UNIVERSIDADE DE TAUBATÉ
Departamento de Engenharia Mecânica

SOFTWARE PARA CONTROLE DE UM MANIPULADOR ROBÓTICO AUXILIADO POR UM SISTEMA DE VISÃO

Valdeci Donizete Gonçalves
Orientador: Prof. Dr. Álvaro M. S. Soares

Taubaté - SP
2004

COMISSÃO JULGADORA

Data:

Resultado: _____

Prof. Dr. Álvaro Manoel de Souza Soares

Assinatura:_____

Prof. Dr. Mauro Hugo Mathias

Assinatura:_____

Prof. Dr. Francisco Carlos Parquet Bizarria

Assinatura:_____

UNIVERSIDADE DE TAUBATÉ
Departamento de Engenharia Mecânica

SOFTWARE PARA CONTROLE DE UM MANIPULADOR ROBÓTICO AUXILIADO POR UM SISTEMA DE VISÃO

Valdeci Donizete Gonçalves

Dissertação apresentada à Pró-reitoria de Pesquisa e de Pós-graduação da Universidade de Taubaté, como parte dos requisitos para obtenção do Título de Mestre pelo Curso de Pós-Graduação em Engenharia Mecânica.

Orientador: Prof. Dr. Álvaro M. S. Soares.

Área de Concentração: Automação

Taubaté – SP
2004

Gonçalves, Valdeci Donizete
Software para controle de um manipulador robótico auxiliado por um sistema de visão/ Valdeci Donizete Gonçalves.--Taubaté: UNITAU, 2004.
102f. : il.

Orientador: Álvaro Manoel de Souza Soares
Dissertação (Mestrado) – Universidade de Taubaté, Departamento de Engenharia Mecânica, 2004.

1.Processamento de Imagem. 2.Robótica. 3.Sistema de Visão– Dissertação. I.Universidade de Taubaté. Departamento de Engenharia Mecânica. II. Título.

DEDICATÓRIA

Dedico este trabalho ao Criador e Redentor da humanidade.

Mt-17:20

“pois em verdade vos digo que, se tiverdes fé como um grão de mostarda direis a este monte: Passa daqui para acolá, e ele há de passar; e nada vos será impossível."

AGRADECIMENTOS

A Deus pela sua Palavra de Verdade.

Ao meu orientador, Prof. Dr. Álvaro Manoel de Souza Soares, pelo apoio constante, sugestões pertinentes e precisas e, principalmente, pela forma amiga e paciente com que me tratou ao longo desse trabalho.

Ao professor Dr. Giorgio E. O. Giacaglia, pela existência do curso de mestrado no departamento.

Ao professor Dr. Francisco José Grandinetti, pelo incentivo e ensinamentos constantes, ao longo desses anos de convivência.

A minha esposa, Luciana, que me fez compreender o mais amplo sentido das palavras: paciência, perseverança, fé.

À Universidade de Taubaté, pelo empréstimo do laboratório e equipamentos para a realização do trabalho.

E a todos aqueles que de forma direta ou indireta auxiliaram na execução desse trabalho,

O meu profundo agradecimento.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

GONÇALVES, V. D. *Software para Controle de um Manipulador Robótico Auxiliado por um Sistema de Visão.* 2004.102p Dissertação (Mestrado de Pós-Graduação em Engenharia Mecânica) – Departamento de Engenharia Mecânica, Universidade de Taubaté, Taubaté.

RESUMO

Este trabalho mostra um robô fazendo uma tarefa de pegar e colocar auxiliado por um sistema de visão. Um projeto experimental foi desenvolvido, sendo composto por um robô didático "Lynxmotion" de quatro graus de liberdade e uma "web cam". O espaço de trabalho do robô foi mapeado usando um "look up table", para minimizar os erros de posição do robô. A idéia principal é que o robô deve estar apto a escolher um parafuso, por exemplo, em qualquer lugar de sua área de trabalho com o auxílio do sistema de visão. No espaço de trabalho nós temos um cesto ou um recipiente que fica em uma posição conhecida e um parafuso que pode ser colocado em qualquer lugar. O sistema de visão adquire a cena da imagem e localiza o centro de gravidade do parafuso. Com esta informação, o robô pega o parafuso e o coloca no cesto. Para diminuir o custo experimental, uma "web cam" conectada a uma porta "USB" do computador foi usada. Um programa de computador, em linguagem "Visual C^{++}" foi desenvolvido para adquirir e tratar as cenas das imagens e controlar o robô. O programa implementa os algoritmos de processamento de imagem e a cinemática direta do robô. A precisão e repetibilidade são discutidas e os resultados mostram que os algoritmos de controle do robô e processamento de imagens trabalham muito bem.

Palavras-chave: Robótica, Sistema de visão e Processamento de Imagem.

GONÇALVES, V. D. *Software for Control of a Robot Manipulator Aiding by a Vision System.* Taubaté, 2004. 102p. Dissertation, Universidade de Taubaté, Taubaté.

ABSTRACT

This paper shows a robot doing a pick and place task, aided by a vision system. An experimental set up was assembled, composed by a didactic Lynxmotion four degree of freedom Robot and a web cam. The robot work space was mapped using a look up table, in order to minimize position errors. The main idea is that the robot must be able to pick, a bolt for example, in any place of its workplace, aided by the vision system. On the workplace we have a basket, place in a known position and a bolt placed anywhere. The vision system acquires the scene image, and locates the bolt gravity center. With this information, the robot pick the bolt and place it on the basket. In order to decrease the experimental costs, a web cam connected to a computer USB port was used. A computer program, in visual C^{++} language, was developed to acquire and treat the scene images and to control the robot. The program implements the Image processing algorithms and the robot direct kinematics. The system accuracy and repeatability are discusses and the results shown that Robotics and Image Processing algorithms work quite well.

Keywords: Robotics, Vision System and Image Processing

1 INTRODUÇÃO

No século XX, com o grande desenvolvimento da indústria de manufatura, os processos automatizados evoluíram muito rapidamente. Nas últimas décadas, foram criados os computadores que se tornaram responsáveis por uma nova era de automação na indústria. Logo apareceram os robôs, que são mecanismos automatizados controlados por uma unidade de processamento. Um robô atualmente se destina a executar muitas das tarefas outrora somente executadas pelo ser humano, e cada vez mais ele precisa das características do homem, e uma grande característica do homem é a visão. No século passado este fato era objeto somente de ficção científica, mas a visão robótica será abordada neste trabalho de uma forma científica e prática, onde um protótipo de um sistema de visão acoplado a um robô foi desenvolvido e testado, sendo capaz de executar a tarefa proposta de "ver" um objeto, achar suas coordenadas no plano, capturar o mesmo, e depositá-lo em um local específico.

Devido à necessidade de um software intuitivo para o controle de um robô auxiliado por um sistema de visão, iniciou-se o desenvolvimento de um aplicativo, que tem por objetivo integrar o controle de um manipulador robótico a um sistema de visão. É importante que este aplicativo possua uma forma intuitiva e seja de fácil operação, proporcionando sua utilização de maneira eficiente, sem pré-requisito de conhecimento de programação por parte do usuário deste poderoso recurso. Será abordado também o tratamento de imagens e o procedimento para controle de um mecanismo robótico através da cinemática direta.

No início, o trabalho foi desenvolvido em linguagem C^{++}, mas devido à necessidade da aquisição de imagens no sistema operacional "windows", foi necessário migrá-lo para a linguagem "Visual C^{++}", onde grande parte do seu código foi alterada devido à mudança de linguagem de programação. A migração para a linguagem Visual permitiu o uso das bibliotecas do "Direct-Show" da "Microsoft", o que tornou possível para o projeto apresentar a inovação de captura de imagem diretamente pela entrada "USB" do microcomputador, dispensando o uso de uma placa de aquisição de imagens.

1.1 Protótipo Desenvolvido

A Figura 1 mostra a ilustração do protótipo desenvolvido, bem como todos os componentes que o integram.

O computador utilizado foi um “notebook” Pentium 3 de 1.0 G hz e 264 M de memória ram. Uma “web cam” que pode trabalhar até com alta resolução de 640 X 480 pixels (alta resolução), sendo usado neste trabalho a resolução média de 320 X 240 com 16 milhões de cores. Essa aquisição dispensa o uso de uma placa de captura de vídeo, sendo a entrada dos dados de imagem feita diretamente para o microcomputador através de uma conexão “USB”. Usou-se um kit robótico da “Lynxmotion”, e foi aplicada nele toda teoria robótica para criar um “drive” específico para o desenvolvimento deste trabalho.

Figura 1 - Protótipo desenvolvido e seus componentes

1.2 Seqüência Descrita em Cada Capítulo

A Figura 2 é um diagrama que mostra a seqüência descrita na realização deste trabalho, desde o estudo da aquisição de imagem, passando pelas ferramentas de controle do robô, e finalmente o desenvolvimento de uma interface gráfica através do programa “Visual C^{++}” da Microsoft versão 6.0.

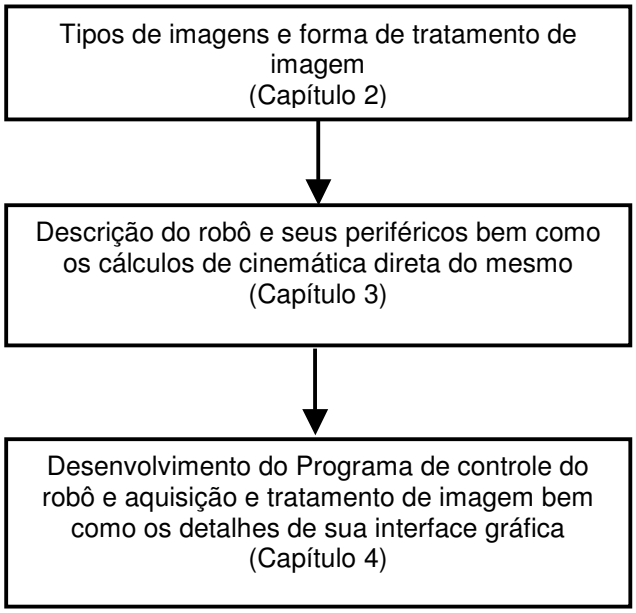


Figura 2 – Diagrama da seqüência descrita em cada capítulo deste trabalho

O capítulo 2 faz uma abordagem a respeito do tratamento de imagens, suas formas de aquisição, alguns tipos de imagens são exemplificados. Na parte do tratamento de imagem, é comentado e explicado sobre o “threshold” e sua aplicação em outros trabalhos. Também é exemplificada a aplicação de sistema de visão na indústria.

O capítulo 3 faz uma abordagem geral sobre o kit robótico utilizado, bem como suas limitações e vantagens. É descrito o tipo do processador utilizado pelo robô, o “Mini-SSC”. As características do robô, sua área de trabalho, os cálculos de cinemática direta também são objeto de estudo deste capítulo.

O capítulo 4 descreve como o programa foi desenvolvido, com o detalhamento de seu desenvolvimento passo a passo. Em seguida, é mostrada a sua parte visual, com uma breve descrição de seu funcionamento e dos recursos disponíveis que permitem ao usuário o controle independente da parte do controle do robô, bem como a integração do sistema do robô com o sistema de visão, que é realizada através de inúmeros controles que são inseridos na interface gráfica do programa.

O capítulo 5 traz a conclusão do trabalho, trazendo comentários sobre suas possíveis aplicações práticas, sobre seu caráter inovador do meio utilizado para a captura de imagem, trazendo algumas sugestões para trabalhos futuros.

2 CONCEITOS BÁSICOS DE TRATAMENTO DE IMAGENS

O objetivo deste capítulo é descrever a teoria básica do processamento de imagens. São demonstrados exemplos de aplicações práticas de sistema de visão na indústria. A aquisição de imagem é descrita, e neste projeto trabalhou-se com o formato “bmp” e o formato “raw”. Uma operação de processamento de imagens, como “thresholding” (limiarização), é descrita para habilitar o processamento de uma imagem em tons de cinza e binarizá-la, para ser tratada pelo programa de processamento de imagens e ser utilizada pelo controle do robô.

2.1 Aquisição de Imagem

Existem muitos métodos de aquisição de imagem que serão descritos neste capítulo. As imagens podem ser divididas em muitos tipos. Sua aquisição depende da iluminação do ambiente, da distância dos objetos, da cor dos objetos, e de muitos outros fatores físicos.

A aquisição de imagens é dividida em aquisição de imagem em tempo real, e a aquisição de quadros momentâneos.

2.1.1 Aquisição de Imagem em Tempo Real

Capturar uma imagem de uma cena no momento que a imagem está sendo usada. Este método de aquisição de imagem é freqüentemente feito em câmeras de vídeo, sendo necessário adquirir no mínimo 25 quadros por segundo. A maioria das câmeras captura 25 quadros por segundo que consistem de 625 linhas (padrão britânico). Isto significa que o sinal não consiste de pixels e de cores, mas de variação de sinal através da intensidade de luz.

Este sinal não é utilizado para o processamento de imagem, antes ele é convertido para pixels descrevendo a intensidade de cada cor do quadro capturado. Para esta conversão é comum utilizar-se de uma placa de aquisição de imagens “frame grabber”, conectada a um computador. Uma “frame grabber” digitaliza a imagem, mas isto pode causar alguns problemas, onde o número de cores e a resolução podem ser reduzidos.

A aquisição da imagem em tempo real nos dá muitas possibilidades de realização na automação robótica, e por isto é utilizado neste projeto.

2.1.2 Aquisição de Fotos de Imagem

A aquisição de fotos de imagem é menos complicada do que a aquisição em tempo real. Seu princípio de operação é o mesmo de um scanner ou de uma câmera fotográfica. A vantagem de um scanner é que ele dá uma grande nitidez da imagem, mas tem que manter uma distância fixa de uma imagem bidimensional. Já a câmera possui a lente que permite ajustar o foco da imagem.

O método de gravar por câmera ou scanner é feito por elementos de CCD (Charge Coupled Devices). Ele é um dispositivo com células de foto sensibilidade à luz, que é usado para criar um mapa de bits da imagem. Eles podem gravar as cores das imagens se forem utilizados filtros. Os sinais analógicos dos capacitores do CCD são convertidos para imagem digital, que é guardada na memória de uma câmera ou transferida para um computador, se um scanner é utilizado. Os arquivos das imagens ficam no disco rígido do computador em um formato requerido. Neste projeto foi utilizada uma “web cam” (câmera digital). As imagens são capturadas em tempo real e tratadas no programa desenvolvido. Os tipos de imagens geradas serão vistos nas seções seguintes.

2.2 Tipos de Imagens

Esta seção descreve alguns métodos para a descrição da imagem e seu armazenamento. A base sobre a descrição das imagens é em tons de escala cinza, ou o formato colorido conhecido como RGB, pois um formato colorido é formado pela combinação dos tons vermelho, verde e azul.

2.2.1 Representação da Imagem

Uma imagem consiste de duas dimensões de matrizes de pixels. Para cada pixel é dado uma cor ou tom de cinza correspondendo ao número de matrizes do tipo da imagem. No caso de uma imagem colorida, cada pixel é representado por três tons de cores, conforme mostrado na Figura 3.

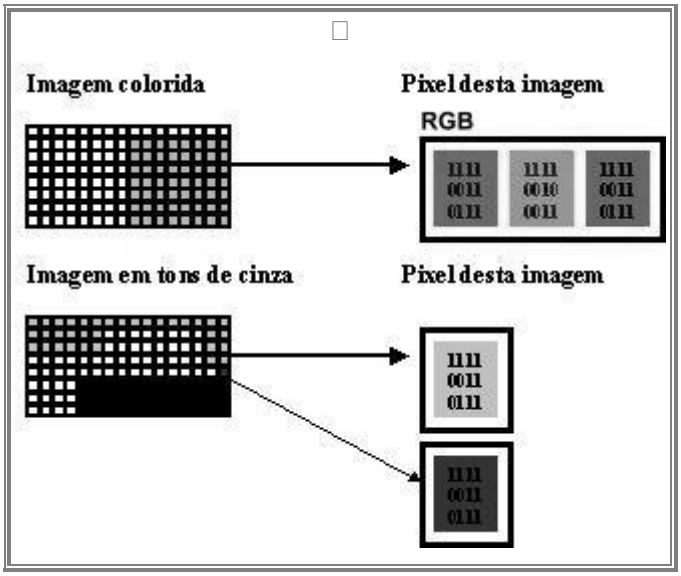


Figura 3 - Pixels RGB, pixels em tons de cinza.

Basicamente há 3 tipos de imagens:

- Preto e branco
- Tons de escala cinza
- Colorida

A imagem preto e branco ou imagem binária tem apenas duas cores e pode ser representada por um bit por pixel. Zero para preto e um para branco.

As imagens em tons de escala cinza possuem variação de tonalidades, no qual 0 (zero) é preto e 255 é branco. Alguns tipos de imagens em tons de cinza podem ter 16 diferentes níveis e eles podem ser armazenados em apenas 4 bits comparados com a imagem com 256 tons de cinza que consome 8 bits da memória.

Imagens coloridas podem ter acima de 24 milhões de cores. Mas elas consistem de três cores básicas que são vermelho, verde e azul. Cada uma dessas cores básicas tem a intensidade de 0 a 255 tons. Este tipo de imagem usa cerca de 24 bits de memória por pixel.

Para este processamento de imagem, a vantagem do número de cores reduzido, ou em tons de cinza, é reduzir o número de memória requerida.

2.2.2 Armazenamento da Imagem

O armazenamento da imagem pode ser feito de diferentes métodos, tais como:

- Vetor de imagens ;
- Pixels de imagens;

- Imagens comprimidas.

Algumas imagens são armazenadas como vetores, em que a informação da imagem é a descrição do vetor de diferentes linhas da imagem. Um exemplo é uma imagem no desenho de AutoCad com a extensão (dwg).

Os pixels das imagens são guardados com a descrição de cores ou níveis de cinza para cada pixel. Antes de o programa ler a imagem, ele sabe o tamanho da mesma. Para reduzir os tamanhos da imagem, diversos métodos de compressão são utilizados. Eles podem ser feitos através de métodos complicados de operações matemáticas ou de forma mais simples.

2.2.3 O Formato BMP

Neste projeto, a aquisição das imagens é feita no formato BMP. Arquivos de “windows bitmap” são armazenados em um dispositivo de formato independente para bitmap (DIB), o que permite ao sistema operacional “Microsoft Windows” mostrar o “bitmap’’ para cada dispositivo de exibição. O termo “dispositivo independente” significa que o bitmap especifica o pixel de cor de uma forma independente do método utilizado para mostrar a cor que ele representa. A extensão para um arquivo windows DIB é .bmp.

Sobre a estrutura de um arquivo bitmap, pode-se dizer que cada bitmap contém um arquivo de cabeçalho, onde se encontra a cor utilizada e a matriz de bytes que define a quantidade de bits.

O arquivo tem a seguinte forma:

```
BITMAPFILEHEADER bmfh;
BITMAPINFOHEADER bmih;
RGBQUAD          aColors[ ];
BYTE             aBitmapBits[ ];
```

O cabeçalho do bitmap contém informação sobre o tipo de tamanho, e o layout do DIB.

O cabeçalho é definido como uma estrutura “BITMAPFILEHEADER”, onde são definidos a especificação da dimensão, o tipo de compressão, e o formato da cor para o bitmap. A tabela de cores é definida como uma matriz de estrutura “RGBQUAD”. A tabela de cor não está presente para bitmaps com 24 cores, porque cada pixel é representado por 24 bits vermelho, verde e azul (RGB). A cor na tabela deve aparecer em ordem de importância. A seguir temos um exemplo de um arquivo Bitmap de 16 cores de bitmap (4 bits por pixel):

Win3DIBFile

BitmapFileHeader

Type	19778
Size	3118
Reserved1	0
Reserved2	0
OffsetBits	118

BitmapInfoHeader

Size	40
Width	80
Height	75
Planes	1
BitCount	4
Compression	0
SizeImage	3000
XpelsPerMeter	0
YpelsPerMeter	0
ColorsUsed	16
ColorsImportant	16

Win3ColorTable

	Blue	Green	Red	Unused
[00000000]	84	252	84	0
[00000001]	252	252	84	0
[00000002]	84	84	252	0
[00000003]	252	84	252	0
[00000004]	84	252	252	0
[00000005]	252	252	252	0
[00000006]	0	0	0	0
[00000007]	168	0	0	0
[00000008]	0	168	0	0
[00000009]	168	168	0	0
[0000000A]	0	0	168	0
[0000000B]	168	0	168	0
[0000000C]	0	168	168	0
[0000000D]	168	168	168	0
[0000000E]	84	84	84	0
[0000000F]	252	84	84	0

Na tabela 1 temos outro exemplo da estrutura de um arquivo no formato BMP:

Tabela 1 - Estrutura de um arquivo BMP.

Estrutura do cabeçalho de um arquivo BMP		
Byte Início	Tamanho em Byte	Conteúdo
0	2	Tipo de Arquivo
2	4	Tamanho em unidades de bytes
6	4	Reservado
10	4	Início da imagem
14	4	Tamanho do cabeçalho da imagem
18	4	Largura
22	4	Altura
26	2	Número de Planos
28	2	Bits por pixel
30	4	Tipo de compressão
34	4	Tamanho da imagem em Bytes
38	4	Resolução Horizontal
42	4	Reseolução vertical
46	4	Número de cores
50	4	Número de cores importantes

2.2.4 O Formato RAW

Diferente do formato BMP, onde temos um cabeçalho dando informações sobre o tamanho da imagem, bem como suas características de cores, no formato RAW não têm esta informação, mas este formato consiste apenas da figura em seu formato de 256 tons de cinza. Neste projeto o sistema de visão adquire a imagem em um formato BMP que após passar por um algoritmo, que será detalhado posteriormente, transforma o formato BMP no formato RAW, que sofrerá o tratamento da imagem.

2.3 Limiarização (Thresholding)

“Thresholding” é uma operação em que uma imagem em tons de cinza é convertido em uma imagem binária. Ele é feito para simplificar a imagem com a qual se vai trabalhar. A conversão para uma imagem binária é feita mudando todos os pixels que estão em níveis de cinza abaixo do nível do “threshold” para zero (preto), e todos os que estiverem igual ou acima do nível do “threshold” ficarão valendo 1 (branco).

