos, elevando os níveis das discussões, contribuindo assim, para uma melhor apropriação dos conhecimentos científicos.

Agradeço também a minha namorada e meus amigos, que estavam presentes, por meio do apoio e da motivação, durante o período acadêmico.

“É a tecnologia que resolve problemas, e não a política”

Jacque Fresco

RESUMO

Este projeto descreve a confecção de um sistema de controle de automação, para controlar motores dispostos em forma de plano cartesiano, utilizando baixa tecnologia e algoritmos computacionais, afim de reduzir o custo e trazer facilidade de implementação, para empresas que estão entrando no ramo de automação. Para tal, é necessário compreender as etapas da automatização, desde a entrada de sinais por comparação, passagem pelo controle, até a atuação dos deslocamentos nos eixos. Os deslocamentos são executados por motores de passo que auxiliam nesse processo, e são controlados por sistemas eletrônicos dedicados, conhecidos como drivers. Estes drivers, por sua vez, são gerenciados por sistemas inteligentes, tal como os microcontroladores, componentes eletrônicos que podem ser programados. Os microcontroladores também fazem parte da recepção das informações entre um usuário e o sistema. Um dos problemas de um de um microcontrolador é operar em baixa velocidade de processamento, visando a busca por algoritmos mais eficazes, como o algoritmo de Bresenham, utilizado para traçar retas em formato de pixels, por meio de operações com números inteiros, ou lógica inteira. A confecção do driver se deu através de circuitos integrados que controlam da melhor forma os motores de passo, juntamente utilizando um microcontrolador para abstrair o acionamento destes circuitos, afim de facilitar o controle dos mesmos. Uma etapa seguinte, foi a objetivação da mesa, com o intuído de testar o sistema. Na confecção do controle, foi estipulado a utilização de protocolos de comunicação, para que um usuário final possa estar se comunicando com o sistema. Este protocolo, têm como objetivo receber pontos de um objeto a ser plotado no plano cartesiano. Ainda no controle, foi implementado o algoritmo de Bresenham, com a finalidade de acionar os motores de forma a executarem os passos, como se fossem pixels. Com a eletrônica projetada e programada, o controle do sistema foi contemplado, podendo ser testado plotando diversos objetos, de diferentes tamanhos. Para testar o desempenho do microcontrolador, foi implementado também um algoritmo para traçar retas, utilizando lógica flutuante. Uma comparação foi estabelecida entre os algoritmos implementados, e foi comprovado a eficácia da utilização do algoritmo de lógica inteira. Por fim, todas as etapas do projeto foram alcançadas. Ainda assim, este sistema pode ser melhorado, otimizando recursos de hardware e software, além da implementação de sistemas que possuem soluções gráficas eficientes, até a questão de aumentar a precisão dos motores de passo.

Palavras-chave: Motor de passo. Microcontrolador. Sistema XY. Automação industrial. Bresenham.

ABSTRACT

This project describes the fabrication of an automation system control, to control motors arranged in a Cartesian plane, using low technology and computer algorithms in order to reduce the cost and bring ease of implementation, for companies that are entering the field of automation. For this it is necessary to understand the stages of automation, since the input signals by comparison, the control passage until the performance of the displacement axes. The displacements are carried by stepper motors to assist in this process, and are controlled by dedicated electronics, known as drivers. These drivers, are managed by intelligent systems such as microcontrollers, electronic components that can be programmed. Microcontrollers are also part of the receipt of information between a user and the system. One problem with a microcontroller is operating in a low processing speed, in order to search for more efficient algorithms such as Bresenham algorithm, used to draw lines on the pixel format using integer operations, or integer logic. The making of the driver was through integrated circuits that control the better way stepper motors along using a microcontroller to abstract the activation of these circuits in order to facilitate the control. A next step was the objectification of the Cartesian table, with the intuited testing the system. In making the control, it was stipulated the use of communication protocols so that an end user may be communicating with the system. This protocol, aim to get points of an object to be plotted on the Cartesian plane. Also in the control, the Bresenham algorithm is implemented in order to drive the motors in order to perform the steps as if they were pixels. With the designed and programmed electronics, control system was contemplated, and can be tested by plotting various objects of different sizes. To test the performance of the microcontroller, has also implemented an algorithm to draw straight lines, using floating logic. A comparison was made between the algorithms implemented, and it has been proven the effectiveness of using the entire logic algorithm. Finally, all milestones were achieved. Still, this system can be improved by optimizing hardware resources and software, in addition to implementing systems that have efficient graphics solutions until the issue of increasing the accuracy of stepper motors.

Key-words: Stepper motor. Microcontroller. XY system. Bresenham. Industrial automation.

LISTA DE ILUSTRAÇÕES

Figura 1 - Fluxo do sistema automatizado 19
Figura 2 - Sensor retro reflexivo em funcionamento.....22
Figura 3 - Sensor retro reflexivo em funcionamento.....22
Figura 4 - Sensor óptico por barreira em funcionamento 23
Figura 5 - Sensor óptico por Barreira em funcionamento.....23
Figura 6 - Circulação do fluxo magnético em um núcleo.....24
Figura 7 - Analogia do motor unipolar 26
Figura 8 - Analogia do motor bipolar 26
Figura 9 - Funcionamento motor de quatro passos unipolar e bipolar 27
Figura 10 - Concepção interna de um motor de relutância variável 28
Figura 11 - Concepção interna de um motor de passo hibrido.....29
Figura 12 - Esquema de funcionamento da Ponte H.....30
Figura 13 - Esquema de funcionamento da Ponte H dupla.....31
Figura 14 – Esquema do acionamento de um motor de passo unipolar 33
Figura 15 - Ponte H de um Motor de passo unipolar com Transistores 35
Figura 16 - Circuito Integrado PBL3770A.....36
Figura 17 - Sequência de operação (PBL3770A).....38
Figura 18 - Aplicação típica do *driver* de motor de passo PBL3770A 38
Figura 19 – PIC16F870 41
Figura 20 - Método principal de um microcontrolador escrito em C 41
Figura 21 – Acionamento do pino RB0.....43
Figura 22 – Codificação no HI-TECH C 43
Figura 23 - Configuration Bits.....44
Figura 24 – Esquema Arduino UNO.....46
Figura 25 – Principais métodos Arduino e MPLAB.....47
Figura 26 – Acionamento Intermitente do pino 13.....47
Figura 27 – Superfície de *Pixels*.....49
Figura 28 – Algoritmo de Bresenham em C 51
Figura 29 – Modelo do *drive*r de controle dos motores de passo.....56
Figura 30 – Controle do *drive*r PBL 3770 com PIC 57
Figura 31 – Inicio do firmware de controle do *drive* PBL 3770 60

Figura 32 – Codificação em C do acionamento em *full step* do motor X.....60
Figura 33 – Simulação do acionamento do motor X em full-step. 61
Figura 34 – Inversão de sentido de giro do motor X.....61
Figura 35 – Gravador de PIC PicKit 2 61
Figura 36 - Layout do *drive*r PBL3770.....62
Figura 37 – *Drive*r PBL 3770.....63
Figura 38 – Montagem mesa.....64
Figura 39 – Trilho e carrinho HIWIN.....64
Figura 40 – Montagem mesa.....65
Figura 41 – Motor de passo TPP17.....65
Figura 42 – Modelo mesa XY completo 66
Figura 43 – Mesa XY.....66
Figura 44 – Envio de um objeto.....69
Figura 45 – Comando SPEED.....69
Figura 46 – Comandos sem parâmetros 69
Figura 47 – Calculando Checksum 70
Figura 48 – Comunicação Java x Arduino.....71
Figura 49 – Controle de passos 73
Figura 50 – Solenóide 74
Figura 51 – Modelo 3D *driver* motor unipolar 75

LISTA DE TABELAS

Tabela 1 - Sequência de acionamentos das chaves de uma ponte H dupla.....32
Tabela 2 - Acionamento das chaves de um motor de passo em full-step 33
Tabela 3 - Acionamento das chaves de um motor de passo em half-step 34
Tabela 4 - Níveis de Corrente em relação às entradas I0 e I1 36
Tabela 5 - Descrições dos pinos do PIC 59
Tabela 6 – Descrição dos bytes do pacote. 67
Tabela 7 – Lista de comandos 68
Tabela 8 – Eficácia do algoritmo de Bresenham em relação a um de ponto flutuante 76
Tabela 9 – Recursos de hardware 78

LISTA DE ABREVIATURAS E SIGLAS

CA	Corrente Alternada
CC	Corrente Contínua
CI	Circuito Integrado
CLP	Controlador Lógico Programável
EEPROM	*Electrically-Erasable Programmable Read-Only Memory*
GHz	GigaHertz
GND	Ground
IDE	*Integrated Development Environment*
MCU	Microcontrolador
MHz	MegaHertz
PIC	*Programmable Interface Controller*
PWM	*Pulse Width Modulation*
RISC	*Reduced Instruction Set Computer*
ULA	Unidade lógica e Aritmética
USB	*Universal Serial Bus*
VCC	Volts em Corrente Contínua

SUMÁRIO

1 INTRODUÇÃO 14
1.1 OBJETIVO GERAL 16
1.2 OBJETIVOS ESPECÍFICOS 16
1.3 JUSTIFICATIVA 16
1.4 ESTRUTURA DO TRABALHO 17
2 AUTOMAÇÃO 18
2.1 AUTOMAÇÃO INDUSTRIAL 18
2.1.1 Malha fechada e malha aberta 20
2.2 SENSORIAMENTO 21
2.2.1 Sensores analógicos 21
2.2.2 Sensores digitais 21
2.2.3 Sensores ópticos 21
2.3 ATUADORES 23
2.3.1 Motores elétricos 24
2.3.2 Motores de passo 25
2.3.2.1 Motor de passo de relutância variável 27
2.3.2.2 Motor de passo de imã permanente 28
2.3.2.3 Motor de passo híbrido 29
2.4 COMPARAÇÃO E CONTROLE 30
2.4.1 Acionamento de motor CC 30
2.4.2 Acionamento de motor de passo bipolar 31
2.4.3 Acionamento de motor de passo unipolar 32
2.4.4 Chaveamento com transistores 34
2.4.5 Circuitos integrados dedicados 35
2.4.5.1 PBL 3770 A 35
2.4.6 Microcontroladores 39
2.4.6.1 Microcontroladores PIC 40
2.4.6.2 Arduino 45
2.5 ALGORITMO DE BRESENHAM 48
2.6 COMUNICAÇÂO DE DADOS 52
2.6.1 Protocolos de comunicação 53

2.7 TRABALHOS CORRELATOS 53
2.7.1 Controle de mesa xy: utilizando motor de passo 53
2.7.2 Algorithm for computer control of a digital plotter 54
2.7.3 Algoritmo de Bresenham: o uso microcontroladores para traçar retas em LCD's 54
3 SISTEMA XY 55
3.1 MONTAGEM DO SISTEMA 55
3.1.1 Desenvolvimento do *drive*r de motor de passo 55
3.1.1.1 Firmware do *drive*r 59
3.1.2 Mesa XY mecânica 63
3.1.3 Programação no Arduino 67
3.1.3.1 Protocolo de comunicação 67
3.1.3.2 Algoritmo de Bresenham 72
3.1.3.3 Controle de pulsos 73
3.1.4 Testes e correções 74
3.1.4.1 Comparação com algoritmo flutuante para plotagem de retas 76
3.2 RESULTADOS OBTIDOS 79
4 CONCLUSÃO 80
REFERÊNCIAS 82
APÊNDICE(S) 85

1 INTRODUÇÃO

Segundo Souza (2005) a maioria das empresas do ramo industrial utilizam em seus controles de processos sistemas mecânicos, eletromecânicos e computacionais, como operadores destes controles. O aumento da qualidade dos produtos fabricados, redução de perdas em matérias primas e redução do tempo de fabricação são alguns dos fatores que levam uma empresa a automatizar os seus processos (FIGUEIREDO, 2010).

Um dos meios de automatização é a utilização de sistemas com motores elétricos, pois são capazes de transportar objetos por esteiras, conseguem produzir esforços repetitivos a longo prazo e aceleram o processo de produção. Trabalhos como acabamento e corte em metais necessitam de motores com precisão. Para este tipo de trabalho geralmente são utilizados motores de passo que de acordo com Queiroz (2002) são dispositivos que convertem pulsos elétricos em movimentos rotativos, possuindo três estágios: parado, ativado com o eixo do motor travado ou girando em etapas.

Para controlar os motores de passo são necessários sistema eletrônicos inteligentes, como por exemplos os microcontroladores. Microcontrolador é um pequeno dispositivo eletrônico, que possui uma inteligência programável. Entretanto os microcontroladores não conseguem fornecer a intensidade de corrente elétrica que os motores exigem. Para abordar este problema é necessário o desenvolvimento de hardwares específicos para a situação. Este tipo de hardware é conhecido como *driver* (SOUZA, 2007)

Os *drivers* devem ser apropriados para as condições de trabalho que os motores exigirão. Um conceito simples é que para se controlar somente um motor, o microcontrolador deve produzir pulsos elétricos, tais como em lógica digital, zeros e uns. O *driver* interpretará estes pulsos e irá girar o motor executando os passos para cada pulso interpretado. Um passo é um giro em um ângulo definido pelo fabricante do motor (BRITES, 2008).

Em sistemas com dois motores de passo, geralmente um motor com eixo horizontal (x) e outro vertical (y), ambos em um plano, como por exemplo, Controle Numérico Computadorizado (CNC), uma máquina para modelagem de peças mecânicas em metais que movimenta uma broca no eixos para a modelagem do

material, existe a necessidade de estabelecer uma sincronia entre ambos os motores, ou seja, para o sistema andar na horizontal (eixo x) somente um motor se deslocará, o mesmo é válido para andar na vertical (eixo y). Agora se o deslocamento for na transversal (eixos x e y) ambos os motores serão controlados. O desafio é controlá-los entre o transversal-horizontal ou transversal-vertical, exigindo soluções matemáticas para aplicar a sincronia. O movimento nestes meios parece assíncrono, pois os motores se deslocarão de forma distinta, e ao traçar uma reta, um se deslocará mais que o outro, entretanto o sistema de controle gerenciará os passos a serem interpretados pelo *drive*r, compensando a inclinação da reta projetada tornando os motores dependentes entre si.

Na prática, as empresas de automação industrial possuem seus sistemas específicos para este tipo de controle. O nível de precisão que os motores de passo exercem, o sistema de gerenciamento do controle feito por um engenheiro e o hardware envolvido, são métodos que posteriormente agregam valor ao produto final, tornando o conhecimento, as tecnologias envolvidas e o custo, difíceis para a aquisição de empresas que estão iniciando na área de automação.

Em análise, a ciência da computação conhece um problema parecido, no qual, trata-se da projeção de *pixels* em formato de retas. Determinado por coordenadas inicial e final, o problema consiste em projetar os *pixels* na tela em formato inclinado de forma rápida. A solução para este problema computacional se dispõe na área de computação gráfica, sendo alcançado por algoritmos.

Segundo Cormen et al (2012) os procedimentos computacionais bem definidos que recebem um conjunto de valores como entrada e retornam valores como saída, são considerados algoritmos. Também pode-se considerar como ferramentas para a resolução de problemas computacionais específicos.

Um destes algoritmos, é o de Bresenham, que de acordo com Moro (2009), utiliza da aritmética com números inteiros como base para acender pixels na tela, determinando quais *pixels* serão destacados para atender o grau de inclinação de uma reta.

1.1 OBJETIVO GERAL

Disponibilizar um sistema xy com motores de passo controlados pelo algoritmo de Bresenham.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos são:

a) projetar um hardware específico para controlar os motores de passo atendendo as requisições dos motores a serem usados, como corrente e tensão;
b) projetar uma mesa mecânica com trilhos, polias e correias para a movimentação nos eixos x e y;
c) controlar os motores de passo por meio dos microcontroladores;
d) desenvolver um protocolo de comunicação para um software controlador;
e) aplicar o algoritmo de Bresenham ao sistema para que possibilite a projeção de percursos em retas de forma rápida.

1.3 JUSTIFICATIVA

Os motores de passo, de acordo com Queiroz (2002), são dispositivos de alta precisão, confiáveis e de fácil controle. Esses motores têm sua aplicação em diversas áreas como, informática e robótica. Grande parte destes motores são introduzidos na automação industrial devido seu torque, porém exigem hardwares com maior robustez.

Ao contrário da maioria dos motores, motores de Corrente Contínua (CC), Corrente Alternada (CA) e servo motores, um motor de passo não se destaca por sua força e velocidade, mas pela precisão que seus movimentos podem exercer (BRITES, 2008).

Para controlar um sistema xy, as empresas de automação industrial, optam pelos motores de passo, pois em relação aos outros têm precisão, não tão boa quanto a um servo, porém possuem custo mais baixo e fáceis de controlar.

Este sistema deve possuir controle eletrônico, para ser manipulado por um usuário, tornando a máquina, o sistema xy, uma ferramenta de auxílio a produção. O controle deve gerenciar os motores e para isso é necessário sincronia. Alguns sistemas como máquinas CNC, utilizam logaritmos, funções exponenciais e somatórios para gerenciar esta sincronia, exigindo avançado conhecimento em programação e hardware, por parte do desenvolvedor do sistema.

Uma possível analogia é a percepção de visualizar o sistema como se fosse um monitor, os *pixels* são as coordenadas, possuindo ponto de referência (0,0) no canto superior esquerdo e sendo mapeado como uma matriz. Assim, compreende-se como os motores devem reagir ao traçar uma reta de um ponto a outro, imagina-se que os *pixels* acenderão para cada deslocamento ao próximo *pixel,* mas para um deslocamento inclinado, por exemplo em 30º, a reta poderá passar entre dois *pixels* e será difícil a visualização por não saber qual pixel será acendido. Tal analogia pode ser aplicada ao mundo físico, porém deve-se satisfazer estas inclinações. Por meio das soluções disponibilizadas pela computação gráfica, é possível manipular os passos dos motores como se fossem *pixels* com uso de algoritmos.

O algoritmo de Bresenham implementará a sincronia entre os motores, que ao contrário de outros algoritmos, como os implementados em máquinas CNC, não necessita de avançados recursos tecnológicos, tornando o sistema viável para a aquisição de empresas que estão iniciando no ramo de automação industrial.

1.4 ESTRUTURA DO TRABALHO

O primeiro capítulo apresenta características sobre o conceito de automação, de forma que, ao particularizar seus processos, se consegue compreender as etapas que contemplam este conceito. Estas particularidades, por sua vez, são descritas de forma mais específica, elevando a capacidade de entendimento sobre cada processo automatizado, para a direção do projeto.

O capítulo seguinte, traz a confecção do sistema xy, que foi descrita de forma a compreender e contemplar as metas para a objetivação do projeto. Também descreve testes e correções, que por sua vez garantiram uma melhoria na execução dos objetivos.

2 AUTOMAÇÃO

Historicamente o processo de automatização, era feito por meio da mecanização de tarefas que necessitavam esforços repetitivos, ou para compensar trabalhos árduos. A invenção da roda é um exemplo de automatização que facilitava trabalho de transporte. O foco era sempre simplificar o trabalho do homem, reduzindo o esforço físico (SILVEIRA; SANTOS, 1998).

Em 1960, a automação é sugerida para processos que recebem controle computacional. De acordo com Moraes e Castrucci (2007), automação é qualquer conjunto de sistemas, sustentado por processos computacionais, cujo o objetivo é substituir o trabalho humano, em beneficio a qualidade dos produtos fabricados, redução das perdas de matérias primas e segurança das pessoas.

Com finalidade em diversas áreas, a automação surge para facilitar várias atividades humanas. Nas áreas residências possuem sistemas bem conhecidos como, maquinas de lavar louças, portões eletrônicos, micro-ondas, entre outros. Áreas públicas, e ou empresariais, caixas eletrônicos, controle de metrôs, semáforos, controle de pontos, etc. E principalmente nas áreas industriais, sistemas automáticos de transporte, robótica, controle de qualidade, sistemas de segurança, processos de produção, entre outros (MARTINS, 2007).

2.1 AUTOMAÇÃO INDUSTRIAL

Segundo Martins (2007), Rosário (2005) e Souza (2005), para que aconteça a automação industrial, é necessário o agrupamento de três áreas tecnológicas. Mecânica, com proposito de trabalhar diretamente com a matéria prima. Eletroeletrônica, sob a intenção de adaptar dispositivos elétricos para suprir o trabalho humano, por meio de motores, e monitorar informações como temperatura e pressão, por meio de sensores. E a informática, possuindo o objetivo de receber dados monitorados e gerencia-los, repassando as informações produzidas a todos os níveis de uma empresa.

Para atingir melhores resultados em qualidade, é necessário um controle de processo produtivo, ou seja, incluso na produção existem conjuntos de variáveis que ao decorrer do processo, alteram a qualidade do produto final. O controle

produtivo tem como objetivo monitorar estas variáveis em tempo real e gerenciar o sistema de produção para satisfazer o processo qualitativo. Por meio da automação industrial os padrões de qualidade são enriquecidos (ALVES, 2005).

Vantagens como, aumentar a produtividade, reduzir os custos do trabalho, minimizar os efeitos da falta de trabalhadores, reduzir ou eliminar as rotinas manuais e das tarefas administrativas, aumentar a segurança do trabalhador, melhorar a qualidade do produto, diminuir o tempo de produção, realizar processos que não podem ser executados manualmente e evitar o auto custo da não automação, são razões para que as empresas justifiquem a incorporação da automação (FIGUEIREDO, 2010; GROOVE, 2011).

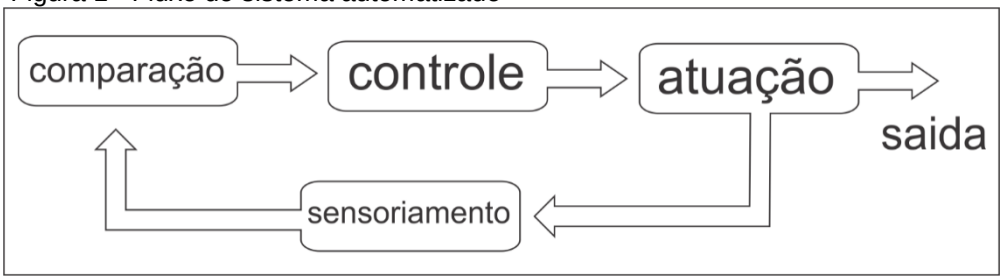
Logo, a automação tem papel fundamental nas indústrias, garantindo a melhoria dos processos produtivos e possibilitando a competitividade no mercado onde atua (MARTINS, 2007).

Ainda de acordo com Martins (2007), os sistemas automatizados podem ser divididos, em subsistemas, para serem melhores compreendidos (figura 1). Por sua vez estes subsistemas possuem propriedades similares e de simples entendimento, divididos em três elementos básicos:

a) sensoriamento: dispositivos que monitoram o meio físico, tal como, pressão, temperatura, velocidade, entre outros;
b) comparação e controle: compara as variáveis monitoradas com valores preestabelecidos e toma decisão de quando controlar o sistema.
c) atuação: os atuadores são gerenciados pelo controle, são os meios que processam diretamente com o material, como motores, e prensas.

Porém Alves (2005) classifica um item a mais para os sistemas automatizados. O programa, que objetiva controlar as interações entre os dispositivos e consegue sustentar todas as informações sobre o processo.

Figura 1 - Fluxo do sistema automatizado



Fonte: Do autor.

Um exemplo, Martins (2007), seria o controle automatizado da temperatura da água dentro de um aquário (processo). O controlador estaria ligado ao termômetro (sensor) com ajuste manual para a temperatura desejada. Este ajuste estaria em constante comparação ao mercúrio dentro do termômetro, permitindo ligar ou desligar uma resistência (atuador) dentro do aquário, para aumentar ou diminuir a temperatura e satisfazer as condições de temperatura ajustadas, mediante a variação do termômetro.

O computador (programa) seria acoplado diante de todas as variáveis que o processo contém. Pode-se por meio do computador inserir pré-ajustes de temperatura, guardar históricos de variação e substituir o controlador em relação ao valor de ajuste, tornando-o somente intercalador entre programa e processo (ALVES, 2005).

2.1.1 Malha fechada e malha aberta

O exemplo do aquário é um processo em malha fechada, pois possui realimentação na saída, onde a medição de temperatura deve ser sempre equilibrada com o ajuste feito no controle. Estes sistemas de malha fechada são aplicados em situações onde deve ocorrer constante ajuste de produção, tal como, linha de produção cerâmica, onde a qualidade dos pisos e azulejos é constantemente monitorada na saída para o reajuste nas condições de entrada como: temperatura, quantidade de esmalte, entre outros (MARTINS, 2007).

Em impasse os sistemas de malha aberta não contêm essa realimentação, não agregando o controle de saída. Porém não o classifica como pior ou melhor, em relação à malha fechada. Por sua vez as aplicações nestes sistemas também são eficientes, tais como, maquinas de lavar roupa ou lava louças que não habilitam as condições para saber se a roupa ou louça esta seca ou lavada. Então possuem controle temporizado e manual, deixando somente o trabalho árduo para a máquina realizar (SILVA, 2007).

2.2 SENSORIAMENTO

As formas de energia do ambiente como, luz, pressão, velocidade, corrente, temperatura, posição, etc. são variáveis que podem ser capturadas através de um sensor e possivelmente monitoradas. Contudo um sensor nem sempre fornece os atributos necessários para serem usados em sistemas de controle. Para satisfazer esse obstáculo o sinal produzido pelo sensor é ampliado ou convertido por meio de outros circuitos, assim possibilitando a leitura por parte do controlador (THOMAZINI; ALBUQUERQUE, 2005).

2.2.1 Sensores analógicos

Resulta variação em sua saída ao longo do tempo, estando sempre em um faixa de operação. Pressão, temperatura, umidade, luminosidade, etc. são grandezas que assumem mudança ao decorrer do tempo, portando só existirão medidas com exatidão por meio de circuitos não digitais (CAPELLI, 2000).

2.2.2 Sensores digitais

Intrinsecamente captura dados semelhantes aos sensores analógicos, porém descarta medidas voláteis. Assumindo somente dois estados, ligado ou desligado, informa ao controle saídas discretas (zeros e uns). Comumente utilizado para contagem, verificar passagem de objetos, entre outros (CAPELLI, 2000).

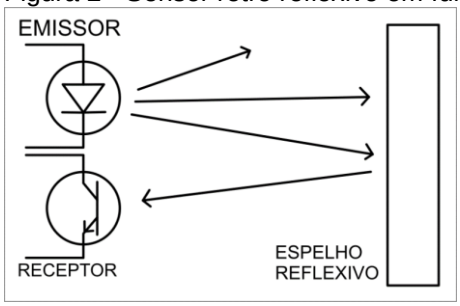
2.2.3 Sensores ópticos

São sensores digitais que trabalham por meio de emissores e receptores de luz, conseguem detectar passagem ou proximidade de materiais sem que exista contato mecânico. O princípio de funcionamento é baseado na emissão de luz do emissor para o receptor com intensidade suficiente para que o receptor comute, acionando a saída (THOMAZINI; ALBUQUERQUE, 2005).

Os sensores ópticos podem ser dispostos de diferentes formas, tais como.