O nível do “threshold” pode ser calculado por métodos matemáticos ou pode ser fixado manualmente para o estudo da imagem e de seu histograma. Pode-se cometer erros ao selecionar o nível de “threshold” como: não incluir todos os pixels

que se quer estudar na nova imagem, os pixels incluídos poderiam não ser da nova imagem. Pode-se definir o "thereshold" da seguinte forma:

Seja R uma imagem e $R_1, R_2, R_3, \ldots R_\delta$ um conjunto finito de regiões de R tal que

$$R = \bigcup_{i=1}^{S} R_i \text{ e } R_i \cap R_j = \emptyset, i \neq j$$

Dada uma imagem *f(x,y)* em tons de cinza, o "*thresholding"* tem como saída uma imagem binária *g(x, y)* onde:

$$g(x,y) = 0, se\ f(x,y) < T$$
$$g(x,y) = 1, se\ f(x,y) \geq T$$

onde T é o limiar ("*threshold"*), *g(x, y) = 1* para os pontos dos objetos da imagem e *g(x, y) = 0* para pontos que pertençam ao fundo da imagem.

2.3.1 Manual "Threshold"

Um "threshold" manual pode ser determinado para a imagem em estudo, como mostra a Figura 4.

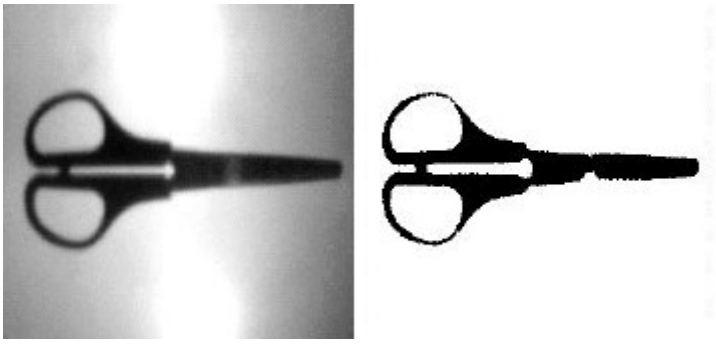


Figura 4 - Exemplo de uma binarização utilizando o thereshold manual.

Os níveis de cinza para diferentes pixels são encontrados usando um programa de processamento de imagem. Também é possível obtê-los pela leitura do arquivo da imagem, mas isto pode ser feito quando conhecemos onde os pixels são requeridos no arquivo.

O princípio da limiarização consiste em separar as regiões de uma imagem quando esta apresenta duas classes (o fundo e o objeto). A forma mais simples, e geral de limiarização, consiste na bipartição do histograma convertendo os pixels, cujo tom de cinza é maior ou igual a um certo valor de limiar (*T*) em brancos e os demais em pretos, como é mostrado na Figura 5.

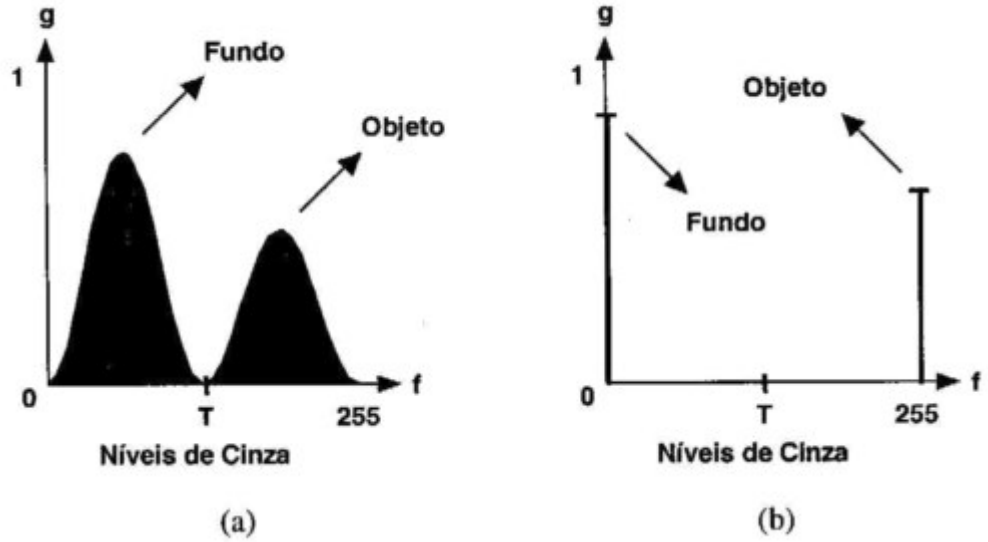


Figura 5 - Limiarização de uma imagem monocromática utilizando limiar *T*: (a) histograma original, (b) histograma da imagem binarizada (MARQUES FILHO e VIEIRA NETO, 1999).

2.3.2 "Thresholding" Automático

O "thresholding" automático consiste em uma operação matemática em que o nível de "threshold" é encontrado através de operações matemáticas. Se o tamanho do objeto é conhecido, seu nível de "threshold" pode ser fixado para o número de pixels que estão presentes na nova imagem. Mas, na maioria dos casos, o tamanho da imagem não é conhecido, então se utilizam outros métodos. Um método matemático é descrito a seguir:

O nível de cinza pode ser encontrado da equação 1.

$$T = max\left\{\frac{t(g)}{P - t(g)} * \left(m(g) - m(G-1)^2\right)\right\} - 1 \qquad \textbf{(1)}$$

T O nível de "threshold"

G O nível de cinza

P O número de pixels da imagem

t(g) O número de pixels até o nível de cinza g e abaixo dele.

m(g) É o nível de cinza para os níveis de cinza entre zero e g

O número de pixels com o nível de cinza de g ou abaixo pode ser encontrado na equação 2:

$$t(g) = \sum_{i=0}^{g} f(i) \quad \textbf{(2)}$$

f(i) O número de pixels até o nível de cinza g.

A média dos níveis de cinza pode ser encontrado pela equação 3:

$$m(g) = \frac{\sum_{i=0}^{g} g * f(i)}{t(g)} \quad \textbf{(3)}$$

O nível de cinza encontrado por essas equações é um nível de cinza na área entre os dois picos do histograma. Conforme o histograma da figura 6 a seguir, podemos ver que o nível de “thereshold” encontrado foi de 85, pois é um nível abaixo do Máximo da equação para o nível de “threshold”, que no exemplo dado o nível máximo é de 86. A detecção automática do nível de “threshold” é difícil em imagens em que a iluminação é inadequada. Quando a imagem é convertida em uma imagem binarizada, é possível obter uma melhor execução nas equações e obter os resultados esperados.

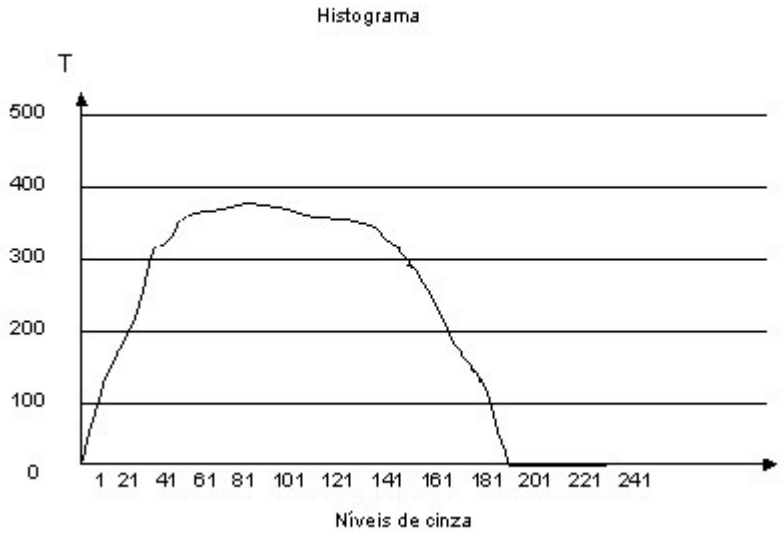


Figura 6 – Histograma.

2.4 Aplicações Práticas de Processamento de Imagens

Nas figuras 7 e 8 a seguir, temos um exemplo da aplicação de um sistema de visão na indústria de alimentos, onde a inspeção é feita para verificar se a embalagem está tampada ou não. Este método é muito útil, pois dispensa a necessidade de um operador ficar somente olhando para ver se existe uma embalagem fora do padrão. E atualmente a fabricação industrial é muito rápida, e se não for utilizado um método automatizado de inspeção, pode-se cometer muitas falhas no produto final. Hoje é exigida nas empresas a qualidade total. Na primeira ilustração da Figura 7 tem-se o processo conforme e na Figura 8 tem-se uma não-conformidade. O sistema de visão envia uma mensagem para um PLC (controlador lógico programável) que retira a embalagem danificada através de um atuador mecânico.

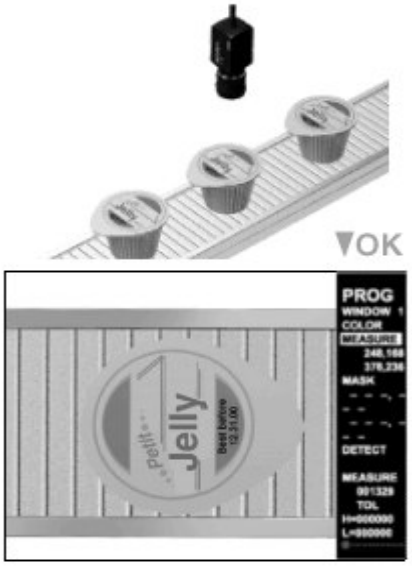


Figura 7- Conforme.

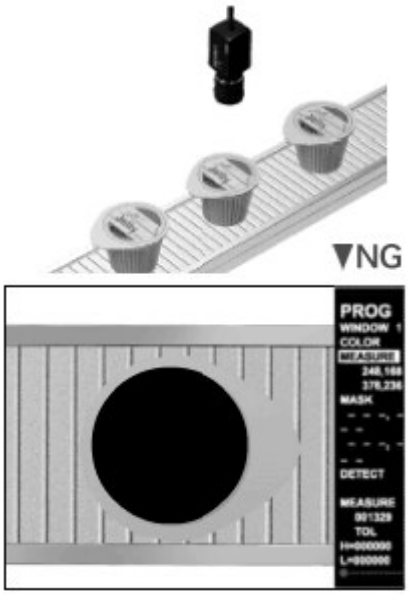


Figura 8 – Não-conformidade.

Outro exemplo é detectar as esferas faltantes nos rolamentos.

Descrição:

O sistema de visão verifica se nos rolamentos faltam esferas.

Vantagens :

Este sistema tem como grande vantagem a sua eficiência, pois esta falha é comum na fabricação de rolamentos e pode ser eliminada com a implantação de um bem sucedido sistema de visão. Podemos ver sua ilustração nas Figuras 9 e 10.

Figura 9 - Rolamento completo.

Figura 10 - Rolamento incompleto.

3 CONCEITOS BÁSICOS DE ROBÓTICA E SUAS APLICAÇÕES

3.1 Robô

Um robô manipulador geralmente é composto por cinco componentes, que lhes permitem executar sua tarefa.

- Braço Mecânico;
- Realizador de tarefas, que é a parte que executa o trabalho. Pode ser uma garra ou qualquer outro manipulador;
- Motores para mover os braços. Os mais usados são servomotores elétricos e motores hidráulicos;
- Controlador: é o processo de entrada que faz o robô executar sua tarefa;
- Sensor, que é colocado para controlar e dar retorno ou habilitar o robô para executar determinado tipo de tarefa.

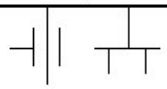
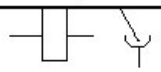
3.1.1 Robôs: Tipos e a Classificação

É necessário entender como um robô trabalha e determinar como se move. Há dois movimentos básicos executados pelos robôs:

- Transversal.
- Rotacional .

O movimento transversal é um movimento linear. O movimento rotacional é um movimento angular. Os tipos diferentes de movimento e os símbolos são mostrados na Tabela2.

Tabela 2 - Símbolos Usados para as Características de Movimento do Robô.

SISTEMA		SÍMBOLO
EIXO DE TRANSLAÇÃO	TELESCÓPICO	
	TRANSVERSAL	
EIXO DE ROTAÇÃO	PIVÔ	
	DOBRADIÇA	
GRIPPER		
FERRAMENTA		

3.1.1.1 Definição de um Robô

Um robô pode ser definido como um dispositivo completamente autocontrolado, que consiste de unidades eletrônicas, elétricas e mecânicas. Geralmente é uma máquina desenvolvida para funcionar no lugar de um ser humano.

O que torna os robôs tão importantes é a idéia de se possuir uma máquina capaz de realizar tarefas que são realizadas por vários homens. Os robôs, segundo a visão fictícia transmitida pelos filmes, são tradicionalmente dispositivos móveis, de aparência humanóide e estrutura metálica. Atualmente, a maioria dos robôs não é humanóide. Ao contrário, eles são máquinas projetadas para realizar funções específicas. Por exemplo, na medicina, braços mecânicos equipados com ferramentas cirúrgicas podem ajudar cirurgiões na realização de operações delicadas. A maioria dos robôs localiza-se ao longo de linhas de montagem, realizando tarefas como soldar, pintar e inspecionar. No campo de CAD/CAM (*Computer-Aid Design / Computer-Aid Manufacturing*), estruturas robóticas têm sido utilizadas para produzir peças como circuitos integrados e modelos sólidos. O Japão é líder na fabricação e utilização de robôs, com a maioria deles empregada nas linhas de montagem da indústria automobilística. Em geral, tais robôs não conseguem aprender novas tarefas, mas realizam procedimentos cuidadosamente controlados por programas de computador.

3.1.1.2 Anatomia de um Robô

A anatomia do robô ocupa-se da construção física do corpo, braço e punho da máquina, sendo que a maioria dos robôs utilizados hoje em fábricas é montada em uma base fixada ao piso. O corpo está ligado à base e o braço, ao corpo. Na extremidade do braço encontra-se o punho, que pode ter inúmeros componentes que lhes proporcionam orientação através de sensores e uma diversidade de ferramentas. Os movimentos relativos entre os diversos componentes do corpo, braço, punho e etc., são proporcionados por uma série de juntas. O conjunto formado pela base, braço e punho é geralmente denominado manipulador. Ligado ao punho está o órgão terminal (garra), o qual não é considerado como parte da anatomia do robô. A junta do braço e do corpo do manipulador é usada para posicionar o órgão terminal, e a junta do punho é usada para orientar o mesmo.

3.1.1.3 Limites Físicos do Robô

Seguem-se as Figuras 11 a 15 com os limites físicos do robô da Lynxmotion, bem como sua área de atuação e suas vistas frontal, lateral e sua altura máxima permitida.

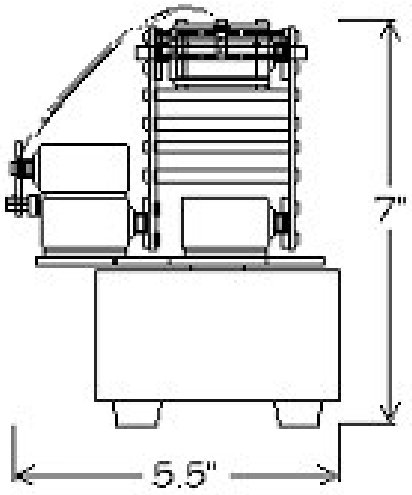


Figura 11 - Vista posterior do robô.

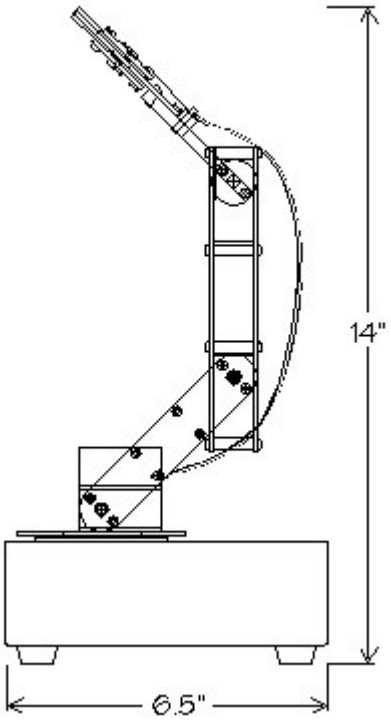


Figura 12 - Alturas máximas.

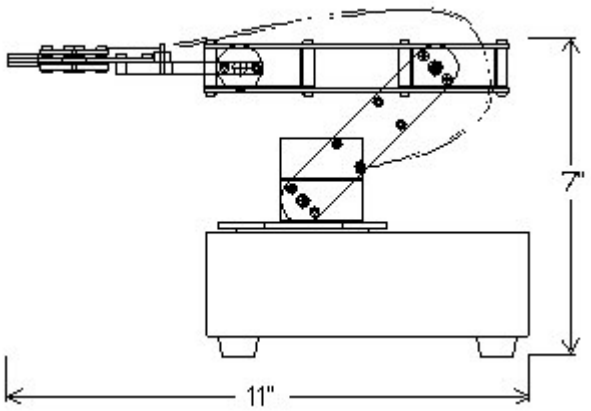


Figura 13 - Vista lateral.

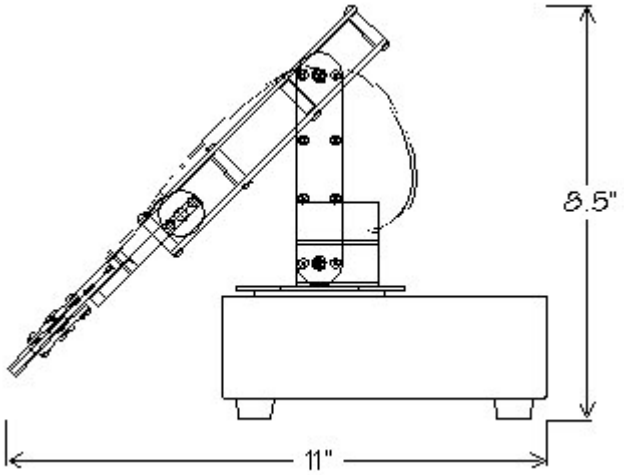


Figura 14 - Limite inferior.

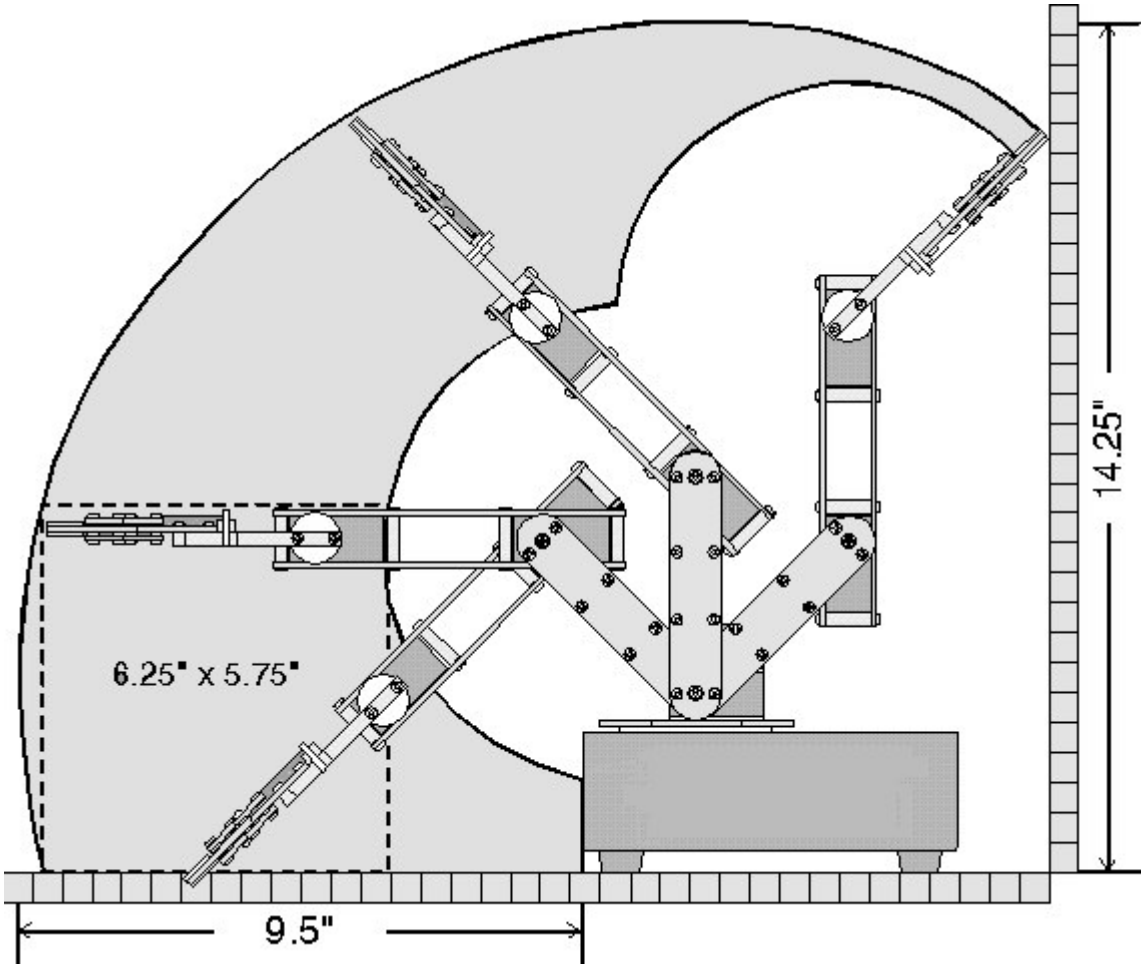


Figura 15 - Limites de afastamento dos braços.

3.1.1.4 Precisão

A precisão refere-se à capacidade de um robô posicionar a extremidade de seu punho em um ponto-meta desejado dentro do volume da área de trabalho, e pode ser definida em termos de resolução espacial, porque a capacidade de atingir um determinado ponto-meta depende da capacidade do robô definir os incrementos de controle para cada um dos movimentos de suas juntas. No pior caso, o ponto desejado se situaria entre dois incrementos de controle adjacentes. Ignorando no momento as imprecisões mecânicas, que reduziriam a precisão do robô, poderíamos inicialmente, definir precisão, nesta hipótese do pior caso, como metade da resolução de controle. Esta definição de precisão se aplica ao pior caso, em que o ponto-meta está diretamente entre dois pontos de controle. Esta definição implica também que a precisão é a mesma em qualquer ponto no volume da área de trabalho do robô. De fato, a precisão do robô é afetada por diversos fatores. Primeiro, a precisão varia

dentro do volume de trabalho, tendendo a piorar quando o braço está no extremo deste e a melhorar quando está mais próximo de sua base. A razão disto é que as imprecisões mecânicas são ampliadas com o braço do robô inteiramente estendido. A expressão "mapa de erros" é utilizada para caracterizar o nível de precisão do robô em função da localização no volume da área de trabalho. Segundo, a precisão é melhorada se o ciclo de movimento for restrito a uma gama de trabalho limitada. Os erros mecânicos tenderão a serem reduzidos quando o robô for exercitado em uma gama restrita de movimentos. A capacidade do robô para alcançar um ponto de referência particular dentro do espaço da área de trabalho limitado é geralmente chamada de precisão local. Quando a precisão é avaliada dentro do volume da área de trabalho total do robô, usa-se o termo precisão global. Um terceiro fator que influencia a precisão é a carga que está sendo carregada pelo robô. Cargas mais pesadas causam maior deflexão dos elos mecânicos do robô, resultando em menor precisão.

A complacência do robô refere-se ao deslocamento da extremidade do punho em resposta a uma força ou torque exercido contra ele. Uma alta complacência significa que o punho é deslocado em uma grande medida por uma força relativamente pequena. O termo "elástico" é geralmente utilizado para descrever um robô com alta complacência. Uma baixa complacência significa que o manipulador é relativamente rígido e não é deslocado em medida significativa. A complacência do manipulador do robô é uma característica direcional, e será maior em certas direções, devido à construção mecânica do robô. A complacência é importante porque reduz a precisão de movimento do robô sob carga. Se o robô estiver manipulando uma carga pesada, o peso da carga fará com que seu braço sofra deflexão. Se o robô estiver pressionando uma ferramenta contra uma peça de trabalho, a força de reação da peça poderá causar flexão do pulso.

3.1.1.5 Órgãos Terminais

Para aplicações industriais, as capacidades do robô básico têm de ser aumentadas por meios de dispositivos adicionais. Podemos nos referir a estes dispositivos como os periféricos do robô. Eles incluem o ferramental que se une ao punho do robô e os sistemas sensoriais que lhe permitem interagir com seu ambiente. Em robótica, o termo "órgão terminal" é utilizado para descrever a mão ou a ferramenta que está conectada ao punho. O órgão terminal representa o ferramental especial que permite a um robô de finalidades gerais realizar uma aplicação particular. Este ferramental particular deve ser projetado especificamente para a aplicação.

3.1.1.6 Controle do Robô

O controle do robô é realizado em seu "cérebro" ou processador, podendo ser feito de várias formas. O controle do robô é dividido na seguinte hierarquia:

Controle dos atuadores: é o controle de cada eixo do robô em bases separadas. É o mais baixo nível de controle;

Controle de percurso: é o controle do robô em coordenação com os eixos para formar o percurso requerido. Este é o nível intermediário de controle;

Controle do movimento: é o controle do robô em coordenação com o ambiente. Este é o mais alto nível de controle.

3.1.1.7 Programação de Robôs

Em sua forma mais básica, um programa de robô pode ser definido como uma trajetória no espaço através da qual o manipulador é comandado a se mover. Esta trajetória também inclui outras ações, tais como controle do órgão terminal e recebimento de sinais dos sensores. A finalidade da programação do robô é instruir-lhe estas ações. Existem vários métodos utilizados para programar robôs. As duas categorias básicas de maior importância comercial atualmente são:

Programação por aprendizagem (*Teaching*): métodos em que o robô é movido fisicamente ponto-a-ponto sobre a trajetória desejada;

Programação por linguagem textual (*Programming*): métodos onde o robô tem sua trajetória definida por um programa.

A programação por aprendizagem consiste em levar o braço do robô a mover-se na seqüência de movimentos requerida e registrá-la na memória do controlador. Os métodos por aprendizagem são utilizados para programar robôs de repetição. Devido a sua facilidade, conveniência e à ampla gama de aplicações adequadas ao mesmo, este método é o mais comum para robôs de repetição.

No caso de robôs de repetição ponto-a-ponto, o procedimento usual é utilizar uma caixa de controle (chamada *teach-in box*) para acionar as juntas do robô a cada um dos pontos desejados no espaço de trabalho e registrar os pontos na memória

para subseqüente repetição. O "*teach-in box"* é equipado com uma série de chaves e mostradores para controlar os movimentos do robô durante o procedimento de instrução.

Robôs de repetição por trajetória contínua também utilizam programação por aprendizagem. Para trajetórias bem definidas, tais como se mover ao longo de uma linha reta entre dois pontos, um *teach-in box* pode ser empregado para informar as localizações dos dois pontos e o controlador do robô então computa a trajetória a ser seguida a fim de mover-se ao longo da trajetória em linha reta. Para movimentos mais complexos (por exemplo, aqueles encontrados em operações de pintura à pistola), geralmente é mais conveniente para o programador mover fisicamente o braço do robô e o órgão terminal através da trajetória de movimento desejada e registrar as posições a intervalos de amostragem espacialmente próximos.

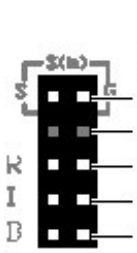
Certos parâmetros do ciclo de movimento, tais como a velocidade do robô, são controlados independentemente quando é estabelecida a tarefa para operar. Conseqüentemente, o programador não precisa estar preocupado com estes aspectos do programa. Sua preocupação principal é assegurar-se de que a seqüência de movimentos está correta.

Os métodos de programação textual empregam uma linguagem de programação própria para estabelecer a lógica e a seqüência do ciclo de trabalho. Um terminal de computador é utilizado para dar entrada nas instruções de programa ao controlador, mas um *teach-in box* também é utilizado para definir os locais dos diversos pontos no espaço de trabalho. A linguagem de programação do robô denomina os pontos como variáveis no programa, e estas são subseqüentemente definidas mostrando-se ao robô suas localizações. Além dos pontos de identificação no espaço de trabalho, as linguagens de robôs permitem o uso de cálculos, fluxo lógico mais detalhado e sub-rotinas nos programas, e um maior uso de sensores e comunicações. Conseqüentemente, o uso das linguagens textuais é freqüente nos robôs inteligentes.

3.1.2 Usando o "Mini SSC II"

O Mini SSC II é um módulo eletrônico que controla oito servos motores de acordo com as instruções recebidas com a velocidade de transmissão de 2400 ou 9600 "baud rate". O Mini SSC II permite interface com duas unidades ou mais unidades permitindo a elas dividirem a linha serial para controlar um total de 16

servos. Esta pode ser ampliada através de unidades programadas com outros endereços de servos até atingir um total de 32 Mini SSC II's controlando 256 servos podendo dividir uma única linha serial. É possível personalizar a operação do Mini SSC II instalando ou removendo "jumpers" (diminuindo os blocos) no cabeçote de configuração, localizado acima no canto direito do controle de circuito, como mostra a Figura 16.



Function	Jumper off	Jumper on	Description
Serial in	always	never	*Do not jumper!* See "Connecting the Mini SSC II"
none	—	—	not used
range	90°	180°	Sets positioning precision and range of motion
identification	0–7	8–15	Sets range of servo addresses
baud rate	2400	9600	Sets baud rate (no parity, 8 data bits, 1 stop bit)

Figura 16 - Seleção dos “jumpers”.

A configuração padrão do Mini SSC II, com todos os "jumpers" removidos é: 2400 baud - servos de 0 a 7 - alcance de movimento = 90°.

Para alterar qualquer dessas configurações, instala-se um bloco "jumper" no plug apropriado, como na Figura17. As alterações passam a funcionar na próxima vez que o Mini SSC II for ligado, então a instalação ou remoção dos "jumpers" deve acontecer com o Mini SSC II desligado.

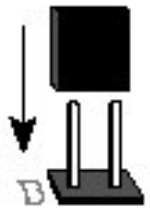


Figura 17 – Bloco do “Jumper”.

Aqui estão mais alguns detalhes para as possibilidades de configuração:

“Range” ou Alcance: sem nenhum "jumper" em alcance, o Mini SSC II controla os servos acima de 90 graus de alcance do movimento. As posições dos servos são expressas em unidades de 0 a 254, então cada unidade corresponde a 0,36 graus de alteração na posição do servo. Em algumas aplicações pode ser necessário trocar a precisão por um alcance mais amplo do movimento. Com um "jumper" instalado em alcance, o Mini SSC II controla os servos acima de até 180 graus, com cada unidade correspondendo a 0,72 graus de mudança na posição.

Nota: Alguns servos não podem ser movidos num ângulo de 180 graus de alcance, eles podem não funcionar quando usados numa posição de valor menor do que 50 ou maior do que 200 com o Mini SSC-II num ângulo de 180 graus.

Identificação: Sem nenhum "jumper" em I, as localizações dos servos são as mesmas dos números impressos abaixo do cabeçote dos servos, de 0 a 7. Com um jumper em I, o Mini SSC II adiciona 8 nessas localizações, então cada localização do servo 0 é igual a 8, para o servo 1 é 9 ... e servo 7 é 15. Isso permite que se conecte dois Mini SSC II a mesma porta serial, e se localize cada um dos 16 servos individualmente nesta configuração.

Baud ou taxa de transmissão: Sem nenhum "jumper" em B, o Mini SSC-II recebe os dados em 2400 baud; com um "jumper" instalado em B, a taxa de transmissão é de 9600 baud. Nos dois casos, a informação deverá ser enviada como 8 data bits, sem paridade, 1 bit parado; a abreviatura será N81. Além disso, os dados deverão ser invertidos, da mesma maneira que sai de uma porta serial padrão.

3.1.2.1 Conectando o “Mini SSC II”

Servos: Servos com padrão de três condutores conectados ao soquete (ex. Conector Futaba -J), podem ser ligados diretamente num cabeçote de três pinos localizado no canto inferior do Mini SSC-II. A Figura 18 mostra como o conector deverá ser alinhado. Se o conector for talhado, o talhe deverá ser alinhado. Se o conector não for talhado, utiliza-se o código de cores para determinar qual pino se conecta a qual terminal. Geralmente os fios elétricos são vermelhos e pretos – os fios pretos se alinham com a ponta numerada do servo conector do Mini SSC II.

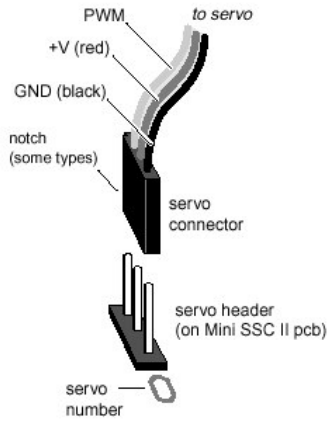


Figura 18 - Conexão do “Flat Cable BS”

Servo Power: Conecta-se a energia (de 4,8 até 6Vdc) para os servos dos fios vermelhos (+) e pretos (-) no SVO do Mini SSC II. Quatro pilhas alcalinas, ou NiCd de 1,5V cada uma, podem ser utilizadas se conectadas em série. Para uma fonte AC/DC use uma corrente linear regulada (não alternada) de 5 volts classificada de pelo menos 1 ampere padrão.

Mini SSC II Power: Conecta-se uma bateria de 9 volts os terminais. Para utilizar outra fonte de energia, remove-se o "snap" e conecta-se de 7 a 15Vdc aos fios; positivo para o vermelho, terra e negativo para o preto. Não se utiliza a fonte de energia do servo para este propósito; a voltagem é muito pequena e a corrente alternada dos servos faz com que este não seja um meio confiável para ligar os circuitos eletrônicos.

Serial Imput: O Mini SSC II requer apenas duas conexões ao computador o serial data e sinal terra. Existem dois terminais para se realizar essas conexões, um plug RJ-11 modular e um par de pinos marcados S (in) no cabeçote de configuração. Conecte o pino "Stamp" usado para o serial output ao S. e o terra "Stamp" (Vss) ao G.

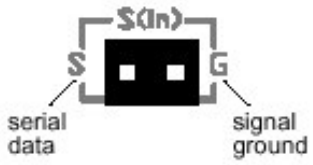


Figura 19 - Conexão do cabo serial.

Para os computadores pessoais, provavelmente o plug RJ-11 será utilizado, como é mostrado na Figura 20.

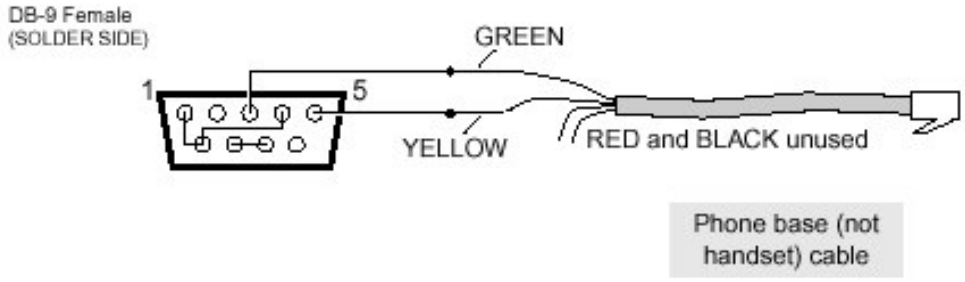


Figura 20 - Conexão do cabo serial PC.

As conexões entre os pinos (1-9-4 e 7-8) são necessárias somente se a linguagem do programa utilizado não for capaz de desligar o "handshaking" do serial "port", como é mostrado na Figura 21.

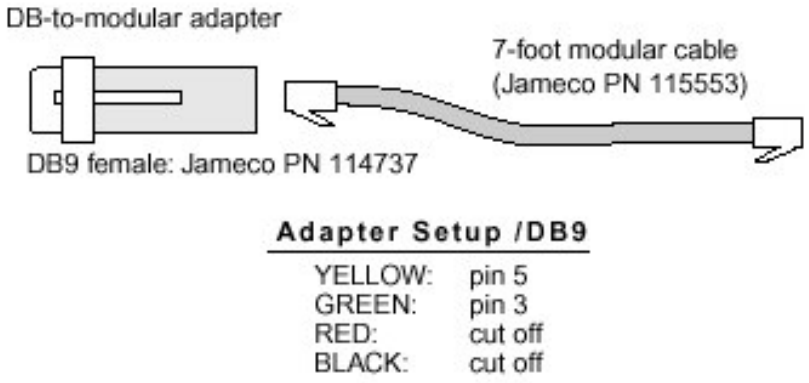


Figura 21 - Conexão do cabo serial PC “Pinagem”.

“Serial Port” de 25 pinos: Se o PC possui um conector em série de 25 pinos, poderá ser usado um adaptador comercial com DB25 até DB9 para conectar a qualquer um dos cabos mostrados acima.

Checagem Inicial.

Quando o Mini SSC II é ligado pela primeira vez, a “sync LED” será ligada e moverá todos os servos para a posição central. Quando conectado os servos, a energia dos servos e a energia do Mini SSC-II, os servos irão imediatamente para o centro. Se eles já estiverem no centro, os servos não irão se mover muito, mas é possível confirmar se a operação está correta, tentando mover um servo mecanicamente. O servo não deverá se mover, e no caso de ele se mover, deverá ser checado o servo e as conexões de energia novamente. Para verificar o link serial, roda-se um dos programas. Se ele estiver formatado corretamente, o serial data será recebido, o “sync LED” irá piscar e o servo irá tomar a sua posição conforme a informação enviada. Se os servos não responderem, é necessário certificar de que a configuração foi feita conforme especificado no programa e que o “pino” da saída serial do computador está conectado corretamente.

Programação para o Mini SSC II.

Comandar um servo para uma nova posição requer o envio de três bytes numa taxa serial apropriada (2400 ou 9600 baud, dependendo da configuração do "jumper" B; veja Configurando o Mini SSC II). Esses bytes consistem de:

Tabela 3 - Comandos para o “Mini-SSC”.

Byte 1	Byte 2	Byte 3
<sync marker (255)>	<servo # (0- 254)>	<position (0 - 254)>

Eles têm que ser enviados como valores individuais de byte, não como textos com representações de números. O “led” do “Mini SSC II” acenderá quando a energia for inicialmente aplicada e ficará ligado até que o primeiro conjunto de instruções de 3 bytes seja recebido completamente. Depois disso, as luzes do LED acendem somente depois de um válido “sync” marcador e um endereço de servo sejam recebidos. Ele ficará aceso até que a posição do byte seja recebida, em seguida ele desliga. Se o programa está enviando muitos dados para o “Mini SSC II”, a iluminação do LED parecerá estar ligada o tempo todo, mas na verdade elas estão piscando muito rapidamente.

Servos são projetados para uso em carros, barcos e aviões controlados por rádio (R/C). Eles utilizam um sinal que é de fácil transmissão e recepção. O sinal consiste em pulsações que variam de 1 a 2 milisegundos de duração, e são repetidas 60 vezes por segundo (PWM). O servo posiciona o seu ângulo de output em proporção à largura da pulsação, como mostra na Figura 22.

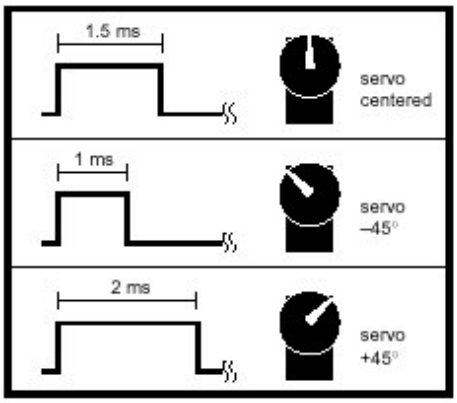


Figura 22 - PWM aplicado.

Nas aplicações em rádio controle, um servo geralmente não precisa de mais de 90 graus de alcance de movimento, uma vez que este usualmente se encontra operando um mecanismo que não pode se mover mais do que 90 graus. Sem nenhum jumper no R do cabeçote (Figura 16), a posição de valor zero corresponde a um pulso de 1 milisegundo e o valor de 254 até 2,16 milisegundos. A alteração mínima no valor de uma posição em uma unidade corresponde a 4 microsegundos de alteração na largura do pulso. A resolução do posicionamento é 0,36 grau / unidade (90 graus/250). A maioria dos servos possui mais de 90 graus de alcance mecânico, então, para

permitir o ajuste das variações dos componentes, da posição de montagem, o “Mini SSC II” permite que seja utilizado este alcance extra. Com um “jumper” na posição R (Figura 16), a posição de valor zero corresponde a uma pulsação de 0,5 milisegundo até 254 que corresponde a 2,53 milisegundos. Com o “jumper” cada unidade passa a ter o valor de 8 microssegundos de alteração na largura do pulso. A resolução do novo posicionamento é de 0,72 grau / unidade (180 graus/250).

3.1.3 Escolha e Construção do Robô

O objetivo deste item é descrever porque o robô “Lynxarm” foi o escolhido para este projeto. O movimento no espaço de trabalho do robô e o método utilizado para o controle do robô, bem como as possibilidades, limitações e a geometria do robô são descritas.

3.1.3.1 Geometria e Limitações do Robô “Lynxmotion”

Este projeto é baseado no kit da “Lynxmotion”. O kit consiste de um robô e uma base. O robô é mostrado na Figura 23, onde é mostrado a vista frontal e lateral do robô.

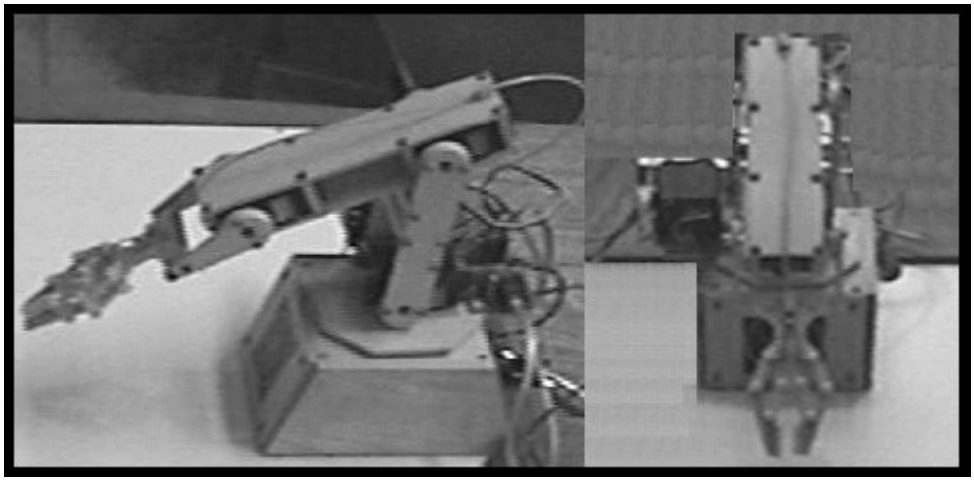


Figura 23 - O robô da “Lynxmotion” usado neste projeto.

Este robô foi projetado para fins didáticos somente, não sendo recomendado para uso industrial. Isso causa alguns problemas em relação à precisão dos movimentos e a possibilidade de reprodução dos movimentos do robô. Os ajustes das partes do robô são um problema no caso de movimentos precisos serem necessários. Isso se dá porque as conexões entre as partes do robô são difíceis de serem montadas precisamente. Este não é o objetivo deste projeto, fazer com que o robô

faça movimentos precisos, mas sim, de ser capaz de controlar os movimentos dentro dos limites do sistema de trabalho adotado pelo sistema de visão.

3.2 Controlando o Robô

Esta seção descreve os ajustes feitos nos ângulos dos servomotores e o método de movimento do robô.

3.2.1 Movendo o Robô

Os movimentos do robô dependem do movimento de suas junta. Há inúmeros métodos possíveis para decidir como mover essas juntas. Alguns métodos calculam o caminho que as juntas do robô devem seguir e outros somente movem uma junta após a outra, ou movem as juntas simultaneamente. Se os servos são movidos com a mesma velocidade simultaneamente, eles não irão parar ao mesmo tempo, a menos que eles tenham os mesmos ângulos para mover. Isto pode levar a um movimento desigual do robô. O método usado neste projeto é mover as juntas simultaneamente, porém com diferentes médias de velocidade. É possível alcançar o ponto final ao mesmo tempo para todos os servomotores. O motivo para o uso da média de velocidade é que os servomotores são controlados ordenando-os numa determinada posição e isso só pode ser feito um a um porque um "serial output" é usado.