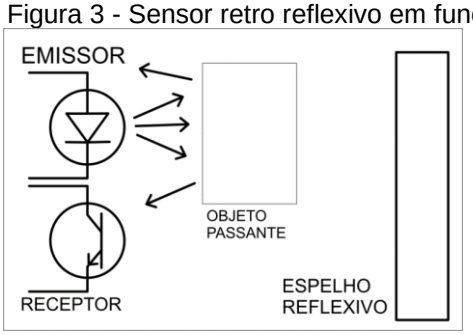
a) sensores ópticos por retro reflexão (figura 2): acoplados juntos e em mesmo sentido, o emissor transmite o feixe de luz a um material reflexivo, assim retornando o feixe ao receptor. O funcionamento (figura 3) consiste em que um objeto atravesse na frente do feixe interrompendo a reflexão, porém se o objeto for de material pouco reflexivo, alguma parte do feixe pode retornar e tornará a leitura duvidosa;

Figura 2 - Sensor retro reflexivo em funcionamento



Fonte: Do autor.

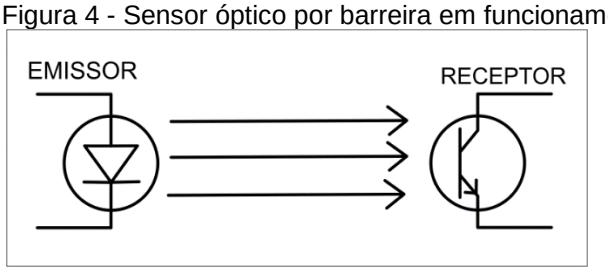
Figura 3 - Sensor retro reflexivo em funcionamento



Fonte: Do autor.

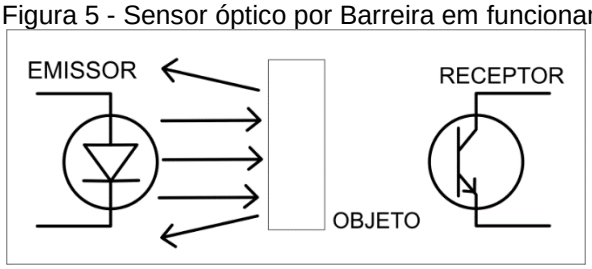
b) sensores ópticos por barreira (figura 4): este sensor é disposto para que forme uma barreira de luz, ou seja, o emissor e o transmissor são separados e alinhados um de frente ao outro. Compensando a retro reflexão, só é detectado um objeto se a barreira for totalmente interrompida, garantindo uma melhor leitura (figura 5).

Figura 4 - Sensor óptico por barreira em funcionamento



Fonte: Do autor.

Figura 5 - Sensor óptico por Barreira em funcionamento



Fonte: Do autor.

Existe uma variedade de sensores para aplicações industriais, comerciais e residenciais como, sensores indutivos que tem finalidade em detectar objetos metálicos, sensores infravermelhos que são utilizados como acionadores de alarmes, sensores ultrassônicos que estipulam distância de objetos, sensores de campo magnético utilizados em aberturas de portas e janelas, entre outros. No processo automatizado os sensores entram como monitores dos acontecimentos na matéria a ser trabalhada, e estão ligados diretamente aos atuadores, construindo um sistema de malha fechada. Existem exemplos como, sensores de posicionamento de uma cabeça, contagem de peças, fim de curso, monitoramento de temperatura e pressão, entre outros. O sensoriamento é utilizado como meio de reorganizar o processo automatizado, respondendo novamente aos atuadores.

2.3 ATUADORES

Grande parte dos sistemas automatizados é dependente de forças mecânicas que interagem diretamente com a produção fabril. Estas forças podem ser derivadas dos motores, que por sua vez, são controlados por sistemas hidráulicos, pneumáticos ou elétricos (BIM, 2009).

2.3.1 Motores elétricos

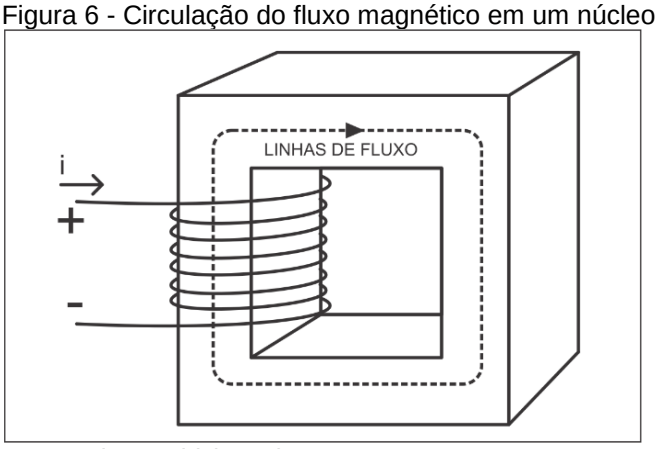
Os motores elétricos têm maior facilidade de uso no quesito controle, pois por meio da eletroeletrônica são mais fáceis comandados. Capazes de transportar objetos por esteiras, conseguir produzir esforços repetitivos em longo prazo e acelerar o processo de produção, entram como atuadores no processo automatizado (QUEIROZ, 2002).

O funcionamento dos motores elétricos é baseado no princípio do eletromagnetismo de James Maxwell, que por meio dos seus estudos definiu que o campo magnético é resultado da variação do campo elétrico, reciprocamente verdadeira (TOLSTOY, 1986).

Ainda de acordo com Tolstoy(1986), o campo magnético tem direção que é definido pelo sentido da corrente que circula em um condutor. Por meio das leis de Maxwell, foi possível definir a direção que o campo está, e posteriormente aplicar estes conceitos as maquinas elétricas rotativas, dando origem aos motores elétricos.

A força eletromagnética aplicada a um condutor em espiral gera um campo magnético variável e incontrolável. Para controlar este campo se aplica um núcleo de ferro ou ferrite fazendo com que o campo magnético circule dentro do núcleo, conforme a figura 6 (FITZGERALD, 2006).

Figura 6 - Circulação do fluxo magnético em um núcleo



Fonte: Fitzgerald (2006).

Os motores elétricos, Fitzgerald (2006), são dispositivos que baseados em equações de Maxwell convertem energia elétrica, por meio do eletromagnetismo, em mecânica, ou seja, seu funcionamento é baseado em aplicar corrente elétrica em

suas bobinas internas, para que por meio da força magnética induza a rotação no eixo do motor.

Divididos em categorias para diferentes aplicações, possuindo características únicas. Os principais tipos de motores são os motores de Corrente Continua (motores CC) e de Corrente Alternada (motores CA). Os demais como, motores de passo, servo motores, motores *brushless*, motores síncronos, etc. são variantes destes motores, porém para aplicações mais específicas (CARVALHO, 2011).

Os motores usados principalmente nas indústrias têm a finalidade de movimentar linhas de produção, como transporte de pisos e azulejos sobre esteiras, prensar materiais de forma ágil e em alguns casos posicionar a matéria prima em um local específico. Para os dois primeiros exemplos são utilizados normalmente motores CA e CC que objetivam força e velocidade no sistema automatizado, porém para a terceira aplicação estes motores necessitariam de um controle robusto e de custo elevado, utilizaria possivelmente sensores de posicionamento para a realimentação no processo de automação, o que acarretaria na desaceleração da linha de produção. Para a solução deste problema desenvolveu-se motores que possuem posicionamento em malha aberta, não necessitando de realimentação, tais motores são conhecidos como motores de passo.

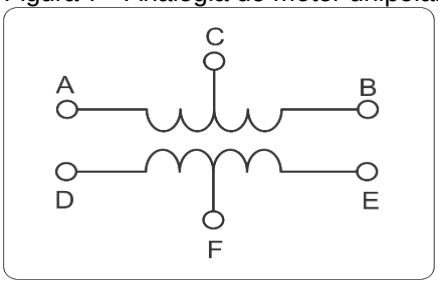
2.3.2 Motores de passo

São motores projetados especificamente para aplicações onde há necessidade de precisão. Podem ser aliados aos sistemas eletrônicos digitais com maior facilidade que outros, e assim dando origem a um controle mais simples. Como por exemplo, os motores de impressoras comuns, onde existem motores para alinhar o papel e para posicionar a cabeça de impressão em locais específicos, assim resultando em uma impressão de melhor qualidade (FITZGERALD, 2006).

Normalmente são utilizados nas indústrias motores unipolares de duas a quatro bobinas (figura 7), estes possuem uma derivação central nos enrolamentos das bobinas (C e F), viabilizando a possibilidade de inverter o campo magnético sem a necessidade de inverter o sentido da corrente nas mesmas, ou seja, ao aplicar um corrente elétrica de C para B, o fluxo magnético será para a direita, aplicando uma

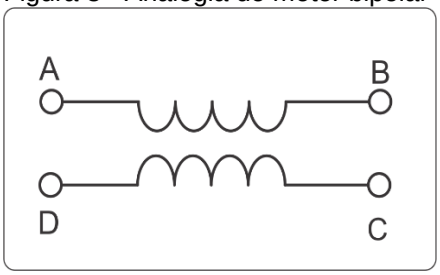
corrente de C para A tem-se o fluxo para a esquerda sem que haja inversão da corrente entre A e B. Outro motor é o bipolar (figura 8), que contém duas bobinas internas apresentando como vantagem um maior torque, além de possuir menor tamanho em relação à proporção do motor, pois energiza toda a bobina, porém exige um controle mais sofisticado, necessitando da inversão da corrente a cada passo executado, em comparação ao unipolar é obrigatório inverter a corrente entre A e B, e este processo eletronicamente não é simples (QUEIROZ, 2002).

Figura 7 - Analogia do motor unipolar



Fonte: Do autor.

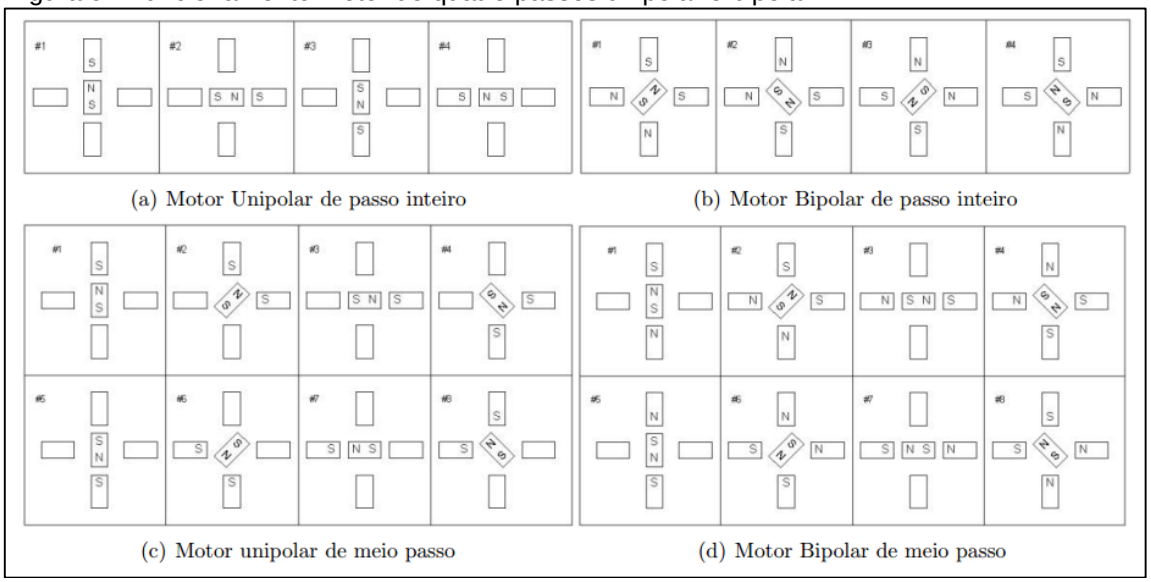
Figura 8 - Analogia do motor bipolar



Fonte: Do autor.

Uma compreensão do funcionamento deste motor é conhecido como motor de quatro passos (figura 9), onde se consiste em utilizar um imã, como se fosse o rotor cercado por bobinas elétricas ou solenóides. Ao energizar uma bobina o imã alinhará seu campo magnético com o campo magnético produzido pela mesma, o passo seguinte é com intuito de energizar a próxima solenóide, porém desenergizando a primeira. Este modelo de energizar uma bobina de cada vez é conhecido por *full-step*, passo inteiro. Pode-se também energizar duas bobinas de cada vez, estas devem estar em sequência para que o alinhamento do imã fique entre ambas, então na análise, se consegue combinar as duas formas de acionamento para duplicar a quantidade de passos do motor, este processo de duplicação é chamado de *half-step*, meio passo (BRITES e SANTOS, 2008).

Figura 9 - Funcionamento motor de quatro passos unipolar e bipolar



Fonte: Brites e Santos (2008).

De acordo com Carvalho (2011) e Queiroz (2002), existem três tipos de motores de passo: Motor de passo de relutância variável, Imã permanente e motor de passo hibrido.

Carvalho (2011) ainda cita aspectos comuns entre os tipos de motores, tais como:

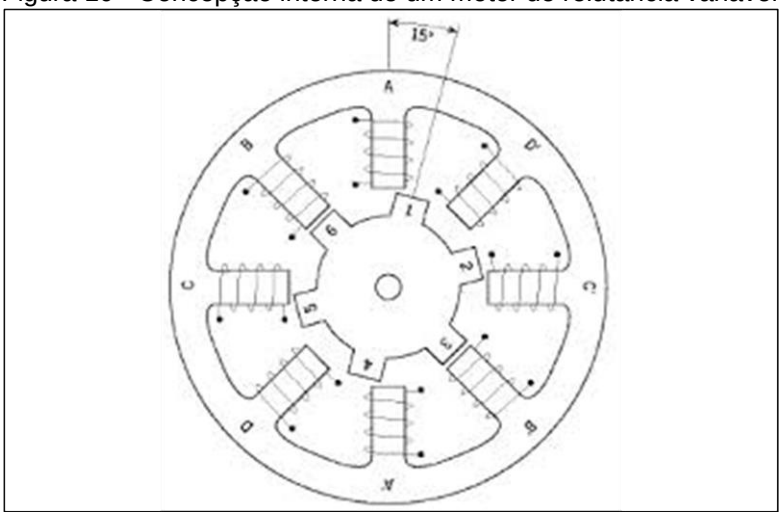
a) o motor consegue girar em baixas velocidades, ainda que este possua carga ligada ao eixo;
b) existe uma proporção entre a velocidade de giro e a frequência de pulsos lógicos aplicados ao motor;
c) o passo determina a precisão do motor. O passo é um giro angular que o motor pode deslocar determinado pelo fabricante do motor.

2.3.2.1 Motor de passo de relutância variável

O rotor, eixo, deste motor possui várias polaridades, uma série de empilhamentos deslocados axialmente, ou seja, dentro do motor conforme a figura 10 existem dentes de material metálico que são energizados por bobinas e geram campo magnético. Normalmente estes motores possuem passo de 5 a 15 graus com torque estático nulo, que facilita na retirada da inércia estática para executar os passos.

O funcionamento deste tipo de motor pode ser compreendido (figura 10), energizando a fase A que acarretará no alinhamento de dois dentes do rotor (1 e 4) com dois dentes do estator, carcaça, por meio do magnetismo. O passo seguinte será para a fase B, para que aconteça o passo angular é necessário desenergizar a fase A e energizar a fase B, novamente ocorrendo o alinhamento dos dentes do rotor. O mesmo é valido para a fase C, seguindo para a fase D, até energizar a fase A novamente, assim rotacionando o eixo do motor (QUEIROZ, 2002).

Figura 10 - Concepção interna de um motor de relutância variável



Fonte: Queiroz (2002).

2.3.2.2 Motor de passo de imã permanente

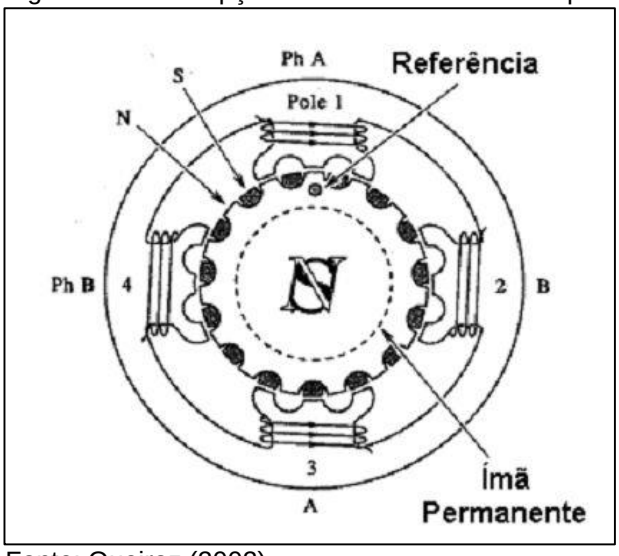
Uma das características deste motor é não possuir dentes no rotor, mas sim imãs permanentes o que acarreta em maior torque. Geralmente com níveis baixos de passo, 45° e 90°, então se entende que gira a velocidades baixas, porém com maior torque que o de relutância variável. Por consequência do seu imã interno, este motor possui torque alto quando o rotor está inativo, levando em consideração este fator, é necessária uma maior corrente nas bobinas do estator para retirar o rotor da inércia e causar o passo (BRITES e SANTOS, 2008).

2.3.2.3 Motor de passo híbrido

Este tipo de motor possui características de ambos os motores anteriores como: alto torque, não apresenta toque estático nulo e opera em velocidades altas e baixas sem perder a força. Possuindo ângulos de passos bem pequenos entre 0,9 a 3,6 graus (100-400 passos por volta). Em seu estator existem pequenos dentes como no motor de relutância variável e um rotor com imã permanente com dentes para facilitar o alinhamento por meio das linhas magnéticas e aumentar a precisão. (BRITES e SANTOS, 2008).

Para demonstrar seu funcionamento, primeiramente entende-se que o número de dentes do rotor se distingue do número de dentes do estator, variando de um a três dentes de diferença. Por exemplo, a posição angular dos dentes do rotor é de 10° por dente, enquanto no rotor são 12° por dente. O imã do rotor não possui os dentes alinhados em cada polo, demonstrados na figura 11 pelas letras N e S, provendo um melhor caminho para guiar fluxo magnético dentro do motor. Com estes conceitos, compreendesse que ao alimentar uma bobina os dentes do rotor tenderão a se alinhar com os dentes do estator que estão sendo energizados, isso acarretara em um pequeno desalinhamento dos demais dentes em relação ao estator. Ao alimentar uma nova bobina, esta pequena diferença dos dentes tenderá a se alinhar com os dentes da nova bobina energizada, executando um pequeno passo (CARVALHO, 2011).

Figura 11 - Concepção interna de um motor de passo hibrido



Fonte: Queiroz (2002).

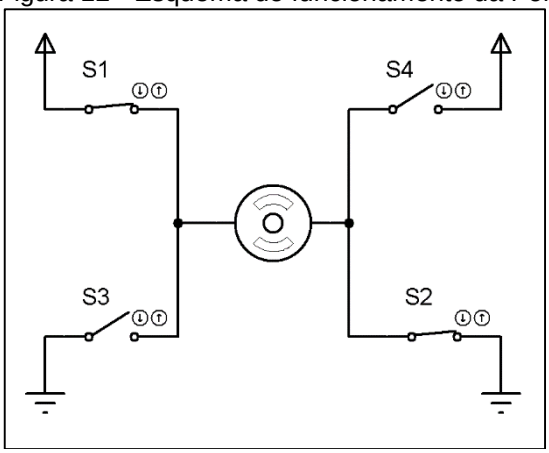
2.4 COMPARAÇÃO E CONTROLE

Uma vez compreendidos os motores de passo, define-se como será seu acionamento. Como é de facilidade e integração eletrônica, existem sistemas que gerenciam o acionamento das correntes em cada fase do motor, tornando-o um dispositivo de fácil aplicação. Os sistemas mais encontrados são os que acionam motores unipolares, que como já visto não há intuito de inverter o sentido da corrente em cada fase do motor, o passo pode ser executado por meio de sistemas lógicos, somente necessitando componentes para o acionamento, ligados diretamente as bobinas. Estes sistemas lógicos controlarão os componentes de acionamento do motor facilitando assim a compreensão eletrônica da aplicação.

2.4.1 Acionamento de motor CC

Um circuito de bastante utilidade para o acionamento de motores CC é conhecido como Ponte H, consistindo em disponibilizar chaves eletrônicas que montadas em um esquema, assumem a forma de um H. O circuito possui quatro chaves mostradas na figura 12 (S1, S2, S3, S4), que são acionadas de forma alternada, aciona-se S1 e S2 e posteriormente S3 e S4. Para cada acionamento das chaves ocorrera a inversão da corrente na bobina do motor, alternando o sentido do fluxo magnético, resultando na inversão do sentido de giro do motor (BRITES e SANTOS, 2008).

Figura 12 - Esquema de funcionamento da Ponte H

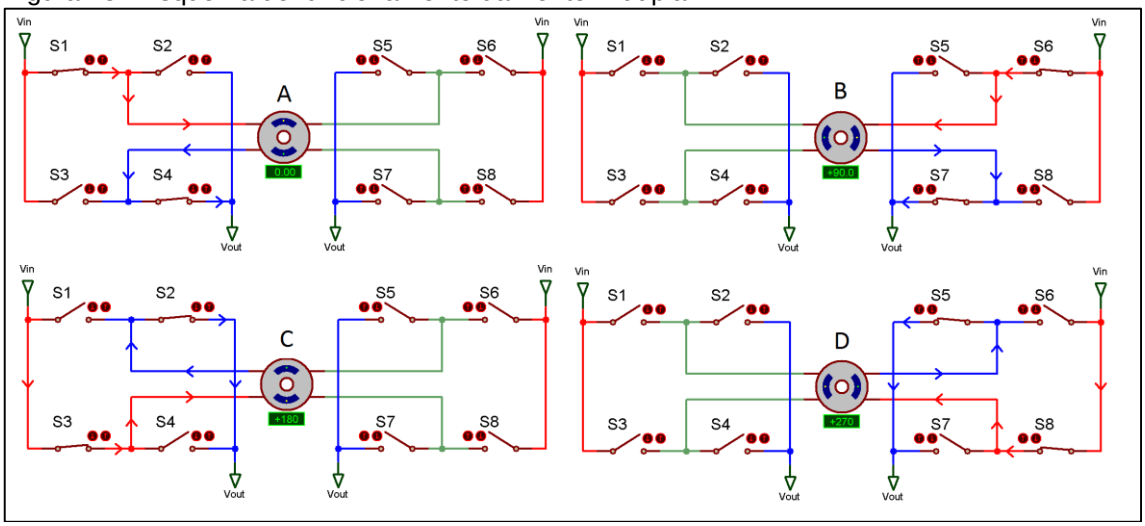


Fonte: Do autor.

2.4.2 Acionamento de motor de passo bipolar

O tipo de acionamento acima é reservado para motores de corrente contínua de uma única bobina. Para ligar um motor de passo, que possui no mínimo duas bobinas, é necessário utilizar oito chaves ou ponte H dupla, controlando o fluxo da corrente e do campo magnético dentro do motor. O acionamento será parecido com quatro chaves, acionando as chaves em sequência fará com que o motor gire. A simulação de um motor bipolar com oito chaves e com configuração em *full-step* pode ser compreendida pela figura 13 (QUEIROZ, 2002).

Figura 13 - Esquema de funcionamento da Ponte H dupla



Fonte: Do autor.

Percebe-se na figura 13 o comportamento da corrente que está circulando nas bobinas do motor, e com a troca das sequencias das chaves a corrente inverterá seu sentido (o que está em vermelho é a entrada da corrente, e em azul sua saída). No esquema A da figura 13, a corrente está circulando entre a chave S1 e a chave S4 energizando uma bobina do estator, fazendo com que o rotor permaneça em 0°. No esquema B ocorre o acionamento da outra bobina executando um passo de 90°. Seguindo a lógica da inversão de corrente nas bobinas, por se tratar de um motor bipolar ao acionar S2 e S3 (esquema C) acarretará na troca do sentido da corrente na primeira bobina executando um novo passo para 180°, se acaso após o esquema B fosse sequenciado o acionamento do esquema A, o rotor voltaria aos 0° e inverteria o sentido de giro. Compreende-se que ao inverter a sequência de

acionamento das chaves ocorrera a inversão de sentido de giro do motor (QUEIROZ, 2002).

Tabela 1 - Sequência de acionamentos das chaves de uma ponte H dupla

HALF-STEP	**FULL-STEP**
S1 - S4 = 0°	S1 - S4 = 0°
S1 - S4 - S5 - S8 = 45°	
S5 - S8 = 90°	S5 - S8 = 90°
S5 - S8 - S2 - S3 = 135°	
S2 - S3 = 180°	S2 - S3 = 180°
S2- S3 - S6 - S7 = 225°	
S6 - S7 = 270°	S6 - S7 = 270°
S6 - S7 - S1 - S4 = 315°	

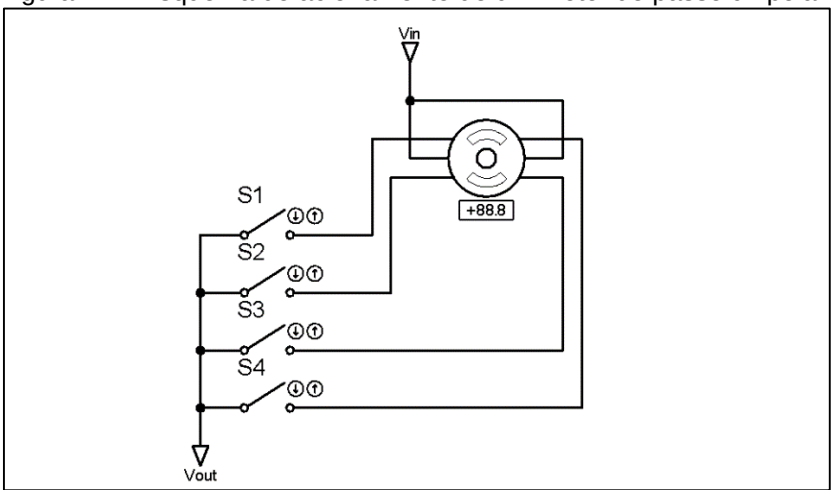
Fonte: Carvalho (2011).

A tabela 1 apresenta os possíveis acionamentos das chaves, em relação à figura 14, que resultarão em um passo em determinado grau. Conforme já mencionado, demonstra também as configurações em meio passo, *half-step*, e passo inteiro, *full-step*. Nota-se que ao acionar quatro chaves intermediarias entre os passos inteiros, obtém-se a metade do passo, ou seja, ao acionar S1 e S4 o passo resultante será 0°, S5 e S8 será 90°, acionando as quatro gera-se um meio passo intermediário a ambos que resulta em 45° (CARVALHO, 2011).

2.4.3 Acionamento de motor de passo unipolar

Para o acionamento deste tipo de motor, compreende-se que o mesmo possui derivação interna sobre as bobinas como já mencionado. Apresenta um acionamento diferente, porém de mais fácil entendimento, já que não há necessidade da inversão da corrente como no motor bipolar. Existem exatamente quatro chaves para o acionamento das bobinas, e são representadas pelas chaves S1, S2, S3 e S4 na figura 14 (BRITES e SANTOS, 2008).

Figura 14 – Esquema do acionamento de um motor de passo unipolar



Fonte: Do autor.

A simplicidade deste controle está no fato da facilidade de controlar o projeto, pois necessita de uma lógica básica e é de fácil confecção. Para mais entendimento, ao se ligar S1 o motor rotacionará para 90º, pois a corrente circulará de Vin para Vout pela chave que esta conectada a uma bobina do motor. Posteriormente desliga-se o S1 e aciona-se S2 fazendo com que o rotor de outro passo de 90°, estacionando em 180° e assim funcionará com a chave S3 e S4. A forma que um motor unipolar pode operar é compreendida pelas tabelas 2 e 3 sendo que as mesmas demonstram o acionamento das chaves em *half-step* e f*ull-step*, zeros equivalem a chave desligada e uma chave ativada. (BRITES e SANTOS, 2008).