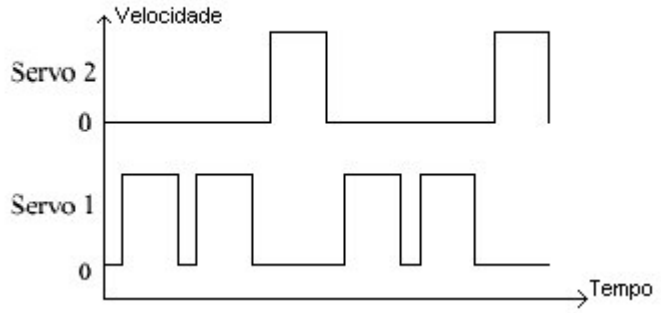


Figura 24 - Sinal de saída de dois servos (a velocidade de servo 1 é o dobro do servo2).

Uma média diferente de velocidade é obtida através do cálculo do número de passos que cada servomotor tem que mover até que realize o movimento. O servo tem que mover 10 passos e o servo dois tem que mover 5 passos. A cada dois passos do servo um, o servo dois move um passo, e assim por diante, como ilustra a Figura 24. Esse método de mover o robô produz um movimento suave sem a utilização de muitos

cálculos, o que é uma vantagem para esse programa. A desvantagem é que não há controle de onde as juntas estão no espaço. Isso não é um problema para este robô, mas pode ser problema no caso de um robô mais flexível, com maior grau de liberdade. O método de movimento controlado pelo robô é determinado, e esta é a base para o programa.

3.3 Área de Trabalho do Robô

Obtém-se a área de trabalho do robô pelos limites dos ângulos e do comprimento dos braços. A área de trabalho é limitada pela base do robô, onde a possível área de trabalho devido aos ângulos é mostrada. Todos os pontos onde o "gripper" pode ser colocado estão dentro da área de trabalho. Para se definir a área de trabalho do robô, foi utilizada a cinemática direta pelos parâmetros de *Dennavit-Hartenberg,* como será demonstrado a seguir. O robô da "Lynxmotion" tem quatro graus de liberdade giratória e uma garra como "gripper". Neste trabalho foram utilizados três graus de liberdade para se obter uma maior precisão em seus movimentos, devido a sua flexibilidade. Um dos movimentos giratórios é o "pivot", e os outros três são movimentos de dobradiça. Este é um robô articulado verticalmente e a cadeia cinética do robô é mostrada na Figura 25.

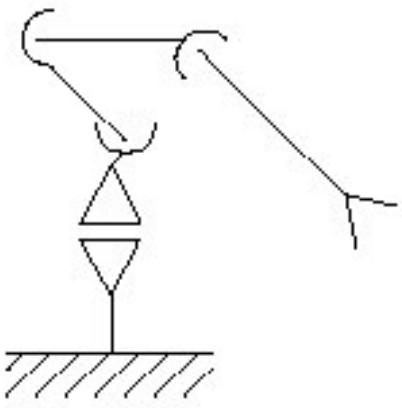


Figura 25 - O desenho esquemático do robô usado neste projeto.

O sistema de coordenadas das juntas e da garra é mostrado na Figura 25, na qual o robô demonstra os sistemas de coordenadas se referem ao robô, na realidade. O "pivot" provoca uma rotação em volta do eixo z e os responsáveis pelo movimento de dobradiça provocam uma rotação em volta do eixo y. O sistema global das coordenadas é posicionado no solo.

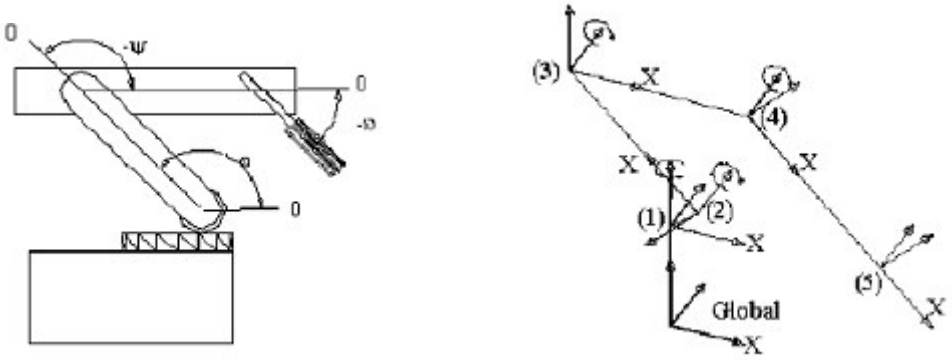


Figura 26 - Desenho do robô e o sistema de eixos coordenados.

O sistema de coordenadas locais é determinado pelo sistema global de coordenadas e as distâncias entre as juntas. A Figura 26 mostra os vetores entre as juntas do robô. O robô é mostrado em duas dimensões porque as distâncias estão presentes somente no plano xz. Os vetores dados na Tabela 4 são os vetores junta a junta relativos ao sistema de coordenadas da primeira junta. Conforme se pode ver na Figura 27.

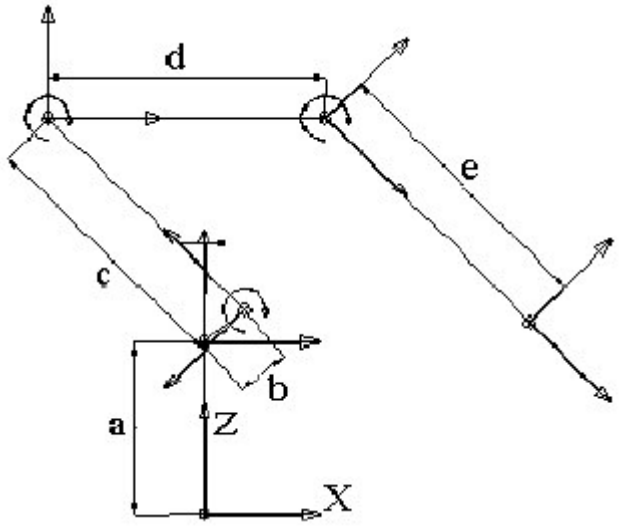


Figura 27 - Os vetores de “a” até “e” entre os pontos de junção do robô.

Tabela 4 - Os valores dos vetores da Figura 27.

	X [mm]	Y [mm]	Z [mm]
a	0	0	65
b	11	0	18
c	95	0	0
d	95	0	0
e	120	0	0

3.3.1 Ângulos do Robô

Para se determinar a posição dos servos, é necessário achar o ângulo entre o sistema de coordenadas e defini-los. Os ângulos, suas direções, e a posição zero são demonstrados na Figura 28.

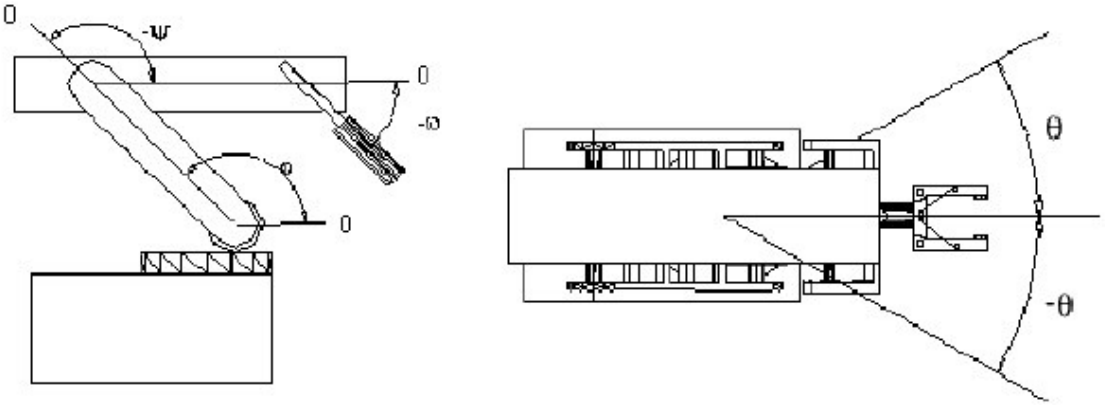


Figura 28 - Os ângulos do robô e sua direção.

3.3.2 Matrizes Relacionadas ao Sistema de Coordenadas

Utilizando-se os parâmetros de Denavit-Hartenberg, foram calculadas as matrizes dos sistemas de coordenadas do robô.

$$ {}^{i-1}T_i = \begin{bmatrix} \cos\theta_i & -\cos\alpha_i \cdot \operatorname{sen}\theta_i & \operatorname{sen}\alpha_i \cdot \operatorname{sen}\theta_i & a_i \cdot \cos\theta_i \\ \operatorname{sen}\theta_i & \cos\alpha_i \cdot \cos\theta_i & -\operatorname{sen}\alpha_i \cdot \operatorname{sen}\theta_i & a_i \cdot \operatorname{sen}\theta_i \\ 0 & \operatorname{sen}\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(4)} $$

Tem-se a matriz do sistema de coordenadas B em relação ao a.

$$ B = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 65 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(5)} $$

A equação cinco é derivada da rotação em volta do eixo z do sistema de coordenadas globais. Quando o primeiro sistema de coordenadas é rotacionado θ em volta do eixo z, a direção do eixo x do primeiro sistema de coordenadas é também rotacionado θ. A direção do eixo x é então [cos(θ), sen(θ),0], como é mostrado na

matriz 5 . Não há nenhum componente no eixo z porque o sistema de coordenadas é rotacionado em volta deste eixo. O eixo y é rotacionado 90° em relação ao eixo x e no plano xy é descrito por [- sen(θ), cos(θ), 0]. Os pontos do eixo z ainda estão na direção z e é, portanto, dado por [0, 0, 1]. A última coluna da matriz dá a posição do primeiro sistema de coordenadas com relação ao global, que é de apenas 65mm de deslocamento na direção z. A seguir, a matriz para a primeira dobradiça, ou seja, a matriz do sistema C em relação ao B.

$$C = \begin{bmatrix} \cos\varphi & 0 & -\sin\varphi & 11 \\ 0 & 1 & 0 & 0 \\ \sin\varphi & 0 & \cos\varphi & 18 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(6)}$$

Para este sistema de coordenadas, a rotação é feita em volta do eixo y. Não há portanto diferença no eixo y. O eixo x possui a mesma direção do sistema de coordenadas anterior se φ é zero. A rotação é descrita como mostra a matriz C da equação seis. A origem para esse sistema de coordenadas é estabelecida [11, 0, 18] em relação ao anterior.

O terceiro sistema de coordenadas é rotacionado em volta do eixo y, e a origem é estabelecida [95, 0, 0] em relação ao anterior. Como resultado, a próxima matriz do sistema D em relação ao C.

$$D = \begin{bmatrix} \cos\varphi & 0 & -\sin\varphi & 95 \\ 0 & 1 & 0 & 0 \\ \sin\varphi & 0 & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(7)}$$

O quarto sistema de coordenadas é rotacionado em volta do eixo y e a origem é estabelecida [95, 0, 0] em relação ao anterior. Como resultado, a próxima matriz do sistema E em relação ao D.

$$E = \begin{bmatrix} \cos\omega & 0 & -\sin\omega & 95 \\ 0 & 1 & 0 & 0 \\ \sin\omega & 0 & \cos\omega & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(8)}$$

O quinto e último sistema de coordenadas dá a posição do “gripper” em relação ao sistema de coordenada anterior. Não há, portanto rotação, mas apenas um deslocamento na direção x em relação ao sistema de coordenada anterior. Como resultado, a próxima matriz do sistema F em relação ao E.

$$F = \begin{bmatrix} 1 & 0 & 0 & 120 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \textbf{(9)}$$

Como se pode notar, essa matriz consiste somente de constantes.

As matrizes relacionadas ao sistema de coordenadas são agora determinadas, e é possível determinar os ângulos, a fim de se alcançar os pontos no espaço.

3.3.3 Procurando um Ponto no Plano X,Y

Este item trata de como o robô atinge um ponto no plano xy. Existem muitos métodos para se encontrar os ângulos entre os braços do robô. O mais óbvio, já descrito aqui, é utilizar as matrizes matemáticas, e através das matrizes estabelecer uma relação entre os ângulos do robô e seu eixo de coordenadas X,Y,Z. O que se quer é guiar o robô para um ponto no plano xy. Esses dados fornecem a matriz genérica do sistema.

$$\begin{bmatrix} x_1 & y_1 & z_1 & p \\ x_2 & y_2 & z_2 & q \\ x_3 & y_3 & z_3 & s \\ 0 & 0 & 0 & 1 \end{bmatrix} = B * C * D * E * F \quad \textbf{(10)}$$

p,q,s: correspondem aos pontos X,Y,Z do posicionamento da extremidade do “gripper” do robô.

B, C, D, E, F: são matrizes dos sistemas de coordenadas locais.

x1, x2, x3: são os componentes do eixo x no sistema de coordenadas, descrevendo a posição do “gripper ”em relação ao sistema de coordenadas global.

y1, y2, y3: são os componentes do eixo y no sistema de coordenadas, descrevendo a posição do “gripper ”em relação ao sistema de coordenadas global.

z1, z2, z3: são os componentes do eixo z no sistema de coordenadas, descrevendo a posição do “gripper”em relação ao sistema de coordenadas global.

A matriz mostra a posição e orientação do “gripper”, em relação ao sistema de coordenadas globais.

3.3.4 Utilizando o Software “MAPLE” para o Cálculo da Cinemática Direta

Será explanado nesta parte do trabalho como foi realizado o cálculo da cinemática direta do robô, o qual vai gerar equações relacionadas com os pontos X,Y,Z do sistema de coordenadas do robô. Estas equações serão implementadas em um programa em linguagem C^{++} que permitirá encontrar os ângulos necessários para que o robô atinja os pontos de coordenada onde se encontra o objeto que ele vai capturar. Estas coordenadas ficarão dispostas no programa fonte em forma de um “look up table” que é um mapeamento da região de trabalho do robô.

Para cada braço do robô, é encontrado uma matriz a partir da matriz genérica pelos parâmetros de Denavit-Hartenberg, sendo que a multiplicação das matrizes vai gerar uma matriz onde, para determinados valores dos ângulos do robô, poderá ser encontrada a posição no espaço X,Y,Z que a garra do robô vai estar. Esta relação é importante, pois o robô será usado para pegar um objeto que estará no plano. A coordenada Z do sistema vai ter um valor constante que corresponde à região do plano que o robô vai estar.

Agora serão renomeados os ângulos mostrados na Figura 28 da seguinte forma:

$\theta = m$

$\varphi = n$

$\psi = p$

$\omega = q$

Para se trabalhar com o robô com 3 graus de liberdade, adotou-se o ângulo (q) = 40 graus. Portanto, a relação a seguir permitirá o cálculo de todos os movimentos do robô no espaço pela variação dos ângulos m,n e p. Este artifício foi usado para minimizar o tempo de resposta pelo programa utilizado, visto que não impede o robô de se deslocar pela sua região de trabalho. Gerando as matrizes e fazendo a multiplicação delas através da utilização do software “Maple”, versão 5.0, tem-se:

A matriz 1.

$$T1 = \begin{bmatrix} \cos(m) & -sin(m) & 0 & 0 \\ sin(m) & \cos(m) & 0 & 0 \\ 0 & 0 & 1 & 65 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(11)}$$

A matriz 2.

$$T2 = \begin{bmatrix} cos(n) & 0 & -sin(n) & 11 \\ 0 & 1 & 0 & 0 \\ sin(n) & 0 & cos(n) & 18 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(12)}$$

A matriz 3.

$$T3 = \begin{bmatrix} cos(p) & 0 & -sin(p) & 95 \\ 0 & 1 & 0 & 0 \\ sin(p) & 0 & cos(p) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(13)}$$

A matriz 4.

$$T4 = \begin{bmatrix} cos(q) & 0 & -sin(q) & 95 \\ 0 & 1 & 0 & 0 \\ sin(q) & 0 & cos(q) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(14)}$$

A matriz 5.

$$T5 = \begin{bmatrix} 1 & 0 & 0 & 120 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(15)}$$

T(Genérica) = T1 x T2 X T3 xT4x T5,

onde T é a matriz do sistema robótico procurada de onde sairão as equações para o sistema de coordenadas.

Esta corresponde a matriz calculada T = T1 x T2 X T3 xT4x T5,

$$\begin{pmatrix} (C - B)\cos(q) + A\sin(q), & -\sin(m), & -(C - B)\sin(q) + A\cos(q), & 120\,(C - B)\cos(q) + 120\,A\sin(q) + 95\,C - 95\,B + 95\cos(m)\cos(n) + 11\cos(m) \\ (F - E)\cos(q) + D\sin(q), & \cos(m), & -(F - E)\sin(q) + D\cos(q), & 120\,(F - E)\cos(q) + 120\,D\sin(q) + 95\,F - 95\,E + 95\sin(m)\cos(n) + 11\sin(m) \\ G, & 0, & H, & 120\,(\sin(n)\cos(p) + \cos(n)\sin(p))\cos(q) + 120\,(-\sin(n)\sin(p) + \cos(n)\cos(p))\sin(q) + 83 + 95\sin(n)\cos(p) + 95\cos(n)\sin(p) + 95\sin(n) \\ 0, & 0, & 0,1 & \end{pmatrix} \quad \textbf{(16)}$$

Fazendo:

A = -cos(m) cos(n) sin(p) - cos(m) sin(n) cos(p)

B = cos(m) sin(n) sin(p)

C = cos(m) cos(n) cos(p)

D = -sin(m) cos(n) sin(p) - sin(m) sin(n) cos(p)

E = sin(m) sin(n) sin(p)

F = sin(m) cos(n) cos(p)

G = (sin(n) cos(p) + cos(n) sin(p)) cos(q) + (-sin(n) sin(p) + cos(n) cos(p)) sin(q)

H = -(sin(n) cos(p) + cos(n) sin(p)) sin(q) + (-sin(n) sin(p) + cos(n) cos(p)) cos(q)

(q) = 40 graus

Da matriz calculada, é retirado o seguinte sistema de equações do “Gripper” do Robô:

$$X = 120\,(C - B)\cos(q) + 120\,A\sin(q) + 95\,C - 95\,B + 95\cos(m)\cos(n) + 11\cos(m) \quad \textbf{(17)}$$

$$Y = 120\,(F - E)\cos(q) + 120\,D\sin(q) + 95\,F - 95\,E + 95\sin(m)\cos(n) + 11\sin(m) \quad \textbf{(18)}$$

$$Z = 120\,(\sin(n)\cos(p) + \cos(n)\sin(p))\cos(q) + 120\,(-\sin(n)\sin(p) + \cos(n)\cos(p))\sin(q) + 83 + 95\sin(n)\cos(p) + 95\cos(n)\sin(p) + 95\sin(n) \quad \textbf{(19)}$$

Este sistema de equações será implementado no programa em linguagem C^{++}, conforme código fonte inserido no apêndice deste trabalho; o qual, calculará para cada ângulo m,n,p os respectivos valores de X,Y, Z do sistema de coordenadas do robô, permitindo inserir os valores no “look up table” no programa de controle do robô. Convém acrescentar que o valor Z é praticamente constante, pois a peça a ser capturada pertence a uma região do plano X,Y de coordenadas.

4 PROGRAMA DE CONTROLE DO SISTEMA DE VISÃO E DO ROBÔ

Neste capítulo, será abordada a lógica do programa feito em linguagem visual C^{++} da Microsoft. Pode-se demonstrar na Figura 29 o fluxograma de todo o processo do programa desenvolvido.

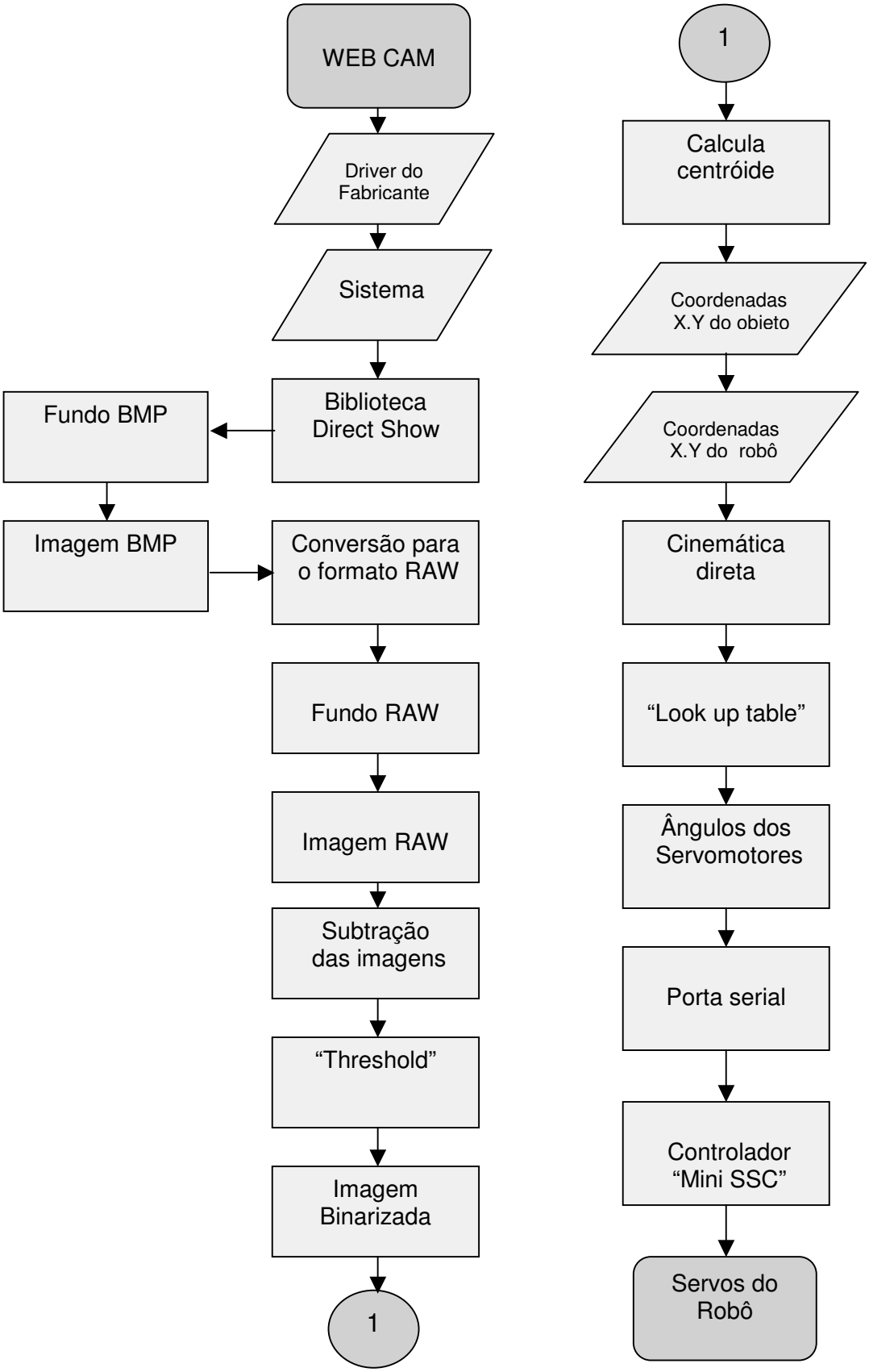


Figura 29 – Fluxograma do processo.

O programa está dividido em três partes principais A, B e C. Temos como entrada do programa uma “Web Cam” e como saída um robô. Na Figura 30 é apresentada a estrutura básica do sistema desenvolvido. Inicialmente a imagem é adquirida de uma “Web Cam”, onde o sistema de visão se encarrega de obter uma imagem e guardar as informações da mesma em um arquivo. No segundo passo temos a parte B do programa, que consiste no sistema de processamento de imagens. Esta parte é responsável pelo processamento da imagem e por transformar a sua posição em uma coordenada X,Y e por manter disponível este valor.

Na parte C do programa, temos o programa de controle do robô, que será responsável pela coleta dos dados das coordenadas X,Y da imagem e pelo envio do robô fisicamente para esta posição. Também fica contida nesta parte a rotina de programação do robô, para que ele pegue o objeto e leve ao local desejado e, após, retorne à posição inicial, onde ficará aguardando um próximo evento.

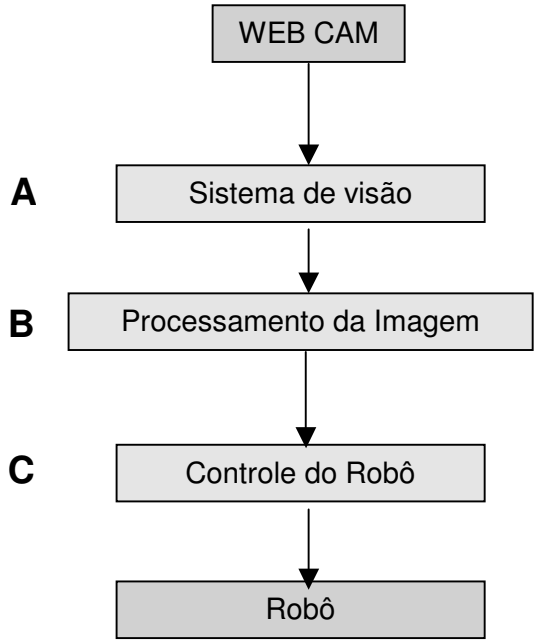


Figura 30 - Estrutura básica do sistema desenvolvido.

Teremos a seguir um detalhamento das partes A, B e C do programa desenvolvido.

4.1 Sistema de Visão

A respeito do sistema de visão, pode-se dizer que é o processo de aquisição de uma imagem. Para isso, diferentes dispositivos de captura de imagem podem ser utilizados, como por exemplo, uma "web cam" ou mesmo utilizando uma placa de captura de vídeo como uma “Frame Grabber” ou placas específicas para uso industrial. Neste trabalho, utilizou-se uma “web cam” trabalhando no sistema operacional windows 98, da microsoft . Utilizou-se a bliblioteca da “Microsoft” chamada “Direct Show” que está integrada no sistema operacional windows 98.

4.1.1 “Direct Show”

O “Direct Show faz uma interface entre o hardware de visão, “web cam” ou “frame grabber”, e o programa principal. Isto permite ao programa ficar independente do sistema de visão, podendo-se utilizar qualquer um dos dispositivos de captura de vídeo descritos, e até mesmo utilizar-se direto a conexão “USB” do microcomputador, sendo então mais versátil e multifuncional para ser usado nos setores industriais. Na Figura 31 tem-se uma introdução ao “direct show” que faz parte do sistema de visão do programa.

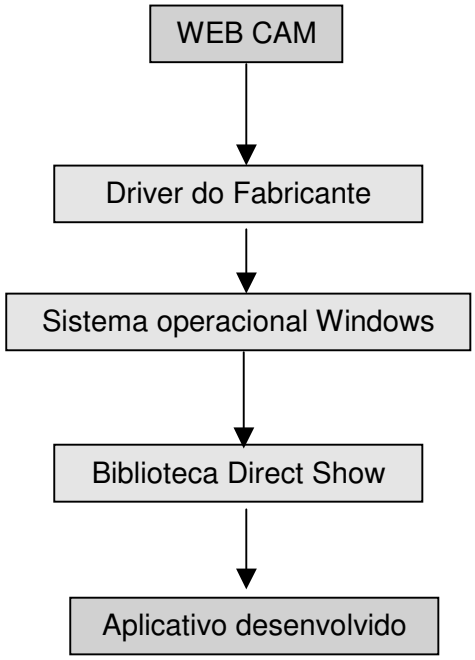


Figura 31 - Introdução ao “Direct Show”.

4.1.2 Funcionamento do Sistema de Visão

Sobre o sistema de visão e seu funcionamento, pode-se dizer, conforme a Figura 32, que explica o item “A” do programa.

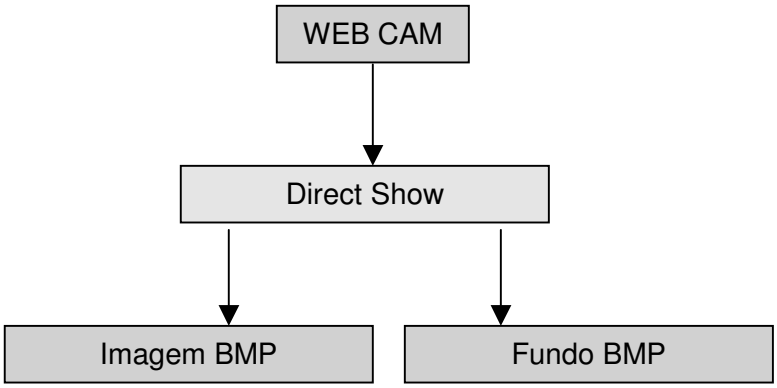


Figura 32 - Sistema de visão.

O programa tem como entrada as imagens enviadas pela “web cam”, e como saída são geradas duas imagens no formato BMP, que serão utilizadas posteriormente. As imagens são gravadas através do “Direct Show” no formato BMP, com o nome de “Imagem.bmp” e “fundo.bmp”. A resolução das imagens, bem como a quantidade de cores, são pré-ajustadas. Para isso, utilizam-se rotinas da biblioteca do “Direct show”.

4.2 Processamento das Imagens

Será detalhada agora a parte B do programa, ou seja, o “Processamento da Imagem”. Nesta parte é onde ocorre todo o tratamento da imagem, onde a imagem é adquirida no formato BMP, será tratada e transformada para 256 tons de cinza e posteriormente binarizada pelo processo de “threshold” para que se possa calcular sua posição X,Y do espaço de trabalho.

Para transformar a imagem inicialmente “R,G,B” de 16 milhões de cores para uma imagem de 256 tons de cinza, utilizou-se a soma dos tons de cores, dividindo por três da seguinte forma: R (256 tons de vermelho), G (256 tons de verde), B(256 tons de azul). Por exemplo, uma imagem colorida RGB (254,45,67), ficaria (254+45+67)/3 = 122 tons de cinza. Este método foi utilizado na simplificação do formato BMP para o formato de imagem RAW que é um formato de 256 tons de cinza. A seguir temos o

fluxograma na Figura 33, que detalha a parte do programa onde ocorre o processamento das imagens.

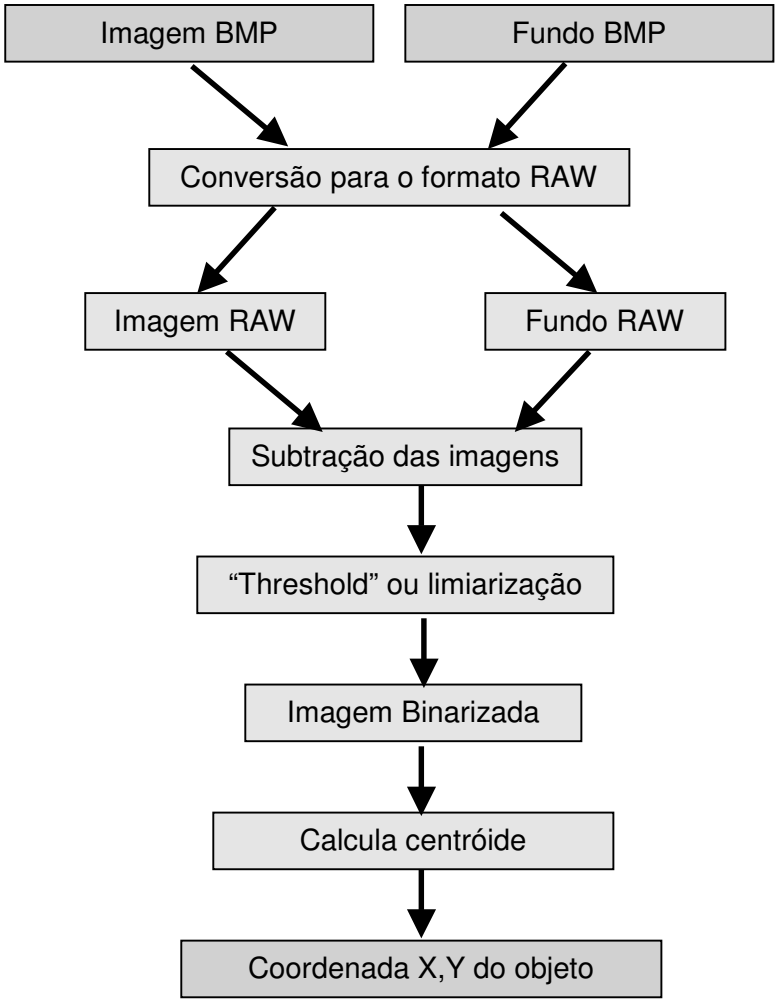


Figura 33 - Processamento da imagem.

A imagem no formato RAW consiste em um vetor onde cada pixel é representado de 0 a 255, totalizando 256 tons de cinza. Na subtração das imagens são considerados os valores “pixel” a “pixel”, tendo-se como resultado a imagem do objeto, que é colocado na área de trabalho do sistema de visão já com 256 tons de cinza, ou seja, no formato RAW. Em seguida é feito o “threshold”, onde a imagem é binarizada e cada pixel da mesma será formado por apenas um bit, sendo preto igual a zero e branco igual a um. Para uma melhor compreensão, pode-se dizer conforme a Tabela 5 a seguir:

Tabela 5 – Relação entre (bits X cores)

1 bit	=	2 cores
8 bits	=	256 cores
24 bits	=	16 milhões de cores

Após o “Threshold”, a imagem binarizada passa por uma contagem binária onde se calcula a sua área e a centróide em coordenadas X,Y. O valor das coordenadas X,Y são enviados para o registro do windows onde fica armazenado para ser utilizado no próximo passo do programa.

4.3 Controle do Robô no Sistema

No passo C do programa, conforme a subdivisão feita anteriormente, será visto o processo do recebimento das coordenadas X,Y da imagem e transformadas nas coordenadas X,Y do robô, ou seja, para onde o robô irá se deslocar para pegar o objeto e capturá-lo, levando-o para a posição desejada. As coordenadas dos pontos do robô foram obtidas pela aplicação da cinemática direta, onde pode-se obter a região de trabalho do robô chamada de “look up table”. As coordenadas X,Y do robô são convertidas em ângulos para servomotores, que na realidade são valores que para cada servomotor varia de 0 a 255, onde cada valor vale aproximadamente 0,36 graus conforme descrito anteriormente. A posição dos ângulos dos servomotores será enviada para a porta serial do microcomputador que se comunica com o microprocessador do robô, o “mini-SSC”. Ocorre então a transformação da posição dos ângulos do robô para “PWM” (modulação de largura de pulso), onde os pulsos controlados a uma determinada tensão são enviados para os servomotores do robô possibilitando-o se deslocar para a posição desejada. Na Figura 34, a seguir, podemos visualizar melhor à parte “C”.

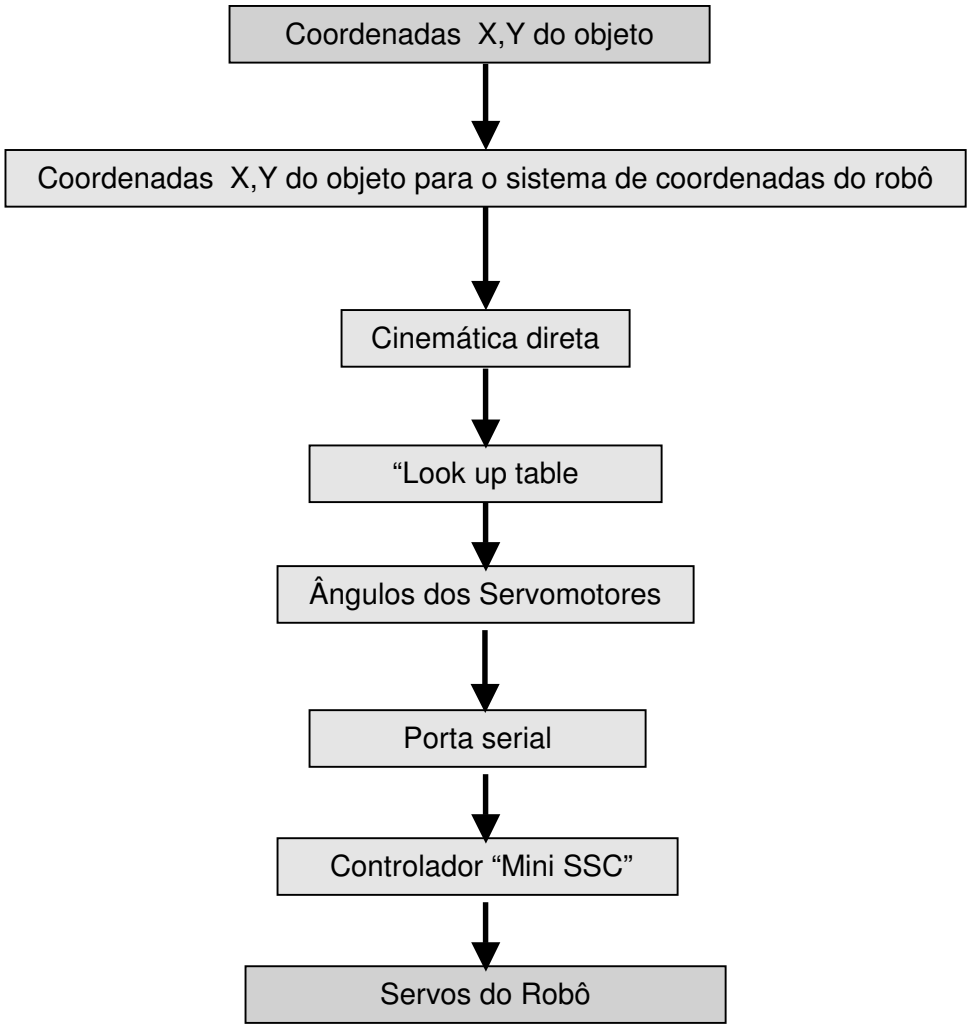


Figura 34 - Controle do robô

4.4 Visualização Gráfica do Programa Desenvolvido

A seguir a visualização gráfica do programa, bem como a explicação de cada parte do mesmo. Como o programa foi construído com a linguagem visual C^{++}, construiu-se uma interface gráfica, conforme Figura 35, que permite ao usuário, além de colocar objetos para serem capturados pelo robô, poder fazer passo a passo as operações de captura de imagem, o cálculo do centróide, bem como a movimentação livre do robô pelo controle individual dos seus servomotores.

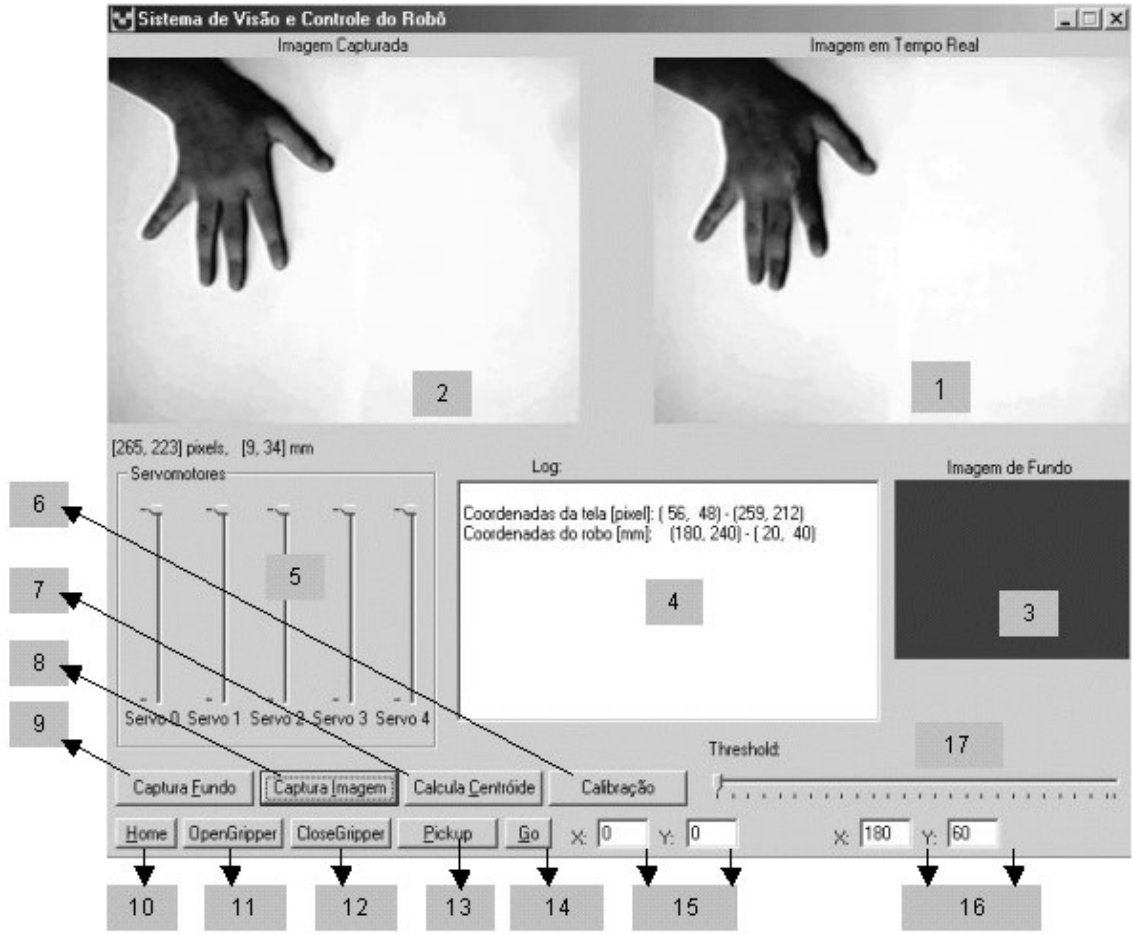


Figura 35 - Visualização gráfica da tela do programa.

Numerou-se de 1 a 17 as principais partes do programa, onde descreveu-se cada uma delas, e em seguida realizou-se uma simulação seqüencial do funcionamento do programa, com um exemplo que será dado a seguir.

1. Imagem em tempo real: é a cena que está ocorrendo, ou seja, o que está sendo visto pela câmera.
2. Imagem capturada: é a imagem da figura a qual se deseja achar o centro de massa em coordenadas X,Y para ser capturada pelo robô.
3. Imagem de fundo: é a imagem que está no campo de ação visual antes de ocorrer o evento do objeto a ser colocado.
4. Tela em Branco: mostra as coordenadas da tela em pixels, em mm e as coordenadas do robô em mm.
5. Servomotores: esta tela mostra o controle manual dos cinco servomotores.
6. Calibração: este botão é utilizado para calibração do sistema estabelecendo uma relação entre os sistemas de coordenadas da tela e o sistema real.

7. Calcula Centróide: este botão quando pressionado pelo usuário aciona o código do programa que encontra o valor numérico do centróide.
8. Captura imagem: botão que serve para capturar a imagem estática do evento.
9. Captura Fundo: este botão captura o fundo inicial do sistema, isto quer dizer que mesmo se for mudado a cor ou o relevo do fundo, o sistema consegue notar esta diferença e fazer a compensação no sistema.
10. Home: botão que ao ser pressionado leva o robô à posição inicial de trabalho.
11. “Open_Gripper”: ao ser pressionado abre totalmente o “gripper” do robô.
12. Close_Gripper : ao ser pressionado fecha totalmente o “gripper” do robô.
13. “Pick_up” : ao ser pressionado, permite que o robô execute a rotina de pegar a peça na região definida pelo sistema de visão, e colocar a mesma em um cesto conforme a programação.
14. “Go”: botão que ao ser pressionado leva o robô às coordenadas que podem ser definidas pelo usuário independente do sistema de visão.
15. X,Y sistema: posição x e y desenvolvida para colocar os valores numéricos calculados pelo botão. Calcula centróide em mm.
16. X,Y do robô: posição x e y desenvolvida para colocar os valores numéricos do eixo de coordenadas do robô em mm. É o local onde o robô pega o objeto no sistema de coordenadas do robô.
17. “Threshold” : esta barra de rolagem permite o ajuste manual do “threshold”, que pode variar de zero a 255.

4.5 Simulação do Programa

Agora será abordado o programa e detalhado o seu funcionamento no que diz respeito à captura da imagem. Os botões serão acionados para que o programa possa ser executado completamente.

4.5.1 Calibração do Sistema de Coordenadas.

Inicialmente, faz-se a conversão do sistema de coordenadas, para que o programa possa alterar o sistema de coordenadas de “pixels” para mm. Para isso é necessário seguir os dados de entrada requeridos pelo sistema, conforme se pode ver nas Figuras 36, 37 e 38.