Tabela 2 - Acionamento das chaves de um motor de passo em full-step

Sequência A					**Sequência B**				
N° de passo	**S1**	**S2**	**S3**	**S4**	**N° de passo**	**S1**	**S2**	**S3**	**S4**
1	1	0	0	0	1	1	1	0	0
2	0	1	0	0	2	0	1	1	0
3	0	0	1	0	3	0	0	1	1
4	0	0	0	1	4	1	0	0	1

Fonte: Brites e Santos (2008).

Tabela 3 - Acionamento das chaves de um motor de passo em half-step

N° de passo	S1	S2	S3	S4
1	1	0	0	0
2	1	1	0	0
3	0	1	0	0
4	0	1	1	0
5	0	0	1	0
6	0	0	1	1
7	0	0	0	1
8	1	0	0	1

Fonte: Brites e Santos (2008).

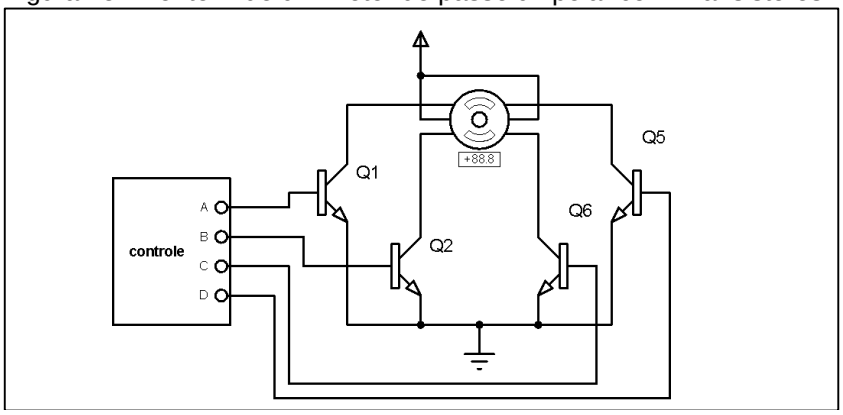
2.4.4 Chaveamento com transistores

A integração dos motores de passo com a eletrônica é projetada por meio de circuitos que atuam como as chaves para o acionamento. Podem ser eles, transistores, portas lógicas ou *drivers* especialmente projetados. Os transistores, Tocci e Widmer (2004), são componentes que podem atuar como chaves e estão em praticamente todos os dispositivos eletrônicos atuais. Possuindo três terminais, base, coletor e emissor, seu funcionamento é padronizado e simples, sendo que ao se impor uma pequena corrente na base obtém-se a passagem de corrente entre coletor e emissor, em outras palavras, a base seria o dedo humano ao tocar em um interruptor dando passagem a eletricidade entre os contatos e ligando talvez uma lâmpada. Esta corrente é limitada pelo modelo do transistor, assim como tensão e é adquirida no seu manual, conhecido como *datasheet*.

O acionamento com transistores, na pratica, deve ser monitorado e executado de forma exata, como ocorre nas simulações com chaves, o giro é feito por meio das sequências, e caso alguma chave for acionada de forma antecipada pode ocorrer um curto circuito nas bobinas do motor, causando mal funcionamento do passo ou em piores casos queimando o circuito de controle (BRITES e SANTOS, 2008).

A ação para interagir com os transistores, representados na figura 15 como Q1, Q2, Q3 e Q4, é derivada de um controlador para garantir a sequência de acionamento sem que exista a possibilidade do chaveamento equivocado. Entende-se que este controlador é manipulado por um indivíduo ou sistema, que por sua vez controla os transistores e os mesmos controlarão o motor (FITZGERALD, 2006).

Figura 15 - Ponte H de um Motor de passo unipolar com Transistores



Fonte: Do autor.

Os transistores são componentes que encadearam uma revolução tecnológica e computacional, foram componentes que agrupados em determinadas utilidades tornaram viável a lógica digital. Como resultado em tecnologia obtém-se uma variedade de componentes e processamento lógico, tais como processadores, portas lógicas, microcontroladores, *drivers*, entre outros. A combinação do ramo eletrônico com o industrial gerou uma satisfatória forma de controle ao processo automatizado (FITZGERALD, 2006).

2.4.5 Circuitos integrados dedicados

Para facilitar o controle existem *drivers*, circuitos integrados especialmente projetados para determinada aplicação, que fazem o papel de chaveamento e proteção do circuito de controle e do motor. Estes *drivers* por sua vez são um agrupamento de tecnologias que gerenciam a corrente internamente, possuindo comparadores, reguladores e transistores para específica aplicação (SOUZA, 2007).

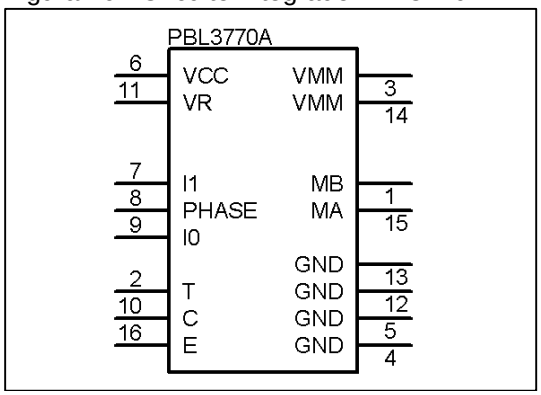
2.4.5.1 PBL 3770 A

Este circuito integrado consiste em entradas compatíveis com lógica digital, um sensor de corrente que serve para limitar a corrente dedicada ao motor (resistor de SENSE), um gerador de *clock* interno e uma ponte H de alta potência. O PBL é destinado a conduzir uma corrente constante, de até 2A, através de um enrolamento de um motor de passo de duas fases. Como o circuito é somente uma

ponte H, são necessários dois *drivers* para controlar um motor de passo bipolar (ERICSSON MICROELETRONICS, 1999, tradução nossa).

Este tipo de circuito possui controle de nível de corrente fornecida ao motor, ou seja, por meio de duas entradas lógicas I0(9) e I1(7) (figura 16), é possível limitar o fornecimento de corrente para as bobinas, podem ser elas: 0%, 20%, 60%, 100%.

Figura 16 - Circuito Integrado PBL3770A



Fonte: Do autor.

A lógica de acionamento destes terminais é definida pela tabela 4:

Tabela 4 - Níveis de Corrente em relação às entradas I0 e I1

Corrente na Bobina	**i0**	**i1**
100%	L	L
60%	H	L
20%	L	H
0%	H	H
L = nível lógico baixo		
H = nível lógico alto		

Fonte: Ericsson Microeletronics (1999).

De acordo com a Ericsson Microeletronics (1999, tradução nossa). Os valores específicos dos diferentes níveis de corrente são determinados pela tensão de referência (VR), pino 11, juntamente com o valor do resistor de SENSE. A corrente do motor (Im) pode ser calculada pela seguinte equação (1):

$$Im = \frac{Vr.0{,}080}{Rsense}[A], \quad em\ nivel\ 100\% \tag{1}$$

Uma metodologia é determinar à corrente que será utilizada, para posteriormente calcular o resistor que fornecerá esta corrente. Reformulando a equação (2):

$$Rsense = \frac{Vr.0,080}{Im}[\Omega], em\ nivel\ 100\% \quad (2)$$

Um exemplo seria determinar uma corrente de 1A com tensão de referência de 5V (3):

$$Rsense = \frac{5V.0,080}{1A}[\Omega] = 0,4\Omega \quad (3)$$

Resultando então um resistor de 0,4Ω (Ohms) que será utilizado como SENSE.

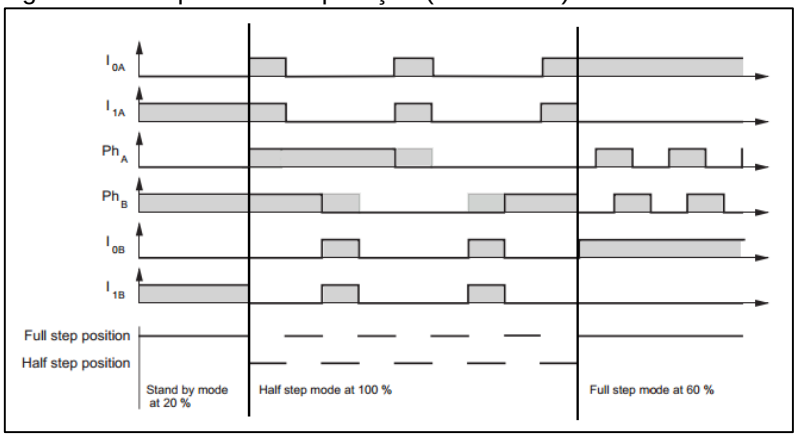
Este resistor deve ser aplicado em ambos os circuitos integrados, pois são necessários dois destes para ativar um motor bipolar. Se este resistor não for aplicado, os passos do motor serão executados de forma irregular, passos fortes e passos fracos intercaladamente, podendo na maioria dos casos causarem a perda de sincronia do rotor me relação ao estator, por não energizar a bobina corretamente ou perder a força de giro do rotor (ERICSSON MICROELETRONICS, 1999, tradução nossa).

O acionamento destes *drivers* deve ser em conjunto, e sequenciados de tal forma que ambos os PBL's possam combinar as características lógicas em seus terminais para assim possibilitar a execução do passo no motor (ERICSSON MICROELETRONICS, 1999, tradução nossa).

A figura 17 é como deve ser feita a combinação dos pinos I0, I1 e *Phase* (PH) de ambos os CI's para as posições em modo de *full-step* e *half-step*. A sequência mais simples de proceder é girar o motor em passo-inteiro em 60%, consistindo em manter nível lógico alto em ambos os pinos I0 dos CI's e baixo no I1. Isto fará com que o motor se mantenha energizado em 60% da capacidade que o *drive* distribui. Com motor energizado o rotor tende a ficar estático, se mantendo em uma posição e dificultando o giro manual, ou seja, se aplicar uma força manual no eixo do rotor o mesmo não se movimentará. Para a execução do passo os terminais

Ph devem ser intercalados conforme mostra a figura. E para a inversão do giro deve se inverter a sequência de intercalação dos terminais de *PHASE*.

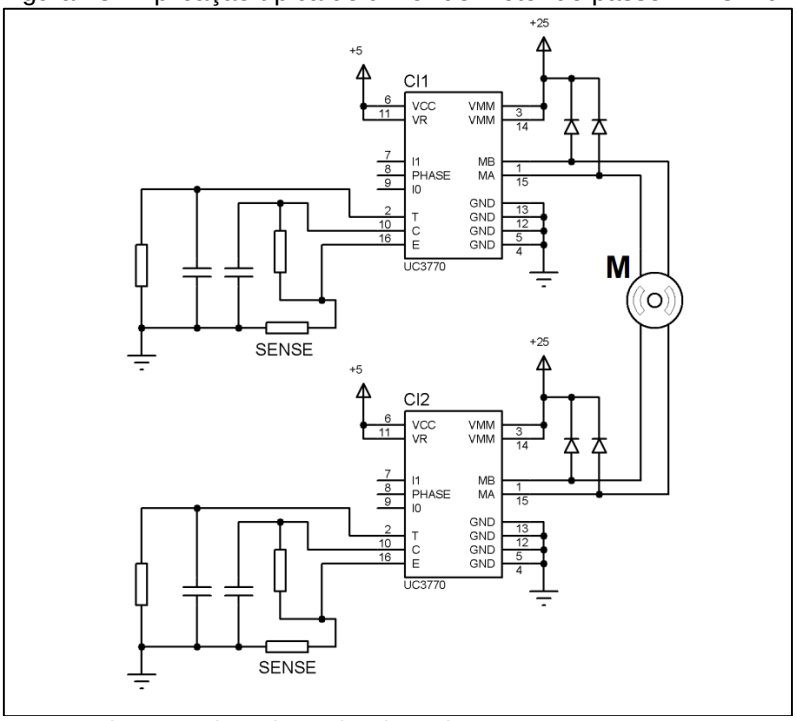
Figura 17 - Sequência de operação (PBL3770A)



Fonte: Ericsson Microeletronics (1999)

A Ericsson Microeletronics (1999, tradução nossa) disponibiliza o circuito de acionamento destes *drivers*, e sugere alguns componentes que farão o papel de *clock* do passo, circuito de proteção do motor e controle, filtros, entre outros (figura 18).

Figura 18 - Aplicação típica do *driver* de motor de passo PBL3770A



Fonte: Ericsson Microeletronics (1999).

Mostrados os circuitos de controle do motor de passo, foi necessário compreender seu funcionamento lógico, e verificou-se que todos os *drivers* aqui presentes possuem entradas digitais compatíveis com microcontroladores (MCU). Microcontroladores são CI's que possuem memória com inteligência programável, podendo ser manipulado para diversas aplicações. São usados em celulares, computadores, tablets, entre outros. Os microcontroladores são flexíveis para acionamentos digitais, pois possuem configurações internas de entradas e saídas, *timers*, contadores, comparadores, conversores, comunicações e outras diversas aplicabilidades dependendo do se deseja resolver (SOUZA, 2007).

2.4.6 Microcontroladores

Souza (2007) define o microcontrolador como um componente eletrônico, detentor de certa inteligência programável, e é utilizado em controle de processos lógicos. O controle de processo a qual Souza se refere pode ser qualquer tipo de controle que envolva sinais de entradas para controlar periféricos, tais como display, motores, solenóides e outros. E lógico, pois dependendo das condições de entradas e saídas serão executadas ações lógicas.

A lógica de operação de um microcontrolador possui uma estrutura na forma de programas, conhecido como *firmware*, e é gravada dentro do MCU. Quando alimentado, ou seja, quando ligado as suas condições energéticas, será executado o programa interno. A inteligência é associada ao núcleo de processamento interno, Unidade Lógica Aritmética (ULA), quanto mais robusta a unidade maior capacidade de processamento (SOUZA, 2007).

Para o acionamento dos *drivers* de motor de passo a escolha dos microcontroladores se tornou viável, pois em relação a outros tipos de acionamento, como Controlador Lógico Programável (CLP), que também desempenha funções de controle através de softwares, o microcontrolador é de fácil aquisição e de menor custo. Diversos são os fabricantes de microcontroladores, sendo mais encontrados e usados Controlador de Interface Programável *(Programmable Interface Controller*) PIC's, da Microchip, e AVR's da Atmel.

2.4.6.1 Microcontroladores PIC

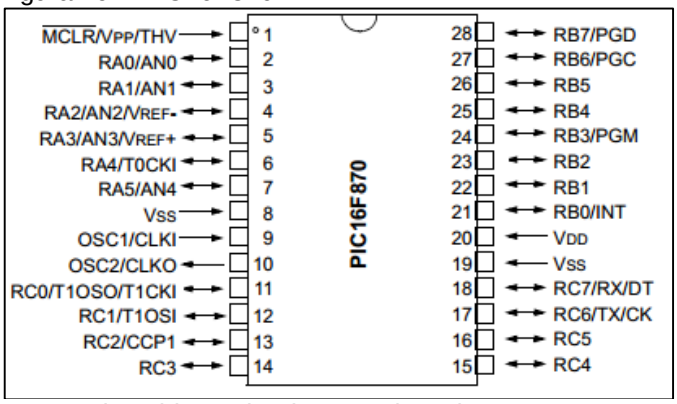
Os PIC's são microcontroladores desenvolvidos pela Microchip possuindo arquitetura interna *Havard*, diferentemente dos microprocessadores que possuem arquitetura Von-Neumann. Basicamente a arquitetura de *Havard* é composta por dois barramentos, no caso do PIC, um de 8 até 32 bits para trafego de dados e outro que pode ser de 12, 14 e 16 bits para instruções dependendo do dispositivo (SOUZA, 2007).

A Microchip aplicou esta arquitetura para garantir maior velocidade de busca e execução das instruções, pois os microcontroladores tendem a trabalhar com baixo *clock*, menor que 100Mhz, em relação aos microprocessadores, até 3Ghz (SOUZA, 2007).

O problema deste microcontrolador ser de pequeno porte, pequena memoria, baixo *clock* e ainda trabalhar com até 16 bits de instrução resulta em uma redução do espaço para o código de instrução, acarretando na aplicação da tecnologia *Reduced Instruction Set Computer* (RISC), computador com set de instruções reduzido. Em média com 35 instruções torna o aprendizado da tecnologia dinâmico e ao mesmo tempo exige habilidade do programador para construir instruções que não são disponibilizadas (SOUZA, 2007).

Os PIC's são divididos em famílias, ou seja, para cada nível de arquitetura de barramento existem diferentes prefixos na nomenclatura, os níveis são 8, 16 e 32 bits de palavras que a ULA processa. Ainda existem subfamílias dentro destas arquiteturas, que são distinguidas pela quantidade de bits por instrução, na família de 8 bits existem os PICs12, 16 e 18, sendo elas de 12, 14 e 16 bits de instrução. Um exemplo de nomenclatura de um PIC seria PIC16F870 (figura 19), definindo um PIC de 8 bits de processamento, 14 bits de instrução, memória *FLASH (F),* pode ser regravado inúmeras vezes, os próximos dígitos 870 são a identificação do microcontrolador (SOUZA, 2007).

Figura 19 – PIC16F870



Fonte: Microchip Technology Inc (2003).

A programação do MCU pode ser feita através de um ambiente de desenvolvimento integrado, MPLAB *Integrated Development Environment* (IDE), disponibilizada gratuitamente pela Microchip. A codificação por sua vez é feita em Assembly acarretando a complexidade no desenvolvimento de códigos. Porém existem *plug-ins* para codificação em C, pascal e basic voltados ao MPLAB, alguns são gratuitos como o *plug-in* HI-TECH C *Lite* que disponibiliza a codificação em C para as famílias 12 e 16. Existem ainda outros compiladores que facilitam a abstração na programação, tais como PIC CCS, mikroC e mikroPASCAL para PIC, por sua vez entende-se que quanto maior a abstração na programação, mais lento o programa ficará por estar mais distante da linguagem de máquina (MICROCHIP TECHNOLOGY INC, 2011, tradução nossa).

O princípio de um programa em um microcontrolador PIC consiste em manter o mesmo sempre executando, em *loop* infinito. Para a programação em C, por meio do *plug-in* HI-TECH C, o método principal deve ser parecido com a figura 20.

Figura 20 - Método principal de um microcontrolador escrito em C

```
void main(void)
{
    // metódos de configuração
    // e inicialização de variaveis e registradores
    while(1)
    {
        // codigo sempre em execução
    }
}
```

Fonte: Do autor.

O objetivo deste *loop* infinito, *while(1),* é manter o MCU sempre em execução para que em uma aplicação eletrônica o mesmo nunca pare de funcionar, ou termine a execução do programa por algum motivo (PEREIRA, 2007).

Os microcontroladores PIC's possuem inúmeros registradores para diversas aplicabilidades, sendo estes, portas, *timers*, PWM, interface serial, comparadores, conversores analógicos-digitais, entre outros (SOUZA, 2007).

Uma compreensão de um registrador seria a porta B. Definida como PORTB no manual, consiste em acessar os pinos referenciados a esta porta de forma digital. Normalmente a porta B dos PIC's da família 16 possui oito *bits*, um *byte,* que podem ser acionados ou desacionados através do código de programação (SOUZA, 2007).

No PIC16F870 (figura 19) a porta B está inserida entre os terminais 21 e 28, do RB0 ao RB7. Para setar uma saída em nível lógico alto, apenas se define o código RB0 = 1, por outro lado o comando RB0 = 0 irá colocar a saída em nível lógico baixo. Esta codificação de acionamento de *bits* da porta é executada dentro do *loop* principal (SOUZA, 2007).

Antes de acionar os terminais de uma porta, é necessário definir os pinos como saída ou entrada digital, portanto configurando o TRIS da porta. Este TRIS é um registrador de direção de dados acessado pelo registrador TRISx, sendo x o nome da porta. Este registrador possui também um *byte* de endereço podendo configurar saídas e entradas diferentes para cada *bit*. Para definir o pino RB0 como saída configura-se a TRISB antes do *loop* principal, sendo executado uma única vez, o valor zero configurará saídas e o valor um entradas (MICROCHIP TECHNOLOGY INC, 2003, tradução nossa).

A figura 21 mostra o *firmware* da configuração do TRISB e do acionamento intermitente do pino RB0, configura-se o pino RB0 como saída digital por meio do comando TRISB0, que se refere ao TRIS do mesmo:

Figura 21 – Acionamento do pino RB0

```
void main(void)
{
    TRISB0 = 0;
    while(1)
    {
        RB0 = 1;
        __delay_ms(500);
        RB0 = 0;
        __delay_ms(500);
    }
}
```

Fonte: Do autor.

O plug-in HI-TECH C para o MPLAB IDE, Microchip technology inc (2011, tradução nossa), possui baixa abstração, é uma programação em C voltado a linguagem de máquina, portanto o processamento do código é mais rápido do que em outros compiladores. Com este *plug-in* é possível acessar diretamente os registradores do MCU e intercalá-los com a codificação em C (figura 22).

Figura 22 – Codificação no HI-TECH C

```
void main(void)
{
    TRISB1 = 1; // entrada, um Botao
    TRISB0 = 0; // saida, uma luz, led

    int cont = 0;

    while(1) // loop
    {
        if(RB1 == 1) // verifica se o botão no pino RB1 foi apertado
        {
            cont++;     // Incrementa um contador
            while(RB1 == 1){} // enquanto o botão estiver pressionado, espera soltar
        }

        if(cont > 5)        // se apertou botão mais que 5 vezes aciona a saida RB0
        {
            RB0 = 1;
        }
    }
}
```

Fonte: Do autor.

Para gravar um programa dentro de um PIC é necessário a aquisição de um gravador especifico, normalmente vendido pelo próprio fabricante. Algumas tecnologias de gravadores possuem projetos distribuídos pela Microchip, como PICSTART Plus e PicKit 2, estes gravadores possuem limitações nos dispositivos a serem gravados, famílias 12, 16 e 18, e gravam em velocidades baixas. Tecnologias de gravadores como ICD 3, suportam todos os microcontroladores e são altamente

velozes na gravação, entretanto são vendidos a alto custo (MICROCHIP TECHNOLOGY INC, 2003, tradução nossa).

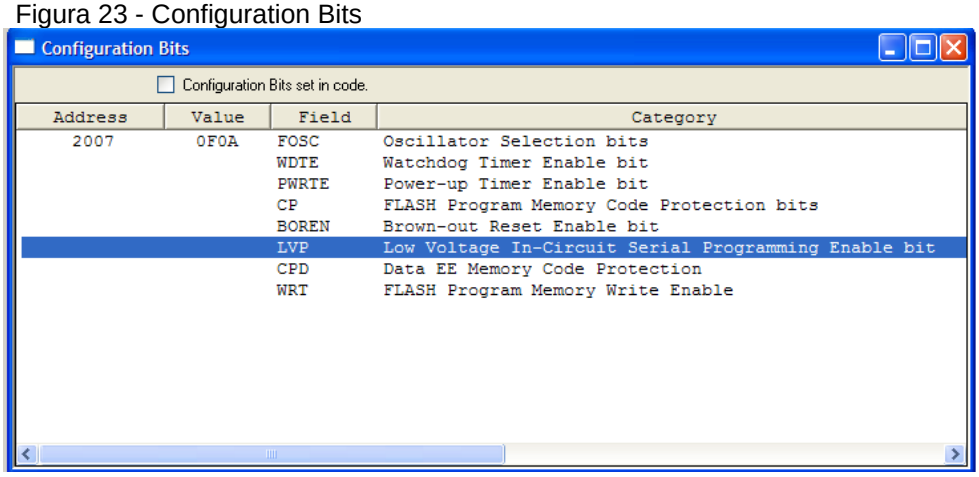
Os PIC's possuem *bits* de configuração, *configuration bits*, que são configurações pré-estabelecidas no momento da gravação do dispositivo. Estas determinam o modo de operação do MCU (PEREIRA, 2007).

Alguns bits de configuração do PIC16F870, Microchip Technology Inc (2003), são definidos como:

a) FOSC: determina a velocidade de operação do MCU;
b) CP: proteção da memória flash do programa, habilitando ou não a leitura;
c) CPD: proteção da memória EEPROM.

É de extrema importância configurar os *bits* quando um PIC é gravado, e por meio do MPLAB IDE podem ser acessados e manipulados pela janela *Configuration Bits* na aba *Configure* (figura 23).

Figura 23 - Configuration Bits



Fonte: Do autor.

O microcontrolador PIC é utilizado no controle de sistemas eletrônicos, com a finalidade de automatizar processos de forma ágil e direta. Como as entradas e saídas podem ser configuradas e acionadas, acarreta na captura de sinais digitais para verificação ou validação de um acionamento para determinado fim. Por possuir baixa abstração, a comunicação direta com um determinado usuário é feita quase sempre que mecanicamente, por meio de botões, sensores e display para visualização das informações. Dificilmente existe a intercalação entre PIC e computador, devido à proximidade da linguagem de máquina (SOUZA, 2007).

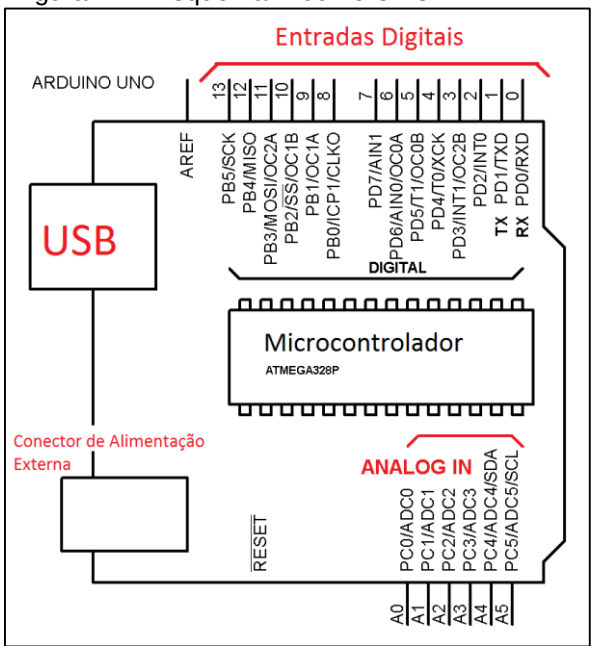
Uma solução para maior abstração aos microcontroladores foi o projeto Arduino, a qual consiste em um microcontrolador AVR, projetado em uma placa eletrônica com comunicação USB, facilitando o desenvolvimento por meio de um computador. O projeto além de ser livre disponibiliza a abstração através da IDE, Arduino IDE, possuindo bibliotecas para controle dos registradores de forma compreensível a iniciantes em MCU's.

2.4.6.2 Arduino

Iniciado na Itália, o projeto Arduino consiste em microcontrolador inserido em um protótipo eletrônico e com IDE específica. Por ser um projeto aberto possui grande acervo de informação na internet, por meio de desenvolvedores, programadores e possíveis apreciadores de tecnologias. Qualquer indivíduo pode alterar e produzir este protótipo, pois os esquemas eletrônicos são livres e disponíveis, contudo o título Arduino é reservado ao projeto original. Os modelos originais mais conhecidos são Arduino UNO, principal protótipo, o Arduino MEGA, possui mais quantidade de entradas e saídas e o Arduino NANO, feito para aplicações de baixo consumo e tamanho (MONK, 2010, tradução nossa).

O esquema eletrônico do Arduino UNO, figura 24, é composto de um microcontrolador AVR ATMEGA328P de 8 bits de processamento, uma porta USB usada para comunicação entre um computador e o protótipo, um conector para possível alimentação externa quando não ligado a USB e conectores de ligação externa para os pinos entradas e saídas (MCROBERTS, 2011).

Figura 24 – Esquema Arduino UNO



Fonte: Do autor.

McRoberts (2011) define ainda que o Arduino possui recursos eletronicamente limitados, somente o microcontrolador, porém é possível incorporar outros dispositivos, conhecidos como s*hields.* Os s*hields* são projetos eletrônicos, placas, separados que integram funções adicionais ao protótipo como: comunicação *bluetooth*, controle de motor de passo, *ethernet*, wi-fi, entre outros. As placas podem ser facilmente encaixadas no Arduino por meio dos pinos de entrada e saída, obtendo novas funcionalidades.

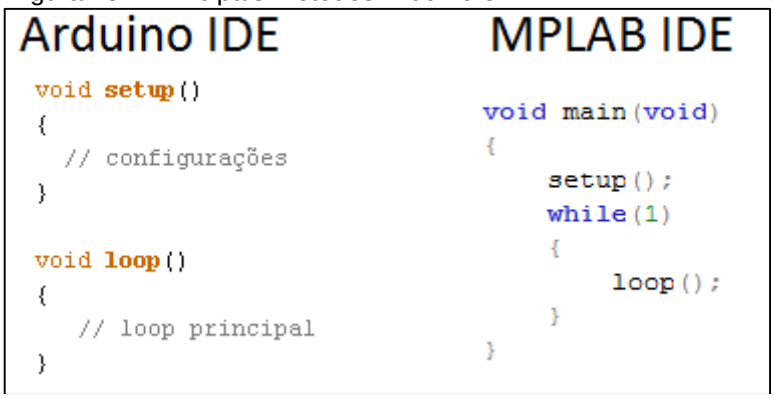
A vantagem do Arduino em relação ao PIC é sua facilidade em programação e gravação. Por meio da comunicação entre a IDE e a porta USB é possível carregar o programa sem utilizar um gravador, este processo possibilita a alteração do programa em tempo real utilizando um computador qualquer (MONK, 2010, tradução nossa).

O MCU do Arduino possui um software interno que possibilita a autogravação por meio da comunicação serial. Conhecido como *bootloader* este software é executado primeiramente, antes do programa desejado. Se ao ligar o protótipo houver comunicação serial exigindo a gravação, o *bootloader* inicia este processo e começa a carregar os dados a serem gravados para a memória flash do microcontrolador. Se não, o *bootloader* aponta para o endereço inicial do programa atual (MCROBERTS, 2011).

O desenvolvimento do software é feito pela Arduino IDE e pode ser baixada no *site* do projeto. Esta IDE possui alta abstração, está mais próximo da compreensão humana ao codificar, devido as bibliotecas que incorporam muitas tecnologias (MCROBERTS, 2011).

A codificação segue o mesmo princípio do código em PIC, também em C, e pode ser interpretada como abstração do método *main*, conforme a figura 25:

Figura 25 – Principais métodos Arduino e MPLAB



Fonte: Do autor.

O método *setup* é executado ao iniciar o programa e pode carregar configurações como: saídas, entradas, configuração serial, inicio de variáveis, entre outros. O método *loop,* é o loop principal do microcontrolador a qual destina-se a executar por tempo indeterminado o programa inserido (MCROBERTS, 2011).

O software para acionar um pino intermitente pode ser compreendido na figura 26.

Figura 26 – Acionamento Intermitente do pino 13

```
void setup() // setup do arduino
{
  pinMode(13,OUTPUT); // define pino 13 como saida
}

void loop() // loop principal do programa
{
  digitalWrite(13,HIGH); // poe em estado alto pino 13
  delay(500); // aguarda 500ms
  digitalWrite(13,LOW); // poe em estado baixo pino 13
  delay(500); // aguarda 500ms
}
```

Fonte: Do autor.

O conector USB possibilita a comunicação serial pela interface RS232, que segundo Moraes (2007) é a tecnologia mais básica para a comunicação serial entre dois dispositivos. A RS232 consiste em enviar *bits* de forma assíncrona, por intervalo de tempo, e pode ser emulada pela comunicação USB do computador.

Ao plugar, por meio de um cabo USB, o Arduino em um computador é requisitado a instalação do *driver* de emulação USB-RS232, o mesmo está contido dentro da pasta onde foi instalada a IDE do Arduino. Após a instalação o computador reconhecerá uma porta serial COM que pode ser acessada por qualquer software que manipule as comunicações seriais do computador (SANTOS, 2008).

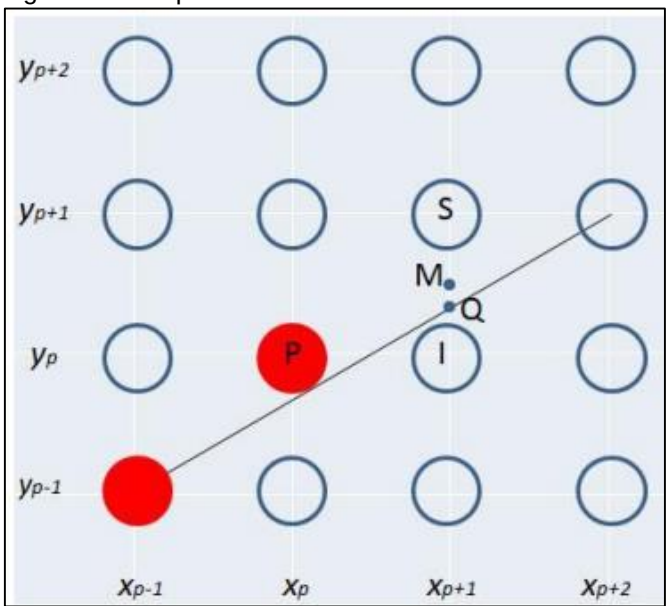
Existem diferentes algoritmos para a implementação em trajetórias xy, os mais encontrados são algoritmos matemáticos que trabalham com ponto flutuante exigindo maior capacidade de processamento, que não é o caso dos microcontroladores. O algoritmo de Bresenham é uma solução rápida e matematicamente eficaz pois trabalha com números inteiros, e pode ser implementado dentro de um microcontrolador para traçar retas no sistema xy.

2.5 ALGORITMO DE BRESENHAM

É relativamente fácil criar um algoritmo para traçar retas e gerar pontos aproximados sobre bases matemáticas de pontos flutuantes. Hoje possivelmente as ULA’s dos computadores possuem instruções para trabalhar com este tipo de sistema flutuante de forma rápida, porém no caso de microcontroladores esta situação se converte em problema devido à baixa velocidade de operação e as reduzidas instruções (MORO, 2009).

O uso dos microcontroladores na automação são um problema se não usados corretamente. Usar ponto flutuante exige do processador vários ciclos de *clock* acarretando na defasagem da comparação e controle no processo automatizado. Para contornar esta situação opta-se por algoritmos de alto nível minimizando o trabalho computacional. Em 1965 Jack E. Bresenham, então funcionário da IBM propõe um algoritmo eficaz para a época que calculava melhor ponto nas trajetórias retilíneas por meio de lógica com números inteiros (MORO, 2009).

Figura 27 – Superfície de *Pixels*



Fonte: Moro (2009).

Assumindo que *pixel* que acabou de ser selecionado P e o próximo que deve ser escolhido seja S ou I, conforme a figura 27. Verifica-se que a trajetória da reta atravessa a coluna *Xp+1* no ponto Q e está logo abaixo do ponto M que é o ponto médio entre os *pixels* a serem escolhidos. Partindo deste princípio é fácil analisar que o ponto Q está mais próximo do *pixel* I, então é necessário um cálculo para dizer de que lado da reta está o ponto M (MORO, 2009).

Bresenham (1965, tradução nossa) parte do princípio da função de reta objetivando eliminar o coeficiente angular (4):

$$y = \frac{\Delta y}{\Delta x}.x + B \tag{4}$$

$$onde\text{: } \Delta x = x2 - x1 \text{ e } \Delta y = y2 - y1$$

Reescrevendo a equação de forma implícita *F(x,y)* para eliminar a fração (5),(6)*:*

$$F(x,y) = y - \frac{\Delta y}{\Delta x}.x - B = 0 \tag{5}$$

$$F(x,y) = x\Delta y - y\Delta x + B\Delta x = 0 \tag{6}$$

Resultando na função (7):

$$F(x,y) = ax + by + c = 0 \quad (7)$$

Sendo que: $a = \Delta y \quad b = -\Delta x \quad c = B\Delta x$

Utilizando esta função é possível analisar que *F(x,y)* é zero para pontos sobre da reta, negativa para pontos acima e positiva abaixo. Logo em seguida é analisado o ponto médio M com base ao *pixel* atual P e verificado qual *pixel* foi escolhido, S ou I. Uma nova comparação lógica é determinada para saber o quanto e qual valor de x e y será incrementado para o próximo *pixel* dependendo do *pixel* atualmente escolhido (BRESENHAM, 1965, tradução nossa).

Bresenham (1965, tradução nossa) por meio dos cálculos e eliminando a fração da equação primitiva da reta encontra equações de decisões, ou métodos que satisfazem o coeficiente angular caso as seleções dos *pixels* sejam diferentes. As equações de decisões são consideradas constantes de Bresenham, e são elas:

$$D_{start} = 2\Delta y - \Delta x \qquad condição\ inicial$$
$$D_{new} = D_{old} + 2\Delta y \qquad se\ I\ for\ escolhido$$
$$D_{new} = D_{old} + 2(\Delta y - \Delta x) \qquad se\ S\ for\ escolhido$$

Por fim o Bresenham (1965, tradução nossa) apresenta condições ainda de recursão, caso a inclinação da reta seja invertida, x1 > x2, para satisfazer o algoritmo por completo. O algoritmo de Bresenham pode ser desenvolvido em qualquer linguagem, figura 28, e em qualquer sistema de modo eficaz, por possuir lógica inteira.

Figura 28 – Algoritmo de Bresenham em C

```
void bresenham1(int x1, int y1, int x2, int y2){
      int slope;
      int dx, dy, incE, incNE, d, x, y;
      // Onde inverte a linha x1 > x2
      if (x1 > x2){
          bresenham1(x2, y2, x1, y1);
           return;
      }
      dx = x2 - x1;
      dy = y2 - y1;

      if (dy < 0){
          slope = -1;
          dy = -dy;
      }
      else{
         slope = 1;
      }
      // Constante de Bresenham
      incE = 2 * dy;
      incNE = 2 * dy - 2 * dx;
      d = 2 * dy - dx;
      y = y1;
      for (x = x1; x <= x2; x++){
          putpixel(x, y);
          if (d <= 0){
            d += incE;
          }
          else{
            d += incNE;
            y += slope;
          }
      }
}
```

Fonte: Moro (2009).

Em um sistema automatizado a última área consiste no software gerenciador de todo sistema. Este software entra como analisador das informações geradas por todo processo automatizado, por meio de comunicações entre o sistema de controle.

O Arduino por possuir maior abstração facilita no processo de comunicação entre um computador e o sistema. Esta comunicação deve ser elaborada para receber os dados de um software e converte-los em passos para os motores. Protocolos devem garantir a exatidão no recebimento das informações enviadas a máquina, de tal maneira que comandos e dados sejam interpretados pelo Arduino.

2.6 COMUNICAÇÂO DE DADOS

A troca de informações entre dois dispositivos através de um meio de comunicação é conhecida como comunicação de dados. Os dispositivos precisam ser parte de um sistema feito com a união de hardware e software (FOROUZAN, 2006).

Forouzan (2006) destaca três características básicas, que garantem a eficiência e o bom funcionamento da comunicação:

a) entrega: deve garantir o recebimento dos dados corretos ao dispositivo de destino;
b) confiabilidade: o sistema deve garantir a entrega dos dados;
c) tempo de atraso: envio dos dados possui um tempo limite, necessitando ser o mais eficiente possível.

Tanenbaum (2002) classifica a comunicação de acordo com o fluxo de dados a serem trafegados:

a) *simplex*: somente um dispositivo é capaz de transmitir e o outro apenas é capaz de receber, comunicação é unidirecional. Rádios são exemplos desta comunicação*;*
b) *half-duplex*: ambos dispositivos recebem e enviam dados, mas não ao mesmo tempo. Quando um transmite o outro recebe e vice-versa. Um exemplo são os radiocomunicadores usados por seguranças e militares;
c) *full-duplex*: este fluxo descreve o compartilhamento da capacidade do meio de transmissão, ou seja, os dispositivos podem enviar e receber dados simultaneamente. Um exemplo de *full-duplex* são os celulares.

Um sistema de comunicação básico, Tanenbaum (2002), possui no mínimo cinco componentes:

a) mensagem: dados que serão transmitidos;
b) transmissor: dispositivo usado para transmitir a mensagem;
c) receptor: dispositivo usado para receber a mensagem;
d) meio: caminho físico por onde trafega a mensagem, do transmissor para o receptor;

e) protocolo: regras que padronizam o envio e o recebimento das informações.

2.6.1 Protocolos de comunicação

Protocolos são um conjunto de regras e procedimentos que determinam a comunicação entre transmissor e receptor. A transgressão ou a não realização destas regras inibirá o entendimento entre os dispositivos e, em alguns casos, não haverá a comunicação (TANENBAUM, 2002).

Forouzan (2006) determina três elementos fundamentais de um protocolo:

a) sintaxe: estrutura e forma de apresentação dos dados. Um exemplo é um protocolo que determina os primeiros *bytes* de comando e endereços e os demais como restante de dados;
b) semântica: determina o sentido de cada conjunto de *bits* enviados e recebidos. A semântica define como os dados serão interpretados e a ação que será tomada baseada nesta interpretação;
c) temporização: define quando os dados serão enviados e qual a velocidade de envio é utilizada.

Existem vários protocolos disponíveis, cada um criado para situações específicas. Na internet são utilizados conjuntos de protocolos denominados TCP/IP. Onde os mais conhecidos são HTTP, FTP, IP, TCP, UDP, SMTP (FOROUZAN, 2006).

2.7 TRABALHOS CORRELATOS

2.7.1 Controle de mesa xy: utilizando motor de passo

Artigo desenvolvido por Sebastião G. dos Santos Filho, professor associado do departamento de engenharia de sistema eletrônicos da escola politécnica da USP, publicado na revista Mecatrônica Atual, volume 1, número 2, em janeiro de 2002. Consiste no desenvolvimento de uma mesa xy com motores de passo de deslocamentos micrométricos objetivando a detecção de material particulado na superfície da mesa. O projeto estuda motores de passo e seus

controles eletrônicos, como chaveamento e controle de pulsos. A mesa é feita por dois blocos metálicos que se movimentam horizontalmente sobre uma superfície, cada bloco possui um parafuso atravessado e encaixado em um motor de passo.

2.7.2 Algorithm for computer control of a digital plotter

Este artigo foi desenvolvido por Jack Elton Bresenham, colaborador da IBM e profissional em Ciência da Computação, publicado no Jornal de Sistemas da IBM em 1965. O Artigo detalha o funcionamento de seu algoritmo aplicado em uma impressora para o movimento do *plotter*. O maior foco do trabalho está em resoluções matemáticas para provar a funcionalidade do algoritmo. Por meio dos cálculos das retas são efetuados testes de tempos em algoritmos flutuantes e inteiros, comprovando uma melhor eficácia computacional.

2.7.3 Algoritmo de Bresenham: o uso microcontroladores para traçar retas em LCD's

Desenvolvido por Jefferson Zortea Moro em 2009 e apresentado em um seminário sugerido pelo Departamento de Engenharia Elétrica da Universidade Federal do Espírito Santo na cidade de Vitória, este artigo propõe a manipulação de *pixels* em um display gráfico utilizando o algoritmo de Bresenham. O projeto utiliza microcontroladores para a manipulação do LCD, e devido à baixa velocidade de *clock* foi necessário a implementação do algoritmo para melhor desempenho. Abordando todo o conceito de funcionamento do algoritmo, Jefferson implementa códigos no MATLAB para posteriormente utilizar no MCU. Além de traçar retas, busca também implementar soluções para projetar curvas e elipses no LCD.

3 SISTEMA XY

Este sistema permite disponibilizar um modelo, que pode ser adaptado para diferentes áreas de atuação na automação industrial, ou seja, foi com o intuito de facilitar o processo produtivo, garantindo um menor custo com tempo na produção, ou confecção de uma máquina com finalidades plotadores. Conhecendo como sistema se comunica é possível sua implementação em qualquer linguagem, ou até mesmo a conversão de softwares já disponíveis no mercado. Alguns exemplos de como o sistema pode ser abrangente, é colocar um canhão de corte a laser, uma máquina serigráfica, uma máquina de bordados, entre outros.

3.1 MONTAGEM DO SISTEMA

Para este sistema, foi determinado a elaboração desde a parte de comparação, controle, atuação e sensoriamento, visando uma melhoria na execução dos processos. Estas etapas, englobam a confecção de um *drive*r eletrônico para os motores de passo, a construção de uma mesa mecânica, a definição de um protocolo de comunicação, permitindo que qualquer software o implemente para controlar o sistema, controle de posição e controle dos atuadores.

3.1.1 Desenvolvimento do *drive*r de motor de passo

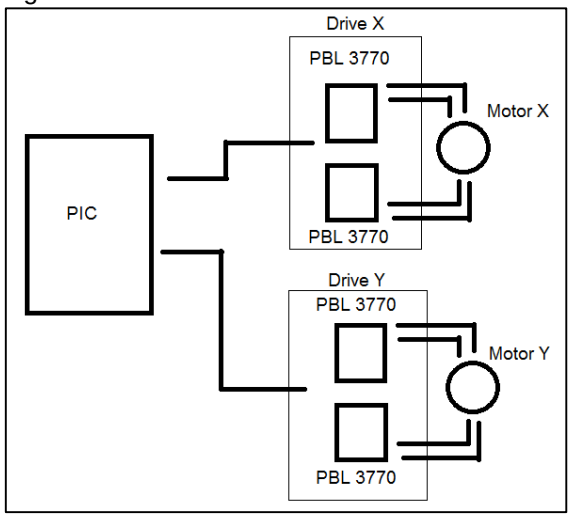
O objetivo de projetar o *drive* para motores de passo, é facilitar o controle lógico dos mesmos. Por meio do MCU é possível elaborar um projeto eletrônico que receba somente sinais de *clock*, servindo como passos, e sinais de sentido de rotação.

Para o projeto é necessário analisar quais condições os motores deverão trabalhar. Utilizando o circuito integrado PBL 3770 os motores deverão trabalhar no máximo a 2A de corrente elétrica e 25 volts de tensão, o intuído de utilizar este CI é aplicar motores bipolares que possuem maior força.

O circuito foi desenvolvido com base no esquema de controle disponibilizado pelo *datasheet* do PBL (figura 18, capitulo 2.4.5.1, p. 38). Utilizando

seus demais componentes foi induzido um microcontrolador, PIC, para os *drivers*, conforme a figura 29.

Figura 29 – Modelo do *driver* de controle dos motores de passo



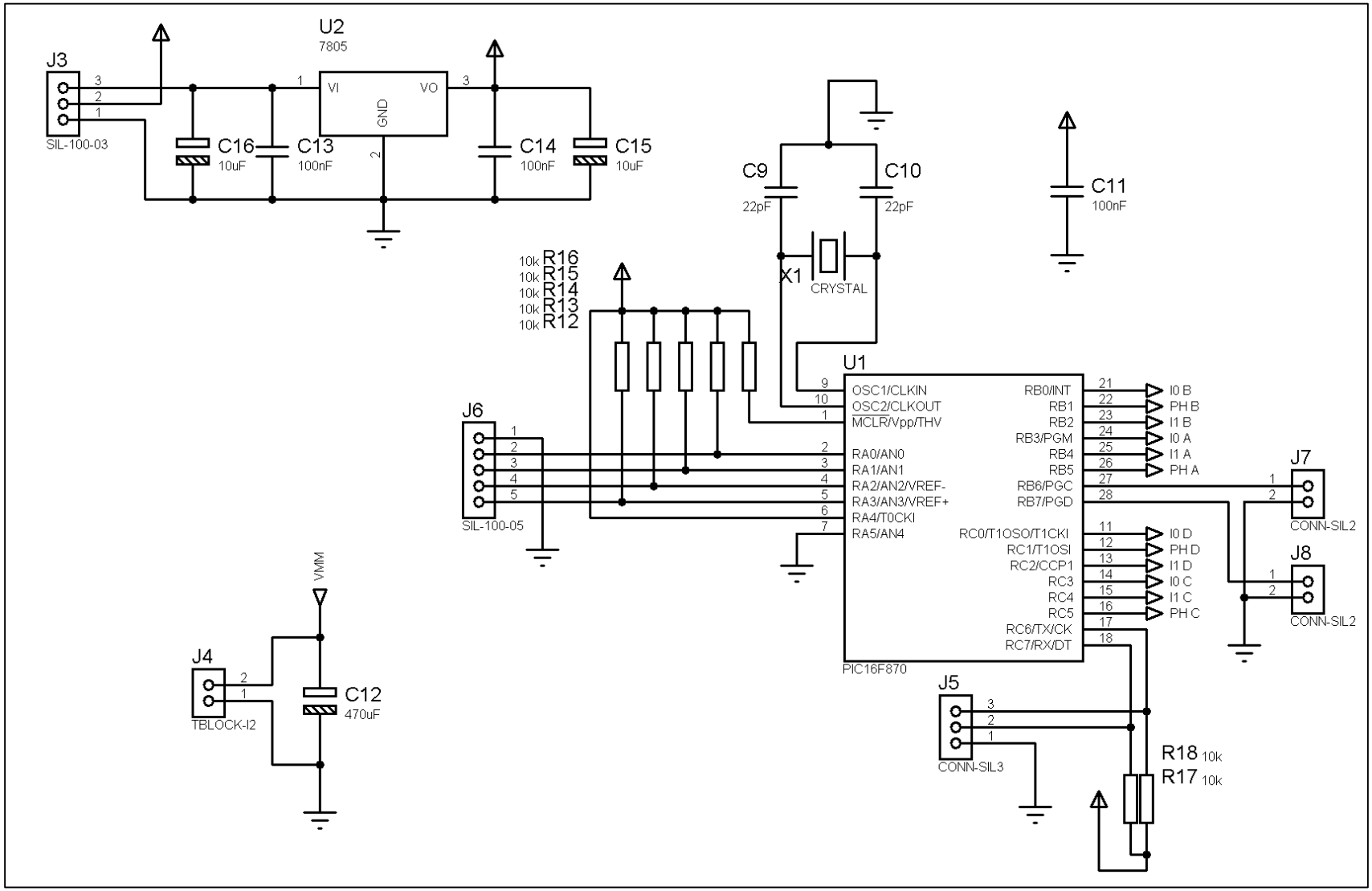
Fonte: Do autor.

Este CI conforme já citado possui controle de corrente que pode ser acessado pelos pinos I0 e I1, com o PIC é possível gerenciar a corrente que os PBL's fornecerão ao motor, é possível também estipular o controle de passos, *half* e *full-step*, aumentando assim a precisão com que o motor trabalhará.

Com o modelo do funcionamento do driver elaborado, foi preparado seu circuito eletrônico, com base no esquema fornecido pelo *datasheet* do 3770, contudo algumas alterações foram visionadas, ao entender que antes de construir, seria necessário estipular alguns conectores para facilitar o encaixe e desencaixe, dos motores, fontes de energia e também a comunicação com o controlador do sistema. O esquema (figura 30) foi projetado com auxílio do software *Proteus 8 Demo*[1], u tilizando a ferramenta ISIS, devido a maior praticidade no uso.

[1] Proteus 8 Demo: Software disponibilizado pela Labcenter Eletronics, que tem como finalidade a projeção de esquemas e layouts para placas de circuito impresso. A versão demo conta ainda com a capacidade simular e *debugar firmwares* de microcontroladores e outros componentes eletrônicos.

Figura 30 – Controle do *driver* PBL 3770 com PIC



Fonte: Do autor.

A figura 30, mostra o circuito de controle do *drive*r, este circuito possui um PIC16F870, com a designação U1 sobre o mesmo. Este PIC foi escolhido por possuir os recursos básicos disponíveis para a construção do *drive*r, contêm três portas (A, B e C), 28 pinos e outros sistemas como comunicação serial e *timers*. Também é possível utilizar um cristal externo, garantindo mais velocidade de processamento. Foi utilizado todas as portas, no MCU, para o projeto de controle do *drive*r. Posteriormente, definiu-se pinos de entrada para sinal de pulsos, sentido de giro dos motores, controle de corrente e controle de passos. Estes pinos por sua vez são:

a) pinos 2 e 3 (RA0 e RA1), determinam sinal de sentido e entrada de pulsos para o motor do eixo X;
b) pinos 4 e 5 (RA2 e RA3), determinam sinal de sentido e entrada de pulsos para o motor do eixo Y;
c) pinos 27 e 28 (RB6 e RB7), junto com os conectores J7 e J8 são jumpers de controle de corrente e passos *half-full*, para ambos os motores.

Os pinos 21 a 26 e 11 a 16 são saídas para o controle dos 3770's. As saídas RB0 ao RB5 controlam os PBL's A e B do acionamento para o motor X (Apêndice A) e as saídas RC0 ao RC5 controlam o acionamento do motor Y, PBL's C e D (Apêndice B).

Para melhor compreensão a linguagem C possui diretivas de programação que ajudam a organizar a programação em baixa abstração. Uma destas diretivas possui a sintaxe "#define" e permite renomear os pinos do PIC com o intuito de tentar elevar o nível de abstração do código desenvolvido. Utilizando esta diretiva, os pinos foram renomeados de modo a atender as entradas dos *drive*r PBL 3770, identificando cada pino de saída do PIC, exatamente com um pino de entrada dos PBL's. As definições estão detalhadas na tabela 5.

Tabela 5 - Descrições dos pinos do PIC

Diretiva	Nomenclatura	Pino	Descrição
#define	CW_CCW_M1	RA0	sentido motor X
#define	*CLOCK*_M1	RA1	*clock* motor X
#define	CW_CCW_M2	RA2	sentido motor Y
#define	*CLOCK*_M2	RA3	*clock* motor Y
#define	I0B	RB0	I0 do PBL3770 B
#define	PHB	RB1	PHASE do PBL3770 B
#define	I1B	RB2	I1 do PBL3770 B
#define	I0A	RB3	I0 do PBL3770 A
#define	I1A	RB4	I1 do PBL3770 A
#define	PHA	RB5	PHASE do PBL3770 A
#define	I_FULL	RB6	Jumper de controle de Corrente
#define	HALF_FULL	RB7	Jumper de controle de Half-Full
#define	I0D	RC0	I0 do PBL3770 D
#define	PHD	RC1	PHASE do PBL3770 D
#define	I1D	RC2	I1 do PBL3770 D
#define	I0C	RC3	I0 do PBL3770 C
#define	I1C	RC4	I1 do PBL3770 C
#define	PHC	RC5	PHASE do PBL3770 C

Fonte: Do autor.

Os demais componentes neste circuito como J3 e J4 são conectores de alimentação, J3 para alimentação no PIC, que por meio do componente 7805, U2, (regulador de tensão para +5volts), garante este nível de tensão. J4 para alimentação de 20V dos motores. O conector J6 serve para entrada dos sinais de passos e sentido de giro. J5 é um conector para utilização futura de comunicação serial, já que o PIC pode ser reprogramado. Componentes que incluem C em sua descrição são capacitores comumente usados para filtros de energia e sinal. Componentes que possuem descrição iniciada por R são resistores para limitação de corrente.

3.1.1.1 Firmware do *drive*r

O *firmware* deverá seguir as regras de controle de níveis nos pinos PH, I0 e I1 em dois 3770 para controlar um motor, estas regras estão dispostas na figura 17, capitulo 2.4.5.1, p. 32. Utilizando a MPLAB IDE com o *plugin* HI-TECH, desenvolveu-se um *firmware* para o PIC16F870 em linguagem C. A primeira parte da codificação foi dividir os acionamentos de forma a habilitar o controle de passos em cada motor e iniciar o sistema por meio do método e loop principais (figura 31).

Figura 31 – Inicio do firmware de controle do *drive* PBL 3770

```
void main(void)
{
    static char valorM1 = 0; //variavel de controle do motor X
    static char valorM2 = 0;

    InicializaSistema(); // configuras as entrada e saidas

    while(1)
    {
        verificaJumper();       // verifica os Jumpers de Corrente e Half-full

        if(Controle.half_full == HALF) // se Jumper de half_full estiver engatado
        {
            valorM1 = acionaHALF_STEP_M1(valorM1); // aciona motor X em half step
            valorM2 = acionaHALF_STEP_M2(valorM2); // aciona motor Y em half step
        }
        else //senão
        {
            valorM1 = acionaFULL_STEP_M1(valorM1); // aciona motor X em full step
            valorM2 = acionaFULL_STEP_M2(valorM2); // aciona motor Y em full step
        }
    }
}
```

Fonte: Do autor.

Os métodos de acionamento que a figura 31 mostra como “acionaFULL_STEP_M1” (motor x) e “acionaFULL_STEP_M2” (motor y), foram implementados com base no acionamento dos *drive*r em *Full-Step*, possuindo maiores detalhes na figura 32. Percebe-se também a utilização das definições de modo a facilitar a transformação do código em acionamento dos *drivers*.

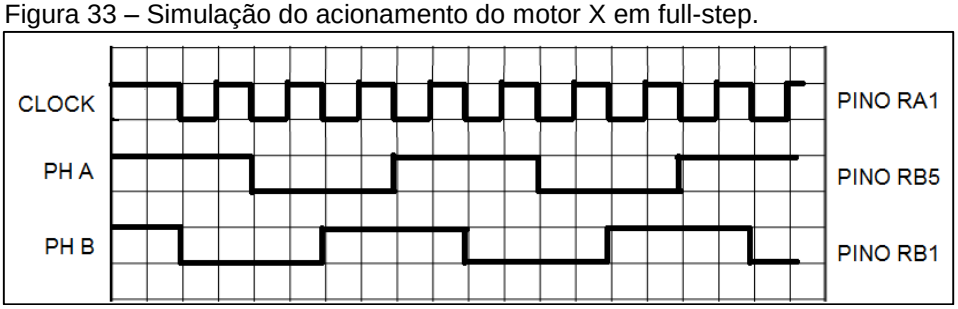
Figura 32 – Codificação em C do acionamento em *full step* do motor X

```
if(!CLOCK_M1 && !CLOCK_M1 && !CLOCK_M1) //verificação do sinal de clock
{
    I0A = 1;    I0B = 1;    // acionamento dos PBL's em Full-step
    I1A = 0;    I1B = 0;

    m1.invertePH = !m1.invertePH;   // variavel de control para inversão do PH

    if(m1.invertePH)
        PHA = !PHA;     // executa os passos no modo de fullstep
    else
        PHB = !PHB;
}
```

Fonte: Do autor.

O software Proteus 8 Demo, disponibiliza a capacidade de simular o firmware dentro do MCU. Para simular é necessário carregar o arquivo compilado do PIC. O Proteus também possui ferramentas para criar *clocks* de pulsos e osciloscópios para a medição das saídas do PIC, ajudando ainda mais na simulação do projeto. A figura 33 mostra os pulsos gerados pelo PIC em PHA e PHB, de acordo com os pulsos recebidos, *CLOCK*, executando a sequência de operação do PBL 3770 em *full-step.*

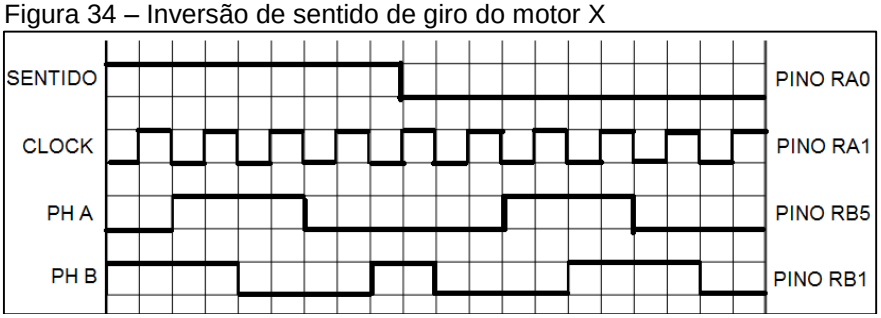
Figura 33 – Simulação do acionamento do motor X em full-step.



Fonte: Do autor.

Também pode ser visualizado o que acontece nas fases (PHASE) quando ocorre a troca de sentido, sendo que o *clock* permanece continuo. Percebe-se o que o *firmware* produz, figura 34, quando existe uma inversão de estado no pino de sentido. A troca de sentido do giro deve ser feita com a inversão de somente uma fase de um 3770, enquanto a outra permanece no mesmo estado.

Figura 34 – Inversão de sentido de giro do motor X



Fonte: Do autor.

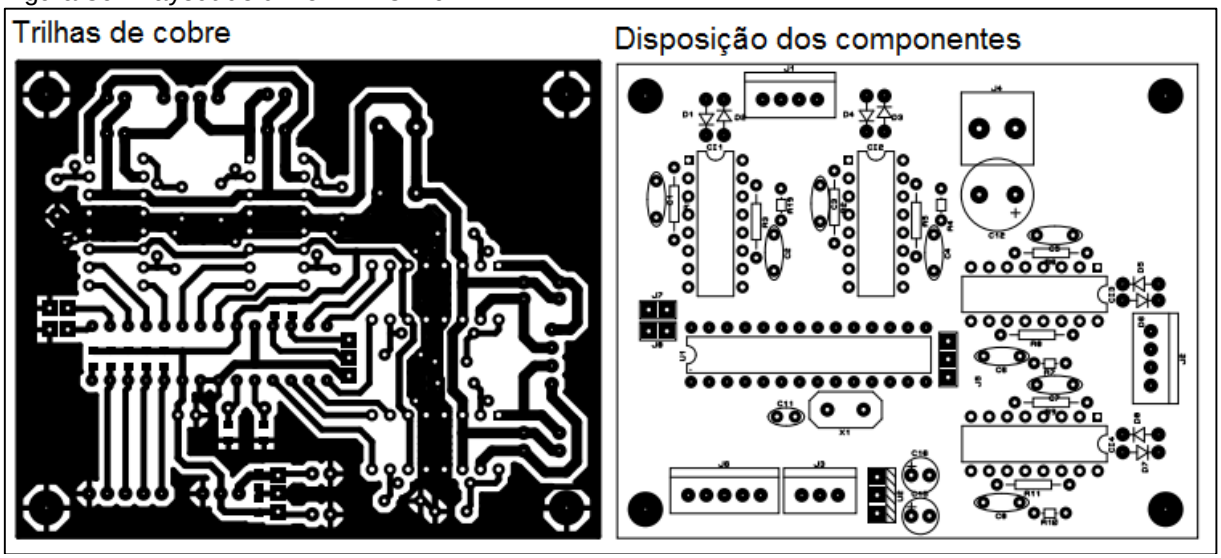
Com a implementação em *full-step* elaborada, é possível a gravação do firmware no PIC por meio de um gravador, foi utilizado o gravador PicKit 2, figura 35, adquirido por compra. Este dispositivo pode gravar inúmeros PIC's de várias famílias, possuindo um conector de fácil encaixe para os MCU's, além do software para gravação ser disponibilizado gratuitamente pela Microchip.

Figura 35 – Gravador de PIC PicKit 2

Fonte: Do autor.

Após o firmware estar gravado no MCU, foi necessário confeccionar o layout do *drive*r. Para esta confecção ainda foi utilizado o software Proteus 8 Demo com a ferramenta ARES, esta ferramenta possibilita a distribuição dos ligamentos entre cada pinos dos componentes do esquema. Observando as conexões dos encaixes dos motores, alimentação e sinais de entradas, foi determinado que colocar os conectores nas bordas do layout facilitaria este processo. Esta parte do projeto pode ser visualizada na figura 36, onde o layout foi produzido de modo a atender algumas características dos *drivers*, como trilhas de dissipação de calor, que consomem maior parte da placa.

Figura 36 - Layout do *drive*r PBL3770

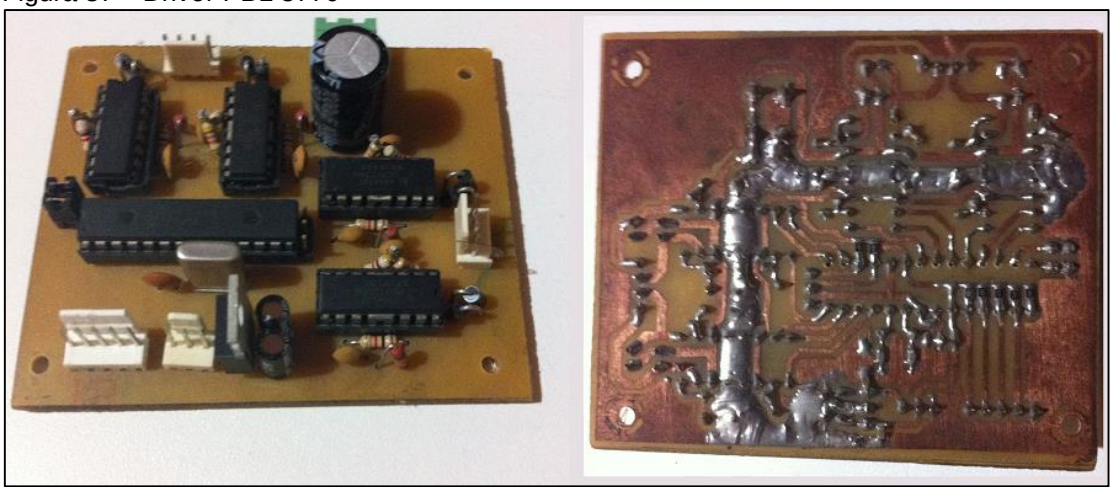


Fonte: Do autor.

O próximo passo foi corroer uma placa de cobre para dar origem ao layout. Esta etapa não necessita de grande conhecimento ou habilidade, requer somente alguns itens que podem ser adquiridos em lojas de produtos eletrônicos e existem inúmeros tutoriais explicando como produzir um hardware caseiro qualquer, na internet. Após corroído, foram necessários o encaixe e a solda dos componentes em seus devidos lugares. Por meio do esquema, foi possível visualizar a localização dos componentes de acordo com suas referências. Ao montar os PBL's foi estipulado uma corrente de 1,5A para o motor trabalhar dentro de seu limite de 2A, para tal, foi necessário escolher um resistor de SENSE que alcançava este valor. Por meio do cálculo, foi definido um resistor de aproximadamente 0,27Ω em cada 3770, este resistor garante uma corrente de 1,48A e é comercialmente viável. O

projeto do hardware do *drive*r PBL 3770 confeccionado pode ser mostrado na figura 37.

Figura 37 – *Drive*r PBL 3770



Fonte: Do autor.

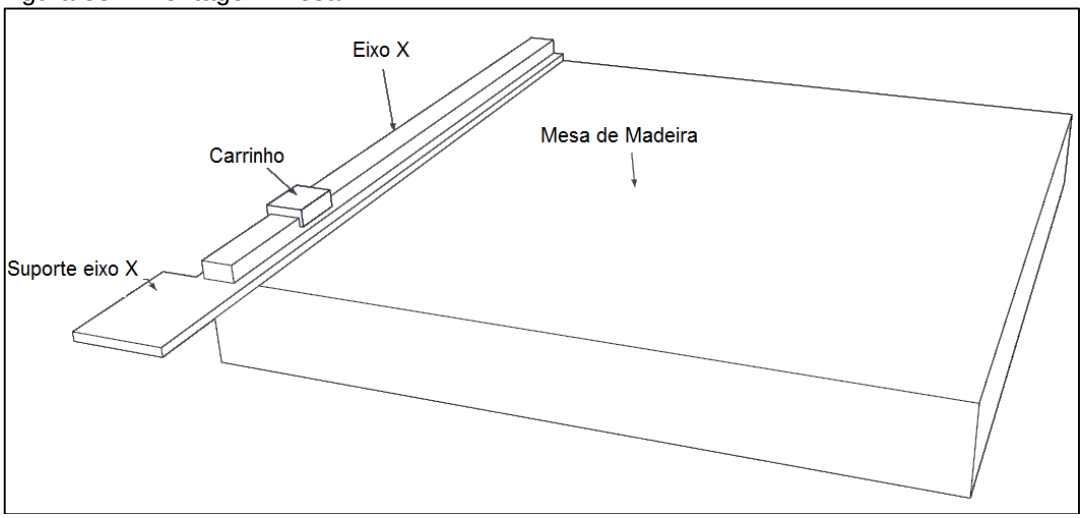
Para testar o *drive*r foi necessário construir um protótipo mecânico do sistema XY, este sistema irá acoplar dois motores de passo e uma cabeça, que será a referência do percurso dos motores.

3.1.2 Mesa XY mecânica

A parte mecânica que compõe a mesa foi projetada com o intuito de testar o *drive*r e o algoritmo de Bresenham. A confecção desta parte foi elaborada utilizando madeiras, ferros e algumas peças mecânicas como polias e correias. Um modelo foi desenhado, utilizando o software Sketchup Make[2], visando a construção parte a parte desta mesa. Utilizando uma madeira como base, foi adaptado um trilho mecânico HIWIN com carrinho, sobre uma base de ferro, que servirá como eixo x conforme a figura 38. Este conjunto HIWIN, figura 39, foi adquirido por doação, com o intuito de ajudar e facilitar a confecção do sistema.

[2] O SketchUp é uma ferramenta de modelagem 3D desenvolvido pela Google e atualmente é um produto da empresa Trimble com versões Make (gratuita) e Pro (profissional paga). Disponível em : http://www.sketchup.com/

Figura 38 – Montagem mesa



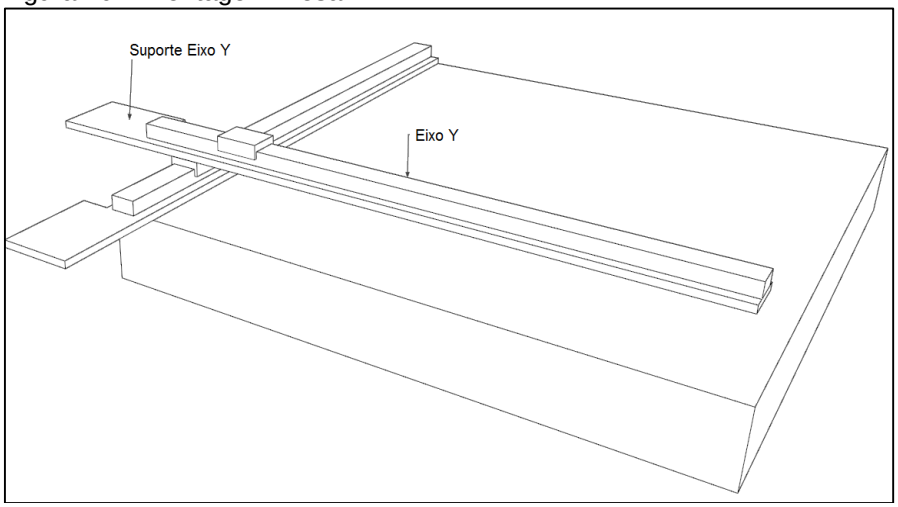
Fonte: Do autor.

Figura 39 – Trilho e carrinho HIWIN

Fonte: HIWIN (2014)

A utilização de um carrinho é exemplificada na figura 40, onde o suporte do eixo Y foi fixada no mesmo, para que quando movimentar o carrinho do eixo X, todo o eixo Y se movimente, alcançando toda a área da mesa. Ainda foi utilizando um trilho e um carrinho HIWIN sobre o novo suporte, para que este desloque a cabeça, que será o percussor de movimentação em ambos trilhos.

Figura 40 – Montagem mesa



Fonte: Do autor.

Com a posição dos eixos determinada, se objetiva a inclusão dos motores de passo, polias e correias aplicadas para que o *drive*r possa controla-los, também foi apontado a localização da cabeça de plotagem. A correia do eixo Y foi fixada na cabeça para sua movimentação, enquanto a correia do eixo X é fixada no suporte do eixo Y.

Os motores de passos escolhidos foram motores híbridos bipolares/unipolares de baixo porte, contendo até 2A de corrente por fase e deslocamento de 1,8° por passo. Ambos são da marca ElectroCraft, modelo TPP17, adquiridos por meio de compra (figura 41).

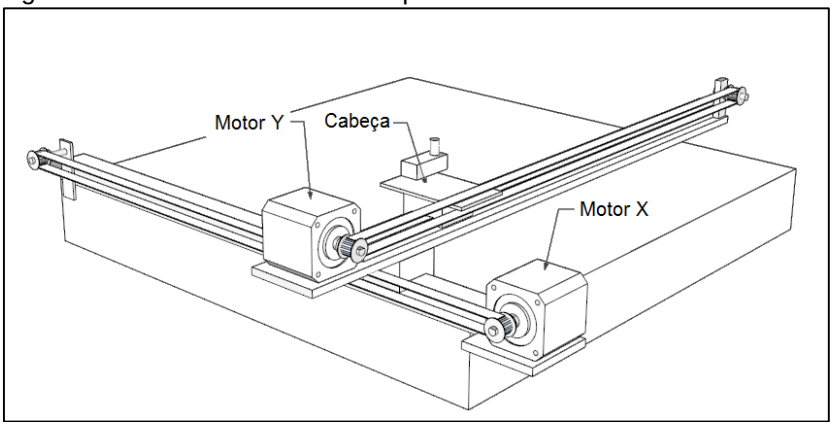
Figura 41 – Motor de passo TPP17

Fonte: ElectroCraft (2014)

O conjunto completo com motores, polias e correias foram acoplados de modo a atender uma melhor área de utilização da mesa. O modelo é mostrado na

figura 42, onde um motor exercerá controle sobre todo o eixo Y e outro sobre a cabeça.

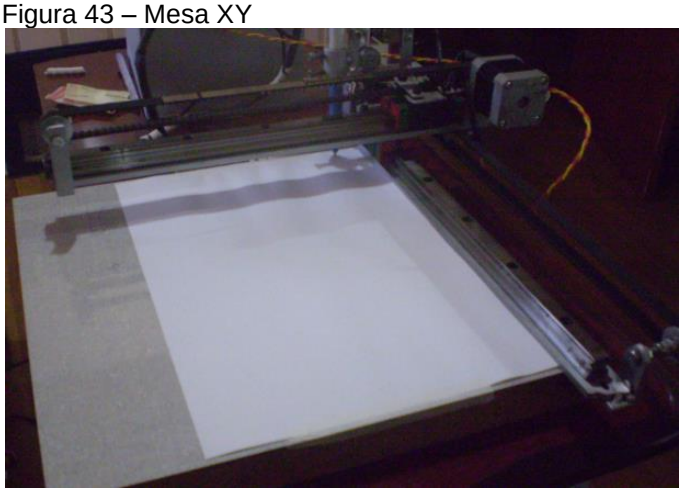
Figura 42 – Modelo mesa XY completo



Fonte: Do autor.

Na figura 43 a mesa foi objetivada contemplando algumas modificações em sua montagem, tais como, posição da cabeça e uma opção de levantar a mesma para que não ocorra a plotagem, caso for uma caneta levantar a mesma para que não desenhe.

Figura 43 – Mesa XY



Fonte: Do autor.

Com a mesa desenvolvida, foi possível iniciar o projeto do software no Arduino. Este software por sua vez controlará todo o sistema, recebendo os objetos por meio de um protocolo de comunicação, interpretando linhas e convertendo-as em pulsos de *clock* para que o *drive*r assimile, assim é possível a transformação de linhas em passos.

3.1.3 Programação no Arduino

Utilizando a IDE do Arduino, o software foi primeiramente baseado em estabelecer uma comunicação com um computador, que por meio deste envia os objetos a qual será interpretado e filtrado, para posterior comunicação com o *drive*r. Em outras palavras, foi necessário a implementação de um protocolo que regerá as regras de comunicação.

3.1.3.1 Protocolo de comunicação

O protocolo de comunicação limitou-se ao tamanho da memória do Arduino UNO, 2024 bytes de memória de dados. Reservando 524 bytes para outras variáveis, foi determinado que o protocolo deve possuir seis bytes em cada pacote, sendo demonstrado da seguinte forma:

Pacote: Comando hiX loX hiY loY Checksum

O pacote do protocolo pode ser explicado pela tabela 6:

Tabela 6 – Descrição dos bytes do pacote.

Byte	Nome	Descrição
1	Comando	Byte de controle que determinará algum comando no protocolo (ver tabela 7)
2	hiX	Byte mais significativo do valor X de um ponto
3	loX	Byte menos significativo do valor X de um ponto
4	hiY	Byte mais significativo do valor Y de um ponto
5	loY	Byte menos significativo do valor Y de um ponto
6	Checksum	Byte de verificação, para saber se o pacote está correto

Fonte: Do autor.

Devido às limitações de memória, o que sobra para o protocolo são 1500 bytes. Determinou-se que cada pacote corresponde a um ponto do objeto, então o mesmo é gravado, eliminando o byte de ckecksum, pois corresponde a validação dos bytes do pacote, e se este pacote estiver correto não há necessidade de gravar este validador, o que reduz a cinco bytes validos. Divide-se então, a memória resultante pelo tamanho dos bytes validos, resultando em um objeto de no máximo 300 pontos que pode ser gravado, antes de ser plotado.

Os comandos podem ser compreendidos como informações iniciais do pacote. É por meio deles, que os restantes dos bytes são interpretados. Foi implementado uma série de comandos, para organizar a comunicação, com o intuito de definir um conjunto de regras para o entendimento do protocolo. Comandos como linha, movimento e fim do objeto, são exemplos de comandos que tratam o comportamento da recepção de um objeto inteiro.

A tabela 7 exemplifica cada comando, byte correspondente e sua descrição.

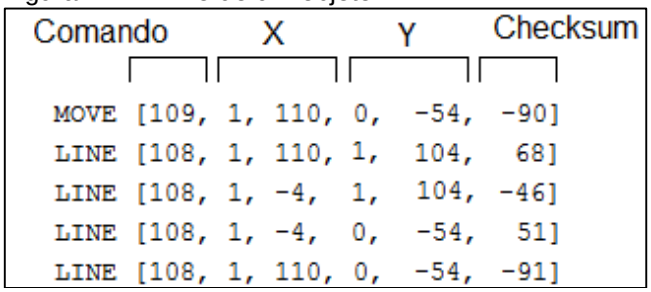
Tabela 7 – Lista de comandos

Comando	**Byte**	**Descrição**
REQUEST	114	Envio: Verifica se o Arduino está se comunicando
LINE	108	Envio: Movimenta plotando do ponto anterior ao ponto deste comando
MOVE	109	Envio: Movimenta não plotando do ponto anterior ao ponto deste comando
END	101	Envio: Fim do objeto, inicia a plotagem
ORIGEM	111	Envio: Retorna a cabeça a origem da mesa, ponto 0;0
SPEED	115	Envio: Determina a velocidade de plotagem dos motores
UP	56	Envio: Movimenta a cabeça para cima
LEFT	52	Envio: Movimenta a cabeça para a esquerda
DOWN	50	Envio: Movimenta a cabeça para baixo
RIGHT	54	Envio: Movimenta a cabeça para a direita
REQUEST_OK	107	Retorno: Arduino respondendo ao comando REQUEST
RESEND	114	Retorno: Arduino pedindo o reenvio do pacote, checksum invalido
NEXT	110	Retorno: Arduino pedindo o próximo pacote
DONE	100	Retorno: Arduino terminou de plotar o objeto
ORIGIN_OK	111	Retorno: Ponto de origem alcançado
SPEED_OK	115	Retorno: Velocidade alterada

Fonte: Do autor.

Alguns comandos se diferenciam de outros em relação aos seus parâmetros. Por exemplo, comandos como LINE, MOVE e SPEED reconhecem os parâmetros que são enviados no pacote. Os comandos LINE e MOVE, são comandos gravados na memória do Arduino e constituem os objetos, por sua vez, é necessário enviar também os valores dos pontos referentes ao mesmo. A figura 44, exemplifica o envio de um objeto contendo os comandos MOVE e LINE.

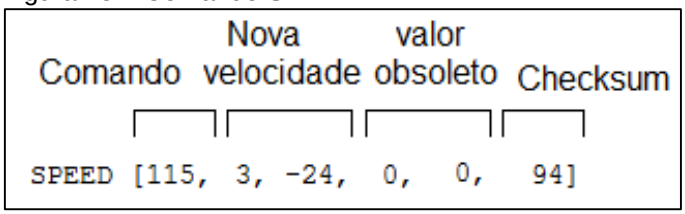
Figura 44 – Envio de um objeto



Fonte: Do autor.

No comando SPEED o parâmetro é somente a velocidade, acarretando a não utilização de dois bytes no pacote, conforme a figura 45.

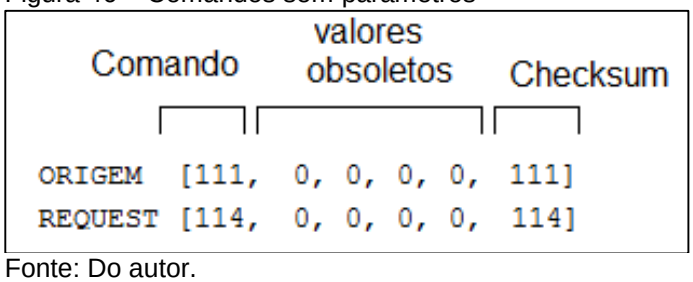
Figura 45 – Comando SPEED



Fonte: Do autor.

Os demais comandos possuem parâmetros obsoletos, ou seja, somente é necessário o envio do comando e seu *checksum*, porém ainda deve seguir a regra de seis bytes do protocolo. A figura 46 mostra como os comandos sem parâmetros funcionam.

Figura 46 – Comandos sem parâmetros



Fonte: Do autor.

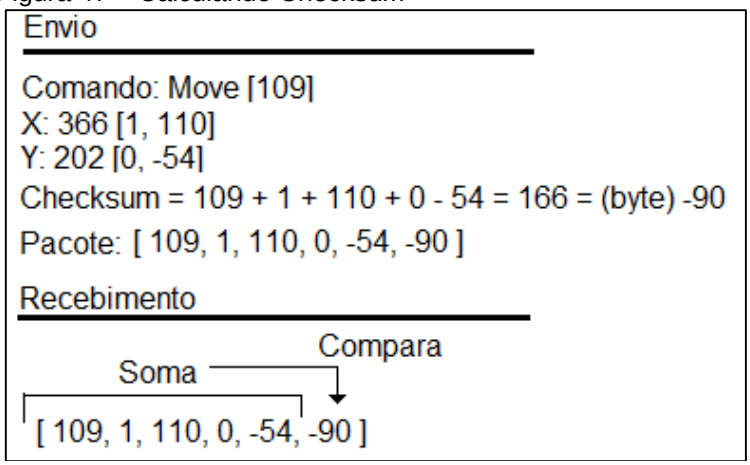
Ainda no protocolo, foi observado que existe uma limitação na área de plotagem do sistema. Cada LINE e MOVE possui um ponto contendo um X e um Y referente ao mesmo. Estes valores do ponto possuem tamanho de dois *bytes* ou 16 *bits*, e ao serem enviados são quebrados em partes significativas para contemplar a comunicação. Analisando estes tamanhos percebeu-se que existe uma resultante

significativa do comprimento pela largura, tal que, ao calcular abrange uma área de $2^{16}.2^{16} = 2^{32}\ passos^2\ ou\ pixels^2$. Com esta definição a área de plotagem se torna extremamente grande, limitando-se somente ao tamanho mecânico do sistema.