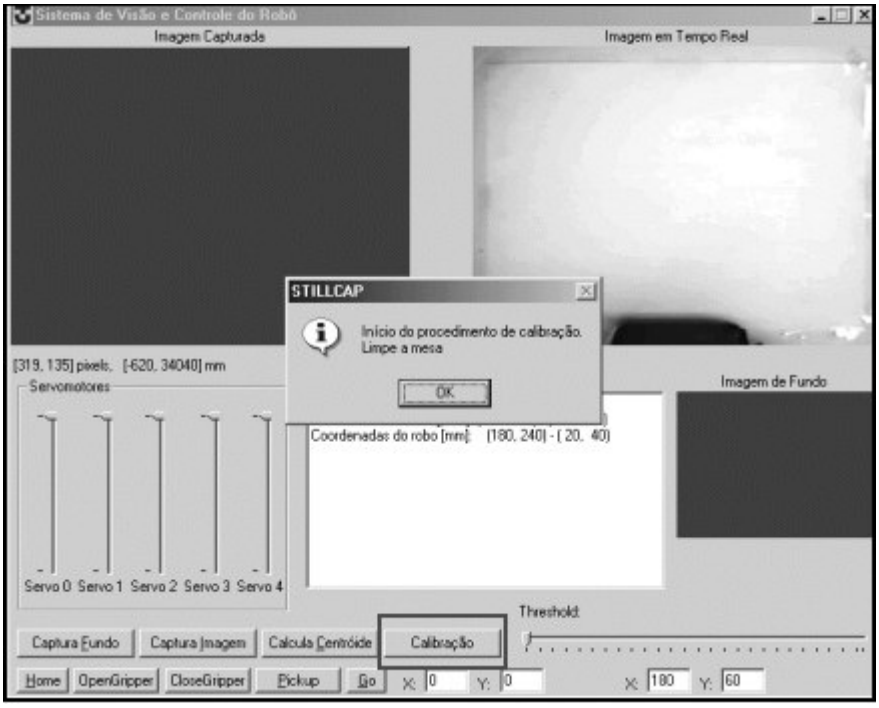


Figura 36 - Calibração do Sistema

No início do procedimento de calibração é necessário limpar a área de trabalho.

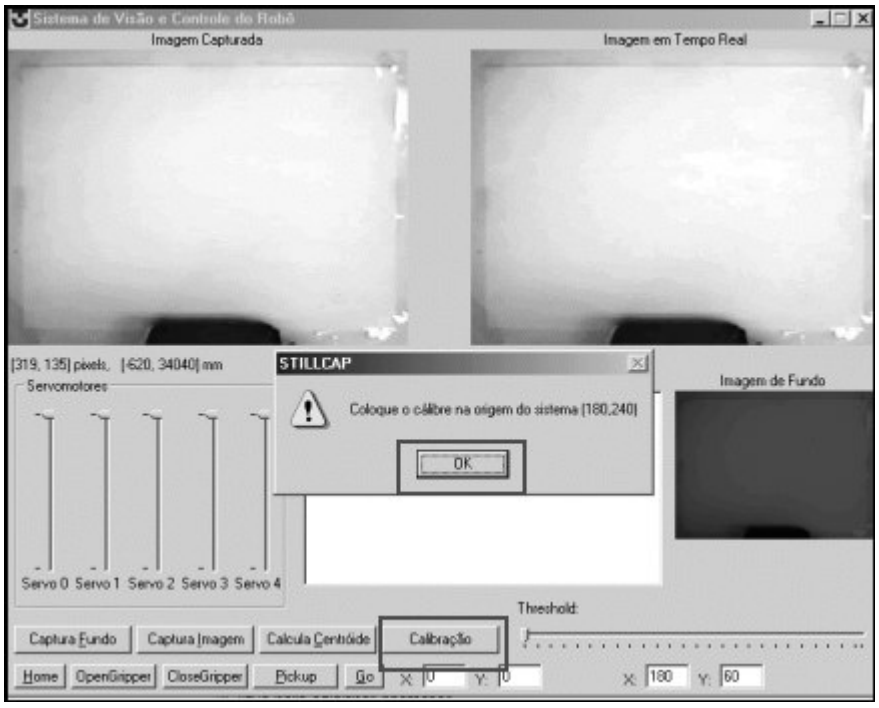


Figura 37 – Continuação da Calibração do Sistema

Em seguida, deve-se colocar o objeto na posição inicial do sistema de coordenadas do robô, e clicar “ok” com o mouse ou apertar a tecla “enter” do PC.

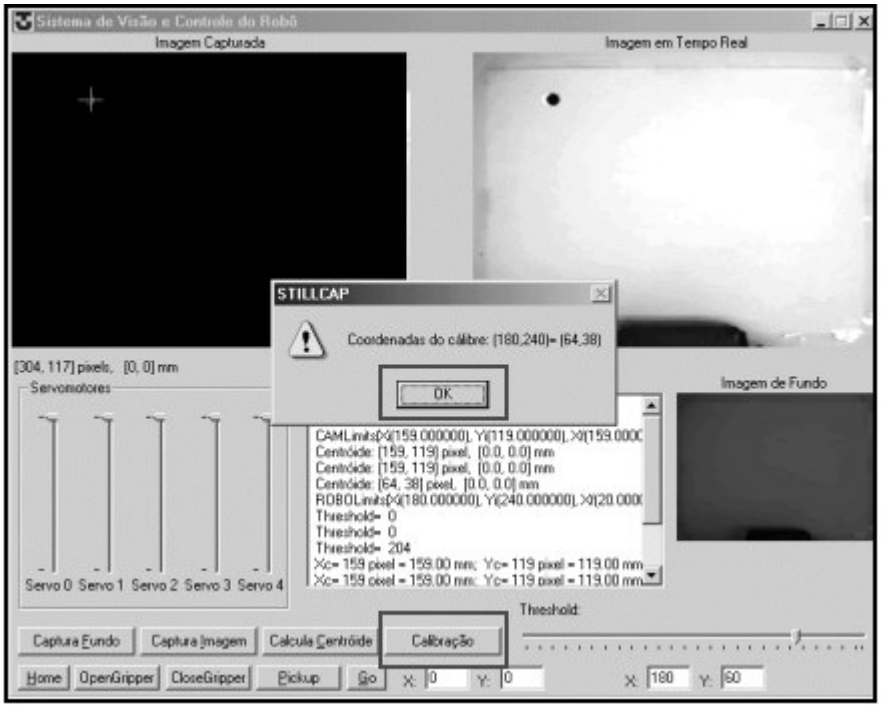


Figura 38 – Finalização da Calibração do Sistema

Nesta fase, o sistema calcula a coordenada x,y do primeiro calibre utilizado.

Este procedimento se repete quando o calibre é colocado no extremo do sistema em uma coordenada conhecida. Com posse dessas duas coordenadas calculadas, o sistema calcula a relação de pixel para mm, e consegue reconhecer o sistema de coordenada real de atuação do robô.

4.5.2 Capturando a Imagem da Tela de Fundo

Nesta fase, o programa captura o fundo, que será utilizado no momento em que a peça é capturada pelo robô. Nota-se que foi colocado o cesto de depósito onde o robô colocará a peça. Uma vez que o sistema capturou este novo fundo, agora ele só “enxergará” a peça que será colocada posteriormente, como se pode ver na Figura 39.

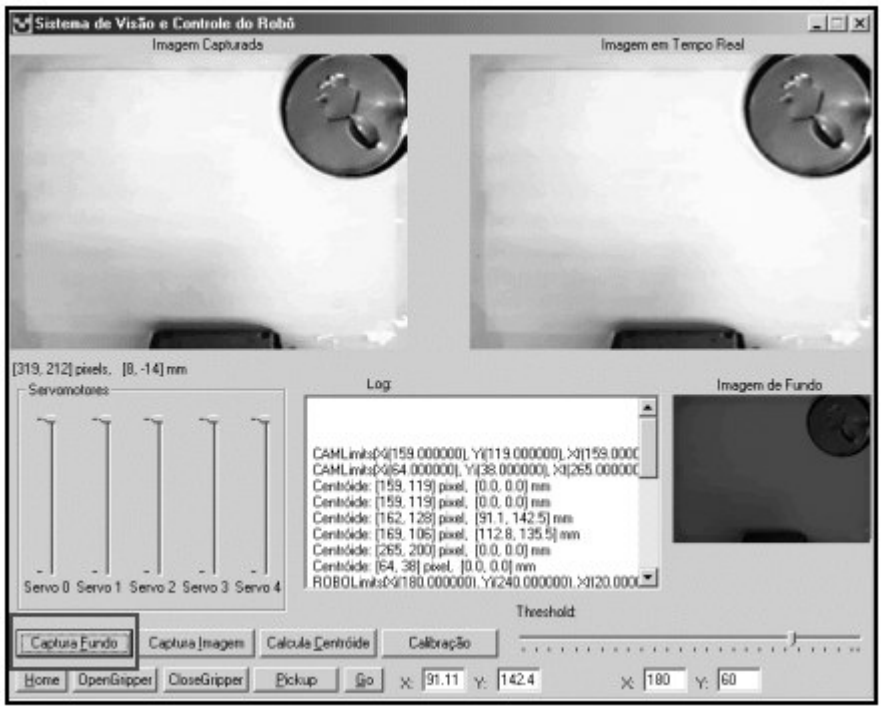


Figura 39 - Captura fundo.

4.5.3 Capturando a Imagem no Campo de Visão da Câmera

Nesta parte da simulação a peça é introduzida no sistema e o botão captura imagem é pressionado, conforme se pode ver na Figura 40.

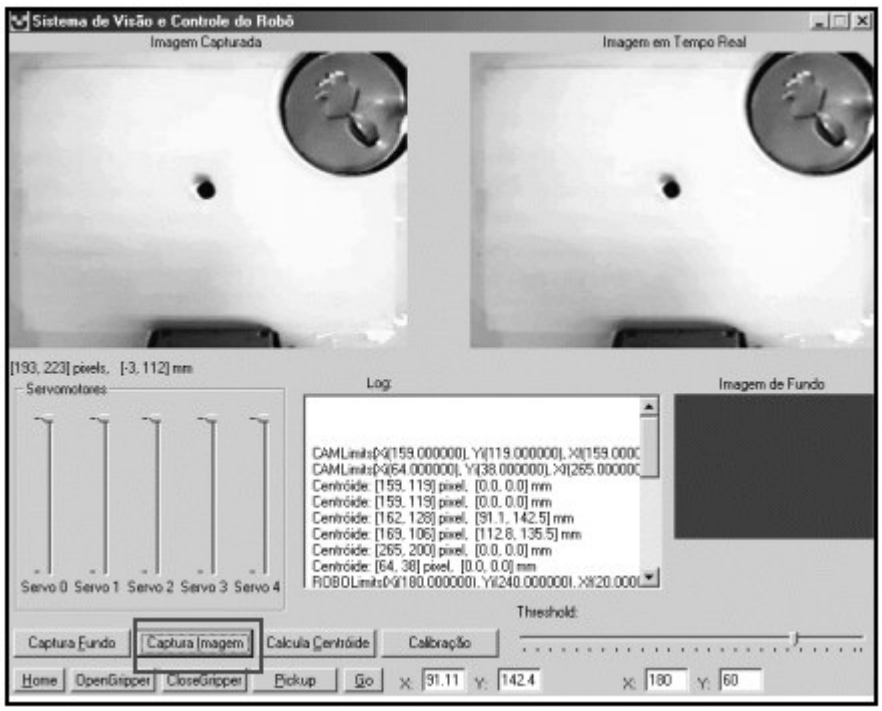


Figura 40 - Captura da Imagem do Objeto

4.5.4 Capturando a Centróide da Imagem do Objeto

Ao ser pressionado o captura centróide, o programa calcula as coordenadas X,Y da peça, enviando os dados para a memória do programa, conforme se pode ver na Figura 41.

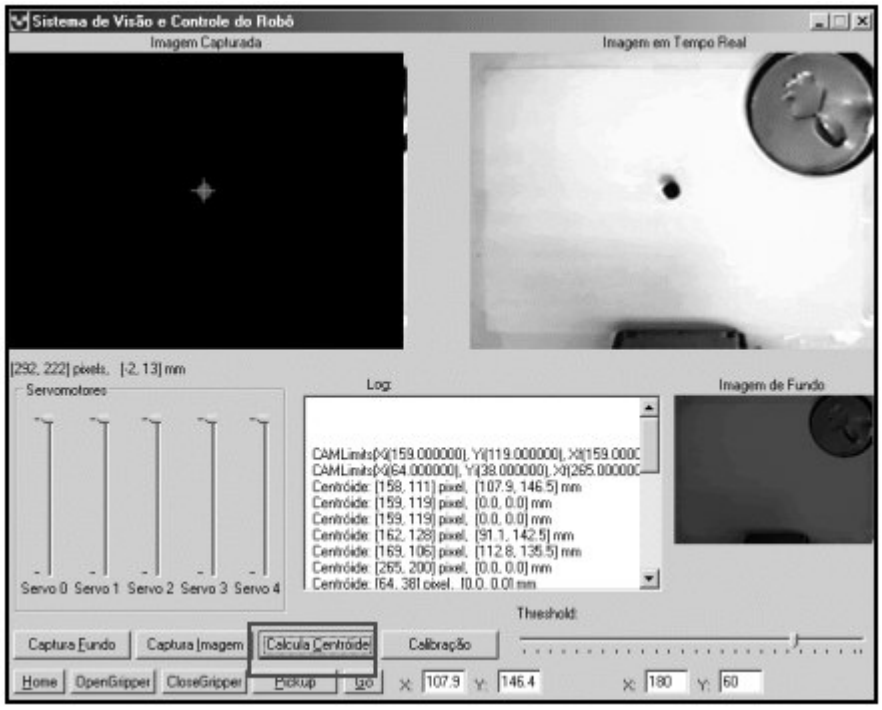


Figura 41 - Calcula Centróide

4.5.5 Comando “Pick Up”

Ao ser pressionado o “pickup”, o robô retira os dados da posição da peça que estavam na memória e executa uma rotina que está em sua programação, que é pegar a peça e levá-la para um lugar onde será depositada. Após o robô executar sua tarefa, ele volta à posição de repouso ficando fora do campo visual do sistema de visão, conforme se pode ver nas Figuras 42, 43 e 44.

Figura 42 - Comando “Pick up”

A Figura 43 mostra o robô depositando a peça no recipiente desejado.

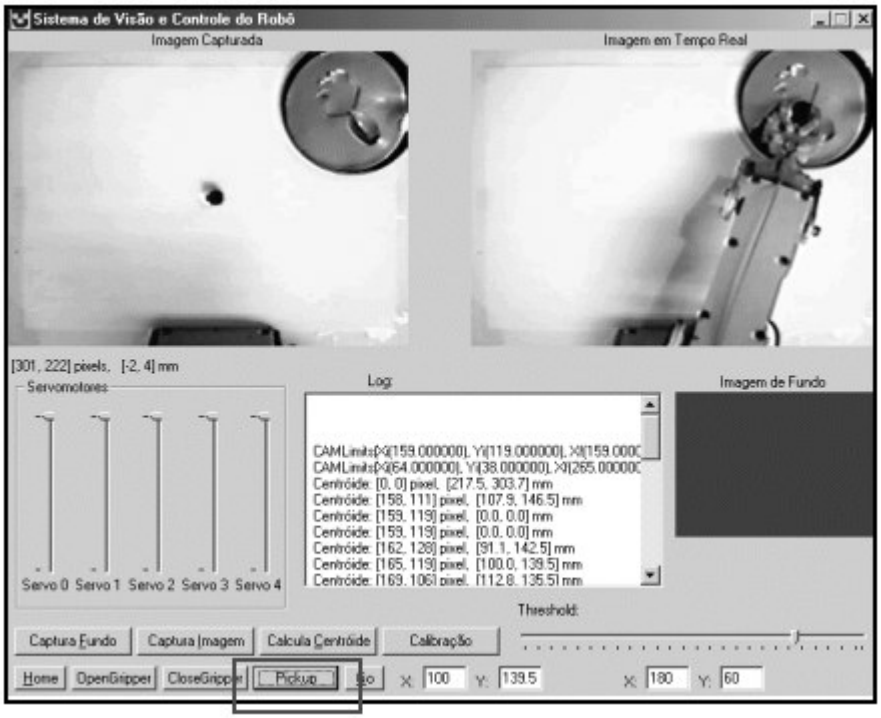


Figura 43 - Robô Depositando a Peça

Após o depósito da peça, o robô retorna à posição “Home” (posição inicial), e fica aguardando o próximo evento.

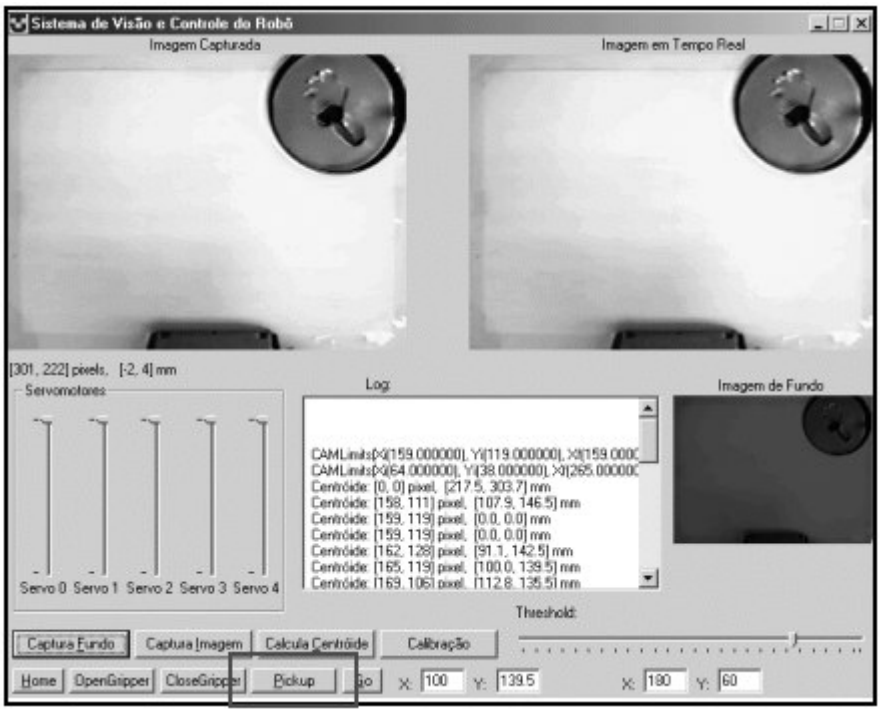


Figura 44 - Robô na posição inicial

5 CONCLUSÃO E PROPOSTA PARA TRABALHOS FUTUROS.

No início, havia um grande desafio e muitos caminhos para a solução do problema, que era tornar acessível para um robô poder "ver" um objeto colocado em sua área de trabalho, e colocar o objeto em um local determinado.

Para isso era necessário escolher um programa de múltipla interface, ou seja, que suportasse um sistema de visão e que também fosse capaz de controlar o robô.

O trabalho começou a ser desenvolvido em linguagem C^{++} versão 3.1 da Borland, onde se programou o robô, independente do sistema de visão. Seria utilizada uma placa de captura de vídeo (frame grabber) do fabricante "Humusoft" devido a sua utilização em diversos outros trabalhos de visão robótica, o que possibilitava acesso a um grande material de pesquisa. Mas foi descartado este tipo de aquisição de imagem, devido a esta placa não estar sendo mais fabricada, e que por trabalhar no sistema operacional "DOS", possui limitações de memória, o que poderia prejudicar o funcionamento do projeto em sua finalização.

Foi encontrado como alternativa desenvolver o restante da programação em linguagem Visual C^{++} versão 6.0 da Microsoft, um software moderno, que com sua linguagem visual aceitaria a migração de parte do programa que já havia sido desenvolvido, com algumas mudanças no seu código fonte. Outra vantagem oferecida por este software é a elaboração de um programa de captura de imagens que dispensaria o uso de uma placa específica de captura, podendo-se capturar uma imagem diretamente da entrada USB do microcomputador, ou também utilizar uma placa de captura com uma câmera e com o "driver" do fabricante.

No projeto construído utilizou-se uma "web cam" com aquisição de imagens através da entrada "USB" do micro. Outra vantagem deste programa é que permite acessar as imagens através das bibliotecas do "Direct Show", que são as rotinas utilizadas no programa na captura da imagem. Para o tratamento de imagens foram desenvolvidos algoritmos que transformam a imagem BMP para o formato RAW, para ser possível realizar os cálculos da posição do centróide da imagem.

Verificou-se também, que o projeto satisfez a condição de um aplicativo de fácil compreensão, e que o usuário não necessita de conhecimento de programação para operar o programa devido ao seu caráter intuitivo. Notou-se também, que o robô utilizado conseguiu realizar sua tarefa de manipulação do objeto colocado em seu campo de trabalho com eficiência, sendo isto difícil de conseguir quando se trabalha com um robô que utiliza motores "DC" com malha aberta. Mas observou-se que com a metodologia aplicada ao robô, ele conseguiu efetuar sua tarefa de pegar e colocar um objeto sem cometer falhas. Notou-se empiricamente que o processo de cálculo da

posição da peça pelo sistema de visão foi satisfatório, indicando sempre a posição correta do objeto colocado no campo de visão da câmera. Observou-se que com a variação da luminosidade do ambiente o sistema conseguia compensar com seu ajuste automático do “threshold”, obtendo um ótimo desempenho, sempre fazendo as correções adequadas, o que permitia o cálculo da posição da peça.

5.1 Contribuição do Trabalho

Após ser concluído o trabalho, notou-se que sua contribuição foi a captura de imagens diretas pelo sistema operacional windows, o que permite a este programa ser usado nos sistemas operacionais windows 98, 2000, ME e XP. Isto traz uma grande vantagem, pois se pode usar qualquer placa de captura de vídeo e qualquer câmera de vídeo, que trabalhem com estes sistemas operacionais. Não sendo um sistema dedicado como os que existem no mercado, onde um hardware somente funciona com um software específico.

5.2 Proposta para Trabalhos Futuros

Este programa pode ser utilizado em trabalhos onde se quer adquirir imagens e trata-las por algoritmos específicos.

Por exemplo: pode ser utilizado em um trabalho onde se deseja fazer uma análise metalográfica automatizada, um trabalho de reconhecimento de padrões, onde se usariam algoritmos para análise e comparação de uma imagem. Neste exemplo, a imagem de um rosto de uma pessoa poderia ser adquirida e comparada com diversos padrões e estabelecida uma porcentagem de igualdade. Pode se utilizado pelo departamento de polícia, pois um poderoso algoritmo desenvolvido poderia reconhecer a imagem de um rosto pela relação de medidas de determinadas partes do mesmo, e reconhecer uma face mesmo que alterada por uma plástica facial. Do ponto de vista do equipamento robótico utilizado neste trabalho, caso seja feito um novo tipo de integração “robô X câmera”, é mais interessante utilizar um robô industrial, devido a sua versatilidade e precisão de movimentos.

Referências Bibliográficas

FERREIRA, M. F. **O reconhecimento de Padrões**. Dissertação de Mestrado. Universidade de Brasília, 1994.