Independentemente de seu tamanho, para se enviar um objeto completo, foi necessário criar uma condição que garantisse o envio e o recebimento dos pacotes de forma correta. Em um protocolo de comunicação, existem vários meios de se garantir tal condição, mas o mais simples de ser implementado é um *checksum*, por se tratar de um pacote pequeno.

O *checksum* tanto citado, é um validador do pacote de envio, ou seja, é uma garantia que o envio é realmente recebido e que nenhum byte tenha se perdido no meio de comunicação. Para criar esta validação simplesmente é somado os cinco primeiros bytes do pacote em uma outra variável byte, posteriormente inserido no sexto byte e enviado. Quando o pacote é recebido, o Arduino soma os cinco bytes e compara com o sexto, validando-os ou não (figura 47).

Figura 47 – Calculando Checksum



Fonte: Do autor.

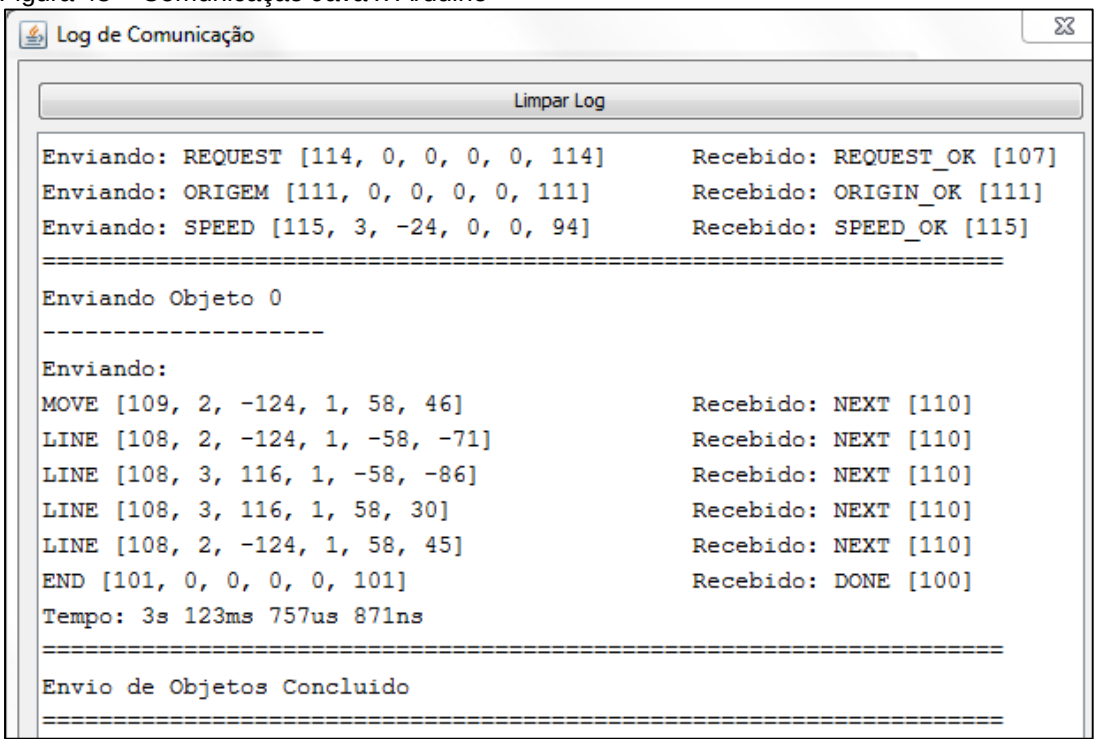
Ao receber os pacotes, o Arduino retorna somente um byte, correspondente ao comando recebido. Esta diferença entre pacotes de envio e de retorno, melhora a velocidade de comunicação, pois não há necessidade de agrupar os bytes dentro do Arduino antes do mesmo responder. Contudo o software do Arduino só reconhecerá pacotes de seis bytes consecutivos, dentro de um limite de tempo.

Este limite de tempo, *timeout*, resolveu problemas de comunicação ao enviar objetos de vinte pontos ou mais, que aleatoriamente geravam *checksum*

invalido, fazendo com que o Arduino não agrupasse os pacotes de forma correta, acarretando no pedido de muitos comandos RESEND's que bloqueavam a comunicação. O *timeout*, funciona quando por dois segundos, não é encontrado um *checksum* valido, assim reiniciando toda a comunicação, e pedindo novamente um RESEND.

Para testar a comunicação, foi elaborado um software em java, utilizando comunicação serial sobre a porta que o Arduino cria. Porem antes do teste ser executado, foi necessário definir algumas características de comunicação, como velocidade de comunicação e bits de dados enviados e recebidos, para tal, foi designado uma velocidade 57,6kbits por segundo e 8 bits de dados. Por sua vez, neste software foi implementado o protocolo, seus comandos e a definição dos pacotes de envio, com seis bytes e *checksum*. A figura 48 mostra toda comunicação com o Arduino ao enviar um objeto.

Figura 48 – Comunicação Java x Arduino



Fonte: Do autor.

Com o protocolo pronto e implementando no Arduino, foi possível carregar objetos para que posteriormente possam ser plotados. Para esta etapa o Arduino precisa ler o objeto carregado e transforma-lo em passos para acionar o *drive*r PBL 3770, neste caso foi necessário transformar o objeto em segmentos, de dois pontos

cada, assim criando uma reta que pode ser manipulada pelo algoritmo de Bresenham.

3.1.3.2 Algoritmo de Bresenham

Para implementar o algoritmo de Bresenham no Arduino, foi necessário organizar a leitura do objeto carregado, e compreender como o algoritmo reconhecerá os parâmetros. Para tal, como o carregamento foi feito em cinco bytes por pacote, foi intuitivo que a leitura seria também em cinco bytes, porém ao analizar o algoritmo descobriu-se que para plotar um pixel, são necessário dois pontos, um de origem e outro de destino.

A cada leitura foi essencial transformar os pontos em segmentos, mas estes não podiam ser criados, pois consumiriam a memória reservada do Arduino. Para resolver este problema é carregado o primeiro ponto, em relação a uma origem, normalmente [0, 0] ou uma localização fixa na mesa, então o algoritmo é utilizado para traçar uma reta entre estes dois pontos. Após traçar a reta, o ponto de relação recebe os valores do anterior, continuando com a leitura do objeto.

Como é gravado também os comandos MOVE e LINE junto aos pontos, foi inserido uma condição antes de traçar a reta, que permite levantar ou abaixar a plotadora quando estes são encontrados.

O algoritmo de Bresenham aqui apresentando (figura 28, capitulo 2.5, p 51) foi insuficiente para tratar todos os trajetos de retas possíveis, ou seja, não consegue plotar quando o X do ponto de origem for maior que o X do ponto de destino. Além de não ser possível a implementação de recursão dentro do Arduino UNO. Para tanto, foi necessário reelaborar a estrutura do algoritmo e eliminar a recursão, visando a troca de valores caso encontre origens maiores que destinos. Esta nova estrutura foi posteriormente testada, de forma que conseguisse plotar os segmentos em todas as direções, compreendido no apêndice C.

Até o momento, se possui o *drive*r de motor de passo, a mesa xy, um protocolo, que possibilita a comunicação para qualquer software que o implemente, e o algoritmo para plotagem das retas. Contudo, resta transformar a plotagem de retas em *clocks* e sentido de giro para acionar o *drive*r.

3.1.3.3 Controle de pulsos

O controle de pulsos serve para transformar o algoritmo de Bresenham de plotagem de pixels em *clocks*. No Bresenham existe um método a qual insere o pixel em determinado ponto, chamado de “put_Pixel”. Este método por sua vez recebe as coordenadas do pixel a ser inserido que são anteriormente manipuladas pelo algoritmo.

Foi substituído este método por um outro chamado “executaPassos”, que também recebe como parâmetro uma coordenada de um ponto. Porém o “executaPassos” difere em relação ao “put_Pixel”, pois ao se inserir os parâmetros, a nova coordenada deve ser subtraída da coordenada anterior, para saber qual é variação de deslocamento. Esta variação traz consigo o sentido de giro e a possível execução do passo.

Como o algoritmo trabalha com deslocamento de pixel, a variação é entre -1 e 1, em inteiros. Analisando este intervalo, conseguiu-se traduzir estes valores, sendo que:

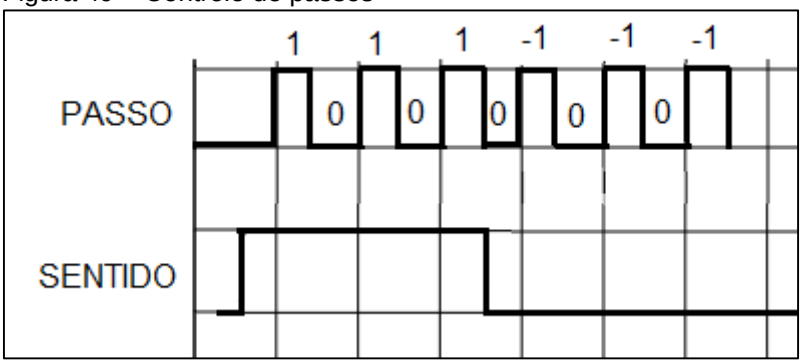
a) 1 executa passos com sentido de giro horário;

b) -1 executa passos com sentido de giro anti-horário;

c) 0 (zero) não executa passos.

Após implementar, o controle de passos foi testado por meio da simulação no osciloscópio do Proteus, conforme a figura 49.

Figura 49 – Controle de passos



Fonte: Do autor.

Com todas as partes do sistema XY elaboradas foi possível a execução dos testes e correções, permitindo analisar o projeto como um todo, desde as partes

eletrônicas, mecânicas e computacionais, resultando em uma melhor concepção da abrangência que automação industrial pode disponibilizar.

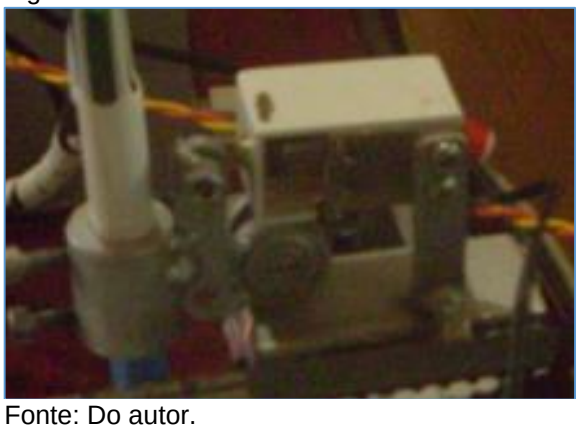
3.1.4 Testes e correções

Anteriormente aos testes, foi desenvolvido um software em java para que este implemente o protocolo de comunicação e aborde alguns recursos gráficos, como criação de polígonos por meio de pontos, com o intuito de testar totalmente o sistema.

Um primeiro problema a ser percebido, foi que o sistema não possui uma fonte de alimentação para o circuito do driver, pois este foi projetado para trabalhar em 20V com 2A de corrente. Um circuito de fonte foi desenvolvido para atender estes requisitos, por meio do Proteus (apêndice D), e foi confeccionado, assim possibilitando o início dos testes.

O primeiro teste a ser executado foi um quadrado, e durante a plotagem foi necessário adaptar um sistema de levantamento da cabeça plotadora, que quando for encontrado o comando MOVE, levante-a. Para tal problema foi adaptado uma solenóide de 20V, adquirida por compra, (figura 49) e alterado o código no Arduino.

Figura 50 – Solenóide



Fonte: Do autor.

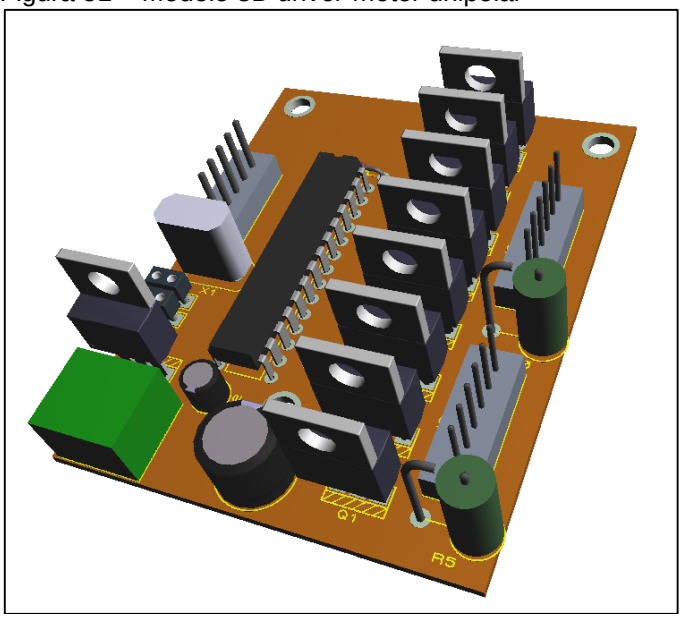
Após adaptada a solenóide, foi refeito o teste com o quadrado, funcionando perfeitamente. Porém ao testar o sistema plotando o mesmo quadrado rotacionado alguns graus, percebeu-se que o *drive*r não conseguia executar os passos de forma correta, desenhando o objeto de forma irregular com distorções.

Analisando a simulação da plotagem deste objeto no Proteus, foi percebido que o *drive*r ao deslocar os motores, aleatoriamente inverte o sentido de um ou de outro, ocasionando a distorção. Foi tentado alterar o *firmware* do *drive*r, e inserir novos PBL 3770, porém estas alterações não resolveram o problema da inversão de sentido indesejada.

Um último recurso, foi confeccionar um novo *drive*r para os motores. Para tal, foi decido construir um *drive*r de controle de motores de passos unipolares, já que os motores adquiridos funcionam como unipolares/bipolares. Este novo *drive*r foi desenvolvido de modo que se comporte semelhante ao anterior, em relação aos sinais de entrada. Contudo para o acionamento dos motores foi feito um sistema com transistores (figura 15, capitulo 2.4.3, p35) que aciona cada bobina independente. Para os passos, foi utilizado sistema *half-step* que garante maior quantidade de passos, e o controle do mesmo pode ser observado na tabela 3, capitulo 2.4.3, p34.

Com estas definições foi implementando o novo *firmware,* utilizando o mesmo PIC, já que pode ser regravado. Também foi reaproveitado alguns componentes da placa anterior, como conectores. Por fim, foi projetado o novo *drive*r no Proteus, confeccionando a nova placa, figura 51, e testado plotando os objetos quadrados, por sua vez foi obtido um retorno adequado. O esquema e layout deste driver pode ser compreendido no apêndice E e F.

Figura 51 – Modelo 3D *driver* motor unipolar



Fonte: Do autor.

Uma outra adaptação foi percebida por não existir uma origem de plotagem na mesa, ou seja, não há ponto de referência em que os motores possam iniciar, como consequência ao ligar o sistema, a localização do ponto de origem será onde a cabeça está situada. Para resolver este problema, foi necessário inserir sensores que situam a posição de referência. Foi adquirido componentes por meio de compra, que são pequenos sensores ópticos por barreira, e são baratos e fáceis de adquirir. Estes sensores, foram adaptados em um canto do sistema, onde foi determinado a referência da mesa.

Após as correções feitas, foi determinado testar a eficácia do algoritmo de Bresenham, em relação a um de ponto flutuante. Para este procedimento, foi estudado e implementado um algoritmo baseado em cálculos de retas (apêndice G), que consiga satisfazer a mesma plotagem que o de Bresenham executa.

3.1.4.1 Comparação com algoritmo flutuante para plotagem de retas

Com os algoritmos implementados, foi definido um conjunto de objetos para serem comparados. Foi estipulado uma média de cinco plotagens cada, para ambos os algoritmos. Os objetos, apêndice H, foram desenhados no software em java desenvolvido para testes e selecionados para a plotagem. A tabela 8 traz os tempos de cada plotagem, mostrando um percentual de eficácia que o algoritmo de Bresenham conseguiu superar em relação ao de ponto flutuante.

Tabela 8 – Eficácia do algoritmo de Bresenham em relação a um de ponto flutuante

Velocidade de plotagem = 1000	Tempo de plotagem do algoritmo em segundos		
Objetos	**Bresenham**	**Flutuante**	**% de Eficácia**
Quadrado	5,210212489	5,416391621	3,96
Triangulo	4,133891307	4,275926132	3,44
Espiral (25 pontos)	9,500472344	9,924888997	4,47
Texto (74 pontos)	16,04793689	17,23144316	7,37
Círculo (50 pontos)	4,137161061	4,696969329	13,53
Média:	7,805934818	8,309123848	6,45

Fonte: Do autor.

Como foi percebido, o algoritmo de Bresenham é pouco eficaz quando os objetos possuem uma quantidade baixa de pontos, porem ao aumentar a quantidade dos mesmos, ocorreu uma significativa diferença de tempo entre as amostras. Uma

observação relevante, é que os objetos não estavam na mesma escala, resultando em uma diferença entre tempos e quantidade de pontos, como exemplo o círculo e o texto. Outros testes foram executados, com finalidade de tentar encontrar alguma outra falha, entretanto não foram encontrados erros que comprometessem o projeto.

Finalizando o projeto, foi organizado uma caixa para a parte de controle do sistema, esta caixa possui toda a eletrônica envolvida, desde o Arduino, a fonte e o novo driver. O intuito de organizar é possibilitar a separação da automação, conceituadas no inicio em partes de controle, sensoriamento e atuadores. Também facilitou no reconhecimento dos recursos utilizados, de modo que estes, são de relevância na confecção do sistema, pois abrangem grande parte da produção do projeto. A maioria destes recursos foram adquiridos por meio de compra, como componentes eletrônicos e elétricos, e outros por doação, parte mecânica. A tabela 9 mostra os recursos de cada parte do projeto, porém descarta a parte de confecção do driver PBL3770, pois não fez parte do projeto finalizado.

Tabela 9 – Recursos de hardware

Drive motor unipolar				
Qnt	Comp.	Fornecedor	Preço unidade	Preço total
12	Resistor 1k	FANCOMP	0,01	0,12
2	Capacitor 33pF 50V	FANCOMP	0,02	0,04
7	Resistor 10k	FANCOMP	0,01	0,07
1	Capacitor 100nF 50V	FANCOMP	0,05	0,05
1	Capacitor 47uF 25V	FANCOMP	0,035	0,035
1	Capacitor 470uF 35V	FANCOMP	0,23	0,23
1	Cristal 12Mhz	FANCOMP	0,25	0,25
1	LM7805	FANCOMP	0,58	0,58
2	Resistor 0.22/ 5W	FANCOMP	0,25	0,5
8	TIP122	FANCOMP	0,6	4,8
1	PIC16F870 - I/SP	FANCOMP	9,8	9,8
2	Conector kk 6 vias 2.54	FANCOMP	0,25	0,5
1	Conector kk 5 vias 2.54	FANCOMP	0,25	0,25
1	Conector TBLOCK 2 vias	FANCOMP	0,65	0,65
1	Soquete 28 pinos	FANCOMP	0,18	0,18
			TOTAL (R$)	18,055
Fonte 20V				
Qnt	Comp.	Fornecedor	Preço unidade	Preço total
1	Transformador 12+12 / 4A	SOLDAFRIA	30	30
1	Capacitor 4700uF 35V	FANCOMP	1,6	1,6
1	Capacitor 1000uF 35V	FANCOMP	0,5	0,5
1	LED 3mm vermelho	FANCOMP	0,08	0,08
1	LED 3mm verde	FANCOMP	0,085	0,085
1	Resistor 10k	FANCOMP	0,01	0,01
1	Resistor 1k	FANCOMP	0,01	0,01
2	Diodos 6A10	FANCOMP	0,28	0,56
1	TIP122	FANCOMP	0,6	0,6
1	Conector TBLOCK 2 vias	FANCOMP	0,65	0,65
1	Conector TBLOCK 3 vias	FANCOMP	0,75	0,75
1	Conector kk 2 vias 2.54	FANCOMP	0,25	0,25
1	Conector kk 2 vias 5.08	FANCOMP	0,4	0,4
			TOTAL (R$)	35,495
Diversos				
Qnt	Comp.	Fornecedor	Preço unidade	Preço total
1	Arduino UNO	FILIPEFLOP	65	65
2	Motor de passo TPP17	EBAY	19,35	38,7
2	Sensores ópticos TCST1103	FANCOMP	0,75	1,5
2	Placa fenolite simples 10 x 10	SOLDAFRIA	1,39	2,78
1	Solenoide 20V	SOLDAFRIA	4,4	4,4
			TOTAL (R$)	112,38
TOTAL (R$) 165,93				

Fonte: Do autor.

3.2 RESULTADOS OBTIDOS

O trabalho desenvolvido, apresenta a implementação completa do driver de controle de motor de passo, que possibilita o controle dos dois motores, X e Y, de forma eletronicamente simplificada.

A confecção da mesa para testes foi completada, porém foram necessárias algumas adequações que melhoraram o sistema, tal como, a opção de levantar e abaixar a plotadora e fixar um sistema de localização de origem.

O protocolo de comunicação, foi o conceito mais elaborado a ser projetado, o que levou a diversos estudos para a redução do tamanho do pacote, de forma que, ainda assim garantisse um bom desempenho na comunicação. Também permite que qualquer software o implemente, devido a facilidade do uso das suas regras.

A funcionalidade de maior interesse, foi visualizar fisicamente o funcionamento do algoritmo de Bresenham, proporcionando um melhor estudo sobre sua lógica, além de garantir um bom desempenho de processamento dos passos.

4 CONCLUSÃO

O trabalho de conclusão de curso aproximou o autor a área de automação industrial, e possibilitou ao mesmo conhecer funcionamentos das diferentes etapas de um processo automatizado. Essa aproximação construiu conhecimentos acerca da integração entre *hardware* e *software*, que possibilitaram a confecção do sistema como um todo.

O processo de confeccionar, foi um obstáculo nas partes eletrônica e mecânica. O estudo de funcionamento dos circuitos integrados envolvidos no *driver* acarretaram dificuldade ao tentar implementar um *firmware* para controlar os mesmos, contudo a elaboração de um *driver* mais simplificado foi viável. Também ocorreram diversas falhas na projeção da mesa mecânica, sendo que, foram necessárias três mesas para alcançar uma que satisfizesse a eletrônica envolvida.

Em analise, a confecção deste sistema traz a liberdade de utilizar motores de passo de várias potências, além de permitir a escalabilidade da área mecânica da mesa e a elaboração de indeterminados softwares que consigam implementar o protocolo. Porém ainda é de difícil entendimento, pois há necessidade da compreensão em comunicação de dados, e conhecimento na implementação de softwares que abrangem tais comunicações.

Para um projeto de maior complexidade seria desenvolver o software controlador, esse por sua vez, implementaria o protocolo e algoritmos de eficiência gráfica, que otimizariam a área de plotagem, encaixando automaticamente os objetos a serem plotados, de forma a ocuparem uma maior área útil.

O custo do controle eletrônico, pode ser reduzido ao trocar o PIC do *driver* por um de custo menor, porém com características parecidas. Uma outra alternativa é elaborar uma placa dedicada envolvendo todo o conjunto de controle e eliminando o valor total do Arduino UNO, resultando somente a utilização de seu microcontrolador.

Uma das observações foi no limite de passos que os motores podem executar. Em *half-step* os motores escolhidos alcançam 400 passos por volta, limitando a precisão. Porém é possível a multiplicação destes passos, por meio de circuitos que gerenciam a corrente nas bobinas do motor. Estes passos que vão

além da capacidade física do motor, são conhecidos como micropasso e necessitam de um elaborado estudo para sua implementação.

Uma outra melhoria, seria incorporar a utilização de chips de memória, que aumentariam a quantidade de pontos a serem plotados, além de permitir o transporte dos arquivos. Para isso, é necessário conhecer as formas de comunicação que implementam tais gravações e leituras.

Por fim, o trabalho de conclusão de curso relacionou características computacionais e industriais, tal que, foi possível o aprimoramento do processo de automatização.

REFERÊNCIAS

ALVES. Toni dos Santos. **Automação Industrial I**. Abrantes: Escola Superior de Tecnologia de Abrantes, 2005. 55 p.

BIM, Edson. **Máquinas Elétricas e Acionamento**: uma introdução. Campinas: UNICAMP-Faculdade de Engenharia Elétrica e Computação, 2009.

BRESENHAM, Jack. **Algorithm for computer control of a digital plotter**. IBM, Systems Journal, 4(1): 25–30, 1965.

BRITES, Felipe Gonçalves; SANTOS, Vinicius Puga de Almeida. **Motor de Passo.** Niterói: UFF, 2008. 15 p.

CAPELLI, Alexandre. **SENSORES INDUSTRIAIS**: FUNCIONAMENTO E APLICAÇÕES PRÁTICAS EM CAMPO. 1. Rio de Janeiro: ANTENNA, 2000. 70p.

CARVALHO, Geraldo. **MÁQUINAS ELETRICAS**: Teoria e Ensaios. 4.ed. São Paulo: Érica, 2011.

CORMEN, Thomas H. et al. **ALGORITMOS:** Teoria e Prática. Tradução Arlete Simille Marques. 3. ed. Rio de Janeiro: Elsevier, 2012.

Ericsson Microelectronics. **PBL 3770A**: High Performance Stepper Motor *Drive* Circuit. Fev. 1999. Disponível em: <http://pdf.datasheetcatalog.com/datasheets/150/35361_DS.pdf>. Acesso em: 07 out. 2013.

FIGUEIREDO, Bruno Oliveira Dourado Arruda de. **MELHORIA DA QUALIDADE DE PROCESSOS INDUSTRIAIS ATRAVÉS DA AUTOMAÇÃO E DA REPROGRAMAÇÃO DO SISTEMA SUPERVISÓRIO.** 2010. 61 f. Monografia (1) – Departamento de Centro de Tecnologia, Universidade Federal do Ceará, Fortaleza, 2010.

FITZGERALD, A.E.; KINGSLEY, Charles, Jr.; UMANS, Stephen D. **MÁQUINAS ELÉTRICAS**: Com introdução a eletrônica de potência. 6.ed. Porto Alegre: Bookman, 2006.

FOROUZAN. Behrouz A. **Comunicação de Dados e Redes de Computadores**. 3. São Paulo: artmed, 2006. 840 p

GROOVER, Mikell P. **AUTOMAÇÃO INDUSTRIAL E SISTEMAS DE MANUFATURA**. 3. ed. São Paulo: Pearson Prentice HALL, 2011.

MARTINS. Geomar Machado. **Princípios de Automação Industrial**. Santa Maria: UFSM, 2007. 215 p.

MCROBERTS. Michael. **Arduino Básico**. 1. São Paulo: Novatec, 2011. 453 p.

Microchip Technology Inc. **HI-TECH C Compiler for PIC10/12/16**: MCUs Version 9.83 Release Notes. Set 2011. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/readme_PICC_983.pdf>. Acessado em 5 nov. 2013.

Microchip Technology Inc. **PIC16F870/871 Datasheet.** 2003. Disponível em: <http://ww1.microchip.com/downloads/en/devicedoc/30569b.pdf>. Acesso em: 04 nov. 2013.

MONK. Simon. **30 Arduino Projects For The Evil Genius**. 1. New York: McGraw-Hill, 2010. 191 p.

MORAES, Cícero Couto de; CASTRUCCI P. de L. **Engenharia de Automação Industrial**. 2ª Ed. Rio de janeiro: LTC, 2007.

MORO, Jefferson Zortea. **MICROCONTROLADORES PARA TRAÇAR RETAS EM LCDs**. Vitória: Departamento de Engenharia Elétrica - Universidade Federal do Espírito Santo, 2009. 6p.

PEREIRA, Fábio. **PIC**: Programação em C. 6.ed. São Paulo: ÉRICA, 2007.

QUEIROZ, Ricardo Alexandro de Andrade. **Motores de Passo**. Salvador: Unifacs, 2002. 19 p.

ROSÁRIO, J. M., **Princípios da Mecatrônica**. 1ª Ed. São Paulo: Prentice-Hall, 2005.

SANTOS, Nuno. **Introdução ao Arduino**. Revista PROGRAMAR, 17.ed, Portugal, 2008.

SANTOS FILHO, Sebastião G. dos (Org.). **CONTROLE DE MESA XY**: utilizando motor de passo. Mecatrônica Atual**,** São Paulo, v. 1, n. 2, p.28-35, jan. 2002.

SILVEIRA, Paulo R.da, SANTOS, Winderson E. **Automação e Controle Discreto.** Editora Érica, São Paulo, 1998.

SOUZA, David José. **Desbravando o PIC: Ampliado e Atualizado para PIC 16F628A**. 11ª ed. São Paulo, SP, Brasil: Érica, 2007.

SOUZA, Rodrigo Barbosa de; **Uma Arquitetura para Sistemas Supervisórios Industriais e sua Aplicação em Processos de Elevação Artificial de Petróleo**, Dissertação de Mestrado, Universidade Federal do Rio Grande do Norte – UFRN, 2005.

TANENBAUM. Andrew. **Computer Networks**. 4. Amsterdam: Campus, 2002. 912 p.

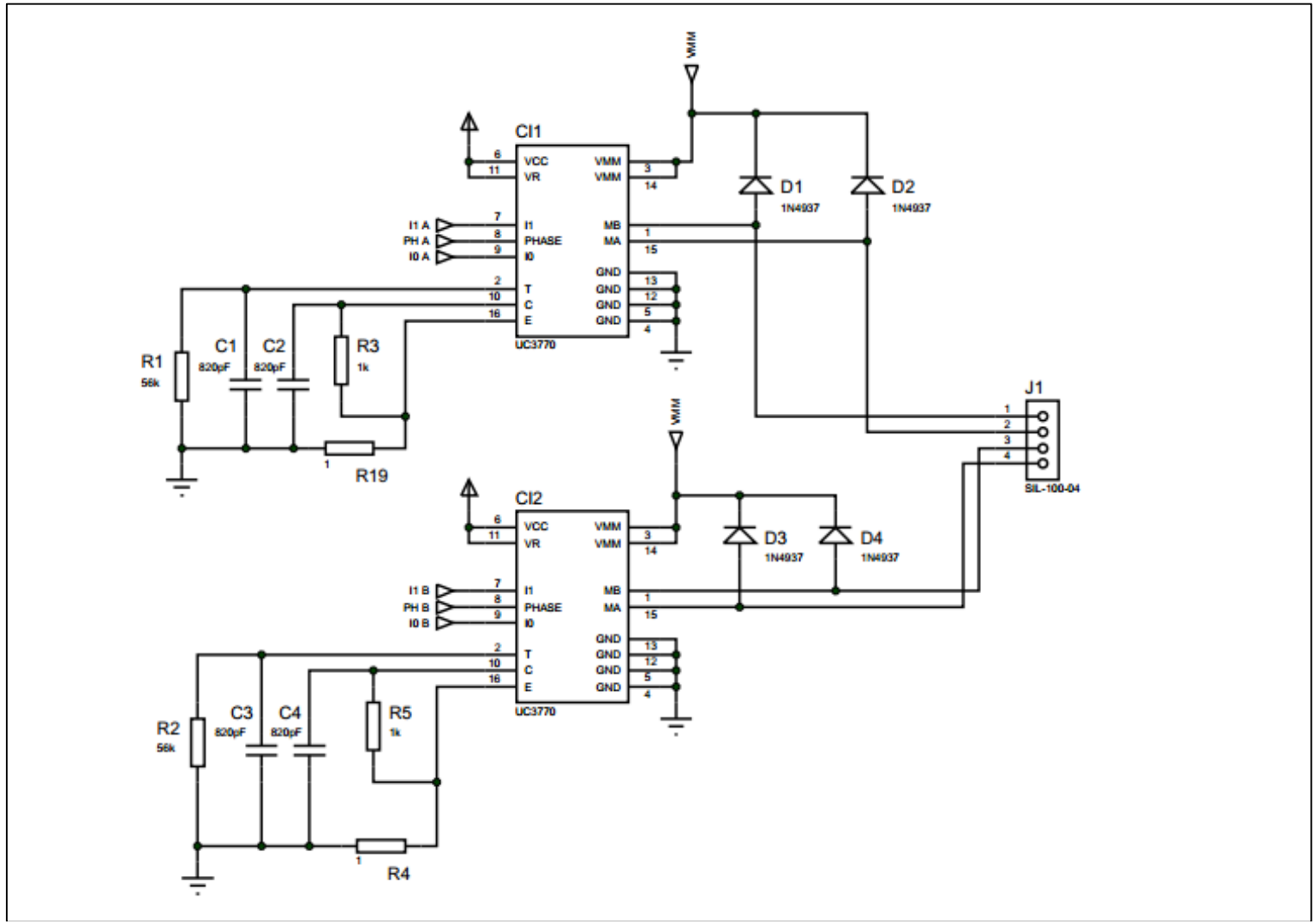
THOMAZINI, Daniel; ALBUQUERQUE Pedro Urbano Braga de. **Sensores Industriais**: Fundamentos e Aplicações. 4.ed. São Paulo: Érica, 2005.

TOCCI, Ronald J.; WIDMER, Neal S. **Sistemas Digitais**: Princípios e Aplicações. 8. ed. São Paulo: Pearson, 2004.

TOLSTOY, Ivan. **James Clerk Maxwell: A Biography**. Chicago: University of Chicago Press, 1982.

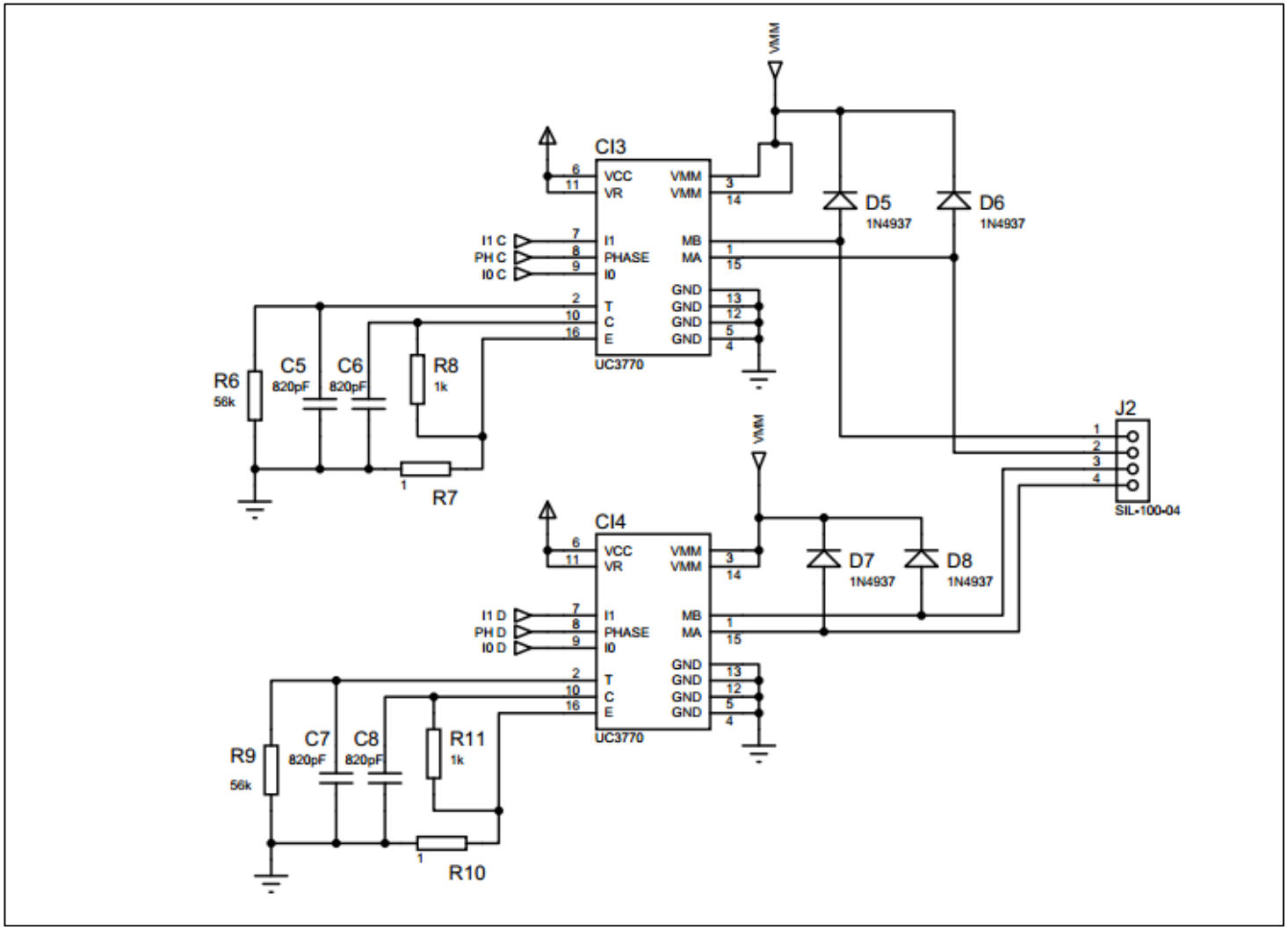
APÊNDICE(S)

APÊNDICE A – *Drive* PBL 3770 Motor X



Fonte: Do autor.

APÊNDICE B – *Drive* PBL 3770 Motor Y



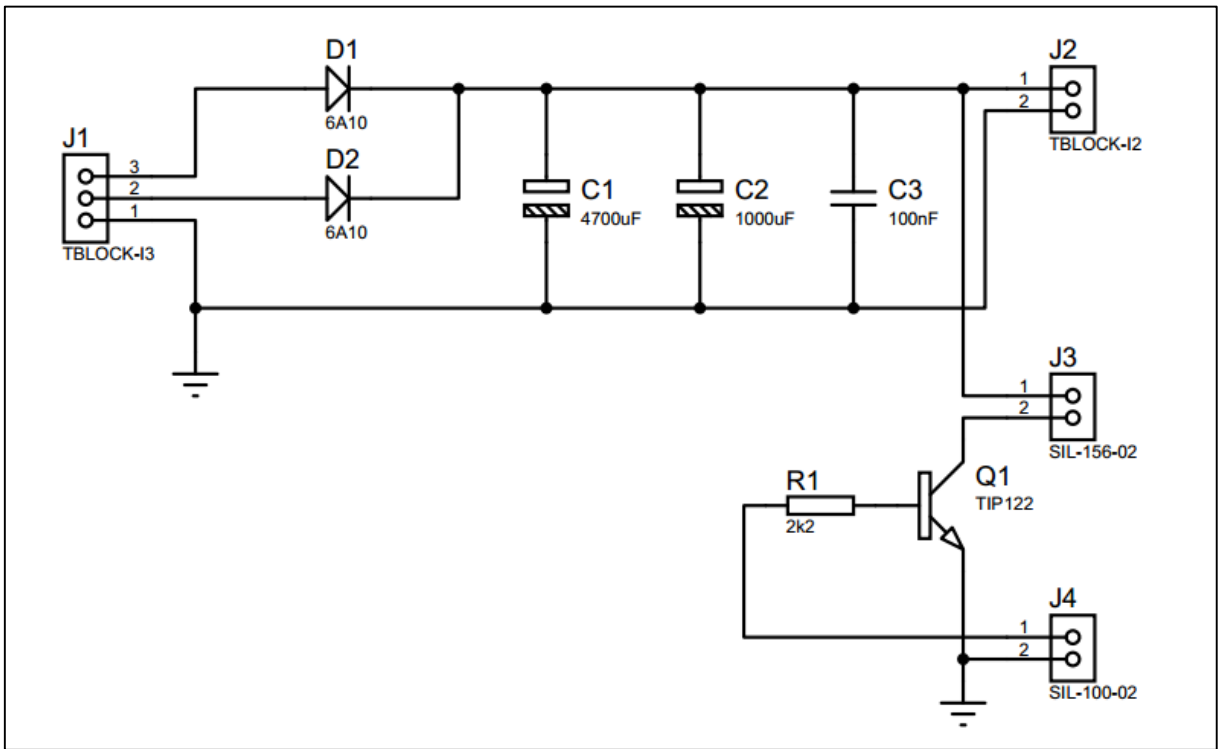
Fonte: Do autor.

APÊNDICE C – Bresenham para Arduino

```
void drawLine_Bresenham( int x1,  int y1,  int x2,  int y2)          {
        int dx, dy, incY, incX;
        int d, incE, incNE;
        int x, y;

        x = x1;
        y = y1;

        if ( x2 >= x1)     {
                dx = x2 - x1;
                incX = 1;
        }       else      {
                dx = x1 - x2;
                incX = -1;
        }

        if( y2 >= y1)      {
                dy = y2 - y1;
                incY = 1;
        }       else      {
                dy = y1 - y2;
                incY = -1;
        }

        if(dx >= dy)       {

                d = (2 * dy - dx);
                incE = 2 * dy;
                incNE = 2 * (dy -dx);

                while(x1 != x2)  {
                        if(d <= 0)          {
                                d = d + incE;
                                x1 = x1 + incX;
                        }       else      {
                                d = d + incNE;
                                x1 = x1 + incX;
```

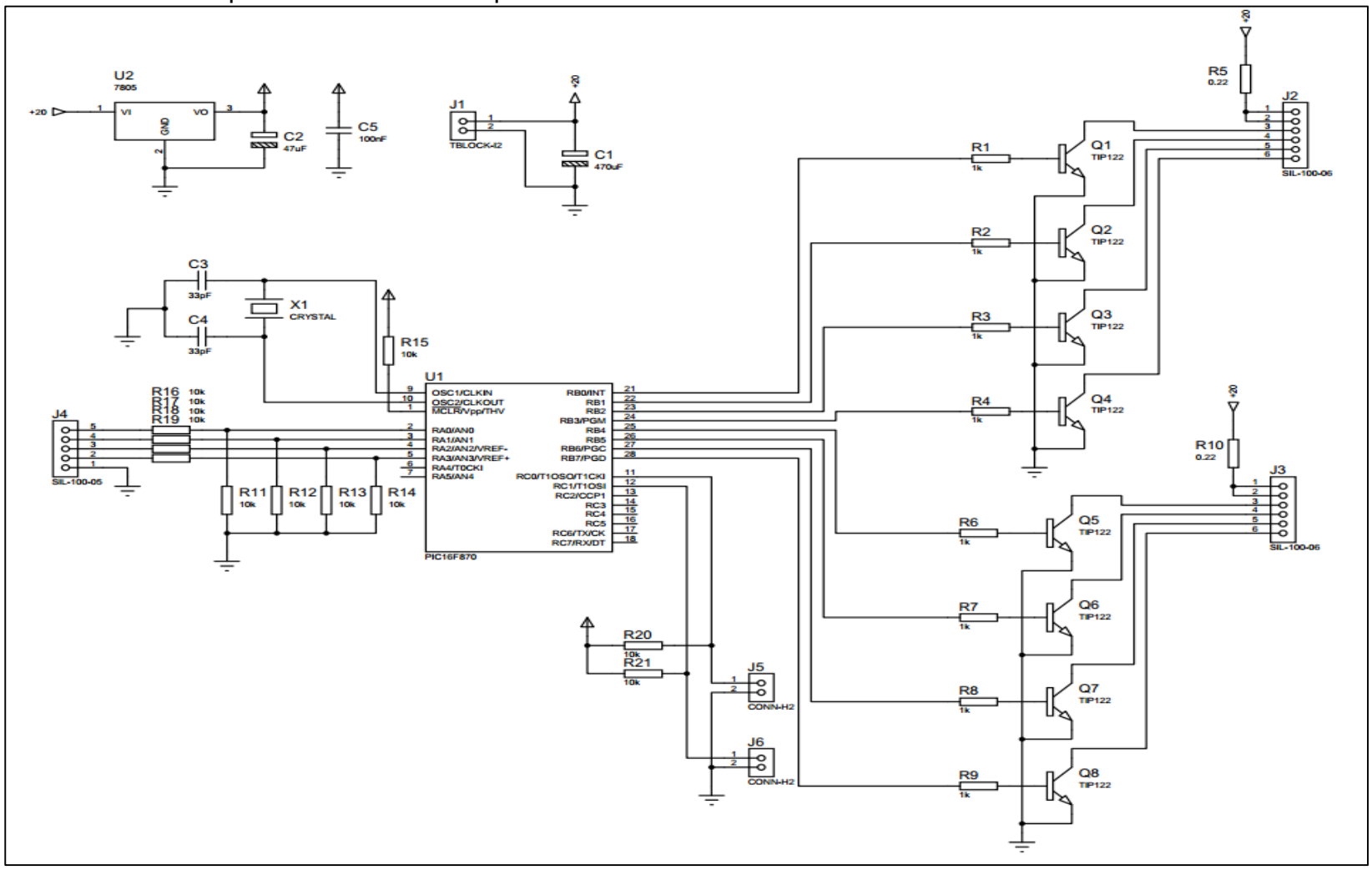
```
                        y1 = y1 + incY;
                }
                //plota pixel
                put_Pixel (x1, y1);
                x = x1;
                y = y1;
        }
    }       else    {

        d = (2 * dx - dy);
        incE = 2 * dx;
        incNE = 2 * (dx - dy);

        while(y1 != y2)  {
                if(d <= 0)          {
                        d = d + incE;
                        y1 = y1 + incY;
                }       else    {
                        d = d + incNE;
                        y1 = y1 + incY;
                        x1 = x1 + incX;
                }
                //plota pixel
                put_Pixel (x1, y1);
                x = x1;
                y = y1;
        }
    }
}
```

APÊNDICE D – Fonte 20V



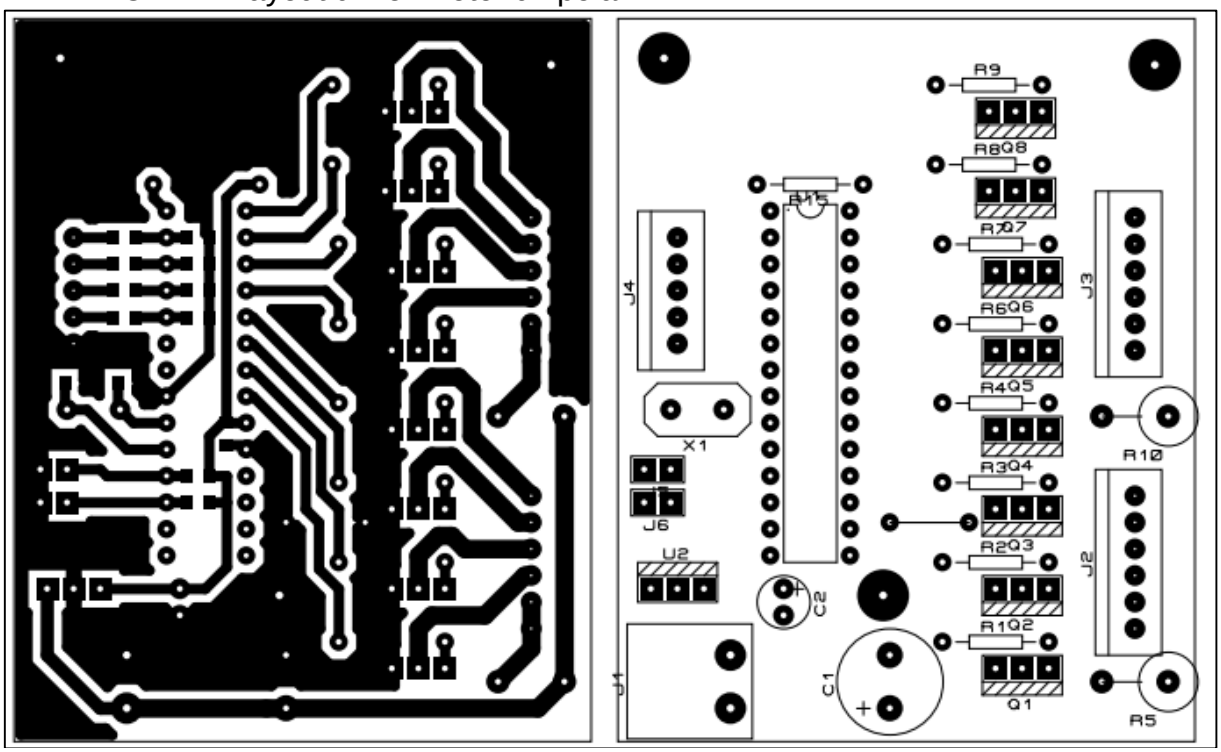
Fonte: Do autor.

APÊNDICE E - Esquema driver motor unipolar



Fonte: Do autor.

APÊNDICE F – Layout driver motor unipolar

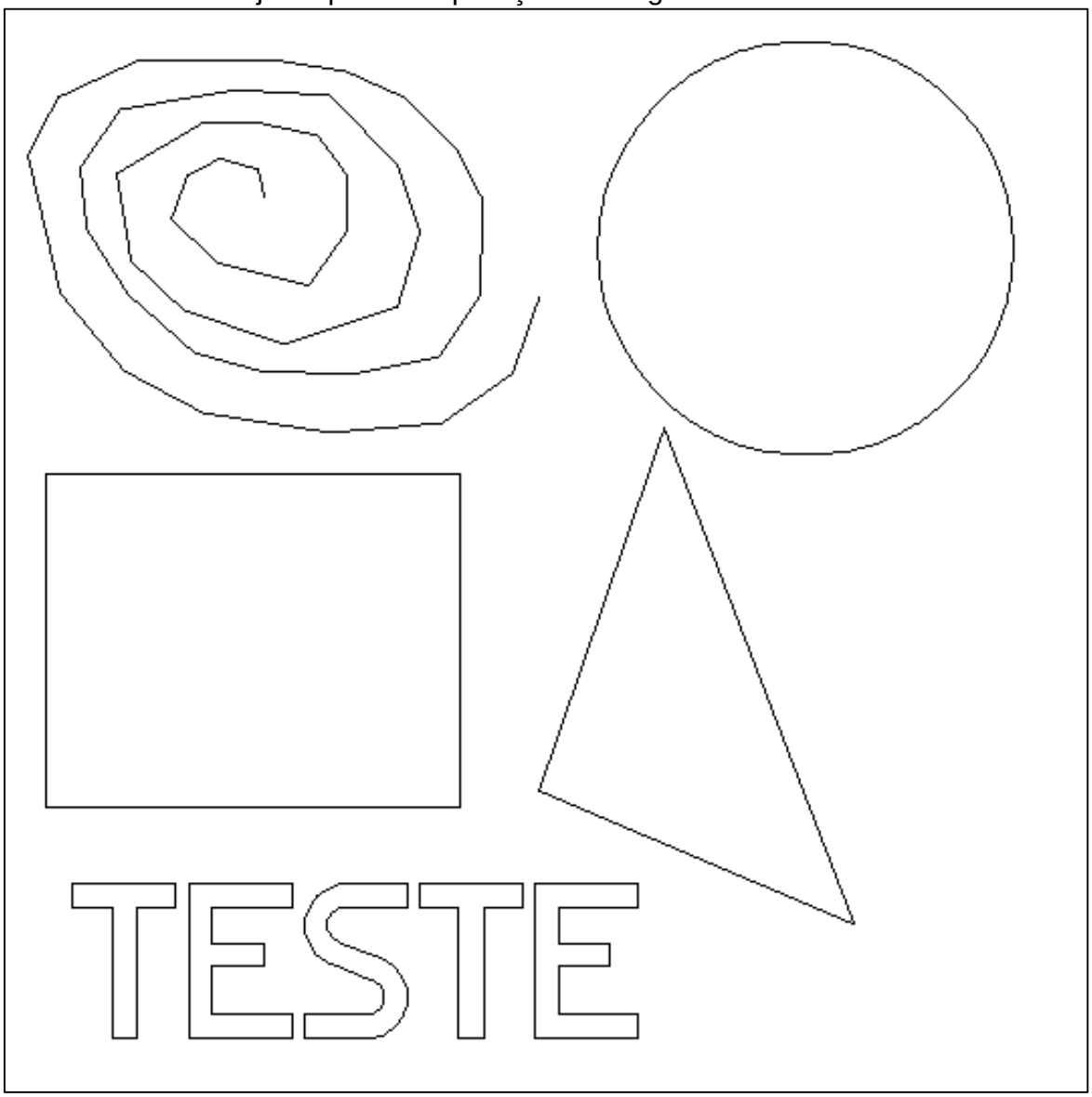


Fonte: Do autor.

APÊNDICE G – Algoritmo ponto flutuante

```
void drawLine_Float(unsigned int x1, unsigned int y1, unsigned int x2, unsigned int y2) {
        int dx = x2 - x1;
        int dy = y2 - y1;
        int x , y;
        int oldX = x1, oldY = y1;

        if(dx == 0 && dy == 0) return;

        if( abs(dx) >= abs(dy) )  {
                int incX = dx < 0 ? -1 : 1;
                float dydx = (float)dy / (float)dx;

                x = x1;

                while(x != x2) {
                        y = (int)( dydx * (x - x1) + y1 );
                        executaPassos(x - oldX, y - oldY);
                        oldX = x;
                        oldY = y;
                        x += incX;
                }
        }  else  {
                int incY = dy < 0 ? -1 : 1;
                float dxdy = (float)dx / (float)dy;

                y = y1;

                while(y != y2) {
                        x = (int)( dxdy * (y - y1) + x1 );
                        executaPassos(x - oldX, y - oldY);
                        oldX = x;
                        oldY = y;
                        y += incY;
                }
         }
}
```

APÊNDICE H – Objetos para comparação dos algoritmos Bresenham e flutuante



Fonte: Do autor.

CONTROLE DE UM SISTEMA XY COM MOTORES DE PASSO POR MEIO DO ALGORITIMO DE BRESENHAM

Israel Burigo[1], Prof Esp. Sérgio Coral[2]

[1,2]Instituto de Informática – Universidade do Extremo Sul Catarinense (UNESC)
CEP 88806-000 – Criciúma – SC – Brasil

isrburigo@gmail.com, sergiocoral@unesc.net

Abstract.

This project describes the fabrication of an automation control system, to control motors arranged in a Cartesian plane, using low technology and computer algorithms in order to reduce the cost and bring ease of implementation, for companies that are entering the field of automation. For this, it was necessary to implement the steps of an automated system from the control part to the action in the displacements of the axes. This control is managed by intelligent electronic circuits, which in turn, use of algorithms to meet the movement in coordinates. It was also necessary to build a prototype of the coordinate table for testing the entire control system, as well as the algorithms used. By way of comparison was found efficiency of the algorithms, enabling improvements in computational entire system.