KHATIB,O. **Real-time Obstacle Avoidance for Manipulators and Mobile Robots**, Int'l Journal of Robotic Research, vol. 5, no.1, p. 90-98, 1986.

OGATA, K. **Modern Control Engieneering**. 3ª. Ed., New Jersey, EUA: Prentice-Hall, 1996, p.997.

QUERIDO, S.C.F. **Implementação de um sistema de aquisição de imagem**. Taubaté, 1999. Trabalho de graduação inter-disciplinar. p.67

SINGH, J. SANJIV and MEGHANAD D. WAGH, **Robot Path Planning Using Intersecting Convex Shapes**: Analysis and Simulation, IEEE Journal of Robotics and Automation, vol. RA-3, n.º 2, 1987.

TOU, J. T., GONZALES, R. C. **Pattern Recognition Pinciples**. Addilson-Wesley Publishing Company, Massachusetts, 1981.

< http://www.lynxmotion.com >(site para o kit do robô) consultado em 20/01/2003

< http://www.webcam32.com > (site da câmera de vídeo utilizada) consultado em 30/01/2003

< http://www.ontrak.net/mfc.htm > (Sending Commands in Visual C^{++} With MFC to ADR Interface) Consultado em 15/01/2003

APÊNDICE

ROTINAS UTILIZADAS PARA A CRIAÇÃO DO SISTEMA DE VISÃO ROBÓTICA E MANIPULAÇÃO DO ROBÔ.

A - Sistema de visão.

Nesta parte está descrito o código da principal parte do programa de visão do robô.

```
// Centroid.cpp: implementation of the CCentroid class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Centroid1.h"
#include <math.h>

//#ifdef _DEBUG
//#undef THIS_FILE
//static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
//#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CCentroid::CCentroid(){
	Fator_Pixel_mm= 1;		// Valores iniciais
	Thresh= 185;  // Valor inicial do Threshold

	// Limites do sistema de coordenas do robô
	Xr_ini= 2;	Xr_fim= 180;
	Yr_ini= 4;	Yr_fim= 240;
}

CCentroid::~CCentroid() {
	delete[] Fig1;
	delete[] Fig2;
}

// Abre dois arquivos em formato Raw
void CCentroid::AbreArquivoRaw(CString Arquivo1, CString Arquivo2) {
	// Abre os arquivos de dados
	FILE *Arq1,*Arq2;
	Arq1= fopen(Arquivo1,"rb");
	Arq2= fopen(Arquivo2,"rb");

	if(Arq1==NULL || Arq2==NULL) {
		AfxMessageBox("Erro na abertura dos arquivos!");
		return;
	}
	XMax=256;	YMax=256;	ImageSize= XMax*YMax;

	// Alocando memória
	Fig1= new unsigned char[ImageSize];
	Fig2= new unsigned char[ImageSize];

	// Copiando dos arquivos (Arq1 e Arq2) para os ponteiros (Fig1 e Fig2)
	fread(Fig1, ImageSize, 1, Arq1);
```

```
	fread(Fig2, ImageSize, 1, Arq2);

	fclose(Arq1);
	fclose(Arq2);
}

// Abre dois arquivos em formato BMP
void CCentroid::AbreArquivoBMP(CString Arquivo1, CString Arquivo2) {
	BITMAPFILEHEADER Header1;
	BITMAPINFOHEADER Header2;
	unsigned char Rgb[3];

	// Abre os arquivos de dados
	FILE *Arq1,*Arq2;
	Arq1= fopen(Arquivo1,"rb");
	Arq2= fopen(Arquivo2,"rb");

	if(Arq1==NULL) {
		AfxMessageBox("Erro na abertura do arquivo: " +Arquivo1);
		return;
	}
	if(Arq2==NULL) {
		AfxMessageBox("Erro na abertura do arquivo: " +Arquivo2);
		return;
	}

	// Copia os dados para dentro dos Headers
	fread((char*)&Header1, sizeof(Header1), 1, Arq1);
	fread((char*)&Header2, sizeof(Header2), 1, Arq1);
	fread((char*)&Header1, sizeof(Header1), 1, Arq2);
	fread((char*)&Header2, sizeof(Header2), 1, Arq2);

	XMax= Header2.biWidth;
	YMax= Header2.biHeight;
	ImageSize= XMax*YMax;

	// Alocando memória
	Fig1= new unsigned char[ImageSize];
	Fig2= new unsigned char[ImageSize];

	// Copiando dos arquivos (Arq1 e Arq2) para os ponteiros (Fig1 e Fig2)
	unsigned char *P1= Fig1, *P2= Fig2;
	for(int x=0; x<XMax; x++) {
		for(int y=0; y<YMax; y++) {
			// Conversão de RGB para tons de cinza
			fread(&Rgb,     sizeof(unsigned char), 3, Arq1);
			*P1= (unsigned char)((Rgb[0] +Rgb[1] +Rgb[2])/3);

			// Conversão de RGB para tons de cinza
			fread(&Rgb,     sizeof(unsigned char), 3, Arq2);
			*P2= (unsigned char)((Rgb[0] +Rgb[1] +Rgb[2])/3);

			P1++;
			P2++;
		}
```

```
        }
        fclose(Arq1);
        fclose(Arq2);
}

void CCentroid::DesenhaFiguras(HWND hwndStill){
        unsigned char *P1= Fig1, *P2= Fig2;

   RECT rc;
   ::GetWindowRect(hwndStill, &rc );
   HDC hdcStill = ::GetDC(hwndStill);
   PAINTSTRUCT ps;
   ::BeginPaint(hwndStill, &ps);  {

                // Desenha a imagem original
                for(int y=YMax-1; y>=0; y--) {
                        for (int x=0; x<XMax; x++) {
                                SetPixel(hdcStill, x/2,       (y+YMax)/2, *P1);
        // Desenha 1a Imagem Original
                                SetPixel(hdcStill, (x+XMax)/2,(y+YMax)/2, *P2);
                // Desenha 2a Imagem Original
                                SetPixel(hdcStill, x/2,       y/2,      abs(*P1 -*P2));   //
Subtração das imagens
                                P1++;
                                P2++;
                        };
                };
        }
   ::EndPaint(hwndStill, &ps);
   ::ReleaseDC( hwndStill, hdcStill );
}

// Desenha a imagem 2 (de fundo)
void CCentroid::DesenhaFundo(HWND hwndStill) {
        unsigned char *P2= Fig2;

   RECT rc;
   ::GetWindowRect(hwndStill, &rc );
   HDC hdcStill = ::GetDC(hwndStill);
   PAINTSTRUCT ps;
   ::BeginPaint(hwndStill, &ps);  {

                // Desenha 2a Imagem Original
                for(int y=YMax-1; y>=0; y--) {
                        for (int x=0; x<XMax; x++) {
                                SetPixel(hdcStill, x/2, y/2, *P2);
                                P2++;
                        };
                };
        }
   ::EndPaint(hwndStill, &ps);
   ::ReleaseDC( hwndStill, hdcStill );
}

void CCentroid::DesenhaCentroid(HWND hwndStill) {
```

```
        unsigned char *P1= Fig1, *P2= Fig2;

    RECT rc;
    ::GetWindowRect(hwndStill, &rc );
    HDC hdcStill = ::GetDC(hwndStill);
    PAINTSTRUCT ps;
    ::BeginPaint(hwndStill, &ps);  {

                for(int y=YMax-1; y>=0; y--)
                        for(int x=0; x<XMax; x++) {
                                // Binarização da imagem
                                if(abs(*P1 -*P2) > Thresh)
                                        SetPixel(hdcStill, x/*+XMax*/, y, 255);         // Pinta
em vermelho
                                else
                                        SetPixel(hdcStill, x/*+XMax*/, y,   0);// Pinta em
preto

                                P1++;
                                P2++;
                        }

                // Desenhando uma cruz no centróide da imagem
                for(y=((int)Yc-10); y<((int)Yc+10); y++) {      // Traço vertical
                        SetPixel(hdcStill, (int)Xc,      y, 0x00FF00);  // Imagem normal
                        SetPixel(hdcStill, (int)Xc+XMax, y, 0x00FF00);       // Imagem
thresh
                }
                for(int x=((int)Xc-10); x<((int)Xc+10); x++) { // Traço horizontal
                        SetPixel(hdcStill,      x, (int)Yc, 0x00FF00);  // Imagem normal
                        SetPixel(hdcStill, x+XMax, (int)Yc, 0x00FF00);       // Imagem
thresh
                }
        }
    ::EndPaint(hwndStill, &ps);
    ::ReleaseDC( hwndStill, hdcStill );
}

// Calcula o centróide da DIFERENÇA entre as duas imagens
void CCentroid::CalcCentroid() {
        unsigned char *P1= Fig1, *P2= Fig2;

        Xc= 0;
        Yc= 0;
        Area = 0;
        for(int y=YMax-1; y>=0; y--)
                for(int x=0; x<XMax; x++) {
                        // Binarização da imagem
                        if(abs(*P1 -*P2) > Thresh) {
                                Xc+= x;
                                Yc+= y;
                                Area++;
                        }
                        P1++;
                        P2++;
```

```
                }

        // Calculando o centróide
        if(Area>0) {
                Xc= Xc/Area;
                Yc= Yc/Area;
        }

        // Calcula os valores em mm
        Area_mm= Area * Fator_Pixel_mm*Fator_Pixel_mm;
        Xc_mm= Xc*Fator_Pixel_mm;
        Yc_mm= Yc*Fator_Pixel_mm;
}

// Calcula o fator que converte entre mm e pixel
void CCentroid::CalcCalib() {
        // Utiliza-se um padrão conhecido, um círculo de 40 mm de diâmetro,
        // e 1256,64 mm² de área

        CalcCentroid();
        if(Area != 0)
                Fator_Pixel_mm= (float)sqrt(1256.64/Area);
}

// Converte valores de x,y  vindos do sistema de coordenadas
// da Câmera (pixels), para o sistema de coordenadas do Robô (mm)
FPoint CCentroid::Calc_Pos_mm(float Xc, float Yc) {
        FPoint RoboCoord={0,0};
        CString S;
        float t;

        if((Xc_fim -Xc_ini) != 0)
                RoboCoord.y= Yr_ini +(Yr_fim -Yr_ini) / (Xc_fim -Xc_ini) * (Xc -Xc_ini);

        if((Yc_fim -Yc_ini) != 0)
                RoboCoord.x= Xr_ini +(Xr_fim -Xr_ini) / (Yc_fim -Yc_ini) * (Yc -Yc_ini);

        return RoboCoord;
}

// Ajusta os limites do sistema de coordenas da câmera
void CCentroid::SetCamLimits(float xi, float yi,float xf, float yf) {
        Xc_ini= xi;             Xc_fim= xf;
        Yc_ini= yi;             Yc_fim= yf;
}


// Ajusta os limites do sistema de coordenas do robô
void CCentroid::SetRoboLimits(float xi, float yi,float xf, float yf) {
        Xr_ini= xi;             Xr_fim= xf;
        Yr_ini= yi;             Yr_fim= yf;
}
```

```
/*************************************************************************
```

B- Processamento da imagem

```
*************************************************************************/
void CStillCapDlg::OnButton1() {
        char s[150];

        Robo.Close_Gripper();

        Obj.AbreArquivoBMP("c:\\StillCap0000.bmp", "c:\\FUNDO.BMP");
        UpdateData();
        Obj.Thresh= m_Tresh;                    // Ajusta o valor do
Threshold
        Obj.DesenhaFiguras(hwndStill);          // Desenha as imagens
        Obj.CalcCentroid();                     // Calcula o centróide da
imagem
        Obj.CalcCalib();                        // Faz a calibração
        //Obj.DesenhaCentroid(hwndStill);       // Desenha o centróide

        sprintf(s, " Area=  %d pixels² = %.1f mm²", Obj.Area, Obj.Area_mm);
        m_List.AddString(s);

        sprintf(s, " Diâmetro= %.2f pixels = %.2f mm", sqrt(Obj.Area*4/3.14159265),
sqrt(Obj.Area_mm*4/3.14159265));
        m_List.AddString(s);

        sprintf(s, " Threshold=  %i", Obj.Thresh);
        m_List.AddString(s);
```

```
        sprintf(s, " Fator (mm² para pixel²)= %.2f", Obj.Fator_Pixel_mm);
        m_List.AddString(s);

        sprintf(s, " Xc= %d pixel = %.2f mm;  Yc= %d pixel = %.2f mm", Obj.Xc,
Obj.Xc_mm, Obj.Yc, Obj.Yc_mm);
        m_List.AddString(s);
}

// Captura a imagem atual, e calcula o centróide da imagem
void CStillCapDlg::OnCalcula() {
        char s[150];

        OnSnap();      // Captura a imagem atual em 'StillCap0000.bmp'

        Obj.AbreArquivoBMP("c:\\StillCap0000.bmp", "c:\\Fundo.bmp");
        UpdateData();
        Obj.Thresh= m_Tresh;                              // Ajusta o valor do
Threshold
        Obj.CalcCentroid();                               // Calcula o centróide da
imagem
        Obj.DesenhaCentroid(hwndStill);           // Desenha o centróide
        Obj.DesenhaFundo(hwndStill2);             // Desenha a imagem de fundo

        sprintf(s, " Area=  %d pixels² = %.1f mm²", Obj.Area, Obj.Area_mm);
        //m_List.AddString(s);

        sprintf(s, " Diâmetro= %.2f pixels = %.2f mm", sqrt(Obj.Area*4/3.14159265),
sqrt(Obj.Area_mm*4/3.14159265));
```

```
        //m_List.AddString(s);

        sprintf(s, " Fator (mm² para pixel²)= %.2f", Obj.Fator_Pixel_mm);
        //m_List.AddString(s);

        sprintf(s, " Xc= %d pixel = %.2f mm;  Yc= %d pixel = %.2f mm", Obj.Xc,
Obj.Xc_mm, Obj.Yc, Obj.Yc_mm);
        m_List.AddString(s);

        sprintf(s, " Threshold=  %i", Obj.Thresh);
        m_List.AddString(s);

        FPoint Ponto= Obj.Calc_Pos_mm((float)Obj.Xc, (float)Obj.Yc);

        sprintf(s, " Centróide: [%d, %d] pixel,  [%.1f, %.1f] mm", Obj.Xc, Obj.Yc,Ponto.x,
Ponto.y);
        m_List.AddString(s);
        m_XText= Ponto.x;
        m_YText= Ponto.y;
        UpdateData(FALSE);
}

// Captura a imagem de fundo
void CStillCapDlg::OnCapturaFundo() {
        // Apaga os arquivos anteriores (limpeza) SE existirem
        if(GetFileAttributes("c:\\StillCap0000.bmp") !=  0xFFFFFFFF)
                if(!DeleteFile("c:\\StillCap0000.bmp"))
                        AfxMessageBox("Falha em apagar 'StillCap0000.bmp'");
```

```
	if(GetFileAttributes("c:\\fundo.bmp") !=  0xFFFFFFFF)
		if(!DeleteFile("c:\\fundo.bmp"))
			AfxMessageBox("Falha em apagar 'fundo.bmp'");

	OnSnap();	// Captura a imagem atual
	// Espera até que o arquivo esteja totalmente criado
	while(GetFileAttributes("c:\\StillCap0000.bmp") ==  0xFFFFFFFF) {
		;
	}

	// Copiar o arquivo 'StillCap0000.bmp'  como
	//'fundo.bmp' (FALSE força a cópia, mesmo que o arquivo já exista)
	if(CopyFile("c:\\StillCap0000.bmp", "c:\\fundo.bmp", FALSE) == 0)
		AfxMessageBox("Falha na cópia de 'StillCap0000.bmp' como
'fundo.bmp'");

	Obj.AbreArquivoBMP("c:\\StillCap0000.bmp", "c:\\Fundo.bmp");
	Obj.DesenhaFundo(hwndStill2);
}

// Procedimento de calibração das escalas e tamanhos
void CStillCapDlg::OnCalib() {
	CWaitCursor WaitCursor;
	CString Texto;
	float Xi, Yi;	// Coordenadas inicias do calibre
	float Xf, Yf;	// Coordenadas finais do calibre

	// Apaga os arquivos anteriores (limpeza)
```

```
        if(!DeleteFile("c:\\StillCap0000.bmp"))
                AfxMessageBox("Falha em apagar 'StillCap0000.bmp'");
        if(!DeleteFile("c:\\fundo.bmp"))
                AfxMessageBox("Falha em apagar 'fundo.bmp'");

        // Manda o robô para a posição inicial
        Robo.Home();

        // Primeiro passo: captura o fundo
        AfxMessageBox("Início do procedimento de calibração. \r\nLimpe a
mesa",MB_ICONINFORMATION);
        OnCapturaFundo();

        Texto.Format("Coloque o calibre na origem do sistema (%.0f,%.0f)", Obj.Xr_ini,
Obj.Yr_ini, MB_ICONEXCLAMATION);
        AfxMessageBox(Texto);
        OnSnap();               // Captura a imagem com o calibre
        OnCalcula();   // Calcula as coordenadas do calibre
        Xi= Obj.Xc_mm;        // armazena os valores das coordenadas do calibre
        Yi= Obj.Yc_mm;
        Texto.Format("Coordenadas do cálibre: (%.0f,%.0f)= (%.0f,%.0f)", Obj.Xr_ini,
Obj.Yr_ini, Xi, Yi,MB_ICONINFORMATION);
        AfxMessageBox(Texto);

        Texto.Format("Coloque o calibre no extremo do sistema (%.0f,%.0f)",
Obj.Xr_fim, Obj.Yr_fim,MB_ICONEXCLAMATION);
        AfxMessageBox(Texto);
```

```
        OnSnap();               // Captura a imagem com o calibre
        OnCalcula();   // Calcula as coordenadas do calibre
        Xf= Obj.Xc_mm;        // armazena os valores das coordenadas do calibre
        Yf= Obj.Yc_mm;
        Texto.Format("Coordenadas do calibre: (%.0f,%.0f)= (%.0f,%.0f)", Obj.Xr_fim,
Obj.Yr_fim, Xf, Yf,MB_ICONINFORMATION);
        AfxMessageBox(Texto);
        AfxMessageBox("Calibração concluída com sucesso!",
MB_ICONINFORMATION);

        // Ajusta os valores encontrados na calibração no robô
        Obj.SetCamLimits(Xi, Yi, Xf, Yf);

        // Guarda os valores encontrados no registro, salvando assim a configuração
atual
        CWinApp* pApp = AfxGetApp();
        Texto.Format("%f", Xi);
        pApp->WriteProfileString("Robo", "Xi", Texto);
        Texto.Format("%f", Yi);
        pApp->WriteProfileString("Robo", "Yi", Texto);
        Texto.Format("%f", Xf);
        pApp->WriteProfileString("Robo", "Xf", Texto);
        Texto.Format("%f", Yf);
        pApp->WriteProfileString("Robo", "Yf", Texto);

        m_List.AddString("");
        Texto.Format(" CAMLimits(Xi(%f), Yi(%f), Xf(%f), Yf(%f))", Obj.Xc_ini,
Obj.Yc_ini, Obj.Xc_fim, Obj.Yc_fim);
```

```
        m_List.AddString(Texto);
        Texto.Format(" ROBOLimits(Xi(%f), Yi(%f), Xf(%f), Yf(%f))", Obj.Xr_ini,
Obj.Yr_ini, Obj.Xr_fim, Obj.Yr_fim);
        m_List.AddString(Texto);
}

// Exibe na tela os valores das coordenadas (x,y) quando o mouse é deslocado na tela
void CStillCapDlg::OnMouseMove(UINT nFlags, CPoint point)  {
        FPoint Ponto;
        CString S;

        if((point.x < 320) && (point.y < 240)) {
                point.y-= 15;   // Deslocamento da figura
                // Tranforma as cordenadas de pixels para mm
                Ponto= Obj.Calc_Pos_mm((float)point.x, (float)point.y);

                S.Format("[%d, %d] pixels,   [%.0f, %.0f] mm", point.x, point.y, Ponto.x,
Ponto.y);
                m_CordStatus.SetWindowText(S);
        }
        CDialog::OnMouseMove(nFlags, point);
}

// Mostra uma mensagem
void CStillCapDlg::OnLButtonDown(UINT nFlags, CPoint point) {
        FPoint Ponto;
        CString S;
```

```
        if((point.x < 320) && (point.y < 240)) {
                point.y-= 15;   // Deslocamento da figura
                Ponto= Obj.Calc_Pos_mm((float)point.x, (float)point.y);
                Robo.Home();
                Robo.Move_Pos(Ponto.x, Ponto.y);
        }

        CDialog::OnLButtonDown(nFlags, point);
}

void CStillCapDlg::OnRoboHome() {
        Robo.Home();
}

void CStillCapDlg::OnRoboOpenGripper() {
        Robo.Open_Gripper();
}

void CStillCapDlg::OnRoboCloseGripper() {
        Robo.Close_Gripper();
}

void CStillCapDlg::OnRoboPickup() {
        FPoint Ponto;

        UpdateData(TRUE);

        Robo.Home();
```

```
        Robo.Open_Gripper();        // Abre a garra

        Robo.Move_Pos(m_XText, m_YText);        // Vai para posição escolhida
        Robo.Close_Gripper();       // Fecha a garra

        Robo.Home();// Volta para Home
        Robo.Move_Pos(m_XDeposito, m_YDeposito);     // Vai para a caixa

        Robo.Open_Gripper();        // Abre a garra
        Robo.Home();          // Volta para Home
}

void CStillCapDlg::OnRoboGo() {
        UpdateData(TRUE);
        Robo.Home();
        Robo.Move_Pos(m_XText, m_YText);// Vai para posição escolhida
}

//*************************************************************************
```