Resumo.

Este projeto descreve a confecção de um sistema de controle de automação, para controlar motores dispostos em forma de plano cartesiano, utilizando baixa tecnologia e algoritmos computacionais, afim de reduzir o custo e trazer facilidade de implementação, para empresas que estão entrando no ramo de automação. Para tal, foi necessário implementar as etapas de um sistema automatizado, desde a parte de controle até a atuação nos deslocamentos dos eixos. Esse controle é gerenciado por circuitos eletrônicos inteligentes, que por sua vez, utilizam de algoritmos para satisfazer o movimento nas coordenadas. Foi também necessário construir um protótipo da mesa de coordenadas, para testar todo o sistema de controle, bem como, ao algoritmos utilizados. Por meio de comparação se verificou a eficiência dos algoritmos, permitindo uma melhoria computacional em todo sistema.

1. Introdução

Segundo Souza (2005) a maioria das empresas do ramo industrial utilizam em seus controles de processos sistemas mecânicos, eletromecânicos e computacionais, como operadores destes controles. O aumento da qualidade dos produtos fabricados, redução de perdas em matérias primas e redução do tempo de fabricação são alguns dos fatores que levam uma empresa a automatizar os seus processos (FIGUEIREDO, 2010).

Um dos meios de automatização é a utilização de sistemas com motores elétricos, pois são capazes de transportar objetos por esteiras, conseguem produzir esforços repetitivos a longo prazo e aceleram o processo de produção. Trabalhos como acabamento e corte em metais necessitam de motores com precisão. Para este tipo de trabalho geralmente são utilizados motores de passo que de acordo com Queiroz (2002) são dispositivos que convertem pulsos elétricos em movimentos rotativos, possuindo três estágios: parado, ativado com o eixo do motor travado ou girando em etapas.

Para controlar os motores de passo são necessários sistema eletrônicos inteligentes, como por

exemplos os microcontroladores. Microcontrolador é um pequeno dispositivo eletrônico, que possui uma inteligência programável. Entretanto os microcontroladores não conseguem fornecer a intensidade de corrente elétrica que os motores exigem. Para abordar este problema é necessário o desenvolvimento de hardwares específicos para a situação. Este tipo de hardware é conhecido como *driver* (SOUZA, 2007)

Os *drivers* devem ser apropriados para as condições de trabalho que os motores exigirão. Um conceito simples é que para se controlar somente um motor, o microcontrolador deve produzir pulsos elétricos, tais como em lógica digital, zeros e uns. O *driver* interpretará estes pulsos e irá girar o motor executando os passos para cada pulso interpretado. Um passo é um giro em um ângulo definido pelo fabricante do motor (BRITES, 2008).

Em sistemas com dois motores de passo, geralmente um motor com eixo horizontal (x) e outro vertical (y), ambos em um plano, como por exemplo, Controle Numérico Computadorizado (CNC), uma máquina para modelagem de peças mecânicas em metais que movimenta uma broca no eixos para a modelagem do material, existe a necessidade de estabelecer uma sincronia entre ambos os motores, ou seja, para o sistema andar na horizontal (eixo x) somente um motor se deslocará, o mesmo é válido para andar na vertical (eixo y). Agora se o deslocamento for na transversal (eixos x e y) ambos os motores serão controlados. O desafio é controlá-los entre o transversal-horizontal ou transversal-vertical, exigindo soluções matemáticas para aplicar a sincronia. O movimento nestes meios parece assíncrono, pois os motores se deslocarão de forma distinta, e ao traçar uma reta, um se deslocará mais que o outro, entretanto o sistema de controle gerenciará os passos a serem interpretados pelo *drive*r, compensando a inclinação da reta projetada tornando os motores dependentes entre si.

Na prática, as empresas de automação industrial possuem seus sistemas específicos para este tipo de controle. O nível de precisão que os motores de passo exercem, o sistema de gerenciamento do controle feito por um engenheiro e o hardware envolvido, são métodos que posteriormente agregam valor ao produto final, tornando o conhecimento, as tecnologias envolvidas e o custo, difíceis para a aquisição de empresas que estão iniciando na área de automação.

Em análise, a ciência da computação conhece um problema parecido, no qual, trata-se da projeção de *pixels* em formato de retas. Determinado por coordenadas inicial e final, o problema consiste em projetar os *pixels* na tela em formato inclinado de forma rápida. A solução para este problema computacional se dispõe na área de computação gráfica, sendo alcançado por algoritmos.

Segundo Cormen et al (2012) os procedimentos computacionais bem definidos que recebem um conjunto de valores como entrada e retornam valores como saída, são considerados algoritmos. Também pode-se considerar como ferramentas para a resolução de problemas computacionais específicos.

Um destes algoritmos, é o de Bresenham, que de acordo com Moro (2009), utiliza da aritmética com números inteiros como base para acender pixels na tela, determinando quais *pixels* serão destacados para atender o grau de inclinação de uma reta.

2. Justificativa

Os motores de passo, de acordo com Queiroz (2002), são dispositivos de alta precisão, confiáveis e de fácil controle. Esses motores têm sua aplicação em diversas áreas como, informática e robótica. Grande parte destes motores são introduzidos na automação industrial

devido seu torque, porém exigem hardwares com maior robustez.

Ao contrário da maioria dos motores, motores de Corrente Contínua (CC), Corrente Alternada (CA) e servo motores, um motor de passo não se destaca por sua força e velocidade, mas pela precisão que seus movimentos podem exercer (BRITES, 2008).

Para controlar um sistema xy, as empresas de automação industrial, optam pelos motores de passo, pois em relação aos outros têm precisão, não tão boa quanto a um servo, porém possuem custo mais baixo e fáceis de controlar.

Este sistema deve possuir controle eletrônico, para ser manipulado por um usuário, tornando a máquina, o sistema xy, uma ferramenta de auxílio a produção. O controle deve gerenciar os motores e para isso é necessário sincronia. Alguns sistemas como máquinas CNC, utilizam logaritmos, funções exponenciais e somatórios para gerenciar esta sincronia, exigindo avançado conhecimento em programação e hardware, por parte do desenvolvedor do sistema.

Uma possível analogia é a percepção de visualizar o sistema como se fosse um monitor, os *pixels* são as coordenadas, possuindo ponto de referência (0,0) no canto superior esquerdo e sendo mapeado como uma matriz. Assim, compreende-se como os motores devem reagir ao traçar uma reta de um ponto a outro, imagina-se que os *pixels* acenderão para cada deslocamento ao próximo *pixel,* mas para um deslocamento inclinado, por exemplo em 30º, a reta poderá passar entre dois *pixels* e será difícil a visualização por não saber qual pixel será acendido. Tal analogia pode ser aplicada ao mundo físico, porém deve-se satisfazer estas inclinações. Por meio das soluções disponibilizadas pela computação gráfica, é possível manipular os passos dos motores como se fossem *pixels* com uso de algoritmos.

O algoritmo de Bresenham implementará a sincronia entre os motores, que ao contrário de outros algoritmos, como os implementados em máquinas CNC, não necessita de avançados recursos tecnológicos, tornando o sistema viável para a aquisição de empresas que estão iniciando no ramo de automação industrial.

3. Desenvolvimento

Este controle permite disponibilizar um modelo, que pode ser adaptado para diferentes áreas de atuação na automação industrial, ou seja, foi com o intuito de facilitar o processo produtivo, garantindo um menor custo com tempo na produção, ou confecção de uma máquina com finalidades plotadores. Conhecendo como sistema se comunica é possível sua implementação em qualquer linguagem, ou até mesmo a conversão de softwares já disponíveis no mercado. Alguns exemplos de como o sistema pode ser abrangente, é colocar um canhão de corte a laser, uma máquina serigráfica, uma máquina de bordados, entre outros.

Para este sistema, foi determinado a elaboração desde a parte de comparação, controle, atuação e sensoriamento, visando uma melhoria na execução dos processos. Estas etapas, englobam a confecção de um *drive*r eletrônico para os motores de passo, a construção de uma mesa mecânica, a definição de um protocolo de comunicação, permitindo que qualquer software o implemente para controlar o sistema, controle de posição e controle dos atuadores.

3.1. Desenvolvimento do driver de motor de passo

O objetivo de projetar o drive para motores de passo, é facilitar o controle lógico dos mesmos. Por meio de um microcontrolador é possível elaborar um projeto eletrônico que receba somente sinais de clock, servindo como passos, e sinais de sentido de rotação.

Para o projeto é necessário analisar quais condições os motores deverão trabalhar. Utilizando

o circuito integrado (CI) PBL 3770 os motores deverão trabalhar no máximo a 2A de corrente elétrica e 25 volts de tensão, o intuído de utilizar este CI é aplicar motores bipolares que possuem maior força.

A implementação do software do microcontrolador, firmware, seguiu as regras de controle de níveis nos pinos dos 3770's para controlar um motor, essas regras estão dispostas no manual do fabricante do CI. E utilizando a MPLAB IDE com o *plugin* HI-TECH, desenvolveu-se um *firmware* para o PIC16F870 em linguagem C. A primeira parte da codificação foi dividir os acionamentos de forma a habilitar o controle de passos em cada motor e iniciar o sistema por meio do método e loop principais.

Após o firmware estar programado, foi necessário confeccionar o layout do *drive*r. Observando as conexões dos encaixes dos motores, alimentação e sinais de entradas, foi determinado que colocar os conectores nas bordas do layout facilitaria este processo.

O próximo passo foi corroer uma placa de cobre para dar origem ao layout. Esta etapa não necessita de grande conhecimento ou habilidade, requer somente alguns itens que podem ser adquiridos em lojas de produtos eletrônicos e existem inúmeros tutoriais explicando como produzir um hardware caseiro qualquer, na internet. Após corroído, foram necessários o encaixe e a solda dos componentes em seus devidos lugares.

Para testar o driver foi necessário construir um protótipo mecânico do sistema XY, este sistema irá acoplar dois motores de passo e uma cabeça, que será a referência do percurso dos motores.

3.2. Mesa XY mecânica

A parte mecânica que compõe a mesa foi projetada com o intuito de testar o driver e o algoritmo de Bresenham. A confecção desta parte foi elaborada utilizando madeiras, ferros e algumas peças mecânicas como polias e correias. Utilizando uma madeira como base, foram adaptados trilho mecânicos, sobre bases de ferro, que servirão como eixos x e y.

Com a posição dos eixos determinada, se objetiva a inclusão dos motores de passo, polias e correias aplicadas para que o driver possa controla-los, também foi apontado a localização da cabeça de plotagem. A correia do eixo Y foi fixada na cabeça para sua movimentação, enquanto a correia do eixo X é fixada no suporte do eixo Y, conforme a figura 1.

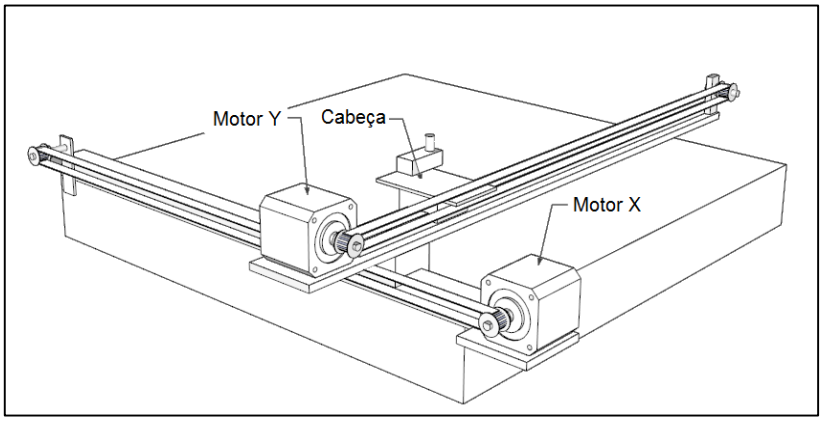


Figura 1 - Montagem mesa

Com a mesa desenvolvida, foi possível iniciar o projeto do software no Arduino. Este software por sua vez controlará todo o sistema, recebendo os objetos por meio de um protocolo de comunicação, interpretando linhas e convertendo-as em pulsos de clock para que o driver assimile, assim é possível a transformação de linhas em passos.

3.3. Programação no Arduino

Utilizando a IDE do Arduino, o software foi primeiramente baseado em estabelecer uma comunicação com um computador, que por meio deste envia os objetos a qual será interpretado e filtrado, para posterior comunicação com o driver. Em outras palavras, foi necessário a implementação de um protocolo que regerá as regras de comunicação.

O protocolo de comunicação limitou-se ao tamanho da memória do Arduino UNO, 2024 bytes de memória de dados. Reservando 524 bytes para outras variáveis, foi determinado que o protocolo deve possuir seis bytes em cada pacote, sendo demonstrado da seguinte forma:

Pacote: Comando hiX loX hiY loY Checksum

O pacote do protocolo pode ser explicado pela tabela 1:

Byte	Nome	Descrição
1	Comando	Byte de controle que determinará algum comando no protocolo
2	hiX	Byte mais significativo do valor X de um ponto
3	loX	Byte menos significativo do valor X de um ponto
4	hiY	Byte mais significativo do valor Y de um ponto
5	loY	Byte menos significativo do valor Y de um ponto
6	Checksum	Byte de verificação, para saber se o pacote está correto

Tabela 1 – Descrição dos bytes do pacote.

Os comandos podem ser compreendidos como informações iniciais do pacote. É por meio deles, que os restantes dos bytes são interpretados. Foi implementado uma série de comandos, para organizar a comunicação, com o intuito de definir um conjunto de regras para o entendimento do protocolo. Comandos como linha, movimento e fim do objeto, são exemplos de comandos que tratam o comportamento da recepção de um objeto inteiro.

O checksum é um validador do pacote de envio, ou seja, é uma garantia que o envio é realmente recebido e que nenhum byte tenha se perdido no meio de comunicação. Para criar esta validação simplesmente é somado os cinco primeiros bytes do pacote em uma outra variável byte, posteriormente inserido no sexto byte e enviado. Quando o pacote é recebido, o Arduino soma os cinco bytes e compara com o sexto, validando-os ou não, figura 2.

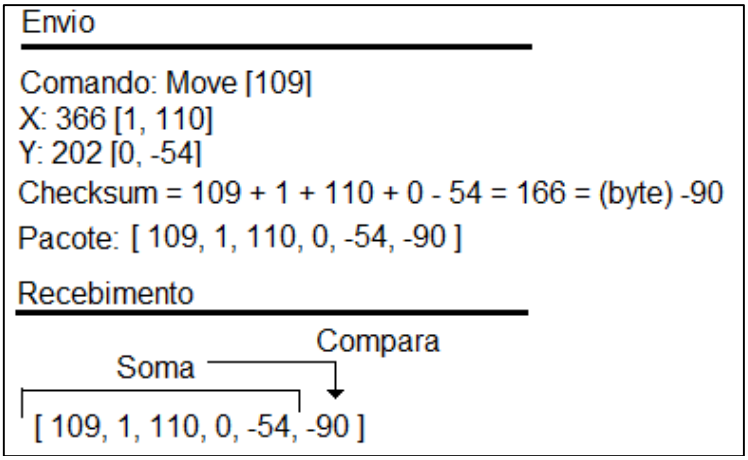


Figura 2 – Calculando Checksum

Ao receber os pacotes, o Arduino retorna somente um byte, correspondente ao comando

recebido. Esta diferença entre pacotes de envio e de retorno, melhora a velocidade de comunicação, pois não há necessidade de agrupar os bytes dentro do Arduino antes do mesmo responder. Contudo o software do Arduino só reconhecerá pacotes de seis bytes consecutivos, dentro de um limite de tempo.

Este limite de tempo, timeout, resolveu problemas de comunicação ao enviar objetos de vinte pontos ou mais, que aleatoriamente geravam checksum invalido, fazendo com que o Arduino não agrupasse os pacotes de forma correta, acarretando no pedido de muitos comandos de pedido de mesmo pacote que bloqueavam a comunicação. O timeout, funciona quando por dois segundos, não é encontrado um checksum valido, assim reiniciando toda a comunicação, e pedindo novamente o mesmo pacote.

Com o protocolo pronto e implementando no Arduino, foi possível carregar objetos para que posteriormente possam ser plotados. Para esta etapa o Arduino precisa ler o objeto carregado e transforma-lo em passos para acionar o driver PBL 3770, neste caso foi necessário transformar o objeto em segmentos, de dois pontos cada, assim criando uma reta que pode ser manipulada pelo algoritmo de Bresenham.

3.3.1. Algoritmo de Bresenham

Para implementar o algoritmo de Bresenham no Arduino, foi necessário organizar a leitura do objeto carregado, e compreender como o algoritmo reconhecerá os parâmetros. Para tal, como o carregamento foi feito em cinco bytes por pacote, foi intuitivo que a leitura seria também em cinco bytes, porém ao analizar o algoritmo descobriu-se que para plotar um pixel, são necessário dois pontos, um de origem e outro de destino.

A cada leitura foi essencial transformar os pontos em segmentos, mas estes não podiam ser criados, pois consumiriam a memória reservada do Arduino. Para resolver este problema é carregado o primeiro ponto, em relação a uma origem, normalmente [0, 0] ou uma localização fixa na mesa, então o algoritmo é utilizado para traçar uma reta entre estes dois pontos. Após traçar a reta, o ponto de relação recebe os valores do anterior, continuando com a leitura do objeto.

No momento, se possui o driver de motor de passo, a mesa xy, um protocolo, que possibilita a comunicação para qualquer software que o implemente, e o algoritmo para plotagem das retas. Contudo, resta transformar a plotagem de retas em clocks e sentido de giro para acionar o driver.

3.3.2. Controle de pulsos

O controle de pulsos serve para transformar o algoritmo de Bresenham de plotagem de pixels em *clocks*. No Bresenham existe um método a qual insere o pixel em determinado ponto. Este método por sua vez recebe as coordenadas do pixel a ser inserido que são anteriormente manipuladas pelo algoritmo.

Como o algoritmo trabalha com deslocamento de pixel, a variação entre cada pixel é de 1 a -1, em inteiros. Analisando este intervalo, conseguiu-se traduzir estes valores, sendo que: 1 executa passos com sentido de giro horário, -1 executa passos com sentido de giro anti-horário e 0 (zero) não executa passos, de acordo com a figura 3.

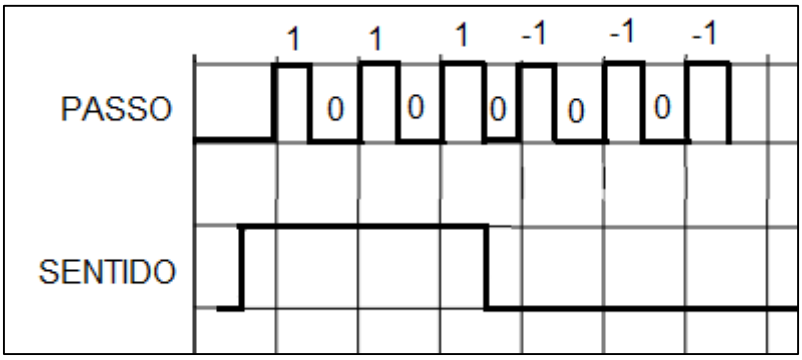


Figura 3 – Controle de pulsos

Com todas as partes do sistema XY elaboradas foi possível a execução dos testes e correções, permitindo analisar o projeto como um todo, desde as partes eletrônicas, mecânicas e computacionais, resultando em uma melhor concepção da abrangência que automação industrial pode disponibilizar.

4. Testes, correções e resultados

Anteriormente aos testes, foi desenvolvido um software em java para que este implemente o protocolo de comunicação e aborde alguns recursos gráficos, como criação de polígonos por meio de pontos, com o intuito de testar totalmente o sistema.

Um primeiro problema a ser percebido, foi que o sistema não possui uma fonte de alimentação para o circuito do driver, pois este foi projetado para trabalhar em 20V com 2A de corrente.

O primeiro teste a ser executado foi um quadrado, e durante a plotagem foi necessário adaptar um sistema de levantamento da cabeça plotadora, que quando for encontrado o comando de movimentar, levante-a. Para tal problema foi adaptado uma solenóide de 20V, adquirida por compra, e alterado o código no Arduino, para este fim.

Após adaptada a solenóide, foi refeito o teste com o quadrado, funcionando perfeitamente. Porém ao testar o sistema plotando o mesmo quadrado rotacionado alguns graus, percebeu-se que o driver não conseguia executar os passos de forma correta, desenhando o objeto de forma irregular com distorções.

Foi percebido que o driver ao deslocar os motores, aleatoriamente inverte o sentido de um ou de outro, ocasionando a distorção, sem que a entrada de sentido fosse acionada. Foi tentado alterar o firmware do driver, e inserir novos PBL 3770, porém estas alterações não resolveram o problema da inversão de sentido indesejada.

Um último recurso, foi confeccionar um novo driver para os motores. Para tal, foi decido construir um driver de controle de motores de passos unipolares, já que os motores adquiridos funcionam como unipolares/bipolares. Este novo driver foi desenvolvido de modo que se comporte semelhante ao anterior, em relação aos sinais de entrada. Contudo para o acionamento dos motores foi feito um sistema com transistores que acionam cada bobina independente.

Uma outra adaptação foi percebida por não existir uma origem de plotagem na mesa, ou seja, não há ponto de referência em que os motores possam iniciar, como consequência ao ligar o sistema, a localização do ponto de origem será onde a cabeça está situada. Para resolver este problema, foi necessário inserir sensores que situam a posição de referência. Foi adquirido componentes por meio de compra, que são pequenos sensores ópticos por barreira, e são

baratos e fáceis de adquirir. Estes sensores, foram adaptados em um canto do sistema, onde foi determinado a referência da mesa.

Após as correções feitas, foi determinado testar a eficácia do algoritmo de Bresenham, em relação a um de ponto flutuante. Para este procedimento, foi estudado e implementado um algoritmo baseado em cálculos de retas, que consiga satisfazer a mesma plotagem que o de Bresenham executa.

Com os algoritmos implementados, foi definido um conjunto de objetos para serem comparados. Foi estipulado uma média de cinco plotagens cada, para ambos os algoritmos. Os objetos, apêndice H, foram desenhados no software em java desenvolvido para testes e selecionados para a plotagem. A tabela 2 traz os tempos de cada plotagem, mostrando um percentual de eficácia que o algoritmo de Bresenham conseguiu superar em relação ao de ponto flutuante.

Velocidade de plotagem = 1000	Tempo de plotagem do algoritmo em segundos		
Objetos	**Bresenham**	**Flutuante**	**% de Eficácia**
Quadrado	5,210212489	5,416391621	3,96
Triangulo	4,133891307	4,275926132	3,44
Espiral (25 pontos)	9,500472344	9,924888997	4,47
Texto (74 pontos)	16,04793689	17,23144316	7,37
Círculo (50 pontos)	4,137161061	4,696969329	13,53
Média:	7,805934818	8,309123848	6,45

Tabela 2 – Eficácia do algoritmo de Bresenham em relação a um de ponto flutuante

Como foi percebido, o algoritmo de Bresenham é pouco eficaz quando os objetos possuem uma quantidade baixa de pontos, porém ao aumentar a quantidade dos mesmos, ocorreu uma significativa diferença de tempo entre as amostras. Uma observação relevante, é que os objetos não estavam na mesma escala, resultando em uma diferença entre tempos e quantidade de pontos, como exemplo o círculo e o texto. Outros testes foram executados, com finalidade de tentar encontrar alguma outra falha, entretanto não foram encontrados erros que comprometessem o projeto.

Finalizando o projeto, foi organizado uma caixa para a parte de controle do sistema, esta caixa possui toda a eletrônica envolvida, desde o Arduino, a fonte e o novo driver. O intuito de organizar é possibilitar a separação da automação, conceituadas em partes de controle, sensoriamento e atuadores. Também facilitou no reconhecimento dos recursos utilizados, de modo que estes, são de relevância na confecção do sistema, pois abrangem grande parte da produção do projeto.

5. Conclusão

O trabalho de conclusão de curso aproximou o autor a área de automação industrial, e possibilitou ao mesmo conhecer funcionamentos das diferentes etapas de um processo automatizado. Essa aproximação construiu conhecimentos acerca da integração entre *hardware* e *software*, que possibilitaram a confecção do sistema como um todo.

O processo de confeccionar, foi um obstáculo nas partes eletrônica e mecânica. O estudo de funcionamento dos circuitos integrados envolvidos no *driver* acarretaram dificuldade ao tentar implementar um *firmware* para controlar os mesmos, contudo a elaboração de um *driver* mais

simplificado foi viável. Também ocorreram diversas falhas na projeção da mesa mecânica, sendo que, foram necessárias três mesas para alcançar uma que satisfizesse a eletrônica envolvida.

Em analise, a confecção deste sistema traz a liberdade de utilizar motores de passo de várias potências, além de permitir a escalabilidade da área mecânica da mesa e a elaboração de indeterminados softwares que consigam implementar o protocolo. Porém ainda é de difícil entendimento, pois há necessidade da compreensão em comunicação de dados, e conhecimento na implementação de softwares que abrangem tais comunicações.

Para um projeto de maior complexidade seria desenvolver o software controlador, esse por sua vez, implementaria o protocolo e algoritmos de eficiência gráfica, que otimizariam a área de plotagem, encaixando automaticamente os objetos a serem plotados, de forma a ocuparem uma maior área útil.

O custo do controle eletrônico, pode ser reduzido ao trocar o PIC do *driver* por um de custo menor, porém com características parecidas. Uma outra alternativa é elaborar uma placa dedicada envolvendo todo o conjunto de controle e eliminando o valor total do Arduino UNO, resultando somente a utilização de seu microcontrolador.

Uma das observações foi no limite de passos que os motores podem executar. Em *half-step* os motores escolhidos alcançam 400 passos por volta, limitando a precisão. Porém é possível a multiplicação destes passos, por meio de circuitos que gerenciam a corrente nas bobinas do motor. Estes passos que vão além da capacidade física do motor, são conhecidos como micropasso e necessitam de um elaborado estudo para sua implementação.

Uma outra melhoria, seria incorporar a utilização de chips de memória, que aumentariam a quantidade de pontos a serem plotados, além de permitir o transporte dos arquivos. Para isso, é necessário conhecer as formas de comunicação que implementam tais gravações e leituras.

Por fim, o trabalho de conclusão de curso relacionou características computacionais e industriais, tal que, foi possível o aprimoramento do processo de automatização.

Referências

ALVES. Toni dos Santos. **Automação Industrial I**. Abrantes: Escola Superior de Tecnologia de Abrantes, 2005. 55 p.

BRITES, Felipe Gonçalves; SANTOS, Vinicius Puga de Almeida. **Motor de Passo.** Niterói: UFF, 2008. 15 p.

CORMEN, Thomas H. et al. **ALGORITMOS:** Teoria e Prática. Tradução Arlete Simille Marques. 3. ed. Rio de Janeiro: Elsevier, 2012.

FIGUEIREDO, Bruno Oliveira Dourado Arruda de. **MELHORIA DA QUALIDADE DE PROCESSOS INDUSTRIAIS ATRAVÉS DA AUTOMAÇÃO E DA REPROGRAMAÇÃO DO SISTEMA SUPERVISÓRIO**. 2010. 61 f. Monografia (1) – Departamento de Centro de Tecnologia, Universidade Federal do Ceará, Fortaleza, 2010.

QUEIROZ, Ricardo Alexandro de Andrade. **Motores de Passo**. Salvador: Unifacs, 2002. 19 p.