C – Controle do robô.

```
*************************************************************************//

// Robo1.cpp: implementation of the CRobo class.

//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "MachVis.h"
```

```
#include "Robo1.h"
#include <conio.h>
#include <math.h>

#ifdef _DEBUG
#undef THIS_FILE
//static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CRobo::CRobo() {
        PORT= 0x3F8; //DEFINICAO DA PORTA DE ENDERECO, COM1 = 0x3F8,
COM2 = 0x2F8
        DELAY= 5;

        Servo_Pos[0]= 122;  Servo_Pos[1]= 202;  Servo_Pos[2]=                  121;
        Servo_Pos[3]= 127;
        New_Pos[0]=  123;  New_Pos[1]=  203;  New_Pos[2]=  123;  New_Pos[3]=
128;
        Act_Pos[0]=  122;   Act_Pos[1]=  202;   Act_Pos[2]=  121;   Act_Pos[3]=
127;
        Home_Pos[0]=  123;  Home_Pos[1]=  253;  Home_Pos[2]=                   67;
        Home_Pos[3]=  128;
```

```
	Open_Com(); // Abre e configura a porta de comunicação COM
}

CRobo::~CRobo() {
}

//***************************************************************************
//Função  Home
//Esta função leva os servos até a posição Home

//***************************************************************************
void CRobo::Home(){
	char i;
	for(i=0;i<4;i++)
		New_Pos[i] = Home_Pos[i];
	Move_Arm();
	TRACE("\nHome()");
void CRobo::Move_Pos(float x, float y) {
   Point p, Tabela[17][21]={
	{{232,159,193,18},{226,167,201,18},{216,176,209,18},{205,183,217,18},{196,18
8,224,18},{187,193,230,18},{177,199,236,18},{163,193,228,7},{150,193,228,6},{133,19
3,228,5},{116,193,228,5},{105,194,228,5},{89,194,228,7},{78,187,224,7},{65,195,229,1
8},{54,190,223,18},{44,185,217,18},{36,179,211,18},{26,172,203,18},{17,165,195,18},{
9,158,185,18}},{{224,156,183,18},{217,162,192,18},{210,165,198,18},{201,170,204,18}
,{192,175,210,18},{182,181,216,18},{171,187,222,18},{159,192,231,21},{148,192,231,2
0},{133,192,231,20},{116,192,231,19},{106,192,231,19},{94,192,231,22},{80,192,231,2
5},{68,182,218,18},{58,176,212,18},{48,170,206,18},{40,166,200,18},{30,162,194,18},{
24,158,186,18},{14,152,179,18}},{{218,148,173,18},{210,153,180,18},{203,158,188,18}
```

,{196,163,196,18},{186,166,200,18},{176,169,204,18},{165,175,210,18},{153,176,212,15},{143,177,212,13},{130,177,212,13},{114,177,212,14},{103,177,212,14},{93,177,212,16},{83,177,212,19},{72,174,206,18},{61,169,201,18},{53,167,199,18},{45,162,192,18},{35,156,185,18},{28,149,175,18},{21,146,172,18}},{{213,141,169,18},{205,146,174,18},{197,151,179,18},{190,156,184,18},{181,160,189,18},{172,164,194,18},{162,167,198,18},{151,165,196,12},{141,165,196,10},{130,165,196,10},{115,165,196,10},{105,165,196,10},{95,165,196,12},{85,165,196,14},{75,165,194,18},{66,163,191,18},{56,158,185,18},{48,153,178,18},{40,148,173,18},{33,145,169,18},{26,138,160,18}},{{208,134,155,18},{200,138,160,18},{192,144,167,18},{185,148,174,18},{177,151,178,18},{169,154,182,18},{160,159,186,18},{146,151,178,4},{137,151,178,4},{128,151,178,3},{113,151,178,3},{104,151,178,5},{95,151,178,7},{86,151,178,7},{77,157,184,18},{68,154,180,18},{60,151,177,18},{53,146,170,18},{44,140,163,18},{37,134,154,18},{30,130,150,18}},{{203,128,144,18},{196,132,150,18},{189,135,156,18},{181,141,162,18},{173,144,169,18},{166,147,173,18},{157,149,175,18},{146,145,168,7},{137,145,168,6},{128,145,168,6},{115,145,168,5},{106,145,168,7},{97,145,168,8},{88,145,168,8},{81,147,173,18},{73,147,173,21},{65,143,168,21},{58,141,163,21},{49,135,156,18},{42,130,148,18},{34,127,143,18}},{{199,122,135,18},{193,125,141,18},{185,129,147,18},{177,134,153,18},{171,137,157,18},{163,140,161,18},{155,141,163,18},{144,142,166,16},{136,142,166,15},{127,142,166,15},{114,142,166,15},{105,142,166,15},{98,142,166,16},{90,142,166,16},{85,140,160,18},{76,138,159,18},{69,134,154,18},{61,130,149,18},{54,126,144,18},{48,123,139,18},{42,120,134,18}},{{195,113,123,18},{189,117,129,18},{182,122,135,18},{175,125,141,18},{168,127,144,18},{161,132,149,18},{153,133,151,18},{143,134,153,15},{135,134,153,15},{128,134,153,15},{115,134,153,15},{105,135,153,16},{98,134,153,16},{92,134,153,16},{82,132,150,18},{76,130,152,18},{70,128,146,18},{65,126,140,18},{58,121,134,18},{51,114,127,18},{45,111,122,18}},{{191,104,111,18},{186,108,117,18},{179,112,123,18},{172,116,129,18},{165,119,133,18},{157,121,136,18},{149,123,139,18},{142,126,141,16},{134,126,141,15},{126,126,141,15},{114,126,141,15},{107,126,141,16},{100,126,141,16},{93,126,141,16},{86,124,141,18},{79,123,137,18},{72,120,133,18},{66,116,126,18}

,{59,110,120,18},{52,107,115,18},{45,102,110,18}},{{189,97,101,18},{184,100,106,18},{177,104,112,18},{170,108,119,18},{164,110,121,18},{157,112,124,18},{150,113,127,18},{143,122,139,29},{137,122,139,29},{129,122,139,25},{123,122,139,25},{109,122,141,28},{103,122,141,28},{95,122,141,28},{89,117,128,18},{81,113,125,18},{75,110,120,18},{69,107,115,18},{62,104,109,18},{57,100,103,18},{50,96,97,18}},{{187,89,88,18},{181,93,95,18},{175,95,100,18},{169,99,103,18},{163,103,109,18},{156,105,112,18},{150,107,115,18},{143,107,110,13},{135,107,117,13},{129,107,117,13},{123,107,117,13},{109,107,117,15},{103,107,117,16},{96,107,117,18},{90,109,116,18},{84,107,113,18},{77,105,110,18},{70,101,106,18},{65,97,99,18},{59,93,92,18},{53,89,86,18}},{{184,82,76,18},{179,85,81,18},{173,88,86,18},{168,92,91,18},{161,95,96,18},{154,97,99,18},{149,99,102,18},{140,95,95,6},{134,95,95,5},{127,95,95,6},{116,95,95,5},{110,95,95,7},{104,95,95,8},{97,95,95,8},{90,100,103,18},{83,96,100,18},{77,95,97,18},{71,92,94,18},{66,88,87,18},{61,86,80,18},{57,82,73,18}},{{182,74,53,18},{178,76,62,18},{172,79,71,18},{166,82,79,18},{160,84,82,18},{154,87,85,18},{148,90,89,18},{143,91,91,19},{136,91,91,18},{129,91,91,17},{118,91,91,17},{112,91,91,19},{105,91,91,19},{99,91,91,23},{92,90,88,18},{86,88,85,18},{80,86,82,18},{74,84,79,18},{68,80,71,18},{62,76,60,18},{58,72,54,18}},{{180,63,38,18},{175,66,43,18},{170,69,48,18},{164,72,53,18},{158,76,58,18},{152,78,67,18},{146,81,71,18},{142,82,77,22},{136,82,77,21},{130,82,77,20},{118,82,77,20},{112,82,77,21},{106,82,77,22},{101,82,77,23},{94,82,75,18},{88,80,72,18},{82,78,63,18},{76,74,56,18},{72,70,50,18},{66,66,44,18},{62,62,38,18}},{{179,51,20,18},{174,56,27,18},{169,61,34,18},{163,64,38,18},{158,67,42,18},{152,69,46,18},{147,71,50,18},{142,73,54,19},{136,73,54,19},{130,73,54,19},{124,73,54,19},{112,73,54,18},{107,73,54,20},{103,73,54,21},{95,72,52,18},{90,70,50,18},{84,68,46,18},{78,63,40,18},{74,59,34,18},{68,55,28,18},{64,51,22,18}},{{177,43,14,30},{172,49,23,30},{167,56,32,30},{162,59,36,30},{157,61,40,30},{151,64,44,30},{145,67,48,30},{141,62,39,19},{136,62,39,18},{130,62,39,18},{120,62,39,19},{113,62,39,21},{109,62,39,21},{104,62,39,21},{95,63,52,30},{90,61,46,30},{86,59,40,30},{82,57,34,30},{76,54,27,30},{72,49,20,30},{68,45,14,30}},{{177,40,12,44},{172,44,18,42},{167,48,24,42},{162,50,28,42},{156,52,32,42},{151,55,36,42},{146,58

```
,40,42},{141,52,24,19},{135,52,24,18},{130,52,24,18},{120,52,24,18},{113,52,24,19},{1
09,52,24,21},{104,52,24,21},{98,59,40,42},{93,55,36,42},{88,54,33,42},{83,53,30,42},{7
9,49,25,42},{75,43,16,42},{71,35,7,46}}};
        // Convertendo de mm para cm
        //x= Xr_fim -x;          y= Yr_fim -y;
        // Como o robô foi calibrado em centímetros, é necessário dividir por dez
        x/= 10;                          y/= 10;
        x= floor(x);             y= floor(y);     // arredonda para inteiro
        // X varia de 0-16
        if(x>18)                 x=18;
        else if(x<2)     x=2;
        // Y varia de 0-20
        if(y>24)                 y=24;
        else if(y<4)     y=4;
        x-=2; y-=4;      // Início dos X's e Y's
        p= Tabela[(unsigned char)x][(unsigned char)y];
        New_Pos[0]= p.m[0];
    New_Pos[1]= p.m[1];
    New_Pos[2]= p.m[2];
    New_Pos[3]= p.m[3];
        TRACE("\nMove_Pos(%.1f, %.1f)", x+2, y+4);
        Move_Arm();
}
void CRobo::VoidMain(){
        // ############################################################
        // void main() {}
        unsigned char last_pos[4] = {123,203,123,128};
        int Teclado=0;
```

```
unsigned char  i=0, j=0;
float Entrada, x=10, y=14, z=0;
while(Teclado!=27) {
        Output_Screen(i, j, x, y);
        Teclado=getch();
        GotoXY(1,1);
        switch(Teclado) {
                case 'a': Open_Gripper();  break;
                case 'f': Close_Gripper();  break;
                case '1': i=0;  break;
                case '2': i=1;  break;
                case '3': i=2;  break;
                case '4': i=3;  break;
                case 'x': j=0;  break;
                case 'y': j=1;  break;
                case 'z': j=2;  break;
                case 'h':
                        New_Pos[0]=Home_Pos[0];
                        New_Pos[1]=Home_Pos[1];
                        New_Pos[2]=Home_Pos[2];
                        New_Pos[3]=Home_Pos[3];
                        break;

                case ']':
                                if(j==0)
                               x++;
                        else if(j==1)
                                y++;
```

```
                    else if(j==2)
                        z++;
                    break;
                case '[':
                     if(j==0)
                    x--;
                    else if(j==1)
                        y--;
                    else if(j==2)
                        z--;
                    break;

                case '+':
                    if(New_Pos[i]<=253)
                        New_Pos[i]+= 1;
                    break;
                case '-':
                    if(New_Pos[i]>= 3)
                        New_Pos[i]-= 1;
                    break;
                case ' ':
                    GotoXY(1,10);
                    printf("                                    ");
                    GotoXY(1,10);
                    printf("Entre com as novas coordenadas (x,y)[%.1f,%.1f]:
", x, y);
                    scanf("%f", &x);
                    scanf("%f", &y);
```

```
                    break;
                case '.':
                    GotoXY(1,10);
                    printf("                                    ");
                    GotoXY(1,10);
                    printf("Entre com a nova coordenada do Servo %d [%d]: ",
i, New_Pos[i]);
                    scanf("%f", &Entrada);
                    if(Entrada<0)
                        Entrada=0;
                    else if(Entrada>255)
                        Entrada=255;
                    New_Pos[i]= (unsigned char)Entrada;
                    break;
                case ',':
                    GotoXY(1,10);
                    printf("                                    ");
                    GotoXY(1,10);
                    printf("Entre com as novas coordenadas: ", i, New_Pos[i]);
                    scanf("%f", &Entrada);
                    if(Entrada<0)
                        Entrada=0;
                    else if(Entrada>255)
                        Entrada=255;
                    last_pos[0]= (unsigned char)Entrada;
                    scanf("%f", &Entrada);
                    if(Entrada<0)
                        Entrada=0;
```

```
                else if(Entrada>255)
                        Entrada=255;
                last_pos[1]= (unsigned char)Entrada;

                scanf("%f", &Entrada);
                if(Entrada<0)
                        Entrada=0;
                else if(Entrada>255)
                        Entrada=255;
                last_pos[2]= (unsigned char)Entrada;

                scanf("%f", &Entrada);
                if(Entrada<0)
                        Entrada=0;
                else if(Entrada>255)
                        Entrada=255;
                last_pos[3]= (unsigned char)Entrada;

                Home();
                New_Pos[0]=last_pos[0];
                New_Pos[1]=last_pos[1];
                New_Pos[2]=last_pos[2];
                New_Pos[3]=last_pos[3];
                Move_Arm();
                break;

        case 'l':
                Home();
```

```
                    New_Pos[0]=last_pos[0];
                    New_Pos[1]=last_pos[1];
                    New_Pos[2]=last_pos[2];
                    New_Pos[3]=last_pos[3];
                    break;
            }
            if(Teclado=='[' || Teclado==']' || Teclado==' ') {
                if(x>13) x=13;
                if(x< 2) x= 2;
                if(y>17) y=17;
                if(y<11) y=11;
                Output_Screen(i, j, x, y);
                Home();
                Open_Gripper();        // abre a garra
                Move_Pos(x, y);        // Vai para posição escolhida
                Close_Gripper(); // fecha a garra
                Home();                        // Volta para Home
                Move_Pos(13, 11);// Vai para a caixa
                Open_Gripper();        // abre a garra
                Home();                        // Volta para Home
            }
            Output_Screen(i, j, x, y);
            if(Teclado!='a' && Teclado!='f')
                Move_Arm();
        }

        getch();
}
```

```
void CRobo::Move_Arm() {
	float diff[4];//difference between servo_pos and New_Pos
	float step[4];//the step for each servo
	float pos[4]; //the position for each servo
	char i;
	char j = 0;
	//TRACE("\nMove_Arm(%d, %d, %d, %d)", New_Pos[0], New_Pos[1],
New_Pos[2], New_Pos[3]);
	for(i=0;i<4;i++) {
		diff[i] = (float)(New_Pos[i]-Servo_Pos[i]);//finds the differences
		if(i>0 && fabs(diff[i]) > fabs(diff[j]))//find the biggest differen
			j=i;
		pos[i] = Servo_Pos[i];//set the pos array
	}
	for(i=0;i<4;i++) {//sets the step for each servo
		if(fabs(diff[j])!=0)//check for division by zero
			step[i] = (float)(diff[i] / fabs(diff[j]));
		else
			step[i] = 0;
	}

	while(Servo_Pos[j] != New_Pos[j])
		//Esta rotina estabelece uma condição para que todos os servos
		//do robô cheguem ao destino ao mesmo tempo
		//Através das diferenças entre as posições.
		{
		for(i=0;i<4;i++)
			{
```

```
                pos[i] += step[i];


                if(diff[i]>0)//check for direction
                        {
                        if(floor(pos[i]) != Servo_Pos[i])
                                Out_Servo((unsigned char)floor(pos[i]),i);    //floor
is used to find
                                                                        //integer
values
                        }


                if(diff[i]<0)//check for direction
                        {
                        if(ceil(pos[i]) != Servo_Pos[i])
                                Out_Servo((unsigned  char)ceil(pos[i]),i);    //ceil  is
used to find


        //integer values
                        }
                }
            }
        for(i=0;i<4;i++)
                Out_Servo(New_Pos[i],i);//secures that the positions is reached
}
void CRobo::Open_Com() {
//*****************************************************************
//open_com function
//Abre a comunicação e inicializa o endereço da porta COM1 do PC
```

```
//*****************************************************************
        outportb(PORT + 1, 0);                  //Turn off interrupts
        outportb(PORT + 3, 0x80);    //Set Dlab on
        outportb(PORT + 0, 0x0C);   //Set Baud rate - Divisor Latch Low Byte
        outportb(PORT + 1, 0x00);   //Set Baud rate - Divisor Latch High Byte
        outportb(PORT + 3, 0x03);    //8 bits, no parity, 1 stop bit
        outportb(PORT + 2, 0x07);    //FIFO control register
        outportb(PORT + 4, 0x03);    //Turn on DTR, RTS, and OUT2
}
void CRobo::Output_Screen(unsigned char i, unsigned char j, float x, float y) {
        GotoXY(1,2);
   printf("Servo 0=%4d,    Servo 1=%4d,   Servo 2=%4d,   Servo 3=%4d",
   New_Pos[0], New_Pos[1], New_Pos[2], New_Pos[3]);
   switch(i) {
        case 0: GotoXY(1,2);
                        printf("Servo 0=%4d,", New_Pos[0]); break;
      case 1: GotoXY(18,2);
                        printf("Servo 1=%4d,", New_Pos[1]); break;
                case 2: GotoXY(34,2);
                        printf("Servo 2=%4d,", New_Pos[2]); break;
                case 3: GotoXY(50,2);
                        printf("Servo 3=%4d,", New_Pos[3]); break;
   }

   GotoXY(1,4);
   printf("X= %3.0f,          Y= %3.0f", x, y);
   switch(j) {
        case 0: GotoXY(1,4);
```

```
                printf("X= %3.0f,", x); break;
        case 1: GotoXY(18,4);
                printf("Y= %3.0f,", y); break;
    }
}
void CRobo::GotoXY(int x, int y) {
        COORD c;

        c.X= (short)x;    c.Y= (short)y;
        SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), c);
}
// Abre a garra
void CRobo::Open_Gripper() {
        const Val= 150;

        TRACE("\nOpen_Gripper(%d)", Val);
        Out_Servo(Val,4);
}
// Fecha a garra
void CRobo::Close_Gripper(){
        const Val= 25;

        TRACE("\nClose_Gripper(%d)", Val);
        Out_Servo(Val,4);
}
void CRobo::Out_Servo(unsigned char position, unsigned char servo_no){
        unsigned char sync_byte = 255;
```

```
        Act_Pos[servo_no]= position;
        while (position < Servo_Pos[servo_no]) {//used to count down
                outportb(PORT,sync_byte);
                Sleep(1);//delay to enable the SSC to differ between the bytes
                outportb(PORT,servo_no);
                Sleep(1);
                outportb(PORT,Servo_Pos[servo_no]);
                Sleep(DELAY);
                Servo_Pos[servo_no]--;
                Out_Status();
        }
        while (position > Servo_Pos[servo_no]) {//used to count up
                outportb(PORT, sync_byte);
                Sleep(1);
                outportb(PORT,servo_no);
                Sleep(1);
                outportb(PORT,Servo_Pos[servo_no]);
                Sleep(DELAY);
                Servo_Pos[servo_no]++;
                Out_Status();
        }
}
// Utilizando outp modo Windows
void CRobo::outportb(unsigned int portid, unsigned char value){
        _outp((unsigned short) portid, (int)value);
}
void CRobo::Out_Status(){
        GotoXY(1,20);
```

```
   printf("new_pos= {%4d,%4d, %4d,%4d};\n",
         New_Pos[0], New_Pos[1], New_Pos[2], New_Pos[3]);
         GotoXY(1,21);
   printf("act_pos= {%4d,%4d, %4d,%4d};",
         Act_Pos[0], Act_Pos[1], Act_Pos[2], Act_Pos[3]);
         GotoXY(1,22);
   printf("diff   = {%4d,%4d, %4d,%4d};",
         Act_Pos[0]-New_Pos[0],    Act_Pos[1]-New_Pos[1],    Act_Pos[2]-New_Pos[2],
Act_Pos[3]-New_Pos[3]);
}
```

// **D - Cinemática direta.** //

// Esta rotina calcula o mapeamento da região de trabalho do robô, através de cálculos em cinemática direta tornando possível a criação do “lock-up table”.//

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
const PI= 3.141592;
const TOL= 2.0;         // Tolerância em x, y, e z (2 mm)
void FindXY(float xx, float yy) {
         FILE *out;
         float m, n, p, x, y, z;
         long i=0, j=0;
         if((out= fopen("\\saida.txt", "wt")) == NULL)
                  printf("Cannot open output file.\n");
         else {
      // Limites mínimo e máximo de m[-45,45], n[0,100] e p[-120,-40
                  const Min_m= -45  *PI/180.0, Max_m=  45  *PI/180.0;
```

```
                const MIn_n=   0  *PI/180.0, Max_n= 100  *PI/180.0;
                const Min_p=-120  *PI/180.0, Max_p= -40  *PI/180.0;
        // Varre m, n e p, em intervalos de 0.01 radianos
                for(m=Min_m; m<=Max_m; m+=.01)
                for(n=Min_n; n<=Min_n; n+=.01)
                        for(p=Min_p; p<=Max_p; p+=.01) {
                // Formula de cinemática direta
                                        x=              210*cos(m)*cos(n)*cos(p)            -
210*cos(m)*sin(n)*sin(p)+95*cos(m)*cos(n)+11*cos(m);
                                        y=              210*sin(m)*cos(n)*cos(p)            -
210*sin(m)*sin(n)*sin(p)+95*sin(m)*cos(n)+11*sin(m);
                                        z= -sin(n)*sin(p)+cos(n)*cos(p)  +210*sin(n)*cos(p)
+210*cos(n)*sin(p)+83+95*sin(n);
                                        // So adquire os pontos dentro da tolerância
estabelecida
                if(fabs(z)<TOL){ // tolerância de z= zero
                                                fprintf(out, "i= %ld, m= %.0f, n= %.0f, p=
%.0f, x= %.3f, y= %.3f, z= %.3f \n", i, m*180.0/PI, n*180.0/PI, p*180.0/PI, x, y, z);
                                                if((fabs(x -xx)<TOL) && (fabs(y -yy) <TOL) )
{
                        printf("i= %ld, m= %.0f§, n= %.0f§, p= %.0f§, x= %.3f, y= %.3f,
z= %.3f \n", i, m*180.0/PI, n*180.0/PI, p*180.0/PI, x, y, z);
        }
                                                i++;     // Contador de pontos adquiridos
                                        }
                                        j++; // Contador de todos os pontos calculados
                                }
```

```
                fprintf(out, "j= %ld", j);// Imprime no arquivo a quantidade total de pontos
calculados
                fclose(out);
        }
}void main() {
        clrscr();
        // Tenta encontrar pontos próximos de x=158, y=133
        FindXY(158, 133);
```

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo